Diophantine Equations and the DPRM Theorem

Jonas Bayer* Marco David* Benedikt Stock*

Abhik Pal Yuri Matiyasevich† Dierk Schleicher

March 17, 2025

Abstract

We present a formalization of Matiyasevich's proof of the DPRM theorem, which states that every recursively enumerable set of natural numbers is Diophantine. This result from 1970 yields a negative solution to Hilbert's 10th problem over the integers. To represent recursively enumerable sets in equations, we implement and arithmetize register machines. We formalize a general theory of Diophantine sets and relations to reason about them abstractly. Using several number-theoretic lemmas, we prove that exponentiation has a Diophantine representation.

Contents

*Equal contribution.

†Contributed by supplying a detailed proof and an initial introduction to Isabelle.

Overview A previous short paper [2] gives an overview of the formalization. In particular, the challenges of implementing the notion of diophantine predicates is discussed and a formal definition of register machines is described. Another meta-publication [1] recounts our learning experience throughout this project.

The present formalisation is based on Yuri Matiyasevich's monograph [5] which contains a full proof of the DPRM theorem. This result or parts of its proof have also been formalized in other interactive theorem provers, notably in Coq [4], Lean [3] and Mizar [7, 6].

Acknowledgements We want to thank everyone who participated in the formalization during the early stages of this project: Deepak Aryal, Bogdan Ciurezu, Yiping Deng, Prabhat Devkota, Simon Dubischar, Malte HaSSler, Yufei Liu and Maria Antonia Oprea. Moreover, we would like to express our sincere gratitude to the entire welcoming and supportive Isabelle community. In particular, we are indebted to Christoph Benzmüller for his expertise, his advice and for connecting us with relevant experts in the field. Among those, we specially want to thank Mathias Fleury for all his help with Isabelle. Finally, we would like to thank the DFG for supporting our attendance at several events and conferences, allowing us to present the project to a broad audience.

1 Diophantine Equations

theory *Parametric-Polynomials*
  imports *Main*
  abbrevs ++ = **+** **and**
    −− = **−** **and**
    ∗∗ = **∗** **and**
    00 = **0** **and**
    11 = **1**
begin

1.1 Parametric Polynomials

This section defines parametric polynomials and builds up the infrastructure to later prove that a given predicate or relation is Diophantine. The formalization follows [5].

type-synonym *assignment* = *nat* ⇒ *nat*

Definition of parametric polynomials with natural number coefficients and their evaluation function

datatype *ppolynomial* =
  Const nat |
  Param nat |
  Var nat |
  Sum ppolynomial ppolynomial (**infixl** ‹**+**› *65*) |
  NatDiff ppolynomial ppolynomial (**infixl** ‹**−**› *65*) |
  Prod ppolynomial ppolynomial (**infixl** ‹**∗**› *70*)

fun *ppeval* :: *ppolynomial* ⇒ *assignment* ⇒ *assignment* ⇒ *nat* **where**
  ppeval (*Const c*) *p v* = *c* |
  ppeval (*Param x*) *p v* = *p x* |
  ppeval (*Var x*) *p v* = *v x* |
  ppeval (*D1* **+** *D2*) *p v* = (*ppeval D1 p v*) + (*ppeval D2 p v*) |

  ppeval (*D1* **−** *D2*) *p v* = (*ppeval D1 p v*) − (*ppeval D2 p v*) |
  ppeval (*D1* **∗** *D2*) *p v* = (*ppeval D1 p v*) ∗ (*ppeval D2 p v*)

definition *Sq-pp* (‹- **^2**› [*99*] *75*) **where** *Sq-pp P* = *P* **∗** *P*

definition *is-dioph-set* :: *nat set* ⇒ *bool* **where**
  is-dioph-set A = (∃ *P1 P2*::*ppolynomial*. ∀ *a*. (*a* ∈ *A*)
    ⟷ (∃ *v*. *ppeval P1* (λ*x*. *a*) *v* = *ppeval P2* (λ*x*. *a*) *v*))

datatype *polynomial* =
  Const nat |
  Param nat |
  Sum polynomial polynomial (**infixl** ‹[+]› *65*) |

```
    NatDiff polynomial polynomial (infixl ‹[−]› 65) |
    Prod polynomial polynomial (infixl ‹[∗]› 70)

fun peval :: polynomial ⇒ assignment ⇒ nat  where
    peval (Const c) p = c |
    peval (Param x) p = p x |
    peval (Sum D1 D2) p = (peval D1 p) + (peval D2 p) |

    peval (NatDiff D1 D2) p = (peval D1 p) − (peval D2 p) |
    peval (Prod D1 D2) p = (peval D1 p) ∗ (peval D2 p)

definition sq-p :: polynomial ⇒ polynomial (‹- [ˆ2]› [99] 75) where sq-p P = P
[∗] P

definition zero-p :: polynomial (‹0›) where zero-p = Const 0
definition one-p :: polynomial (‹1›) where one-p = Const 1

lemma sq-p-eval: peval (P[ˆ2]) p = (peval P p)ˆ2
  unfolding sq-p-def by (simp add: power2-eq-square)

fun convert :: polynomial ⇒ ppolynomial  where
    convert (Const c) = (ppolynomial.Const c) |
    convert (Param x) = (ppolynomial.Param x) |
    convert (D1 [+] D2) = (convert D1) + (convert D2) |
    convert (D1 [−] D2) = (convert D1) − (convert D2) |
    convert (D1 [∗] D2) = (convert D1) ∗ (convert D2)

lemma convert-eval: peval P a = ppeval (convert P) a v
  by (induction P, auto)

definition list-eval :: polynomial list ⇒ assignment ⇒ (nat ⇒ nat) where
    list-eval PL a = nth (map (λx. peval x a) PL)

end
```

1.2 Variable Assignments

The following theory defines manipulations of variable assignments and proves elementary facts about these. Such preliminary results will later be necesary to e.g. prove that conjunction is diophantine.

```
theory Assignments
  imports Parametric-Polynomials
begin

definition shift :: nat list ⇒ nat ⇒ assignment where
  shift l a ≡ λi. l ! (i + a)

definition push :: assignment ⇒ nat ⇒ assignment where
  push a n i = (if i = 0 then n else a (i−1))
```

definition *push-list* :: *assignment* ⇒ *nat list* ⇒ *nat* ⇒ *nat* **where**
push-list a ns i = (*if i* < *length ns then* (*ns*!*i*) *else a* (*i* − *length ns*))

lemma *push0*: *push a n 0* = *n*
by (*auto simp*: *push-def*)

lemma *push-list-empty*: *push-list a* [] = *a*
unfolding *push-list-def* **by** *auto*

lemma *push-list-singleton*: *push-list a* [*n*] = *push a n*
unfolding *push-list-def push-def* **by** *auto*

lemma *push-list-eval*: *i* < *length ns* ⟹ *push-list a ns i* = *ns*!*i*
unfolding *push-list-def* **by** *auto*

lemma *push-list1*: *push* (*push-list a ns*) *n* = *push-list a* (*n* # *ns*)
unfolding *push-def push-list-def* **by** *fastforce*

lemma *push-list2-aux*: (*push-list* (*push a n*) *ns*) *i* = *push-list a* (*ns* @ [*n*]) *i*
unfolding *push-def push-list-def* **by** (*auto simp*: *nth-append*)

lemma *push-list2*: (*push-list* (*push a n*) *ns*) = *push-list a* (*ns* @ [*n*])
unfolding *push-list2-aux* **by** *auto*

fun *pull-param* :: *ppolynomial* ⇒ *ppolynomial* ⇒ *ppolynomial* **where**
pull-param (*ppolynomial.Param 0*) *repl* = *repl* |
pull-param (*ppolynomial.Param* (*Suc n*)) - = (*ppolynomial.Param n*) |
pull-param (*D1* + *D2*) *repl* = (*pull-param D1 repl*) + (*pull-param D2 repl*) |
pull-param (*D1* − *D2*) *repl* = (*pull-param D1 repl*) − (*pull-param D2 repl*) |
pull-param (*D1* ∗ *D2*) *repl* = (*pull-param D1 repl*) ∗ (*pull-param D2 repl*) |
pull-param P repl = *P*

fun *var-set* :: *ppolynomial* ⇒ *nat set* **where**
var-set (*ppolynomial.Const c*) = {} |
var-set (*ppolynomial.Param x*) = {} |
var-set (*ppolynomial.Var x*) = {*x*} |
var-set (*D1* + *D2*) = *var-set D1* ∪ *var-set D2* |
var-set (*D1* − *D2*) = *var-set D1* ∪ *var-set D2* |
var-set (*D1* ∗ *D2*) = *var-set D1* ∪ *var-set D2*

definition *disjoint-var* :: *ppolynomial* ⇒ *ppolynomial* ⇒ *bool* **where**
disjoint-var P Q = (*var-set P* ∩ *var-set Q* = {})

named-theorems *disjoint-vars*

lemma *disjoint-var-sym*: *disjoint-var P Q* = *disjoint-var Q P*

unfolding *disjoint-var-def* **by** *auto*

lemma *disjoint-var-sum*[*disjoint-vars*]: *disjoint-var* (*P1* + *P2*) *Q* = (*disjoint-var* *P1 Q* ∧ *disjoint-var P2 Q*)
unfolding *disjoint-var-def* **by** *auto*

lemma *disjoint-var-diff*[*disjoint-vars*]: *disjoint-var* (*P1* − *P2*) *Q* = (*disjoint-var* *P1 Q* ∧ *disjoint-var P2 Q*)
unfolding *disjoint-var-def* **by** *auto*

lemma *disjoint-var-prod*[*disjoint-vars*]: *disjoint-var* (*P1* ∗ *P2*) *Q* = (*disjoint-var* *P1 Q* ∧ *disjoint-var P2 Q*)
unfolding *disjoint-var-def* **by** *auto*

lemma *aux-var-set*:
assumes ∀ *i* ∈ *var-set P*. *x i* = *y i*
shows *ppeval P a x* = *ppeval P a y*
using *assms* **by** (*induction P*, *auto*)

First prove that disjoint variable sets allow the unification into one variable assignment

definition *zip-assignments* :: *ppolynomial* ⇒ *ppolynomial* ⇒ *assignment* ⇒ *assignment* ⇒ *assignment*
where *zip-assignments P Q v w i* = (*if i* ∈ *var-set P then v i else w i*)

lemma *help-eval-zip-assignments1*:
shows *ppeval P1 a* (λ*i*. *if i* ∈ *var-set P1* ∪ *var-set P2 then v i else w i*)
= *ppeval P1 a* (λ*i*. *if i* ∈ *var-set P1 then v i else w i*)
using *aux-var-set* **by** *auto*

lemma *help-eval-zip-assignments2*:
shows *ppeval P2 a* (λ*i*. *if i* ∈ *var-set P1* ∪ *var-set P2 then v i else w i*)
= *ppeval P2 a* (λ*i*. *if i* ∈ *var-set P2 then v i else w i*)
using *aux-var-set* **by** *auto*

lemma *eval-zip-assignments1*:
fixes *v w*
assumes *disjoint-var P Q*
defines *x* ≡ *zip-assignments P Q v w*
shows *ppeval P a v* = *ppeval P a x*
using *assms*
apply (*induction P arbitrary*: *x*)
unfolding *x-def zip-assignments-def*
using *help-eval-zip-assignments1 help-eval-zip-assignments2*
by (*auto simp add*: *disjoint-vars*)

lemma *eval-zip-assignments2*:
fixes *v w*
assumes *disjoint-var P Q*

defines *x* ≡ *zip-assignments P Q v w*
shows *ppeval Q a w* = *ppeval Q a x*
using *assms*
apply (*induction Q arbitrary*: *P x*)
unfolding *x-def zip-assignments-def*
using *disjoint-var-sym disjoint-vars*
by (*auto simp*: *disjoint-var-def*) (*smt* (*z3*) *inf-commute*)+

lemma *zip-assignments-correct*:
assumes *ppeval P1 a v* = *ppeval P2 a v* **and** *ppeval Q1 a w* = *ppeval Q2 a w*
and *disjoint-var* (*P1* + *P2*) (*Q1* + *Q2*)
defines *x* ≡ *zip-assignments* (*P1* + *P2*) (*Q1* + *Q2*) *v w*
shows *ppeval P1 a x* = *ppeval P2 a x* **and** *ppeval Q1 a x* = *ppeval Q2 a x*
proof −
from *assms*(*3*) **have** *disjoint-var P1* (*Q1* + *Q2*)
by (*auto simp*: *disjoint-var-sum*)
moreover have *ppeval P1 a x* = *ppeval P1 a* (*zip-assignments P1* (*Q1* + *Q2*) *v w*)
unfolding *x-def zip-assignments-def* **using** *help-eval-zip-assignments1* **by** *auto*
ultimately have *p1*: *ppeval P1 a x* = *ppeval P1 a v*
using *eval-zip-assignments1*[*of P1*] **by** *auto*

from *assms*(*3*) **have** *disjoint-var P2* (*Q1* + *Q2*)
by (*auto simp*: *disjoint-var-sum*)
moreover have *ppeval P2 a x* = *ppeval P2 a* (*zip-assignments P2* (*Q1* + *Q2*) *v w*)
unfolding *x-def zip-assignments-def* **using** *help-eval-zip-assignments2* **by** *auto*
ultimately have *p2*: *ppeval P2 a x* = *ppeval P2 a v*
using *eval-zip-assignments1*[*of P2*] **by** *auto*

from *p1 p2* **show** *ppeval P1 a x* = *ppeval P2 a x*
using *assms*(*1*) **by** *auto*
next
have *disjoint-var* (*P1* + *P2*) *Q1*
using *assms*(*3*) *disjoint-var-sum disjoint-var-sym* **by** *auto*
moreover have *ppeval Q1 a x* = *ppeval Q1 a* (*zip-assignments* (*P1* + *P2*) *Q1 v w*)
unfolding *x-def zip-assignments-def* **using** *help-eval-zip-assignments1* **by** *auto*
ultimately have *q1*: *ppeval Q1 a x* = *ppeval Q1 a w*
using *eval-zip-assignments2*[*of - Q1*] **by** *auto*

from *assms*(*3*) **have** *disjoint-var* (*P1* + *P2*) *Q2*
using *assms*(*3*) *disjoint-var-sum disjoint-var-sym* **by** *auto*
moreover have *ppeval Q2 a x* = *ppeval Q2 a* (*zip-assignments* (*P1* + *P2*) *Q2 v w*)
unfolding *x-def zip-assignments-def* **using** *help-eval-zip-assignments2* **by** *auto*
ultimately have *q2*: *ppeval Q2 a x* = *ppeval Q2 a w*
using *eval-zip-assignments2*[*of - Q2*] **by** *auto*

from *q1 q2* **show** *ppeval Q1 a x = ppeval Q2 a x*
using *assms*(*2*) **by** *auto*
qed

lemma *disjoint-var-unifies*:
assumes ∃ *v1*. *ppeval P1 a v1 = ppeval P2 a v1* **and** ∃ *v2*. *ppeval Q1 a v2 = ppeval Q2 a v2*
and *disjoint-var* (*P1* + *P2*) (*Q1* + *Q2*)
shows ∃ *v*. *ppeval P1 a v = ppeval P2 a v* ∧ *ppeval Q1 a v = ppeval Q2 a v*
using *assms zip-assignments-correct* **by** (*auto*) *metis*

A function to manipulate variables in ppolynomials

fun *push-var* :: *ppolynomial* ⇒ *nat* ⇒ *ppolynomial* **where**
push-var (*ppolynomial.Var x*) *n* = *ppolynomial.Var* (*x* + *n*) |
push-var (*D1* + *D2*) *n* = *push-var D1 n* + *push-var D2 n* |
push-var (*D1* − *D2*) *n* = *push-var D1 n* − *push-var D2 n* |
push-var (*D1* ∗ *D2*) *n* = *push-var D1 n* ∗ *push-var D2 n* |
push-var D n = *D*

lemma *push-var-bound*: *x* ∈ *var-set* (*push-var P* (*Suc n*)) ⟹ *x* > *n*
by (*induction P*, *auto*)

definition *pull-assignment* :: *assignment* ⇒ *nat* ⇒ *assignment* **where**
pull-assignment v n = (λ*x*. *v* (*x*+*n*))

lemma *push-var-pull-assignment*:
shows *ppeval* (*push-var P n*) *a v* = *ppeval P a* (*pull-assignment v n*)
by (*induction P*, *auto simp*: *pull-assignment-def*)

lemma *max-set*: *finite A* ⟹ ∀ *x*∈*A*. *x* ≤ *Max A*
using *Max-ge* **by** *blast*

fun *push-param* :: *polynomial* ⇒ *nat* ⇒ *polynomial* **where**
push-param (*Const c*) *n* = *Const c* |
push-param (*Param x*) *n* = *Param* (*x* + *n*) |
push-param (*Sum D1 D2*) *n* = *Sum* (*push-param D1 n*) (*push-param D2 n*) |
push-param (*NatDiff D1 D2*) *n* = *NatDiff* (*push-param D1 n*) (*push-param D2 n*) |
push-param (*Prod D1 D2*) *n* = *Prod* (*push-param D1 n*) (*push-param D2 n*)

definition *push-param-list* :: *polynomial list* ⇒ *nat* ⇒ *polynomial list* **where**
push-param-list s k ≡ *map* (λ*x*. *push-param x k*) *s*

lemma *push-param0*: *push-param P 0* = *P*
by (*induction P*, *auto*)

lemma *push-push-aux*: *peval* (*push-param P* (*Suc m*)) (*push a n*) = *peval* (*push-param P m*) *a*
  by (*induction P*, *auto simp*: *push-def*)

lemma *push-push*:
  shows *length ns* = *n* ⟹ *peval* (*push-param P n*) (*push-list a ns*) = *peval P a*
proof (*induction ns arbitrary*: *n*)
  case *Nil*
  then show *?case* **by** (*auto simp*: *push-list-empty push-param0*)
next
  case (*Cons n ns*)
  thus *?case*
    using *push-push-aux*[**where** *?a* = *push-list a ns*]
    by (*auto simp add*: *length-Cons push-list1*)
qed

lemma *push-push-simp*:
  shows *peval* (*push-param P* (*length ns*)) (*push-list a ns*) = *peval P a*
proof (*induction ns*)
  case *Nil*
  then show *?case* **by** (*auto simp*: *push-list-empty push-param0*)
next
  case (*Cons n ns*)
  thus *?case*
    using *push-push-aux*[**where** *?a* = *push-list a ns*]
    by (*auto simp add*: *length-Cons push-list1*)
qed

lemma *push-push1*: *peval* (*push-param P 1*) (*push a k*) = *peval P a*
  using *push-push*[**where** *?ns* = [*k*]] **by** (*auto simp*: *push-list-singleton*)

lemma *push-push-map*: *length ns* = *n* ⟹
  list-eval (*map* (λ*x*. *push-param x n*) *ls*) (*push-list a ns*) = *list-eval ls a*
  unfolding *list-eval-def* **apply** (*induction ls*, *simp*)
  apply (*induction ns*, *auto*)
  apply (*metis length-map list.size*(*3*) *nth-equalityI push-push*)
  by (*metis length-Cons length-map map-nth push-push*)

lemma *push-push-map-i*: *length ns* = *n* ⟹ *i* < *length ls* ⟹
  peval (*map* (λ*x*. *push-param x n*) *ls* ! *i*) (*push-list a ns*) = *list-eval ls a i*
  unfolding *list-eval-def* **by** (*auto simp*: *push-push-map push-push*)

lemma *push-push-map1*: *i* < *length ls* ⟹
  peval (*map* (λ*x*. *push-param x 1*) *ls* ! *i*) (*push a n*) = *list-eval ls a i*
  unfolding *list-eval-def* **using** *push-push1* **by** (*auto*)

end

1.3 Diophantine Relations and Predicates

theory *Diophantine-Relations*
imports *Assignments*
begin

datatype *relation* =
NARY nat list ⇒ *bool polynomial list*
| *AND relation relation* (**infixl** ‹[∧]› *35*)
| *OR relation relation* (**infixl** ‹[∨]› *30*)
| *EXIST-LIST nat relation* (‹[∃ -] -› *10*)

fun *eval* :: *relation* ⇒ *assignment* ⇒ *bool* **where**
eval (*NARY R PL*) *a* = *R* (*map* (λ*P*. *peval P a*) *PL*)
| *eval* (*AND D1 D2*) *a* = (*eval D1 a* ∧ *eval D2 a*)
| *eval* (*OR D1 D2*) *a* = (*eval D1 a* ∨ *eval D2 a*)
| *eval* ([∃ *n*] *D*) *a* = (∃ *ks*::*nat list*. *n* = *length ks* ∧ *eval D* (*push-list a ks*))

definition *is-dioph-rel* :: *relation* ⇒ *bool* **where**
is-dioph-rel DR = (∃ P_1 P_2::*ppolynomial*. ∀ *a*. (*eval DR a*) ⟷ (∃ *v*. *ppeval* P_1 *a* *v* = *ppeval* P_2 *a v*))

definition *UNARY* :: (*nat* ⇒ *bool*) ⇒ *polynomial* ⇒ *relation* **where**
UNARY R P = *NARY* (λ*l*. *R* (*l*!*0*)) [*P*]

lemma *unary-eval*: *eval* (*UNARY R P*) *a* = *R* (*peval P a*)
unfolding *UNARY-def* **by** *simp*

definition *BINARY* :: (*nat* ⇒ *nat* ⇒ *bool*) ⇒ *polynomial* ⇒ *polynomial* ⇒ *relation* **where**
BINARY R P_1 P_2 = *NARY* (λ*l*. *R* (*l*!*0*) (*l*!*1*)) [P_1, P_2]

lemma *binary-eval*: *eval* (*BINARY R* P_1 P_2) *a* = *R* (*peval* P_1 *a*) (*peval* P_2 *a*)
unfolding *BINARY-def* **by** *simp*

definition *TERNARY* :: (*nat* ⇒ *nat* ⇒ *nat* ⇒ *bool*)
⇒ *polynomial* ⇒ *polynomial* ⇒ *polynomial* ⇒ *relation* **where**
TERNARY R P_1 P_2 P_3 = *NARY* (λ*l*. *R* (*l*!*0*) (*l*!*1*) (*l*!*2*)) [P_1, P_2, P_3]

lemma *ternary-eval*: *eval* (*TERNARY R* P_1 P_2 P_3) *a* = *R* (*peval* P_1 *a*) (*peval* P_2 *a*) (*peval* P_3 *a*)
unfolding *TERNARY-def* **by** *simp*

definition *QUATERNARY* :: (*nat* ⇒ *nat* ⇒ *nat* ⇒ *nat* ⇒ *bool*)
⇒ *polynomial* ⇒ *polynomial* ⇒ *polynomial* ⇒ *polynomial* ⇒ *relation* **where**
QUATERNARY R P_1 P_2 P_3 P_4 = *NARY* (λ*l*. *R* (*l*!*0*) (*l*!*1*) (*l*!*2*) (*l*!*3*)) [P_1, P_2, P_3, P_4]

definition *EXIST* :: *relation* ⇒ *relation* (‹[∃] -› *10*) **where**
([∃] *D*) = ([∃ *1*] *D*)

definition *TRUE* **where** *TRUE* = *UNARY* ((=) *0*) (*Const 0*)

Bounded constant all quantifier (i.e. recursive conjunction)

fun *ALLC-LIST* :: *nat list* ⇒ (*nat* ⇒ *relation*) ⇒ *relation* (‹[∀ *in* -] -›) **where**
[∀ *in* []] *DF* = *TRUE* |
[∀ *in* (*l* # *ls*)] *DF* = (*DF l* [∧] [∀ *in ls*] *DF*)

lemma *ALLC-LIST-eval-list-all*: *eval* ([∀ *in L*] *DF*) *a* = *list-all* (λ*l*. *eval* (*DF l*) *a*) *L*
by (*induction L*, *auto simp*: *TRUE-def UNARY-def*)

lemma *ALLC-LIST-eval*: *eval* ([∀ *in L*] *DF*) *a* = (∀ *k*<*length L*. *eval* (*DF* (*L*!*k*)) *a*)
by (*simp add*: *ALLC-LIST-eval-list-all list-all-length*)

definition *ALLC* :: *nat* ⇒ (*nat* ⇒ *relation*) ⇒ *relation* (‹[∀<-] -›) **where**
[∀<*n*] *D* ≡ [∀ *in* [*0*..<*n*]] *D*

lemma *ALLC-eval*: *eval* ([∀<*n*] *DF*) *a* = (∀ *k*<*n*. *eval* (*DF k*) *a*)
unfolding *ALLC-def* **by** (*simp add*: *ALLC-LIST-eval*)

fun *concat* :: *'a list list* ⇒ *'a list* **where**
concat [] = [] |
concat (*l* # *ls*) = *l* @ *concat ls*

fun *splits* :: *'a list* ⇒ *nat list* ⇒ *'a list list* **where**
splits L [] = [] |
splits L (*n* # *ns*) = (*take n L*) # (*splits* (*drop n L*) *ns*)

lemma *split-concat*:
splits (*map f* (*concat pls*)) (*map length pls*) = *map* (*map f*) *pls*
by (*induction pls*, *auto*)

definition *LARY* :: (*nat list list* ⇒ *bool*) ⇒ (*polynomial list list*) ⇒ *relation* **where**
LARY R PLL = *NARY* (λ*l*. *R* (*splits l* (*map length PLL*))) (*concat PLL*)

lemma *LARY-eval*:
fixes *PLL* :: *polynomial list list*
shows *eval* (*LARY R PLL*) *a* = *R* (*map* (*map* (λ*P*. *peval P a*)) *PLL*)
unfolding *LARY-def* **apply** (*induction PLL*, *simp*)
subgoal for *pl pls* **by** (*induction pl*, *auto simp*: *split-concat*)
done

lemma *or-dioph*:
assumes *is-dioph-rel A* **and** *is-dioph-rel B*
shows *is-dioph-rel* (*A* [∨] *B*)

proof −
  from *assms* **obtain** *PA1 PA2* **where** *PA*: ∀ *a*. (*eval A a*) ⟷ (∃ *v*. *ppeval PA1 a v* = *ppeval PA2 a v*)
    by (*auto simp*: *is-dioph-rel-def*)
  from *assms* **obtain** *PB1 PB2* **where** *PB*: ∀ *a*. (*eval B a*) ⟷ (∃ *v*. *ppeval PB1 a v* = *ppeval PB2 a v*)
    by (*auto simp*: *is-dioph-rel-def*)

  show *?thesis*
    unfolding *is-dioph-rel-def*
    apply (*rule exI*[*of - PA1* ∗ *PB1* + *PA2* ∗ *PB2*])
    apply (*rule exI*[*of - PA1* ∗ *PB2* + *PA2* ∗ *PB1*])
    using *PA PB* **by** (*auto*) (*metis crossproduct-eq add.commute*)+

qed

lemma *exists-disjoint-vars*:
  fixes *Q1 Q2* :: *ppolynomial*
  fixes *A* :: *relation*
  assumes *is-dioph-rel A*
  shows ∃ *P1 P2*. *disjoint-var* (*P1* + *P2*) (*Q1* + *Q2*)
        ∧ (∀ *a*. *eval A a* ⟷ (∃ *v*. *ppeval P1 a v* = *ppeval P2 a v*))
proof −
  obtain *P1 P2* **where** *p-defs*: ∀ *a*. *eval A a* ⟷ (∃ *v*. *ppeval P1 a v* = *ppeval P2 a v*)
    using *assms is-dioph-rel-def* **by** *auto*

  define *n*::*nat* **where** *n* ≡ *Max* (*var-set* (*Q1* + *Q2*))

  define *P1′ P2′* **where** *p′-defs*: *P1′* ≡ *push-var P1* (*Suc n*) *P2′* ≡ *push-var P2* (*Suc n*)

  have *disjoint-var* (*P1′* + *P2′*) (*Q1* + *Q2*)
  proof −
    have *finite* (*var-set* (*Q1* + *Q2*))
      apply (*induction Q1*, *auto*)
      by (*induction Q2*, *auto*)+

    hence ∀ *x* ∈ *var-set* (*Q1* + *Q2*). *x* ≤ *n*
      unfolding *n-def* **using** *Max.coboundedI* **by** *blast*

    moreover have ∀ *x* ∈ *var-set* (*P1′* + *P2′*). *x* > *n*
      unfolding *p′-defs* **using** *push-var-bound* **by** *auto*

    ultimately show *?thesis*
      unfolding *disjoint-var-def* **by** *fastforce*
  qed

moreover have ∀ *a*. *eval A a* ⟷ (∃ *v*. *ppeval P1′ a v* = *ppeval P2′ a v*)
unfolding *p′-defs* **apply** (*auto simp add*: *p-defs push-var-pull-assignment pull-assignment-def*)
subgoal for *a v* **by** (*rule exI*[*of* - *λi*. *v* (*i* − *Suc n*)]) *auto*
done

ultimately show *?thesis*
by *auto*
qed

lemma *and-dioph*:
assumes *is-dioph-rel A* **and** *is-dioph-rel B*
shows *is-dioph-rel* (*A* [∧] *B*)
proof −
from *assms*(*1*) **obtain** *PA1 PA2* **where** *PA*: ∀ *a*. (*eval A a*) ⟷ (∃ *v*. *ppeval PA1 a v* = *ppeval PA2 a v*)
by (*auto simp*: *is-dioph-rel-def*)
from *assms*(*2*) **obtain** *PB1 PB2* **where** *disj*: *disjoint-var* (*PB1* + *PB2*) (*PA1* + *PA2*)
and *PB*: (∀ *a*. *eval B a* ⟷ (∃ *v*. *ppeval PB1 a v* = *ppeval PB2 a v*))
using *exists-disjoint-vars*[*of B*] **by** *blast*

from *disjoint-var-unifies* **have** *unified*: ∀ *a*. (*eval* (*A* [∧] *B*) *a*)
⟷ (∃ *v*. *ppeval PA1 a v* = *ppeval PA2 a v* ∧ *ppeval PB1 a v* = *ppeval PB2 a v*)
using *PA PB disj disjoint-var-sym* **by** *simp blast*

have *h0*: *p1* = *p2* ⟷ *p1*^*2* + *p2*^*2* = *2*∗*p1*∗*p2* **for** *p1 p2* :: *nat*
apply (*auto simp*: *algebra-simps power2-eq-square*)
using *crossproduct-eq* **by** *fastforce*

have *p1* = *p2* ∧ *q1* = *q2* ⟷ *p1*^*2* + *p2*^*2* + *q1*^*2* + *q2*^*2* = *2*∗*p1*∗*p2* + *2*∗*q1*∗*q2* **for** *p1 p2 q1 q2* :: *nat*
proof (*rule*)
assume *p1* = *p2* ∧ *q1* = *q2*
thus $p1^2 + p2^2 + q1^2 + q2^2 = 2 * p1 * p2 + 2 * q1 * q2$
by (*auto simp*: *algebra-simps power2-eq-square*)
next
assume $p1^2 + p2^2 + q1^2 + q2^2 = 2 * p1 * p2 + 2 * q1 * q2$
hence $(int\ p1)^2 + (int\ p2)^2 + (int\ q1)^2 + (int\ q2)^2 - 2 * int\ p1 * int\ p2 - 2 * int\ q1 * int\ q2 = 0$
by (*auto*) (*smt* (*verit*, *best*) *mult-2 of-nat-add of-nat-mult power2-eq-square*)
hence (*int p1* − *int p2*)^*2* + (*int q1* − *int q2*)^*2* = *0*
by (*simp add*: *power2-diff*)
hence *int p1* = *int p2* **and** *int q1* = *int q2*
by (*simp add*: *sum-power2-eq-zero-iff*)+

thus *p1 = p2 ∧ q1 = q2*
by *auto*
qed

thus *?thesis*
apply (*simp only*: *is-dioph-rel-def*)
apply (*rule exI*[*of - PA1*^**2** + *PA2*^**2** + *PB1*^**2** + *PB2*^**2**])
apply (*rule exI*[*of -* (*ppolynomial.Const 2*) ∗ *PA1* ∗ *PA2* + (*ppolynomial.Const 2*) ∗ *PB1* ∗ *PB2*])
apply (*subst unified*)
by (*simp add*: *Sq-pp-def power2-eq-square*)
qed

definition *eq* (**infix** ‹[=]› *50*) **where** *eq Q R ≡ BINARY* (=) *Q R*
definition *lt* (**infix** ‹[<]› *50*) **where** *lt Q R ≡ BINARY* (<) *Q R*
definition *le* (**infix** ‹[≤]› *50*) **where** *le Q R ≡ Q* [<] *R* [∨] *Q* [=] *R*
definition *gt* (**infix** ‹[>]› *50*) **where** *gt Q R ≡ R* [<] *Q*
definition *ge* (**infix** ‹[≥]› *50*) **where** *ge Q R ≡ Q* [>] *R* [∨] *Q* [=] *R*

named-theorems *defs*
lemmas [*defs*] = *zero-p-def one-p-def eq-def lt-def le-def gt-def ge-def LARY-eval UNARY-def BINARY-def TERNARY-def QUATERNARY-def ALLC-LIST-eval ALLC-eval*

named-theorems *dioph*
lemmas [*dioph*] = *or-dioph and-dioph*

lemma *true-dioph*[*dioph*]: *is-dioph-rel TRUE*
unfolding *TRUE-def UNARY-def is-dioph-rel-def* **by** *auto*

lemma *eq-dioph*[*dioph*]: *is-dioph-rel* (*Q* [=] *R*)
unfolding *is-dioph-rel-def*
apply (*rule exI*[*of - convert Q*])
apply (*rule exI*[*of - convert R*])
using *convert-eval BINARY-def* **by** (*auto simp*: *eq-def*)

lemma *lt-dioph*[*dioph*]: *is-dioph-rel* (*Q* [<] *R*)
unfolding *is-dioph-rel-def*
apply (*rule exI*[*of -* (*ppolynomial.Const 1*) + (*ppolynomial.Var 0*) + *convert Q*])
apply (*rule exI*[*of - convert R*])
using *convert-eval BINARY-def* **apply** (*auto simp*: *lt-def*)
by (*metis add.commute add.right-neutral less-natE*)

definition *zero* (‹[*0*=] -› [*60*] *60*) **where**[*defs*]: *zero Q ≡* **0** [=] *Q*
lemma *zero-dioph*[*dioph*]: *is-dioph-rel* ([*0*=] *Q*)
unfolding *zero-def* **by** (*auto simp*: *eq-dioph*)

lemma *gt-dioph*[*dioph*]: *is-dioph-rel* (*Q* [>] *R*)
unfolding *gt-def* **by** (*auto simp*: *lt-dioph*)

lemma *le-dioph*[*dioph*]: *is-dioph-rel* (*Q* [≤] *R*)
unfolding *le-def* **by** (*auto simp*: *lt-dioph eq-dioph or-dioph*)

lemma *ge-dioph*[*dioph*]: *is-dioph-rel* (*Q* [≥] *R*)
unfolding *ge-def* **by** (*auto simp*: *gt-dioph eq-dioph or-dioph*)

Bounded Constant All Quantifier, dioph rules

lemma *ALLC-LIST-dioph*[*dioph*]: *list-all* (*is-dioph-rel* ∘ *DF*) *L* ⟹ *is-dioph-rel* ([∀ *in L*] *DF*)
by (*induction L*, *auto simp add*: *dioph*)

lemma *ALLC-dioph*[*dioph*]: ∀ *i*<*n*. *is-dioph-rel* (*DF i*) ⟹ *is-dioph-rel* ([∀<*n*] *DF*)
unfolding *ALLC-def* **using** *ALLC-LIST-dioph*[*of DF* [*0*..<*n*]] **by** (*auto simp*: *list-all-length*)

end

1.4 Existential quantification is Diophantine

theory *Existential-Quantifier*
imports *Diophantine-Relations*
begin

lemma *exist-list-dioph*[*dioph*]:
fixes *D*
assumes *is-dioph-rel D*
shows *is-dioph-rel* ([∃ *n*] *D*)
proof (*induction n*)
case *0*
then show *?case*
using *assms* **unfolding** *is-dioph-rel-def* **by** (*auto simp*: *push-list-empty*)
next
case (*Suc n*)

have *h*: (λ*i*. *if i* = *0 then v 0 else v i*) = *v* **for** *v*::*assignment*
by *auto*

have *eval* ([∃ *Suc n*] *D*) *a* = (∃ *k*::*nat*. *eval* ([∃ *n*] *D*) (*push a k*)) **for** *a*
apply (*simp add*: *push-list2*)
by (*smt* (*z3*) *Zero-not-Suc add-Suc-right append-Nil2 length-Cons length-append list.size*(*3*) *nat.inject rev-exhaust*)
moreover from *Suc is-dioph-rel-def* **obtain** P_1 P_2 **where**
∀ *a*. *eval* ([∃ *n*] *D*) *a* = (∃ *v*. *ppeval* P_1 *a v* = *ppeval* P_2 *a v*)
by *auto*

ultimately have *t1*: *eval* ([∃ *Suc n*] *D*) *a* = (∃ *k*::*nat*. (∃ *v*. *ppeval* P_1 (*push a k*) *v*
= *ppeval* P_2 (*push a k*) *v*)) **for** *a*
by *simp*

define *f* :: *ppolynomial* ⇒ *ppolynomial* **where**
f ≡ λ*P*. *pull-param* (*push-var P 1*) (*Var 0*)
have *ppeval P* (*push a k*) *v* = *ppeval* (*f P*) *a* (*push v k*) **for** *P a k v*
apply (*induction P*, *auto simp*: *push-def f-def*)
by (*metis* (*no-types*, *lifting*) *Suc-pred ppeval.simps*(*2*) *pull-param.simps*(*2*))
then have *t2*: *eval* ([∃ *Suc n*] *D*) *a* = (∃ *k*::*nat*. (∃ *v*. *ppeval* (*f* P_1) *a* (*push v k*)
= *ppeval* (*f* P_2) *a* (*push v k*))) **for** *a*
using *t1* **by** *auto*
moreover have (∃ *k*::*nat*. ∃ *v*. *ppeval P a* (*push v k*) = *ppeval Q a* (*push v k*))
⟷ (∃ *v*. *ppeval P a v* = *ppeval Q a v*) **for** *P Q a*
unfolding *push-def*
apply *auto*
subgoal for *v*
apply (*rule exI*[*of - v 0*])
apply (*rule exI*[*of -* λ*i*. *v* (*i* + *1*)])
by (*auto simp*: *h cong*: *if-cong*)
done
ultimately have *eval* ([∃ *Suc n*] *D*) *a* = (∃ *v*. *ppeval* (*f* P_1) *a v* = *ppeval* (*f* P_2) *a v*) **for** *a*
by *auto*

thus *?case*
unfolding *is-dioph-rel-def* **by** *auto*
qed

lemma *exist-dioph*[*dioph*]:
fixes *D*
assumes *is-dioph-rel D*
shows *is-dioph-rel* ([∃] *D*)
unfolding *EXIST-def* **using** *assms* **by** (*auto simp*: *exist-list-dioph*)

lemma *exist-eval*[*defs*]:
shows *eval* ([∃] *D*) *a* = (∃ *k*. *eval D* (*push a k*))
unfolding *EXIST-def* **apply** (*simp add*: *push-list-def*)
by (*metis length-Suc-conv list.exhaust list.size*(*3*) *nat.simps*(*3*) *push-list-singleton*)

end

1.5 Mod is Diophantine

theory *Modulo-Divisibility*
imports *Existential-Quantifier*
begin

Divisibility is diophantine

```
definition dvd (‹DVD - -› 1000) where DVD Q R ≡ (BINARY (dvd) Q R)

lemma dvd-repr:
  fixes a b :: nat
  shows a dvd b ⟷ (∃x. x ∗ a = b)
  using dvd-class.dvd-def by auto

lemma dvd-dioph[dioph]: is-dioph-rel (DVD Q R)
proof −
  define Q' R' where pushed-defs: Q' ≡ push-param Q 1 R' ≡ push-param R 1
  define D where D ≡ [∃] (Param 0 [∗] Q' [=] R')

  have eval (DVD Q R) a = eval D a for a
   unfolding D-def pushed-defs defs using push-push1 apply (auto simp: push0)
    unfolding dvd-def by (auto simp: dvd-repr binary-eval)

  moreover have is-dioph-rel D
    unfolding D-def by (auto simp: dioph)

  ultimately show ?thesis
    by (auto simp: is-dioph-rel-def)
qed

declare dvd-def[defs]


definition mod (‹MOD - - -› 1000)
  where MOD A B C ≡ (TERNARY (λa b c. a mod b = c mod b) A B C)
declare mod-def[defs]

lemma mod-repr:
  fixes a b c :: nat
  shows a mod b = c mod b ⟷ (∃x y. c + x∗b = a + y∗b)
  by (metis mult.commute nat-mod-eq-iff)

lemma mod-dioph[dioph]:
  fixes A B C
  defines D ≡ (MOD A B C)
  shows is-dioph-rel D
proof −
  define A' B' C' where pushed-defs: A' ≡ push-param A 2 B' ≡ push-param B
2 C' ≡ push-param C 2
  define DS where DS ≡ [∃2] (Param 0 [∗] B' [+] C' [=] Param 1 [∗] B' [+]
A')

  have eval DS a = eval D a for a
  proof
    show eval DS a ⟹ eval D a
    unfolding DS-def defs D-def mod-def
```

by *auto* (*metis mod-mult-self3 push-push-simp pushed-defs*(*1*) *pushed-defs*(*2*) *pushed-defs*(*3*))
show *eval D a* ⟹ *eval DS a*
unfolding *DS-def defs D-def mod-def*
apply (*auto simp add*: *mod-repr*)
subgoal for *x y*
apply (*rule exI*[*of* - [*x*, *y*]])
unfolding *pushed-defs* **by** (*simp add*: *push-push*[**where** *?n* = *2*] *push-list-eval*)
done
qed

moreover have *is-dioph-rel DS*
unfolding *DS-def* **by** (*simp add*: *dioph*)

ultimately show *?thesis*
by (*auto simp*: *is-dioph-rel-def*)
qed

declare *mod-def*[*defs*]

end

2 Exponentiation is Diophaninte

2.1 Expressing Exponentiation in terms of the alpha function

theory *Exponentiation*
imports *Complex-Main*
begin

locale *Exp-Matrices*
begin

2.1.1 2x2 matrices and operations

datatype *mat2* = *mat* (*mat-11* : *int*) (*mat-12* : *int*) (*mat-21* : *int*) (*mat-22* : *int*)
datatype *vec2* = *vec* (*vec-1*: *int*) (*vec-2*: *int*)

fun *mat-plus*:: *mat2* ⇒ *mat2* ⇒ *mat2* **where**
mat-plus A B = *mat* (*mat-11 A* + *mat-11 B*) (*mat-12 A* + *mat-12 B*)
(*mat-21 A* + *mat-21 B*) (*mat-22 A* + *mat-22 B*)

fun *mat-mul*:: *mat2* ⇒ *mat2* ⇒ *mat2* **where**
mat-mul A B = *mat* (*mat-11 A* ∗ *mat-11 B* + *mat-12 A* ∗ *mat-21 B*)
(*mat-11 A* ∗ *mat-12 B* + *mat-12 A* ∗ *mat-22 B*)
(*mat-21 A* ∗ *mat-11 B* + *mat-22 A* ∗ *mat-21 B*)
(*mat-21 A* ∗ *mat-12 B* + *mat-22 A* ∗ *mat-22 B*)

fun *mat-pow*:: *nat* ⇒ *mat2* ⇒ *mat2* **where**
  mat-pow 0 - = mat 1 0 0 1 |
  mat-pow n A = mat-mul A (mat-pow (n − 1) A)

lemma *mat-pow-2*[*simp*]: *mat-pow 2 A = mat-mul A A*
  by (*simp add*: *numeral-2-eq-2*)

fun *mat-det*::*mat2* ⇒ *int* **where**
  mat-det M = mat-11 M ∗ mat-22 M − mat-12 M ∗ mat-21 M

fun *mat-scalar-mult*::*int* ⇒ *mat2* ⇒ *mat2* **where**
  mat-scalar-mult a M = mat (a ∗ mat-11 M) (a ∗ mat-12 M) (a ∗ mat-21 M) (a ∗ mat-22 M)

fun *mat-minus*:: *mat2* ⇒ *mat2* ⇒ *mat2* **where**
  mat-minus A B = mat (mat-11 A − mat-11 B) (mat-12 A − mat-12 B)
      (mat-21 A − mat-21 B) (mat-22 A − mat-22 B)

fun *mat-vec-mult*:: *mat2* ⇒ *vec2* ⇒ *vec2* **where**
  mat-vec-mult M v = vec (mat-11 M ∗ vec-1 v + mat-12 M ∗ vec-2 v)
      (mat-21 M ∗ vec-1 v + mat-21 M ∗ vec-2 v)

definition *ID* :: *mat2* **where** *ID = mat 1 0 0 1*
declare *mat-det.simps*[*simp del*]

2.1.2 Properties of 2x2 matrices

lemma *mat-neutral-element*: *mat-mul ID N = N* **by** (*auto simp*: *ID-def*)

lemma *mat-associativity*: *mat-mul (mat-mul D B) C = mat-mul D (mat-mul B C)*
  apply *auto* **by** *algebra+*

lemma *mat-exp-law*: *mat-mul (mat-pow n M) (mat-pow m M) = mat-pow (n+m) M*
  apply (*induction n*, *auto*) **by** (*metis mat2.sel(1,2) mat-associativity mat-mul.simps*)+

lemma *mat-exp-law-mult*: *mat-pow (n∗m) M = mat-pow n (mat-pow m M)* (**is** *?P n*)
  apply (*induction n*, *auto*) **using** *mat-exp-law* **by** (*metis mat-mul.simps*)

lemma *det-mult*: *mat-det (mat-mul M1 M2) = (mat-det M1) ∗ (mat-det M2)*
  by (*auto simp*: *mat-det.simps algebra-simps*)

2.1.3 Special second-order recurrent sequences

Equation 3.2

fun *α*:: *nat* ⇒ *nat* ⇒ *int* **where**
    α b 0 = 0 |

```
    α b (Suc 0) = 1 |
  alpha-n: α b (Suc (Suc n)) = (int b) ∗ (α b (Suc n)) − (α b n)
```

Equation 3.3

```
lemma alpha-strictly-increasing:
  shows int b ≥ 2 =⇒ α b n < α b (Suc n) ∧ 0 < α b (Suc n)
proof (induct n)
  case 0
  show ?case by simp
next
  case (Suc n)
  have pos: 0 < α b (Suc n)
    using Suc by fastforce
  have α b (Suc n) ≤ (int b) ∗ (α b (Suc n)) − α b (Suc n) using pos Suc by
simp
  also have ... < α b (Suc (Suc n)) using Suc by fastforce
  finally show ?case using pos Suc by simp
qed

lemma alpha-strictly-increasing-general:
  fixes b n m::nat
  assumes b > 2 ∧ m > n
  shows α b m > α b n
proof −
  from alpha-strictly-increasing assms have S2: α b n < α b m
    by (smt less-imp-of-nat-less lift-Suc-mono-less of-nat-0-less-iff pos2)
  show ?thesis using S2 by simp
qed
```

Equation 3.4

```
lemma alpha-superlinear: b > 2 =⇒ int n ≤ α b n
  apply (induction n, auto)
  by (smt Suc-1 alpha-strictly-increasing less-imp-of-nat-less of-nat-1 of-nat-Suc)
```

A simple consequence that's often useful; could also be generalized to alpha using alpha linear

```
lemma alpha-nonnegative:
  shows b > 2 =⇒ α b n ≥ 0
  using of-nat-0-le-iff alpha-superlinear order-trans by blast
```

Equation 3.5

```
lemma alpha-linear: α 2 n = n
proof(induct n rule: nat-less-induct)
  case (1 n)
  have s0: n=0 =⇒ α 2 n = n by simp
  have s1: n=1 =⇒ α 2 n = n by simp
  note hyp = ‹∀ m < n. α 2 m = m›
  from hyp have s2: n>1 =⇒ α 2 (n−1) = n−1 ∧ α 2 (n−2) = n−2 by simp
```

```
  have s3: n>1 ⟹ α 2 (Suc (Suc (n−2))) = 2∗α 2 (Suc (n−2)) − α 2 (n−2)
by simp
  have s4: n>1 ⟹ Suc (Suc (n−2)) = n by simp
  have s5: n>1 ⟹ Suc (n−2) = n−1 by simp
  from s3 s4 s5 have s6: n>1 ⟹ α 2 n = 2∗α 2 (n−1) − α 2 (n−2) by simp
  from s2 s6 have s7: n>1 ⟹ α 2 n = 2∗(n−1) − (n−2) by simp
  from s7 have s8: n>1 ⟹ α 2 n = n by simp
  from s0 s1 s8 show ?case by linarith
qed
```

Equation 3.6 (modified)

```
lemma alpha-exponential-1: b > 0 ⟹ int b ^ n ≤ α (b + 1) (n+1)
proof(induction n)
case 0
  thus ?case by(simp)
next
  case (Suc n)
  hence ((int b)∗(int b)^n) ≤ (int b)∗(α (b+1) (n+1)) by simp
  hence r2: ((int b)^(Suc n)) ≤ (int (b+1))∗(α (b+1) (n+1)) − (α (b+1) (n+1))

    by (simp add: algebra-simps)
  have (int b+1) ∗(α (b+1) (n+1)) − (α (b+1) (n+1)) ≤ (int b+1)∗(α (b+1)
(n+1)) − α (b+1) n
   using alpha-strictly-increasing Suc by (smt Suc-eq-plus1 of-nat-0-less-iff of-nat-Suc)
  thus ?case using r2 by auto
qed

lemma alpha-exponential-2: int b>2 ⟹ α b (n+1) ≤ (int b)^(n)
proof(induction n)
  case 0
  thus ?case by simp
next
  case (Suc n)
  hence s1: α b (n+2) ≤ (int b)^(n+1) − α b n by simp
  have (int b)^(n+1) − (α b n) ≤ (int b)^(n+1)
   using alpha-strictly-increasing Suc by (smt α.simps(1) alpha-superlinear of-nat-1
of-nat-add
                                            of-nat-le-0-iff of-nat-less-iff one-add-one)
  thus ?case using s1 by simp
qed
```

2.1.4 First order relation

Equation 3.7 - Definition of A

```
fun A :: nat ⇒ nat ⇒ mat2 where
  A b 0 = mat 1 0 0 1 |
  A-n: A b n = mat (α b (n + 1)) (−(α b n)) (α b n) (−(α b (n − 1)))
```

Equation 3.9 - Definition of B

fun *B* :: *nat* ⇒ *mat2* **where**
B b = *mat* (*int b*) (−*1*) *1 0*

declare *A.simps*[*simp del*]
declare *B.simps*[*simp del*]

Equation 3.8

lemma *A-rec*: *b>2* ⟹ *A b* (*Suc n*) = *mat-mul* (*A b n*) (*B b*)
by (*induction n*, *auto simp*: *A.simps B.simps*)

Equation 3.10

lemma *A-pow*: *b>2* ⟹ *A b n* = *mat-pow n* (*B b*)
apply (*induction n*, *auto simp*: *A.simps B.simps*)
subgoal by (*smt A.elims Suc-eq-plus1 α.simps α.simps*(*2*) *mat2.sel*)
subgoal for *n* **apply** (*cases n=0*, *auto*)
using *A.simps*(*2*)[*of b n−1*] *gr0-conv-Suc mult.commute* **by** *auto*
subgoal by (*metis A.simps*(*2*) *Suc-eq-plus1 α.simps*(*2*) *mat2.sel*(*1*) *mat-pow.elims*)
subgoal by (*metis A.simps*(*2*) *α.simps*(*1*) *add.inverse-neutral mat2.sel*(*2*) *mat-pow.elims*)
done

2.1.5 Characteristic equation

Equation 3.11

lemma *A-det*: *b>2* ⟹ *mat-det* (*A b n*) = *1*
apply (*auto simp*: *A-pow*, *induction n*, *simp add*: *mat-det.simps*)
using *det-mult* **apply** (*auto simp del*: *mat-mul.simps*) **by** (*simp add*: *B.simps mat-det.simps*)

Equation 3.12

lemma *alpha-det1*:
assumes *b>2*
shows (*α b* (*Suc n*))^*2* − (*int b*) ∗ *α b* (*Suc n*) ∗ *α b n* + (*α b n*)^*2* = *1*
proof(*cases n* = *0*)
case *True*
thus *?thesis* **by** *auto*
next
case *False*
hence *A b n* = *mat* (*α b* (*n* + *1*)) (−(*α b n*)) (*α b n*) (−(*α b* (*n* − *1*))) **using** *A.elims neq0-conv* **by** *blast*
hence *mat-det* (*A b n*) = (*α b n*)^*2* − (*α b* (*Suc n*)) ∗ *α b* (*n−1*)
apply (*auto simp*: *mat-det.simps*) **by** (*simp add*: *power2-eq-square*)
moreover hence ... = (*α b* (*Suc n*))^*2* − *b* ∗ (*α b* (*Suc n*)) ∗ *α b n* + (*α b n*)^*2*
using *False alpha-n*[*of b n−1*] **apply**(*auto simp add*: *algebra-simps*)
by (*metis Suc-1 distrib-left mult.commute mult.left-commute power-Suc power-one-right*)
ultimately show *?thesis* **using** *A-det assms* **by** *auto*
qed

Equation 3.12

lemma *alpha-det2*:
  assumes *b*>*2* *n*>*0*
  shows $(\alpha\ b\ (n-1))\hat{}2 - (int\ b) * (\alpha\ b\ (n-1) * (\alpha\ b\ n)) + (\alpha\ b\ n)\hat{}2 = 1$
  using *alpha-det1 assms* **by** (*smt One-nat-def Suc-diff-Suc diff-zero mult.commute mult.left-commute*)

Equations 3.14 to 3.17

lemma *alpha-char-eq*:
  fixes *x y b*:: *nat*
  shows $(y < x \wedge x * x + y * y = 1 + b * x * y) \Longrightarrow (\exists m.\ int\ y = \alpha\ b\ m \wedge int\ x = \alpha\ b\ (Suc\ m))$
proof (*induct y arbitrary*: *x rule*:*nat-less-induct*)
  case (*1 n*)

  note *pre* = ‹$n < x \wedge (x * x + n * n = 1 + b * x * n)$›

  have *h0*: $int\ (x * x + n * n) = int\ (x * x) + int\ (n * n)$ **by** *simp*
  from *pre h0* **have** *pre1*: $int\ x * int\ x + int(n * n) = int\ 1 + int(b * x * n)$ **by** *simp*

  have *i0*: $int\ (n * n) = int\ n * int\ n$ **by** *simp*
  have *i1*: $int\ (b * x * n) = int\ b * int\ x * int\ n$ **by** *simp*
  from *pre1 i0 i1* **have** *pre2*: $int\ x * int\ x + int\ n * int\ n = 1 + int\ b * int\ x * int\ n$ **by** *simp*

  from *pre2* **have** *j0*: $int\ n * int\ n - 1 = int\ b * int\ x * int\ n - int\ x * int\ x$ **by** *simp*
  have *j1*:... $= int\ x * (int\ b * int\ n - int\ x)$ **by** (*simp add*: *right-diff-distrib*)
  from *j0 j1* **have** *pre3*:$int\ n * int\ n - 1 = int\ x * (int\ b * int\ n - int\ x)$ **by** *simp*

  have *k0*: $int\ n * int\ n - 1 < int\ n * int\ n$ **by** *simp*
  from *pre3 k0* **have** *k1*:$int\ n * int\ n > int\ x * (int\ b * int\ n - int\ x)$ **by** *simp*
  from *pre* **have** *k2*: $int\ n \le int\ x$ **by** *simp*
  from *k2* **have** *k3*: $int\ x * int\ n \ge int\ n * int\ n$ **by** (*simp add*: *mult-mono*)
  from *k1 k3* **have** *k4*: $int\ x * int\ n > int\ x * (int\ b * int\ n - int\ x)$ **by** *linarith*
  from *pre k4* **have** *k5*: $int\ n > int\ b * int\ n - int\ x$ **by** *simp*

  from *pre* **have** *l0*:$n = 0 \Longrightarrow x = 1$ **by** *simp*
  from *l0* **have** *l1*: $n = 0 \Longrightarrow x = Suc\ 0$ **by** *simp*
  from *l1* **have** *l2*: $n = 0 \Longrightarrow int\ n = \alpha\ b\ 0 \wedge int\ x = \alpha\ b\ (Suc\ 0)$ **by** *simp*
  from *l2* **have** *l3*: $n = 0 \Longrightarrow \exists m.\ int\ n = \alpha\ b\ m \wedge int\ x = \alpha\ b\ (Suc\ m)$ **by** *blast*

  have *m0*: $n > 0 \Longrightarrow int\ n * int\ n - 1 \ge 0$ **by** *simp*
  from *pre3 m0* **have** *m1*: $n > 0 \Longrightarrow int\ x * (int\ b * int\ n - int\ x) \ge 0$ **by** *simp*
  from *m1* **have** *m2*: $n > 0 \Longrightarrow int\ b * int\ n - int\ x \ge 0$ **using** *zero-le-mult-iff* **by** *force*

  from *j0* **have** *n0*: $int\ x * int\ x - int\ b * int\ x * int\ n + int\ n * int\ n = 1$ **by**

```
simp
  have n1: (int b ∗ int n − int x) ∗ (int b ∗ int n − int x) = int b ∗ int n ∗ (int
b ∗ int n − int x) − int x ∗ (int b ∗ int n − int x) by (simp add: left-diff-distrib)
  from n1 have n2: int n ∗ int n − int b ∗ int n ∗ (int b ∗ int n − int x) + (int
b ∗ int n − int x) ∗ (int b ∗ int n − int x) = int n ∗ int n − int x ∗ (int b ∗ int
n − int x) by simp
  from n0 n2 j1 have n3: int n ∗ int n − int b ∗ int n ∗ (int b ∗ int n − int x)
+ (int b ∗ int n − int x) ∗ (int b ∗ int n − int x) = 1 by linarith
  from n3 have n4: int n ∗ int n + (int b ∗ int n − int x) ∗ (int b ∗ int n − int
x) = 1 + int b ∗ int n ∗ (int b ∗ int n − int x) by simp
  have n5: int b ∗ int n = int (b ∗ n) by simp
  from n5 m2 have n6: n > 0 ⟹ int b ∗ int n − int x = int (b ∗ n − x) by
linarith
  from n4 n6 have n7: n > 0 ⟹ int (n ∗ n + (b ∗ n − x) ∗ (b ∗ n − x)) = int
(1 + b ∗ n ∗ (b ∗ n − x)) by simp
  from n7 have n8: n > 0 ⟹ n ∗ n + (b ∗ n − x) ∗ (b ∗ n − x) = 1 + b ∗ n
∗ (b ∗ n − x) using of-nat-eq-iff by blast

  note hyp = ‹∀ m<n. ∀ x. m < x ∧ x ∗ x + m ∗ m = 1 + b ∗ x ∗ m ⟶
        (∃ ma. int m = α b ma ∧ int x = α b (Suc ma))›

  from k5 n6 n8 have o0: n > 0 ⟹ (b ∗ n − x) < n ∧ n ∗ n + (b ∗ n − x) ∗
(b ∗ n − x) = 1 + b ∗ n ∗ (b ∗ n − x) by simp
  from o0 hyp have o1: n > 0 ⟹ (∃ ma. int (b ∗ n − x) = α b ma ∧ int n = α
b (Suc ma)) by simp

  from o1 l3 n6 show ?case by force
qed

lemma alpha-char-eq2:
  assumes (x∗x + y∗y = 1 + b ∗ x ∗ y) b>2
  shows  (∃ n. int x = α b n)
proof −
  have x ≠ y
  proof(rule ccontr, auto)
    assume x=y
    hence 2∗x∗x = 1+b∗x∗x using assms by simp
   hence 2∗x∗x ≥ 1+2∗x∗x using assms by (metis add-le-mono le-less mult-le-mono1)
    thus False by auto
  qed
  thus ?thesis
  proof(cases x<y)
    case True
    hence ∃ n. int x = α b n ∧ int y = α b (Suc n) using alpha-char-eq assms
      by (simp add: add.commute power2-eq-square)
    thus ?thesis by auto
  next
    case False
    hence ∃ j. int y = α b j ∧ int x = α b (Suc j) using alpha-char-eq assms ‹x ≠
```

```
y› by auto
    thus ?thesis by blast
  qed
qed
```

2.1.6 Divisibility properties

The following lemmas are needed in the proof of equation 3.25

```
lemma representation:
  fixes k m :: nat
  assumes k > 0 n = m mod k l = (m−n)div k
 shows m = n+k∗l ∧ 0≤n ∧ n≤k−1 by (metis Suc-pred′ assms le-add2 le-add-same-cancel2

     less-Suc-eq-le minus-mod-eq-div-mult minus-mod-eq-mult-div mod-div-mult-eq
     mod-less-divisor neq0-conv nonzero-mult-div-cancel-left)

lemma div-3251:
  fixes b k m:: nat
  assumes b>2 and k>0
  defines n ≡ m mod k
  defines l ≡ (m−n) div k
  shows A b m = mat-mul (A b n) (mat-pow l (A b k))
proof −
  from assms(2) l-def n-def representation have m: m = n+k∗l ∧ 0≤n ∧ n≤k−1
by simp
  from A-pow assms(1) have Abm2: A b m = mat-pow m (B b) by simp
  from m have Bm: mat-pow m (B b) =mat-pow (n+k∗l) (B b) by simp
  from mat-exp-law have as1: mat-pow (n+k∗l) (B b)
                         = mat-mul (mat-pow n (B b)) (mat-pow (k∗l) (B b)) by simp
  from mat-exp-law-mult have as2: mat-pow (k∗l) (B b) = mat-pow l (mat-pow k
(B b))
    by (metis mult.commute)
  from A-pow assms have Abn: mat-pow n (B b) = A b n by simp
  from A-pow assms(1) have Ablk: mat-pow l (mat-pow k (B b)) = mat-pow l (A
b k) by simp
 from Ablk Abm2 Abn Bm as1 as2 show Abm: A b m = mat-mul (A b n) (mat-pow
l (A b k)) by simp
qed

lemma div-3252:
  fixes a b c d m :: int and l :: nat
  defines M ≡ mat a b c d
  assumes mat-21 M mod m = 0
  shows (mat-21 (mat-pow l M)) mod m = 0  (is ?P l)
proof(induction l)
  show ?P 0 by simp
next
  fix l assume IH: ?P l
  define Ml where Ml = mat-pow l M
```

have *S1*: *mat-pow* (*Suc*(*l*)) *M* = *mat-mul M* (*mat-pow l M*) **by** *simp*
have *S2*: *mat-21* (*mat-mul M Ml*) = *mat-21 M* ∗ *mat-11 Ml* + *mat-22 M* ∗ *mat-21 Ml*
by (*rule-tac mat-mul.induct mat-plus.induct*, *auto*)
have *S3*: *mat-21* (*mat-pow* (*Suc*(*l*)) *M*) = *mat-21 M* ∗ *mat-11 Ml* + *mat-22 M* ∗ *mat-21 Ml*
using *S1 S2 Ml-def* **by** *simp*
from *assms*(*2*) **have** *S4*: (*mat-21 M* ∗ *mat-11 Ml*) *mod m* = *0* **by** *auto*
from *IH Ml-def* **have** *S5*: *mat-22 M* ∗ *mat-21 Ml mod m* = *0* **by** *auto*
from *S4 S5* **have** *S6*: (*mat-21 M* ∗ *mat-11 Ml* + *mat-22 M* ∗ *mat-21 Ml*) *mod m* = *0* **by** *auto*
from *S3 S6* **show** *?P* (*Suc*(*l*)) **by** *simp*
qed

lemma *div-3253*:
fixes *a b c d m*:: *int* **and** *l* :: *nat*
defines *M* ≡ *mat a b c d*
assumes *mat-21 M mod m* = *0*
shows ((*mat-11* (*mat-pow l M*)) − *a*^*l*) *mod m* = *0* (**is** *?P l*)
proof(*induction l*)
show *?P 0* **by** *simp*
next
fix *l* **assume** *IH*: *?P l*
define *Ml* **where** *Ml* = *mat-pow l M*
from *Ml-def* **have** *S1*: *mat-pow* (*Suc*(*l*)) *M* = *mat-mul M Ml* **by** *simp*
have *S2*: *mat-11* (*mat-mul M Ml*) = *mat-11 M* ∗ *mat-11 Ml* + *mat-12 M* ∗ *mat-21 Ml*
by (*rule-tac mat-mul.induct mat-plus.induct*, *auto*)
hence *S3*: *mat-11* (*mat-pow* (*Suc*(*l*)) *M*) = *mat-11 M* ∗ *mat-11 Ml* + *mat-12 M* ∗ *mat-21 Ml*
using *S1* **by** *simp*
from *M-def Ml-def assms*(*2*) *div-3252* **have** *S4*: *mat-21 Ml mod m* = *0* **by** *auto*
from *IH Ml-def* **have** *S5*: (*mat-11 Ml* − *a* ^ *l*) *mod m* = *0* **by** *auto*
from *IH M-def* **have** *S6*: (*mat-11 M* −*a*) *mod m* = *0* **by** *simp*
from *S4* **have** *S7*: (*mat-12 M* ∗ *mat-21 Ml*) *mod m* = *0* **by** *auto*
from *S5 S6* **have** *S8*: (*mat-11 M* ∗ *mat-11 Ml*− *a*^(*Suc*(*l*))) *mod m* = *0*
by (*metis M-def mat2.sel*(*1*) *mod-0 mod-mult-right-eq mult-zero-right power-Suc right-diff-distrib*)
have *S9*: (*mat-11 M* ∗ *mat-11 Ml* − *a*^(*Suc*(*l*)) + *mat-12 M* ∗ *mat-21 Ml*) *mod m* = *0*
using *S7 S8* **by** *auto*
from *S9* **have** *S10*: (*mat-11 M* ∗ *mat-11 Ml* + *mat-12 M* ∗ *mat-21 Ml* − *a*^(*Suc*(*l*))) *mod m* = *0* **by** *smt*
from *S3 S10* **show** *?P* (*Suc*(*l*)) **by** *auto*
qed

Equation 3.25

lemma *divisibility-lemma1*:
fixes *b k m*:: *nat*

```
  assumes b>2 and k>0
  defines n ≡ m mod k
  defines l ≡ (m−n) div k
  shows α b m mod α b k = α b n ∗ (α b (k+1)) ^ l mod α b k
proof −
  from assms(2) l-def n-def representation have m: m = n+k∗l ∧ 0≤n ∧ n≤k−1
by simp
  consider (eq0) n = 0 | (neq0) n > 0 by auto
  thus ?thesis
  proof cases
    case eq0
    have Abm-gen: A b m = mat-mul (A b n) (mat-pow l (A b k))
      using assms div-3251 l-def n-def by blast
    have Abk: mat-pow l (A b k) = mat-pow l (mat (α b (k+1)) (−α b k) (α b k)
(−α b (k−1)))
      using assms(2) neq0-conv by (metis A.elims)
    from eq0 have Abm: A b m = mat-pow l (mat (α b (k+1)) (−α b k) (α b k)
(−α b (k−1)))
      using A-pow ‹b>2› apply (auto simp: A.simps B.simps)
     by (metis Abk Suc-eq-plus1 add.left-neutral m mat-exp-law-mult mult.commute)
   have Abm1: mat-21 (A b m) = α b m by (metis A.elims α.simps(1) mat2.sel(3))
     have Abm2: mat-21 (mat-pow l (mat (α b (k+1)) (−α b k) (α b k) (−α b
(k−1)))) mod (α b k) = 0
      using Abm div-3252 by simp
    from Abm Abm1 Abm2 have MR0: α b m mod α b k = 0 by simp
    from MR0 eq0 show ?thesis by simp
  next case neq0
    from assms have Abm-gen: A b m = mat-mul (A b n) (mat-pow l (A b k))
      using div-3251 l-def n-def by blast
    from assms(2) neq0-conv have Abk: mat-pow l (A b k)
              = mat-pow l (mat (α b (k+1)) (−α b k) (α b k) (−α b (k−1))) by
(metis A.elims)
  from n-def neq0 have N0: n>0 by simp
  define M where M = mat (α b (n + 1)) (−(α b n)) (α b n) (−(α b (n − 1)))
   define N where N = mat-pow l (mat (α b (k+1)) (−α b k) (α b k) (−α b
(k−1)))
  from Suc-pred' neq0 have Abn: A b n = mat (α b (n + 1)) (−(α b n)) (α b n)
(−(α b (n − 1)))
    by (metis A.elims neq0-conv)
   from Abm-gen Abn Abk M-def N-def have Abm: A b m = mat-mul M N by
simp

  from Abm have S1: mat-21 (mat-mul M N) = mat-21 M ∗ mat-11 N + mat-22
M ∗ mat-21 N
    by (rule-tac mat-mul.induct mat-plus.induct, auto)
  have S2: mat-21 (A b m) = α b m by (metis A.elims α.simps(1) mat2.sel(3))
   from S1 S2 Abm have S3: α b m = mat-21 M ∗ mat-11 N + mat-22 M ∗
mat-21 N by simp
  from S3 have S4: (α b m − (mat-21 M ∗ mat-11 N + mat-22 M ∗ mat-21 N))
```

mod (*α b k*) = *0* **by** *simp*
  from *M-def* **have** *S5*: *mat-21 M* = *α b n* **by** *simp*
  from *div-3253 N-def* **have** *S6*: (*mat-11 N* − (*α b* (*k*+*1*)) ^ *l*) *mod* (*α b k*) = *0* **by** *simp*
  from *N-def Abm div-3252* **have** *S7*: *mat-21 N mod* (*α b k*) = *0* **by** *simp*
  from *S4 S7* **have** *S8*: (*α b m* − *mat-21 M* ∗ *mat-11 N*) *mod* (*α b k*) = *0* **by** *algebra*
  from *S5 S6* **have** *S9*: (*mat-21 M* ∗ *mat-11 N* − (*α b n*) ∗ (*α b* (*k*+*1*)) ^ *l*) *mod* (*α b k*) = *0*
    by (*metis mod-0 mod-mult-left-eq mult.commute mult-zero-left right-diff-distrib′*)
  from *S8 S9* **show** *?thesis*
  proof −
    have (*mat-21 M* ∗ *mat-11 N* − *α b m*) *mod α b k* = *0*
      using *S8* **by** *presburger*
    hence ∀ *i*. (*α b m* − (*mat-21 M* ∗ *mat-11 N* − *i*)) *mod α b k* = *i mod α b k*
      by (*metis* (*no-types*) *add.commute diff-0-right diff-diff-eq2 mod-diff-right-eq*)
    thus *?thesis*
      by (*metis* (*no-types*) *S9 diff-0-right mod-diff-right-eq*)
  qed
    qed
  qed

Prerequisite lemma for 3.27

lemma *div-coprime*:
  assumes *b*>*2 n* ≥ *0*
    shows *coprime* (*α b k*) (*α b* (*k*+*1*)) (**is** *?P*)
proof(*rule ccontr*)
  assume *as*: ¬ *?P*
  define *n* **where** *n* = *gcd* (*α b k*) (*α b* (*k*+*1*))
  from *n-def* **have** *S1*: *n* > *1*
    using *alpha-det1 as assms*(*1*) *coprime-iff-gcd-eq-1 gcd-pos-int right-diff-distrib′*

    by (*smt add.commute plus-1-eq-Suc*)
  have *S2*: (*α b* (*Suc k*))^*2* − (*int b*) ∗ *α b* (*Suc k*) ∗ (*α b k*) + (*α b k*)^*2* = *1*
    using *alpha-det1 assms* **by** *auto*
  from *n-def* **have** *D1*: *n dvd* (*α b* (*k*+*1*))^*2* **by** (*simp add*: *numeral-2-eq-2*)
  from *n-def* **have** *D2*: *n dvd* (− (*int b*) ∗ *α b* (*k*+*1*) ∗ (*α b k*)) **by** *simp*
  from *n-def* **have** *D3*: *n dvd* (*α b k*)^*2* **by** (*simp add*: *gcd-dvdI1*)
  have *S3*: *n dvd* ((*α b* (*Suc k*))^*2* − (*int b*) ∗ *α b* (*Suc k*) ∗ (*α b k*) + (*α b k*)^*2*)

    using *D1 D2 D3* **by** *simp*
  from *S2 S3* **have** *S4*: *n dvd 1* **by** *simp*
  from *S4 n-def as is-unit-gcd* **show** *False* **by** *blast*
qed

Equation 3.27

lemma *divisibility-lemma2*:
  fixes *b k m*:: *nat*
  assumes *b*>*2* **and** *k*>*0*

```
  defines n ≡ m mod k
  defines l ≡ (m−n) div k
  assumes α b k dvd α b m
    shows α b k dvd α b n
proof −
  from assms(2) l-def n-def representation have m: 0≤n ∧ n≤k−1 by simp
  from divisibility-lemma1 assms(1) assms(2) l-def n-def have S1:
    (α b m) mod (α b k)  = (α b n) ∗ (α b (k+1)) ^ l mod (α b k) by blast
  from S1 assms(5) have S2: (α b k) dvd ((α b n) ∗ (α b (k+1)) ^ l) by auto
  show ?thesis
    using S1 div-coprime S2 assms(1) apply auto
    using coprime-dvd-mult-left-iff coprime-power-right-iff by blast
qed
```

Equation 3.23 - main result of this section

```
theorem divisibility-alpha:
  assumes b>2 and k>0
    shows α b k dvd α b m ⟷ k dvd m (is ?P ⟷ ?Q)
proof
  assume Q: ?Q
  define n where n = m mod k
  have N: n=0 by (simp add: Q n-def)
  from N have Abn: α b n = 0 by simp
  from Abn divisibility-lemma1 assms(1) assms(2) mult-eq-0-iff n-def show ?P
    by (metis dvd-0-right dvd-imp-mod-0 mod-0-imp-dvd)
next
  assume P: ?P
  define n where n = m mod k
  define l where l = (m−n) div k
  define B where B = (mat (int b) (−1) 1 0)
  have S1: (α b n) mod (α b k) = 0
    using divisibility-lemma2 assms(1) assms(2) n-def P by simp
  from n-def assms(2) have m: n < k using mod-less-divisor by blast
  from alpha-strictly-increasing m assms(1) have S2: α b n < α b k
    by (smt less-imp-of-nat-less lift-Suc-mono-less of-nat-0-less-iff pos2)
  from S1 S2 have S3: n=0
   by (smt alpha-superlinear assms(1) mod-pos-pos-trivial neq0-conv of-nat-0-less-iff)
  from S3 n-def show ?Q by auto
qed
```

2.1.7 Divisibility properties (continued)

Equation 3.28 - main result of this section

```
lemma divisibility-equations:
  assumes 0: m = k∗l and b>2 m>0
  shows  A b m = mat-pow l (mat-minus (mat-scalar-mult (α b k) (B b))
                                       (mat-scalar-mult (α b (k−1)) ID))
   apply (auto simp del: mat-pow.simps mat-mul.simps mat-minus.simps mat-scalar-mult.simps
               simp add: A-pow mult.commute[of k l] assms mat-exp-law-mult)
```

```
    using A-pow[of b k] ‹m>0›
    apply (auto simp: A.simps ‹m>0› ID-def B.simps)
     using A.simps(2) alpha-n One-nat-def Suc-eq-plus1 Suc-pred assms ‹m>0›
assms
        mult.commute nat-0-less-mult-iff
    by (smt mat-exp-law-mult)

lemma divisibility-cong:
  fixes e f :: int
  fixes l :: nat
  fixes M :: mat2
  assumes mat-22 M = 0 mat-21 M = 1
  shows (mat-21 (mat-pow l (mat-minus (mat-scalar-mult e M) (mat-scalar-mult
f ID)))) mod e^2 = (−1)^(l−1)∗l∗e∗f^(l−1)∗(mat-21 M) mod e^2
        ∧ mat-22 (mat-pow l (mat-minus (mat-scalar-mult e M) (mat-scalar-mult f
ID))) mod e^2 = (−1)^l ∗f^l mod e^2
(is ?P l ∧ ?Q l )
proof(induction l)
  case 0
  then show ?case by simp
next
  case (Suc l)
  have S2: mat-pow (Suc(l)) (mat-minus (mat-scalar-mult e M) (mat-scalar-mult
f ID)) =
    mat-mul (mat-pow l (mat-minus (mat-scalar-mult e M) (mat-scalar-mult f ID)))
(mat-minus (mat-scalar-mult e M) (mat-scalar-mult f ID))
    using mat-exp-law[of l - 1] mat2.sel by (auto, metis)+
 define a1 where a1 = mat-11 (mat-minus (mat-scalar-mult e M) (mat-scalar-mult
f ID))
 define b1 where b1 = mat-12 (mat-minus (mat-scalar-mult e M) (mat-scalar-mult
f ID))
 define c1 where c1 = mat-21 (mat-minus (mat-scalar-mult e M) (mat-scalar-mult
f ID))
 define d1 where d1 = mat-22 (mat-minus (mat-scalar-mult e M) (mat-scalar-mult
f ID))
  define a where a = mat-11 M
  define b where b = mat-12 M
  define c where c = mat-21 M
  define d where d = mat-22 M
   define g where g = mat-21 (mat-pow l (mat-minus (mat-scalar-mult e M)
(mat-scalar-mult f ID)))
   define h where h = mat-22 (mat-pow l (mat-minus (mat-scalar-mult e M)
(mat-scalar-mult f ID)))
 from S2 g-def a1-def h-def c1-def have S3: mat-21 (mat-pow (Suc(l)) (mat-minus
(mat-scalar-mult e M) (mat-scalar-mult f ID))) = g∗a1 + h∗c1
    by simp
 from S2 g-def b1-def h-def d1-def have S4: mat-22 (mat-pow (Suc(l)) (mat-minus
(mat-scalar-mult e M) (mat-scalar-mult f ID)))
    = g∗b1+h∗d1 by simp
```

have S5: mat-11 (mat-scalar-mult e M) = e*a **by** (simp add: a-def)
have S6: mat-12 (mat-scalar-mult e M) = e*b **by** (simp add: b-def)
have S7: mat-21 (mat-scalar-mult e M) = e*c **by** (simp add: c-def)
have S8: mat-22 (mat-scalar-mult e M) = e*d **by** (simp add: d-def)
from a1-def S5 **have** S9: a1 = e*a−f**by** (simp add: Exp-Matrices.ID-def)
from b1-def S6 **have** S10: b1 = e*b **by** (simp add: Exp-Matrices.ID-def)
from c1-def S7 **have** S11: c1 = e*c **by** (simp add: Exp-Matrices.ID-def)
from S11 assms(2) c-def **have** S115: c1 = e **by** simp
from d1-def S8 **have** S12: d1 = e*d − f **by** (simp add: Exp-Matrices.ID-def)
from S12 assms(1) d-def **have** S125: d1 = − f **by** simp
from assms(2) c-def Suc g-def c-def **have** S13: g mod e^2 = (−1)^(l−1)*l*e*f^(l−1)*c mod e^2 **by** blast
from assms(2) c-def S13 **have** S135: g mod e^2 = (−1)^(l−1)*l*e*f^(l−1) mod e^2 **by** simp
from Suc h-def **have** S14: h mod e^2 = (−1)^l *f^l mod e^2 **by** simp
from S10 S135 **have** S27: g*b1 mod e^2 = (−1)^(l−1)*l*e*f^(l−1)*e*b mod e^2 **by** (metis mod-mult-left-eq mult.assoc)
from S27 **have** S28: g*b1 mod e^2 = 0 mod e^2 **by** (simp add: power2-eq-square)
from S125 S14 mod-mult-cong **have** S29: h*d1 mod e^2 = (−1)^l *f^l*(− f) mod e^2 **by** blast
from S29 **have** S30: h*d1 mod e^2 = (−1)^(l+1) *f^l*f mod e^2 **by** simp
from S30 **have** S31: h*d1 mod e^2 = (−1)^(l+1) *f^(l+1) mod e^2 **by** (metis mult.assoc power-add power-one-right)
from S31 **have** F2: ?Q (Suc(l)) **by** (metis S28 S4 Suc-eq-plus1 add.left-neutral mod-add-cong)
from S9 S13 **have** S15: g*a1 mod e^2 = ((−1)^(l−1)*l*e*f^(l−1)*c*(e*a−f))mod e^2 **by** (metis mod-mult-left-eq)
have S16: ((−1)^(l−1)*l*e*f^(l−1)*c*(e*a−f)) = ((−1)^(l−1)*l*e^2*f^(l−1)*c*a) − f*(−1)^(l−1)*l*e*f^(l−1)*c **by** algebra
have S17: ((−1)^(l−1)*l*e^2*f^(l−1)*c*a) mod e^2 = 0 mod e^2 **by** simp
from S17 **have** S18: (((−1)^(l−1)*l*e^2*f^(l−1)*c*a) − f*(−1)^(l−1)*l*e*f^(l−1)*c) mod e^2 =
− f*(−1)^(l−1)*l*e*f^(l−1)*c mod e^2
proof −
have f1: ∀ i ia. (ia::int) − (0 − i) = ia + i
by auto
have ∀ i ia. ((0::int) − ia) * i = 0 − ia * i
by auto
then show ?thesis **using** f1
proof −
have f1: ⋀i. (0::int) − i = − i
by presburger
then have ⋀i. (i − − ((− 1) ^ (l − 1) * int l * e^2 * f ^ (l − 1) * c * a)) mod e^2 = i mod e^2
by (metis (no-types) S17 ‹∀ i ia. ia − (0 − i) = ia + i› add.right-neutral mod-add-right-eq)
then have ⋀i. ((− 1) ^ (l − 1) * int l * e^2 * f ^ (l − 1) * c * a − i) mod e^2 = − i mod e^2
using f1 **by** (metis ‹∀ i ia. ia − (0 − i) = ia + i› uminus-add-conv-diff)

then show *?thesis*
using *f1* ‹∀ *i ia*. (*0* − *ia*) ∗ *i* = *0* − *ia* ∗ *i*› **by** *presburger*
qed
qed
from *S15 S16 S18* **have** *S19*: *g*∗*a1 mod e*^*2* = − *f*∗(−*1*)^(*l*−*1*)∗*l*∗*e*∗*f*^(*l*−*1*)∗*c* *mod e*^*2* **by** *presburger*
from *S11 S14* **have** *S20*: *h*∗*c1 mod e*^*2* = (−*1*)^*l* ∗*f*^*l*∗*e*∗*c mod e*^*2* **by** (*metis mod-mult-left-eq mult.assoc*)
from *S19 S20* **have** *S21*: (*g*∗*a1* + *h*∗*c1*) *mod e*^*2* = (− *f*∗(−*1*)^(*l*−*1*)∗*l*∗*e*∗*f*^(*l*−*1*)∗*c* + (−*1*)^*l* ∗*f*^*l*∗*e*∗*c*) *mod e*^*2* **using** *mod-add-cong* **by** *blast*
from *assms*(*2*) *c-def* **have** *S22*: (− *f*∗(−*1*)^(*l*−*1*)∗*l*∗*e*∗*f*^(*l*−*1*)∗*c* + (−*1*)^*l* ∗*f*^*l*∗*e*∗*c*) *mod e*^*2*=(− *f*∗(−*1*)^(*l*−*1*)∗*l*∗*e*∗*f*^(*l*−*1*) + (−*1*)^*l* ∗*f*^*l*∗*e*) *mod e*^*2* **by** *simp*
have *S23*: (− *f*∗(−*1*)^(*l*−*1*)∗*l*∗*e*∗*f*^(*l*−*1*) + (−*1*)^*l* ∗*f*^*l*∗*e*) *mod e*^*2* = (*f*∗(−*1*)^(*l*)∗*l*∗*e*∗*f*^(*l*−*1*) + (−*1*)^*l* ∗*f*^*l*∗*e*) *mod e*^*2*
by (*smt One-nat-def Suc-pred mult.commute mult-cancel-left2 mult-minus-left neq0-conv of-nat-eq-0-iff power.simps*(*2*))
have *S24*: (*f*∗(−*1*)^(*l*)∗*l*∗*e*∗*f*^(*l*−*1*) + (−*1*)^*l* ∗*f*^*l*∗*e*) *mod e*^*2* = ((−*1*)^(*l*)∗*l*∗*e*∗*f*^*l* + (−*1*)^*l* ∗*f*^*l*∗*e*) *mod e*^*2*
by (*smt One-nat-def Suc-pred mult.assoc mult.commute mult-eq-0-iff neq0-conv of-nat-eq-0-iff power.simps*(*2*))
have *S25*: ((−*1*)^(*l*)∗*l*∗*e*∗*f*^*l* + (−*1*)^*l* ∗*f*^*l*∗*e*) *mod e*^*2* = ((−*1*)^(*l*)∗(*l*+*1*)∗*e*∗*f*^*l*) *mod e*^*2*
proof −
have *f1*: ∀ *i ia*. (*ia*::*int*) ∗ *i* = *i* ∗ *ia*
by *simp*
then have *f2*: ∀ *i ia*. (*ia*::*int*) ∗ *i* − − *i* = *i* ∗ (*ia* − − *1*)
by (*metis* (*no-types*) *mult.right-neutral mult-minus-left right-diff-distrib'*)
have ∀ *n*. *int n* − − *1* = *int* (*n* + *1*)
by *simp*
then have *e* ∗ (*f* ^ *l* ∗ (*int l* ∗ (− *1*) ^ *l* − − ((− *1*) ^ *l*))) *mod* e^2 = *e* ∗ (*f* ^ *l* ∗ ((− *1*) ^ *l* ∗ *int* (*l* + *1*))) *mod* e^2
using *f2* **by** *presburger*
then have ((− *1*) ^ *l* ∗ *int l* ∗ *e* ∗ *f* ^ *l* − − ((− *1*) ^ *l*) ∗ *f* ^ *l* ∗ *e*) *mod* e^2 = (− *1*) ^ *l* ∗ *int* (*l* + *1*) ∗ *e* ∗ *f* ^ *l mod* e^2
using *f1*
proof −
have *f1*: ⋀*i ia ib*. (*i*::*int*) ∗ (*ia* ∗ *ib*) = *ia* ∗ (*i* ∗ *ib*)
by *simp*
then have ⋀*i ia ib*. (*i*::*int*) ∗ (*ia* ∗ *ib*) − − (*i* ∗ *ib*) = (*ia* − − *1*) ∗ (*i* ∗ *ib*)
by (*metis* (*no-types*) ‹∀ *i ia*. *ia* ∗ *i* = *i* ∗ *ia*› *f2*)
then show *?thesis*
using *f1* **by** (*metis* (*no-types*) ‹∀ *i ia*. *ia* ∗ *i* = *i* ∗ *ia*› ‹*e* ∗ (*f* ^ *l* ∗ (*int l* ∗ (− *1*) ^ *l* − − ((− *1*) ^ *l*))) *mod* e^2 = *e* ∗ (*f* ^ *l* ∗ ((− *1*) ^ *l* ∗ *int* (*l* + *1*))) *mod* e^2› *f2 mult-minus-right*)
qed
then show *?thesis*
by *simp*
qed

```
 from S21 S22 S23 S24 S25 have S26: (g*a1 + h*c1) mod e^2 = ((−1)^(l)*(l+1)*e*f^l)
mod e^2 by presburger
  from S3 S26 have F1: ?P (Suc(l)) by (metis Suc-eq-plus1 assms(2) diff-Suc-1
mult.right-neutral)
  from F1 F2 show ?case by simp
qed

lemma divisibility-congruence:
  assumes m = k*l and b>2 m>0
  shows α b m mod (α b k)^2 = ((−1)^(l−1)*l*(α b k)*(α b (k−1))^(l−1)) mod
(α b k)^2
proof −
   have S0: α b m = mat-21 (A b m) by (metis A.elims assms(3) mat2.sel(3)
neq0-conv)
  from assms S0 divisibility-equations have S1: α b m =
    mat-21 ( mat-pow l (mat-minus (mat-scalar-mult (α b k) (B b))
                                     (mat-scalar-mult (α b (k−1)) ID))) by auto
 have S2: mat-21 (B b) = 1 using Binomial.binomial-ring by (simp add: Exp-Matrices.B.simps)
  have S3: mat-22 (B b) = 0  by (simp add: Exp-Matrices.B.simps)
  from S1 S2 S3 divisibility-cong show ?thesis by (metis mult.right-neutral)
qed
```

Main result section 3.5

```
theorem divisibility-alpha2:
  assumes b>2 m>0
  shows (α b k)^2 dvd (α b m) ⟷ k*(α b k) dvd m (is ?P ⟷ ?Q)
proof
  assume Q: ?Q
  then show ?P
  proof(cases k dvd m)
    case True
    then obtain l where mkl: m = k * l by blast
    from Q assms mkl have S0: l mod α b k = 0 by simp
   from S0 have S1: l*(α b k) mod (α b k)^2 = 0 by (simp add: power2-eq-square)
    from S1 have S2: ((−1)^(l−1)*l*(α b k)*(α b (k−1))^(l−1)) mod (α b k)^2
= 0
      proof −
        have ∀i. α b k * (int l * i) mod (α b k)² = 0
       by (metis (no-types) S1 mod-0 mod-mult-left-eq mult.assoc mult.left-commute
mult-zero-left)
        then show ?thesis
          by (simp add: mult.assoc mult.left-commute)
    qed
    from assms divisibility-congruence mkl have S3:
      α b m mod (α b k)^2 = ((−1)^(l−1)*l*(α b k)*(α b (k−1))^(l−1)) mod (α
b k)^2 by simp
    from S2 S3 have S4: α b m mod (α b k)^2 = 0 by linarith
    then show ?thesis by auto
  next
```

case *False*
then show *?thesis* **using** *Q dvd-mult-left int-dvd-int-iff* **by** *blast*
qed
next
assume *P*: *?P*
show *?Q*
proof(*cases k dvd m*)
case *True*
then obtain *l* **where** *mkl*: *m = k ∗ l* **by** *blast*
from *assms mkl divisibility-congruence* **have** *S0*:
α b m mod (α b k)^2 = ((−1)^(l−1)∗l∗(α b k)∗(α b (k−1))^(l−1)) mod (α b k)^2 **by** *simp*
from *S0 P* **have** *S1*: *(α b k)^2 dvd ((−1)^(l−1)∗l∗(α b k)∗(α b (k−1))^(l−1))* **by** *auto*
from *S1* **have** *S2*: *(α b k)^2 dvd l∗(α b k)∗(α b (k−1))^(l−1)*
by (*metis* (*no-types*, *opaque-lifting*) *Groups.mult-ac*(*1*) *dvd-trans dvd-triv-right left-minus-one-mult-self*)
from *S2* **have** *S3*: *(α b k) dvd l∗(α b (k−1))^(l−1)*
by (*metis* (*full-types*) *Exp-Matrices.alpha-superlinear assms*(*1*) *assms*(*2*) *mkl*

mult.assoc mult.commute mult-0 not-less-zero of-nat-le-0-iff power2-eq-square zdvd-mult-cancel)
from *div-coprime Suc-eq-plus1 Suc-pred′ assms*(*1*) *assms*(*2*) *mkl less-imp-le-nat nat-0-less-mult-iff*
have *S4*: *coprime (α b k) (α b (k−1))* **by** (*metis coprime-commute*)
hence *coprime (α b k) ((α b (k−1))^(l−1))* **using** *coprime-power-right-iff* **by** *blast*
hence *(α b k) dvd l* **using** *S3* **using** *coprime-dvd-mult-left-iff* **by** *blast*
then show *?thesis* **by** (*simp add*: *mkl*)
next
case *False*
then show *?thesis*
apply(*cases 0<k*)
subgoal using *divisibility-alpha*[*of b k m*] *assms* **using** *dvd-mult-left P* **by** *auto*
subgoal using *Exp-Matrices.alpha-strictly-increasing-general assms*(*1*) *P* **by** *fastforce*
done
qed
qed

2.1.8 Congruence properties

In this section we will need the inverse matrices of A and B

fun *A-inv* :: *nat ⇒ nat ⇒ mat2* **where**
A-inv b n = mat (−α b (n−1)) (α b n) (−α b n) (α b (n+1))

fun *B-inv* :: *nat ⇒ mat2* **where**
B-inv b = mat 0 1 (−1) b

lemma *A-inv-aux*: $b>2 \Longrightarrow n>0 \Longrightarrow \alpha\ b\ n * \alpha\ b\ n - \alpha\ b\ (Suc\ n) * \alpha\ b\ (n - Suc\ 0) = 1$
apply (*induction n*, *auto*) **subgoal for** *n* **using** *alpha-det1*[*of b n*] **apply** *auto*
by *algebra* **done**

lemma *A-inverse*[*simp*]: $b>2 \Longrightarrow n>0 \Longrightarrow$ *mat-mul* (*A-inv b n*) (*A b n*) = *ID*
using *mat2.expand*[*of mat-mul* (*A-inv b n*) (*A b n*) *ID*] **apply** *rule*
using *ID-def A.simps*(*2*)[*of - n−1*] *ID-def* **apply** (*auto*)
subgoal using *mat2.sel*(*1*)[*of 1 0 0 1*] **apply** (*auto*)
using *A-inv-aux*[*of b n*] **by** (*auto simp*: *mult.commute*)
subgoal by (*metis mat2.sel*(*2*))
subgoal by (*metis mat2.sel*(*3*))
subgoal using *mat2.sel*(*4*)[*of 1 0 0 1*] **apply** (*auto*)
using *A-inv-aux*[*of b n*] **by** (*auto simp*: *mult.commute*)
done

lemma *B-inverse*[*simp*]: *mat-mul* (*B b*) (*B-inv b*) = *ID* **using** *B.simps ID-def* **by** *auto*

declare *A-inv.simps B-inv.simps*[*simp del*]

Equation 3.33

lemma *congruence*:
assumes *b1 mod q = b2 mod q*
shows α *b1 n mod q* = α *b2 n mod q*
proof (*induct n rule*:*nat-less-induct*)
case (*1 n*)
note *hyps* = ‹$\forall m<n.\ \alpha\ b1\ m\ mod\ q = \alpha\ b2\ m\ mod\ q$›
have *n0*:(α *b1 0*) *mod q* = (α *b2 0*) *mod q* **by** *simp*
have *n1*:(α *b1 1*) *mod q* = (α *b2 1*) *mod q* **by** *simp*
from *hyps* **have** *s1*: $n>1 \Longrightarrow \alpha\ b1\ (n-1)\ mod\ q = \alpha\ b2\ (n-1)\ mod\ q$ **by** *auto*
from *hyps* **have** *s2*: $n>1 \Longrightarrow \alpha\ b1\ (n-2)\ mod\ q = \alpha\ b2\ (n-2)\ mod\ q$ **by** *auto*
have *s3*: $n>1 \Longrightarrow \alpha\ b1\ (Suc\ (Suc\ n)) = (int\ b1) * (\alpha\ b1\ (Suc\ n)) - (\alpha\ b1\ n)$ **by** *simp*
from *s3* **have** *s4*: $n>1 \Longrightarrow (\alpha\ b1\ n = (int\ b1*(\alpha\ b1\ (n-1)) - \alpha\ b1\ (n-2)))$
by (*smt Suc-1 Suc-diff-Suc diff-Suc-1 alpha-n lessE*)
have *sw*: $n>1 \Longrightarrow \alpha\ b2\ (Suc\ (Suc\ n)) = (int\ b2) * (\alpha\ b2\ (Suc\ n)) - (\alpha\ b2\ n)$ **by** *simp*
from *sw* **have** *sx*: $n>1 \Longrightarrow (\alpha\ b2\ n = (int\ b2*(\alpha\ b2\ (n-1)) - \alpha\ b2\ (n-2)))$
by (*smt Suc-1 Suc-diff-Suc diff-Suc-1 alpha-n lessE*)
from *n0 n1 s1 s2 s3 s4 assms*(*1*) *mod-mult-cong* **have** *s5*: $n>1$
$\Longrightarrow b1*(\alpha\ b1\ (n-1))\ mod\ q = b2*(\alpha\ b2\ (n-1))\ mod\ q$ **by** (*smt mod-mult-eq of-nat-mod*)
from *hyps* **have** *sq*: $n>1 \Longrightarrow \alpha\ b1\ (n-2)\ mod\ q = \alpha\ b2\ (n-2)\ mod\ q$ **by** *simp*
from *s5 sq* **have** *sd*: $n>1 \Longrightarrow -((\alpha\ b1\ (n-2)))\ mod\ q = -((\alpha\ b2\ (n-2)))\ mod\ q$
by (*metis mod-minus-eq*)

from *sd s5 mod-add-cong* **have** *s6*: $n>1 \Longrightarrow (b1*(\alpha\ b1\ (n-1)) - \alpha\ b1\ (n-2))$ $mod\ q$
$= (b2*(\alpha\ b2\ (n-1)) - \alpha\ b2\ (n-2))\ mod\ q$ **by** *force*
from *s4* **have** *sa*: $n>1 \Longrightarrow(b1*(\alpha\ b1\ (n-1)) - \alpha\ b1\ (n-2))\ mod\ q = (\alpha\ b1\ n)\ mod\ q$ **by** *simp*
from *sx* **have** *sb*: $n>1 \Longrightarrow(b2*(\alpha\ b2\ (n-1)) - \alpha\ b2\ (n-2))\ mod\ q = (\alpha\ b2\ n)\ mod\ q$ **by** *simp*
from *sb sa s6 sx* **have** *s7*: $n>1 \Longrightarrow (\alpha\ b1\ n)\ mod\ q = (b2*(\alpha\ b2\ (n-1)) - \alpha\ b2\ (n-2))\ mod\ q$ **by** *simp*
from *s7 sx s6* **have** *s9*: $\alpha\ b1\ n\ mod\ q = \alpha\ b2\ n\ mod\ q$
by (*metis One-nat-def* α*.simps*(*1*) α*.simps*(*2*) *less-Suc0 nat-neq-iff*)
from *s9 n0 n1* **show** *?case* **by** *simp*
qed

Equation 3.34

lemma *congruence2*:
fixes *b1* :: *nat*
assumes $b>=2$
shows $(\alpha\ b\ n)\ mod\ (b - 2) = n\ mod\ (b - 2)$
proof−
from *alpha-linear* **have** *S1*: $\alpha\ (nat\ 2)\ n = n$ **by** *simp*
define *q* **where** $q = b - (nat\ 2)$
from *q-def assms le-mod-geq* **have** *S4*: $b\ mod\ q = 2\ mod\ q$ **by** *auto*
from *assms S4 congruence* **have** *SN*: $(\alpha\ b\ n)\ mod\ q = (\alpha\ 2\ n)\ mod\ q$ **by** *blast*
from *S1 SN q-def zmod-int* **show** *?thesis* **by** *simp*
qed

lemma *congruence-jpos*:
fixes *b m j l* :: *nat*
assumes $b>2$ **and** $2*l*m+j>0$
defines $n \equiv 2*l*m+j$
shows $A\ b\ n = mat\text{-}mul\ (mat\text{-}pow\ l\ (mat\text{-}pow\ 2\ (A\ b\ m)))\ (A\ b\ j)$
proof−
from *A-pow assms*(*1*) **have** *Abm2*: $A\ b\ n = mat\text{-}pow\ n\ (B\ b)$ **by** *simp*
from *Abm2 n-def* **have** *Bn*: $mat\text{-}pow\ n\ (B\ b) =mat\text{-}pow\ (2*l*m+j)\ (B\ b)$ **by** *simp*
from *mat-exp-law* **have** *as1*: $mat\text{-}pow\ (2*l*m+j)\ (B\ b) = mat\text{-}mul\ (mat\text{-}pow\ l\ (mat\text{-}pow\ m\ (mat\text{-}pow\ 2\ (B\ b))))\ (mat\text{-}pow\ (j)\ (B\ b))$
by (*metis* (*no-types*, *lifting*) *mat-exp-law-mult mult.commute*)
from *A-pow assms*(*1*) *B.elims mult.commute mat-exp-law-mult* **have** *as2*: $mat\text{-}mul\ (mat\text{-}pow\ l\ (mat\text{-}pow\ m\ (mat\text{-}pow\ 2\ (B\ b))))\ (mat\text{-}pow\ (j)\ (B\ b))$
$= mat\text{-}mul\ (mat\text{-}pow\ l\ (mat\text{-}pow\ 2\ (A\ b\ m)))\ (A\ b\ j)$ **by** *metis*
from *as2 as1 Abm2 Bn* **show** *?thesis* **by** *auto*
qed

lemma *congruence-inverse*: $b>2 \Longrightarrow mat\text{-}pow\ (n+1)\ (B\text{-}inv\ b) = A\text{-}inv\ b\ (n+1)$
apply (*induction n*, *simp add*: *B-inv.simps*, *auto*) **by** (*auto simp add*: *B-inv.simps*)

lemma *congruence-inverse2*:
fixes *n b* :: *nat*
assumes *b>2*
shows *mat-mul* (*mat-pow n* (*B b*)) (*mat-pow n* (*B-inv b*)) = *mat 1 0 0 1*
proof(*induct n*)
case *0*
thus *?case* **by** *simp*
next
case (*Suc n*)
have *S1*: *mat-pow* (*Suc*(*n*)) (*B b*) = *mat-mul* (*B b*) (*mat-pow n* (*B b*)) **by** *simp*
have *S2*: *mat-pow* (*Suc*(*n*)) (*B-inv b*) = *mat-mul* (*mat-pow n* (*B-inv b*)) (*B-inv b*)
proof −
have ∀ *i ia ib ic*. *mat-pow 1* (*mat ic ib ia i*) = *mat ic ib ia i*
by *simp*
hence ∀ *m ma mb*. *mat-pow 1 m* = *m* ∨ *mat-mul mb m* ≠ *ma* **by** (*metis mat2.exhaust*)
thus *?thesis*
by (*metis* (*no-types*) *One-nat-def add-Suc-right diff-Suc-Suc diff-zero mat-exp-law mat-pow.simps*(*1*) *mat-pow.simps*(*2*))
qed
define *C* **where** *C*= (*B b*)
define *D* **where** *D* = *mat-pow n C*
define *E* **where** *E* = *B-inv b*
define *F* **where** *F* = *mat-pow n E*
from *S1 S2 C-def D-def E-def F-def* **have** *S3*: *mat-mul* (*mat-pow* (*Suc*(*n*)) *C*) (*mat-pow* (*Suc*(*n*)) *E*) = *mat-mul* (*mat-mul C D*) (*mat-mul F E*) **by** *simp*
from *S3 mat-associativity mat2.exhaust C-def D-def E-def F-def* **have** *S4*: *mat-mul* (*mat-pow* (*Suc*(*n*)) *C*) (*mat-pow* (*Suc*(*n*)) *E*)
= *mat-mul C* (*mat-mul* (*mat-mul D F*) *E*) **by** *metis*
from *S4 Suc.hyps mat-neutral-element C-def D-def E-def F-def* **have** *S5*: *mat-mul* (*mat-pow* (*Suc*(*n*)) *C*) (*mat-pow* (*Suc*(*n*)) *E*) = *mat-mul C E* **by** *simp*
from *S5 C-def E-def* **show** *?case* **using** *B-inverse ID-def* **by** *auto*
qed

lemma *congruence-mult*:
fixes *m* :: *nat*
assumes *b>2*
shows *n>m ==> mat-pow* (*nat*(*int n*− *int m*)) (*B b*) = *mat-mul* (*mat-pow n* (*B b*)) (*mat-pow m* (*B-inv b*))
proof(*induction n*)
case *0*
thus *?case* **by** *simp*
next
case (*Suc n*)
consider (*eqm*) *n* == *m* | (*gm*) *n* < *m* | (*lm*) *n>m* **by** *linarith*
thus *?case*
proof *cases*
case *gm*

from *Suc.prems gm not-less-eq* **show** *?thesis* **by** *simp*
next case *lm*
have *S1*: *mat-pow* (*nat*(*int* (*Suc*(*n*)) − *int m*)) (*B b*) = *mat-mul* (*B b*) (*mat-pow* (*nat*(*int n* − *int m*)) (*B b*))
by (*metis Suc.prems Suc-diff-Suc diff-Suc-1 diff-Suc-Suc mat-pow.simps*(*2*) *nat-minus-as-int*)
from *lm S1 Suc.IH* **have** *S2*: *mat-pow* (*nat*(*int* (*Suc*(*n*)) − *int m*)) (*B b*) = *mat-mul* (*B b*) (*mat-mul* (*mat-pow n* (*B b*)) (*mat-pow m* (*B-inv b*))) **by** *simp*
from *S2 mat-associativity mat2.exhaust* **have** *S3*: *mat-pow* (*nat*(*int* (*Suc*(*n*)) − *int m*)) (*B b*) = *mat-mul* (*mat-mul* (*B b*) (*mat-pow n* (*B b*))) (*mat-pow m* (*B-inv b*)) **by** *metis*
from *S3* **show** *?thesis* **by** *simp*
next case *eqm*
from *eqm* **have** *S1*: *nat*(*int* (*Suc*(*n*))− *int m*) = *1* **by** *auto*
from *S1* **have** *S2*: *mat-pow* (*nat*(*int* (*Suc*(*n*))− *int m*)) (*B b*) == *B b* **by** *simp*
from *eqm* **have** *S3*: (*mat-pow* (*Suc*(*n*)) (*B b*)) = *mat-mul* (*B b*) (*mat-pow m* (*B b*)) **by** *simp*
from *S3* **have** *S35*: *mat-mul* (*mat-pow* (*Suc*(*n*)) (*B b*)) (*mat-pow m* (*B-inv b*)) = *mat-mul* (*mat-mul* (*B b*) (*mat-pow m* (*B b*))) (*mat-pow m* (*B-inv b*)) **by** *simp*
from *mat2.exhaust S35 mat-associativity* **have** *S4*: *mat-mul* (*mat-pow* (*Suc*(*n*)) (*B b*)) (*mat-pow m* (*B-inv b*))
= *mat-mul* (*B b*) (*mat-mul* (*mat-pow m* (*B b*)) (*mat-pow m* (*B-inv b*))) **by** *smt*
from *congruence-inverse2 assms* **have** *S5*: *mat-mul* (*mat-pow m* (*B b*)) (*mat-pow m* (*B-inv b*)) = *mat 1 0 0 1* **by** *simp*
have *S6*: *mat-mul* (*B b*) (*B-inv b*) = *mat 1 0 0 1* **using** *ID-def B-inverse* **by** *auto*
from *S5 S6 eqm* **have** *S7*: *mat-mul* (*mat-pow n* (*B b*)) (*mat-pow m* (*B-inv b*)) = *mat 1 0 0 1* **by** *metis*
from *S7* **have** *S8*: *mat-mul* (*B b*) (*mat-mul* (*mat-pow n* (*B b*)) (*mat-pow m* (*B-inv b*))) == *B b* **by** *simp*
from *eqm S2 S4 S8* **show** *?thesis* **by** *simp*
qed
qed

lemma *congruence-jneg*:
fixes *b m j l* :: *nat*
assumes *b*>*2* **and** *2*∗*l*∗*m* > *j* **and** *j*>=*1*
defines *n* ≡ *nat*(*int 2*∗*l*∗*m*− *int j*)
shows *A b n* = *mat-mul* (*mat-pow l* (*mat-pow 2* (*A b m*))) (*A-inv b j*)
proof−
from *A-pow assms*(*1*) **have** *Abm2*: *A b n* = *mat-pow n* (*B b*) **by** *simp*
from *Abm2 n-def* **have** *Bn*: *A b n* = *mat-pow* (*nat*(*int 2*∗*l*∗*m*− *int j*)) (*B b*) **by** *simp*
from *Bn congruence-mult assms*(*1*) *assms*(*2*) **have** *Bn2*: *A b n* = *mat-mul* (*mat-pow* (*2*∗*l*∗*m*) (*B b*)) (*mat-pow j* (*B-inv b*)) **by** *fastforce*
from *assms*(*1*) *assms*(*3*) *congruence-inverse Bn2 add.commute le-Suc-ex* **have** *Bn3*: *A b n* = *mat-mul* (*mat-pow* (*2*∗*l*∗*m*) (*B b*)) (*A-inv b j*) **by** *smt*
from *Bn3 A-pow assms*(*1*) *mult.commute B.simps mat-exp-law-mult* **have** *as3*:

A b n = mat-mul (mat-pow l (mat-pow 2 (A b m))) (A-inv b j) **by** *metis*
  from *as3 A-pow add.commute assms(1) mat-exp-law mat-exp-law-mult* **show** *?thesis* **by** *simp*
qed

lemma *matrix-congruence*:
  fixes *Y Z* :: *mat2*
  fixes *b m j l* :: *nat*
  assumes *b>2*
  defines *X ≡ mat-mul Y Z*
  defines *a ≡ mat-11 Y* **and** *b0≡ mat-12 Y* **and** *c ≡ mat-21 Y* **and** *d ≡ mat-22 Y*
  defines *e ≡ mat-11 Z* **and** *f ≡ mat-12 Z* **and** *g ≡ mat-21 Z* **and** *h ≡ mat-22 Z*
  defines *v ≡ α b (m+1) − α b (m−1)*
  assumes *a mod v = a1 mod v* **and** *b0 mod v = b1 mod v* **and** *c mod v = c1 mod v* **and** *d mod v = d1 mod v*
  shows *mat-21 X mod v = (c1∗e+d1∗g) mod v ∧ mat-22 X mod v = (c1∗f+ d1∗h) mod v* (**is** *?P ∧ ?Q*)
proof −

  from *X-def mat2.exhaust-sel c-def e-def d-def g-def* **have** *P1*: *mat-21 X = (c∗e+d∗g)*
    using *mat2.sel* **by** *auto*
  from *assms(14) mod-mult-cong* **have** *P2*: *(c∗e) mod v = (c1∗e) mod v* **by** *blast*
  from *assms(15) mod-mult-cong* **have** *P3*: *(d∗g) mod v = (d1∗g) mod v* **by** *blast*
  from *P2 P3 mod-add-cong* **have** *P4*: *(c∗e+d∗g) mod v = (c1∗e+d1∗g) mod v* **by** *blast*
  from *P1 P4* **have** *F1*: *?P* **by** *simp*

  from *X-def mat2.exhaust-sel c-def f-def d-def h-def mat2.sel(4) mat-mul.simps*
  have *Q1*: *mat-22 X = (c∗f+d∗h)* **by** *metis*
  from *assms(14) mod-mult-cong* **have** *Q2*: *(c∗f) mod v = (c1∗f) mod v* **by** *blast*
  from *assms(15) mod-mult-cong* **have** *Q3*: *(d∗h) mod v = (d1∗h) mod v* **by** *blast*
  from *Q1 Q2 Q3 mod-add-cong* **have** *F2*: *?Q* **by** *fastforce*
  from *F1 F2* **show** *?thesis* **by** *auto*
qed

3.38

lemma *congruence-Abm*:
  fixes *b m n* :: *nat*
  assumes *b>2*
  defines *v ≡ α b (m+1) − α b (m−1)*
  shows *(mat-21 (mat-pow n (mat-pow 2 (A b m))) mod v = 0 mod v)*
  ∧ (mat-22 (mat-pow n (mat-pow 2 (A b m))) mod v = ((−1)^n) mod v) (**is** *?P n ∧ ?Q n*)
proof(*induct n*)
case *0*
  from *mat2.exhaust* **have** *S1*: *mat-pow 0 (mat-pow 2 (A b m)) = mat 1 0 0 1* **by** *simp*

thus *?case* **by** *simp*
next
case (*Suc n*)
define *Z* **where** *Z* = *mat-pow 2* (*A b m*)
define *Y* **where** *Y* = *mat-pow n Z*
define *X* **where** *X* = *mat-mul Y Z*
define *c* **where** *c* = *mat-21 Y*
define *d* **where** *d* = *mat-22 Y*
define *e* **where** *e* = *mat-11 Z*
define *f* **where** *f* = *mat-12 Z*
define *g* **where** *g* = *mat-21 Z*
define *h* **where** *h* = *mat-22 Z*
define *d1* **where** *d1* = (−*1*)^*n mod v*
from *d-def d1-def Z-def Y-def Suc.hyps* **have** *S1*: *d mod v* = *d1 mod v* **by** *simp*
from *matrix-congruence assms*(*1*) *X-def v-def c-def d-def e-def d1-def g-def S1*
have *S2*: *mat-21 X mod v* = (*c*∗*e*+*d1*∗*g*) *mod v* **by** *blast*
from *Z-def Y-def c-def Suc.hyps* **have** *S3*: *c mod v* = *0 mod v* **by** *simp*
consider (*eq0*) *m* = *0* | (*g0*) *m*>*0* **by** *blast*
hence *S4*: *g mod v* = *0*
proof *cases*
case *eq0*
from *eq0* **have** *S1*: *A b m* = *mat 1 0 0 1* **using** *A.simps* **by** *simp*
from *S1 Z-def div-3252 g-def* **show** *?thesis* **by** *simp*
next
case *g0*
from *g0 A.elims neq0-conv*
have *S1*: *A b m* = *mat* (*α b* (*m* + *1*)) (−(*α b m*)) (*α b m*) (−(*α b* (*m* − *1*)))
by *metis*
from *S1 assms*(*1*) *mat2.sel*(*3*) *mat-mul.simps mat-pow.simps*
have *S2*: *mat-21* (*mat-pow 2* (*A b m*)) = (*α b m*)∗(*α b* (*m*+*1*)) + (−*α b* (*m*−*1*))∗(*α b m*)
by (*auto*)
from *S2 g-def Z-def g0 A.elims neq0-conv*
have *S3*: *g* = (*α b* (*m*+*1*))∗(*α b m*)− (*α b m*)∗(*α b* (*m*−*1*)) **by** *simp*
from *S3 g-def v-def mod-mult-self1-is-0 mult.commute right-diff-distrib* **show** *?thesis* **by** *metis*
qed
from *S2 S3 S4 Z-def div-3252 g-def mat2.exhaust-sel mod-0* **have** *F1*: *?P* (*Suc*(*n*)) **by** *metis*

from *d-def d1-def Z-def Y-def Suc.hyps* **have** *Q1*: *d mod v* = *d1 mod v* **by** *simp*
from *matrix-congruence assms*(*1*) *X-def v-def c-def d-def f-def d1-def h-def S1*
have *Q2*: *mat-22 X mod v* = (*c*∗*f*+*d1*∗*h*) *mod v* **by** *blast*
from *Z-def Y-def c-def Suc.hyps* **have** *Q3*: *c mod v* = *0 mod v* **by** *simp*
consider (*eq0*) *m* = *0* | (*g0*) *m*>*0* **by** *blast*
hence *Q4*: *h mod v* = (−*1*) *mod v*
proof *cases*
case *eq0*
from *eq0* **have** *S1*: *A b m* = *mat 1 0 0 1* **using** *A.simps* **by** *simp*

from *eq0 v-def* **have** *S2*: $v = 1$ **by** *simp*
from *S1 S2* **show** *?thesis* **by** *simp*
next
case *g0*
from *g0 A.elims neq0-conv* **have** *S1*: $A\ b\ m = mat\ (\alpha\ b\ (m + 1))\ (-(\alpha\ b\ m))\ (\alpha\ b\ m)\ (-(\alpha\ b\ (m - 1)))$ **by** *metis*
from *S1 A-pow assms(1) mat2.sel(4) mat-exp-law mat-exp-law-mult mat-mul.simps mult-2*
have *S2*: $mat\text{-}22\ (mat\text{-}pow\ 2\ (A\ b\ m)) = (\alpha\ b\ m)*(-(\alpha\ b\ m)) + (-(\alpha\ b\ (m - 1)))*(-(\alpha\ b\ (m - 1)))$
by *auto*
from *S2 Z-def h-def* **have** *S3*: $h = -(\alpha\ b\ m)*(\alpha\ b\ m) + (\alpha\ b\ (m - 1))*(\alpha\ b\ (m - 1))$ **by** *simp*
from *v-def add.commute diff-add-cancel mod-add-self2* **have** *S4*: $(\alpha\ b\ (m - 1))\ mod\ v = \alpha\ b\ (m+1)\ mod\ v$ **by** *metis*
from *S3 S4 mod-diff-cong mod-mult-left-eq mult.commute mult-minus-right uminus-add-conv-diff*
have *S5*: $h\ mod\ v = (-(\alpha\ b\ m)*(\alpha\ b\ m) + (\alpha\ b\ (m - 1))*(\alpha\ b\ (m + 1)))\ mod\ v$ **by** *metis*
from *One-nat-def add.right-neutral add-Suc-right α.elims diff-Suc-1 g0 le-imp-less-Suc le-simps(1) neq0-conv Suc-diff-1 alpha-n*
have *S6*: $\alpha\ b\ (m + 1) = b*\ (\alpha\ b\ m) - \alpha\ b\ (m-1)$
by (*smt Suc-eq-plus1 Suc-pred′ α.elims alpha-superlinear assms(1) g0 nat.inject of-nat-0-less-iff of-nat-1 of-nat-add*)
from *S6* **have** *S7*: $(\alpha\ b\ (m - 1))*(\alpha\ b\ (m + 1)) = (int\ b) * (\alpha\ b\ (m-1) * (\alpha\ b\ m)) - (\alpha\ b\ (m-1))\hat{}2$
proof −
have *f1*: $\forall\, i\ ia.\ -((ia{::}int) * i) = ia * - i$ **by** *simp*
have $\forall\, i\ ia\ ib\ ic.\ (ic{::}int) * (ib * ia) + ib * i = ib * (ic * ia + i)$ **by** (*simp add*: *distrib-left*)
thus *?thesis* **using** *f1* **by** (*metis S6 ab-group-add-class.ab-diff-conv-add-uminus power2-eq-square*)
qed
from *S7* **have** *S8*: $(-(\alpha\ b\ m)*(\alpha\ b\ m) + (\alpha\ b\ (m - 1))*(\alpha\ b\ (m + 1)))$
$= -1*(\alpha\ b\ (m-1))\hat{}2 + (int\ b) * (\alpha\ b\ (m-1) * (\alpha\ b\ m)) - (\alpha\ b\ m)\hat{}2$ **by** (*simp add*: *power2-eq-square*)
from *alpha-det2 assms(1) g0* **have** *S9*: $-1*(\alpha\ b\ (m-1))\hat{}2 + (int\ b) * (\alpha\ b\ (m-1) * (\alpha\ b\ m)) - (\alpha\ b\ m)\hat{}2 = -1$ **by** *smt*
from *S5 S8 S9* **show** *?thesis* **by** *simp*
qed
from *Q2 Q3 Q4 Suc-eq-plus1 add.commute add.right-neutral d1-def mod-add-right-eq mod-mult-left-eq mod-mult-right-eq mult.right-neutral*
mult-minus1 mult-minus-right mult-zero-left power-Suc **have** *Q5*: $mat\text{-}22\ X\ mod\ v = (-1)\hat{}(n+1)\ mod\ v$ **by** *metis*
from *Q5 Suc-eq-plus1 X-def Y-def Z-def mat-exp-law mat-exp-law-mult mult.commute mult-2 one-add-one* **have** *F2*: *?Q* (*Suc*(*n*)) **by** *metis*
from *F1 F2* **show** *?case* **by** *blast*
qed

3.36 requires two lemmas 361 and 362

lemma *361*:
fixes *b m j l* :: *nat*
assumes *b>2*
defines $n \equiv 2{*}l{*}m + j$
defines $v \equiv \alpha\ b\ (m{+}1) - \alpha\ b\ (m{-}1)$
shows $(\alpha\ b\ n)$ *mod* $v = ((-1)\hat{}l * \alpha\ b\ j)$ *mod* v
proof −
define *Y* **where** *Y* = *mat-pow l* (*mat-pow 2* (*A b m*))
define *Z* **where** *Z* = *A b j*
define *X* **where** *X* = *mat-mul Y Z*
define *c* **where** *c* = *mat-21 Y*
define *d* **where** *d* = *mat-22 Y*
define *e* **where** *e* = *mat-11 Z*
define *g* **where** *g* = *mat-21 Z*
define *d1* **where** *d1* = $(-1)\hat{}l$ *mod v*
from *congruence-Abm assms*(*1*) *d-def v-def Y-def d1-def* **have** *S0*: *d mod v* = *d1 mod v* **by** *simp-all*
from *matrix-congruence assms*(*1*) *X-def v-def c-def d-def e-def d1-def g-def S0* **have** *S1*: *mat-21 X mod v* = $(c{*}e{+}d1{*}g)$ *mod v* **by** *blast*
from *congruence-Abm d1-def v-def mod-mod-trivial* **have** *S2*: *d1 mod v* = $(-1)\hat{}l$ *mod v* **by** *blast*
from *congruence-Abm Y-def assms*(*1*) *c-def v-def* **have** *S3*: *c mod v* = *0* **by** *simp*
from *Z-def g-def A.elims* α*.simps*(*1*) *mat2.sel*(*3*) *mat2.exhaust* **have** *S4*: $g = \alpha\ b\ j$ **by** *metis*
from *A-pow assms*(*1*) *mat-exp-law mat-exp-law-mult mult-2 mult-2-right n-def X-def Y-def Z-def* **have** *S5*: *A b n* = *X* **by** *metis*
from *S5 A.elims* α*.simps*(*1*) *mat2.sel*(*3*) *Z-def Y-def* **have** *S6*: *mat-21 X* = $\alpha\ b\ n$ **by** *metis*
from *S2 S3 S4 S6 S1 add.commute mod-0 mod-mult-left-eq mod-mult-self2 mult-zero-left zmod-eq-0-iff* **show** *?thesis* **by** *metis*
qed

lemma *362*:
fixes *b m j l* :: *nat*
assumes *b>2* **and** $2{*}l{*}m > j$ **and** *j>=1*
defines $n \equiv 2{*}l{*}m - j$
defines $v \equiv \alpha\ b\ (m{+}1) - \alpha\ b\ (m{-}1)$
shows $(\alpha\ b\ n)$ *mod* $v = -((-1)\hat{}l * \alpha\ b\ j)$ *mod* v
proof −
define *Y* **where** *Y* = *mat-pow l* (*mat-pow 2* (*A b m*))
define *Z* **where** *Z* = *A-inv b j*
define *X* **where** *X* = *mat-mul Y Z*
define *c* **where** *c* = *mat-21 Y*
define *d* **where** *d* = *mat-22 Y*
define *e* **where** *e* = *mat-11 Z*
define *g* **where** *g* = *mat-21 Z*
define *d1* **where** *d1* = $(-1)\hat{}l$ *mod v*
from *congruence-Abm assms*(*1*) *d-def v-def Y-def d1-def* **have** *S0*: *d mod v* =

d1 mod v **by** *simp-all*
from *matrix-congruence assms(1) X-def v-def c-def d-def e-def d1-def g-def S0*
have *S1*: *mat-21 X mod v* = *(c∗e+d1∗g) mod v* **by** *blast*
from *congruence-Abm d1-def v-def mod-mod-trivial* **have** *S2*: *d1 mod v* = *(−1)^l mod v* **by** *blast*
from *congruence-Abm Y-def assms(1) c-def v-def* **have** *S3*: *c mod v* = *0* **by** *simp*
from *Z-def g-def* **have** *S4*: *g* = − *α b j* **by** *simp*
from *congruence-jneg assms(1) assms(2) assms(3) n-def X-def Y-def Z-def* **have** *S5*: *A b n* = *X* **by** *(simp add: nat-minus-as-int)*
from *S5 A.elims α.simps(1) mat2.sel(3) Z-def Y-def* **have** *S6*: *mat-21 X* = *α b n* **by** *metis*
from *S2 S3 S4 S6 S1 add.commute mod-0 mod-mult-left-eq mod-mult-self2 mult-minus-right mult-zero-left zmod-eq-0-iff* **show** *?thesis* **by** *metis*
qed

Equation 3.36

lemma *36*:
fixes *b m j l* :: *nat*
assumes *b>2*
assumes $(n = 2 * l * m + j \vee (n = 2 * l * m - j \wedge 2 * l * m > j \wedge j \geq 1))$
defines $v \equiv \alpha\ b\ (m+1) - \alpha\ b\ (m-1)$
shows $(\alpha\ b\ n)\ mod\ v = \alpha\ b\ j\ mod\ v \vee (\alpha\ b\ n)\ mod\ v = -\alpha\ b\ j\ mod\ v$ **using** *assms(2)*
apply(*auto*)
subgoal using *361 assms(1) v-def*
apply(*cases even l*)
by *simp+*
subgoal using *362 assms(1) v-def*
apply(*cases even l*)
by *simp+*
done

2.1.9 Diophantine definition of a sequence alpha

definition *alpha-equations* :: *nat* ⇒ *nat* ⇒ *nat* ⇒ *nat* ⇒ *nat* ⇒ *nat* ⇒ *nat* ⇒ *nat* ⇒ *nat* ⇒ *nat* ⇒ *nat* ⇒ *bool* **where**
alpha-equations a b c r s t u v w x y = (
— 3.41 $b > 3 \wedge$
— 3.42 $u^2 + t^2 = 1 + b * u * t \wedge$
— 3.43 $s^2 + r^2 = 1 + b * s * r \wedge$
— 3.44 $r < s \wedge$
— 3.45 $u^2\ dvd\ s \wedge$
— 3.46 $v + 2 * r = (b) * s \wedge$
— 3.47 $w\ mod\ v = b\ mod\ v \wedge$
— 3.48 $w\ mod\ u = 2\ mod\ u \wedge$
— 3.49 $2 < w \wedge$
— 3.50 $x^2 + y^2 = 1 + w * x * y \wedge$

— 3.51 $2 * a < u \wedge$
— 3.52 $2 * a < v \wedge$
— 3.53 $a\ mod\ v = x\ mod\ v \wedge$
— 3.54 $2 * c < u \wedge$
— 3.55 $c\ mod\ u = x\ mod\ u$)

The sufficiency

lemma *alpha-equiv-suff*:
  fixes *a b c*::*nat*
  assumes ∃ *r s t u v w x y*. *alpha-equations a b c r s t u v w x y*
  shows *3* < *b* ∧ *int a* = (α *b c*)
proof −
  from *assms* **obtain** *r s t u v w x y* **where** *eq*: *alpha-equations a b c r s t u v w x y* **by** *auto*
  have *41*: *b* > *3* **using** *alpha-equations-def eq* **by** *auto*
  have *42*: *u*^*2* + *t* ^ *2* = *1* +*b* ∗ *u* ∗ *t* **using** *alpha-equations-def eq* **by** *auto*
  have *43*: *s*^*2* + *r* ^ *2* = *1* + *b* ∗ *s* ∗ *r* **using** *alpha-equations-def eq* **by** *auto*
  have *44*: *r* < *s* **using** *alpha-equations-def eq* **by** *auto*
  have *45*: *u* ^ *2 dvd s* **using** *alpha-equations-def eq* **by** *auto*
  have *46*: *v* + *2* ∗ *r* = *b* ∗ *s* **using** *alpha-equations-def eq* **by** *auto*
  have *47*: *w mod v* = *b mod v* **using** *alpha-equations-def eq* **by** *auto*
  have *48*: *w mod u* = *2 mod u* **using** *alpha-equations-def eq* **by** *auto*
  have *49*: *2* < *w* **using** *alpha-equations-def eq* **by** *auto*
  have *50*: *x* ^ *2* + *y* ^ *2* = *1* + *w* ∗ *x* ∗ *y* **using** *alpha-equations-def eq* **by** *auto*
  have *51*: *2* ∗ *a* < *u* **using** *alpha-equations-def eq* **by** *auto*
  have *52*: *2* ∗ *a* < *v* **using** *alpha-equations-def eq* **by** *auto*
  have *53*: *a mod v* = *x mod v* **using** *alpha-equations-def eq* **by** *auto*
  have *54*: *2* ∗ *c* < *u* **using** *alpha-equations-def eq* **by** *auto*
  have *55*: *int c mod u* = *x mod u* **using** *alpha-equations-def eq* **by** *auto*

  have *b* > *2* **using** ‹*b*>*3*› **by** *auto*
  have *u* > *0* **using** *51* **by** *auto*

Equation 3.56

  have ∃ *k*. *u* = α *b k* **using** *42 alpha-char-eq2* **by** (*simp add*: ‹*2* < *b*› *power2-eq-square*)
  then obtain *k* **where** *56*: *u* = α *b k* **by** *auto*

Equation 3.57

  have ∃ *m*. *s* = α *b m* ∧ *r* = α *b* (*m*−*1*) **using** *43 44 alpha-char-eq*[*of r s b*] *diff-Suc-1*
    by (*metis power2-eq-square*)
  then obtain *m* **where** *57*: *s* = α *b m* ∧ *r* = α *b* (*m*−*1*) **by** *auto*

  have *m-pos*: *m* ≠ *0* **using** *44 57 not-less-eq* **by** *fastforce*
  have *alpha-pos*: α *b m* > *0* **using** *44 57* **by** *linarith*

Equation 3.58

  have ∃ *n*. *x* = α *w n* **using** *50 alpha-char-eq2* **by** (*simp add*: *49 power2-eq-square*)

```
then obtain n where 58: x = α w n by auto
```

Equation 3.59

```
have ∃l j. (n = 2 ∗ l ∗ m + j ∨ n = 2 ∗ l ∗ m − j ∧ 2 ∗ l ∗ m > j ∧ j ≥ 1)
∧ j ≤ m
 proof −
   define q where q = n mod m
   obtain p where p-def: n = p ∗ m + q using mod-div-decomp q-def by auto
   have q1: q ≤ m using 44 57
    by (metis diff-le-self le-0-eq le-simps(1) linorder-not-le mod-less-divisor nat-int
q-def)
   consider (c1) even p | (c2) odd p by auto
   thus ?thesis
   proof(cases)
     case c1
     thus ?thesis using p-def q1 by blast
   next
     case c2
     obtain d where p=2∗d+1 using c2 oddE by blast
     define l where l=d+1
     hence jpt: l>0 by simp
     from ‹p=2∗d+1› l-def have c21: p=2∗l−1 by auto
     have c22: n=2∗l∗m−(m−q)
      by (metis Nat.add-diff-assoc2 add.commute c21 diff-diff-cancel diff-le-self jpt
mult-eq-if
          mult-is-0 neq0-conv p-def q1 zero-neq-numeral)
     thus ?thesis using diff-le-self
     by (metis add.left-neutral diff-add-inverse2 diff-zero less-imp-diff-less mult.right-neutral

          mult-eq-if mult-zero-right not-less zero-less-diff)
   qed
 qed


then obtain l j where 59: (n = 2 ∗ l ∗ m + j ∨ n = 2 ∗ l ∗ m − j ∧ 2 ∗ l ∗
m > j ∧ j ≥ 1) ∧ j ≤ m by auto
```

Equation 3.60

```
have 60: u dvd m
  using 45 56 57 divisibility-alpha2[of b m k] ‹b>2›
  by (metis dvd-trans dvd-triv-right int-dvd-int-iff m-pos neq0-conv of-nat-power)
```

Equation 3.61

```
have 61: v = α b (m+1)− α b (m−1)
proof−
 have v = b∗(α b m) − 2∗(α b (m−1)) using 46 57 by (metis add-diff-cancel-right′
mult-2 of-nat-add of-nat-mult)
  thus ?thesis using alpha-n[of b m−1] m-pos by auto
qed
```

Equation 3.62.1

have *a mod v = α b n mod v* **using** *53 58 47 congruence[of w v b n]* **by** (*simp add*: *zmod-int*)
hence *a mod v = α b j mod v ∨ a mod v = −α b j mod v* **using** *36[of b] 61 59* ‹*2 < b*› **by** *auto*

hence *62*: *v dvd (a+ α b j) ∨ v dvd (a − α b j)* **using** *mod-eq-dvd-iff zmod-int* **by** *auto*

Equation 3.63

have *631*: *2∗α b j ≤ 2∗ α b m* **using** *59 alpha-strictly-increasing-general[of b j m]* ‹*2 < b*› **by** *force*

have *b − 2 ≥ 2* **using** *41* **by** *simp*
moreover have *α b m > 0* **using** *44 57* **by** *linarith*
ultimately have *632*: *2 ∗ α b m ≤ (b − 2) ∗ α b m* **by** *auto*

have *(b − 2) ∗ α b m = b ∗ α b m − 2∗ α b m* **using** ‹*2 < b*›
by (*simp add*: *int-distrib(4) mult.commute of-nat-diff*)
moreover have *b ∗ α b m − 2∗ α b m < b ∗ α b m − 2 ∗ α b (m − 1)* **using** *44 57* **by** *linarith*
ultimately have *633*: *(b − 2) ∗ α b m < b ∗ α b m − 2 ∗ α b (m − 1)* **by** *auto*

have *634*: *b ∗ α b m − 2 ∗ α b (m − 1) = v* **using** *61 alpha-n[of b m−1] m-pos* **by** *simp*

have *63*: *2∗α b j < v* **using** *631 632 633 634* **by** *auto*

Equation 3.64

hence *64*: *a = α b j*
proof(*cases 0 < a + α b j*)
case *True*
moreover have *a + α b j < v* **using** *52 63* **by** *linarith*
ultimately show *?thesis* **using** *62*
apply *auto*
subgoal using *zdvd-not-zless* **by** *blast*
subgoal
by (*smt* ‹*2 < b*› *alpha-superlinear dvd-add-triv-left-iff negative-zle zdvd-not-zless*)
done
next
case *False*
hence *j = 0* **using** ‹*2 < b*› *alpha-strictly-increasing-general* **by** *force*
thus *?thesis* **using** *False* **by** *auto*
qed

Equation 3.65

have *65*: *c mod u = n mod u*

proof −
have *c mod u* = α *w n mod u* **using** *55 58 zmod-int* **by** (*simp add*:)
moreover have ... = *n mod u* **using** *48 alpha-linear congruence zmod-int* **by** *presburger*
ultimately show *?thesis* **by** *linarith*
qed

Equation 3.66

have *2* ∗ *j* ≤ *2* ∗ α *b j* ∧ *2* ∗ *a* < *u*
using *51 alpha-superlinear* ‹*b*>*2*› **by** *auto*
hence *66*: *2*∗*j* < *u* **using** *64* **by** *linarith*

Equation 3.67

have *652*: *u dvd* (*n*+*j*) ∨ *u dvd* (*n*−*j*) **using** *60 59* **by** *auto*

hence *c* = *j* **using** *66 54*
proof−
have *c* + *j* < *u* **using** *66 54* **by** *linarith*
thus *?thesis* **using** *652*
apply *auto*
subgoal
by (*metis 65 add-cancel-right-right dvd-eq-mod-eq-0 mod-add-left-eq mod-if not-add-less2 not-gr-zero*)
subgoal
by (*metis 59 60 65 66 Nat.add-diff-assoc2* ‹⟦*c* + *j* < *u*; *u dvd n* + *j*⟧ ⟹ *c* = *j*›
add-diff-cancel-right′ *add-lessD1 dvd-mult le-add2 le-less mod-less mod-nat-eqI mult-2*)
done
qed

show *?thesis* **using** ‹*b*>*3*› *64* ‹*c*=*j*› **by** *auto*
qed

3.7.2 The necessity

lemma *add-mod*:
fixes *p q* :: *int*
assumes *p mod 2* = *0 q mod 2* = *0*
shows (*p*+*q*) *mod 2* = *0* ∧ (*p*−*q*) *mod 2* = *0*
using *assms*(*1*) *assms*(*2*) **by** *auto*

lemma *one-odd*:
fixes *b n* :: *nat*
assumes *b*>*2*
shows (α *b n*) *mod 2* = *1* ∨ (α *b* (*n*+*1*)) *mod 2* = *1*
proof(*rule ccontr*)
assume *asm*: ¬(α *b n mod 2* = *1* ∨ α *b* (*n* + *1*) *mod 2* = *1*)
from *asm* **have** *step1*: (α *b n mod 2* = *0* ∧ α *b* (*n* + *1*) *mod 2* = *0*) **by** *simp*

from *step1* **have** *s1*: (α *b n*)^*2 mod 2* = *0* ∧ (α *b* (*n*+*1*))^*2 mod 2* = *0* **by** *auto*
from *step1* **have** *s2*: (*int b*)∗(α *b n*)∗(α *b* (*n*+*1*)) *mod 2* = *0* **by** *auto*
from *s1* **have** *s3*: ((α *b* (*n*+*1*))^*2* + (α *b n*)^*2*) *mod 2* = *0* **by** *auto*
from *s2 s3 add-mod* **have** *s4*: ((α *b* (*n*+*1*))^*2* + (α *b n*)^*2* − (*int b*)∗((α *b n*)∗(α *b* (*n*+*1*)))) *mod 2* = *0*
by (*simp add*: *Groups.mult-ac*(*2*) *Groups.mult-ac*(*3*))
have *s5*: (α *b* (*n*+*1*))^*2*+(α *b n*)^*2*−(*int b*)∗((α *b n*)∗(α *b* (*n*+*1*)))=(α *b* (*n*+*1*))^*2*−(*int b*)∗(α *b* (*n*+*1*)∗(α *b n*))+(α *b n*)^*2* **by** *simp*
from *s4 s5* **have** *s6*: ((α *b* (*n*+*1*))^*2*−(*int b*)∗(α *b* (*n*+*1*)∗(α *b n*))+(α *b n*)^*2*) *mod 2* = *0*
proof −
have *f1*: $(\alpha\ b\ (n + 1))^2 - int\ b * (\alpha\ b\ (n + 1) * \alpha\ b\ n) = (\alpha\ b\ (n + 1))^2 + - 1 * (int\ b * (\alpha\ b\ (n + 1) * \alpha\ b\ n))$
by *simp*
have *f2*: $(\alpha\ b\ (n + 1))^2 + - 1 * (int\ b * (\alpha\ b\ (n + 1) * \alpha\ b\ n)) + (\alpha\ b\ n)^2 = (\alpha\ b\ (n + 1))^2 + (\alpha\ b\ n)^2 + - 1 * (int\ b * (\alpha\ b\ n * \alpha\ b\ (n + 1)))$
by *simp*
have $((\alpha\ b\ (n + 1))^2 + (\alpha\ b\ n)^2 + - 1 * (int\ b * (\alpha\ b\ n * \alpha\ b\ (n + 1))))$ *mod 2* = *0*
using *s4* **by** *fastforce*
thus *?thesis* **using** *f2 f1* **by** *presburger*
qed
from *s6 alpha-det1* **show** *False* **by** (*simp add*: *assms mult.assoc*)
qed

lemma *oneodd*:
fixes *b n* :: *nat*
assumes *b*>*2*
shows *odd* (α *b n*) = *True* ∨ *odd* (α *b* (*n*+*1*)) = *True*
using *assms odd-iff-mod-2-eq-one one-odd* **by** *auto*

lemma *cong-solve-nat*: *a* ≠ *0* ⟹ ∃ *x*. (*a*∗*x*) *mod n* = (*gcd a n*) *mod n*
for *a n* :: *nat*
apply (*cases n*=*0*)
apply *auto*
apply (*insert bezout-nat* [*of a n*], *auto*)
by (*metis mod-mult-self4*)

lemma *cong-solve-coprime-nat*: *coprime* (*a*::*nat*) (*n*::*nat*) ⟹ ∃ *x*. (*a*∗*x*) *mod n* = *1 mod n*
using *cong-solve-nat*[*of a n*] *coprime-iff-gcd-eq-1*[*of a n*] **by** *fastforce*

lemma *chinese-remainder-aux-nat*:
fixes *m1 m2* :: *nat*
assumes *a*:*coprime m1 m2*
shows ∃ *b1 b2*. *b1 mod m1* = *1 mod m1* ∧ *b1 mod m2* = *0 mod m2* ∧ *b2 mod m1* = *0 mod m1* ∧ *b2 mod m2* = *1 mod m2*
proof −

from *cong-solve-coprime-nat* [*OF a*] **obtain** *x1* **where** *1*: (*m1*∗*x1*) *mod m2* = *1 mod m2* **by** *auto*
from *a* **have** *b*: *coprime m2 m1*
by (*simp add*: *coprime-commute*)
from *cong-solve-coprime-nat* [*OF b*] **obtain** *x2* **where** *2*: (*m2*∗*x2*) *mod m1* = *1 mod m1* **by** *auto*
have (*m1*∗*x1*) *mod m1* = *0* **by** *simp*
have (*m2*∗*x2*) *mod m2* = *0* **by** *simp*
show *?thesis* **using** *1 2*
by (*metis mod-0 mod-mult-self1-is-0*)
qed

lemma *cong-scalar2-nat*: *a mod m* = *b mod m* ⟹ (*k*∗*a*) *mod m* = (*k*∗*b*) *mod m*
for *a b k* :: *nat*
by (*rule mod-mult-cong*) *simp-all*

lemma *chinese-remainder-nat*:
fixes *m1 m2* :: *nat*
assumes *a*: *coprime m1 m2*
shows ∃ *x*. *x mod m1* = *u1 mod m1* ∧ *x mod m2* = *u2 mod m2*
proof −
from *chinese-remainder-aux-nat* [*OF a*] **obtain** *b1 b2* **where** *b1 mod m1* = *1 mod m1* **and** *b1 mod m2* = *0 mod m2* **and**
b2 mod m1 = *0 mod m1* **and** *b2 mod m2* = *1 mod m2* **by** *force*
let *?x* = *u1*∗*b1*+*u2*∗*b2*
have *?x mod m1* = (*u1*∗*1*+*u2*∗*0*) *mod m1*
apply (*rule mod-add-cong*)
apply(*rule cong-scalar2-nat*)
apply (*rule* ‹*b1 mod m1* = *1 mod m1*›)
apply(*rule cong-scalar2-nat*)
apply (*rule* ‹*b2 mod m1* = *0 mod m1*›)
done
hence *1*:*?x mod m1* = *u1 mod m1* **by** *simp*
have *?x mod m2* = (*u1*∗*0*+*u2*∗*1*) *mod m2*
apply (*rule mod-add-cong*)
apply(*rule cong-scalar2-nat*)
apply (*rule* ‹*b1 mod m2* = *0 mod m2*›)
apply(*rule cong-scalar2-nat*)
apply (*rule* ‹*b2 mod m2* = *1 mod m2*›)
done
hence *?x mod m2* = *u2 mod m2* **by** *simp*
with *1* **show** *?thesis* **by** *blast*
qed

lemma *nat-int1*: ∀ (*w*::*nat*) (*u*::*int*).*u*>*0* ⟹ (*w mod nat u* = *2 mod nat u* ⟹ *int w mod u* = *2 mod u*)
by *blast*

lemma *nat-int2*: ∀ (*w*::*nat*) (*b*::*nat*) (*v*::*int*).*u*>*0* ⟹ (*w mod nat v* = *b mod nat*

```
v ⟹ int w mod v = int b mod v)
  by (metis mod-by-0 nat-eq-iff zmod-int)

lemma lem:
  fixes u t::int and b::nat
  assumes u^2−int b∗u∗t+t^2=1 u≥0 t≥0
  shows (nat u)^2+(nat t)^2=1+b∗(nat u)∗(nat t)
proof −
  define U where U=nat u
  define T where T=nat t
   from U-def T-def assms have UT: int U=u ∧ int T = t using int-eq-iff by
blast
  from UT have UT1: int (b∗U∗T) = b∗u∗t by simp
  from UT have UT2: int (U^2+T^2)=u^2+t^2 by simp
  from UT2 assms have sth: int (U^2+T^2)≥b∗u∗t by auto
  from sth assms have sth1: U^2+T^2≥b∗U∗T using UT1 by linarith
  from sth1 of-nat-diff have sth2: int (U^2+T^2−b∗U∗T) = int (U^2+T^2) −
int (b∗U∗T) by blast
 from UT1 UT2 have UT3: int (U^2+T^2)−int (b∗U∗T)=u^2+t^2−b∗u∗t by
simp
  from sth2 UT3 assms have sth4: int (U^2+T^2−b∗U∗T) = 1
    by linarith
  from sth4 have sth5: U^2+T^2−b∗U∗T=1 by simp
  from sth5 have sth6: U^2+T^2=1+b∗U∗T by simp
  show ?thesis using sth6 U-def T-def by simp
qed
```

The necessity

```
lemma alpha-equiv-nec:
  b > 3 ∧ a = α b c ⟹ ∃ r s t u v w x y. alpha-equations a b c r s t u v w x y
proof −
  assume assms: b > 3 ∧ a = α b c
  have s1: ∃(k::nat) (u::int) (t::int).u=α b k ∧ odd u = True ∧ 2∗int a<u ∧ u<t
∧ u^2−(int b)∗u∗t+t^2=1 ∧ k>0 ∧ t = α b (k+1)
  proof −
    define j::nat where j=2∗(a)+1
    have rd: j>0 by (simp add: j-def)
    consider (c1) odd (α b j) = True | (c2) odd (α b (j+1)) = True
      using assms oneodd by fastforce
    thus ?thesis
    proof cases
      case c1
      define k::nat where k=j
      define u::int where u=α b k
      define t::int where t=α b (k+1)
      have stp: k>0 by (simp add: k-def j-def)
     from alpha-strictly-increasing assms have abc: u<t by (simp add: u-def t-def)
      have c11: odd u = True by (simp add: c1 k-def u-def)
    from alpha-det1 u-def t-def alpha-det2 assms(1) have bcd: u^2−(int b)∗u∗t+t^2=1
```

by (*metis* (*no-types*, *lifting*) *One-nat-def Suc-1 Suc-less-eq add-diff-cancel-right′ add-gr-0 less-Suc-eq mult.assoc numeral-3-eq-3*)
have *c12*: *int k>2∗a* **by** (*simp add*: *k-def j-def*)
from *alpha-superlinear c12* **have** *c13*: *2∗a<u*
by (*smt add-lessD1 assms*(*1*) *numeral-Bit1 numeral-One one-add-one u-def*)
from *c11 c13 k-def u-def t-def abc bcd stp* **show** *?thesis* **by** *auto*
next
case *c2*
define *k*::*nat* **where** *k=j+1*
define *u*::*int* **where** *u=α b k*
define *t*::*int* **where** *t=α b* (*k+1*)
have *stc*: *k>0* **by** (*simp add*: *k-def j-def*)
from *alpha-strictly-increasing assms* **have** *abc*: *u<t* **by** (*simp add*: *u-def t-def*)
from *c2 k-def u-def* **have** *c21*: *odd u = True* **by** *auto*
from *alpha-det1 u-def t-def alpha-det2 assms*(*1*) **have** *bcd*: *u^2−*(*int b*)*∗u∗t+t^2=1*
by (*metis* (*no-types*, *lifting*) *One-nat-def Suc-1 Suc-less-eq add-diff-cancel-right′ add-gr-0 less-Suc-eq mult.assoc numeral-3-eq-3*)
have *c22*: *int k>2∗a* **by** (*simp add*: *k-def j-def*)
from *alpha-superlinear c22* **have** *c23*: *2∗a<u*
by (*smt add-lessD1 assms*(*1*) *numeral-Bit1 numeral-One one-add-one u-def*)
from *c21 c23 abc bcd k-def u-def t-def* **show** *?thesis* **by** *auto*
qed
qed
then obtain *k u t* **where** *u=α b k ∧ odd u = True ∧ 2∗int a<u ∧ u<t ∧ u^2−*(*int b*)*∗u∗t+t^2=1 ∧ k>0 ∧ t= α b* (*k+1*) **by** *force*
define *m* **where** *m=*(*nat u*)*∗k*
define *s* **where** *s=α b m*
define *r* **where** *r=α b* (*m−1*)
note *udef* = ‹$u = \alpha\ b\ k \wedge odd\ u = True \wedge 2 * int\ a < u \wedge u < t \wedge u^2 - int\ b * u * t + t^2 = 1 \wedge 0 < k \wedge t = \alpha\ b\ (k{+}1)$›
from *assms* **have** *s211*: *int b > 3* **by** *simp*
from *assms alpha-superlinear* **have** *a354*: *c≤a*
by (*simp add*: *nat-int-comparison*(*3*))
from *a354 udef* **have** *354*: *2∗ int c<u* **by** *simp*
from *alpha-superlinear s211 m-def udef* **have** *rd*: *α b k ≥ int k* **by** *simp*
from *alpha-strictly-increasing s211 s1 m-def s-def udef r-def* **have** *s212*: *α b* (*m−1*) *< α b m*
by (*smt One-nat-def Suc-pred nat-0-less-mult-iff zero-less-nat-eq*)
from *s212 r-def s-def* **have** *344*: *r<s* **by** *simp*
from *alpha-det2 assms s-def r-def m-def* **have** *s22*: *r^2−int b∗r∗s+s^2=1* **by** (*smt One-nat-def Suc-eq-plus1 udef add-lessD1 alpha-superlinear mult.assoc nat-0-less-mult-iff numeral-3-eq-3 of-nat-0 of-nat-less-iff one-add-one zero-less-nat-eq*)
from *s22* **have** *343*: *s^2−int b∗s∗r+r^2=1* **by** *algebra*
from *m-def udef* **have** *xyz*: (*int k*)*∗*(*α b k*) *dvd* (*int m*) *∧ k dvd m* **by** *simp*
from *xyz divisibility-alpha2* **have** *wxyz*: (*α b k*)*∗*(*α b k*) *dvd* (*α b m*) **by** (*smt assms dvd-mult-div-cancel int-nat-eq less-imp-le-nat m-def mult-pos-pos neq0-conv not-less not-less-eq numeral-2-eq-2 numeral-3-eq-3 of-nat-0-less-iff power2-eq-square udef*)

from *wxyz udef s-def* **have** *345*: *u^2 dvd s* **by** (*simp add*: *power2-eq-square*)
define *v* **where** *v* = *b∗s−2∗r*
from *v-def s-def r-def alpha-n* **have** *370*: *v* = *α b* (*m+1*) − *α b* (*m−1*)
by (*smt Suc-eq-plus1 add-diff-inverse-nat diff-Suc-1 neq0-conv not-less-eq s212 zero-less-diff*)
have *371*: *v* = *b∗α b m* − *2∗α b* (*m−1*) **using** *v-def s-def r-def* **by** *simp*
from *alpha-strictly-increasing assms m-def udef* **have** *asd*: *α b m* > *0*
by (*smt Suc-pred nat-0-less-mult-iff s211 zero-less-nat-eq*)
from *assms asd 371* **have** *372*: *v≥4∗α b m −2∗α b* (*m−1*) **by** *simp*
from *372 assms* **have** *373*: *v>2∗α b m* ∧ *4∗α b m −2∗α b* (*m−1*) > *2∗α b m* **using** *s212* **by** *linarith*
from *373 assms alpha-superlinear* **have** *374*: *2∗α b m* ≥ *2∗m* ∧ *v>2∗m*
by (*smt One-nat-def Suc-eq-plus1 add-lessD1 distrib-right mult.left-neutral numeral-3-eq-3 of-nat-add one-add-one*)
from *udef* **have** *pre1*: *k≥1* ∧ *u≥1* **using** *rd* **by** *linarith*
from *pre1 374 m-def* **have** *pre2*: *m≥u* **by** *simp*
from *pre2 374* **have** *375*: *2∗m≥2∗u* ∧ *v>2∗u* **by** *simp*
from *375 udef* **have** *376*: *2∗u>2∗a* ∧ *v>2∗a* **using** *pre1* **by** *linarith*
have *u-v-coprime*: *coprime u v*
proof −
obtain *d*::*nat* **where** *d*: *d* = *gcd u v*
by (*metis gcd-int-def*)
from ‹*d* = *gcd u v*› **have** *ddef*: *d dvd u* ∧ *d dvd v* **by** *simp*
from *345 ddef* **have** *stp1*: *d dvd s* **using** *dvd-mult-left dvd-trans s-def udef wxyz* **by** *blast*
from *v-def stp1 ddef* **have** *stp2*: *d dvd 2∗r* **by** *algebra*
from *ddef udef* **have** *d-odd*: *odd d* **using** *dvd-trans* **by** *auto*
have *r2even*: *even* (*2∗r*) **by** *simp*
from *stp2 d-odd r2even* **have** *stp3*: (*2∗d*) *dvd* (*2∗r*) **by** *fastforce*
from *stp3* **have** *stp4*: *d dvd r* **by** *simp*
from *stp1 stp4* **have** *stp5*: *d dvd s^2* ∧ *d dvd* (*−int b∗s∗r*) ∧ *d dvd r^2* **by** (*simp add*: *power2-eq-square*)
from *stp5* **have** *stp6*: *d dvd* (*s^2−int b∗s∗r+r^2*) **by** *simp*
from *343 stp6* **have** *stp7*: *d dvd 1* **by** *simp*
show *?thesis* **using** *stp7 d* **by** *auto*
qed
have *wdef*: ∃ *w*::*nat*. *int w mod u* = *2 mod u* ∧ *int w mod v* = *int b mod v* ∧ *w>2*
proof −
from *pre1 m-def* **have** *mg*: *m≥1* **by** *auto*
from *s-def r-def 344* **have** *srg*: *s−r≥1* **by** *simp*
from *assms* **have** *bg*: *b≥4* **by** *simp*
from *bg* **have** *bsr*: ((*int b*)*∗s−2∗r*)≥(*4∗s−2∗r*) **using** *372 r-def s-def v-def* **by** *blast*
have *t1*: *v≥2+2∗s* **using** *bsr srg v-def* **by** *simp*
from *s-def* **have** *sg*: *s≥1* **using** *asd* **by** *linarith*
from *sg t1* **have** *t2*: *v≥4* **by** *simp*
from *u-v-coprime* **have** *u-v-coprime1*:*coprime* (*nat u*) (*nat v*) **using** *pre1 t2*
using *coprime-int-iff* **by** *fastforce*

obtain *z::nat* **where** *z mod nat u = 2 mod nat u ∧ z mod nat v = b mod nat v* **using** *chinese-remainder-nat u-v-coprime1* **by** *force*
note *zdef = ‹z mod nat u = 2 mod nat u ∧ z mod nat v = b mod nat v›*
from *t2 pre1* **have** *t31*: *nat v≥4 ∧ nat u≥1* **by** *auto*
from *t31* **have** *t3*: *nat v∗nat u≥4* **using** *mult-le-mono* **by** *fastforce*
define *w::nat* **where** *w=z+ nat u∗nat v*
from *w-def t3* **have** *t4*: *w≥4* **by** (*simp add*: *mult.commute*)
have *t51*: (*nat u∗nat v*) *mod nat u = 0 ∧* (*nat u∗nat v*) *mod nat v = 0* **using** *algebra* **by** *simp*
from *t51 w-def* **have** *t5*: *w mod nat u = z mod nat u* **by** *presburger*
from *t51 w-def* **have** *t6*: *w mod nat v = z mod nat v* **by** *presburger*
from *t5 t6 zdef* **have** *t7*: *w mod nat u = 2 mod nat u ∧ w mod nat v = b mod nat v* **by** *simp*
from *t7 t31* **have** *t8*: *int w mod u = 2 mod u ∧ int w mod v = int b mod v*
using *nat-int1 nat-int2* **by** *force*
from *t4 t8* **show** *?thesis* **by** *force*
qed
obtain *w::nat* **where** *int w mod u = 2 mod u ∧ int w mod v = int b mod v ∧ w>2* **using** *wdef* **by** *force*
note *wd = ‹int w mod u = 2 mod u ∧ int w mod v = int b mod v ∧ w>2›*
define *x* **where** *x = α w c*
define *y* **where** *y = α w* (*c+1*)
from *alpha-det1 wd x-def y-def* **have** *350*: *x^2−int w∗x∗y+y^2 =1* **by** (*metis add-gr-0 alpha-det2 diff-add-inverse2 less-one mult.assoc*)
from *x-def wd congruence* **have** *353*: *a mod v = x mod v*
by (*smt 374 assms int-nat-eq nat-int nat-mod-distrib*)
from *congruence2 wd x-def* **have** *379*: *x mod int* (*w−2*) *= int c mod* (*int w−2*)
using *int-ops*(*6*) *zmod-int* **by** *auto*
from *wd* **have** *wc*: *u dvd* (*int w−2*) **using** *mod-diff-cong mod-eq-0-iff-dvd* **by** *fastforce*
from *wc* **have** *wb*: *∃ k1. u∗k1=int w−2* **by** (*metis dvd-def*)
obtain *k1* **where** *u∗k1=int w−2* **using** *wb* **by** *force*
note *k1def = ‹u∗k1=int w−2›*
define *r1* **where** *r1=int c mod* (*int w−2*)
from *r1-def 379* **have** *wa*: *r1 = x mod* (*int w−2*)
using *int-ops*(*6*) *wd* **by** *auto*
obtain *k2* **where** *int c =* (*int w−2*)*∗k2+r1* **by** (*metis mult-div-mod-eq r1-def*)
note *k2def = ‹int c =* (*int w−2*)*∗k2+r1›*
from *k2def k1def* **have** *a355*: *int c = u∗k1∗k2+r1* **by** *simp*
from *udef k1def k2def* **have** *bh*: *u∗k1∗k2 mod u = 0* **by** (*metis mod-mod-trivial mod-mult-left-eq mod-mult-self1-is-0 mult-eq-0-iff*)
from *a355 bh* **have** *b355*: (*u∗k1∗k2+r1*) *mod u = r1 mod u* **by** (*simp add*: *mod-eq-dvd-iff*)
from *a355 b355* **have** *c355*: *int c mod u = r1 mod u* **by** *simp*
from *wa* **have** *waa*: *∃ k3. x = k3∗*(*int w−2*)*+r1* **by** (*metis div-mult-mod-eq*)
obtain *k3* **where** *x=k3∗*(*int w−2*)*+r1* **using** *waa* **by** *force*
from *k1def ‹x = k3∗*(*int w−2*)*+r1›* **have** *d355*: *x=u∗k1∗k3+r1* **by** *simp*
from *udef k1def* **have** *ch*: *u∗k1∗k3 mod u = 0* **by** (*metis mod-mod-trivial mod-mult-left-eq mod-mult-self1-is-0 mult-eq-0-iff*)

from *d355 ch* **have** *e355*: (*u*∗*k1*∗*k3*+*r1*) *mod u* = *r1 mod u* **by** (*simp add*: *mod-eq-dvd-iff*)
from *d355 e355* **have** *f355*: *x mod u* = *r1 mod u* **by** *simp*
from *c355 f355* **have** *355*: *int c mod u* = *x mod u* **by** *simp*
from *assms s1 wdef udef 343 344 345 v-def wd 350 376 353 354 355* **have** *prefinal*: *u^2*−*b*∗*u*∗*t*+*t^2*=*1* ∧ *s^2*−*b*∗*s*∗*r*+*r^2*=*1* ∧ *r*<*s*
∧ *u^2 dvd s* ∧ *b*∗*s*=*v*+*2*∗*r* ∧ *w mod v* = *b mod v* ∧ *w mod u* = *2 mod u* ∧ *w*>*2* ∧ *x^2*−*w*∗*x*∗*y*+*y^2*=*1* ∧
2∗*a*<*u* ∧ *2*∗*a*<*v* ∧ *a mod v* = *x mod v* ∧ *2*∗*c*<*u* ∧ *c mod u* = *x mod u* **by** *fastforce*
from *alpha-strictly-increasing* **have** *s-pos*: *s*≥*0* **using** *asd s-def* **by** *linarith*
define *S* **where** *S*=*nat s*
from *alpha-strictly-increasing* **have** *r-pos*: *r*≥*0* **using** *asd r-def* **by** (*smt One-nat-def Suc-1 alpha-superlinear assms*(*1*) *lessI less-trans numeral-3-eq-3 of-nat-0-le-iff*)
define *R* **where** *R*=*nat r*
from *udef alpha-strictly-increasing* **have** *ut-pos*:*u*≥*0* ∧ *t*≥*0* **using** *pre1* **by** *linarith*
from *assms* **have** *a-pos*: *a*≥*0* **using** *a354* **by** *linarith*
from *a-pos* **have** *v-pos*: *v*≥*0* **using** *376* **by** *linarith*
from *x-def y-def* **have** *xy-pos*: *x*≥*0* ∧ *y*≥*0* **by** (*smt alpha-superlinear of-nat-0-le-iff wd*)
define *U* **where** *U*=*nat u*
define *T* **where** *T*=*nat t*
define *V* **where** *V*=*nat v*
define *X* **where** *X*=*nat x*
define *Y* **where** *Y*=*nat y*
from *lem U-def T-def S-def R-def X-def Y-def prefinal* **have** *lem1*: *U^2*+*T^2*=*1*+*b*∗*U*∗*T* ∧ *S^2*+*R^2*=*1*+*b*∗*S*∗*R* ∧ *X^2*+*Y^2*=*1*+*w*∗*X*∗*Y* **using** *s-pos ut-pos r-pos xy-pos* **by** *blast*
from *R-def S-def* **have** *lem2*: *R*<*S* **using** *r-def s-def r-pos s-pos* **using** *s212* **by** *linarith*
from *U-def S-def* **have** *lem3*: *U^2 dvd S* **using** *345 ut-pos s-pos*
by (*metis int-dvd-int-iff int-nat-eq of-nat-power*)
have *aq*: *int b*∗*s*≥*2*∗*r* **using** *v-def v-pos* **by** *simp*
from *aq* **have** *aq1*: *nat* (*int b*∗*s*) ≥ *nat* (*2*∗*r*) **by** *simp*
from *s-pos r-pos assms* **have** *aq2*: *nat* (*int b*∗*s*) = *b*∗(*nat s*) ∧ *nat* (*2*∗*r*) = *2*∗(*nat r*) **by** (*simp add*: *nat-mult-distrib*)
from *aq1 aq2* **have** *aq3*: *b*∗*S*≥*2*∗*R* **using** *S-def R-def* **by** *simp*
from *aq3* **have** *aq4*: *int* (*b*∗*S*−*2*∗*R*) = *int* (*b*∗*S*)−*int* (*2*∗*R*) **using** *of-nat-diff* **by** *blast*
have *aq5*: *int* (*b*∗*S*) = *int b*∗*int S* ∧ *int* (*2*∗*R*) = *2*∗*int R* **by** *simp*
from *aq4 aq5* **have** *aq6*: *int* (*b*∗*S*−*2*∗*R*) = *int b*∗*s*−*2*∗*r* **using** *R-def S-def r-pos s-pos* **by** *simp*
from *aq6 v-def v-pos V-def* **have** *lem4*: *b*∗*S*−*2*∗*R* = *V* **by** *simp*
from *prefinal v-pos V-def ut-pos U-def xy-pos X-def a-pos* **have** *lem5*: *w mod V* = *b mod V* ∧ *w mod U* = *2 mod U* ∧ *a mod V*=*X mod V* ∧ *c mod U* = *X mod U*
by (*metis int-nat-eq nat-int of-nat-numeral zmod-int*)
from *a-pos ut-pos v-pos U-def V-def prefinal* **have** *lem6*: *2*∗*nat a*<*U* ∧ *2*∗*nat a*<*V* ∧ *2*∗*c*<*U* **by** *auto*

from *prefinal* **have** *lem7*: *w>2* **by** *simp*
have *third-last*: ∀ *b s v r*::*nat*. $b * s = v + 2 * r \longleftrightarrow$ *int* $(b * s) =$ *int* $(v + 2 * r)$ **using** *of-nat-eq-iff* **by** *blast*
have *onemore*: ∀ *u t b*. $u \hat{~} 2 + t \hat{~} 2 = 1 + b * u * t \longleftrightarrow$ *int u* $\hat{~} 2 +$ *int t* $\hat{~} 2 = 1 +$ *int b* $*$ *int u* $*$ *int t*
by (*metis* (*no-types*) *nat-int of-nat-1 of-nat-add of-nat-mult of-nat-power*)
from *lem1 lem2 lem3 lem4 lem5 lem6 lem7 third-last onemore* **show** *?thesis*
unfolding *Exp-Matrices.alpha-equations-def*[*of a b c*] **apply** *auto*
using *assms* **apply** *blast*
apply (*rule exI*[*of - R*], *rule exI*[*of - S*], *rule exI*[*of - T*], *rule exI*[*of - U*], *simp*)
apply (*rule exI*[*of - V*], *simp*)
apply (*rule exI*[*of - w*], *simp*)
apply (*rule exI*[*of - X*], *simp*)
using *aq4 aq5 lem5* **by** *auto*
qed

2.1.10 Exponentiation is Diophantine

Equations 3.80-3.83

lemma *86*:
fixes *b r* **and** *q*::*int*
defines $m \equiv b * q - q * q - 1$
shows $(q * \alpha\ b\ (r + 1) - \alpha\ b\ r)$ *mod* $m = (q \hat{~} (r + 1))$ *mod* m
proof(*induction r*)
case *0*
show *?case* **by** *simp*
next
case (*Suc n*)
from *m-def* **have** *a0*: $(q * q - b * q + 1)$ *mod* $m =$ $((-(q * q - b * q + 1))$ *mod* $m + (q * q - b * q + 1)$ *mod* $m)$ *mod* m **by** *simp*
have *a1*: ... = *0* **by** (*simp add*:*mod-add-eq*)
from *a0 a1* **have** *a2*: $(q * q - b * q + 1)$ *mod* $m = 0$ **by** *simp*

from *a2* **have** *b0*: $(b * q - 1)$ *mod* $m = ((q * q - b * q + 1)$ *mod* $m + (b * q - 1)$ *mod* $m)$ *mod* m **by** *simp*
have *b1*: ... $= (q * q)$ *mod* m **by** (*simp add*: *mod-add-eq*)
from *b0 b1* **have** *b2*: $(b * q - 1)$ *mod* $m =$ $(q * q)$ *mod* m **by** *simp*

have $(q * (\alpha\ b\ (Suc\ n + 1)) - \alpha\ b\ (Suc\ n))$ *mod* $m = (q * ($*int* $b * \alpha\ b\ (Suc\ n) - \alpha\ b\ n) - \alpha\ b\ (Suc\ n))$ *mod* m **by** *simp*
also have ... $= ((b * q - 1) * \alpha\ b\ (Suc\ n) - q * \alpha\ b\ n)$ *mod* m **by** *algebra*
also have ... $= (((b * q - 1) * \alpha\ b\ (Suc\ n))$ *mod* $m - (q * \alpha\ b\ n)$ *mod* $m)$ *mod* m **by** (*simp add*: *mod-diff-eq*)
also have ... $= ((((b * q - 1)$ *mod* $m) * ((\alpha\ b\ (Suc\ n))$ *mod* $m))$ *mod* $m - (q * \alpha\ b\ n)$ *mod* $m)$ *mod* m **by** (*simp add*: *mod-mult-eq*)
also have ... $= ((((q * q)$ *mod* $m) * ((\alpha\ b\ (Suc\ n))$ *mod* $m))$ *mod* $m - (q * \alpha\ b\ n)$ *mod* $m)$ *mod* m **by** (*simp add*: *b2*)
also have ... $= (((q * q) * (\alpha\ b\ (Suc\ n)))$ *mod* $m - (q * \alpha\ b\ n)$ *mod* $m)$ *mod* m **by** (*simp add*: *mod-mult-eq*)

also have ... = $((q * q) * (\alpha\ b\ (Suc\ n)) - q * \alpha\ b\ n)$ *mod m* **by** (*simp add*: *mod-diff-eq*)
also have ... = $(q * (q * (\alpha\ b\ (Suc\ n)) - \alpha\ b\ n))$ *mod m* **by** *algebra*
also have ... = $((q\ mod\ m) * ((q * (\alpha\ b\ (Suc\ n)) - \alpha\ b\ n)\ mod\ m))$ *mod m* **by** (*simp add*: *mod-mult-eq*)
finally have *c0*: $(q * (\alpha\ b\ (Suc\ n + 1)) - \alpha\ b\ (Suc\ n))\ mod\ m = ((q\ mod\ m) * ((q * (\alpha\ b\ (Suc\ n)) - \alpha\ b\ n)\ mod\ m))\ mod\ m$ **by** *simp*
from *Suc.IH* **have** *c1*: ... = $((q\ \hat{}\ (n + 2)))$ *mod m* **by** (*simp add*: *mod-mult-eq*)

from *c0 c1* **show** *?case* **by** *simp*
qed

This is a more convenient version of (86)

lemma *860*:
fixes *b r* **and** *q*::*int*
defines $m \equiv b * q - q * q - 1$
shows $(q * \alpha\ b\ r - (int\ b * \alpha\ b\ r - \alpha\ b\ (Suc\ r)))\ mod\ m = (q\ \hat{}\ r)\ mod\ m$
proof(*cases r=0*)
case *True*
then show *?thesis* **by** *simp*
next
case *False*
thus *?thesis* **using** *alpha-n*[*of b r−1*] *86*[*of q b r−1*] *m-def* **by** *auto*
qed

We modify the equivalence (88) in a similar manner

lemma *88*:
fixes *b r p q*:: *nat*
defines $m \equiv int\ b * int\ q - int\ q * int\ q - 1$
assumes $int\ q\ \hat{}\ r < m$ **and** $q > 0$
shows $int\ p = int\ q\ \hat{}\ r \longleftrightarrow int\ p < m \land (q * \alpha\ b\ r - (int\ b * \alpha\ b\ r - \alpha\ b\ (Suc\ r)))\ mod\ m = int\ p\ mod\ m$
using *Exp-Matrices.860 assms*(*2*) *m-def* **by** *auto*

lemma *89*:
fixes *r p q* :: *nat*
assumes $q > 0$
defines $b \equiv nat\ (\alpha\ (q + 4)\ (r + 1)) + q * q + 2$
defines $m \equiv int\ b * int\ q - int\ q * int\ q - 1$
shows $int\ q\ \hat{}\ r < m$
proof −
have *a0*: $int\ q * int\ q - 2 * int\ q + 1 = (int\ q - 1) * (int\ q - 1)$ **by** *algebra*
from *assms* **have** *a1*: $int\ q * int\ q * int\ q \geq int\ q * int\ q$ **by** *simp*
from *assms a0 a1* **have** *a2*: ... $> (int\ q - 1) * (int\ q - 1)$ **by** *linarith*

from *alpha-strictly-increasing* **have** *c0*: $\alpha\ (q + 4)\ (r + 1) > 0$ **by** *simp*
from *c0* **have** *c1*: $\alpha\ (q + 4)\ (r + 1) = int\ (nat\ (\alpha\ (q + 4)\ (r + 1)))$ **by** *simp*

then have *b1*: $(q+3)\ \hat{}\ r \leq \alpha\ (q + 4)\ (r + 1)$ **using** *alpha-exponential-1*[*of*

q+3]
  by(*auto simp add*: *add.commute*)
 have *b3*: *int q* ^ *r* ≤ (*q* + *3*) ^ *r* **by** (*simp add*: *power-mono*)
 also have *b4*: ... ≤ (*q* + *3*) ^ *r* ∗ *int q* **using** *assms* **by** *simp*
 also from *assms b1* **have** *b5*: ... ≤ α (*q* + *4*) (*r* + *1*) ∗ *int q* **by** *simp*
 also from *a2* **have** *b6*: ... < α (*q* + *4*) (*r* + *1*) ∗ *int q* + *int q* ∗ *int q* ∗ *int q* − (*int q* − *1*) ∗ (*int q* − *1*) **by** *simp*
 also have *b7*: ... = (α (*q* + *4*) (*r* + *1*) + *int q* ∗ *int q* + *2*) ∗ *q* − *int q* ∗ *int q* − *1* **by** *algebra*
 also from *assms m-def* **have** *b8*: ... = *m* **using** *c1* **by** *auto*
 finally show *?thesis* **by** *linarith*
qed
end

The final equivalence

theorem *exp-alpha*:
 fixes *p q r* :: *nat*
 shows *p* = *q* ^ *r* ⟷ ((*q* = *0* ∧ *r* = *0* ∧ *p* = *1*) ∨
   (*q* = *0* ∧ *0* < *r* ∧ *p* = *0*) ∨
   (*q* > *0* ∧ (∃ *b m*.
    b = *Exp-Matrices.α* (*q* + *4*) (*r* + *1*) + *q* ∗ *q* + *2* ∧
    m = *b* ∗ *q* − *q* ∗ *q* − *1* ∧
    p < *m* ∧
   p mod m = ((*q* ∗ *Exp-Matrices.α b r*) − (*int b* ∗ *Exp-Matrices.α b r* − *Exp-Matrices.α b* (*r* + *1*))) *mod m*)))
proof(*cases q>0*)
 case *True*
 show *?thesis* (**is** *?P* = *?Q*)
 proof (*rule*)
  assume *?P*
  define *b* **where** *b* = *nat* (*Exp-Matrices.α* (*q* + *4*) (*r* + *1*)) + *q* ∗ *q* + *2*
  define *m* **where** *m* = *int b* ∗ *int q* − *int q* ∗ *int q* − *1*
  have *sg1*: *int b* = *Exp-Matrices.α* (*q* + *4*) (*Suc r*) + *int q* ∗ *int q* + *2* **using** *b-def*
  proof−
  have *0* ≤ (*Exp-Matrices.α* (*q* + *4*) (*r* + *1*)) **using** *Exp-Matrices.alpha-exponential-1*[*of q+3 r*]
    apply (*simp add*: *add.commute*) **using** *zero-le-power*[*of int q+3 r*] **by** *linarith*
   then show *?thesis* **using** *b-def int-nat-eq*[*of* (*Exp-Matrices.α* (*q* + *4*) (*r* + *1*))] **by** *simp*
  qed
  have *sg2*: *q* ^ *r* < *b* ∗ *q* − *Suc* (*q* ∗ *q*) **using** *True Exp-Matrices.89*[*of q r*]
   of-nat-less-of-nat-power-cancel-iff[*of q r b* ∗ *q* − *Suc* (*q* ∗ *q*)]
    b-def int-ops(*6*)[*of b* ∗ *q Suc* (*q* ∗ *q*)] *of-nat-1 of-nat-add of-nat-mult plus-1-eq-Suc* **by** *smt*
  have *sg3*: *int* (*q* ^ *r mod* (*b* ∗ *q* − *Suc* (*q* ∗ *q*)))
     = (*int q* ∗ *Exp-Matrices.α b r* − (*int b* ∗ *Exp-Matrices.α b r* − *Exp-Matrices.α b* (*Suc r*)))

mod int (*b* ∗ *q* − *Suc* (*q* ∗ *q*))
proof−
have *int b* ∗ *int q* − *int q* ∗ *int q* − *1* = *b* ∗ *q* − *Suc* (*q* ∗ *q*)
using ‹*q* ^ *r* < *b* ∗ *q* − *Suc* (*q* ∗ *q*)› *int-ops*(*6*) **by** *auto*
then show *?thesis* **using** *Exp-Matrices.860*[*of q b r*] **by** (*simp add*: *zmod-int*)
qed
from *sg1 sg2 sg3 True* **show** *?Q*
by (*smt* (*verit*) *Suc-eq-plus1-left* ‹*p* = *q* ^ *r*› *add.commute diff-diff-eq of-nat-mult*)
next
assume *Q*: *?Q* (**is** *?A* ∨ *?B* ∨ *?C*)
thus *?P*
proof (*elim disjE*)
show *?A* ⟹ *?P* **by** *auto*
show *?B* ⟹ *?P* **by** *auto*
show *?C* ⟹ *?P*
proof−
obtain *b* **where** *b-def*: *int b* = *Exp-Matrices.α* (*q* + *4*) (*Suc r*) + *int q* ∗ *int q* + *2* **using** *Q True* **by** *auto*
have *prems3*: *p* < *b* ∗ *q* − *Suc* (*q* ∗ *q*) **using** *Q True b-def* **apply** (*simp add*: *add.commute*) **by** (*metis of-nat-eq-iff*)
have *prems4*: *int p* = (*int q* ∗ *Exp-Matrices.α b r* − ((*Exp-Matrices.α* (*q* + *4*) (*Suc r*) +
int q ∗ *int q* + *2*) ∗ *Exp-Matrices.α b r* − *Exp-Matrices.α b* (*Suc r*))) *mod int* (*b* ∗ *q* − *Suc* (*q* ∗ *q*))
using *Q True b-def* **apply** (*simp add*: *add.commute*) **by** (*metis mod-less of-nat-eq-iff*)
define *m* **where** *m* = *int b* ∗ *int q* − *int q* ∗ *int q* − *1*
have *int q* ^ *r* < *int b* ∗ *int q* − *int q* ∗ *int q* − *1* **using** *Exp-Matrices.89*[*of q r*] *b-def True*
by (*smt Exp-Matrices.alpha-strictly-increasing One-nat-def Suc-eq-plus1 int-nat-eq nat-2*
numeral-Bit0 of-nat-0-less-iff of-nat-add of-nat-mult one-add-one)
moreover have *int p* < *m* **by** (*smt gr-implies-not0 int-ops*(*6*) *int-ops*(*7*) *less-imp-of-nat-less*
m-def of-nat-Suc of-nat-eq-0-iff prems3)
moreover have (*int q* ∗ *Exp-Matrices.α b r* − (*int b* ∗ *Exp-Matrices.α b r* − *Exp-Matrices.α b* (*Suc r*))) *mod m* = *int p mod m*
using *prems4* **by** (*smt calculation*(*2*) *int-ops*(*6*) *m-def mod-pos-pos-trivial of-nat-0-le-iff*
of-nat-1 of-nat-add of-nat-mult plus-1-eq-Suc b-def)
ultimately show *?thesis* **using** *True Exp-Matrices.88*[*of q r b p*] *m-def* **by** *simp*
qed
qed
qed
next
case *False*
then show *?thesis* **by** *auto*
qed

lemma *alpha-equivalence*:
fixes *a b c*
shows $3 < b \wedge int\ a = Exp\text{-}Matrices.\alpha\ b\ c \longleftrightarrow (\exists\, r\ s\ t\ u\ v\ w\ x\ y.\ Exp\text{-}Matrices.alpha\text{-}equations\ a\ b\ c\ r\ s\ t\ u\ v\ w\ x\ y)$
using *Exp-Matrices.alpha-equiv-suff Exp-Matrices.alpha-equiv-nec*
by *meson+*

end

2.2 Diophantine description of alpha function

theory *Alpha-Sequence*
imports *Modulo-Divisibility Exponentiation*
begin

The alpha function is diophantine

definition *alpha* (‹[- = α - -]› *1000*)
where $[X = \alpha\ B\ N] \equiv (TERNARY\ (\lambda b\ n\ x.\ b > 3 \wedge x = Exp\text{-}Matrices.\alpha\ b\ n)\ B\ N\ X)$

lemma *alpha-dioph*[*dioph*]:
fixes *B N X*
defines $D \equiv [X = \alpha\ B\ N]$
shows *is-dioph-rel D*
proof −
define *r s t u v w x y* **where** *param-defs*:
$r == (Param\ 0)\ s == (Param\ 1)\ t == (Param\ 2)\ u == (Param\ 3)\ \ v == (Param\ 4)$
$w == (Param\ 5)\ x == (Param\ 6)\ y == (Param\ 7)$
define $B'\ X'\ N'$ **where** *pushed-defs*: $B' == (push\text{-}param\ B\ 8)\ X' == (push\text{-}param\ X\ 8)$
$N' == (push\text{-}param\ N\ 8)$

define *DR1* **where** $DR1 \equiv B'\ [>]\ (Const\ 3)\ [\wedge]\ (Const\ 1\ [+]\ B'\ [*]\ u\ [*]\ t\ [=]\ u[\hat{}2]\ [+]\ t[\hat{}2])$
define *DR2* **where** $DR2 \equiv (Const\ 1\ [+]\ B'\ [*]\ s\ [*]\ r\ [=]\ s[\hat{}2]\ [+]\ r[\hat{}2])\ [\wedge]\ r\ [<]\ s$
define *DR3* **where** $DR3 \equiv (DVD\ (u[\hat{}2])\ s)\ [\wedge]\ (v\ [+]\ (Const\ 2)\ [*]\ r\ [=]\ B'\ [*]\ s)$
define *DR4* **where** $DR4 \equiv (MOD\ B'\ v\ w)\ [\wedge]\ (MOD\ (Const\ 2)\ u\ w)\ [\wedge]\ (Const\ 2)\ [<]\ w$
define *DR5* **where** $DR5 \equiv (Const\ 1\ [+]\ w\ [*]\ x\ [*]\ y\ [=]\ x[\hat{}2]\ [+]\ y[\hat{}2])$
define *DR6* **where** $DR6 \equiv (Const\ 2)\ [*]\ X'\ [<]\ u\ [\wedge]\ (Const\ 2)\ [*]\ X'\ [<]\ v$
$[\wedge]\ (MOD\ x\ v\ X')\ [\wedge]\ (Const\ 2)\ [*]\ N'\ [<]\ u\ [\wedge]\ MOD\ x\ u\ N'$

define *DR* **where** $DR \equiv [\exists\, 8]\ DR1\ [\wedge]\ DR2\ [\wedge]\ DR3\ [\wedge]\ DR4\ [\wedge]\ DR5\ [\wedge]\ DR6$

```
  note DR-defs = DR1-def DR2-def DR3-def DR4-def DR5-def DR6-def

  have is-dioph-rel DR
    unfolding DR-def DR-defs
    by (auto simp: dioph)

  moreover have eval D a = eval DR a for a
  proof -
    define x-ev b n where evaled-defs: x-ev ≡ peval X a b ≡ peval B a n ≡ peval
N a
    have h: eval D a = (∃ r s t u v w x y::nat. Exp-Matrices.alpha-equations x-ev b
n r s t u v w x y)
      unfolding D-def alpha-def evaled-defs defs using alpha-equivalence by simp

    show ?thesis
    proof (rule)
      assume eval D a
      then obtain r s t u v w x y :: nat where Exp-Matrices.alpha-equations x-ev
b n r s t u v w x y
        using h by auto
      then show eval DR a
        unfolding evaled-defs Exp-Matrices.alpha-equations-def
        unfolding DR-def DR-defs defs param-defs apply (auto simp: sq-p-eval)
        apply (rule exI[of - [r, s, t, u, v, w, x, y]])
          unfolding pushed-defs by (auto simp add: push-push[where ?n = 8]
push-list-eval)
    next
      assume eval DR a
      then show eval D a

        unfolding DR-def DR-defs defs param-defs apply (auto simp: sq-p-eval)
         unfolding pushed-defs apply (auto simp add: push-push[where ?n = 8]
push-list-eval)

        unfolding h Exp-Matrices.alpha-equations-def evaled-defs
        subgoal for ks
          apply (rule exI[of - ks!0]) apply (rule exI[of - ks!1]) apply (rule exI[of
- ks!2])
          apply (rule exI[of - ks!3]) apply (rule exI[of - ks!4]) apply (rule exI[of
- ks!5])
         apply (rule exI[of - ks!6]) apply (rule exI[of - ks!7])
         by simp-all
        done
    qed
  qed

  ultimately show ?thesis
```

by (*auto simp*: *is-dioph-rel-def*)
qed

declare *alpha-def*[*defs*]

end

2.3 Exponentiation is a Diophantine Relation

theory *Exponential-Relation*
imports *Alpha-Sequence Exponentiation*
begin

definition *exp-equations* :: *nat* ⇒ *nat* ⇒ *nat* ⇒ *nat* ⇒ *nat* ⇒ *bool* **where**
exp-equations p q r b m = (*b* = *Exp-Matrices.α* (*q* + *4*) (*r* + *1*) + *q* ∗ *q* + *2* ∧
m + *q*^*2* + *1* = *b* ∗ *q* ∧
p < *m* ∧
(*p* + *b* ∗ *Exp-Matrices.α b r*) *mod m* = (*q* ∗ *Exp-Matrices.α b r* +
Exp-Matrices.α b (*r* + *1*)) *mod m*)

lemma *exp-repr*:
fixes *p q r* :: *nat*
shows *p* = *q*^*r* ⟷ ((*q* = *0* ∧ *r* = *0* ∧ *p* = *1*) ∨
(*q* = *0* ∧ *0* < *r* ∧ *p* = *0*) ∨
(*q* > *0* ∧ (∃ *b m* :: *nat*. *exp-equations p q r b m*))) (**is** *?P* ⟷ *?Q*)
proof
assume *P*: *?P*

consider (*c1*)*q* = *0* ∧ *r* = *0* ∧ *p* = *1* | (*c2*) *q* = *0* ∧ *0* < *r* ∧ *p* = *0* | (*c3*) *q* > *0* ∧
(∃ *b m*. *b* = *Exp-Matrices.α* (*q* + *4*) (*r* + *1*) + *q* ∗ *q* + *2* ∧ *m* = *b* ∗ *q* − *q* ∗ *q* − *1* ∧
p < *m* ∧ *p mod m* = (*q* ∗ *Exp-Matrices.α b r* − (*b* ∗ *Exp-Matrices.α b r* −
Exp-Matrices.α b (*r* + *1*))) *mod m*) **using** *exp-alpha*[*of p q r*] *P* **by** *auto*
then show *?Q* **using** *P*
proof *cases*
case *c1*
then show *?thesis* **by** *auto*
next
case *c2*
then show *?thesis* **by** *auto*
next
case *c3*
obtain *b m* **where**

```
      b-def: b = Exp-Matrices.α (q + 4) (r + 1) + q ∗ q + 2 and
      m = b ∗ q − q ∗ q − 1 and
      p < m and
      int (p mod m) = (int q ∗ Exp-Matrices.α b r − (int b ∗ Exp-Matrices.α b r
−
      Exp-Matrices.α b (r + 1))) mod int m
      using exp-alpha[of p q r] c3 by blast
    then have exp-equations p q r b m unfolding exp-equations-def
     apply(intro conjI, auto simp add: power2-eq-square) using mod-add-right-eq
by smt
    then show ?thesis using c3 by blast
  qed
next
  assume ?Q
  then show ?P
  proof (elim disjE)

    show q = 0 ∧ r = 0 ∧ p = 1 ⟹ p = q ^ r by auto
    show q = 0 ∧ 0 < r ∧ p = 0 ⟹ p = q ^ r by auto

    assume prems: 0 < q ∧ (∃ b m. exp-equations p q r b m)
    obtain b m where cond: exp-equations p q r b m using prems by auto

    hence int b = Exp-Matrices.α (q + 4) (r + 1) + int (q ∗ q) + 2 ∧
                                  m = b ∗ q − q ∗ q − 1 ∧ p < m
      unfolding exp-equations-def power2-eq-square by auto

    moreover have int (p mod m) = (int q ∗ Exp-Matrices.α b r −
                        (int b ∗ Exp-Matrices.α b r − Exp-Matrices.α b (r + 1)))
mod int m
      using cond unfolding exp-equations-def
      using mod-diff-cong[of (p + b ∗ Exp-Matrices.α b r) m (q ∗ Exp-Matrices.α
b r +
        Exp-Matrices.α b (r + 1)) b ∗ Exp-Matrices.α b r b ∗ Exp-Matrices.α b r]
      unfolding diff-diff-eq2 by auto
    ultimately show p = q ^ r using prems exp-alpha by auto
  qed
qed

definition exp (‹[- = - ^ -]› 1000)
  where [Q = R ^ S] ≡ (TERNARY (λa b c. a = b ^ c) Q R S)

lemma exp-dioph[dioph]:
  fixes P Q R :: polynomial
  defines D ≡ [P = Q ^ R]
  shows is-dioph-rel D
proof −
  define P′ Q′ R′ where pushed-def:
    P′ ≡ (push-param P 5) Q′ ≡ (push-param Q 5) R′ ≡ (push-param R 5)
```

```
  define b m a0 a1 a2 where params-def: b = Param 0 m = Param 1 a0 = Param
2
    a1 = Param 3 a2 = Param 4

  define S1 where S1 ≡ [0=] Q [∧] [0=] R [∧] P [=] 1 [∨]
                        [0=] Q [∧] (Const 0) [<] R [∧] [0=] P
  define S2 where S2 ≡ [a0 = α (Q' [+] (Const 4)) (R' [+] 1)]
                        [∧] b [=] (a0 [+] Q'[^2] [+] Const 2)
  define S3 where S3 ≡ (m [+] Q'[^2] [+] Const 1) [=] b [*] Q'
                        [∧] P' [<] m
  define S4 where S4 ≡ [a1 = α b R']
                        [∧] [a2 = α b (R' [+] 1)]
                        [∧] MOD (P' [+] b [*] a1) m (Q' [*] a1 [+] a2)

  note S-defs = S1-def S2-def S3-def S4-def

  define S where S ≡ S1 [∨] (Q [>] Const 0) [∧] ([∃ 5] S2 [∧] S3 [∧] S4)

  have is-dioph-rel S
    unfolding S-def S-defs by (auto simp: dioph)

  moreover have eval S a = eval D a for a
  proof −
    define p q r where evaled-defs: p = peval P a q = peval Q a r = peval R a

    show ?thesis
    proof (rule)
      assume eval S a
      then show eval D a
        unfolding S-def S-defs defs apply (simp add: sq-p-eval)
        unfolding D-def exp-def defs apply simp-all
        unfolding pushed-def params-def apply (auto simp add: push-push[where
?n = 5] push-list-eval)
        unfolding exp-repr exp-equations-def apply simp
        subgoal for ks
          apply (rule exI[of - ks!0], auto)
          subgoal by (simp add: power2-eq-square)
          subgoal apply (rule exI[of - ks!1], auto)
               by (smt int-ops(7) mult-Suc of-nat-Suc of-nat-add power2-eq-square
zmod-int)
          done
        done
    next
      assume eval D a
      then obtain b-val m-val where cond: (q = 0 ∧ r = 0 ∧ p = 1) ∨
                                          (q = 0 ∧ 0 < r ∧ p = 0) ∨
                                          (q > 0 ∧ exp-equations p q r b-val m-val)
        unfolding D-def exp-def exp-repr evaled-defs ternary-eval by auto
```

```
      moreover define a0-val a1-val a2-val where
        a0-val ≡ nat (Exp-Matrices.α (q + 4) (r + 1))
        a1-val ≡ nat (Exp-Matrices.α b-val r)
        a2-val ≡ nat (Exp-Matrices.α b-val (r + 1))
      ultimately show eval S a
        unfolding S-def S-defs defs evaled-defs apply (simp add: sq-p-eval)
        apply (elim disjE)
        subgoal unfolding defs by simp
        subgoal unfolding defs by simp
        subgoal apply(elim conjE) apply(intro disjI2, intro conjI)
          subgoal by simp
          subgoal premises prems
          proof−
            have bg3: 3 < b-val
            proof−
              have b-val = Exp-Matrices.α (q + 4) (r + 1) + int q ∗ int q + 2
                using cond prems(4) evaled-defs(2) unfolding exp-equations-def by
linarith
             moreover have int q ∗ int q > 0 using evaled-defs(2) prems by simp
              moreover have Exp-Matrices.α (q + 4) (r + 1) > 0
                using Exp-Matrices.alpha-superlinear[of q+4 r+1] by linarith
              ultimately show ?thesis by linarith
            qed
            show ?thesis apply (rule exI[of - [b-val, m-val, a0-val, a1-val, a2-val]],
intro conjI)
              using prems
              unfolding exp-equations-def pushed-def params-def
            using push-list-def push-push bg3 Exp-Matrices.alpha-nonnegative apply
simp-all
          subgoal using push-list-def by (smt Exp-Matrices.alpha-strictly-increasing
int-nat-eq
                  nat-int numeral-Bit0 numeral-One of-nat-1 of-nat-add of-nat-power
plus-1-eq-Suc
                    power2-eq-square)
               subgoal using push-list-def apply auto by (smt One-nat-def Suc-1
Suc-less-eq
                  int-nat-eq less-Suc-eq nat-int numeral-3-eq-3 of-nat-add of-nat-mult
zmod-int)
            done
          qed
        done
      done
    qed
  qed

  ultimately show ?thesis
    by (auto simp: is-dioph-rel-def)
qed
```

declare *exp-def*[*defs*]

end

2.4 Digit function is Diophantine

```
theory Digit-Function
  imports Exponential-Relation Digit-Expansions.Bits-Digits
begin

definition digit (‹[ - = Digit - - -]› [999] 1000)
  where [D = Digit AA K BASE] ≡ (QUATERNARY (λd a k b. b > 1
                                   ∧ d = nth-digit a k b) D AA K BASE)
lemma mod-dioph2[dioph]:
  fixes A B C
  defines D ≡ (MOD A B C)
  shows is-dioph-rel D
proof −
  define A′ B′ C′ where pushed-defs: A′ ≡ push-param A 2 B′ ≡ push-param B
2 C′ ≡ push-param C 2
  define DS where DS ≡ [∃ 2] (Param 0 [*] B′ [+] C′ [=] Param 1 [*] B′ [+]
A′)

  have eval DS a = eval D a for a
  proof
    show eval DS a ⟹ eval D a
      unfolding DS-def defs D-def mod-def
      by auto (metis add.commute mod-mult-self1 push-push-simp pushed-defs(1)
                pushed-defs(2) pushed-defs(3))
    show eval D a ⟹ eval DS a
      unfolding DS-def defs D-def mod-def
      apply (auto simp: mod-repr)
      subgoal for x y
        apply (rule exI[of - [x, y]])
      unfolding pushed-defs by (simp add: push-push[where ?n = 2] push-list-eval)
      done
  qed

  moreover have is-dioph-rel DS
    unfolding DS-def by (simp add: dioph)

  ultimately show ?thesis
    by (auto simp: is-dioph-rel-def)
qed

lemma digit-dioph[dioph]:
  fixes D A B K :: polynomial
  defines DR ≡ [D = Digit A K B]
  shows is-dioph-rel DR
```

proof −
define D' A' B' K' **where** *pushed-defs*:
$D' == (push\text{-}param\ D\ 4)$ $A' == (push\text{-}param\ A\ 4)$ $B' == (push\text{-}param\ B\ 4)$ $K' == (push\text{-}param\ K\ 4)$

define x y **where** *param-defs*:
$x == (Param\ 0)$ $y == (Param\ 1)$

define *DS* **where** $DS \equiv [\exists 4]\ (\ B'\ [>]\ Const\ 1\ \ [\wedge]$
$[(Param\ 2) = B'\ \hat{}\ (K'\ [+]\ Const\ 1)]\ [\wedge]$
$[(Param\ 3) = B'\ \hat{}\ K']\ [\wedge]$
$A'\ [=]\ x\ [*]\ (Param\ 2)\ [+]\ D'\ [*]\ (Param\ 3)\ [+]\ y\ [\wedge]$
$D'\ [<]\ B'\ [\wedge]$
$y\ [<]\ (Param\ 3))$
have *eval DS a* = *eval DR a* **for** *a*
proof
show *eval DS a* $\Longrightarrow$ *eval DR a*
unfolding *DS-def defs DR-def digit-def* **apply** *auto*
unfolding *pushed-defs push-push* **using** *pushed-defs push-push digit-gen-equiv* **by** *auto*

assume *asm*: *eval DR a*
then obtain *x-val y-val* **where** *cond*: $(peval\ A\ a) = x\text{-}val * \ (peval\ B\ a)\hat{}((peval\ K\ a)+1)$
$+ (peval\ D\ a) * \ (peval\ B\ a)\hat{}(peval\ K\ a) + y\text{-}val$
$\wedge (peval\ D\ a) < (peval\ B\ a)$
$\wedge\ y\text{-}val < \ (peval\ B\ a)\hat{}(peval\ K\ a)$
unfolding *DS-def defs DR-def digit-def* **using** *digit-gen-equiv* **by** *auto metis*
show *eval DS a*
using *asm* **unfolding** *DS-def defs DR-def digit-def* **apply** *auto*
apply (*rule exI*[*of* - [*x-val*, *y-val*, $(peval\ B\ a)\ \hat{}\ ((peval\ K\ a) + 1)$,
$(peval\ B\ a)\ \hat{}\ (peval\ K\ a)$]])
unfolding *pushed-defs* **using** *param-defs push-push push-list-def cond* **by** *auto+*
qed

moreover have *is-dioph-rel DS*
unfolding *DS-def* **by** (*simp add*: *dioph*)

ultimately show *?thesis*
by (*auto simp*: *is-dioph-rel-def*)
qed

declare *digit-def*[*defs*]

end

2.5 Binomial Coefficient is Diophantine

theory *Binomial-Coefficient*
  imports *Digit-Function*
begin

lemma *bin-coeff-diophantine*:
  shows $c = a\ choose\ b \longleftrightarrow (\exists\, u.(u = 2\hat{}(Suc\ a) \wedge c = nth\text{-}digit\ ((u{+}1)\hat{}a)\ b\ u))$
proof−
  have $(u + 1)\hat{}a = (\sum k{\le}a.\ (a\ choose\ k) * u\ \hat{}\ k)$ **for** u
    using *binomial*[*of u 1 a*] **by** *auto*
  moreover have $a\ choose\ k < 2\ \hat{}\ Suc\ a$ **for** k
    using *binomial-le-pow2*[*of a k*] **by** (*simp add*: *le-less-trans*)
  ultimately have $nth\text{-}digit\ (((2\ \hat{}\ Suc\ a) + 1)\hat{}a)\ b\ (2\ \hat{}\ Suc\ a) = a\ choose\ b$
      using *nth-digit-gen-power-series*[*of* $\lambda k.(a\ choose\ k)$ *a a b*] **by** (*simp add*: *atLeast0AtMost*)
  then show *?thesis* **by** *auto*
qed

definition *binomial-coefficient* (‹[- = - *choose* -]› *1000*)
  where $[A = B\ choose\ C] \equiv (TERNARY\ (\lambda a\ b\ c.\ a = b\ choose\ c)\ A\ B\ C)$

lemma *binomial-coefficient-dioph*[*dioph*]:
  fixes $A\ B\ C$ :: *polynomial*
  defines $DR \equiv [C = A\ choose\ B]$
  shows *is-dioph-rel DR*
proof −
  define $A'\ B'\ C'$ **where** *pushed-def*:
    $A' \equiv (push\text{-}param\ A\ 2)\ B' \equiv (push\text{-}param\ B\ 2)\ C' \equiv (push\text{-}param\ C\ 2)$

  define *DS* **where** $DS \equiv [\exists\, 2]\ [Param\ 0 = Const\ 2\ \hat{}\ (A'\ [+]\ \mathbf{1})]$
      $[\wedge]\ [Param\ 1 = (Param\ 0\ [+]\ \mathbf{1})\ \hat{}\ A']$
      $[\wedge]\ [C' = Digit\ (Param\ 1)\ B'\ (Param\ 0)]$

  have $eval\ DS\ a = eval\ DR\ a$ **for** a
  proof −
    have $eval\ DS\ a = (peval\ C\ a = nth\text{-}digit\ ((2\ \hat{}\ Suc\ (peval\ A\ a) + 1)\hat{}\ peval\ A\ a)$
        $(peval\ B\ a)\ (2\ \hat{}\ Suc\ (peval\ A\ a)))$
      unfolding *DS-def defs pushed-def* **apply** (*auto simp add*: *push-push*)
      apply (*rule exI*[*of* - $[2 * 2\ \hat{}\ peval\ A\ a,\ Suc\ (2 * 2\ \hat{}\ peval\ A\ a)\ \hat{}\ peval\ A\ a]$])
      apply (*auto simp add*: *push-push push-list-eval*)
      by (*metis* (*mono-tags*, *lifting*) *Suc-lessI mult-pos-pos n-not-Suc-n*
          numeral-2-eq-2 one-eq-mult-iff pos2 zero-less-power)

    then show *?thesis*
    unfolding *DR-def binomial-coefficient-def defs* **by** (*simp add*: *bin-coeff-diophantine*)
  qed

```
  moreover have is-dioph-rel DS
    unfolding DS-def by (auto simp: dioph)

  ultimately show ?thesis
    by (auto simp: is-dioph-rel-def)
qed

declare binomial-coefficient-def[defs]
```

odd function is diophantine

```
lemma odd-dioph-repr:
  fixes a :: nat
  shows odd a ⟷ (∃ x::nat. a = 2∗x + 1)
  by (meson dvd-triv-left even-plus-one-iff oddE)

definition odd-lift (‹ODD -› [999] 1000)
  where ODD A ≡ (UNARY (odd) A)

lemma odd-dioph[dioph]:
  fixes A
  defines DR ≡ (ODD A)
  shows is-dioph-rel DR
proof −
  define DS where DS ≡ [∃] (push-param A 1) [=] Const 2 [∗] Param 0 [+]
Const 1

  have eval DS a = eval DR a for a
    unfolding DS-def DR-def odd-lift-def defs using push-push1 by (simp add:
odd-dioph-repr push0)

  moreover have is-dioph-rel DS
    unfolding DS-def by (auto simp: dioph)

  ultimately show ?thesis
    by (auto simp: is-dioph-rel-def)
qed

declare odd-lift-def[defs]

end
```

2.6 Binary orthogonality is Diophantine

```
theory Binary-Orthogonal
 imports Binomial-Coefficient Digit-Expansions.Binary-Operations Lucas-Theorem.Lucas-Theorem
begin

lemma equiv-with-lucas: nth-digit = Lucas-Theorem.nth-digit-general
  unfolding nth-digit-def Lucas-Theorem.nth-digit-general-def by simp
```

```
lemma lm0241-ortho-binom-equiv:(a ⊥ b) ⟷ odd ((a + b) choose b) (is ?P ⟷
?Q)
proof
  assume ?P
  hence dig0:(∀ i. (nth-bit a i) ∗ (nth-bit b i) = 0)
    using ortho-mult-equiv
    by auto
  hence (∀ i. (nth-bit a i) ∗ (nth-bit b i) ≠ 1)
    by presburger
  hence dcons:(∀ i. ¬(((nth-bit a i) = 1) ∧ ((nth-bit b i) = 1)))
    by auto
  hence (∀ i. (bin-carry a b i) = 0) using no-carry-mult-equiv dig0
    by blast
  hence dsum:(∀ i. (nth-bit (a + b) i) = (nth-bit a i) + (nth-bit b i))
    by (metis One-nat-def add.commute add-cancel-right-left add-self-mod-2
        dig0 mult-is-0 not-mod2-eq-Suc-0-eq-0 nth-bit-def one-mod-two-eq-one
        sum-digit-formula)

  have bdd-ab-exists:(∃ p. (a + b) < 2^(Suc p))
    using aux1-lm0241-pow2-up-bound by auto
  then obtain p where bdd-ab:(a + b) < 2^(Suc p) by blast
  moreover from bdd-ab have b < 2^(Suc p) by auto

  ultimately have ((a + b) choose b) mod 2 =
        (∏ i≤p. ((nth-digit (a + b) i 2) choose (nth-digit b i 2))) mod 2
    using lucas-theorem[of a+b 2 p b] bdd-ab two-is-prime-nat
    by (simp add: equiv-with-lucas)

  then have a-choose-b-digit-prod: ((a + b) choose b) mod 2 =
        (∏ i≤p. ((nth-bit (a + b) i) choose (nth-bit b i))) mod 2
    using nth-digit-base2-equiv
    by (auto cong: prod.cong)

  have (∀ i. ((nth-bit (a + b) i) choose (nth-bit b i) = 1))
    using aux2-lm0241-single-digit-binom[where ?a=nth-bit (a + b) i
                                            and ?b=nth-bit b i]
    by (metis add.commute add.right-neutral binomial-n-0 binomial-n-n dig0 dsum
        mult-is-0)
  hence f0:1 = (∏ i<p. (nth-bit (a + b) i) choose (nth-bit b i))
    by simp
  hence f1:... = ... mod 2 by simp
  hence f2:... = ((a + b) choose b) mod 2
    using a-choose-b-digit-prod by (simp add: ‹∀ i. (a + b) ¡ i choose b ¡ i = 1›)
  then show ?Q using f0 by fastforce
next
  assume ?Q
  hence a-choose-b-odd:((a + b) choose b) mod 2 = 1
    using odd-iff-mod-2-eq-one by blast
```

have *bdd-ab-exists*:($\exists$ *p*. (*a* + *b*) < *2*^(*Suc p*))
  using *aux1-lm0241-pow2-up-bound* **by** *auto*
then obtain *p* **where** *bdd-ab*:(*a* + *b*) < *2*^(*Suc p*) **by** *blast*
moreover from *bdd-ab* **have** *bdd-b*: *b* < *2*^(*Suc p*) **by** *auto*

ultimately have ((*a* + *b*) *choose b*) *mod 2* =
    ($\prod$ *i*≤*p*. ((*nth-digit* (*a* + *b*) *i 2*) *choose* (*nth-digit b i 2*))) *mod 2*
  using *lucas-theorem*[*of a+b 2 p b*] *bdd-ab two-is-prime-nat*
  by (*simp add*: *equiv-with-lucas*)

then have *a-choose-b-digit-prod*: ((*a* + *b*) *choose b*) *mod 2* =
    ($\prod$ *i*≤*p*. ((*nth-bit* (*a* + *b*) *i*) *choose* (*nth-bit b i*))) *mod 2*
  using *nth-digit-base2-equiv nth-digit-def*
  by (*auto cong*: *prod.cong*)
hence *all-prod-one-mod2*:... = *1* **using** *a-choose-b-odd* **by** *linarith*

have *choose-bdd*:($\forall$ *i*. *1* ≥ (*nth-bit* (*a* + *b*) *i*) *choose* (*nth-bit b i*))
  using *nth-bit-bounded*
  by (*metis One-nat-def binomial-n-0 choose-one not-mod2-eq-Suc-0-eq-0*
    nth-bit-def order-refl)
hence *1* ≥ ($\prod$ *i*≤*p*. ((*nth-bit* (*a* + *b*) *i*) *choose* (*nth-bit b i*)))
  using *all-prod-one-mod2* **by** (*meson prod-le-1 zero-le*)
hence ($\prod$ *i*≤*p*. ((*nth-bit* (*a* + *b*) *i*) *choose* (*nth-bit b i*))) *mod 2* =
      ($\prod$ *i*≤*p*. ((*nth-bit* (*a* + *b*) *i*) *choose* (*nth-bit b i*)))
  using *all-prod-one-mod2* **by** *linarith*
hence ... = *1*
  using *all-prod-one-mod2* **by** *linarith*
hence *sub-pq-one*:$\forall$ *i*≤*p*. (*nth-bit* (*a* + *b*) *i*) *choose* (*nth-bit b i*) = *1*
  using
    aux4-lm0241-prod-one[**where** *?n*=*p* **and** *?f*=(λ*i*. (*nth-bit* (*a* + *b*) *i*) *choose*
(*nth-bit b i*))]
    choose-bdd **by** *blast*

have $\forall$ *r* > *p*. (*a* + *b*) < *2*^*r* **using** *bdd-ab*
  by(*metis One-nat-def Suc-lessI lessI less-imp-add-positive numeral-2-eq-2*
    power-strict-increasing-iff trans-less-add1)
hence $\forall$ *r* > *p*. *r* ≥ *p* ⟶ (*a* + *b*) < *2*^*r* **by** *auto*
hence *ab-digit-bdd*:$\forall$ *r* > *p*. *r* ≥ *p* ⟶ *nth-bit* (*a* + *b*) *r* = *0*
  using *nth-bit-def* **by** *simp*

have $\forall$ *k* > *p*. *b* < *2* ^ *k* **using** *bdd-b*
  by(*metis One-nat-def Suc-lessI lessI less-imp-add-positive numeral-2-eq-2*
    power-strict-increasing-iff trans-less-add1)
hence *b-digit-bdd*:$\forall$ *k* > *p*. *k* ≥ *p* ⟶ *nth-bit b k* = *0*
  using *nth-bit-def*
  by (*simp add*: ‹$\forall$ *k*>*p*. *b* < *2* ^ *k*›)

from *b-digit-bdd ab-digit-bdd aux3-lm0241-binom-bounds*

have ∀ *i*. $i \geq p \longrightarrow$ (*nth-bit* (*a* + *b*) *i*) *choose* (*nth-bit b i*) = *1*
by (*simp add*: *le-less sub-pq-one*)

hence ∀ *i*. ((*nth-bit* (*a* + *b*) *i*) *choose* (*nth-bit b i*)) = *1*
using *sub-pq-one not-less* **by** (*metis linear*)
hence ∀ *i*. ¬(*nth-bit a i* = *1* ∧ *nth-bit b i* = *1*) **using** *aux5-lm0241* **by** *blast*
hence ∀ *i*. ((*nth-bit a i* = *0* ∧ *nth-bit b i* = *1*) ∨
(*nth-bit a i* = *1* ∧ *nth-bit b i* = *0*) ∨
(*nth-bit a i* = *0* ∧ *nth-bit b i* = *0*))
by (*auto simp add*:*nth-bit-def nth-digit-bounded*; *metis nat.simps*(*3*))
hence ∀ *i*. (*nth-bit a i*) ∗ (*nth-bit b i*) = *0* **using** *mult-is-0* **by** *blast*
then show *?P* **using** *ortho-mult-equiv* **by** *blast*
qed

definition *orthogonal* (**infix** ‹[⊥]› *50*)
where *P* [⊥] *Q* ≡ (*BINARY* (λ*a b*. *a* ⊥ *b*) *P Q*)

lemma *orthogonal-dioph*[*dioph*]:
fixes *P Q*
defines *DR* ≡ (*P* [⊥] *Q*)
shows *is-dioph-rel DR*
proof −
define *P′ Q′* **where** *pushed-def*: *P′* ≡ *push-param P 1 Q′* ≡ *push-param Q 1*

define *DS* **where** *DS* ≡ [∃] [*Param 0* = (*P′* [+] *Q′*) *choose Q′*] [∧] *ODD* (*Param 0*)

have *eval DS a* = *eval DR a* **for** *a*
unfolding *DS-def DR-def orthogonal-def pushed-def defs*
using *push-push1 lm0241-ortho-binom-equiv* **by** (*simp add*: *push0*)

moreover have *is-dioph-rel DS*
unfolding *DS-def* **by** (*simp add*: *dioph*)

ultimately show *?thesis*
by (*auto simp*: *is-dioph-rel-def*)
qed

declare *orthogonal-def*[*defs*]

end

2.7 Binary masking is Diophantine

theory *Binary-Masking*
imports *Binary-Orthogonal*
begin

lemma *lm0243-masks-binom-equiv*: (*b* ⪯ *c*) ⟷ *odd* (*c choose b*) (**is** *?P* ⟷ *?Q*)

```
proof-
  consider (lt) b < c | (eq) b = c | (gt) b > c using nat-neq-iff by blast
  then show ?thesis
  proof(cases)
    case lt
    hence ∃ a. c = a + b using less-imp-add-positive semiring-normalization-rules(24)
by blast
    then obtain a where a-def:c = a + b ..
    have a ⊥ b ⟷ b ≼ a + b (is ?P ⟷ ?Q)
    proof
      assume ?P
      then show ?Q
        using ortho-mult-equiv no-carry-mult-equiv masks-leq-equiv[of b a+b]
              sum-digit-formula nth-bit-bounded
        by auto (metis add.commute add.right-neutral lessI less-one mod-less
                    nat-less-le one-add-one plus-1-eq-Suc zero-le)
    next
      assume ?Q
      have ?Q ⟹ ∀ k. a ¡ k + b ¡ k ≤ 1
      proof(rule ccontr)
        assume ¬(∀ k. a ¡ k + b ¡ k ≤ 1)
        then obtain k where k1:¬(a ¡ k + b ¡ k ≤ 1) and k2:∀ r<k. a ¡ r + b ¡ r
≤ 1
          by (auto dest: obtain-smallest)
        have c1: bin-carry a b k = 1
        using ‹?Q› masks-leq-equiv sum-digit-formula carry-bounded nth-bit-bounded
k1
          by (metis add.commute add.left-neutral add-self-mod-2 less-one nat-less-le
not-le)
        then show False using carry-digit-impl[of a b k] k2 by auto
      qed
      then show ?P
        using ‹?Q› ortho-mult-equiv no-carry-mult-equiv masks-leq-equiv[of b a+b]
              sum-digit-formula nth-bit-bounded
        by auto (metis add-le-same-cancel2 le-0-eq le-SucE)
    qed
    then show ?thesis using lm0241-ortho-binom-equiv a-def by auto
  next
    case eq
    hence odd (c choose b) by simp
    moreover have b ≼ c using digit-wise-equiv masks-leq-equiv eq by blast
    ultimately show ?thesis by simp
  next
    case gt
    hence ¬ odd (c choose b) by (simp add: binomial-eq-0)
    moreover have ¬ (b ≼ c) using masks-leq-equiv masks-leq gt not-le by blast
    ultimately show ?thesis by simp
  qed
qed
```

definition *masking* (‹- [⪯] -› *60*)
  where *P* [⪯] *Q* ≡ (*BINARY* (λ*a b*. *a* ⪯ *b*) *P Q*)

lemma *masking-dioph*[*dioph*]:
  fixes *P Q*
  defines *DR* ≡ (*P* [⪯] *Q*)
  shows *is-dioph-rel DR*
proof −
  define *P′ Q′* **where** *pushed-def*: *P′* ≡ *push-param P 1 Q′* ≡ *push-param Q 1*

  define *DS* **where** *DS* ≡ [∃] [*Param 0* = *Q′ choose P′*] [∧] *ODD Param 0*

  have *eval DS a* = *eval DR a* **for** *a*
    unfolding *DS-def DR-def defs pushed-def masking-def*
    using *push-push1* **by** (*simp add*: *push0 lm0243-masks-binom-equiv*)

  moreover have *is-dioph-rel DS*
    unfolding *DS-def* **by** (*simp add*: *dioph*)

  ultimately show *?thesis*
    by (*auto simp*: *is-dioph-rel-def*)
qed

declare *masking-def*[*defs*]

end

2.8 Binary and is Diophantine

theory *Binary-And*
  imports *Binary-Masking Binary-Orthogonal*
begin

lemma *lm0244*: (*a* && *b*) ⪯ *a*
proof (*induct a b rule*: *bitAND-nat.induct*)
  case (*1 uu*)
  then show *?case* **by** *simp*
next
  case (*2 v n*)
  then show *?case*
    apply (*auto simp add*: *mult.commute*)
      by (*smt One-nat-def add-cancel-left-right even-succ-div-two masks.elims*(*3*)
mod-Suc-le-divisor
      mod-by-Suc-0 mod-mod-trivial mod-mult-self4 mult-numeral-1-right mult-zero-right
          nonzero-mult-div-cancel-left not-mod2-eq-Suc-0-eq-0 numeral-2-eq-2 nu-
meral-One
        odd-two-times-div-two-succ zero-neq-numeral)
qed

```
lemma lm0245: (a && b) ⪯ b
  by (subst bitAND-commutes) (simp add: lm0244)

lemma bitAND-lt-left: m && n ≤ m
  using lm0244 masks-leq by auto
lemma bitAND-lt-right: m && n ≤ n
  using lm0245 masks-leq by auto

lemmas bitAND-lt = bitAND-lt-right bitAND-lt-left

lemma auxm3-lm0246:
  shows bin-carry a b k = bin-carry a b k mod 2
  using bin-carry-bounded by auto

lemma auxm2-lm0246:
  assumes (∀ r< n.(nth-bit a r + nth-bit b r ≤ 1))
  shows (nth-bit (a+b) n) = (nth-bit a n + nth-bit b n) mod 2
  using assms no-carry by auto

lemma auxm1-lm0246: a ⪯ (a+b) ⟹ (∀ n. nth-bit a n + nth-bit b n ≤ 1) (is ?P
⟹ ?Q)
proof−
{
  assume asm: ¬?Q
  then obtain n where n1:¬(nth-bit a n + nth-bit b n ≤ 1)
    and n2:∀ r < n. (nth-bit a r + nth-bit b r ≤ 1)
    using obtain-smallest by (auto dest: obtain-smallest)
  hence ab1: nth-bit a n =1  ∧  nth-bit b n = 1 using nth-bit-def by auto
  hence nth-bit (a+b) n = 0 using n2 auxm2-lm0246 by auto
 hence ¬?P using masks-leq-equiv ab1 by auto (metis One-nat-def not-one-le-zero)
} then show ?P ⟹ ?Q by auto
qed

lemma aux0-lm0246:a ⪯ (a+b) ⟶(a+b)¡ n = a ¡ n + b ¡ n
proof−
  show ?thesis using auxm1-lm0246 auxm2-lm0246 less-Suc-eq-le numeral-2-eq-2
by auto
qed

lemma aux1-lm0246:a⪯b ⟶ (∀ n. nth-bit (b−a) n = nth-bit b n − nth-bit a n)
 using aux0-lm0246 masks-leq by auto (metis add-diff-cancel-left′ le-add-diff-inverse)

lemma lm0246:(a − (a && b)) ⊥ (b − (a && b))
  apply (subst ortho-mult-equiv)
  apply (rule allI) subgoal for k
  proof(cases nth-bit a k = 0)
    case True
     have nth-bit (a− (a && b)) k = 0 by (auto simp add: lm0244 aux1-lm0246
```

```
True)
    then show ?thesis by simp
  next
    case False
    then show ?thesis proof(cases nth-bit b k = 0)
      case True
      have nth-bit (b− (a && b)) k = 0 by (auto simp add: lm0245 aux1-lm0246
True)
      then show ?thesis by simp
    next
      case False2: False
      have nth-bit a k = 1 using False nth-bit-def by auto
      moreover have nth-bit b k = 1 using False2 nth-bit-def by auto
      ultimately have nth-bit (b− (a && b)) k = 0
        by (auto simp add: lm0245 aux1-lm0246 bitAND-digit-mult)
      then show ?thesis by simp
    qed
  qed
done

lemma aux0-lm0247:(nth-bit a k) ∗ (nth-bit b k) ≤ 1
  using eq-iff nth-bit-def by fastforce

lemma lm0247-masking-equiv:
  fixes a b c :: nat
  shows (c = a && b) ⟷ (c ⪯ a ∧ c ⪯ b ∧ (a − c) ⊥ (b − c)) (is ?P ⟷ ?Q)
proof (rule)
  assume ?P
  thus ?Q
    apply (auto simp add: lm0244 lm0245)
    using lm0246 orthogonal.simps by blast
next
  assume Q: ?Q
  have (∀k. nth-bit c k ≤ nth-bit a k ∧ nth-bit c k ≤ nth-bit b k)
    using Q masks-leq-equiv by auto
  moreover have (∀k x. nth-bit x k ≤ 1)
    by (auto simp add: nth-bit-def)
  ultimately have f0:(∀k. nth-bit c k ≤ ((nth-bit a k) ∗ (nth-bit b k)))
    by (metis mult.right-neutral mult-0-right not-mod-2-eq-0-eq-1 nth-bit-def)
  show ?Q ⟹ ?P
  proof (rule ccontr)
    assume contr:c ≠ a && b
    have k-exists:(∃k. (nth-bit c k) < ((nth-bit a k) ∗ (nth-bit b k)))
      using bitAND-mult-equiv by (meson f0 contr le-less)
    then obtain k
      where (nth-bit c k) < ((nth-bit a k) ∗ (nth-bit b k)) ..
    hence abc-kth:((nth-bit c k) = 0) ∧ ((nth-bit a k) = 1) ∧ ((nth-bit b k) = 1)
      using aux0-lm0247 less-le-trans
      by (metis One-nat-def Suc-leI nth-bit-bounded less-le less-one one-le-mult-iff)
```

```
    hence (nth-bit (a − c) k) = 1 ∧ (nth-bit (b − c) k) = 1
      by (auto simp add: abc-kth aux1-lm0246 Q)
    hence ¬ ((a − c) ⊥ (b − c))
      by (metis mult.left-neutral not-mod-2-eq-0-eq-1 ortho-mult-equiv)
    then show False
      using Q by blast
  qed
qed

definition binary-and (‹[- = - && -]› 1000)
  where [A = B && C] ≡ (TERNARY (λa b c. a = b && c) A B C)

lemma binary-and-dioph[dioph]:
  fixes A B C :: polynomial
  defines DR ≡ [A = B && C]
  shows is-dioph-rel DR
proof −
  define DS where DS ≡ (A [⪯] B) [∧] (A [⪯] C) [∧] (B [−] A) [⊥] (C [−] A)

  have eval DS a = eval DR a for a
    unfolding DS-def DR-def binary-and-def defs
    by (simp add: lm0247-masking-equiv)

  moreover have is-dioph-rel DS
    unfolding DS-def by (auto simp: dioph)

  ultimately show ?thesis
    by (auto simp: is-dioph-rel-def)
qed

declare binary-and-def[defs]


definition binary-and-attempt :: polynomial ⇒ polynomial ⇒ polynomial (‹- &?
-›) where
  A &? B ≡ Const 0

end
```

3 Register Machines

3.1 Register Machine Specification

```
theory RegisterMachineSpecification
  imports Main
begin
```

3.1.1 Basic Datatype Definitions

The following specification of register machines is inspired by [8] (see [9] for the corresponding AFP article).

type-synonym *register* = *nat*
type-synonym *tape* = *register list*

type-synonym *state* = *nat*
datatype *instruction* =
  isadd: *Add* (*modifies* : *register*) (*goes-to* : *state*) |
  issub: *Sub* (*modifies* : *register*) (*goes-to* : *state*) (*goes-to-alt* : *state*) |
  ishalt: *Halt*
where
  modifies Halt = *0* |
  goes-to-alt (*Add* - *next*) = *next*

type-synonym *program* = *instruction list*

type-synonym *configuration* = (*state* ∗ *tape*)

3.1.2 Essential Functions to operate the Register Machine

definition *read* :: *tape* ⇒ *program* ⇒ *state* ⇒ *nat*
  where *read t p s* = *t* ! (*modifies* (*p*!*s*))

definition *fetch* :: *state* ⇒ *program* ⇒ *nat* ⇒ *state* **where**
  fetch s p v = (*if issub* (*p*!*s*) ∧ *v* = *0 then goes-to-alt* (*p*!*s*)
      else if ishalt (*p*!*s*) *then s*
      else goes-to (*p*!*s*))

definition *update* :: *tape* ⇒ *instruction* ⇒ *tape* **where**
  update t i = (*if ishalt i then t*
      else if isadd i then list-update t (*modifies i*) (*t*!(*modifies i*) + *1*)
      else list-update t (*modifies i*) (*if t*!(*modifies i*) = *0 then 0 else* (*t*!(*modifies i*)) − *1*))

definition *step* :: *configuration* ⇒ *program* ⇒ *configuration*
  where
    (*step ic p*) = (*let nexts* = *fetch* (*fst ic*) *p* (*read* (*snd ic*) *p* (*fst ic*));
      nextt = *update* (*snd ic*) (*p*!(*fst ic*))
      in (*nexts*, *nextt*))

fun *steps* :: *configuration* ⇒ *program* ⇒ *nat* ⇒ *configuration*
  where
    steps-zero: (*steps c p 0*) = *c*
  | *steps-suc*: (*steps c p* (*Suc n*)) = (*step* (*steps c p n*) *p*)

3.1.3 Validity Checks and Assumptions

fun *instruction-state-check* :: *nat* ⇒ *instruction* ⇒ *bool*
where *isc-halt*: *instruction-state-check* - *Halt* = *True*
| *isc-add*: *instruction-state-check m* (*Add* - *ns*) = (*ns* < *m*)
| *isc-sub*: *instruction-state-check m* (*Sub* - *ns1 ns2*) = ((*ns1* < *m*) & (*ns2* < *m*))

fun *instruction-register-check* :: *nat* ⇒ *instruction* ⇒ *bool*
where *instruction-register-check* - *Halt* = *True*
| *instruction-register-check n* (*Add reg* -) = (*reg* < *n*)
| *instruction-register-check n* (*Sub reg* - -) = (*reg* < *n*)

fun *program-state-check* :: *program* ⇒ *bool*
where *program-state-check p* = *list-all* (*instruction-state-check* (*length p*)) *p*

fun *program-register-check* :: *program* ⇒ *nat* ⇒ *bool*
where *program-register-check p n* = *list-all* (*instruction-register-check n*) *p*

fun *tape-check-initial* :: *tape* ⇒ *nat* ⇒ *bool*
where *tape-check-initial t a* = (*t* ≠ [] ∧ *t*!*0* = *a* ∧ (∀ *l*>*0*. *t* ! *l* = *0*))

fun *program-includes-halt* :: *program* ⇒ *bool*
where *program-includes-halt p* = (*length p* > *1* ∧ *ishalt* (*p* ! (*length p* −*1*)) ∧ (∀ *k*<*length p*−*1*. ¬ *ishalt* (*p*!*k*)))

Is Valid and Terminates

definition *is-valid*
where *is-valid c p* = (*program-includes-halt p* ∧ *program-state-check p*
∧ (*program-register-check p* (*length* (*snd c*))))

definition *is-valid-initial*
where *is-valid-initial c p a* = ((*is-valid c p*)
∧ (*tape-check-initial* (*snd c*) *a*)
∧ (*fst c* = *0*))

definition *correct-halt*
where *correct-halt c p q* = (*ishalt* (*p* ! (*fst* (*steps c p q*))) — halting
∧ (∀ *l*<(*length* (*snd c*)). *snd* (*steps c p q*) ! *l* = *0*))

definition *terminates* :: *configuration* ⇒ *program* ⇒ *nat* ⇒ *bool*
where *terminates c p q* = ((*q*>*0*)
∧ (*correct-halt c p q*)
∧ (∀ *x*<*q*. ¬ *ishalt* (*p* ! (*fst* (*steps c p x*)))))

definition *initial-config* :: *nat* ⇒ *nat* ⇒ *configuration* **where**
initial-config n a = (*0*, (*a* # *replicate n 0*))

end

3.2 Simple Properties of Register Machines

```
theory RegisterMachineProperties
  imports RegisterMachineSpecification
begin

lemma step-commutative: steps (step c p) p t = step (steps c p t) p
  by (induction t; auto)

lemma step-fetch-correct:
  fixes t :: nat
    and c :: configuration
    and p :: program
  assumes is-valid c p
  defines ct ≡ (steps c p t)
  shows fst (steps (step c p) p t) = fetch (fst ct) p (read (snd ct) p (fst ct))
  using ct-def step-commutative step-def by auto
```

3.2.1 From Configurations to a Protocol

Register Values

```
definition R :: configuration ⇒ program ⇒ nat ⇒ nat ⇒ nat
  where R c p n t = (snd (steps c p t)) ! n

fun RL :: configuration ⇒ program ⇒ nat ⇒ nat ⇒ nat ⇒ nat  where
  RL c p b 0 l = ((snd c) ! l) |
  RL c p b (Suc t) l = ((snd c) ! l) + b ∗ (RL (step c p) p b t l)

lemma RL-simp-aux:
  ‹snd c ! l + b ∗ RL (step c p) p b t l =
    RL c p b t l + b ∗ (b ^ t ∗ snd (step (steps c p t) p) ! l)›
  by (induction t arbitrary: c)
    (auto simp: step-commutative algebra-simps)

declare RL.simps[simp del]
lemma RL-simp:
  RL c p b (Suc t) l = (snd (steps c p (Suc t)) ! l) ∗ b ^(Suc t) + (RL c p b t l)
proof (induction t arbitrary: p c b)
  case 0
  thus ?case by (auto simp: RL.simps)
next
  case (Suc t p c b)
  show ?case
    by (subst RL.simps)
      (auto simp: Suc step-commutative algebra-simps RL-simp-aux)
qed
```

State Values

```
definition S :: configuration ⇒ program ⇒ nat ⇒ nat ⇒ nat
```

where *S c p k t* = (*if* (*fst* (*steps c p t*) = *k*) *then* (*Suc 0*) *else 0*)

definition *S2* :: *configuration* ⇒ *nat* ⇒ *nat*
where *S2 c k* = (*if* (*fst c*) = *k then 1 else 0*)

fun *SK* :: *configuration* ⇒ *program* ⇒ *nat* ⇒ *nat* ⇒ *nat* ⇒ *nat*
where *SK c p b 0 k* = (*S2 c k*) |
SK c p b (*Suc t*) *k* = (*S2 c k*) + *b* ∗ (*SK* (*step c p*) *p b t k*)

lemma *SK-simp-aux*:
‹*SK c p b* (*Suc* (*Suc t*)) *k* =
S2 (*steps c p* (*Suc* (*Suc t*))) *k* ∗ *b* ^ *Suc* (*Suc t*) + *SK c p b* (*Suc t*) *k*›
by (*induction t arbitrary*: *c*) (*auto simp*: *step-commutative algebra-simps*)

declare *SK.simps*[*simp del*]
lemma *SK-simp*:
SK c p b (*Suc t*) *k* = (*S2* (*steps c p* (*Suc t*)) *k*) ∗ *b* ^(*Suc t*) + (*SK c p b t k*)
proof (*induction t arbitrary*: *p c b k*)
case *0*
thus *?case* **by** (*auto simp*: *SK.simps*)
next
case (*Suc t p c b k*)
show *?case*
by (*auto simp*: *Suc algebra-simps step-commutative SK-simp-aux*)
qed

Zero-Indicator Values

definition *Z* :: *configuration* ⇒ *program* ⇒ *nat* ⇒ *nat* ⇒ *nat* **where**
Z c p n t = (*if* (*R c p n t* > *0*) *then 1 else 0*)

definition *Z2* :: *configuration* ⇒ *nat* ⇒ *nat* **where**
Z2 c n = (*if* (*snd c*) ! *n* > *0 then 1 else 0*)

fun *ZL* :: *configuration* ⇒ *program* ⇒ *nat* ⇒ *nat* ⇒ *nat* ⇒ *nat*
where *ZL c p b 0 l* = (*Z2 c l*) |
ZL c p b (*Suc t*) *l* = (*Z2 c l*) + *b* ∗ (*ZL* (*step c p*) *p b t l*)

lemma *ZL-simp-aux*:
Z2 c l + *b* ∗ *ZL* (*step c p*) *p b t l* =
ZL c p b t l + *b* ∗ (*b* ^ *t* ∗ *Z2* (*step* (*steps c p t*) *p*) *l*)
by (*induction t arbitrary*: *c*) (*auto simp*: *step-commutative algebra-simps*)

declare *ZL.simps*[*simp del*]
lemma *ZL-simp*:
ZL c p b (*Suc t*) *l* = (*Z2* (*steps c p* (*Suc t*)) *l*) ∗ *b* ^(*Suc t*) + (*ZL c p b t l*)
proof (*induction t arbitrary*: *p c b*)
case *0*
thus *?case* **by** (*auto simp*: *ZL.simps*)
next

```
  case (Suc t p c b)
  show ?case
   by (subst ZL.simps) (auto simp: Suc step-commutative algebra-simps ZL-simp-aux)
qed
```

3.2.2 Protocol Properties

```
lemma Z-bounded: Z c p l t ≤ 1
  by (auto simp: Z-def)

lemma S-bounded: S c p k t ≤ 1
  by (auto simp: S-def)

lemma S-unique: ∀ k≤length p. (k ≠ fst (steps c p t) ⟶ S c p k t = 0)
  by (auto simp: S-def)


fun cells-bounded :: configuration ⇒ program ⇒ nat ⇒ bool where
  cells-bounded conf p c = ((∀ l<(length (snd conf)). ∀ t. 2^c > R conf p l t)
                            ∧  (∀ k t. 2^c > S conf p k t)
                            ∧  (∀ l t. 2^c > Z conf p l t))

lemma steps-tape-length-invar:  length (snd (steps c p t)) = length (snd c)
  by (induction t; auto simp add: step-def update-def)

lemma step-is-valid-invar: is-valid c p ⟹ is-valid (step c p) p
  by (auto simp add: step-def update-def is-valid-def)

fun fetch-old
  where
    (fetch-old p s (Add r next) -) = next
  | (fetch-old p s (Sub r next nextalt) val) = (if val = 0 then nextalt else next)
  | (fetch-old p s Halt -) = s

lemma fetch-equiv:
  assumes i = p!s
  shows fetch s p v = fetch-old p s i v
  by (cases i; auto simp: assms fetch-def)


lemma p-contains: is-valid-initial ic p a ⟹ (fst (steps ic p t)) < length p
proof −
  assume asm: is-valid-initial ic p a
  hence fst ic = 0 using is-valid-initial-def is-valid-def by blast
  hence 0: ic = (0, snd ic) by (metis prod.collapse)
  show ?thesis using 0 asm
  apply (induct t) apply auto[1]
  subgoal by (auto simp add: is-valid-initial-def is-valid-def)
```

```
  apply (cases p ! fst (steps ic p t))
  apply (auto simp add: list-all-length fetch-equiv step-def
              is-valid-initial-def is-valid-def fetch-old.elims)
 by (metis RegisterMachineSpecification.isc-add RegisterMachineSpecification.isc-sub

      fetch-old.elims) +
qed

lemma steps-is-valid-invar: is-valid c p ⟹ is-valid (steps c p t) p
  by (induction t; auto simp add: step-def update-def is-valid-def)

lemma terminates-halt-state: terminates ic p q ⟹ is-valid-initial ic p a
                              ⟹ ishalt (p ! (fst (steps ic p q)))
proof −
  assume terminate: terminates ic p q
  assume is-val: is-valid-initial ic p a
  have 1 < length p using is-val is-valid-initial-def[of ic p a]
    is-valid-def[of ic p] program-includes-halt.simps
    by blast
  hence p ≠ [] by auto
  hence p ! (length p − 1) = last p using List.last-conv-nth[of p] by auto
  thus ?thesis
     using terminate terminates-def correct-halt-def is-val is-valid-def[of ic p] by
auto
qed

lemma R-termination:
  fixes l :: register and ic :: configuration
  assumes is-val: is-valid ic p and terminate: terminates ic p q and l: l < length
(snd ic)
  shows ∀ t≥q. R ic p l t = 0
proof −
  have ishalt: ishalt (p ! fst (steps ic p q))
    using terminate terminates-def correct-halt-def is-valid-def is-val by auto
  have halt: ishalt (p ! fst (steps ic p (q + t))) for t
    apply (induction t)
    using terminate terminates-def ishalt step-def fetch-def by auto
  have l<(length (snd ic)) ⟶R ic p l (q+t) = 0 for t
    apply (induction t arbitrary: l)
    subgoal using terminate terminates-def correct-halt-def R-def by auto
    subgoal using R-def step-def halt update-def by auto
    done
  thus ?thesis using le-Suc-ex l by force
qed

lemma terminate-c-exists: is-valid ic p ⟹ terminates ic p q ⟹ ∃ c>1. cells-bounded
ic p c
proof −
  assume is-val: is-valid ic p
```

assume *terminate*: *terminates ic p q*
define *n* **where** $n \equiv length\ (snd\ ic)$
define *rmax* **where** $rmax \equiv Max\ (\{k.\ \exists\, l<n.\ \exists\, t<q.\ k = R\ ic\ p\ l\ t\} \cup \{2\})$
have $\forall\, l<n.\ \forall\, t<q.\ R\ ic\ p\ l\ t \in \{k.\ \exists\, l<n.\ \exists\, t<q.\ k = R\ ic\ p\ l\ t\}$ **by** *auto*
hence $\forall\, t<q.\ \forall\, l<n.\ R\ ic\ p\ l\ t \leq rmax$ **using** *rmax-def* **by** *auto*
moreover have $\forall\, t \geq q.\ \forall\, l<n.\ R\ ic\ p\ l\ t \leq rmax$
using *rmax-def R-termination terminate n-def is-val* **by** *auto*
ultimately have *r*: $\forall\, l<n.\ \forall\, t.\ R\ ic\ p\ l\ t \leq rmax$ **using** *not-le-imp-less* **by** *blast*
have *gt2*: $rmax \geq 2$ **using** *rmax-def* **by** *auto*
hence *sz*: $(\forall\, k\ t.\ rmax > S\ ic\ p\ k\ t) \wedge (\forall\, l\ t.\ rmax > Z\ ic\ p\ l\ t)$
using *S-bounded Z-bounded S-def Z-def* **by** *auto*
have $(\forall\, l<n.\ \forall\, t.\ R\ ic\ p\ l\ t < 2\hat{\ }rmax) \wedge (\forall\, k\ t.\ S\ ic\ p\ k\ t < 2\hat{\ }rmax)$
$\wedge\ (\forall\, l\ t.\ Z\ ic\ p\ l\ t < 2\hat{\ }rmax)$
using *less-exp*[*of rmax*] *r sz* **by** (*metis le-neq-implies-less dual-order.strict-trans*)
moreover have $rmax > 1$ **using** *gt2* **by** *auto*
ultimately show *?thesis* **using** *n-def* **by** *auto*
qed

end

3.3 Simulation of a Register Machine

theory *RegisterMachineSimulation*
imports *RegisterMachineProperties Digit-Expansions.Binary-Operations*
begin

definition $B :: nat \Rightarrow nat$ **where**
$(B\ c) = 2\hat{\ }(Suc\ c)$

definition $RLe\ c\ p\ b\ q\ l = (\sum t = 0..q.\ b\hat{\ }t * R\ c\ p\ l\ t)$
definition $SKe\ c\ p\ b\ q\ k = (\sum t = 0..q.\ b\hat{\ }t * S\ c\ p\ k\ t)$
definition $ZLe\ c\ p\ b\ q\ l = (\sum t = 0..q.\ b\hat{\ }t * Z\ c\ p\ l\ t)$

fun *sum-radd* :: $program \Rightarrow register \Rightarrow (nat \Rightarrow nat) \Rightarrow nat$
where *sum-radd* $p\ l\ f = (\sum k = 0..length\ p-1.\ if\ isadd\ (p!k) \wedge l = modifies\ (p!k)\ then\ f\ k\ else\ 0)$

abbreviation *sum-radd-abbrev* (‹$\sum$ R+ - - -› [999, 999, 999] 1000)
where ($\sum$ R+ $p\ l\ f$) $\equiv$ (*sum-radd* $p\ l\ f$)

fun *sum-rsub* :: $program \Rightarrow register \Rightarrow (nat \Rightarrow nat) \Rightarrow nat$
where *sum-rsub* $p\ l\ f = (\sum k = 0..length\ p-1.\ if\ issub\ (p!k) \wedge l = modifies\ (p!k)\ then\ f\ k\ else\ 0)$

abbreviation *sum-rsub-abbrev* (‹$\sum$ R− - - - › [999, 999, 999] 1000)
where ($\sum$ R− $p\ l\ f$) $\equiv$ (*sum-rsub* $p\ l\ f$)

fun *sum-sadd* :: $program \Rightarrow state \Rightarrow (nat \Rightarrow nat) \Rightarrow nat$

where *sum-sadd p d f* = ($\sum$ *k* = *0..length p*−*1*. *if isadd* (*p*!*k*) ∧ *d* = *goes-to* (*p*!*k*) *then f k else 0*)

abbreviation *sum-sadd-abbrev* (‹$\sum$ *S*+ - - - › [*999*, *999*, *999*] *1000*)
where ($\sum$ *S*+ *p d f*) ≡ (*sum-sadd p d f*)

fun *sum-ssub-nzero* :: *program* ⇒ *state* ⇒ (*nat* ⇒ *nat*) ⇒ *nat*
where *sum-ssub-nzero p d f* = ($\sum$ *k* = *0..length p*−*1*. *if issub* (*p*!*k*) ∧ *d* = *goes-to* (*p*!*k*) *then f k else 0*)

abbreviation *sum-ssub-nzero-abbrev* (‹$\sum$ *S*− - - - › [*999*, *999*, *999*] *1000*)
where ($\sum$ *S*− *p d f*) ≡ (*sum-ssub-nzero p d f*)

fun *sum-ssub-zero* :: *program* ⇒ *state* ⇒ (*nat* ⇒ *nat*) ⇒ *nat*
where *sum-ssub-zero p d f* = ($\sum$ *k* = *0..length p*−*1*. *if issub* (*p*!*k*) ∧ *d* = *goes-to-alt* (*p*!*k*) *then f k else 0*)

abbreviation *sum-ssub-zero-abbrev* (‹$\sum$ *S0* - - - › [*999*, *999*, *999*] *1000*)
where ($\sum$ *S0 p d f*) ≡ (*sum-ssub-zero p d f*)

declare *sum-radd.simps*[*simp del*]
declare *sum-rsub.simps*[*simp del*]
declare *sum-sadd.simps*[*simp del*]
declare *sum-ssub-nzero.simps*[*simp del*]
declare *sum-ssub-zero.simps*[*simp del*]

Special sum cong lemmas

lemma *sum-sadd-cong*:
assumes ∀ *k*. *k* ≤ *length p*−*1* ∧ *isadd* (*p*!*k*) ∧ *l* = *goes-to* (*p*!*k*) ⟶ *f k* = *g k*
shows $\sum$ *S*+ *p l f* = $\sum$ *S*+ *p l g*
unfolding *sum-sadd.simps*
by (*rule sum.cong*, *simp*) (*rule if-cong*, *simp-all add*: *assms*)

lemma *sum-ssub-nzero-cong*:
assumes ∀ *k*. *k* ≤ *length p* − *1* ∧ *issub* (*p*!*k*) ∧ *l* = *goes-to* (*p*!*k*) ⟶ *f k* = *g k*
shows $\sum$ *S*− *p l f* = $\sum$ *S*− *p l g*
unfolding *sum-ssub-nzero.simps*
by (*rule sum.cong*, *simp*) (*rule if-cong*, *simp-all add*: *assms*)

lemma *sum-ssub-zero-cong*:
assumes ∀ *k*. *k* ≤ *length p* − *1* ∧ *issub* (*p*!*k*) ∧ *l* = *goes-to-alt* (*p*!*k*) ⟶ *f k* = *g k*
shows $\sum$ *S0 p l f* = $\sum$ *S0 p l g*
unfolding *sum-ssub-zero.simps*
by (*rule sum.cong*, *simp*) (*rule if-cong*, *simp-all add*: *assms*)

lemma *sum-radd-cong*:
assumes ∀ *k*. *k* ≤ *length p* − *1* ∧ *isadd* (*p*!*k*) ∧ *l* = *modifies* (*p*!*k*) ⟶ *f k* = *g k*

shows $\sum R+ \ p \ l \ f = \sum R+ \ p \ l \ g$
unfolding *sum-radd.simps*
by (*rule sum.cong*, *simp*) (*rule if-cong*, *simp-all add*: *assms*)

lemma *sum-rsub-cong*:
assumes $\forall k.\ k \le length\ p - 1 \wedge issub\ (p!k) \wedge l = modifies\ (p!k) \longrightarrow f\ k = g\ k$
shows $\sum R- \ p \ l \ f = \sum R- \ p \ l \ g$
unfolding *sum-rsub.simps*
by (*rule sum.cong*, *simp*) (*rule if-cong*, *simp-all add*: *assms*)

Properties and simple lemmas

lemma *RLe-equivalent*: $RL\ c\ p\ b\ q\ l = RLe\ c\ p\ b\ q\ l$
by (*induction q arbitrary*: *c*) (*auto simp add*: *RLe-def R-def RL.simps*(*1*) *RL-simp*)

lemma *SKe-equivalent*: $SK\ c\ p\ b\ q\ k = SKe\ c\ p\ b\ q\ k$
by (*induction q arbitrary*: *c*) (*auto simp add*: *SKe-def S-def SK.simps*(*1*) *S2-def SK-simp*)

lemma *ZLe-equivalent*: $ZL\ c\ p\ b\ q\ l = ZLe\ c\ p\ b\ q\ l$
by (*induction q arbitrary*: *c*) (*auto simp add*: *ZLe-def ZL.simps*(*1*) *R-def Z2-def Z-def ZL-simp*)

lemma *sum-radd-distrib*: $a * (\sum R+ \ p \ l \ f) = (\sum R+ \ p \ l \ (\lambda k.\ a * f\ k))$
by (*auto simp add*: *sum-radd.simps sum-distrib-left*; *smt mult-is-0 sum.cong*)

lemma *sum-rsub-distrib*: $a * (\sum R- \ p \ l \ f) = (\sum R- \ p \ l \ (\lambda k.\ a * f\ k))$
by (*auto simp add*: *sum-rsub.simps sum-distrib-left*; *smt mult-is-0 sum.cong*)

lemma *sum-sadd-distrib*: $a * (\sum S+ \ p \ d \ f) = (\sum S+ \ p \ d \ (\lambda k.\ a * f\ k))$ **for** *a*
by (*auto simp add*: *sum-sadd.simps sum-distrib-left*; *smt mult-is-0 sum.cong*)

lemma *sum-ssub-nzero-distrib*: $a * (\sum S- \ p \ d \ f) = (\sum S- \ p \ d \ (\lambda k.\ a * f\ k))$ **for** *a*
by (*auto simp add*: *sum-ssub-nzero.simps sum-distrib-left*; *smt mult-is-0 sum.cong*)

lemma *sum-ssub-zero-distrib*: $a * (\sum S0 \ p \ d \ f) = (\sum S0 \ p \ d \ (\lambda k.\ a * f\ k))$ **for** *a*
by (*auto simp add*: *sum-ssub-zero.simps sum-distrib-left*; *smt mult-is-0 sum.cong*)

lemma *sum-distrib*:
fixes $SX :: program \Rightarrow nat \Rightarrow (nat \Rightarrow nat) \Rightarrow nat$
and $p :: program$

assumes *SX-simps*: $\bigwedge h.\ SX\ p\ x\ h = (\sum k = 0..length\ p-1.\ if\ g\ x\ k\ then\ h\ k\ else\ 0)$

shows $SX\ p\ x\ h1 + SX\ p\ x\ h2 = SX\ p\ x\ (\lambda k.\ h1\ k + h2\ k)$
by (*subst SX-simps*)+ (*auto simp*: *sum.distrib*[*symmetric*] *intro*: *sum.cong*)

lemma *sum-commutative*:
fixes *SX* :: *program* ⇒ *nat* ⇒ (*nat* ⇒ *nat*) ⇒ *nat*
and *p* :: *program*

assumes *SX-simps*: $\bigwedge h$. *SX p x h* = ($\sum$ *k* = *0..length p*−*1*. *if g x k then h k else 0*)

shows ($\sum$ *t*=*0..q*::*nat*. *SX p x* (λ*k*. *f k t*))
= (*SX p x* (λ*k*. $\sum$ *t*=*0..q*. *f k t*))
proof (*induction q*)
case *0*
then show *?case* **by** (*auto*)
next
case (*Suc q*)
have *SX-add*: *SX p x h1* + *SX p x h2* = *SX p x* (λ*k*. *h1 k* + *h2 k*) **for** *h1 h2*
by (*subst sum-distrib*[**where** *?h1.0* = *h1*]) (*auto simp*: *SX-simps*) +

have *h1*: ($\sum$ *t*≤(*Suc q*). *SX p x* (λ*k*. *f k t*)) = *SX p x* (λ*k*. *f k* (*Suc q*)) + *sum* (λ*t*. *SX p x* (λ*k*. *f k t*)) {*0..q*}
by (*auto simp add*: *sum.atLeast0-atMost-Suc add.commute atMost-atLeast0*)
also have *h2*: ... = *SX p x* (λ*k*. *f k* (*Suc q*)) + *SX p x* (λ*k*. *sum* (*f k*){*0..q*})
using *Suc.IH Suc.prems* **by** *auto*
also have *h3*: ... = *SX p x* (λ*k*. *sum* (*f k*) {*0..*(*Suc q*)})
by (*subst SX-add*) (*auto simp*: *atLeast0-atMost-Suc*)
finally show *?case* **using** *Suc.IH* **by** (*simp add*: *atMost-atLeast0*)
qed

lemma *sum-radd-commutative*: ($\sum$ *t*=*0..*(*q*::*nat*). $\sum$ *R+ p l* (λ*k*. *f k t*)) = ($\sum$ *R+ p l* (λ*k*. $\sum$ *t*=*0..q*. *f k t*))
by (*rule sum-commutative sum-radd.simps*) +
lemma *sum-rsub-commutative*: ($\sum$ *t*=*0..*(*q*::*nat*). $\sum$ *R*− *p l* (λ*k*. *f k t*)) = ($\sum$ *R*− *p l* (λ*k*. $\sum$ *t*=*0..q*. *f k t*))
by (*rule sum-commutative sum-rsub.simps*) +
lemma *sum-sadd-commutative*: ($\sum$ *t*=*0..*(*q*::*nat*). $\sum$ *S+ p l* (λ*k*. *f k t*)) = ($\sum$ *S+ p l* (λ*k*. $\sum$ *t*=*0..q*. *f k t*))
by (*rule sum-commutative sum-sadd.simps*) +
lemma *sum-ssub-nzero-commutative*: ($\sum$ *t*=*0..*(*q*::*nat*). $\sum$ *S*− *p l* (λ*k*. *f k t*)) = ($\sum$ *S*− *p l* (λ*k*. $\sum$ *t*=*0..q*. *f k t*))
by (*rule sum-commutative sum-ssub-nzero.simps*) +
lemma *sum-ssub-zero-commutative*: ($\sum$ *t*=*0..*(*q*::*nat*). $\sum$ *S0 p l* (λ*k*. *f k t*)) = ($\sum$ *S0 p l* (λ*k*. $\sum$ *t*=*0..q*. *f k t*))
by (*rule sum-commutative sum-ssub-zero.simps*) +

lemma *sum-int*: *c* ≤ *a* + *b* ⟹ *int*(*a* + *b* − *c*) = *int*(*a*) + *int*(*b*) − *int*(*c*)
by (*simp add*: *SMT.int-plus*)

lemma *ZLe-bounded*: *b* > *2* ⟹ *ZLe c p b q l* < *b* ^(*Suc q*)
using *Z-bounded ZLe-def*
proof (*induction q*)

case *0*
then show *?case* **by** (*simp add*: *Z-bounded ZLe-def Z-def*)
next
case (*Suc q*)
have $ZLe\ c\ p\ b\ (Suc\ q)\ l = b\ \hat{}\ (Suc\ q) * Z\ c\ p\ l\ (Suc\ q) + ZLe\ c\ p\ b\ q\ l$
by (*auto simp*: *ZLe-def*)
also have $ZLe\ c\ p\ b\ q\ l < b\ \hat{}\ (Suc\ q)$ **using** *Suc.IH*
by (*auto simp*: *ZLe-def Z-def Suc.prems*(*1*))
also have $b\ \hat{}\ (Suc\ q) * Z\ c\ p\ l\ (Suc\ q) \leq b\ \hat{}\ (Suc\ q)$ **using** *Suc.prems*(*1*)
by (*auto simp*: *Z-def*)
finally have $ZLe\ c\ p\ b\ (Suc\ q)\ l < 2 * b\ \hat{}\ (Suc\ q)$
by *auto*
also have ... $< b\ \hat{}\ Suc\ (Suc\ q)$
using *Suc.prems*(*1*) **by** *auto*
finally show *?case* **by** *simp*
qed

lemma *SKe-bounded*: $b > 2 \Longrightarrow SKe\ c\ p\ b\ q\ k < b\ \hat{}\ (Suc\ q)$
proof (*induction q*)
case *0*
then show *?case* **by** (*auto simp add*: *SKe-def S-bounded S-def*)
next
case (*Suc q*)
have $SKe\ c\ p\ b\ (Suc\ q)\ k = b\ \hat{}\ (Suc\ q) * S\ c\ p\ k\ (Suc\ q) + SKe\ c\ p\ b\ q\ k$
by (*auto simp*: *SKe-def*)
also have $SKe\ c\ p\ b\ q\ k < b\ \hat{}\ (Suc\ q)$ **using** *Suc.IH*
by (*auto simp*: *Suc.prems*(*1*))
also have $b\ \hat{}\ (Suc\ q) * S\ c\ p\ k\ (Suc\ q) \leq b\ \hat{}\ (Suc\ q)$ **using** *Suc.prems*(*1*)
by (*auto simp*: *S-def*)
finally have $SKe\ c\ p\ b\ (Suc\ q)\ k < 2 * b\ \hat{}\ (Suc\ q)$
by *auto*
also have ... $< b\ \hat{}\ Suc\ (Suc\ q)$
using *Suc.prems*(*1*) **by** *auto*
finally show *?case* **by** *simp*
qed

lemma *mult-to-bitAND*:
assumes *cells-bounded*: *cells-bounded ic p c*
and $c > 1$
and $b = B\ c$

shows $(\sum t=0..q.\ b\hat{}t * (Z\ ic\ p\ l\ t * S\ ic\ p\ k\ t))$
$= ZLe\ ic\ p\ b\ q\ l\ \&\&\ SKe\ ic\ p\ b\ q\ k$
proof (*induction q arbitrary*: *ic p c l k*)
case *0*
then show *?case* **using** *S-bounded Z-bounded*
by (*auto simp add*: *SKe-def ZLe-def bitAND-single-bit-mult-equiv*)
next
case (*Suc q*)

have *b4*: $b > 2$ **using** *assms(2−3)* **apply** (*auto simp add*: *B-def*)
by (*metis One-nat-def Suc-less-eq2 lessI numeral-2-eq-2 power-gt1*)

have *ske*: $SKe\ ic\ p\ b\ q\ k < b\ \hat{}\ (Suc\ q)$ **using** *SKe-bounded b4* **by** *auto*
have *zle*: $ZLe\ ic\ p\ b\ q\ l < b\ \hat{}\ (Suc\ q)$ **using** *ZLe-bounded b4* **by** *auto*

have *ih*: $(\sum t = 0..q.\ b\ \hat{}\ t * (Z\ ic\ p\ l\ t * S\ ic\ p\ k\ t)) = ZLe\ ic\ p\ b\ q\ l\ \&\&\ SKe\ ic\ p\ b\ q\ k$
using *Suc.IH* **by** *auto*

have $(\sum t = 0..Suc\ q.\ b\ \hat{}\ t * (Z\ ic\ p\ l\ t * S\ ic\ p\ k\ t))$
$= b\ \hat{}(Suc\ q) * (Z\ ic\ p\ l\ (Suc\ q) * S\ ic\ p\ k\ (Suc\ q)) + (\sum t = 0..q.\ b\ \hat{}\ t * (Z\ ic\ p\ l\ t * S\ ic\ p\ k\ t))$
by (*auto simp*: *sum.atLeast0-atMost-Suc add.commute*)

also have ... $= b\ \hat{}\ (Suc\ q) * (Z\ ic\ p\ l\ (Suc\ q) * S\ ic\ p\ k\ (Suc\ q)) + (ZLe\ ic\ p\ b\ q\ l\ \&\&\ SKe\ ic\ p\ b\ q\ k)$
by (*auto simp add*: *ih*)

also have ... $= b\ \hat{}\ (Suc\ q) * (Z\ ic\ p\ l\ (Suc\ q)\ \&\&\ S\ ic\ p\ k\ (Suc\ q)) + (ZLe\ ic\ p\ b\ q\ l\ \&\&\ SKe\ ic\ p\ b\ q\ k)$
using *bitAND-single-bit-mult-equiv S-bounded Z-bounded* **by** (*auto*)

also have ... $= (b\ \hat{}(Suc\ q) * Z\ ic\ p\ l\ (Suc\ q) + ZLe\ ic\ p\ b\ q\ l)\ \&\&\ (b\ \hat{}\ (Suc\ q) * S\ ic\ p\ k\ (Suc\ q) + SKe\ ic\ p\ b\ q\ k)$
using *bitAND-linear ske zle*
by (*auto*) (*smt B-def assms(3) bitAND-linear mult.commute power-Suc power-mult*)

also have ... $= (ZLe\ ic\ p\ b\ (Suc\ q)\ l\ \&\&\ SKe\ ic\ p\ b\ (Suc\ q)\ k)$
by (*auto simp*: *ZLe-def SKe-def add.commute*)

finally show *?case* **by** *simp*
qed

lemma *sum-bt*:
fixes *b q* :: *nat*
assumes $b > 2$
shows $(\sum t = 0..q.\ b\hat{}t) < b\ \hat{}\ (Suc\ q)$
using *assms*
proof (*induction q*, *auto*)
fix *qb* :: *nat*
assume $sum\ ((\hat{})\ b)\ \{0..qb\} < b * b\ \hat{}\ qb$
then have *f1*: $sum\ ((\hat{})\ b)\ \{0..qb\} < b\ \hat{}\ Suc\ qb$
by *fastforce*
have $b\ \hat{}\ Suc\ qb * 2 < b\ \hat{}\ Suc\ (Suc\ qb)$
using *assms* **by** *force*
then have $2 * b\ \hat{}\ Suc\ qb < b\ \hat{}\ Suc\ (Suc\ qb)$
by *simp*

```
then have b ^ Suc qb + sum ((^) b) {0..qb} < b ^ Suc (Suc qb)
  using f1 by linarith
then show sum ((^) b) {0..qb} + b * b ^ qb < b * (b * b ^ qb)
  by simp
qed

lemma mult-to-bitAND-state:
  assumes cells-bounded: cells-bounded ic p c
and c: c > 1
and b: b = B c

shows (∑ t=0..q. b^t * ((1 − Z ic p l t) * S ic p k t))
       = ((∑ t = 0..q. b^t) − ZLe ic p b q l) && SKe ic p b q k
proof (induction q arbitrary: ic p c l k)
  case 0
  show ?case using Z-def S-def ZLe-def SKe-def by auto
next
  case (Suc q)

  have b4: b > 2 using assms(2−3) apply (auto simp add: B-def)
    by (metis One-nat-def Suc-less-eq2 lessI numeral-2-eq-2 power-gt1)

  have ske: SKe ic p b q k < b ^(Suc q) using SKe-bounded b4 by auto
  have zle: ZLe ic p b q l < b ^(Suc q) using ZLe-bounded b4 by auto
  define cst where cst ≡ Suc q
  define e where e ≡ ∑ t = 0..Suc q. b^t

  have (∑ t = 0..q. b^t) < b ^(Suc q)
    using sum-bt b4 by auto
  hence zle2: (∑ t = 0..q. b^t) − ZLe ic p b q l < b ^(Suc q)
    using less-imp-diff-less by blast

  have (∑ t = 0..x. b^t) − ZLe ic p b x l = (∑ t=0..x. b^t − b^t * Z ic p l t)
for x
    unfolding ZLe-def
    using Z-bounded sum-subtractf-nat[where ?f = (^) b and ?g = λt. b ^ t * Z
ic p l t]
    by auto
  hence aux-sum: (∑ t = 0..x. b^t) − ZLe ic p b x l = (∑ t=0..x. b^t * (1 − Z
ic p l t)) for x
    using diff-Suc-1 diff-mult-distrib2 by auto

  have aux1: b ^(Suc q) * (1 − Z ic p l (Suc q)) + (∑ t=0..q. b^t * (1 − Z ic p
l t))
                 = (∑ t = 0..cst. b ^ t * (1 − Z ic p l t))
    by (auto simp: sum.atLeast0-atMost-Suc cst-def)
  also have aux2: ... = (∑ t = 0..cst. b ^ t) − ZLe ic p b cst l
    unfolding e-def ZLe-def using aux-sum[of cst]
    by (auto simp: ZLe-def)
```

finally have *aux-add-sub*:
$(b \hat{}(Suc\ q) * (1 - Z\ ic\ p\ l\ (Suc\ q)) + ((\sum t = 0..q.\ b\hat{}t) - ZLe\ ic\ p\ b\ q\ l))$
$= (e - ZLe\ ic\ p\ b\ (Suc\ q)\ l)$
by (*auto simp*: *cst-def e-def aux-sum*)

hence *ih*: $(\sum t = 0..q.\ b \hat{}\ t * ((1 - Z\ ic\ p\ l\ t) * S\ ic\ p\ k\ t))$
$= (\sum t = 0..q.\ b\hat{}t) - ZLe\ ic\ p\ b\ q\ l\ \&\&\ SKe\ ic\ p\ b\ q\ k$
using *Suc*[*of ic p l k*] **by** *auto*

have $(\sum t = 0..Suc\ q.\ b \hat{}\ t * ((1 - Z\ ic\ p\ l\ t) * S\ ic\ p\ k\ t))$
$= (\sum t = 0..\quad q.\ b \hat{}\ t * ((1 - Z\ ic\ p\ l\ t) * S\ ic\ p\ k\ t))$
$+ b\hat{}(Suc\ q) * ((1 - Z\ ic\ p\ l\ (Suc\ q)) * S\ ic\ p\ k\ (Suc\ q))$
by (*auto cong*: *sum.cong*)

also have ... $= ((\sum t = 0..q.\ b\hat{}t) - ZLe\ ic\ p\ b\ q\ l\ \&\&\ SKe\ ic\ p\ b\ q\ k)$
$+ b\hat{}(Suc\ q) * ((1 - Z\ ic\ p\ l\ (Suc\ q)) * S\ ic\ p\ k\ (Suc\ q))$
using *ih* **by** *auto*

also have ... $= ((\sum t = 0..q.\ b\hat{}t) - ZLe\ ic\ p\ b\ q\ l\ \&\&\ SKe\ ic\ p\ b\ q\ k)$
$+\ b\hat{}(Suc\ q) * ((1 - Z\ ic\ p\ l\ (Suc\ q))\ \&\&\ S\ ic\ p\ k\ (Suc\ q))$
using *bitAND-single-bit-mult-equiv* **by** (*simp add*: *S-def*)

also have ... $= (b \hat{}(Suc\ q) * (1 - Z\ ic\ p\ l\ (Suc\ q)) + ((\sum t = 0..q.\ b\hat{}t) - ZLe$
$ic\ p\ b\ q\ l))$
$\&\&\ (b \hat{}\ (Suc\ q) * S\ ic\ p\ k\ (Suc\ q) + SKe\ ic\ p\ b\ q\ k)$
using *bitAND-linear ske zle2 B-def b*
by (*smt add-ac*(*2*) *mult-ac*(*2*) *bitAND-linear power.simps*(*2*) *power-mult power-mult-distrib*)
also have ... $= (e - ZLe\ ic\ p\ b\ (Suc\ q)\ l\ \&\&\ SKe\ ic\ p\ b\ (Suc\ q)\ k)$
using *SKe-def aux-add-sub* **by** (*auto simp*: *add.commute*)

finally show *?case* **by** (*auto simp*: *e-def*)
qed

end

3.4 Single step relations

3.4.1 Registers

theory *SingleStepRegister*
imports *RegisterMachineSimulation*
begin

lemma *single-step-add*:
fixes *c* :: *configuration*
and *p* :: *program*
and *l* :: *register*
and *t a* :: *nat*

defines *cs* ≡ *fst* (*steps c p t*)

assumes *is-val*: *is-valid-initial c p a*
  and *l*: $l <$ *length tape*

shows ($\sum$ *R+ p l* (λk. *S c p k t*))
  = (*if isadd* (*p*!*cs*) $\wedge$ *l* = *modifies* (*p*!*cs*) *then 1 else 0*)
proof −
 have *ic*: *c* = (*0*, *snd c*)
  using *is-val* **by** (*auto simp add*: *is-valid-initial-def*) (*metis prod.collapse*)

 have *add-if*: ($\sum$ *k* = *0*..*length p*−*1*. *if isadd* (*p* ! *k*) $\wedge$ *modifies* (*p* ! *cs*) = *modifies* (*p* ! *k*)
  then S c p k t else 0)
  = ($\sum$ *k* = *0*..*length p*−*1*. *if k*=*cs then*
  if isadd (*p* ! *k*) $\wedge$ *modifies* (*p* ! *cs*) = *modifies* (*p* ! *k*) *then S c p k t else 0 else 0*)
  apply (*rule sum.cong*)
  using *S-unique cs-def* **by** *auto*

 have *bound*: *fst* (*steps c p t*) $\leq$ *length p* − *1* **using** *is-val ic p-contains*[*of c p a t*]
  by (*auto simp add*: *dual-order.strict-implies-order*)

 thus *?thesis* **using** *S-unique add-if*
  apply (*auto simp add*: *sum-radd.simps add-if S-def cs-def*)
  by (*smt S-def sum.cong*)
qed

lemma *single-step-sub*:
 fixes *c* :: *configuration*
  and *p* :: *program*
  and *l* :: *register*
  and *t a* :: *nat*

defines *cs* $\equiv$ *fst* (*steps c p t*)

assumes *is-val*: *is-valid-initial c p a*

shows ($\sum$ *R*− *p l* (λk. *Z c p l t* ∗ *S c p k t*))
  = (*if issub* (*p*!*cs*) $\wedge$ *l* = *modifies* (*p*!*cs*) *then Z c p l t else 0*)
proof −
 have *fst c* = *0* **using** *is-val* **by** (*auto simp add*: *is-valid-initial-def*)
 hence *ic*: *c* = (*0*, *snd c*) **by** (*metis prod.collapse*)

 have *bound*: *cs* $\leq$ *length p* − *1* **using** *is-val ic p-contains*[*of c p a t*]
  by (*auto simp add*: *dual-order.strict-implies-order cs-def*)

 have *sub-if*: ($\sum$ *k* = *0*..*length p*−*1*. *if issub* (*p* ! *k*) $\wedge$ *modifies* (*p* ! *cs*) = *modifies* (*p* ! *k*)

then 1 ∗ (*if cs* = *k then* (*Suc 0*) *else 0*) *else 0*)
=(∑ *k* = *0*..*length p*−*1*. *if k* = *cs then*
(*if issub* (*p* ! *k*) ∧ *modifies* (*p* ! *cs*) = *modifies* (*p* ! *k*)
then (*Suc 0*) ∗ (*if cs* = *k then* (*Suc 0*) *else 0*)
else 0) *else 0*)
apply (*rule sum.cong*) **using** *cs-def* **by** *auto*

show *?thesis* **using** *bound sub-if*
apply (*auto simp add*: *sum-rsub.simps cs-def Z-def S-def R-def*)
by (*metis One-nat-def cs-def*)
qed

lemma *lm04-06-one-step-relation-register-old*:
fixes *l*::*register*
and *ic*::*configuration*
and *p*::*program*

defines *s* ≡ *fst ic*
and *tape* ≡ *snd ic*

defines *m* ≡ *length p*
and *tape′* ≡ *snd* (*step ic p*)

assumes *is-val*: *is-valid ic p*
and *l*: ‹*l* < *length tape*›

shows (*tape′*!*l*) = (*tape*!*l*) + (*if isadd* (*p*!*s*) ∧ *l* = *modifies* (*p*!*s*) *then 1 else 0*)
− *Z ic p l 0* ∗ (*if issub* (*p*!*s*) ∧ *l* = *modifies* (*p*!*s*) *then 1*
else 0)
proof −
show *?thesis*
using *l*
apply (*cases* ‹*p*!*s*›)
apply (*auto simp*: *assms*(*1*−*4*) *step-def update-def*)
using *nth-digit-0* **by** (*auto simp add*: *Z-def R-def*)
qed

lemma *lm04-06-one-step-relation-register*:
fixes *l* :: *register*
and *c* :: *configuration*
and *p* :: *program*
and *t* :: *nat*
and *a* :: *nat*

defines *r* ≡ *R c p*
defines *s* ≡ *S c p*

assumes *is-val*: *is-valid-initial c p a*

and *l*: *l* < *length* (*snd c*)

shows *r l* (*Suc t*) = *r l t* + (∑ *R*+ *p l* (λ*k*. *s k t*))
− (∑ *R*− *p l* (λ*k*. (*Z c p l t*) ∗ *s k t*))
proof −
define *cs* **where** *cs* ≡ *fst* (*steps c p t*)

have *add*: (∑ *R*+ *p l* (λ*k*. *s k t*))
= (*if isadd* (*p*!*cs*) ∧ *l* = *modifies* (*p*!*cs*) *then 1 else 0*)
using *single-step-add*[*of c p a l snd c t*] *is-val l s-def cs-def* **by** *auto*

have *sub*: (∑ *R*− *p l* (λ*k*. *Z c p l t* ∗ *s k t*))
= (*if issub* (*p*!*cs*) ∧ *l* = *modifies* (*p*!*cs*) *then Z c p l t else 0*)
using *single-step-sub is-val l s-def cs-def Z-def R-def* **by** *auto*

have *lhs*: *r l* (*Suc t*) = *snd* (*steps c p* (*Suc t*)) ! *l*
by (*simp add*: *r-def R-def del*: *steps.simps*)

have *rhs*: *r l t* = *snd* (*steps c p t*) ! *l*
by (*simp add*: *r-def R-def del*: *steps.simps*)

have *valid-time*: *is-valid* (*steps c p t*) *p* **using** *steps-is-valid-invar is-val*
by (*auto simp add*: *is-valid-initial-def*)

have *l-time*: *l* < *length* (*snd* (*steps c p t*)) **using** *l steps-tape-length-invar* **by** *auto*

from *lhs rhs* **have** *r l* (*Suc t*) = *r l t* + (*if isadd* (*p*!*cs*) ∧ *l* = *modifies* (*p*!*cs*) *then 1 else 0*)
− (*if issub* (*p*!*cs*) ∧ *l* = *modifies* (*p*!*cs*) *then Z c p l t else 0*)
using *l-time valid-time lm04-06-one-step-relation-register-old steps.simps cs-def nth-digit-0*
Z-def R-def **by** *auto*

thus *?thesis* **using** *add sub* **by** *simp*
qed

end

3.4.2 States

theory *SingleStepState*
imports *RegisterMachineSimulation*
begin

lemma *lm04-07-one-step-relation-state*:
fixes *d* :: *state*
and *c* :: *configuration*
and *p* :: *program*

and *t* :: *nat*
and *a* :: *nat*

defines $r \equiv R\ c\ p$
defines $s \equiv S\ c\ p$
defines $z \equiv Z\ c\ p$
defines $cs \equiv fst\ (steps\ c\ p\ t)$

assumes *is-val*: *is-valid-initial c p a*
and $d < length\ p$

shows $s\ d\ (Suc\ t) = (\sum S{+}\ p\ d\ (\lambda k.\ s\ k\ t))$
$+ (\sum S{-}\ p\ d\ (\lambda k.\ z\ (modifies\ (p!k))\ t * s\ k\ t))$
$+ (\sum S0\ p\ d\ (\lambda k.\ (1 - z\ (modifies\ (p!k))\ t) * s\ k\ t))$
$+ (if\ ishalt\ (p!cs) \wedge d = cs\ then\ Suc\ 0\ else\ 0)$
proof −
have *ic*: $c = (0,\ snd\ c)$
using *is-val* **by** (*auto simp add*: *is-valid-initial-def*) (*metis prod.collapse*)
have *cs-bound*: $cs < length\ p$ **using** *ic is-val p-contains*[*of c p a t*] *cs-def* **by** *auto*

have $(\sum k = 0..length\ p-1.$
$if\ isadd\ (p\ !\ k) \wedge goes\text{-}to\ (p\ !\ fst\ (steps\ c\ p\ t)) = goes\text{-}to\ (p\ !\ k)$
$then\ if\ fst\ (steps\ c\ p\ t) = k$
$then\ Suc\ 0\ else\ 0\ else\ 0)$
$=(\sum k = 0..length\ p-1.$
$if\ fst\ (steps\ c\ p\ t) = k$
$then\ if\ isadd\ (p\ !\ k) \wedge goes\text{-}to\ (p\ !\ fst\ (steps\ c\ p\ t)) = goes\text{-}to$
$(p\ !\ k)$
$then\ Suc\ 0\ else\ 0\ else\ 0)$
apply (*rule sum.cong*) **by** *auto*
hence *add*: $(\sum S{+}\ p\ d\ (\lambda k.\ s\ k\ t)) = (if\ isadd\ (p!cs) \wedge d = goes\text{-}to\ (p!cs)\ then$
$Suc\ 0\ else\ 0)$
apply (*auto simp add*: *sum-sadd.simps s-def S-def cs-def*)
using *cs-bound cs-def* **by** *auto*

have $(\sum k = 0..length\ p-1.$
$if\ issub\ (p\ !\ k) \wedge goes\text{-}to\ (p\ !\ fst\ (steps\ c\ p\ t)) = goes\text{-}to\ (p\ !\ k)$
$then\ z\ (modifies\ (p\ !\ k))\ t * (if\ fst\ (steps\ c\ p\ t) = k\ then\ Suc\ 0\ else$
$0)\ else\ 0)$
$= (\sum k = 0..length\ p-1.\ if\ k{=}cs\ then$
$if\ issub\ (p\ !\ k) \wedge goes\text{-}to\ (p\ !\ fst\ (steps\ c\ p\ t)) = goes\text{-}to\ (p\ !\ k)$
$then\ z\ (modifies\ (p\ !\ k))\ t\ \ else\ 0\ else\ 0)$
apply (*rule sum.cong*)
using *z-def Z-def cs-def* **by** *auto*
hence *sub-zero*: $(\sum S{-}\ p\ d\ (\lambda k.\ z\ (modifies\ (p!k))\ t * s\ k\ t))$
$= (if\ issub\ (p!cs) \wedge d = goes\text{-}to\ (p!cs)\ then\ z\ (modifies\ (p!cs))\ t\ else\ 0)$
apply (*auto simp add*: *sum-ssub-nzero.simps s-def S-def cs-def*)
using *cs-bound cs-def* **by** *auto*

have (∑ *k* = *0*..*length p*−*1*.
if issub (*p* ! *k*) ∧ *goes-to-alt* (*p* ! *fst* (*steps c p t*)) = *goes-to-alt* (*p* ! *k*)
then (*Suc 0* − *z* (*modifies* (*p* ! *k*)) *t*) ∗ (*if fst* (*steps c p t*) = *k then Suc 0 else 0*) *else 0*)
= (∑ *k* = *0*..*length p*−*1*. *if k*=*cs then*
if issub (*p* ! *k*) ∧ *goes-to-alt* (*p* ! *fst* (*steps c p t*)) = *goes-to-alt* (*p* ! *k*)
then (*Suc 0* − *z* (*modifies* (*p* ! *k*)) *t*) *else 0 else 0*)
apply (*rule sum.cong*) **using** *z-def Z-def cs-def* **by** *auto*
hence *sub-nzero*: (∑ *S0 p d* (λ*k*. (*1* − *z* (*modifies* (*p*!*k*)) *t*) ∗ *s k t*))
= (*if issub* (*p*!*cs*) ∧ *d* = *goes-to-alt* (*p*!*cs*) *then* (*1* − *z* (*modifies* (*p*!*cs*)) *t*) *else 0*)
apply (*auto simp*: *sum-ssub-zero.simps s-def S-def cs-def*)
using *cs-bound cs-def* **by** *auto*

have *s d* (*Suc t*) = (*if isadd* (*p*!*cs*) ∧ *d* = *goes-to* (*p*!*cs*) *then Suc 0 else 0*)
+ (*if issub* (*p*!*cs*) ∧ *d* = *goes-to* (*p*!*cs*) *then z* (*modifies* (*p*!*cs*)) *t else 0*)
+ (*if issub* (*p*!*cs*) ∧ *d* = *goes-to-alt* (*p*!*cs*) *then* (*1* − *z* (*modifies* (*p*!*cs*)) *t*) *else 0*)
+ (*if ishalt* (*p*!*cs*) ∧ *d* = *cs then Suc 0 else 0*)
apply (*cases p*!*cs*)
by (*auto simp*: *s-def S-def step-def fetch-def cs-def z-def Z-def Z-bounded R-def read-def*)

thus *?thesis* **using** *add sub-zero sub-nzero* **by** *auto*
qed

end

3.5 Multiple step relations

3.5.1 Registers

theory *MultipleStepRegister*
imports *SingleStepRegister*
begin

lemma *lm04-22-multiple-register*:
fixes *c* :: *nat*
and *l* :: *register*
and *ic* :: *configuration*
and *p* :: *program*
and *q* :: *nat*
and *a* :: *nat*

defines *b* == *B c*
and *m* == *length p*
and *n* == *length* (*snd ic*)

assumes *is-val*: *is-valid-initial ic p a*

```
assumes c-gt-cells: cells-bounded ic p c
assumes l: l < n
and 0 < l
and q: q > 0

assumes terminate: terminates ic p q

assumes c: c > 1

  defines r == RLe ic p b q
      and z == ZLe ic p b q
      and s == SKe ic p b q

shows r l = b ∗ r l
          + b ∗ (∑ R+ p l s)
          − b ∗ (∑ R− p l (λk. z l && s k))
proof −
 have 0: snd ic ! l = 0 using assms(4, 7) by (cases ic; auto simp add: is-valid-initial-def)

  have b^(Suc t) ∗ (∑ R− p l (λk. (Z ic p l t) ∗ S ic p k t)) ≤ b^(Suc t) ∗  R ic
p l t  for t
     proof (cases t=0)
       case True
       hence R ic p l 0 = 0 by (auto simp add: 0 R-def)
       thus ?thesis by (auto simp add: True Z-def sum-rsub.simps)
     next
       case False
       define cs where cs ≡ fst (steps ic p t)
       have sub: (∑ R− p l (λk. Z ic p l t ∗ S ic p k t))
        = (if issub (p!cs) ∧ l = modifies (p!cs) then Z ic p l t else 0)
         using single-step-sub Z-def R-def is-val l n-def cs-def by auto
       show ?thesis using sub by (auto simp add: sum-rsub.simps R-def Z-def)
     qed

    from this have positive: b^(Suc t) ∗ (∑ R− p l (λk. (Z ic p l t) ∗ S ic p k t))
                   ≤ b^(Suc t) ∗  R ic p l t
                   + b^(Suc t) ∗ (∑ R+ p l (λk. S ic p k t)) for t
      by (auto simp add: Nat.trans-le-add1)

  have commute-add: (∑ t=0..q−1. ∑ R+ p l (λk. b^t ∗ S ic p k t))
      = ∑ R+ p l (λk. ∑ t=0..q−1. (b^t ∗ S ic p k t))
    using sum-radd-commutative[of p l λk t. b^t ∗ S ic p k t  q−1] by auto

  have r-q: l<n ⟶ R ic p l q = 0
    using terminate terminates-def correct-halt-def by (auto simp: n-def R-def)
  hence z-q: l<n ⟶ Z ic p l q = 0
    using terminate terminates-def correct-halt-def by (auto simp: Z-def)
  have ∀ k<length p−1. ¬ ishalt (p!k)
    using is-val is-valid-initial-def[of ic p a] is-valid-def[of ic p]
```

program-includes-halt.simps **by** *blast*
hence *s-q*: $\forall k < length\ p - 1.\ S\ ic\ p\ k\ q = 0$
using *terminate terminates-def correct-halt-def S-def* **by** *auto*

from *r-q* **have** *rq*: $(\sum x = 0..q - 1.\ int\ b\ \hat{}\ x * int\ (snd\ (steps\ ic\ p\ x)\ !\ l)) =$
$(\sum x = 0..q.\ int\ b\ \hat{}\ x * int\ (snd\ (steps\ ic\ p\ x)\ !\ l))$
by (*auto simp*: *r-q R-def l*;
smt Suc-pred mult-0-right of-nat-0 of-nat-mult power-mult-distrib q sum.atLeast0-atMost-Suc zero-power)

have $(\sum t = 0..q - 1.\ b\ \hat{}\ t * (Z\ ic\ p\ l\ t * S\ ic\ p\ k\ t))$
$+ (b\hat{}(Suc\ (q-1)) * (Z\ ic\ p\ l\ (Suc\ (q-1)) * S\ ic\ p\ k\ (Suc\ (q-1))))$
$= (\sum t = 0..Suc\ (q-1).\ b\ \hat{}\ t * (Z\ ic\ p\ l\ t * S\ ic\ p\ k\ t))$ **for** *k*
using *comm-monoid-add-class.sum.atLeast0-atMost-Suc* **by** *auto*
hence *zq*: $(\sum t = 0..q - 1.\ b\ \hat{}\ t * (Z\ ic\ p\ l\ t * S\ ic\ p\ k\ t))$
$= (\sum t = 0..q.\ b\ \hat{}\ t * (Z\ ic\ p\ l\ t * S\ ic\ p\ k\ t))$ **for** *k*
using *z-q q l* **by** *auto*

have (*if isadd* $(p\ !\ k) \wedge l =$ *modifies* $(p\ !\ k)$ *then* $\sum t = 0..q - Suc\ 0.\ b\ \hat{}\ t * S\ ic\ p\ k\ t$ *else 0*)
= (*if isadd* $(p\ !\ k) \wedge l =$ *modifies* $(p\ !\ k)$ *then* $\sum t = 0..q.\ b\ \hat{}\ t * S\ ic\ p\ k\ t$ *else 0*) **for** *k*
proof (*cases p*!*k*)
case (*Add x11 x12*)
have *sep*: $(\sum t = 0..q-1.\ b\ \hat{}\ t * S\ ic\ p\ k\ t) + b\hat{}q * S\ ic\ p\ k\ q$
$= (\sum t = 0..(Suc\ (q-1)).\ b\ \hat{}\ t * S\ ic\ p\ k\ t)$
using *comm-monoid-add-class.sum.atLeast0-atMost-Suc*[*of* $\lambda t.\ b\hat{}t * S\ ic\ p\ k\ t\ q-1$] *q*
by *auto*
have *ishalt* (*p* ! (*fst* (*steps ic p q*)))
using *terminates-halt-state*[*of ic p*] *is-val terminate* **by** *auto*
hence $S\ ic\ p\ k\ q = 0$ **using** *Add S-def*[*of ic p k q*] **by** *auto*
with *sep q* **have** $(\sum t = 0..q - Suc\ 0.\ b\ \hat{}\ t * S\ ic\ p\ k\ t) = (\sum t = 0..q.\ b\ \hat{}\ t * S\ ic\ p\ k\ t)$
by *auto*
thus *?thesis* **by** *auto*
next
case (*Sub x21 x22 x23*)
then show *?thesis* **by** *auto*
next
case *Halt*
then show *?thesis* **by** *auto*
qed

hence *add-q*: $\sum R+\ p\ l\ (\lambda k.\ \sum t=0..(q-1).\ b\hat{}t * S\ ic\ p\ k\ t)$
$= \sum R+\ p\ l\ (\lambda k.\ \sum t=0..q.\ b\hat{}t * S\ ic\ p\ k\ t)$
using *sum-radd.simps single-step-add*[*of ic p a l snd ic*] *is-val l n-def* **by** *auto*

have *r l* = (∑ *t* = *0*..*q*. *b*^*t* ∗ *R ic p l t*) **using** *r-def RLe-def* **by** *auto*
also have ... = *R ic p l 0* + (∑ *t* = *1*..*q*. *b*^*t* ∗ *R ic p l t*)
by (*auto simp*: *q comm-monoid-add-class.sum.atLeast-Suc-atMost*)
also have ... = (∑ *t* ∈ {*1*..*q*}. *b*^*t* ∗ *R ic p l t*)
by (*simp add*: *R-def 0*)
also have ... = (∑ *t* ∈ (*Suc* ‘ {*0*..(*q*−*1*)}). *b*^*t* ∗ *R ic p l t*) **using** *q* **by** *auto*
also have ... = (*sum* ((λ*t*. *b*^*t* ∗ *R ic p l t*) ∘ *Suc*)) {*0*..(*q*−*1*)}
using *comm-monoid-add-class.sum.reindex*[*of Suc* {*0*..(*q*−*1*)} (λ*t*. *b*^*t* ∗ *R ic p l t*)] **by** *auto*
also have ... = (∑ *t* = *0*..(*q*−*1*). *b*^(*Suc t*) ∗(*R ic p l t*
+ (∑ *R*+ *p l* (λ*k*. *S ic p k t*))
− (∑ *R*− *p l* (λ*k*. (*Z ic p l t*) ∗ *S ic p k t*))))
using *lm04-06-one-step-relation-register*[*of ic p a l*] *is-val l*
by (*simp add*: *n-def m-def*)

also have ... = (∑ *t* ∈ {*0*..(*q*−*1*)}. *b*^(*Suc t*) ∗ *R ic p l t*
+ *b*^(*Suc t*) ∗ (∑ *R*+ *p l* (λ*k*. *S ic p k t*))
− *b*^(*Suc t*) ∗ (∑ *R*− *p l* (λ*k*. (*Z ic p l t*) ∗ *S ic p k t*)))
by (*auto simp add*: *algebra-simps*)

finally have *int* (*r l*) = (∑ *t* ∈ {*0*..(*q*−*1*)}. *int*(
b^(*Suc t*) ∗ *R ic p l t*
+ *b*^(*Suc t*) ∗ (∑ *R*+ *p l* (λ*k*. *S ic p k t*))
− *b*^(*Suc t*) ∗ (∑ *R*− *p l* (λ*k*. (*Z ic p l t*) ∗ *S ic p k t*))))
by *auto*

also have ... = (∑ *t* ∈ {*0*..(*q*−*1*)}. *int* (*b*^(*Suc t*) ∗ *R ic p l t*)
+ *int* (*b*^(*Suc t*) ∗ (∑ *R*+ *p l* (λ*k*. *S ic p k t*)))
− *int* (*b*^(*Suc t*) ∗ (∑ *R*− *p l* (λ*k*. (*Z ic p l t*) ∗ *S ic p k t*))))
by (*simp only*: *sum-int positive*)

also have ... = (∑ *t* ∈ {*0*..(*q*−*1*)}. *int* (*b*^(*Suc t*) ∗ *R ic p l t*)
+ (*int* (*b*^(*Suc t*) ∗ (∑ *R*+ *p l* (λ*k*. *S ic p k t*)))
− *int* (*b*^(*Suc t*) ∗ (∑ *R*− *p l* (λ*k*. (*Z ic p l t*) ∗ *S ic p k t*)))))
by (*simp add*: *add-diff-eq*)

also have ... = (∑ *t* ∈ {*0*..(*q*−*1*)}. *int*(*b*^(*Suc t*) ∗ *R ic p l t*))
+ (∑ *t* ∈ {*0*..(*q*−*1*)}. *int*(*b*^(*Suc t*) ∗ (∑ *R*+ *p l* (λ*k*. *S ic p k t*)))
− *int*(*b*^(*Suc t*) ∗ (∑ *R*− *p l* (λ*k*. (*Z ic p l t*) ∗ *S ic p k t*))))
by (*auto simp only*: *sum.distrib*)

also have ... = *int b* ∗ *int* (∑ *t* ∈ {*0*..(*q*−*1*)}. *b*^*t* ∗ *R ic p l t*)
+ *int b* ∗ (∑ *t* ∈ {*0*..(*q*−*1*)}. *int*(*b*^*t* ∗ (∑ *R*+ *p l* (λ*k*. *S ic p k t*)))
− *int*(*b*^*t* ∗ (∑ *R*− *p l* (λ*k*. (*Z ic p l t*) ∗ *S ic p k t*))))
by (*auto simp*: *sum-distrib-left mult.assoc right-diff-distrib*)

also have ... = *int b ∗ int* (∑ *t* ∈ {*0*..(*q*−*1*)}. *b*^*t* ∗ *R ic p l t*)
+ *int b* ∗ (∑ *t* ∈ {*0*..(*q*−*1*)}. *int*(*b*^*t* ∗ (∑ *R*+ *p l* (λ*k*. *S ic p k t*))))
− *int b* ∗ (∑ *t* ∈ {*0*..(*q*−*1*)}. *int*(*b*^*t* ∗ (∑ *R*− *p l* (λ*k*. (*Z ic p l t*) ∗ *S ic p k t*))))
by (*auto simp add*: *sum.distrib int-distrib*(*4*) *sum-subtractf*)

also have ... = *int b ∗ int* (∑ *t* ∈ {*0*..(*q*−*1*)}. *b*^*t* ∗ *R ic p l t*)
+ *int b* ∗ (∑ *t* ∈ {*0*..(*q*−*1*)}. *int*(∑ *R*+ *p l* (λ*k*. *b*^*t* ∗ *S ic p k t*)))
− *int b* ∗ (∑ *t* ∈ {*0*..(*q*−*1*)}. *int*(∑ *R*− *p l* (λ*k*. *b*^*t* ∗ (*Z ic p l t* ∗ *S ic p k t*))))
using *sum-radd-distrib sum-rsub-distrib* **by** *auto*

also have ... = *int b ∗ int* (∑ *t* = *0*..*q*−*1*. *b*^*t* ∗ *R ic p l t*)
+ *int b ∗ int* (∑ *t* = *0*..*q*−*1*. ∑ *R*+ *p l* (λ*k*. *b*^*t* ∗ *S ic p k t*))
− *int b ∗ int* (∑ *t* = *0*..*q*−*1*. ∑ *R*− *p l* (λ*k*. *b*^*t* ∗ (*Z ic p l t* ∗ *S ic p k t*)))
by *auto*

also have ... = *int b ∗ int* (∑ *t* = *0*..*q*−*1*. *b*^*t* ∗ *R ic p l t*)
+ *int b ∗ int* (∑ *R*+ *p l* (λ*k*. ∑ *t*=*0*..*q*−*1*. *b*^*t* ∗ *S ic p k t*))
− *int b ∗ int* (∑ *R*− *p l* (λ*k*. ∑ *t*=*0*..*q*−*1*. *b*^*t* ∗ (*Z ic p l t* ∗ *S ic p k t*)))
using *sum-rsub-commutative*[*of p l* λ*k t*. *b*^*t* ∗ (*Z ic p l t* ∗ *S ic p k t*) *q*−*1*]
using *sum-radd-commutative*[*of p l* λ*k t*. *b*^*t* ∗ *S ic p k t* *q*−*1*] **by** *auto*

also have ... = *int b ∗ int* (∑ *t*=*0*..*q*. *b*^*t* ∗ *R ic p l t*)
+ *int b ∗ int* (∑ *R*+ *p l* (λ*k*. ∑ *t*=*0*..*q*−*1*. *b*^*t* ∗ *S ic p k t*))
− *int b ∗ int* (∑ *R*− *p l* (λ*k*. ∑ *t*=*0*..*q*−*1*. *b*^*t* ∗ (*Z ic p l t* ∗ *S ic p k t*)))
by (*auto simp*: *rq R-def*; *smt One-nat-def rq*)

also have ... = *int b ∗ int* (∑ *t*=*0*..*q*. *b*^*t* ∗ *R ic p l t*)
+ *int b ∗ int* (∑ *R*+ *p l* (λ*k*. ∑ *t*=*0*..*q*. *b*^*t* ∗ *S ic p k t*))
− *int b ∗ int* (∑ *R*− *p l* (λ*k*. ∑ *t*=*0*..*q*. *b*^*t* ∗ (*Z ic p l t* ∗ *S ic p k t*)))
using *zq add-q* **by** *auto*

also have ... = *int b ∗ int* (*RLe ic p b q l*)
+ *int b ∗ int* (∑ *R*+ *p l* (*SKe ic p b q*))
− *int b ∗ int* (∑ *R*− *p l* (λ*k*. ∑ *t*=*0*..*q*. *b*^*t* ∗ (*Z ic p l t* ∗ *S ic p k t*)))
by (*auto simp*: *RLe-def*; *metis SKe-def*)

also have ... = *int b ∗ int* (*RLe ic p b q l*)
+ *int b ∗ int* (∑ *R*+ *p l* (*SKe ic p b q*))
− *int b ∗ int* (∑ *R*− *p l* (λ*k*. *ZLe ic p b q l* && *SKe ic p b q k*))
using *mult-to-bitAND c-gt-cells b-def c* **by** *auto*

finally have *int*(*r l*) = *int b ∗ int* (*r l*)

+ *int* b ∗ *int* ($\sum$ *R+ p l s*)
− *int* b ∗ *int* ($\sum$ *R− p l* (λ*k*. *z l* && *s k*))
by (*auto simp*: *r-def s-def z-def*)

hence *r l* = *b* ∗ *r l*
+ *b* ∗ $\sum$ *R+ p l s*
− *b* ∗ $\sum$ *R− p l* (λ*k*. *z l* && *s k*)
using *int-ops*(*5*) *int-ops*(*7*) *nat-int nat-minus-as-int* **by** *presburger*

thus *?thesis* **by** *simp*
qed

lemma *lm04-23-multiple-register1*:
fixes *c* :: *nat*
and *l* :: *register*
and *ic* :: *configuration*
and *p* :: *program*
and *q* :: *nat*
and *a* :: *nat*

defines *b* == *B c*
and *m* == *length p*
and *n* == *length* (*snd ic*)

assumes *is-val*: *is-valid-initial ic p a*
assumes *c-gt-cells*: *cells-bounded ic p c*
assumes *l*: *l* = *0*
and *q*: *q* > *0*

assumes *c*: *c* > *1*

assumes *terminate*: *terminates ic p q*

defines *r* == *RLe ic p b q*
and *z* == *ZLe ic p b q*
and *s* == *SKe ic p b q*

shows *r l* = *a* + *b* ∗ *r l*
+ *b* ∗ ($\sum$ *R+ p l s*)
− *b* ∗ ($\sum$ *R− p l* (λ*k*. *z l* && *s k*))
proof −
have *n*: *n* > *0* **using** *is-val*
by (*auto simp add*: *is-valid-initial-def n-def*)

have *0*: *snd ic* ! *l* = *a*
using *assms* **by** (*cases ic*; *auto simp add*: *is-valid-initial-def List.hd-conv-nth*)

find-theorems *hd ?l* = *?l* ! *0*

have *bound-fst-ic*: (*if fst ic* ≤ *length p*−*1 then 1 else 0*) ≤ *Suc 0* **by** *auto*
have (*if issub* (*p* ! *k*) ∧ *l* = *modifies* (*p* ! *k*) *then if fst ic* = *k then 1 else 0 else 0*)
= (*if k* = *fst ic* ∧ *issub* (*p* ! *k*) ∧ *l* = *modifies* (*p* ! *k*) *then 1 else 0*) **for** *k* **by** *auto*
hence (*if issub* (*p* ! *k*) ∧ *l* = *modifies* (*p* ! *k*) *then if fst ic* = *k then Suc 0 else 0 else 0*)
≤ (*if k* = *fst ic then 1 else 0*) **for** *k*
apply (*cases p*!*k*)
apply (*cases modifies* (*p*!*k*))
by *auto*
hence *sub*: (∑ *k* = *0*..*length p*−*1*. *if issub* (*p* ! *k*) ∧ *l* = *modifies* (*p* ! *k*)
then if fst ic = *k then Suc 0 else 0 else 0*) ≤ *Suc 0*
using *Groups-Big.ordered-comm-monoid-add-class.sum-mono*[*of* {*0*..*length p*−*1*}
λ*k*. (*if issub* (*p* ! *k*) ∧ *l* = *modifies* (*p* ! *k*) *then if fst ic* = *k then Suc 0 else 0 else 0*)
λ*k*. (*if k* = *fst ic then 1 else 0*)] *bound-fst-ic Orderings.ord-class.ord-eq-le-trans*
by *auto*

have *b*^(*Suc t*) ∗ (∑ *R*− *p l* (λ*k*. (*Z ic p l t*) ∗ *S ic p k t*)) ≤ *b*^(*Suc t*) ∗ *R ic p l t* **for** *t*
proof (*cases t*=*0*)
case *True*
hence *a* = *R ic p l 0* **by** (*auto simp add*: *0 R-def*)
thus *?thesis*
apply (*cases a*=*0*)
subgoal by (*auto simp add*: *True R-def Z-def sum-rsub.simps*)
subgoal
apply (*auto simp add*: *True R-def Z-def sum-rsub.simps S-def*)
using *sub* **by** *auto*
done
next
case *False*
define *cs* **where** *cs* ≡ *fst* (*steps ic p t*)
have *sub*: (∑ *R*− *p l* (λ*k*. *Z ic p l t* ∗ *S ic p k t*))
= (*if issub* (*p*!*cs*) ∧ *l* = *modifies* (*p*!*cs*) *then Z ic p l t else 0*)
using *single-step-sub Z-def R-def is-val l n-def cs-def n* **by** *auto*
show *?thesis* **using** *sub* **by** (*auto simp add*: *sum-rsub.simps R-def Z-def*)
qed

from *this* **have** *positive*: *b*^(*Suc t*) ∗ (∑ *R*− *p l* (λ*k*. (*Z ic p l t*) ∗ *S ic p k t*))
≤ *b*^(*Suc t*) ∗ *R ic p l t*
+ *b*^(*Suc t*) ∗ (∑ *R*+ *p l* (λ*k*. *S ic p k t*)) **for** *t*
by (*auto simp add*: *Nat.trans-le-add1*)

have *distrib-add*: ⋀*t*. *b*^*t* ∗ ∑ *R*+ *p l* (λ*k*. *S ic p k t*) = ∑ *R*+ *p l* (λ*k*. *b* ^ *t* ∗ *S ic p k t*)
by (*simp add*: *sum-radd-distrib*)
have *distrib-sub*: ⋀*t*. *b*^*t* ∗ ∑ *R*− *p l* (λ*k*. *Z ic p l t* ∗ *S ic p k t*)

$= \sum R-$ *p l* (λk. *b* ^ *t* ∗ (*Z ic p l t* ∗ *S ic p k t*))
by (*simp add*: *sum-rsub-distrib*)

have *commute-add*: ($\sum$ *t*=*0*..*q*−*1*. $\sum R+$ *p l* (λk. *b*^*t* ∗ *S ic p k t*))
= $\sum R+$ *p l* (λk. $\sum$ *t*=*0*..*q*−*1*. (*b*^*t* ∗ *S ic p k t*))
using *sum-radd-commutative*[*of p l* λk *t*. *b*^*t* ∗ *S ic p k t q*−*1*] **by** *auto*

have *length* (*snd ic*) > *0* **using** *is-val is-valid-initial-def*[*of ic p a*] **by** *auto*
hence *r-q*: *R ic p l q* = *0*
using *terminate terminates-def correct-halt-def l* **by** (*auto simp*: *n-def R-def*)
hence *z-q*: *Z ic p l q* = *0*
using *terminate* **by** (*auto simp*: *Z-def*)

have ∀ *k*<*length p*−*1*. ¬ *ishalt* (*p*!*k*)
using *is-val is-valid-initial-def*[*of ic p a*] *is-valid-def*[*of ic p*]
program-includes-halt.simps **by** *blast*
hence *s-q*: ∀ *k* < *length p* − *1*. *S ic p k q* = *0*
using *terminate terminates-def correct-halt-def* **by** (*auto simp*: *S-def*)

from *r-q* **have** *rq*: ($\sum$ *x* = *0*..*q* − *1*. *int b* ^ *x* ∗ *int* (*snd* (*steps ic p x*) ! *l*)) =
($\sum$ *x* = *0*..*q*. *int b* ^ *x* ∗ *int* (*snd* (*steps ic p x*) ! *l*))
by (*auto simp*: *r-q R-def*; *smt Suc-pred mult-0-right of-nat-0 of-nat-mult power-mult-distrib*
q
sum.atLeast0-atMost-Suc zero-power)

have ($\sum$ *t* = *0*..*q* − *1*. *b* ^ *t* ∗ (*Z ic p l t* ∗ *S ic p k t*))
+ (*b*^(*Suc* (*q*−*1*)) ∗ (*Z ic p l* (*Suc* (*q*−*1*)) ∗ *S ic p k* (*Suc* (*q*−*1*))))
= ($\sum$ *t* = *0*..*Suc* (*q*−*1*). *b* ^ *t* ∗ (*Z ic p l t* ∗ *S ic p k t*)) **for** *k*
using *comm-monoid-add-class.sum.atLeast0-atMost-Suc* **by** *auto*
hence *zq*: ($\sum$ *t* = *0*..*q* − *1*. *b* ^ *t* ∗ (*Z ic p l t* ∗ *S ic p k t*))
= ($\sum$ *t* = *0*..*q*. *b* ^ *t* ∗ (*Z ic p l t* ∗ *S ic p k t*)) **for** *k*
using *z-q q* **by** *auto*

have (*if isadd* (*p* ! *k*) ∧ *l* = *modifies* (*p* ! *k*) *then* $\sum$ *t* = *0*..*q* − *Suc 0*. *b* ^ *t* ∗ *S*
ic p k t else 0)
= (*if isadd* (*p* ! *k*) ∧ *l* = *modifies* (*p* ! *k*) *then* $\sum$ *t* = *0*..*q*. *b* ^ *t* ∗ *S ic p k t*
else 0) **for** *k*
proof (*cases p*!*k*)
case (*Add x11 x12*)
have *sep*: ($\sum$ *t* = *0*..*q*−*1*. *b* ^ *t* ∗ *S ic p k t*) + *b*^*q* ∗ *S ic p k q*
= ($\sum$ *t* = *0*..(*Suc* (*q*−*1*)). *b* ^ *t* ∗ *S ic p k t*)
using *comm-monoid-add-class.sum.atLeast0-atMost-Suc*[*of* λt. *b*^*t* ∗ *S ic p*
k t q−*1*] *q*
by *auto*
have *ishalt* (*p* ! (*fst* (*steps ic p q*)))
using *terminates-halt-state*[*of ic p*] *is-val terminate* **by** *auto*
hence *S ic p k q* = *0* **using** *Add S-def*[*of ic p k q*] **by** *auto*
with *sep q* **have** ($\sum$ *t* = *0*..*q* − *Suc 0*. *b* ^ *t* ∗ *S ic p k t*) = ($\sum$ *t* = *0*..*q*. *b* ^
t ∗ *S ic p k t*)

```
        by auto
      thus ?thesis by auto
    next
      case (Sub x21 x22 x23)
      then show ?thesis by auto
    next
      case Halt
      then show ?thesis by auto
    qed

    hence add-q: ∑R+ p l (λk. ∑t=0..(q−1). b^t ∗ S ic p k t)
               = ∑R+ p l (λk. ∑t=0..q. b^t ∗ S ic p k t)
    using sum-radd.simps single-step-add[of ic p a l snd ic] is-val l n-def by auto


  have r l = (∑t = 0..q. b^t ∗ R ic p l t) using r-def RLe-def by auto
  also have ... = R ic p l 0 + (∑t = 1..q. b^t ∗ R ic p l t)
    by (auto simp: q comm-monoid-add-class.sum.atLeast-Suc-atMost)
  also have ... = a + (∑t ∈ {1..q}. b^t ∗ R ic p l t)
    by (simp add: R-def 0)
  also have ... = a + (∑t = (Suc 0)..(Suc (q−1)). b^t ∗ R ic p l t) using q by
auto
  also have ... = a + (∑t ∈ (Suc ` {0..(q−1)}). b^t ∗ R ic p l t) by auto
  also have ... = a + (sum ((λt. b^t ∗ R ic p l t) ∘ Suc)) {0..(q−1)}
     using comm-monoid-add-class.sum.reindex[of Suc {0..(q−1)} (λt. b^t ∗ R ic
p l t)] by auto
  also have ... = a + (∑t = 0..(q−1). b^(Suc t) ∗ R ic p l (Suc t)) by auto
  also have ... = a + (∑t = 0..(q−1). b^(Suc t) ∗(R ic p l t
                                          + (∑R+ p l (λk. S ic p k t))
                                          − (∑R− p l (λk. (Z ic p l t) ∗ S ic p k t))))
    using lm04-06-one-step-relation-register[of ic p a l] is-val n n-def l
    by (auto simp add: n-def m-def)

  also have ... = a + (∑t ∈ {0..(q−1)}. b^(Suc t) ∗  R ic p l t
                          + b^(Suc t) ∗ (∑R+ p l (λk. S ic p k t))
                          − b^(Suc t) ∗ (∑R− p l (λk. (Z ic p l t) ∗ S ic p k t)))
    by (auto simp add: algebra-simps)

  finally have int (r l) = int a +(∑t ∈ {0..(q−1)}. int(
                            b^(Suc t) ∗  R ic p l t
                          + b^(Suc t) ∗ (∑R+ p l (λk. S ic p k t))
                          − b^(Suc t) ∗ (∑R− p l (λk. (Z ic p l t) ∗ S ic p k t))))
    by auto

  also have ... = int a + (∑t ∈ {0..(q−1)}. int (b^(Suc t) ∗  R ic p l t)
                          + int (b^(Suc t) ∗ (∑R+ p l (λk. S ic p k t)))
                          − int (b^(Suc t) ∗ (∑R− p l (λk. (Z ic p l t) ∗ S ic p k
t))))
    by (simp only: sum-int positive)
```

also have ... = *int a* + ($\sum$ *t* ∈ {*0*..(*q*−*1*)}. *int* (*b*^(*Suc t*) ∗ *R ic p l t*)
+ (*int* (*b*^(*Suc t*) ∗ ($\sum$ *R*+ *p l* (λ*k*. *S ic p k t*)))
− *int* (*b*^(*Suc t*) ∗ ($\sum$ *R*− *p l* (λ*k*. (*Z ic p l t*) ∗ *S ic p k t*)))))
by (*simp add*: *add-diff-eq*)

also have ... = *int a* + ($\sum$ *t* ∈ {*0*..(*q*−*1*)}. *int*(*b*^(*Suc t*) ∗ *R ic p l t*))
+ ($\sum$ *t* ∈ {*0*..(*q*−*1*)}. *int*(*b*^(*Suc t*) ∗ ($\sum$ *R*+ *p l* (λ*k*. *S ic p k t*)))
− *int*(*b*^(*Suc t*) ∗ ($\sum$ *R*− *p l* (λ*k*. (*Z ic p l t*) ∗ *S ic p k t*))))
by (*auto simp only*: *sum.distrib*)

also have ... = *int a* + *int b* ∗ *int* ($\sum$ *t* ∈ {*0*..(*q*−*1*)}. *b*^*t* ∗ *R ic p l t*)
+ *int b* ∗ ($\sum$ *t* ∈ {*0*..(*q*−*1*)}. *int*(*b*^*t* ∗ ($\sum$ *R*+ *p l* (λ*k*. *S ic p k t*)))
− *int*(*b*^*t* ∗ ($\sum$ *R*− *p l* (λ*k*. (*Z ic p l t*) ∗ *S ic p k t*))))
by (*auto simp*: *sum-distrib-left mult.assoc right-diff-distrib*)

also have ... = *int a* + *int b* ∗ *int* ($\sum$ *t* ∈ {*0*..(*q*−*1*)}. *b*^*t* ∗ *R ic p l t*)
+ *int b* ∗ ($\sum$ *t* ∈ {*0*..(*q*−*1*)}. *int*(*b*^*t* ∗ ($\sum$ *R*+ *p l* (λ*k*. *S ic p k t*))))
− *int b* ∗ ($\sum$ *t* ∈ {*0*..(*q*−*1*)}. *int*(*b*^*t* ∗ ($\sum$ *R*− *p l* (λ*k*. (*Z ic p l t*) ∗ *S ic p k t*))))
by (*auto simp add*: *sum.distrib int-distrib*(*4*) *sum-subtractf*)

also have ... = *int a* + *int b* ∗ *int* ($\sum$ *t* ∈ {*0*..(*q*−*1*)}. *b*^*t* ∗ *R ic p l t*)
+ *int b* ∗ ($\sum$ *t* ∈ {*0*..(*q*−*1*)}. *int*($\sum$ *R*+ *p l* (λ*k*. *b*^*t* ∗ *S ic p k t*)))
− *int b* ∗ ($\sum$ *t* ∈ {*0*..(*q*−*1*)}. *int*($\sum$ *R*− *p l* (λ*k*. *b*^*t* ∗ (*Z ic p l t* ∗ *S ic p k t*))))
using *distrib-add distrib-sub* **by** *auto*

also have ... = *int a* + *int b* ∗ *int* ($\sum$ *t* = *0*..*q*−*1*. *b*^*t* ∗ *R ic p l t*)
+ *int b* ∗ *int* ($\sum$ *t* = *0*..*q*−*1*. $\sum$ *R*+ *p l* (λ*k*. *b*^*t* ∗ *S ic p k t*))
− *int b* ∗ *int* ($\sum$ *t* = *0*..*q*−*1*. $\sum$ *R*− *p l* (λ*k*. *b*^*t* ∗ (*Z ic p l t* ∗ *S ic p k t*)))
by *auto*

also have ... = *int a* + *int b* ∗ *int* ($\sum$ *t* = *0*..*q*−*1*. *b*^*t* ∗ *R ic p l t*)
+ *int b* ∗ *int* ($\sum$ *R*+ *p l* (λ*k*. $\sum$ *t*=*0*..*q*−*1*. *b*^*t* ∗ *S ic p k t*))
− *int b* ∗ *int* ($\sum$ *R*− *p l* (λ*k*. $\sum$ *t*=*0*..*q*−*1*. *b*^*t* ∗ (*Z ic p l t* ∗ *S ic p k t*)))
using *sum-rsub-commutative*[*of p l* λ*k t*. *b*^*t* ∗ (*Z ic p l t* ∗ *S ic p k t*) *q*−*1*]
using *sum-radd-commutative*[*of p l* λ*k t*. *b*^*t* ∗ *S ic p k t q*−*1*] **by** *auto*

also have ... = *int a* + *int b* ∗ *int* ($\sum$ *t*=*0*..*q*. *b*^*t* ∗ *R ic p l t*)
+ *int b* ∗ *int* ($\sum$ *R*+ *p l* (λ*k*. $\sum$ *t*=*0*..*q*−*1*. *b*^*t* ∗ *S ic p k t*))
− *int b* ∗ *int* ($\sum$ *R*− *p l* (λ*k*. $\sum$ *t*=*0*..*q*−*1*. *b*^*t* ∗ (*Z ic p l t* ∗ *S ic p k t*)))
by (*auto simp*: *rq R-def*; *smt One-nat-def rq*)

also have ... = *int a* + *int b* ∗ *int* (∑ *t*=*0*..*q*. *b*^*t* ∗ *R ic p l t*)
+ *int b* ∗ *int* (∑ *R*+ *p l* (λ*k*. ∑ *t*=*0*..*q*. *b*^*t* ∗ *S ic p k t*))
− *int b* ∗ *int* (∑ *R*− *p l* (λ*k*. ∑ *t*=*0*..*q*. *b*^*t* ∗ (*Z ic p l t* ∗ *S ic p k t*)))
using *zq add-q* **by** *auto*

also have ... = *int a* + *int b* ∗ *int* (*RLe ic p b q l*)
+ *int b* ∗ *int* (∑ *R*+ *p l* (*SKe ic p b q*))
− *int b* ∗ *int* (∑ *R*− *p l* (λ*k*. ∑ *t*=*0*..*q*. *b*^*t* ∗ (*Z ic p l t* ∗ *S ic p k t*)))
by (*auto simp*: *RLe-def*; *metis SKe-def*)

also have ... = *int a* + *int b* ∗ *int* (*RLe ic p b q l*)
+ *int b* ∗ *int* (∑ *R*+ *p l* (*SKe ic p b q*))
− *int b* ∗ *int* (∑ *R*− *p l* (λ*k*. *ZLe ic p b q l* && *SKe ic p b q k*))
using *mult-to-bitAND c-gt-cells b-def c* **by** *auto*

finally have *int*(*r l*) = *int a* + *int b* ∗ *int* (*r l*)
+ *int b* ∗ *int* (∑ *R*+ *p l s*)
− *int b* ∗ *int* (∑ *R*− *p l* (λ*k*. *z l* && *s k*))
by (*auto simp*: *r-def s-def z-def*)

hence *r l* = *a* + *b* ∗ *r l*
+ *b* ∗ ∑ *R*+ *p l s*
− *b* ∗ ∑ *R*− *p l* (λ*k*. *z l* && *s k*)
using *int-ops*(*5*) *int-ops*(*7*) *nat-int nat-minus-as-int* **by** *presburger*

thus *?thesis* **by** *simp*
qed

end

3.5.2 States

theory *MultipleStepState*
imports *SingleStepState*
begin

lemma *lm04-24-multiple-step-states*:
fixes *c* :: *nat*
and *l* :: *register*
and *ic* :: *configuration*
and *p* :: *program*
and *q* :: *nat*
and *a* :: *nat*

defines *b* == *B c*
and *m* == *length p*

assumes *is-val*: *is-valid-initial ic p a*
assumes *c-gt-cells*: *cells-bounded ic p c*
assumes *d*: $d \le m{-}1$ **and** $0 < d$
and *q*: $q > 0$

assumes *terminate*: *terminates ic p q*

assumes *c*: $c > 1$

defines $r \equiv$ *RLe ic p b q*
and $z \equiv$ *ZLe ic p b q*
and $s \equiv$ *SKe ic p b q*
and $e \equiv \sum t = 0..q.\ b\hat{\ }t$

shows $s\ d = b * (\sum S{+}\ p\ d\ s)$
$+\ b * (\sum S{-}\ p\ d\ (\lambda k.\ z\ (modifies\ (p!k))\ \&\&\ s\ k))$
$+\ b * (\sum S0\ p\ d\ (\lambda k.\ (e - z\ (modifies\ (p!k)))\ \&\&\ s\ k))$
proof –
have *program-includes-halt p*
using *is-val is-valid-initial-def*[*of ic p a*] *is-valid-def*[*of ic p*] **by** *auto*

have *halt-term0*: $t \le q{-}1 \longrightarrow$ (*if ishalt* (*p*!(*fst* (*steps ic p t*))) $\wedge$ *d* = *fst* (*steps ic p t*)
then Suc 0 else 0) = *0* **for** *t*
using *terminate terminates-def* **by** *auto*

have *single-step*: *S ic p d* (*Suc t*) = ($\sum S{+}$ *p d* (λk. *S ic p k t*))
+ ($\sum S{-}$ *p d* (λk. *Z ic p* (*modifies* (*p*!*k*)) *t* ∗ *S ic p k t*))
+ ($\sum S0$ *p d* (λk. (*1* − *Z ic p* (*modifies* (*p*!*k*)) *t*) ∗ *S ic p k t*))
+ (*if ishalt* (*p*!(*fst* (*steps ic p t*))) $\wedge$ *d* = *fst* (*steps ic p t*) *then Suc 0 else 0*) **for** *t*
using *lm04-07-one-step-relation-state*[*of ic p a d*] *is-val* ‹*d*>*0*› *d*
by (*simp add*: *m-def*)

have *b*: *b* > *0* **using** *b-def B-def* **by** *auto*
have *halt*: *ishalt* (*p*!*fst*(*steps ic p q*)) **using** *terminate terminates-def correct-halt-def* **by** *auto*
have *add-conditions*: (*if isadd* (*p* ! *k*) $\wedge$ *d* = *goes-to* (*p* ! *k*)
then ($\sum t = 0..q - Suc\ 0.\ b\ \hat{}\ t * S\ ic\ p\ k\ t$) + *b* ^ *q* ∗ *S ic p k q else 0*)
= (*if isadd* (*p* ! *k*) $\wedge$ *d* = *goes-to* (*p* ! *k*)
then $\sum t = 0..q - Suc\ 0.\ b\ \hat{}\ t * S\ ic\ p\ k\ t$ *else 0*) **for** *k*
apply (*cases p*!*k*; *cases d* = *goes-to* (*p*!*k*)) **using** *q S-def b halt* **by** *auto*
have *b* ∗ *b* ^ (*q* − *Suc 0*) = *b* ^ (*q* − *Suc 0* + *Suc 0*) **using** *q*
by (*simp add*: *power-eq-if*)
have (λk. ($\sum t = 0..(q{-}1)$. *b*^*t* ∗ *S ic p k t*) + *b*^(*Suc* (*q*−*1*)) ∗ *S ic p k* (*Suc* (*q*−*1*)))
= (λk. ($\sum t = 0..(Suc\ (q{-}1))$. *b*^*t* ∗ *S ic p k t*)) **by** *auto*
hence $\sum S{+}$ *p d* (λk. ($\sum t = 0..(q{-}1)$. *b*^*t* ∗ *S ic p k t*) + *b*^*q* ∗ *S ic p k* (*Suc* (*q*−*1*)))

$= \sum S+\ p\ d\ (\lambda k.\ \sum t = 0..(Suc\ (q-1)).\ b\hat{}t * S\ ic\ p\ k\ t)$ **using** *q*
by *auto*
hence *add-q*: $\sum S+\ p\ d\ (\lambda k.\ \sum t = 0..(q-1).\ b\hat{}t * S\ ic\ p\ k\ t)$
$= \sum S+\ p\ d\ (\lambda k.\ \sum t = 0..q.\ b\hat{}t * S\ ic\ p\ k\ t)$
by (*auto simp add*: *sum-sadd.simps q add-conditions*)

have *issub* $(p!k) \Longrightarrow b\ \hat{}(Suc\ (q-1)) * (Z\ ic\ p\ (modifies\ (p\ !\ k))\ (Suc\ (q-1)) *$
$(if\ fst\ (steps\ ic\ p\ (Suc\ (q-1))) = k\ then\ Suc\ 0\ else\ 0)) = 0$ **for** *k*
by (*auto simp*: *q halt*)
hence *sum-equiv-nzero*: *issub* $(p!k) \Longrightarrow$
$(\sum t = 0..q-1.\ b\ \hat{}\ t * (Z\ ic\ p\ (modifies\ (p\ !\ k))\ t *$
$(if\ fst\ (steps\ ic\ p\ t) = k\ then\ Suc\ 0\ else\ 0)))$
$= (\sum t = 0..(Suc\ (q-1)).\ b\ \hat{}\ t * (Z\ ic\ p\ (modifies\ (p\ !\ k))\ t *$
$(if\ fst\ (steps\ ic\ p\ t) = k\ then\ Suc\ 0\ else\ 0)))$ **for** *k*
using *sum.atLeast0-atMost-Suc*[*of* $\lambda t.\ b\ \hat{}\ t * (Z\ ic\ p\ (modifies\ (p\ !\ k))\ t$
$* (if\ fst\ (steps\ ic\ p\ t) = k\ then\ Suc\ 0\ else\ 0))\ q-1$] **by**
auto
hence *sub-nzero-conditions*: $(if\ issub\ (p\ !\ k) \wedge d = goes\text{-}to\ (p\ !\ k)\ then$
$\sum t = 0..q - Suc\ 0.\ b\ \hat{}\ t * (Z\ ic\ p\ (modifies\ (p\ !\ k))\ t * S\ ic\ p\ k\ t)\ else\ 0)$
$= (if\ issub\ (p\ !\ k) \wedge d = goes\text{-}to\ (p\ !\ k)\ then$
$\sum t = 0..q.\ b\ \hat{}\ t * (Z\ ic\ p\ (modifies\ (p\ !\ k))\ t * S\ ic\ p\ k\ t)\ else\ 0)$ **for** *k*
apply (*cases issub* $(p!k)$) **using** *q S-def halt b* **by** *auto*
have $(\lambda k.\ (\sum t{=}0..(q-1).\ b\hat{}t * (Z\ ic\ p\ (modifies\ (p!k))\ t * S\ ic\ p\ k\ t))$
$+ b\hat{}(Suc\ (q-1)) * (Z\ ic\ p\ (modifies\ (p!k))\ (Suc\ (q-1)) * S\ ic\ p\ k$
$(Suc\ (q-1))))$
$= (\lambda k.\ \sum t{=}0..(Suc\ (q-1)).\ b\hat{}t * (Z\ ic\ p\ (modifies\ (p!k))\ t * S\ ic\ p\ k\ t))$ **by**
auto
hence *sub-nzero-q*: $(\sum S-\ p\ d\ (\lambda k.\ \sum t{=}0..(q-1).\ b\hat{}t * (Z\ ic\ p\ (modifies\ (p!k))$
$t * S\ ic\ p\ k\ t)))$
$= (\sum S-\ p\ d\ (\lambda k.\ \sum t{=}0..q.\ b\hat{}t * (Z\ ic\ p\ (modifies\ (p!k))\ t * S\ ic$
$p\ k\ t)))$
by (*auto simp*: *sum-ssub-nzero.simps q sub-nzero-conditions*)

have *issub* $(p!k) \Longrightarrow b\ \hat{}(Suc\ (q-1)) * ((Suc\ 0 - Z\ ic\ p\ (modifies\ (p\ !\ k))\ (Suc$
$(q-1)))$
$* S\ ic\ p\ k\ (Suc\ (q-1))) = 0$ **for** *k* **using** *q halt S-def* **by** *auto*
hence *sum-equiv-zero*: *issub* $(p!k) \Longrightarrow$
$(\sum t = 0..q-1.\ b\ \hat{}\ t * ((Suc\ 0 - Z\ ic\ p\ (modifies\ (p\ !\ k))\ t) * S\ ic\ p\ k\ t))$
$= (\sum t = 0..Suc\ (q-1).\ b\ \hat{}\ t * ((Suc\ 0 - Z\ ic\ p\ (modifies\ (p\ !\ k))\ t) * S\ ic$
$p\ k\ t))$ **for** *k*
using *sum.atLeast0-atMost-Suc*[*of* $\lambda t.\ b\ \hat{}\ t * ((Suc\ 0 - Z\ ic\ p\ (modifies\ (p\ !$
$k))\ t)$
$* S\ ic\ p\ k\ t)\ q-1$] **by** *auto*
have $(if\ issub\ (p\ !\ k) \wedge d = goes\text{-}to\text{-}alt\ (p\ !\ k)\ then$
$\sum t = 0..q - Suc\ 0.\ b\ \hat{}\ t * ((Suc\ 0 - Z\ ic\ p\ (modifies\ (p\ !\ k))\ t) * S\ ic$
$p\ k\ t)\ else\ 0)$
$= (if\ issub\ (p\ !\ k) \wedge d = goes\text{-}to\text{-}alt\ (p\ !\ k)\ then$
$\sum t = 0..q.\ b\ \hat{}\ t * ((Suc\ 0 - Z\ ic\ p\ (modifies\ (p\ !\ k))\ t) * S\ ic\ p\ k\ t)$
$else\ 0)$ **for** *k*

apply (*cases issub* (*p*!*k*)) **using** *sum-equiv-zero*[*of k*] *q* **by** *auto*
hence *sub-zero-q*: (∑ *S0 p d* (λ*k*.∑ *t*=*0*..*q*−*1*. *b*^*t* ∗ ((*1* − *Z ic p* (*modifies*(*p*!*k*)) *t*) ∗ *S ic p k t*)))
= (∑ *S0 p d* (λ*k*.∑ *t*=*0*..*q*. *b*^*t* ∗ ((*1* − *Z ic p* (*modifies* (*p*!*k*)) *t*) ∗ *S ic p k t*)))
using *sum-ssub-zero.simps q* **by** *auto*

have *s d* = (∑ *t* = *0*..*q*. *b*^*t* ∗ *S ic p d t*) **using** *s-def SKe-def* **by** *auto*
also have ... = *S ic p d 0* + (∑ *t* = *1*..*q*. *b*^*t* ∗ *S ic p d t*)
by (*auto simp*: *q comm-monoid-add-class.sum.atLeast-Suc-atMost*)
also have ... = (∑ *t* = *1*..*q*. *b*^*t* ∗ *S ic p d t*)
using *S-def* ‹*d*>*0*› *is-val is-valid-initial-def*[*of ic p a*] **by** *auto*
also have ... = (∑ *t* ∈ (*Suc* ‘ {*0*..(*q*−*1*)}). *b*^*t* ∗ *S ic p d t*) **using** *q* **by** *auto*
also have ... = (*sum* ((λ*t*. *b*^*t* ∗ *S ic p d t*) ∘ *Suc*)) {*0*..(*q*−*1*)}
using *comm-monoid-add-class.sum.reindex*[*of Suc* {*0*..(*q*−*1*)} (λ*t*. *b*^*t* ∗ *S ic p d t*)] **by** *auto*
also have ... = (∑ *t* = *0*..(*q*−*1*). *b*^(*Suc t*) ∗((∑ *S*+ *p d* (λ*k*. *S ic p k t*))
+ (∑ *S*− *p d* (λ*k*. *Z ic p* (*modifies* (*p*!*k*)) *t* ∗ *S ic p k t*))
+ (∑ *S0 p d* (λ*k*. (*1* − *Z ic p* (*modifies* (*p*!*k*)) *t*) ∗ *S ic p k t*))
+ (*if ishalt* (*p*!(*fst* (*steps ic p t*))) ∧ *d* = *fst* (*steps ic p t*)
then Suc 0 else 0)))
using *single-step* **by** *auto*
also have ... = (∑ *t* = *0*..(*q*−*1*). *b*^(*Suc t*) ∗((∑ *S*+ *p d* (λ*k*. *S ic p k t*))
+ (∑ *S*− *p d* (λ*k*. *Z ic p* (*modifies* (*p*!*k*)) *t* ∗ *S ic p k t*))
+ (∑ *S0 p d* (λ*k*. (*1* − *Z ic p* (*modifies* (*p*!*k*)) *t*) ∗ *S ic p k t*))))
using *halt-term0* **by** *auto*
also have ... = (∑ *t* = *0*..(*q*−*1*). (*b*^(*Suc t*) ∗ (∑ *S*+ *p d* (λ*k*. *S ic p k t*))
+ *b*^(*Suc t*) ∗ (∑ *S*− *p d* (λ*k*. *Z ic p* (*modifies* (*p*!*k*)) *t* ∗ *S ic p k t*))
+ *b*^(*Suc t*) ∗ (∑ *S0 p d* (λ*k*. (*1* − *Z ic p* (*modifies* (*p*!*k*)) *t*) ∗ *S ic p k t*))))
by (*simp add*: *algebra-simps*)
also have ... = (∑ *t* = *0*..(*q*−*1*). (*b*^(*Suc t*) ∗ (∑ *S*+ *p d* (λ*k*. *S ic p k t*))))
+(∑ *t*=*0*..(*q*−*1*). *b*^(*Suc t*)∗(∑ *S*− *p d* (λ*k*. *Z ic p* (*modifies* (*p*!*k*)) *t* ∗ *S ic p k t*)))
+(∑ *t*=*0*..(*q*−*1*). *b*^(*Suc t*)∗(∑ *S0 p d* (λ*k*. (*1* − *Z ic p* (*modifies* (*p*!*k*)) *t*) ∗ *S ic p k t*)))
by (*auto simp only*: *sum.distrib*)
also have ... = *b* ∗ (∑ *t* = *0*..(*q*−*1*). (*b*^*t* ∗ (∑ *S*+ *p d* (λ*k*. *S ic p k t*))))
+ *b*∗(∑ *t*=*0*..(*q*−*1*). *b*^*t*∗(∑ *S*− *p d* (λ*k*. *Z ic p* (*modifies* (*p*!*k*)) *t* ∗ *S ic p k t*)))
+ *b*∗(∑ *t*=*0*..(*q*−*1*). *b*^*t*∗(∑ *S0 p d* (λ*k*. (*1* − *Z ic p* (*modifies* (*p*!*k*)) *t*) ∗ *S ic p k t*)))
by (*auto simp*: *algebra-simps sum-distrib-left*)
also have ... = *b* ∗ (∑ *t* = *0*..(*q*−*1*). (∑ *S*+ *p d* (λ*k*. *b*^*t* ∗ *S ic p k t*)))
+ *b*∗(∑ *t*=*0*..(*q*−*1*). (∑ *S*− *p d* (λ*k*. *b*^*t* ∗ (*Z ic p* (*modifies* (*p*!*k*)) *t* ∗ *S*

ic p k t))))
    + *b*∗(∑ *t*=*0*..(*q*−*1*). (∑ *S0 p d* (λ*k*. *b*^*t* ∗ ((*1* − *Z ic p* (*modifies* (*p*!*k*)) *t*) ∗ *S ic p k t*))))
  using *sum-sadd-distrib sum-ssub-nzero-distrib sum-ssub-zero-distrib* **by** *auto*
 also have ... = *b* ∗ (∑ *S*+ *p d* (λ*k*. ∑ *t* = *0*..(*q*−*1*). *b*^*t* ∗ *S ic p k t*))
    + *b*∗(∑ *S*− *p d* (λ*k*. ∑ *t*=*0*..(*q*−*1*). *b*^*t* ∗ (*Z ic p* (*modifies* (*p*!*k*)) *t* ∗ *S ic p k t*)))
    + *b*∗(∑ *S0 p d* (λ*k*. ∑ *t*=*0*..(*q*−*1*). *b*^*t* ∗ ((*1* − *Z ic p* (*modifies* (*p*!*k*)) *t*) ∗ *S ic p k t*)))
  using *sum-sadd-commutative sum-ssub-nzero-commutative sum-ssub-zero-commutative* **by** *auto*

 finally have *eq1*: *s d* = *b* ∗ (∑ *S*+ *p d* (λ*k*. ∑ *t* = *0*..*q*. *b*^*t* ∗ *S ic p k t*))
    + *b*∗(∑ *S*− *p d* (λ*k*. ∑ *t*=*0*..*q*. *b*^*t* ∗ (*Z ic p* (*modifies* (*p*!*k*)) *t* ∗ *S ic p k t*)))
    + *b*∗(∑ *S0 p d* (λ*k*. ∑ *t*=*0*..*q*. *b*^*t* ∗ ((*1* − *Z ic p* (*modifies* (*p*!*k*)) *t*) ∗ *S ic p k t*)))
  using *add-q sub-nzero-q sub-zero-q* **by** *auto*
 also have ... = *b* ∗ (∑ *S*+ *p d* (λ*k*. *s k*))
    + *b*∗(∑ *S*− *p d* (λ*k*. ∑ *t*=*0*..*q*. *b*^*t* ∗ (*Z ic p* (*modifies* (*p*!*k*)) *t* ∗ *S ic p k t*)))
    + *b*∗(∑ *S0 p d* (λ*k*. ∑ *t*=*0*..*q*. *b*^*t* ∗ ((*1* − *Z ic p* (*modifies* (*p*!*k*)) *t*) ∗ *S ic p k t*)))
  using *SKe-def s-def* **by** *auto*
 finally have *s d* = *b* ∗ (∑ *S*+ *p d s*)
    + *b*∗(∑ *S*− *p d* (λ*k*. ∑ *t*=*0*..*q*. *b*^*t* ∗ (*Z ic p* (*modifies* (*p*!*k*)) *t* ∗ *S ic p k t*)))
    + *b*∗(∑ *S0 p d* (λ*k*. ∑ *t*=*0*..*q*. *b*^*t* ∗ ((*1* − *Z ic p* (*modifies* (*p*!*k*)) *t*) ∗ *S ic p k t*)))
  by *auto*
 also have ... = *b* ∗ (∑ *S*+ *p d s*)
    + *b*∗(∑ *S*− *p d* (λ*k*. *ZLe ic p b q* (*modifies* (*p*!*k*)) && *SKe ic p b q k*))
    + *b*∗(∑ *S0 p d* (λ*k*. ∑ *t*=*0*..*q*. *b*^*t* ∗ ((*1* − *Z ic p* (*modifies* (*p*!*k*)) *t*) ∗ *S ic p k t*)))
  using *mult-to-bitAND c-gt-cells b-def c* **by** *auto*
 finally have *s d* = *b* ∗ (∑ *S*+ *p d s*)
    + *b*∗(∑ *S*− *p d* (λ*k*. *ZLe ic p b q* (*modifies* (*p*!*k*)) && *SKe ic p b q k*))
    + *b*∗(∑ *S0 p d* (λ*k*. (*e* − *ZLe ic p b q* (*modifies* (*p*!*k*))) && *SKe ic p b q k*))
  using *mult-to-bitAND-state c-gt-cells b-def c e-def* **by** *auto*
 thus *?thesis* **using** *s-def z-def* **by** *auto*
qed

lemma *lm04-25-multiple-step-state1*:
fixes *c* :: *nat*
  and *l* :: *register*
  and *ic* :: *configuration*
  and *p* :: *program*
  and *q* :: *nat*

and *a* :: *nat*

defines *b* == *B c*
and *m* == *length p*

assumes *is-val*: *is-valid-initial ic p a*
assumes *c-gt-cells*: *cells-bounded ic p c*
assumes *d*: *d=0*
and *q*: *q* > *0*

assumes *terminate*: *terminates ic p q*

assumes *c*: *c* > *1*

defines *r* ≡ *RLe ic p b q*
and *z* ≡ *ZLe ic p b q*
and *s* ≡ *SKe ic p b q*
and *e* ≡ ∑ *t* = *0*..*q*. *b*^*t*

shows *s d* = *1* + *b* ∗ (∑ *S+ p d s*)
+ *b* ∗ (∑ *S− p d* (λ*k*. *z* (*modifies* (*p*!*k*)) && *s k*))
+ *b* ∗ (∑ *S0 p d* (λ*k*. (*e* − *z* (*modifies* (*p*!*k*))) && *s k*))
proof −
have *program-includes-halt p*
using *is-val is-valid-initial-def*[*of ic p a*] *is-valid-def*[*of ic p*] **by** *auto*
hence *p* ≠ [] **by** *auto*
have ¬ *ishalt* (*p*!*d*) **using** *d m-def* ‹*program-includes-halt p*› **by** *auto*
hence (*if ishalt* (*p* ! *fst* (*steps ic p t*)) ∧ *d* = *fst* (*steps ic p t*) *then Suc 0 else 0*) = *0* **for** *t*
by *auto*
hence *single-step*: ⋀*t*. *S ic p d* (*Suc t*) = (∑ *S+ p d* (λ*k*. *S ic p k t*))
+ (∑ *S− p d* (λ*k*. *Z ic p* (*modifies* (*p*!*k*)) *t* ∗ *S ic p k t*))
+ (∑ *S0 p d* (λ*k*. (*1* − *Z ic p* (*modifies* (*p*!*k*)) *t*) ∗ *S ic p k t*))
using *lm04-07-one-step-relation-state*[*of ic p a d*] *is-val d* ‹*p* ≠ []› **by** (*simp add*: *m-def*)

have *b*: *b* > *0* **using** *b-def B-def* **by** *auto*
have *halt*: *ishalt* (*p*!*fst*(*steps ic p q*)) **using** *terminate terminates-def correct-halt-def* **by** *auto*
have *add-conditions*: (*if isadd* (*p* ! *k*) ∧ *d* = *goes-to* (*p* ! *k*)
then (∑ *t* = *0*..*q* − *Suc 0*. *b* ^ *t* ∗ *S ic p k t*) + *b* ^ *q* ∗ *S ic p k q else 0*)
= (*if isadd* (*p* ! *k*) ∧ *d* = *goes-to* (*p* ! *k*)
then ∑ *t* = *0*..*q* − *Suc 0*. *b* ^ *t* ∗ *S ic p k t else 0*) **for** *k*
apply (*cases p*!*k*; *cases d* = *goes-to* (*p*!*k*)) **using** *q S-def b halt* **by** *auto*
have *b* ∗ *b* ^(*q* − *Suc 0*) = *b* ^(*q* − *Suc 0* + *Suc 0*) **using** *q*
by (*simp add*: *power-eq-if*)
have (λ*k*. (∑ *t* = *0*..(*q*−*1*). *b*^*t* ∗ *S ic p k t*) + *b*^(*Suc* (*q*−*1*)) ∗ *S ic p k* (*Suc* (*q*−*1*)))
= (λ*k*. (∑ *t* = *0*..(*Suc* (*q*−*1*)). *b*^*t* ∗ *S ic p k t*)) **by** *auto*

hence $\sum S+\ p\ d\ (\lambda k.\ (\sum t = 0..(q-1).\ \ \ \ b\hat{\ }t * S\ ic\ p\ k\ t) + b\hat{\ }q * S\ ic\ p\ k\ (Suc\ (q-1)))$
$= \sum S+\ p\ d\ (\lambda k.\ \sum t = 0..(Suc\ (q-1)).\ b\hat{\ }t * S\ ic\ p\ k\ t)$ **using** q
by *auto*
hence *add-q*: $\sum S+\ p\ d\ (\lambda k.\ \sum t = 0..(q-1).\ b\hat{\ }t * S\ ic\ p\ k\ t)$
$= \sum S+\ p\ d\ (\lambda k.\ \sum t = 0..q.\ b\hat{\ }t * S\ ic\ p\ k\ t)$
by (*auto simp add*: *sum-sadd.simps q add-conditions*)

have *issub* $(p!k) \Longrightarrow b\ \hat{\ }(Suc\ (q-1)) * (Z\ ic\ p\ (modifies\ (p\ !\ k))\ (Suc\ (q-1)) *$
$(if\ fst\ (steps\ ic\ p\ (Suc\ (q-1))) = k\ then\ Suc\ 0\ else\ 0)) = 0$ **for** k
by (*auto simp*: *q halt*)
hence *sum-equiv-nzero*: *issub* $(p!k) \Longrightarrow$
$(\sum t = 0..q-1.\ b\ \hat{\ }\ t * (Z\ ic\ p\ (modifies\ (p\ !\ k))\ t *$
$(if\ fst\ (steps\ ic\ p\ t) = k\ then\ Suc\ 0\ else\ 0)))$
$= (\sum t = 0..(Suc\ (q-1)).\ b\ \hat{\ }\ t * (Z\ ic\ p\ (modifies\ (p\ !\ k))\ t *$
$(if\ fst\ (steps\ ic\ p\ t) = k\ then\ Suc\ 0\ else\ 0)))$ **for** k
using *sum.atLeast0-atMost-Suc*[*of* $\lambda t.\ b\ \hat{\ }\ t * (Z\ ic\ p\ (modifies\ (p\ !\ k))\ t$
$* (if\ fst\ (steps\ ic\ p\ t) = k\ then\ Suc\ 0\ else\ 0))\ q-1$] **by** *auto*
hence *sub-nzero-conditions*: $(if\ issub\ (p\ !\ k) \wedge d = goes\text{-}to\ (p\ !\ k)\ then$
$\sum t = 0..q - Suc\ 0.\ b\ \hat{\ }\ t * (Z\ ic\ p\ (modifies\ (p\ !\ k))\ t * S\ ic\ p\ k\ t)\ else\ 0)$
$= (if\ issub\ (p\ !\ k) \wedge d = goes\text{-}to\ (p\ !\ k)\ then$
$\sum t = 0..q.\ b\ \hat{\ }\ t * (Z\ ic\ p\ (modifies\ (p\ !\ k))\ t * S\ ic\ p\ k\ t)\ else\ 0)$ **for** k
apply (*cases issub* $(p!k)$) **using** *q S-def halt b* **by** *auto*
have $(\lambda k.\ (\sum t{=}0..(q-1).\ b\hat{\ }t * (Z\ ic\ p\ (modifies\ (p!k))\ t * S\ ic\ p\ k\ t))$
$+ b\hat{\ }(Suc\ (q-1)) * (Z\ ic\ p\ (modifies\ (p!k))\ (Suc\ (q-1)) * S\ ic\ p\ k\ (Suc\ (q-1))))$
$= (\lambda k.\ \sum t{=}0..(Suc\ (q-1)).\ b\hat{\ }t * (Z\ ic\ p\ (modifies\ (p!k))\ t * S\ ic\ p\ k\ t))$ **by** *auto*
hence *sub-nzero-q*: $(\sum S-\ p\ d\ (\lambda k.\ \sum t{=}0..(q-1).\ b\hat{\ }t * (Z\ ic\ p\ (modifies\ (p!k))\ t * S\ ic\ p\ k\ t)))$
$= (\sum S-\ p\ d\ (\lambda k.\ \sum t{=}0..q.\ b\hat{\ }t * (Z\ ic\ p\ (modifies\ (p!k))\ t * S\ ic\ p\ k\ t)))$
by (*auto simp*: *sum-ssub-nzero.simps q sub-nzero-conditions*)

have *issub* $(p!k) \Longrightarrow b\ \hat{\ }(Suc\ (q-1)) * ((Suc\ 0 - Z\ ic\ p\ (modifies\ (p\ !\ k))\ (Suc\ (q-1)))$
$* S\ ic\ p\ k\ (Suc\ (q-1))) = 0$ **for** k **using** *q halt S-def* **by** *auto*
hence *sum-equiv-zero*: *issub* $(p!k) \Longrightarrow$
$(\sum t = 0..q-1.\ b\ \hat{\ }\ t * ((Suc\ 0 - Z\ ic\ p\ (modifies\ (p\ !\ k))\ t) * S\ ic\ p\ k\ t))$
$= (\sum t = 0..Suc\ (q-1).\ b\ \hat{\ }\ t * ((Suc\ 0 - Z\ ic\ p\ (modifies\ (p\ !\ k))\ t) * S\ ic\ p\ k\ t))$ **for** k
using *sum.atLeast0-atMost-Suc*[*of* $\lambda t.\ b\ \hat{\ }\ t * ((Suc\ 0 - Z\ ic\ p\ (modifies\ (p\ !\ k))\ t)$
$* S\ ic\ p\ k\ t)\ q-1$] **by** *auto*
have $(if\ issub\ (p\ !\ k) \wedge d = goes\text{-}to\text{-}alt\ (p\ !\ k)\ then$
$\sum t = 0..q - Suc\ 0.\ b\ \hat{\ }\ t * ((Suc\ 0 - Z\ ic\ p\ (modifies\ (p\ !\ k))\ t) * S\ ic\ p\ k\ t)\ else\ 0)$
$= (if\ issub\ (p\ !\ k) \wedge d = goes\text{-}to\text{-}alt\ (p\ !\ k)\ then$

$\sum t = 0..q.\ b\ \hat{}\ t * ((Suc\ 0\ -\ Z\ ic\ p\ (modifies\ (p\ !\ k))\ t) * S\ ic\ p\ k\ t)$ *else 0*) **for** *k*
apply (*cases issub* (*p*!*k*)) **using** *sum-equiv-zero*[*of k*] *q* **by** *auto*
hence *sub-zero-q*: $(\sum S0\ p\ d\ (\lambda k.\sum t{=}0..q{-}1.\ b\hat{}t * ((1\ -\ Z\ ic\ p\ (modifies(p!k))\ t) * S\ ic\ p\ k\ t)))$
$= (\sum S0\ p\ d\ (\lambda k.\sum t{=}0..q.\ b\hat{}t * ((1\ -\ Z\ ic\ p\ (modifies\ (p!k))\ t) * S\ ic\ p\ k\ t)))$
using *sum-ssub-zero.simps q* **by** *auto*

have *S0*: *S ic p d 0* = *1* **using** *S-def is-val is-valid-initial-def*[*of ic p a*] *d* **by** *auto*

have $s\ d = (\sum t = 0..q.\ b\hat{}t * S\ ic\ p\ d\ t)$ **using** *s-def SKe-def* **by** *auto*
also have ... $= S\ ic\ p\ d\ 0 + (\sum t = 1..q.\ b\hat{}t * S\ ic\ p\ d\ t)$
by (*auto simp*: *q comm-monoid-add-class.sum.atLeast-Suc-atMost*)
also have ... $= b\hat{}0 * S\ ic\ p\ d\ 0 + (\sum t = 1..q.\ b\hat{}t * S\ ic\ p\ d\ t)$
using *S-def is-val is-valid-initial-def*[*of ic p a*] **by** *auto*
also have ... $= 1 + (\sum t \in (Suc\ `\ \{0..(q{-}1)\}).\ b\hat{}t * S\ ic\ p\ d\ t)$ **using** *q S0* **by** *auto*
also have ... $= 1 + (sum\ \ ((\lambda t.\ b\hat{}t * S\ ic\ p\ d\ t) \circ Suc))\ \{0..(q{-}1)\}$
using *comm-monoid-add-class.sum.reindex*[*of Suc* $\{0..(q{-}1)\}$ $(\lambda t.\ b\hat{}t * S\ ic\ p\ d\ t)$] **by** *auto*
also have ... $= 1 + (\sum t = 0..(q{-}1).\ b\hat{}(Suc\ t) * ((\sum S{+}\ p\ d\ (\lambda k.\ S\ ic\ p\ k\ t))$
$+ (\sum S{-}\ p\ d\ (\lambda k.\ Z\ ic\ p\ (modifies\ (p!k))\ t * S\ ic\ p\ k\ t))$
$+ (\sum S0\ p\ d\ (\lambda k.\ (1\ -\ Z\ ic\ p\ (modifies\ (p!k))\ t) * S\ ic\ p\ k\ t))))$
using *single-step* **by** *auto*
also have ... $= 1 + (\sum t = 0..(q{-}1).\ (b\hat{}(Suc\ t) * (\sum S{+}\ p\ d\ (\lambda k.\ S\ ic\ p\ k\ t))$
$+ b\hat{}(Suc\ t) * (\sum S{-}\ p\ d\ (\lambda k.\ Z\ ic\ p\ (modifies\ (p!k))\ t * S\ ic\ p\ k\ t))$
$+ b\hat{}(Suc\ t) * (\sum S0\ p\ d\ (\lambda k.\ (1\ -\ Z\ ic\ p\ (modifies\ (p!k))\ t) * S\ ic\ p\ k\ t))))$
by (*simp add*: *algebra-simps*)
also have ... $= 1 + (\sum t = 0..(q{-}1).\ (b\hat{}(Suc\ t) * (\sum S{+}\ p\ d\ (\lambda k.\ S\ ic\ p\ k\ t))))$
$+(\sum t{=}0..(q{-}1).\ b\hat{}(Suc\ t)*(\sum S{-}\ p\ d\ (\lambda k.\ Z\ ic\ p\ (modifies\ (p!k))\ t * S\ ic\ p\ k\ t)))$
$+(\sum t{=}0..(q{-}1).\ b\hat{}(Suc\ t)*(\sum S0\ p\ d\ (\lambda k.\ (1\ -\ Z\ ic\ p\ (modifies\ (p!k))\ t) * S\ ic\ p\ k\ t)))$
by (*auto simp only*: *sum.distrib*)
also have ... $= 1 + b * (\sum t = 0..(q{-}1).\ (b\hat{}t * (\sum S{+}\ p\ d\ (\lambda k.\ S\ ic\ p\ k\ t))))$
$+ b*(\sum t{=}0..(q{-}1).\ b\hat{}t*(\sum S{-}\ p\ d\ (\lambda k.\ Z\ ic\ p\ (modifies\ (p!k))\ t * S\ ic\ p\ k\ t)))$
$+ b*(\sum t{=}0..(q{-}1).\ b\hat{}t*(\sum S0\ p\ d\ (\lambda k.\ (1\ -\ Z\ ic\ p\ (modifies\ (p!k))\ t) * S\ ic\ p\ k\ t)))$
by (*auto simp*: *algebra-simps sum-distrib-left*)
also have ... $= 1 + b * (\sum t = 0..(q{-}1).\ (\sum S{+}\ p\ d\ (\lambda k.\ b\hat{}t * S\ ic\ p\ k\ t)))$
$+ b*(\sum t{=}0..(q{-}1).\ (\sum S{-}\ p\ d\ (\lambda k.\ b\hat{}t * (Z\ ic\ p\ (modifies\ (p!k))\ t * S\ ic\ p\ k\ t))))$

+ b*(∑ t=0..(q−1). (∑ S0 p d (λk. b^t * ((1 − Z ic p (modifies (p!k)) t) * S ic p k t))))
using *sum-sadd-distrib sum-ssub-nzero-distrib sum-ssub-zero-distrib* **by** *auto*
also have ... = 1 + b * (∑ S+ p d (λk. ∑ t = 0..(q−1). b^t * S ic p k t))
+ b*(∑ S− p d (λk. ∑ t=0..(q−1). b^t * (Z ic p (modifies (p!k)) t * S ic p k t)))
+ b*(∑ S0 p d (λk. ∑ t=0..(q−1). b^t * ((1 − Z ic p (modifies (p!k)) t) * S ic p k t)))
using *sum-sadd-commutative sum-ssub-nzero-commutative sum-ssub-zero-commutative* **by** *auto*

finally have *eq1*: s d = 1 + b * (∑ S+ p d (λk. ∑ t = 0..q. b^t * S ic p k t))
+ b*(∑ S− p d (λk. ∑ t=0..q. b^t * (Z ic p (modifies (p!k)) t * S ic p k t)))
+ b*(∑ S0 p d (λk. ∑ t=0..q. b^t * ((1 − Z ic p (modifies (p!k)) t) * S ic p k t)))
using *add-q sub-nzero-q sub-zero-q* **by** *auto*
also have ... = 1 + b * (∑ S+ p d (λk. s k))
+ b*(∑ S− p d (λk. ∑ t=0..q. b^t * (Z ic p (modifies (p!k)) t * S ic p k t)))
+ b*(∑ S0 p d (λk. ∑ t=0..q. b^t * ((1 − Z ic p (modifies (p!k)) t) * S ic p k t)))
using *SKe-def s-def* **by** *auto*
finally have s d = 1 + b * (∑ S+ p d s)
+ b*(∑ S− p d (λk. ∑ t=0..q. b^t * (Z ic p (modifies (p!k)) t * S ic p k t)))
+ b*(∑ S0 p d (λk. ∑ t=0..q. b^t * ((1 − Z ic p (modifies (p!k)) t) * S ic p k t)))
by *auto*
also have ... = 1 + b * (∑ S+ p d s)
+ b*(∑ S− p d (λk. ZLe ic p b q (modifies (p!k)) && SKe ic p b q k))
+ b*(∑ S0 p d (λk. ∑ t=0..q. b^t * ((1 − Z ic p (modifies (p!k)) t) * S ic p k t)))
using *mult-to-bitAND c-gt-cells b-def c* **by** *auto*
finally have s d = 1 + b * (∑ S+ p d s)
+ b*(∑ S− p d (λk. ZLe ic p b q (modifies (p!k)) && SKe ic p b q k))
+ b*(∑ S0 p d (λk. (e − ZLe ic p b q (modifies (p!k))) && SKe ic p b q k))
using *mult-to-bitAND-state c-gt-cells b-def c e-def* **by** *auto*
thus *?thesis* **using** *s-def z-def* **by** *auto*
qed

lemma *halting-condition-04-27*:
fixes c :: *nat*
and l :: *register*
and ic :: *configuration*
and p :: *program*
and q :: *nat*
and a :: *nat*

defines $b == B\ c$
and $m == length\ p - 1$

assumes *is-val*: *is-valid-initial ic p a*
and *q*: $q > 0$

assumes *terminate*: *terminates ic p q*

shows $SKe\ ic\ p\ b\ q\ m = b\ \hat{}\ q$
proof −
have *halt*: *ishalt* (*p* ! (*fst* (*steps ic p q*)))
using *terminate terminates-def correct-halt-def* **by** *auto*
have $\forall k<length\ p - 1.\ \neg\ ishalt\ (p!k)$ **using** *is-val is-valid-initial-def*[*of ic p a*] *is-valid-def*[*of ic p*] *program-includes-halt.simps* **by** *blast*
hence $ishalt\ (p!k) \Longrightarrow k \geq length\ p - 1$ **for** *k* **using** *not-le-imp-less* **by** *auto*
hence *gt*: $fst\ (steps\ ic\ p\ q) \geq m$ **using** *halt m-def* **by** *auto*
have $fst\ (steps\ ic\ p\ q) \leq m$
using *p-contains*[*of ic p a q*] *is-val m-def* **by** *auto*
hence *q-steps-m*: $fst\ (steps\ ic\ p\ q) = m$ **using** *gt* **by** *auto*
hence *1*: $S\ ic\ p\ m\ q = 1$ **using** *S-def* **by** *auto*

have *ishalt* (*p*!*m*) **using** *q-steps-m halt* **by** *auto*
have $\forall t<q.\ \neg\ ishalt\ (p\ !\ (fst\ (steps\ ic\ p\ t)))$ **using** *terminate terminates-def* **by** *auto*
hence $\forall t<q.\ \neg\ (fst\ (steps\ ic\ p\ t) = m)$ **using** ‹*ishalt* (*p*!*m*)› **by** *auto*
hence *0*: $t \leq q-1 \Longrightarrow S\ ic\ p\ m\ t = 0$ **for** *t* **using** *q S-def*[*of ic p m t*] **by** *auto*

have $SKe\ ic\ p\ b\ q\ m = (\sum t = 0..(Suc\ (q-1)).\ b\ \hat{}\ t * S\ ic\ p\ m\ t)$ **by** (*auto simp*: *q SKe-def*)
also have ... $= (\sum t = 0..(q-1).\ b\hat{}t * S\ ic\ p\ m\ t) + b\ \hat{}(Suc\ (q-1)) * S\ ic\ p\ m\ (Suc\ (q-1))$
by *auto*
also have ... $= b\ \hat{}\ q$ **using** *0 1 q* **by** *auto*
finally show *?thesis* **by** *auto*
qed

lemma *state-q-bound*:
fixes *c* :: *nat*
and *l* :: *register*
and *ic* :: *configuration*
and *p* :: *program*
and *q* :: *nat*
and *a* :: *nat*

defines $b == B\ c$
and $m == length\ p - 1$

assumes *is-val*: *is-valid-initial ic p a*

and *q*: $q > 0$
and *terminate*: *terminates ic p q*
and *c*: $c > 0$

assumes $k<m$

shows *SKe ic p b q k* $<$ *b* ^ *q*
proof −
from *b-def* **have** $b>1$ **using** *B-def* **apply** *auto*
by (*metis One-nat-def one-less-numeral-iff power-gt1-lemma semiring-norm*(*76*))
hence *b*: $b > 2$ **using** *c b-def B-def*
by (*smt One-nat-def Suc-le-lessD less-Suc-eq-le less-trans-Suc linorder-neqE-nat numeral-2-eq-2 power-Suc0-right power-inject-exp*)
from ‹$k<m$› **have** ¬ *ishalt* (*p*!*k*) **using** *is-val*
by (*simp add*: *is-valid-def is-valid-initial-def is-val m-def*)
hence *S ic p k q* = *0* **using** *terminate terminates-def correct-halt-def S-def* **by** *auto*
hence *SKe ic p b q k* = ($\sum$ *t* = *0*..*q*−*1*. *b* ^ *t* ∗ *S ic p k t*)
using ‹$q>0$› **apply** (*auto cong*: *sum.cong simp*: *SKe-def*) **by** (*metis* (*no-types*, *lifting*) *Suc-pred*
add-cancel-right-right mult-0-right sum.atLeast0-atMost-Suc)
also have ... ≤ ($\sum$ *t* = *0*..*q*−*1*. *b*^*t*) **by** (*auto simp add*: *S-def gr-implies-not0 sum-mono*)
also have ... < *b* ^ *q*
using ‹$q>0$› *sum-bt*
by (*metis Suc-diff-1 b*)

finally show *?thesis* **by** *auto*
qed

end

3.6 Masking properties

theory *MachineMasking*
imports *RegisterMachineSimulation* ../*Diophantine*/*Binary-And*
begin

definition *E* :: *nat* ⇒ *nat* ⇒ *nat* **where**
(*E q b*) = ($\sum$ *t* = *0*..*q*. *b*^*t*)

lemma *e-geom-series*:
assumes $b \geq 2$
shows (*E q b* = *e*) ⟷ ((*b*−*1*) ∗ *e* = *b*^(*Suc q*) − *1*) (**is** *?P* ⟷ *?Q*)
proof−
have *sum* ((^) (*int b*)) {..<*Suc q*} = *sum* ((^) *b*) {*0*..*q*} **by** (*simp add*: *atLeast0AtMost lessThan-Suc-atMost*)
then have (*int b* − *1*) ∗ (*E q b*) = *int b* ^ *Suc q* − *1*

using *E-def* **by** (*metis power-diff-1-eq*)
moreover have *int b ^ Suc q − 1 = b ^ (Suc q) − 1* **using** *one-le-power*[*of int b Suc q*] *assms*
by (*simp add*: *of-nat-diff*)
moreover have *int b − 1 = b − 1* **using** *assms* **by** *auto*
ultimately show *?thesis* **using** *assms*
by (*metis Suc-1 Suc-diff-le Zero-not-Suc diff-Suc-Suc int-ops*(*7*) *mult-cancel-left of-nat-eq-iff*)
qed

definition *D* :: *nat ⇒ nat ⇒ nat ⇒ nat* **where**
(*D q c b*) = (∑ *t* = *0*..*q*. (*2^c − 1*) ∗ *b^t*)

lemma *d-geom-series*:
assumes *b* = *2^*(*Suc c*)
shows (*D q c b* = *d*) ⟷ ((*b*−*1*) ∗ *d* = (*2^c − 1*) ∗ (*b^*(*Suc q*) − *1*)) (**is** *?P* ⟷ *?Q*)
proof−
have *D q c b* = (*2^c − 1*) ∗ *E q b* **by** (*auto simp*: *E-def D-def sum-distrib-left sum-distrib-right*)
moreover have *b* ≥ *2* **using** *assms* **by** *fastforce*
ultimately show *?thesis* **by** (*smt e-geom-series mult.left-commute mult-cancel-left*)
qed

definition *F* :: *nat ⇒ nat ⇒ nat ⇒ nat* **where**
(*F q c b*) = (∑ *t* = *0*..*q*. *2^c* ∗ *b^t*)

lemma *f-geom-series*:
assumes *b* = *2^*(*Suc c*)
shows (*F q c b* = *f*) ⟷ ((*b*−*1*) ∗ *f* = *2^c* ∗ (*b^*(*Suc q*) − *1*))
proof−
have *F q c b* = *2^c* ∗ *E q b* **by** (*auto simp*: *E-def F-def sum-distrib-left sum-distrib-right*)
moreover have *b* ≥ *2* **using** *assms* **by** *fastforce*
ultimately show *?thesis* **by** (*smt e-geom-series mult.left-commute mult-cancel-left*)
qed

lemma *aux-lt-implies-mask*:
assumes *a* < *2^k*
shows ∀ *r*≥*k*. *a* ¡ *r* = *0*
using *nth-bit-def assms* **apply** *auto*
proof −
fix *r* :: *nat*
assume *a1*: *a* < *2 ^ k*
assume *a2*: *k* ≤ *r*

from *a1* **have** *a div 2 ^ k = 0*
  by *simp*
then have *2 = (0::nat) ∨ a < 2 ^ r*
  using *a2* **by** (*metis* (*no-types*) *div-le-mono nat-zero-less-power-iff neq0-conv not-le power-diff*)
then show *a div 2 ^ r mod 2 = 0*
  by *simp*
qed

lemma *lt-implies-mask*:
fixes *a b* :: *nat*
assumes ∃*k. a < 2^k* ∧ (∀*r<k. nth-bit b r = 1*)
shows *a* ⪯ *b*
proof −
obtain *k* **where** *assms*: *a < 2^k* ∧ (∀*r<k. nth-bit b r = 1*) **using** *assms* **by** *auto*
have *k1*: ∀*r<k. a* ¡ *r* ≤ *b* ¡ *r* **using** *nth-bit-bounded*
  by (*simp add*: ‹*a < 2 ^ k* ∧ (∀*r<k. b* ¡ *r = 1*)›)
hence *k2*: ∀*r≥k. a* ¡ *r = 0* **using** *aux-lt-implies-mask assms* **by** *auto*
show *?thesis* **using** *masks-leq-equiv*
  by *auto* (*metis k1 k2 le0 not-less*)
qed

lemma *mask-conversed-shift*:
fixes *a b k* :: *nat*
assumes *asm*: *a* ⪯ *b*
shows *a ∗ 2^k* ⪯ *b ∗ 2^k*
proof −
have *shift*: *x* ⪯ *y* ⟹ *2∗x* ⪯ *2∗y* **for** *x y* **by** (*induction x*; *auto*)
have *a ∗ 2 ^ k* ⪯ *b ∗ 2 ^ k* ⟹ *2 ∗* (*a ∗ 2 ^ k*) ⪯ *2 ∗* (*b ∗ 2 ^ k*) **for** *k*
  using *shift*[*of a∗2^k b∗2^k*] **by** *auto*
thus *?thesis* **by** (*induction k*; *auto simp*: *asm shift algebra-simps*)
qed

lemma *base-summation-bound*:
fixes *c q* :: *nat*
  and *f* :: (*nat* ⇒ *nat*)

defines *b*: *b* ≡ *B c*
assumes *bound*: ∀*t. f t < 2 ^ Suc c* − (*1*::*nat*)

shows (∑ *t = 0..q. f t ∗ b^t*) < *b^*(*Suc q*)
proof (*induction q*)
case *0*
then show *?case* **using** *B-def b bound less-imp-diff-less not-less-eq*
  by *auto blast*
next
case (*Suc q*)
have (∑ *t = 0..Suc q. f t ∗ b ^ t*) = *f* (*Suc q*) *∗ b ^* (*Suc q*) + (∑ *t = 0..q. f t*

```
* b ^ t)
    by (auto cong: sum.cong)
  also have ... < (f (Suc q) + 1) * b ^(Suc q)
    using Suc.IH by auto
  also have ... < b * b ^(Suc q)
   by (metis bound b less-diff-conv B-def mult-less-cancel2 zero-less-numeral zero-less-power)
  finally show ?case by auto
qed

lemma mask-conserved-sum:
  fixes y c q :: nat
    and x :: (nat ⇒ nat)

defines b: b ≡ B c
assumes mask: ∀ t. x t ≼ y
assumes xlt: ∀ t. x t ≤ 2 ^ c − Suc 0
assumes ylt: y ≤ 2 ^ c − Suc 0

shows (∑ t = 0..q. x t * b^t) ≼ (∑ t = 0..q. y * b^t)
proof (induction q)
  case 0
  then show ?case
    using mask by auto
next
  case (Suc q)

  have xb: ∀ t. x t < 2^Suc c − Suc 0
    using xlt
     by (smt Suc-pred leI le-imp-less-Suc less-SucE less-trans n-less-m-mult-n nu-
meral-2-eq-2
        power.simps(2) zero-less-numeral zero-less-power)
  have yb: y < 2^c
    using ylt b B-def leI order-trans by fastforce

  have sumxlt: (∑ t = 0..q. x t * b ^ t) < b^(Suc q)
    using base-summation-bound xb b B-def by auto
  have sumylt: (∑ t = 0..q. y * b ^ t) < b^(Suc q)
    using base-summation-bound yb b B-def by auto

  have ((∑ t = 0..Suc q. x t * b ^ t) ≼ (∑ t = 0..Suc q. y * b ^ t))
      = (x (Suc q) * b^Suc q + (∑ t = 0..q. x t * b ^ t) ≼
              y * b^Suc q + (∑ t = 0..q. y * b ^ t))
    by (auto simp: atLeast0-lessThan-Suc add.commute)
  also have ... = (x (Suc q) * b^Suc q ≼ y * b^Suc q)
              ∧ (∑ t = 0..q. x t * b ^ t) ≼ (∑ t = 0..q. y * b ^ t)
    using mask-linear[where ?t = Suc c * Suc q] sumxlt sumylt Suc.IH b B-def
    apply auto
   apply (smt mask mask-conversed-shift power-Suc power-mult power-mult-distrib)
    by (smt mask mask-linear power-Suc power-mult power-mult-distrib)
```

```
  finally show ?case using mask-linear Suc.IH B-def
    by (metis (no-types, lifting) b mask mask-conversed-shift power-mult)
qed

lemma aux-powertwo-digits:
  fixes k c :: nat
  assumes k < c
  shows nth-bit (2^c) k = 0
proof −
  have h: (2::nat) ^ c div 2 ^ k = 2 ^(c − k)
    by (simp add: assms less-imp-le power-diff)
  thus ?thesis
    by (auto simp: h nth-bit-def assms)
qed

lemma obtain-digit-rep:
  fixes x c :: nat
  shows x && 2^c = (∑ t<Suc c. 2^t ∗ (nth-bit x t) ∗ (nth-bit (2^c) t))
proof −
  have x && 2^c ⪯ 2^c by (simp add: lm0245)
  hence x && 2^c ≤ 2^c by (simp add: masks-leq)
  hence h: x && 2^c < 2^Suc c
     by (smt Suc-lessD le-neq-implies-less lessI less-trans-Suc n-less-m-mult-n nu-
meral-2-eq-2
        power-Suc zero-less-power)
  have ∀ t. (x && 2^c) ¡ t = (nth-bit x t) ∗ (nth-bit (2^c) t)
    using bitAND-digit-mult by auto
  then show ?thesis using h digit-sum-repr[of (x && 2^c) Suc c]
    by (auto) (simp add: mult.commute semiring-normalization-rules(19))
qed

lemma nth-digit-bitAND-equiv:
  fixes x c :: nat
  shows 2^c ∗ nth-bit x c = (x && 2^c)
proof −
  have d1: nth-bit (2^c) c = 1
    using nth-bit-def by auto

  moreover have x && 2^c = (2::nat)^c ∗ (x ¡ c) ∗ (((2::nat)^c) ¡ c)
           + (∑ t<c. (2::nat)^t ∗ (x ¡ t) ∗ (((2::nat)^c) ¡ t))
    using obtain-digit-rep by (auto cong: sum.cong)

  moreover have (∑ t<c. 2^t ∗ (nth-bit x t) ∗ (nth-bit ((2::nat)^c) t)) = 0
    using aux-powertwo-digits by auto

  ultimately show ?thesis using d1
    by auto
qed
```

```
lemma bitAND-single-digit:
  fixes x c :: nat
assumes 2 ^ c ≤ x
assumes x < 2 ^ Suc c

shows nth-bit x c = 1
proof −
  obtain b where b: x = 2^c + b
    using assms(1) le-Suc-ex by auto
  have bb: b < 2^c
    using assms(2) b by auto
  have (2 ^ c + b) div 2 ^ c mod 2 = (1 + b div 2 ^ c) mod 2
    by (auto simp: div-add-self1)
  also have ... = 1
    by (auto simp: bb)
  finally show ?thesis
    by (simp only: nth-bit-def b)
qed

lemma aux-bitAND-distrib: 2 ∗ (a && b) = (2 ∗ a) && (2 ∗ b)
  by (induct a b rule: bitAND-nat.induct; auto)

lemma bitAND-distrib: 2^c ∗ (a && b) = (2^c ∗ a) && (2^c ∗ b)
proof (induction c)
  case 0
  then show ?case by auto
next
  case (Suc c)
  have 2 ^ Suc c ∗ (a && b) = 2 ∗ (2 ^ c ∗ (a && b)) by auto
  also have ... = 2 ∗ ((2^c ∗ a) && (2^c ∗ b)) using Suc.IH by auto
  also have ... = ((2^Suc c ∗ a) && (2^Suc c ∗ b))
    using aux-bitAND-distrib[of 2^c ∗ a 2^c ∗ b]
    by (auto simp add: ab-semigroup-mult-class.mult-ac(1))
  finally show ?case by auto
qed

lemma bitAND-linear-sum:
  fixes x y :: nat ⇒ nat
    and c :: nat
    and q :: nat

defines b: b == 2 ^ Suc c

assumes xb: ∀ t. x t < 2 ^ Suc c − 1
assumes yb: ∀ t. y t < 2 ^ Suc c − 1

shows (∑ t = 0..q. (x t && y t) ∗ b^t) =
       (∑ t = 0..q. x t ∗ b^t) && (∑ t = 0..q. y t ∗ b^t)
proof (induction q)
```

case *0*
then show *?case*
by (*auto simp*: *b B-def*)
next
case (*Suc q*)
have $(\sum t = 0..Suc\ q.\ (x\ t\ \&\&\ y\ t) * b\ \hat{}\ t) = (x\ (Suc\ q)\ \&\&\ y\ (Suc\ q)) * b\ \hat{}$
$Suc\ q$
$+ (\sum t = 0..q.\ (x\ t\ \&\&\ y\ t) * b\ \hat{}\ t)$
by (*auto cong*: *sum.cong*)

moreover have *h0*: $(x\ (Suc\ q)\ \&\&\ y\ (Suc\ q)) * b\ \hat{}\ Suc\ q$
$= (x\ (Suc\ q) * b\hat{}Suc\ q)\ \&\&\ (y\ (Suc\ q) * b\hat{}Suc\ q)$
using *b bitAND-distrib* **by** (*auto*) (*smt mult.commute power-Suc power-mult*)

moreover have *h1*: $(\sum t = 0..q.\ (x\ t\ \&\&\ y\ t) * b\ \hat{}\ t)$
$= (\sum t = 0..q.\ x\ t * b\hat{}t)\ \&\&\ (\sum t = 0..q.\ y\ t * b\hat{}t)$
using *Suc.IH* **by** *auto*

ultimately have *h2*: $(\sum t = 0..Suc\ q.\ (x\ t\ \&\&\ y\ t) * b\ \hat{}\ t)$
$= ((x\ (Suc\ q) * b\hat{}Suc\ q)\ \&\&\ (y\ (Suc\ q) * b\hat{}Suc\ q))$
$+ ((\sum t = 0..q.\ x\ t * b\hat{}t)\ \&\&\ (\sum t = 0..q.\ y\ t * b\hat{}t))$
by *auto*

have *sumxb*: $(\sum t = 0..q.\ x\ t * b\ \hat{}\ t) < b\ \hat{}\ Suc\ q$
using *base-summation-bound xb b B-def* **by** *auto*
have *sumyb*: $(\sum t = 0..q.\ y\ t * b\ \hat{}\ t) < b\ \hat{}\ Suc\ q$
using *base-summation-bound yb b B-def* **by** *auto*

have *h3*: $((x\ (Suc\ q) * b\hat{}Suc\ q)\ \&\&\ (y\ (Suc\ q) * b\hat{}Suc\ q))$
$+ ((\sum t = 0..q.\ x\ t * b\hat{}t)\ \&\&\ (\sum t = 0..q.\ y\ t * b\hat{}t))$
$= ((\sum t = 0..q.\ x\ t * b\hat{}t) + x\ (Suc\ q) * b\hat{}Suc\ q)$
$\&\&\ ((\sum t = 0..q.\ y\ t * b\hat{}t) + y\ (Suc\ q) * b\hat{}Suc\ q)$
using *sumxb sumyb bitAND-linear h2 h0*
by (*auto*) (*smt add.commute b power-Suc power-mult*)

thus *?case* **using** *h2* **by** (*auto cong*: *sum.cong*)
qed

lemma *dmask-aux0*:
fixes $x :: nat$
assumes $x > 0$
shows $(2\ \hat{}\ x - Suc\ 0)\ div\ 2 = 2\ \hat{}\ (x - 1) - Suc\ 0$
proof −
have *0*: $(2\hat{}x - Suc\ 0)\ div\ 2 = (2\hat{}x - 2)\ div\ 2$
by (*smt Suc-diff-Suc Suc-pred assms dvd-power even-Suc even-Suc-div-two nat-power-eq-Suc-0-iff*
neq0-conv numeral-2-eq-2 zero-less-diff zero-less-power)

moreover have *divides*: $(2::nat)\ dvd\ 2\hat{}x$

```
    by (simp add: assms dvd-power[of x 2::nat])
  moreover have (2^x − 2::nat) div 2 = 2^x div 2 − 2 div 2
    using div-plus-div-distrib-dvd-left[of 2 2^x 2] divides
    by auto
  moreover have ... = 2 ^(x − 1) − Suc 0
    by (simp add: Suc-leI assms power-diff)
  ultimately have 1: (2 ^x − Suc 0) div 2 = 2 ^(x − 1) − Suc 0
     by (smt One-nat-def)
  thus ?thesis by simp
qed

lemma dmask-aux:
  fixes c :: nat
  shows d < c ⟹ (2^c − Suc 0) div 2^d = 2 ^(c − d) − Suc 0
proof (induction d)
  case 0
  then show ?case by (auto)
next
  case (Suc d)
  have d: d < c using Suc.prems by auto
  have (2 ^ c − Suc 0) div 2 ^Suc d = (2 ^ c − Suc 0) div 2 ^ d div 2
    by (auto) (metis mult.commute div-mult2-eq)
  also have ... = (2 ^(c − d) − Suc 0) div 2
    by (subst Suc.IH) (auto simp: d)
  also have ... = 2 ^(c − Suc d) − Suc 0
    apply (subst dmask-aux0[of c − d])
    using d by (auto)
  finally show ?case by auto
qed



lemma register-cells-masked:
  fixes l :: register
    and t :: nat
    and ic :: configuration
    and p :: program

assumes cells-bounded: cells-bounded ic p c
assumes l: l < length (snd ic)

shows R ic p l t ⪯ 2^c − 1
proof −
  have a: R ic p l t ≤ 2^c − 1 using cells-bounded less-Suc-eq-le
    using l by fastforce
  have b: r < c ⟹ nth-bit (2^c − 1) r = 1 for r
    apply (auto simp: nth-bit-def)
    apply (subst dmask-aux)
    apply (auto)
```

by (*metis Suc-pred dvd-power even-Suc mod-0-imp-dvd not-mod2-eq-Suc-0-eq-0 zero-less-diff zero-less-numeral zero-less-power*)
show *?thesis* **using** *lt-implies-mask cells-bounded l*
by (*auto*) (*metis One-nat-def b*)
qed

lemma *lm04-15-register-masking*:
fixes *c* :: *nat*
and *l* :: *register*
and *ic* :: *configuration*
and *p* :: *program*
and *q* :: *nat*

defines *b* == *B c*
defines *d* == *D q c b*

assumes *cells-bounded*: *cells-bounded ic p c*
assumes *l*: *l* < *length* (*snd ic*)

defines *r* == *RLe ic p b q*

shows *r l* ⪯ *d*
proof −
have ⋀*t*. *R ic p l t* ⪯ *2^c* − *1* **using** *cells-bounded l*
by (*rule register-cells-masked*)
hence *rmasked*: ∀ *t*. *R ic p l t* ⪯ *2^c* − *1*
by (*intro allI*)

have *rlt*: ∀ *t*. *R ic p l t* ≤ *2^c* − *1*
using *cells-bounded less-Suc-eq-le l* **by** *fastforce*

have *rlmasked*: (∑ *t* = *0*..*q*. *R ic p l t* ∗ *b^t*) ⪯ (∑ *t* = *0*..*q*. (*2^c* − *1*) ∗ *b^t*)
using *rmasked rlt b-def B-def mask-conserved-sum* **by** (*auto*)

thus *?thesis*
by (*auto simp*: *r-def d-def D-def RLe-def mult.commute cong*: *sum.cong*)
qed

lemma *zero-cells-masked*:
fixes *l* :: *register*
and *t* :: *nat*
and *ic* :: *configuration*
and *p* :: *program*

assumes *l*: *l* < *length* (*snd ic*)

shows *Z ic p l t* ⪯ *1*
proof −

have *nth-bit 1 0 = 1* **by** (*auto simp*: *nth-bit-def*)
thus *?thesis* **apply** (*auto*) **apply** (*rule lt-implies-mask*)
by (*metis* (*full-types*) *One-nat-def Suc-1 Z-bounded less-Suc-eq-le less-one power-one-right*)
qed

lemma *lm04-15-zero-masking*:
fixes *c* :: *nat*
and *l* :: *register*
and *ic* :: *configuration*
and *p* :: *program*
and *q* :: *nat*

defines *b* == *B c*
defines *e* == *E q b*

assumes *cells-bounded*: *cells-bounded ic p c*
assumes *l*: *l* < *length* (*snd ic*)
assumes *c*: *c* > *0*

defines *z* == *ZLe ic p b q*

shows $z\ l \preceq e$
proof −
have $\bigwedge t.\ Z\ ic\ p\ l\ t \preceq 1$ **using** *l*
by (*rule zero-cells-masked*)
hence *zmasked*: $\forall t.\ Z\ ic\ p\ l\ t \preceq 1$
by (*intro allI*)

have *zlt*: $\forall t.\ Z\ ic\ p\ l\ t \leq 2\ \hat{}\ c - 1$
using *cells-bounded less-Suc-eq-le* **by** *fastforce*

have *1*: (*1*::*nat*) $\leq 2\ \hat{}\ c - 1$ **using** *c*
by (*simp add*: *Nat.le-diff-conv2 numeral-2-eq-2 self-le-power*)

have *rlmasked*: $(\sum t = 0..q.\ Z\ ic\ p\ l\ t * b\hat{}t) \preceq (\sum t = 0..q.\ 1 * b\hat{}t)$
using *zmasked zlt 1 b-def B-def mask-conserved-sum*[*of Z ic p l 1*]
by (*auto*)

thus *?thesis*
by (*auto simp*: *z-def e-def E-def ZLe-def mult.commute cong*: *sum.cong*)
qed

lemma *lm04-19-zero-register-relations*:
fixes *c* :: *nat*
and *l* :: *register*
and *t* :: *nat*
and *ic* :: *configuration*
and *p* :: *program*

assumes *cells-bounded*: *cells-bounded ic p c*
assumes *l*: *l* < *length* (*snd ic*)

defines *z* == *Z ic p*
defines *r* == *R ic p*

shows *2^c* ∗ *z l t* = (*r l t* + *2^c* − *1*) && *2^c*
proof −
  have *a1*: *R ic p l t* ≠ *0* ⟹ *2^c* ≤ *R ic p l t* + *2^c* − *1*
    by *auto*
  have *a2*: *R ic p l t* + *2^c* − *1* < *2^Suc c* **using** *cells-bounded*
    by (*simp add*: *l less-imp-diff-less*)

  have *Z ic p l t* = *nth-bit* (*R ic p l t* + *2^c* − *1*) *c*
    apply (*cases R ic p l t* = *0*)
    subgoal by (*auto simp add*: *Z-def R-def nth-bit-def*)
    subgoal using *cells-bounded bitAND-single-digit a1 a2 Z-def*
      by *auto*
    done

  also have *2^c* ∗ *nth-bit* (*R ic p l t* + *2^c* − *1*) *c* = ((*R ic p l t* + *2^c* − *1*) &&
2^c)
    using *nth-digit-bitAND-equiv* **by** *auto*

  finally show *?thesis* **by** (*auto simp*: *z-def r-def*)
qed

lemma *lm04-20-zero-definition*:
  fixes *c* :: *nat*
    and *l* :: *register*
    and *ic* :: *configuration*
    and *p* :: *program*
    and *q* :: *nat*

defines *b* == *B c*
defines *f* == *F q c b*
defines *d* == *D q c b*

assumes *cells-bounded*: *cells-bounded ic p c*
assumes *l*: *l* < *length* (*snd ic*)

assumes *c*: *c* > *0*

defines *z* == *ZLe ic p b q*
defines *r* == *RLe ic p b q*

shows *2^c* ∗ *z l* = (*r l* + *d*) && *f*
proof −

have ⋀t. 2^c ∗ Z ic p l t = (R ic p l t + 2^c − 1) && 2^c
by (rule lm04-19-zero-register-relations cells-bounded l) +
hence raw-sums: (∑ t = 0..q. 2^c ∗ Z ic p l t ∗ b^t)
= (∑ t = 0..q. ((R ic p l t + 2^c − 1) && 2^c) ∗ b^t)
by auto

have (∑ t = 0..q. 2^c ∗ Z ic p l t ∗ b^t) = 2^c ∗ (∑ t = 0..q. Z ic p l t ∗ b^t)
by (auto simp: sum-distrib-left mult.assoc cong: sum.cong)
also have ... = 2^c ∗ z l
by (auto simp: z-def ZLe-def mult.commute)
finally have lhs: (∑ t = 0..q. 2^c ∗ Z ic p l t ∗ b^t) = 2^c ∗ z l
by auto

have (∑ t = 0..q. (R ic p l t + (2^c − 1)) ∗ b^t)
= (∑ t = 0..q. R ic p l t ∗ b^t + (2^c − 1) ∗ b^t)
apply (rule sum.cong)
apply (auto simp: add.commute mult.commute)
subgoal for x **using** distrib-left[of b^x R ic p l x 2^c − 1] **by** (auto simp: algebra-simps)
done
also have ... = (∑ t = 0..q. (R ic p l t ∗ b^t)) + (∑ t = 0..q. (2^c − 1) ∗ b^t)
by (rule sum.distrib)
also have ... = r l + d
by (auto simp: r-def RLe-def d-def D-def mult.commute)
finally have split-sums: (∑ t = 0..q. (R ic p l t + (2^c − 1)) ∗ b^t) = r l + d
by auto

have a1: (2::nat) ^ c < (2::nat) ^ Suc c − 1 **using** c **by** (induct c, auto, fastforce)
have a2: ∀ t. R ic p l t + 2 ^ c − 1 ≤ 2 ^ Suc c **using** cells-bounded B-def
by (simp add: less-imp-diff-less l) (simp add: Suc-leD l less-imp-le-nat numeral-Bit0)
have (∑ t = 0..q. ((R ic p l t + 2^c − 1) && 2^c) ∗ b^t)
= (∑ t = 0..q. (R ic p l t + 2^c − 1) ∗ b^t) && (∑ t = 0..q. 2^c ∗ b^t)
using bitAND-linear-sum[of λt. R ic p l t + 2^c − 1 c λt. 2^c]
cells-bounded b-def B-def a1 a2
apply auto
by (smt One-nat-def Suc-less-eq Suc-pred a1 add.commute add-gr-0 l mult-2
nat-add-left-cancel-less power-Suc zero-less-numeral zero-less-power)
also have ... = (∑ t = 0..q. (R ic p l t + 2^c − 1) ∗ b^t) && f
by (auto simp: f-def F-def)
also have ... = (r l + d) && f **using** split-sums
by auto
finally have rhs: (∑ t = 0..q. ((R ic p l t + 2^c − 1) && 2^c) ∗ b^t) = (r l + d) && f
by auto

show ?thesis **using** raw-sums lhs rhs
by auto

qed

lemma *state-mask*:
fixes *c* :: *nat*
  and *l* :: *register*
  and *ic* :: *configuration*
  and *p* :: *program*
  and *q* :: *nat*
  and *a* :: *nat*

defines $b \equiv B\ c$
    and $m \equiv$ *length* $p - 1$

defines $e \equiv E\ q\ b$

assumes *is-val*: *is-valid-initial ic p a*
    and *q*: $q > 0$
    and $c > 0$

assumes *terminate*: *terminates ic p q*
  shows *SKe ic p b q k* $\preceq$ *e*
proof −
  have $1 \leq 2 \hat{}\ c - Suc\ 0$ **using** ‹c>0› **by** (*metis One-nat-def Suc-leI one-less-numeral-iff*

one-less-power semiring-norm(76) zero-less-diff)
  have *Smask*: *S ic p k t* $\preceq 1$ **for** *t* **by** (*simp add*: *S-def*)
  have *Sbound*: *S ic p k t* $\leq 2 \hat{}\ c - Suc\ 0$ **for** *t* **using** ‹1≤2^c−Suc 0› **by** (*simp add*: *S-def*)
  have *rlmasked*: $(\sum t = 0..q.\ S\ ic\ p\ k\ t * b\hat{}t) \preceq (\sum t = 0..q.\ 1 * b\hat{}t)$
    using *b-def B-def Smask Sbound mask-conserved-sum*[*of S ic p k 1*] ‹1 ≤ 2^c−Suc 0› **by** *auto*

  thus *?thesis* **using** *SKe-def e-def E-def* **by** (*auto simp*: *mult.commute*)
qed

lemma *state-sum-mask*:
fixes *c* :: *nat*
  and *l* :: *register*
  and *ic* :: *configuration*
  and *p* :: *program*
  and *q* :: *nat*
  and *a* :: *nat*

defines $b \equiv B\ c$
    and $m \equiv$ *length* $p - 1$

defines $e \equiv E\ q\ b$

assumes *is-val*: *is-valid-initial ic p a*

and *q*: *q > 0*
and *c > 0*
and *b > 1*

assumes *M≤m*

assumes *terminate*: *terminates ic p q*
shows (∑ *k≤M. SKe ic p b q k*) ⪯ *e*
proof −
have *e-aux*: *nth-digit e t b* = (*if t≤q then 1 else 0*) **for** *t*
unfolding *e-def E-def b-def B-def*
using ‹*b>1*› *b-def nth-digit-gen-power-series*[*of λk. Suc 0 c q*]
by (*auto simp*: *b-def B-def*)

have *state-unique*: ∀ *k≤m. S ic p k t = 1* ⟶ (∀ *j≠k. S ic p j t = 0*) **for** *t*
using *S-def* **by** (*induction t*, *auto*)

have *h1*: ∀ *t. nth-digit* (∑ *k≤M. SKe ic p b q k*) *t b* ≤ (*if t≤q then 1 else 0*)
proof − **{**
fix *t*
have *aux-bound-1*: (∑ *k≤M. S ic p k t′*) ≤ *1* **for** *t′*
proof (*cases* ∃ *k≤M. S ic p k t′ = 1*)
case *True*
then obtain *k* **where** *k*: *k≤M* ∧ *S ic p k t′ = 1* **by** *auto*
moreover have ∀ *j≤M. j ≠ k* ⟶ *S ic p j t′ = 0*
using *state-unique* ‹*M<=m*› *k S-def*
by (*auto*) (*presburger*)
ultimately have (∑ *k≤M. S ic p k t′*) = *1*
using *S-def* **by** *auto*
then show *?thesis*
by *auto*
next
case *False*
then show *?thesis* **using** *S-bounded*
by (*auto*) (*metis* (*no-types*, *lifting*) *S-def atMost-iff eq-imp-le le-SucI sum-nonpos*)
qed
hence *aux-bound-2*: ⋀*t′*. (∑ *k≤M. S ic p k t′*) < *2^c*
by (*metis Suc-1* ‹*c>0*› *le-less-trans less-Suc-eq one-less-power*)

have *h2*: (∑ *k≤M. SKe ic p b q k*) = (∑ *t = 0..q.* ∑ *k≤M. b ^ t ∗ S ic p k t*)
unfolding *SKe-def* **using** *sum.swap* **by** *auto*
hence (∑ *k≤M. SKe ic p b q k*) = (∑ *t = 0..q. b^t ∗* (∑ *k≤M. S ic p k t*))
unfolding *SKe-def* **by** (*simp add*: *sum-distrib-left*)

hence *nth-digit* (∑ *k≤M. SKe ic p b q k*) *t b* = (*if t≤q then* (∑ *k≤M. S ic p k t*) *else 0*)
using ‹*c>0*› *aux-bound-2 h2* **unfolding** *SKe-def*

using *nth-digit-gen-power-series*[*of* λt. ($\sum k \leq M$. *S ic p k t*) *c q t*]
by (*smt B-def Groups.mult-ac*(*2*) *assms*(*7*) *aux-bound-1 b-def le-less-trans sum.cong*)
hence *nth-digit* ($\sum k \leq M$. *SKe ic p b q k*) *t b* $\leq$ (*if* $t \leq q$ *then 1 else 0*)
using *aux-bound-1* **by** *auto*
} **thus** *?thesis* **by** *auto*
qed
moreover have $\forall t > q$. *nth-digit* ($\sum k \leq M$. *SKe ic p b q k*) *t b* = *0*
by (*metis* (*full-types*) *h1 le-0-eq not-less*)
ultimately have $\forall t$. $\forall i <$ *Suc c*. *nth-digit* ($\sum k \leq M$. *SKe ic p b q k*) *t b* ¡ *i*
$\leq$ *nth-digit e t b* ¡ *i*
using *aux-lt-implies-mask linorder-neqE-nat e-aux*
by (*smt One-nat-def le-0-eq le-SucE less-or-eq-imp-le nat-zero-less-power-iff numeral-2-eq-2 zero-less-Suc*)

hence $\forall t$. $\forall i <$ *Suc c*. ($\sum k \leq M$. *SKe ic p b q k*) ¡ (*Suc c* $*$ *t* + *i*) $\leq$ *e* ¡ (*Suc c* $*$ *t* + *i*)
using *digit-gen-pow2-reduct*[**where** *?c* = *Suc c* **and** *?a* = ($\sum k \leq M$. *SKe ic p b q k*)]
using *digit-gen-pow2-reduct*[**where** *?c* = *Suc c* **and** *?a* = *e*]
by (*simp add*: *b-def B-def*)
moreover have $\forall j$. $\exists t\ i$. *i* < *Suc c* $\wedge$ *j* = (*Suc c* $*$ *t* + *i*)
using *mod-less-divisor zero-less-Suc*
by (*metis add.commute mod-mult-div-eq*)
ultimately have $\forall j$. ($\sum k \leq M$. *SKe ic p b q k*) ¡ *j* $\leq$ *e* ¡ *j*
by *metis*

thus *?thesis*
using *masks-leq-equiv* **by** *auto*
qed

end

4 Arithmetization of Register Machines

4.1 A first definition of the arithmetizing equations

theory *MachineEquations*
imports *MultipleStepRegister MultipleStepState MachineMasking*
begin

definition *mask-equations* :: *nat* $\Rightarrow$ (*register* $\Rightarrow$ *nat*) $\Rightarrow$ (*register* $\Rightarrow$ *nat*) $\Rightarrow$ *nat* $\Rightarrow$ *nat* $\Rightarrow$ *nat* $\Rightarrow$ *nat* $\Rightarrow$ *bool*
where (*mask-equations n r z c d e f*) = (($\forall l < n$. (*r l*) $\preceq$ *d*)
$\wedge$ ($\forall l < n$. (*z l*) $\preceq$ *e*)
$\wedge$ ($\forall l < n$. *2*^*c* $*$ (*z l*) = (*r l* + *d*) && *f*))

definition *reg-equations* :: *program* $\Rightarrow$ (*register* $\Rightarrow$ *nat*) $\Rightarrow$ (*register* $\Rightarrow$ *nat*) $\Rightarrow$

$(state \Rightarrow nat)$
$\Rightarrow$ $nat \Rightarrow nat \Rightarrow nat \Rightarrow nat \Rightarrow bool$ **where**
$(reg\text{-}equations\ p\ r\ z\ s\ b\ a\ n\ q) = ($
— 4.22 $(\forall\, l{>}0.\ l < n \longrightarrow r\ l =\quad b{*}r\ l + b{*}\sum R{+}\ p\ l\ (\lambda k.\ s\ k) - b{*}\sum R{-}\ p\ l\ (\lambda k.\ s\ k\ \&\&\ z\ l))$
$\wedge$ — 4.23 $(\quad r\ 0 = a + b{*}r\ 0 + b{*}\sum R{+}\ p\ 0\ (\lambda k.\ s\ k) - b{*}\sum R{-}\ p\ 0\ (\lambda k.\ s\ k\ \&\&\ z\ 0))$
$\wedge$ $(\forall\, l{<}n.\ r\ l < b\ \hat{}\ q))$ — Extra equation not in Matiyasevich's book. Needed to show that all registers are empty at time q

definition $state\text{-}equations :: program \Rightarrow (state \Rightarrow nat) \Rightarrow (register \Rightarrow nat) \Rightarrow$
$nat \Rightarrow nat \Rightarrow nat \Rightarrow nat \Rightarrow bool$ **where**
$state\text{-}equations\ p\ s\ z\ b\ e\ q\ m = ($
— 4.24 $(\forall\, d{>}0.\ d{\le}m \longrightarrow s\ d =\quad b{*}\sum S{+}\ p\ d\ (\lambda k.\ s\ k) + b{*}\sum S{-}\ p\ d\ (\lambda k.\ s\ k\ \&\&\ z\ (modifies\ (p!k)))$
$+ b{*}\sum S0\ p\ d\ (\lambda k.\ s\ k\ \&\&\ (e - z\ (modifies\ (p!k)))))$
$\wedge$ — 4.25 $(\quad s\ 0 = 1 + b{*}\sum S{+}\ p\ 0\ (\lambda k.\ s\ k) + b{*}\sum S{-}\ p\ 0\ (\lambda k.\ s\ k\ \&\&\ z\ (modifies\ (p!k)))$
$+ b{*}\sum S0\ p\ 0\ (\lambda k.\ s\ k\ \&\&\ (e - z\ (modifies\ (p!k)))))$
$\wedge$ — 4.27 $(s\ m = b\hat{}q)$
$\wedge$ $(\forall\, k{\le}m.\ s\ k \preceq e) \wedge\ (\forall\, k{<}m.\ s\ k < b\ \hat{}\ q)$ — these equations are not from the book
$\wedge$ $(\forall\, M{\le}m.\ (\sum k{\le}M.\ s\ k) \preceq e)$ — this equation is added, too)

definition $state\text{-}unique\text{-}equations :: program \Rightarrow (state{\Rightarrow}nat) \Rightarrow nat \Rightarrow nat \Rightarrow bool$ **where**
$state\text{-}unique\text{-}equations\ p\ s\ m\ e = ((\sum k{=}0..m.\ s\ k) \preceq e \wedge (\forall\, k{\le}m.\ s\ k \preceq e))$

definition $rm\text{-}constants :: nat \Rightarrow nat \Rightarrow nat \Rightarrow nat \Rightarrow nat \Rightarrow nat \Rightarrow nat \Rightarrow bool$ **where**
$rm\text{-}constants\ q\ c\ b\ d\ e\ f\ a = ($
— 4.14 $(b = B\ c)$
$\wedge$ — 4.16 $(d = D\ q\ c\ b)$
$\wedge$ — 4.18 $(e = E\ q\ b)$ — 4.19 left out (compare book)
$\wedge$ — 4.21 $(f = F\ q\ c\ b)$
$\wedge$ — extra equation not in the book $c > 0$
$\wedge$ — 4.26 $(a < 2\hat{}c) \wedge (q{>}0))$

definition $rm\text{-}equations\text{-}old :: program \Rightarrow nat \Rightarrow nat \Rightarrow nat \Rightarrow bool$ **where**
$rm\text{-}equations\text{-}old\ p\ q\ a\ n = ($
$\exists\ b\ c\ d\ e\ f :: nat.$
$\exists\ r\ z :: register \Rightarrow nat.$
$\exists\ s :: state \Rightarrow nat.$

mask-equations n r z c d e f
∧ *reg-equations p r z s b a n q*
∧ *state-equations p s z b e q* (*length p* − *1*)
∧ *rm-constants q c b d e f a*)

end

4.2 Preliminary commutation relations

theory *CommutationRelations*
imports *RegisterMachineSimulation MachineEquations*
begin

lemma *aux-commute-bitAND-sum*:
fixes *N C* :: *nat*
and *fxt* :: *nat* ⇒ *nat*
shows ∀ *i*≤*N*. ∀ *j*≤*N*. *i* ≠ *j* ⟶ (∀ *k*. (*fct i*) ¡ *k* ∗ (*fct j*) ¡ *k* = *0*)
⟹ (∑ *k* ≤ *N*. *fct k* && *C*) = (∑ *k* ≤ *N*. *fct k*) && *C*
proof (*induct N*)
case *0*
then show *?case* **by** *auto*
next
case (*Suc N*)
assume *Suc-assms*: ∀ *i*≤*Suc N*. ∀ *j*≤*Suc N*. *i* ≠ *j* ⟶ (∀ *k*. *fct i* ¡ *k* ∗ *fct j* ¡ *k* = *0*)

have (∑ *k*≤*Suc N*. *fct k* && *C*) = (∑ *k*≤*N*. *fct k* && *C*) + (*fct* (*Suc N*) && *C*)
by (*auto cong*: *sum.cong*)
also have ... = (*sum fct* {..*N*} && *C*) + (*fct* (*Suc N*) && *C*)
using *Suc* **by** *auto*
also have ... = (*sum fct* {..*N*} + *fct* (*Suc N*)) && *C*
proof −
let *?a* = *sum fct* {..*N*} && *C*
let *?b* = *fct* (*Suc N*) && *C*

have *formula*: ∀ *d*. (*?a* + *?b*) ¡ *d* = (*?a* ¡ *d* + *?b* ¡ *d* + *bin-carry ?a ?b d*) *mod 2*
using *sum-digit-formula* **by** *auto*

have *nocarry4*: ∀ *n*≤*Suc N*. ∀ *d*. (*sum fct* {..*n*} ¡ *d* > *0* ⟶ (∃ *i*≤*n*. (*fct i*) ¡ *d* > *0*))
∧ *bin-carry* (*sum fct* {..<*n*}) (*fct n*) *d* = *0*
proof −
{
fix *n*

```
have n ≤ Suc N ⟹
      ∀ d. ((∀ i≤n. (fct i) ¡ d = 0)
     ⟶ sum fct {..n} ¡ d = 0) ∧ bin-carry (sum fct {..<n}) (fct n) d = 0
proof (induction n)
  case 0
  then show ?case by (simp add: bin-carry-def)
next
  case (Suc m)

    from Suc Suc.prems have ex: ∀ d. sum fct {..<Suc m} ¡ d > 0 ⟶
(∃ i<Suc m. fct i ¡ d = 1)
    using nth-bit-def
      by (metis One-nat-def Suc-less-eq lessThan-Suc-atMost less-Suc-eq
nat-less-le
          not-mod2-eq-Suc-0-eq-0)


  have h1: ∀ d. sum fct {..<Suc m} ¡ d + fct (Suc m) ¡ d ≤ 1
  proof −
    {
      fix d
      have sum fct {..<Suc m} ¡ d + fct (Suc m) ¡ d ≤ 1
      proof (cases sum fct {..<Suc m} ¡ d = 0)
        case True
        have fct (Suc m) ¡ d ≤ 1
          using nth-bit-def by auto
        then show ?thesis using True by auto
      next
        case False
        then have ∃ i<Suc m. fct i ¡ d > 0
          using ex by (metis neq0-conv zero-less-one)
        then obtain i where i: i<Suc m ∧ fct i ¡ d > 0 by auto
        hence i ≤ Suc N using Suc.prems by auto
        hence ∀ j≤Suc N. i ≠ j ⟶ (∀ k. fct i ¡ k ∗ fct j ¡ k = 0)
          using Suc-assms by auto
        then have fct (Suc m) ¡ d = 0
          using Suc.prems i nat-neq-iff
          by (auto) (blast)
        moreover from False have sum fct {..<Suc m} ¡ d = 1
          by (simp add: nth-bit-def)
        ultimately show ?thesis by auto
      qed
    }
    thus ?thesis by auto
  qed


  from h1 have h2: ∀ d. bin-carry (sum fct {..<Suc m}) (fct (Suc m)) d =
```

```
0
          using carry-digit-impl by (metis Suc-1 Suc-n-not-le-n)
        then have nocarry3: ∀ d. bin-carry (sum fct {..m}) (fct (Suc m)) d = 0
          by (simp add: lessThan-Suc-atMost)


        {
          fix d
          assume a: ∀ i≤Suc m. (fct i) ¡ d = 0
          have sum fct {..Suc m} ¡ d = (sum fct {..m} + fct (Suc m)) ¡ d
            by auto
          also have ... =
             (sum fct {..m} ¡ d + fct (Suc m) ¡ d + bin-carry (sum fct {..m}) (fct
(Suc m)) d) mod 2
            using sum-digit-formula[of sum fct {..m} fct (Suc m) d] by auto
          finally have sum fct {..Suc m} ¡ d = 0
            using nocarry3 Suc a by auto
        }
        with h2 show ?case by auto
      qed

      then have n ≤ Suc N ⟹ ∀ d. (sum fct {..n} ¡ d > 0 ⟶ (∃ i≤n. (fct i) ¡
d > 0))
                    ∧ bin-carry (sum fct {..<n}) (fct n) d = 0
        by auto
    }
    thus ?thesis by auto
  qed

  from Suc-assms have h3: ∀ d. ?a ¡ d + ?b ¡ d ≤ 1
  proof −
    have ∀ d. ?a ¡ d + ?b ¡ d = (sum fct {..N} ¡ d + fct (Suc N) ¡ d) ∗ C ¡ d
      using bitAND-digit-mult add-mult-distrib by auto
    then have ∀ d. ?a ¡ d + ?b ¡ d ≤ (sum fct {..N} ¡ d + fct (Suc N) ¡ d)
      using nth-bit-def by auto
    thus ?thesis
       using sum-carry-formula nocarry4 no-carry-mult-equiv nth-bit-bounded bi-
tAND-digit-mult
     by (metis One-nat-def add.commute add-decreasing le-Suc-eq lessThan-Suc-atMost
             nat-1-eq-mult-iff)
  qed
  from h3 have h4: ∀ d. bin-carry ?a ?b d = 0
    by (metis Suc-1 Suc-n-not-le-n carry-digit-impl)


  have h5: ∀ d. bin-carry (sum fct {..N}) (fct (Suc N)) d = 0
    using nocarry4 lessThan-Suc-atMost by auto

  from formula h3 h4 have ∀ d. ( ?a + ?b ) ¡ d = ?a ¡ d + ?b ¡ d
```

by (*metis* (*no-types*, *lifting*) *add-cancel-right-left add-le-same-cancel1 add-self-mod-2 le-zero-eq not-mod2-eq-Suc-0-eq-0 nth-bit-def one-mod-two-eq-one plus-1-eq-Suc*)

then have $\forall d.$ (*?a* + *?b*) ¡ d = *sum fct* {..N} ¡ d ∗ C ¡ d + *fct* (*Suc N*) ¡ d ∗ C ¡ d
using *bitAND-digit-mult* **by** *auto*

then have $\forall d.$ (*?a* + *?b*) ¡ d = (*sum fct* {..N} ¡ d + *fct* (*Suc N*) ¡ d) ∗ C ¡ d
by (*simp add*: *add-mult-distrib*)
moreover have $\forall d.$ *sum fct* {..N} ¡ d + *fct* (*Suc N*) ¡ d
= (*sum fct* {..N} ¡ d + *fct* (*Suc N*) ¡ d + *bin-carry* (*sum fct* {..N}) (*fct* (*Suc N*)) d) *mod 2*
using *h5 sum-carry-formula*
by (*metis add-diff-cancel-left′ add-diff-cancel-right′ mult-div-mod-eq mult-is-0*)
ultimately have $\forall d.$ (*?a* + *?b*) ¡ d = (*sum fct* {..N} + *fct* (*Suc N*)) ¡ d ∗ C ¡ d
using *sum-digit-formula* **by** *auto*

then have $\forall d.$ (*?a* + *?b*) ¡ d = ((*sum fct* {..N} + *fct* (*Suc N*)) && C) ¡ d
using *bitAND-digit-mult* **by** *auto*
thus *?thesis* **using** *digit-wise-equiv* **by** *blast*
qed
ultimately show *?case* **by** *auto*
qed

lemma *aux-commute-bitAND-sum-if*:
fixes *N const* :: *nat*
assumes *nocarry*: $\forall i \le N.\ \forall j \le N.\ i \neq j \longrightarrow (\forall k.\ (fct\ i)$ ¡ $k * (fct\ j)$ ¡ $k = 0)$
shows ($\sum k \le N.$ *if cond k then fct k* && *const else 0*)
= ($\sum k \le N.$ *if cond k then fct k else 0*) && *const*
proof −
from *nocarry* **have** *nocarry-if*:
$\forall i \le N.\ \forall j \le N.\ i \neq j \longrightarrow (\forall k.$ (*if cond i then fct i else 0*) ¡ k ∗ (*if cond j then fct j else 0*) ¡ $k = 0)$
by (*metis* (*full-types*) *aux1-digit-wise-equiv mult.commute mult-zero-left*)

have (*if cond k then fct k* && *const else 0*) = (*if cond k then fct k else 0*) && *const* **for** *k*
by *auto*
hence ($\sum k \le N.$ *if cond k then fct k* && *const else 0*)
= ($\sum k \le N.$ (*if cond k then fct k else 0*) && *const*)
by *auto*
also have ... = ($\sum k \le N.$ *if cond k then fct k else 0*) && *const*
using *nocarry-if aux-commute-bitAND-sum*[**where** *?fct* = $\lambda k.$ (*if cond k then fct k else 0*)]
by *blast*
ultimately show *?thesis* **by** *auto*
qed

```
lemma mod-mod:
  fixes x a b :: nat
  shows x mod 2^a mod 2^b = x mod 2^(min a b)
  by (metis min.commute take-bit-eq-mod take-bit-take-bit)

lemma carry-gen-pow2-reduct:
  assumes c>0
  defines b: b ≡ 2 ^ (Suc c)
  assumes nth-digit x (t−1) (2^Suc c) ¡ c = 0
      and nth-digit y (t−1) (2^Suc c) ¡ c = 0
    shows k≤c ⟹ bin-carry (nth-digit x t b) (nth-digit y t b) k
              = bin-carry x y (Suc c ∗ t + k)
proof (induction k)
  case 0
  then show ?case
  proof (cases t=0)
    case True
    then show ?thesis using bin-carry-def by auto
  next
    case False
    hence t>0 by auto
    from assms(3) have x ¡ (Suc c ∗ (t − 1) + c) = 0
      using digit-gen-pow2-reduct[of c Suc c x t−1] by auto
    moreover have y ¡ (Suc c ∗ (t − 1) + c) = 0
      using assms(4) digit-gen-pow2-reduct[of c Suc c y t−1] by auto
    moreover have Suc c ∗ (t − 1) + c = t + c ∗ t − Suc 0
      using add.left-commute gr0-conv-Suc ‹t>0› by auto
    ultimately have (x ¡ (t + c ∗ t − Suc 0) + y ¡ (t + c ∗ t − Suc 0)
      + bin-carry x y (t + c ∗ t − Suc 0)) ≤ 1 using carry-bounded by auto
    hence bin-carry x y (Suc c ∗ t) = 0
      using sum-carry-formula[of x y Suc c ∗ t − 1] ‹c>0› ‹t>0› by auto

    moreover have bin-carry (nth-digit x t b) (nth-digit y t b) 0 = 0
      using 0 bin-carry-def by auto
    ultimately show ?thesis by auto
  qed
next
  case (Suc k)
  have k<Suc c ⟹ x ¡ (Suc c ∗ t + k) = nth-digit x t b ¡ k
    using digit-gen-pow2-reduct[of k Suc c x t] b by auto
  moreover have k<Suc c ⟹ y ¡ (Suc c ∗ t + k) = nth-digit y t b ¡ k
    using digit-gen-pow2-reduct[of k Suc c y t] b by auto
  ultimately show ?case using Suc
      sum-carry-formula[of nth-digit x t b nth-digit y t b k]
      sum-carry-formula[of x y Suc c ∗ t + k]
      by auto
qed
```

```
lemma nth-digit-bound:
  fixes c defines b ≡ 2 ^ (Suc c)
  shows nth-digit x t b < 2 ^ (Suc c)
  using nth-digit-def b-def by auto

lemma digit-wise-block-additivity:
  fixes c
  defines b ≡ 2 ^ Suc c
  assumes nth-digit x (t−1) (2^Suc c) ¡ c = 0
     and nth-digit y (t−1) (2^Suc c) ¡ c = 0
     and k≤c
     and c>0
  shows nth-digit (x+y) t b ¡ k = (nth-digit x t b + nth-digit y t b) ¡ k
proof −
  have k < Suc c using ‹k≤c› by simp
  have x: nth-digit x t b ¡ k = x ¡ (Suc c ∗ t + k)
    using digit-gen-pow2-reduct[of k Suc c x t] b-def ‹k<Suc c› by auto
  have y: nth-digit y t b ¡ k = y ¡ (Suc c ∗ t + k)
    using digit-gen-pow2-reduct[of k Suc c y t] b-def ‹k<Suc c› by auto

  have (nth-digit x t b + nth-digit y t b) ¡ k
     = ((nth-digit x t b) ¡ k + (nth-digit y t b) ¡ k
     + bin-carry (nth-digit x t b) (nth-digit y t b) k) mod 2
    using sum-digit-formula[of nth-digit x t b nth-digit y t b k] by auto
  also have ... = (x ¡ (Suc c ∗ t + k) + y ¡ (Suc c ∗ t + k)
     + bin-carry (nth-digit x t b) (nth-digit y t b) k) mod 2
    using x y by auto
  also have ... = (x ¡ (Suc c ∗ t + k) + y ¡ (Suc c ∗ t + k)
     + bin-carry x y (Suc c ∗ t + k)) mod 2
    using carry-gen-pow2-reduct[of c x t y k] assms by auto
  also have ... = (x + y) ¡ (Suc c ∗ t + k)
    using sum-digit-formula by auto
  also have ... = nth-digit (x+y) t b ¡ k
    using digit-gen-pow2-reduct[of k Suc c x+y t] b-def ‹k<Suc c› by auto
  finally show ?thesis by auto
qed

lemma block-additivity:
  assumes c > 0
  defines b ≡ 2 ^ Suc c
  assumes nth-digit x (t−1) b ¡ c = 0
     and nth-digit y (t−1) b ¡ c = 0
     and nth-digit x t b ¡ c = 0
     and nth-digit y t b ¡ c = 0

  shows nth-digit (x+y) t b = nth-digit x t b + nth-digit y t b
proof −
  {
    have nth-digit x t b < b using nth-digit-bound b-def by auto
```

hence *x-digit-bound*: ⋀*k*. *k*≥*Suc c* ⟶ *nth-digit x t b* ¡ *k* = *0*
using *nth-bit-def b-def aux-lt-implies-mask b-def* **by** *metis*

have *nth-digit y t b* < *b* **using** *nth-digit-bound b-def* **by** *auto*
hence *y-digit-bound*: ⋀*k*. *k*≥*Suc c* ⟶ *nth-digit y t b* ¡ *k* = *0*
using *nth-bit-def b-def aux-lt-implies-mask b-def* **by** *metis*

fix *k*
assume *k*: *k*≥*Suc c*
have *carry0*: *bin-carry* (*nth-digit x t b*) (*nth-digit y t b*) *k* = *0*
proof −

have *base*: *bin-carry* (*nth-digit x t b*) (*nth-digit y t b*) (*Suc c*) = *0*
using *sum-carry-formula*[**where** *?k* = *c*] *bin-carry-bounded*[**where** *?k* = *c*]
using *assms*(*5*−*6*) **by** (*metis Suc-eq-plus1 add-cancel-left-left mod-div-trivial*)

{
fix *n*
assume *n*: *n* ≥ *Suc c*
assume *IH*: *bin-carry* (*nth-digit x t b*) (*nth-digit y t b*) *n* = *0*
have *bin-carry* (*nth-digit x t b*) (*nth-digit y t b*) (*Suc n*)
= (*nth-digit x t b* ¡ *n* + *nth-digit y t b* ¡ *n*
+ *bin-carry* (*nth-digit x t b*) (*nth-digit y t b*) *n*) *div 2*
using *sum-carry-formula*[*of nth-digit x t b nth-digit y t b*] **by** *auto*
also have ... = *bin-carry* (*nth-digit x t b*) (*nth-digit y t b*) *n div 2*
using *x-digit-bound y-digit-bound n* **by** *auto*
also have ... = *0* **using** *IH* **by** *auto*

finally have *bin-carry* (*nth-digit x t b*) (*nth-digit y t b*) (*Suc n*) = *0*
by *auto*
}

then show *?thesis*
using *k base*
using *nat-induct-at-least*[**where** *?P* = λ*k*. *bin-carry* (*nth-digit x t b*)
(*nth-digit y t b*) *k* = *0*]
by *auto*
qed

have (*nth-digit x t b* + *nth-digit y t b*) ¡ *k*
= (*nth-digit x t b* ¡ *k* + *nth-digit y t b* ¡ *k*
+ *bin-carry* (*nth-digit x t b*) (*nth-digit y t b*) *k*) *mod 2*
using *sum-digit-formula*[*of nth-digit x t b nth-digit y t b k*] **by** *auto*
hence *separate-sum*: (*nth-digit x t b* + *nth-digit y t b*) ¡ *k* = *0*
using *x-digit-bound y-digit-bound carry0 k* **by** *auto*

have *nth-digit* (*x*+*y*) *t b* < *b*
using *nth-digit-bound b-def* **by** *auto*

```
    hence xy-sum: nth-digit (x+y) t b ¡ k = 0
      using nth-bit-def b-def aux-lt-implies-mask b-def k by metis
    from xy-sum separate-sum have k-ge: nth-digit (x+y) t b ¡ k
                                  = (nth-digit x t b + nth-digit y t b) ¡ k
      by auto
  }
  hence k-ge: k≥Suc c ⟶ nth-digit (x+y) t b ¡ k
                        = (nth-digit x t b + nth-digit y t b) ¡ k for k
    by auto

  moreover have k-lt:k<Suc c ⟶ nth-digit (x+y) t b ¡ k
                             = (nth-digit x t b + nth-digit y t b) ¡ k for k
    using digit-wise-block-additivity assms by auto

  ultimately have nth-digit (x+y) t b ¡ k
                 = (nth-digit x t b + nth-digit y t b) ¡ k for k
    by(cases k<Suc c; auto)

  thus ?thesis using digit-wise-equiv[of nth-digit (x+y) t b] by auto
qed

lemma block-to-sum:
  assumes c>0
  defines b: b ≡ 2 ^(Suc c)
  assumes yltx-digits: ∀ t'. nth-digit y t' b ≤ nth-digit x t' b
  shows y mod b^t ≤ x mod b^t
proof(cases t=0)
  case True
  then show ?thesis by auto
next
  case False
  show ?thesis using yltx-digits apply(induct t, auto) using yltx-digits
     by (smt add.commute add-left-cancel add-mono-thms-linordered-semiring(1)
mod-mult2-eq
        mult-le-cancel2 nth-digit-def semiring-normalization-rules(7))
qed

lemma narry-gen-pow2-reduct:
  assumes c>0
  defines b: b ≡ 2 ^(Suc c)
  assumes yltx-digits: ∀ t'. nth-digit y t' b ≤ nth-digit x t' b
  shows k≤c ⟹ bin-narry (nth-digit x t b) (nth-digit y t b) k
    = bin-narry x y (Suc c ∗ t + k)
proof (induction k)
  case 0
  then show ?case
  proof (cases t=0)
    case True
    then show ?thesis by (simp add: bin-narry-def)
```

```
  next
    case False
    have bin-narry x y (Suc c ∗ t) = 0 using yltx-digits block-to-sum bin-narry-def
assms
      by (metis not-less power-mult)
    then show ?thesis by (simp add: bin-narry-def)
  qed
next
  case (Suc k)
  have ylex: y≤x using yltx-digits digitwise-leq b Suc-1 lessI power-gt1 by metis
  have k<Suc c ⟹ x ¡ (Suc c ∗ t + k) = nth-digit x t b ¡ k
    using digit-gen-pow2-reduct[of k Suc c x t] b by auto
  moreover have k<Suc c ⟹ y ¡ (Suc c ∗ t + k) = nth-digit y t b ¡ k
    using digit-gen-pow2-reduct[of k Suc c y t] b by auto
  ultimately show ?case using Suc yltx-digits
      dif-narry-formula[of (nth-digit y t b) (nth-digit x t b) k]
      dif-narry-formula[of y x Suc c ∗ t + k] ylex by auto
qed

lemma digit-wise-block-subtractivity:
  fixes c
  defines b ≡ 2 ^ Suc c
  assumes yltx-digits: ∀ t'. nth-digit y t' b ≤ nth-digit x t' b
      and k≤c
      and c>0
  shows nth-digit (x−y) t b ¡ k = (nth-digit x t b − nth-digit y t b) ¡ k
proof −
  have x: nth-digit x t b ¡ k = x ¡ (Suc c ∗ t + k)
    using digit-gen-pow2-reduct[of k Suc c x t] b-def ‹k≤c› by auto
  have y: nth-digit y t b ¡ k = y ¡ (Suc c ∗ t + k)
    using digit-gen-pow2-reduct[of k Suc c y t] b-def ‹k≤c› by auto

  have b > 1 using ‹c>0› b-def
    by (metis Suc-1 lessI power-gt1)
  then have yltx: y ≤ x using digitwise-leq yltx-digits by auto

  have (nth-digit x t b − nth-digit y t b) ¡ k
      = ((nth-digit x t b) ¡ k + (nth-digit y t b) ¡ k
        + bin-narry (nth-digit x t b) (nth-digit y t b) k) mod 2
    using dif-digit-formula yltx-digits by auto
  also have ... = (x ¡ (Suc c ∗ t + k) + y ¡ (Suc c ∗ t + k)
                  + bin-narry (nth-digit x t b) (nth-digit y t b) k) mod 2
    using x y by auto
  also have ... = (x ¡ (Suc c ∗ t + k) + y ¡ (Suc c ∗ t + k)
                  + bin-narry x y (Suc c ∗ t + k)) mod 2
  using narry-gen-pow2-reduct using assms(3) assms(4) b-def yltx-digits by auto
  also have ... = nth-digit (x−y) t b ¡ k
   using digit-gen-pow2-reduct[of k Suc c x−y t] b-def ‹k≤c› dif-digit-formula yltx
```

by *auto*
finally show *?thesis* **by** *auto*
qed

lemma *block-subtractivity*:
assumes $c > 0$
defines $b \equiv 2 \;\hat{}\; Suc\ c$
assumes *block-wise-lt*: $\forall t'.\ nth\text{-}digit\ y\ t'\ b \leq nth\text{-}digit\ x\ t'\ b$
shows $nth\text{-}digit\ (x-y)\ t\ b = nth\text{-}digit\ x\ t\ b - nth\text{-}digit\ y\ t\ b$
proof −
have $k \leq c \longrightarrow nth\text{-}digit\ (x-y)\ t\ b$ ¡ $k = (x - y)$ ¡ $(Suc\ c * t + k)$ **for** *k*
using *digit-gen-pow2-reduct*[*of k Suc c x−y t*] *b-def* **by** *auto*
have $k \leq c \longrightarrow nth\text{-}digit\ x\ t\ b$ ¡ $k = x$ ¡ $(Suc\ c * t + k)$ **for** *k*
using *digit-gen-pow2-reduct*[*of k Suc c x t*] *b-def* **by** *auto*
have $k \leq c \longrightarrow nth\text{-}digit\ y\ t\ b$ ¡ $k = y$ ¡ $(Suc\ c * t + k)$ **for** *k*
using *digit-gen-pow2-reduct*[*of k Suc c y t*] *b-def* **by** *auto*

have *k-le*: $k \leq c \longrightarrow nth\text{-}digit\ (x-y)\ t\ b$ ¡ k
$= (nth\text{-}digit\ x\ t\ b - nth\text{-}digit\ y\ t\ b)$ ¡ k **for** *k*
using *assms digit-wise-block-subtractivity* **by** *auto*

have $nth\text{-}digit\ x\ t\ b - nth\text{-}digit\ y\ t\ b < b$ **using**
nth-digit-bound b-def **by** (*simp add*: *less-imp-diff-less*)
hence *diff*: $k \geq Suc\ c \longrightarrow (nth\text{-}digit\ x\ t\ b - nth\text{-}digit\ y\ t\ b)$ ¡ $k = 0$ **for** *k*
using *nth-bit-def b-def aux-lt-implies-mask b-def* **by** *metis*

have $nth\text{-}digit\ (x-y)\ t\ b < b$ **using** *nth-digit-bound b-def* **by** *auto*
hence $k \geq Suc\ c \longrightarrow nth\text{-}digit\ (x-y)\ t\ b$ ¡ $k = 0$ **for** *k*
using *nth-bit-def b-def aux-lt-implies-mask b-def* **by** *metis*
with *diff* **have** *k-gt*: $k > c \longrightarrow nth\text{-}digit\ (x-y)\ t\ b$ ¡ k
$= (nth\text{-}digit\ x\ t\ b - nth\text{-}digit\ y\ t\ b)$ ¡ k **for** *k*
by *auto*
from *k-le k-gt* **have** $nth\text{-}digit\ (x-y)\ t\ b$ ¡ k
$= (nth\text{-}digit\ x\ t\ b - nth\text{-}digit\ y\ t\ b)$ ¡ k **for** *k* **by**(*cases* $k>c$; *auto*)
thus *?thesis* **using** *digit-wise-equiv*[*of nth-digit x t b − nth-digit y t b*
nth-digit (*x−y*) *t b*] **by** *auto*
qed

lemma *bitAND-nth-digit-commute*:
assumes *b-def*: $b = 2\hat{}(Suc\ c)$
shows *nth-digit* (*x* && *y*) *t b* = *nth-digit x t b* && *nth-digit y t b*
proof −
{
fix *k*
assume *k*: $k < Suc\ c$
have *prod*: *nth-digit* (*x* && *y*) *t b* ¡ *k* = (*x* && *y*) ¡ (*Suc c* ∗ *t* + *k*)
using *digit-gen-pow2-reduct*[*of - Suc c x* && *y t*] *b-def k* **by** *auto*
moreover have *x*: *nth-digit x t b* ¡ *k* = *x* ¡ (*Suc c* ∗ *t* + *k*)
using *digit-gen-pow2-reduct*[*of - Suc c x*] *b-def k* **by** *auto*

moreover have *y*: *nth-digit y t b* ¡ *k* = *y* ¡ (*Suc c* ∗ *t* + *k*)
using *digit-gen-pow2-reduct*[*of - Suc c y*] *b-def k* **by** *auto*
moreover have (*x* && *y*) ¡ (*Suc c* ∗ *t* + *k*) = (*x* ¡ (*Suc c* ∗ *t* + *k*)) ∗ (*y* ¡ (*Suc c* ∗ *t* + *k*))
using *bitAND-digit-mult* **by** *auto*

ultimately have *nth-digit* (*x* && *y*) *t b* ¡ *k*
= *nth-digit x t b* ¡ *k* ∗ *nth-digit y t b* ¡ *k*
by *auto*

also have ... = (*nth-digit x t b* && *nth-digit y t b*) ¡ *k*
using *bitAND-digit-mult* **by** *auto*

finally have *nth-digit* (*x* && *y*) *t b* ¡ *k*
= (*nth-digit x t b* && *nth-digit y t b*) ¡ *k*
by *auto*
}

then have *nth-digit* (*x* && *y*) *t b* ¡ *k*
= (*nth-digit x t b* && *nth-digit y t b*) ¡ *k* **for** *k*
by (*metis aux-lt-implies-mask b-def bitAND-digit-mult leI mult-eq-0-iff nth-digit-bound*)

then show *?thesis* **using** *b-def digit-wise-equiv*[*of nth-digit* (*x* && *y*) *t b*] **by** *auto*
qed

lemma *bx-aux*:
shows *b*>*1* ⟹ *nth-digit* (*b*^*x*) *t′ b* = (*if x*=*t′ then 1 else 0*)
by (*cases t′* > *x*, *auto simp*: *nth-digit-def*)
(*metis dvd-imp-mod-0 dvd-power leI less-SucI nat-neq-iff power-diff zero-less-diff*)

context
fixes *c b* :: *nat*
assumes *b-def*: *b* ≡ *2*^(*Suc c*)
assumes *c-gt0*: *c*>*0*
begin

lemma *b-gt1*: *b*>*1* **using** *c-gt0 b-def*
using *one-less-numeral-iff one-less-power semiring-norm*(*76*) **by** *blast*

Commutation relations with sums

lemma *finite-sum-nth-digit-commute*:
fixes *M* :: *nat*
shows ∀ *t*. ∀ *k*≤*M*. *nth-digit* (*fct k*) *t b* < *2*^*c* ⟹
∀ *t*. (∑ *i*=*0*..*M*. *nth-digit* (*fct i*) *t b*) < *2*^*c* ⟹
nth-digit (∑ *i*=*0*..*M*. *fct i*) *t b* = (∑ *i*=*0*..*M*. (*nth-digit* (*fct i*) *t b*))
proof (*induct M arbitrary*: *t*)

case *0* **thus** *?case* **by** *auto*
next
case (*Suc M*)

have *assm1*: ∀ *t*. ∀ *k*≤*Suc M*. *nth-digit* (*fct k*) *t b* < *2^c*
using *Suc.prems* **by** *auto*
have *assm1-reduced*: ∀ *t*. ∀ *k*≤*M*. *nth-digit* (*fct k*) *t b* < *2^c*
using *assm1* **by** *auto*
have *nocarry2*: ∀ *t*. ∀ *k*≤*Suc M*. *nth-digit* (*fct k*) *t b* ¡ *c* = *0*
using *assm1 nth-bit-def* **by** *auto*

have *assm2*:
∀ *t*. *nth-digit* (*fct* (*Suc M*)) *t b* + (∑ *i*=*0*..*M*. *nth-digit* (*fct i*) *t b*) < *2^c*
using *Suc.prems* **by** (*simp add*: *add.commute*)
hence *assm2-reduced*: ∀ *t*. (∑ *i*=*0*..*M*. *nth-digit* (*fct i*) *t b*) < *2^c*
using *Suc.prems*(*2*) *add-lessD1* **by** *fastforce*

have *IH*: ∀ *t*. *nth-digit* (∑ *i*=*0*..*M*. *fct i*) *t b*
= (∑ *i*=*0*..*M*. *nth-digit* (*fct i*) *t b*)
using *assm1-reduced assm2-reduced Suc.hyps* **by** *blast*
then have *assm2-IH-commuted*: ∀ *t*. *nth-digit* (∑ *i*=*0*..*M*. *fct i*) *t b* < *2^c*
using *assm2-reduced* **by** *auto*
hence *nocarry3*: ∀ *t*. *nth-digit* (∑ *i*=*0*..*M*. *fct i*) *t b* ¡ *c* = *0*
using *aux-lt-implies-mask* **by** *blast*

have *1*: *nth-digit* (*sum fct* {*0*..*M*}) (*t* − *1*) *b* ¡ *c* = *0* **using** *nocarry3* **by** *auto*
have *2*: *nth-digit* (*fct* (*Suc M*)) (*t* − *1*) *b* ¡ *c* = *0* **using** *nocarry2* **by** *auto*
have *3*: *nth-digit* (*sum fct* {*0*..*M*}) *t b* ¡ *c* = *0* **using** *nocarry3* **by** *auto*
have *4*: *nth-digit* (*fct* (*Suc M*)) *t b* ¡ *c* = *0* **using** *nocarry2* **by** *auto*
from *block-additivity* **show** *?case* **using** *1 2 3 4 c-gt0 Suc b-def*
using *IH* **by** *auto*
qed

lemma *sum-nth-digit-commute-aux*:
fixes *g*
defines *SX-def*: *SX* ≡ λ*l m* (*fct* :: *nat* ⇒ *nat*). (∑ *k* = *0*..*m*. *if g l k then fct k else 0*)
assumes *nocarry*: ∀ *t*. ∀ *k*≤*M*. *nth-digit* (*fct k*) *t b* < *2^c*
and *nocarry-sum*: ∀ *t*. (*SX l M* (λ*k*. *nth-digit* (*fct k*) *t b*)) < *2^c*
shows *nth-digit* (*SX l M fct*) *t b* = *SX l M* (λ*k*. *nth-digit* (*fct k*) *t b*)
proof −
have *aux*: *nth-digit* (*if g l i then fct i else 0*) *t b*
= (*if g l i then nth-digit* (*fct i*) *t b else 0*) **for** *i t*
using *aux1-digit-wise-gen-equiv b-gt1* **by** *auto*

from *nocarry* **have** ∀ *t*. ∀ *k*≤*M*. *nth-digit* (*if g l k then fct k else 0*) *t b* < *2^c*
using *aux* **by** *auto*

moreover from *nocarry-sum* **have**

$\forall t.\ (\sum i = 0..M.\ nth\text{-}digit\ (if\ g\ l\ i\ then\ fct\ i\ else\ 0)\ t\ b) < 2\ \hat{}\ c$
unfolding *SX-def* **by** (*auto simp*: *aux*)

ultimately have $nth\text{-}digit\ (\sum k = 0..M.\ if\ g\ l\ k\ then\ fct\ k\ else\ 0)\ t\ b$
$= (\sum k = 0..M.\ nth\text{-}digit\ (if\ g\ l\ k\ then\ fct\ k\ else\ 0)\ t\ b)$
using *finite-sum-nth-digit-commute*[**where** $?fct = \lambda k.\ if\ g\ l\ k\ then\ fct\ k\ else\ 0$]
by (*simp add*: *SX-def*)
moreover have $\forall k.\ nth\text{-}digit\ (if\ g\ l\ k\ then\ fct\ k\ else\ 0)\ t\ b$
$= (if\ g\ l\ k\ then\ nth\text{-}digit\ (fct\ k)\ t\ b\ else\ 0)$
by (*auto simp*: *nth-digit-def*)
ultimately show *?thesis* **by** (*auto simp*: *SX-def*)
qed

lemma *sum-nth-digit-commute*:
fixes *g*
defines *SX-def*: $SX \equiv \lambda p\ l\ (fct :: nat \Rightarrow nat).\ (\sum k = 0..length\ p - 1.\ if\ g\ l\ k\ then\ fct\ k\ else\ 0)$
assumes *nocarry*: $\forall t.\ \forall k \le length\ p - 1.\ nth\text{-}digit\ (fct\ k)\ t\ b < 2\hat{}c$
and *nocarry-sum*: $\forall t.\ (SX\ p\ l\ (\lambda k.\ nth\text{-}digit\ (fct\ k)\ t\ b)) < 2\hat{}c$
shows $nth\text{-}digit\ (SX\ p\ l\ fct)\ t\ b = SX\ p\ l\ (\lambda k.\ nth\text{-}digit\ (fct\ k)\ t\ b)$
proof −
let $?m = length\ p - 1$

have $\forall t.\ (\sum k = 0..?m.\ if\ g\ l\ k\ then\ nth\text{-}digit\ (fct\ k)\ t\ b\ else\ 0) < 2\hat{}c$
using *nocarry-sum* **unfolding** *SX-def* **by** *blast*

then have $nth\text{-}digit\ (\sum k = 0..length\ p - 1.\ if\ g\ l\ k\ then\ fct\ k\ else\ 0)\ t\ b$
$= (\sum k = 0..length\ p - 1.\ if\ g\ l\ k\ then\ nth\text{-}digit\ (fct\ k)\ t\ b\ else\ 0)$
using *nocarry*
using *sum-nth-digit-commute-aux*[**where** $?M = length\ p - 1$]
by *auto*

then show *?thesis* **using** *SX-def* **by** *auto*
qed

Commute inside, need assumption for all partial sums

lemma *finite-sum-nth-digit-commute2*:
fixes *M* :: *nat*
shows $\forall t.\ \forall k \le M.\ nth\text{-}digit\ (fct\ k)\ t\ b < 2\hat{}c \Longrightarrow$
$\forall t.\ \forall m \le M.\ nth\text{-}digit\ (\sum i=0..m.\ fct\ i)\ t\ b < 2\hat{}c \Longrightarrow$
$nth\text{-}digit\ (\sum i=0..M.\ fct\ i)\ t\ b = (\sum i=0..M.\ (nth\text{-}digit\ (fct\ i)\ t\ b))$
proof (*induct M arbitrary*: *t*)
case *0* **thus** *?case* **by** *auto*
next
case (*Suc M*)

have *assm1*: $\forall t.\ \forall k \le Suc\ M.\ nth\text{-}digit\ (fct\ k)\ t\ b < 2\hat{}c$
using *Suc.prems* **by** *auto*
have *assm1-reduced*: $\forall t.\ \forall k \le M.\ nth\text{-}digit\ (fct\ k)\ t\ b < 2\hat{}c$

using *assm1* **by** *auto*
have *nocarry2*: ∀ *t*. ∀ *k*≤*Suc M*. *nth-digit* (*fct k*) *t b* ¡ *c* = *0*
using *assm1 nth-bit-def* **by** *auto*

have *assm2*:
∀ *t*. *nth-digit* (∑ *i*=*0*..*M*. (*fct i*)) *t b* < *2*^*c*
using *Suc.prems* **by** (*simp add*: *add.commute*)
hence *nocarry3*: ∀ *t*. *nth-digit* (∑ *i*=*0*..*M*. *fct i*) *t b* ¡ *c* = *0*
using *aux-lt-implies-mask* **by** *blast*

have *1*: *nth-digit* (*sum fct* {*0*..*M*}) (*t* − *1*) *b* ¡ *c* = *0* **using** *nocarry3* **by** *auto*
have *2*: *nth-digit* (*fct* (*Suc M*)) (*t* − *1*) *b* ¡ *c* = *0* **using** *nocarry2* **by** *auto*
have *3*: *nth-digit* (*sum fct* {*0*..*M*}) *t b* ¡ *c* = *0* **using** *nocarry3* **by** *auto*
have *4*: *nth-digit* (*fct* (*Suc M*)) *t b* ¡ *c* = *0* **using** *nocarry2* **by** *auto*
from *block-additivity* **show** *?case* **using** *1 2 3 4 c-gt0 Suc b-def* **by** *auto*
qed

lemma *sum-nth-digit-commute-aux2*:
fixes *g*
defines *SX-def*: *SX* ≡ λ*l m* (*fct* :: *nat* ⇒ *nat*). (∑ *k* = *0*..*m*. *if g l k then fct k else 0*)
assumes *nocarry*: ∀ *t*. ∀ *k*≤*M*. *nth-digit* (*fct k*) *t b* < *2*^*c*
and *nocarry-sum*: ∀ *t*. ∀ *m*≤*M*. *nth-digit* (*SX l m fct*) *t b* < *2*^*c*
shows *nth-digit* (*SX l M fct*) *t b* = *SX l M* (λ*k*. *nth-digit* (*fct k*) *t b*)
proof −
have *aux*: *nth-digit* (*if g l i then fct i else 0*) *t b*
= (*if g l i then nth-digit* (*fct i*) *t b else 0*) **for** *i t*
using *aux1-digit-wise-gen-equiv b-gt1* **by** *auto*

from *nocarry* **have** ∀ *t*. ∀ *k*≤*M*. *nth-digit* (*if g l k then fct k else 0*) *t b* < *2*^*c*
using *aux* **by** *auto*

moreover from *nocarry-sum* **have**
∀ *t*. ∀ *m*≤*M*. *nth-digit* (∑ *i* = *0*..*m*. (*if g l i then fct i else 0*)) *t b* < *2* ^ *c*
unfolding *SX-def* **by** (*auto simp*: *aux*)

ultimately have *nth-digit* (∑ *k* = *0*..*M*. *if g l k then fct k else 0*) *t b*
= (∑ *k* = *0*..*M*. *nth-digit* (*if g l k then fct k else 0*) *t b*)
using *finite-sum-nth-digit-commute2*[**where** *?fct* = λ*k*. *if g l k then fct k else 0*]
by (*simp add*: *SX-def*)
moreover have ∀ *k*. *nth-digit* (*if g l k then fct k else 0*) *t b*
= (*if g l k then nth-digit* (*fct k*) *t b else 0*)
by (*auto simp*: *nth-digit-def*)
ultimately show *?thesis* **by** (*auto simp*: *SX-def*)
qed

lemma *sum-nth-digit-commute2*:
fixes *g p*

defines *SX-def*: $SX \equiv \lambda p\ l\ (fct :: nat \Rightarrow nat).\ (\sum k = 0..length\ p - 1.\ if\ g\ l\ k\ then\ fct\ k\ else\ 0)$
assumes *nocarry*: $\forall t.\ \forall k \leq length\ p - 1.\ nth\text{-}digit\ (fct\ k)\ t\ b < 2\hat{}c$
and *nocarry-sum*: $\forall t.\ \forall m \leq length\ p - 1.\ nth\text{-}digit\ (SX\ (take\ (Suc\ m)\ p)\ l\ fct)\ t\ b < 2\hat{}c$
shows $nth\text{-}digit\ (SX\ p\ l\ fct)\ t\ b = SX\ p\ l\ (\lambda k.\ nth\text{-}digit\ (fct\ k)\ t\ b)$
proof −
have $\forall m \leq length\ p - 1.\ (SX\ (take\ (Suc\ m)\ p)\ l\ fct) = (\sum k = 0..m.\ if\ g\ l\ k\ then\ fct\ k\ else\ 0)$
unfolding *SX-def*
by (*auto*) (*metis* (*no-types*, *lifting*) *One-nat-def diff-Suc-1 min-absorb2 min-diff*)
hence $\forall t\ m.\ m \leq length\ p - 1 \longrightarrow$
$nth\text{-}digit\ (\sum k = 0..m.\ if\ g\ l\ k\ then\ fct\ k\ else\ 0)\ t\ b < 2\ \hat{}\ c$
using *nocarry-sum* **by** *auto*

then have $nth\text{-}digit\ (\sum k = 0..length\ p - 1.\ if\ g\ l\ k\ then\ fct\ k\ else\ 0)\ t\ b$
$= (\sum k = 0..length\ p - 1.\ if\ g\ l\ k\ then\ nth\text{-}digit\ (fct\ k)\ t\ b\ else\ 0)$
using *nocarry*
using *sum-nth-digit-commute-aux2*[**where** $?M = length\ p - 1$ **and** $?fct = fct$ **and** $?g = g$]
by *blast*

then show *?thesis* **using** *SX-def* **by** *auto*
qed

end

end

4.3 From multiple to single step relations

theory *MultipleToSingleSteps*
imports *MachineEquations CommutationRelations ../Diophantine/Binary-And*
begin

This file contains lemmas that are needed to prove the <– direction of conclusion4.5 in the file MachineEquationEquivalence. In particular, it is shown that single step equations follow from the multiple step relations. The key idea of Matiyasevich's proof is to code all register and state values over the time into one large number. A further central statement in this file shows that the decoding of these numbers back to the single cell contents is indeed correct.

context
fixes $a :: nat$
and $ic :: configuration$
and $p :: program$
and $q :: nat$
and $r\ z :: register \Rightarrow nat$
and $s :: state \Rightarrow nat$

and *b c d e f* :: *nat*
and *m n* :: *nat*
and *Req Seq Zeq*

assumes *m-def*: $m \equiv$ *length p* − *1*
and *n-def*: $n \equiv$ *length* (*snd ic*)

assumes *is-val*: *is-valid-initial ic p a*

assumes *m-eq*: *mask-equations n r z c d e f*
and *r-eq*: *reg-equations p r z s b a n q*
and *s-eq*: *state-equations p s z b e q m*
and *c-eq*: *rm-constants q c b d e f a*

assumes *Seq-def*: *Seq* = (λ*k t. nth-digit* (*s k*) *t b*)
and *Req-def*: *Req* = (λ*l t. nth-digit* (*r l*) *t b*)
and *Zeq-def*: *Zeq* = (λ*l t. nth-digit* (*z l*) *t b*)

begin

Basic properties

lemma *n-gt0*: *n>0*
using *n-def is-val is-valid-initial-def*[*of ic p a*] *is-valid-def*
by *auto*

lemma *f-def*: *f* = ($\sum$ *t* = *0..q. 2^c* ∗ *b^t*)
using *c-eq* **by** (*simp add*: *rm-constants-def F-def*)
lemma *e-def*: *e* = ($\sum$ *t* = *0..q. b^t*)
using *c-eq* **by** (*simp add*: *rm-constants-def E-def*)
lemma *d-def*: *d* = ($\sum$ *t* = *0..q.* (*2^c* − *1*) ∗ *b^t*)
using *c-eq* **by** (*simp add*: *D-def rm-constants-def*)
lemma *b-def*: *b* = *2^*(*Suc c*)
using *c-eq* **by** (*simp add*: *B-def rm-constants-def*)

lemma *b-gt1*: *b* > *1* **using** *c-eq B-def rm-constants-def* **by** *auto*

lemma *c-gt0*: *c* > *0* **using** *rm-constants-def c-eq* **by** *auto*
lemma *h0*: *1* < (*2::nat*)*^c*
using *c-gt0 one-less-numeral-iff one-less-power semiring-norm*(*76*) **by** *blast*

lemma *rl-fst-digit-zero*:
assumes *l* < *n*
shows *nth-digit* (*r l*) *t b* ¡ *c* = *0*
proof −
have *2 ^ c* − (*Suc 0*) < *2 ^ Suc c* **using** *c-gt0* **by** (*simp add*: *less-imp-diff-less*)
hence ∀ *t. nth-digit d t b* = (*if t≤q then 2^c* − *1 else 0*)
using *nth-digit-gen-power-series*[*of* λ*x. 2^c* − *1 c*] *d-def c-gt0 b-def*

```
    by (simp add: d-def)
  then have d-lead-digit-zero: ∀ t. (nth-digit d t b) ¡ c = 0
    by (auto simp: nth-bit-def)

  from m-eq have (r l) ⪯ d
    by (simp add: mask-equations-def assms)
  then have ∀ k. (r l) ¡ k ≤ d ¡ k
    by (auto simp: masks-leq-equiv)
  then have ∀ t. (nth-digit (r l) t b ¡ c) ≤ (nth-digit d t b ¡ c)
    using digit-gen-pow2-reduct[where ?c = Suc c] by (auto simp: b-def)
  thus ?thesis
    by (auto simp: d-lead-digit-zero)
qed

lemma e-mask-bound:
  assumes x ⪯ e
  shows nth-digit x t b ≤ 1
proof −
   have x-bounded: nth-digit x t′ b ≤ nth-digit e t′ b for t′
   proof −
     have ∀ t′. x ¡ t′ ≤ e ¡ t′ using assms masks-leq-equiv by auto
     then have k-lt-c: ∀ t′. ∀ k′<Suc c. nth-digit x t′ b ¡ k′
                                        ≤ nth-digit e t′ b ¡ k′
       using digit-gen-pow2-reduct by (auto simp: b-def) (metis power-Suc)

     have k≥Suc c ⟹ x mod (2 ^ Suc c) div 2 ^ k = 0 for k x::nat
       by (simp only: drop-bit-take-bit flip: take-bit-eq-mod drop-bit-eq-div) simp
     then have ∀ k≥Suc c. nth-digit x y b ¡ k = 0 for x y
       using b-def nth-bit-def nth-digit-def by auto
     then have k-gt-c: ∀ t′. ∀ k′≥Suc c. nth-digit x t′ b ¡ k′
                                    ≤ nth-digit e t′ b ¡ k′
       by auto

     from k-lt-c k-gt-c have nth-digit x t′ b ≤ nth-digit e t′ b for t′
       using bitwise-leq by (meson not-le)
    thus ?thesis by auto
  qed

  have ∀ k. Suc 0 < 2^c using c-gt0 h0 by auto
  hence e-aux: nth-digit e tt b ≤ 1 for tt
    using e-def b-def c-gt0 nth-digit-gen-power-series[of λk. Suc 0 c q] by auto

  show ?thesis using e-aux[of t] x-bounded[of t] using le-trans by blast
qed


lemma sk-bound:
  shows ∀ t k. k≤length p − 1 ⟶ nth-digit (s k) t b ≤ 1
proof −
```

have ∀ $k \leq$ *length p* − *1*. *s k* ⪯ *e* **using** *s-eq state-equations-def m-def* **by** *auto*
thus *?thesis* **using** *e-mask-bound* **by** *auto*
qed

lemma *sk-bitAND-bound*:
shows ∀ *t k*. *k*≤*length p* − *1* ⟶ *nth-digit* (*s k* && *x k*) *t b* ≤ *1*
using *bitAND-nth-digit-commute sk-bound bitAND-lt masks-leq*
by (*auto simp*: *b-def*) (*meson dual-order.trans*)

lemma *s-bound*:
shows ∀ *j*<*m*. *s j* < *b* ^ *q*
using *s-eq state-equations-def* **by** *auto*

lemma *sk-sum-masked*:
shows ∀ *M*≤*m*. (∑ *k*≤*M*. *s k*) ⪯ *e*
using *s-eq state-equations-def* **by** *auto*

lemma *sk-sum-bound*:
shows ∀ *t M*. *M*≤*length p* − *1* ⟶ *nth-digit* (∑ *k*≤*M*. *s k*) *t b* ≤ *1*
using *sk-sum-masked e-mask-bound m-def* **by** *auto*

lemma *sum-sk-bound*:
shows (∑ *k*≤*length p* − *1*. *nth-digit* (*s k*) *t b*) ≤ *1*
proof −
have ∀ *t m*. *m* ≤ *length p* − *1* ⟶ *nth-digit* (*sum s* {*0..m*}) *t b* < *2* ^ *c*
using *sk-sum-bound b-def c-gt0 h0*
by (*metis atLeast0AtMost le-less-trans*)
moreover have ∀ *t k*. *k* ≤ *length p* − *1* ⟶ *nth-digit* (*s k*) *t b* < *2* ^ *c*
using *sk-bound b-def c-gt0 h0*
by (*metis le-less-trans*)

ultimately have *nth-digit* (∑ *k*≤*length p* − *1*. *s k*) *t b*
= (∑ *k*≤*length p* − *1*. *nth-digit* (*s k*) *t b*)
using *b-def c-gt0*
using *finite-sum-nth-digit-commute2*[**where** *?M* = *length p* − *1*]
by (*simp add*: *atMost-atLeast0*)

thus *?thesis* **using** *sk-sum-bound* **by** (*metis order-refl*)
qed

lemma *bitAND-sum-lt*: (∑ *k*≤*length p* − *1*. *nth-digit* (*s k* && *x k*) *t b*)
≤ (∑ *k*≤*length p* − *1*. *Seq k t*)
proof −
have (∑ *k*≤*length p* − *1*. *nth-digit* (*s k* && *x k*) *t b*)
= (∑ *k*≤*length p* − *1*. *nth-digit* (*s k*) *t b* && *nth-digit* (*x k*) *t b*)
using *bitAND-nth-digit-commute b-def* **by** *auto*
also have ... ≤ (∑ *k*≤*length p* − *1*. *nth-digit* (*s k*) *t b*)
using *bitAND-lt* **by** (*simp add*: *sum-mono*)
finally show *?thesis* **using** *Seq-def* **by** *auto*

```
qed

lemma states-unique-RAW:
  ∀k≤m. Seq k t = 1 ⟶ (∀j≤m. j ≠ k ⟶ Seq j t = 0)
proof -
  {
    fix k
    assume k≤m
    assume skt-1: Seq k t = 1

    have ∀j≤m. j ≠ k ⟶ Seq j t = 0
    proof -
      {
        fix j
        assume j≤m
        assume j ≠ k
        let ?fct = (λk. Seq k t)

        have Seq j t = 0
        proof (rule ccontr)
          assume Seq j t ≠ 0
          then have Seq j t + Seq k t > 1
            using skt-1 by auto
          moreover have sum ?fct {0..m} ≥ sum ?fct {j, k}
            using ‹j≤m› ‹k≤m› sum-mono2
                 by (metis atLeastAtMost-iff empty-subsetI finite-atLeastAtMost in-
sert-subset le0)
          ultimately have (∑k≤m. Seq k t) > 1
            by (simp add: ‹j ≠ k› atLeast0AtMost)
          thus False
            using sum-sk-bound[where ?t = t]
            by (auto simp: Seq-def m-def)
        qed
      }
      thus ?thesis by auto
    qed
  }
  thus ?thesis by auto
qed

lemma block-sum-radd-bound:
  shows ∀t. (∑R+ p l (λk. nth-digit (s k) t b)) ≤ 1
proof -
  {
    fix t
    have (∑R+ p l (λk. Seq k t)) ≤ (∑k≤length p - 1. Seq k t)
      unfolding sum-radd.simps
      by (simp add: atMost-atLeast0 sum-mono)
```

```
    hence (∑R+ p l (λk. Seq k t)) ≤ 1
      using sum-sk-bound[of t] Seq-def
      using dual-order.trans by blast
  }
  thus ?thesis using Seq-def by auto
qed

lemma block-sum-rsub-bound:
  shows ∀t. (∑R− p l (λk. nth-digit (s k && z l) t b)) ≤ 1
proof −
  {
    fix t
    have (∑R− p l (λk. nth-digit (s k && z l) t b))
          ≤ (∑k≤length p − 1. nth-digit (s k && z l) t b)
      unfolding sum-rsub.simps
      by (simp add: atMost-atLeast0 sum-mono)
    also have ... ≤ (∑k≤length p − 1. Seq k t)
      using bitAND-sum-lt[where ?x = λk. z l] by blast
    finally have (∑R− p l (λk. nth-digit (s k && z l) t b)) ≤ 1
      using sum-sk-bound[of t] Seq-def
      using dual-order.trans by blast
  }
  thus ?thesis using Seq-def by auto
qed

lemma block-sum-rsub-special-bound:
  shows ∀t. (∑R− p l (λk. nth-digit (s k) t b)) ≤ 1
proof −
  {
    fix t
    have (∑R− p l (λk. nth-digit (s k) t b))
          ≤ (∑k≤length p − 1. nth-digit (s k) t b)
      unfolding sum-rsub.simps
      by (simp add: atMost-atLeast0 sum-mono)
    then have (∑R− p l (λk. nth-digit (s k) t b)) ≤ 1
      using sum-sk-bound[of t]
      using dual-order.trans by blast
  }
  thus ?thesis using Seq-def by auto
qed

lemma block-sum-sadd-bound:
  shows ∀t. (∑S+ p j (λk. nth-digit (s k) t b)) ≤ 1
proof −
  {
    fix t
    have (∑S+ p j (λk. Seq k t)) ≤ (∑k≤length p − 1. Seq k t)
      unfolding sum-sadd.simps
      by (simp add: atMost-atLeast0 sum-mono)
```

hence $(\sum S+ \; p \; j \; (\lambda k. \; Seq \; k \; t)) \leq 1$
using *sum-sk-bound*[*of t*] *Seq-def*
using *dual-order.trans* **by** *blast*
}
thus *?thesis* **using** *Seq-def* **by** *auto*
qed

lemma *block-sum-ssub-bound*:
shows $\forall t. \; (\sum S- \; p \; j \; (\lambda k. \; nth\text{-}digit \; (s \; k \; \&\& \; z \; (l \; k)) \; t \; b)) \leq 1$
proof −
{
fix *t*
have $(\sum S- \; p \; j \; (\lambda k. \; nth\text{-}digit \; (s \; k \; \&\& \; z \; (l \; k)) \; t \; b))$
$\leq (\sum k{\leq}length \; p - 1. \; nth\text{-}digit \; (s \; k \; \&\& \; z \; (l \; k)) \; t \; b)$
unfolding *sum-ssub-nzero.simps*
by (*simp add*: *atMost-atLeast0 sum-mono*)
also have ... $\leq (\sum k{\leq}length \; p - 1. \; Seq \; k \; t)$
using *bitAND-sum-lt*[**where** $?x = \lambda k. \; z \; (l \; k)$] **by** *blast*
finally have $(\sum S- \; p \; j \; (\lambda k. \; nth\text{-}digit \; (s \; k \; \&\& \; z \; (l \; k)) \; t \; b)) \leq 1$
using *sum-sk-bound*[*of t*] *Seq-def*
using *dual-order.trans* **by** *blast*
}
thus *?thesis* **using** *Seq-def* **by** *auto*
qed

lemma *block-sum-szero-bound*:
shows $\forall t. \; (\sum S0 \; p \; j \; (\lambda k. \; nth\text{-}digit \; (s \; k \; \&\& \; (e - z \; (l \; k))) \; t \; b)) \leq 1$
proof −
{
fix *t*
have $(\sum S0 \; p \; j \; (\lambda k. \; nth\text{-}digit \; (s \; k \; \&\& \; e - z \; (l \; k)) \; t \; b))$
$\leq (\sum k{\leq}length \; p - 1. \; nth\text{-}digit \; (s \; k \; \&\& \; e - z \; (l \; k)) \; t \; b)$
unfolding *sum-ssub-zero.simps*
by (*simp add*: *atMost-atLeast0 sum-mono*)
also have ... $\leq (\sum k{\leq}length \; p - 1. \; Seq \; k \; t)$
using *bitAND-sum-lt*[**where** $?x = \lambda k. \; e - z \; (l \; k)$] **by** *blast*
finally have $(\sum S0 \; p \; j \; (\lambda k. \; nth\text{-}digit \; (s \; k \; \&\& \; e - z \; (l \; k)) \; t \; b)) \leq 1$
using *sum-sk-bound*[*of t*] *Seq-def*
using *dual-order.trans* **by** *blast*
}
thus *?thesis* **using** *Seq-def* **by** *auto*
qed

lemma *sum-radd-nth-digit-commute*:
shows $nth\text{-}digit \; (\sum R+ \; p \; l \; (\lambda k. \; s \; k)) \; t \; b = \sum R+ \; p \; l \; (\lambda k. \; nth\text{-}digit \; (s \; k) \; t \; b)$
proof −
have *a1*: $\forall t. \; \forall k{\leq}length \; p - 1. \; nth\text{-}digit \; (s \; k) \; t \; b < 2\hat{\ }c$
using *sk-bound h0* **by** (*meson le-less-trans*)

have *a2*: ∀ *t*. (∑ *R*+ *p l* (λ*k*. *nth-digit* (*s k*) *t b*)) < *2*^*c*
using *block-sum-radd-bound h0* **by** (*meson le-less-trans*)

show *?thesis*
using *a1 a2 c-gt0 b-def* **unfolding** *sum-radd.simps*
using *sum-nth-digit-commute*[**where** *?g* = λ*l k*. *isadd* (*p* ! *k*) ∧ *l* = *modifies* (*p* ! *k*)]
by *blast*
qed

lemma *sum-rsub-nth-digit-commute*:
shows *nth-digit* (∑ *R*− *p l* (λ*k*. *s k* && *z l*)) *t b*
= ∑ *R*− *p l* (λ*k*. *nth-digit* (*s k* && *z l*) *t b*)
proof −
have *a1*: ∀ *t*. ∀ *k*≤*length p* − *1*. *nth-digit* (*s k* && *z l*) *t b* < *2*^*c*
using *sk-bitAND-bound*[**where** *?x* = λ*k*. *z l*] *h0 le-less-trans* **by** *blast*

have *a2*: ∀ *t*. (∑ *R*− *p l* (λ*k*. *nth-digit* (*s k* && *z l*) *t b*)) < *2*^*c*
using *block-sum-rsub-bound h0* **by** (*meson le-less-trans*)

show *?thesis*
using *a1 a2 c-gt0 b-def* **unfolding** *sum-rsub.simps*
using *sum-nth-digit-commute*
[**where** *?g* = λ*l k*. *issub* (*p* ! *k*) ∧ *l* = *modifies* (*p* ! *k*) **and** *?fct* = λ*k*. *s k* && *z l*]
by *blast*
qed

lemma *sum-sadd-nth-digit-commute*:
shows *nth-digit* (∑ *S*+ *p j* (λ*k*. *s k*)) *t b* = ∑ *S*+ *p j* (λ*k*. *nth-digit* (*s k*) *t b*)
proof −
have *a1*: ∀ *t*. ∀ *k*≤*length p* − *1*. *nth-digit* (*s k*) *t b* < *2*^*c*
using *sk-bound h0* **by** (*meson le-less-trans*)

have *a2*: ∀ *t*. (∑ *S*+ *p j* (λ*k*. *nth-digit* (*s k*) *t b*)) < *2*^*c*
using *block-sum-sadd-bound h0* **by** (*meson le-less-trans*)

show *?thesis*
using *a1 a2 b-def c-gt0* **unfolding** *sum-sadd.simps*
using *sum-nth-digit-commute*[**where** *?g* = λ*j k*. *isadd* (*p* ! *k*) ∧ *j* = *goes-to* (*p* ! *k*)]
by *blast*
qed

lemma *sum-ssub-nth-digit-commute*:
shows *nth-digit* (∑ *S*− *p j* (λ*k*. *s k* && *z* (*l k*))) *t b*
= ∑ *S*− *p j* (λ*k*. *nth-digit* (*s k* && *z* (*l k*)) *t b*)
proof −
have *a1*: ∀ *t*. ∀ *k*≤*length p* − *1*. *nth-digit* (*s k* && *z* (*l k*)) *t b* < *2*^*c*

using *sk-bitAND-bound*[**where** *?x* = *λk. z* (*l k*)] *h0 le-less-trans* **by** *blast*

have *a2*: ∀ *t*. (∑ *S*− *p j* (*λk. nth-digit* (*s k* && *z* (*l k*)) *t b*)) < *2^c*
using *block-sum-ssub-bound h0* **by** (*meson le-less-trans*)

show *?thesis*
using *a1 a2 b-def c-gt0* **unfolding** *sum-ssub-nzero.simps*
using *sum-nth-digit-commute*
[**where** *?g* = *λj k. issub* (*p* ! *k*) ∧ *j* = *goes-to* (*p* ! *k*) **and** *?fct* = *λk. s k* && *z* (*l k*)]
by *blast*
qed

lemma *sum-szero-nth-digit-commute*:
shows *nth-digit* (∑ *S0 p j* (*λk. s k* && (*e* − *z* (*l k*)))) *t b*
= ∑ *S0 p j* (*λk. nth-digit* (*s k* && (*e* − *z* (*l k*))) *t b*)
proof −
have *a1*: ∀ *t*. ∀ *k*≤*length p* − *1*. *nth-digit* (*s k* && (*e* − *z* (*l k*))) *t b* < *2^c*
using *sk-bitAND-bound*[**where** *?x* = *λk. e* − *z* (*l k*)] *h0 le-less-trans* **by** *blast*

have *a2*: ∀ *t*. (∑ *S0 p j* (*λk. nth-digit* (*s k* && (*e* − *z* (*l k*))) *t b*)) < *2^c*
using *block-sum-szero-bound h0* **by** (*meson le-less-trans*)

show *?thesis*
using *a1 a2 b-def c-gt0* **unfolding** *sum-ssub-zero.simps*
using *sum-nth-digit-commute*
[**where** *?g* = *λj k. issub* (*p* ! *k*) ∧ *j* = *goes-to-alt* (*p* ! *k*) **and** *?fct* = *λk. s k* && *e* − *z* (*l k*)]
by *blast*
qed

lemma *block-bound-impl-fst-digit-zero*:
assumes *nth-digit x t b* ≤ *1*
shows (*nth-digit x t b*) ¡ *c* = *0*
using *assms* **apply** (*auto simp*: *nth-bit-def*)
by (*metis* (*no-types*, *opaque-lifting*) *c-gt0 div-le-dividend le-0-eq le-Suc-eq mod-0 mod-Suc*
mod-div-trivial numeral-2-eq-2 power-eq-0-iff power-mod)

lemma *sum-radd-block-bound*:
shows *nth-digit* (∑ *R*+ *p l* (*λk. s k*)) *t b* ≤ *1*
using *block-sum-radd-bound sum-radd-nth-digit-commute* **by** *auto*
lemma *sum-radd-fst-digit-zero*:
shows (*nth-digit* (∑ *R*+ *p l s*) *t b*) ¡ *c* = *0*
using *sum-radd-block-bound block-bound-impl-fst-digit-zero* **by** *auto*

lemma *sum-sadd-block-bound*:
shows *nth-digit* (∑ *S*+ *p j* (*λk. s k*)) *t b* ≤ *1*
using *block-sum-sadd-bound sum-sadd-nth-digit-commute* **by** *auto*

lemma *sum-sadd-fst-digit-zero*:
  shows (*nth-digit* ($\sum$ *S+ p j s*) *t b*) ¡ *c = 0*
  using *sum-sadd-block-bound block-bound-impl-fst-digit-zero* **by** *auto*

lemma *sum-ssub-block-bound*:
  shows *nth-digit* ($\sum$ *S− p j* ($\lambda k.$ *s k* && *z* (*l k*))) *t b* $\leq$ *1*
  using *block-sum-ssub-bound sum-ssub-nth-digit-commute* **by** *auto*
lemma *sum-ssub-fst-digit-zero*:
  shows (*nth-digit* ($\sum$ *S− p j* ($\lambda k.$ *s k* && *z* (*l k*))) *t b*) ¡ *c = 0*
  using *sum-ssub-block-bound block-bound-impl-fst-digit-zero* **by** *auto*

lemma *sum-szero-block-bound*:
  shows *nth-digit* ($\sum$ *S0 p j* ($\lambda k.$ *s k* && (*e − z* (*l k*)))) *t b* $\leq$ *1*
  using *block-sum-szero-bound sum-szero-nth-digit-commute* **by** *auto*
lemma *sum-szero-fst-digit-zero*:
  shows (*nth-digit* ($\sum$ *S0 p j* ($\lambda k.$ *s k* && (*e − z* (*l k*)))) *t b*) ¡ *c = 0*
  using *sum-szero-block-bound block-bound-impl-fst-digit-zero* **by** *auto*

lemma *sum-rsub-special-block-bound*:
  shows *nth-digit* ($\sum$ *R− p l* ($\lambda k.$ *s k*)) *t b* $\leq$ *1*
proof −
  have *a1*: $\forall$ *t k. k* $\leq$ *length p − 1* $\longrightarrow$ *nth-digit* (*s k*) *t b* < *2^c*
    using *sk-bound h0 le-less-trans* **by** *blast*
  have *a2*: $\forall$ *t.* $\sum$ *R− p l* ($\lambda k.$ *nth-digit* (*s k*) *t b*) < *2^c*
    using *block-sum-rsub-special-bound h0 le-less-trans* **by** *blast*

  have *nth-digit* ($\sum$ *R− p l* ($\lambda k.$ *s k*)) *t b* = $\sum$ *R− p l* ($\lambda k.$ *nth-digit* (*s k*) *t b*)
    using *a1 a2 b-def c-gt0* **unfolding** *sum-rsub.simps*
    using *sum-nth-digit-commute*[**where** *?g* = λ*l k. issub* (*p* ! *k*) $\wedge$ *l = modifies* (*p*
! *k*)]
    by *blast*

  thus *?thesis*
    using *block-sum-rsub-special-bound* **by** *auto*
qed

lemma *sum-state-special-block-bound*:
  shows *nth-digit* ($\sum$ *S+ p j* ($\lambda k.$ *s k*)
        + $\sum$ *S0 p j* ($\lambda k.$ *s k* && (*e − z* (*l k*)))) *t b* $\leq$ *1*
proof −
  have *aux-sum-zero*:
    $\sum$ *S0 p j* ($\lambda k.$ *nth-digit* (*s k*) *t b* && *nth-digit* (*e − z* (*l k*)) *t b*)
      $\leq$ $\sum$ *S0 p j* ($\lambda k.$ *nth-digit* (*s k*) *t b*)
    unfolding *sum-ssub-zero.simps*
    by (*auto simp*: *bitAND-lt sum-mono*)

  have *aux-addsub-excl*: (*if isadd* (*p* ! *k*) *then Seq k t else 0*)
                      + (*if issub* (*p* ! *k*) *then Seq k t else 0*)
                    = (*if isadd* (*p* ! *k*) $\vee$ *issub* (*p* ! *k*) *then Seq k t else 0*) **for** *k*

by *auto*

have *aux-sum-add-lt*:
$\sum S+ \ p \ j \ (\lambda k. \ Seq \ k \ t) \leq (\sum k = 0..length \ p - 1. \ if \ isadd \ (p \ ! \ k) \ then \ Seq \ k \ t \ else \ 0)$
unfolding *sum-sadd.simps* **by** (*simp add*: *sum-mono*)
have *aux-sum-sub-lt*:
$\sum S0 \ p \ j \ (\lambda k. \ Seq \ k \ t) \leq (\sum k = 0..length \ p - 1. \ if \ issub \ (p \ ! \ k) \ then \ Seq \ k \ t \ else \ 0)$
unfolding *sum-ssub-zero.simps* **by** (*simp add*: *sum-mono*)

have *nth-digit* $(\sum S+ \ p \ j \ (\lambda k. \ s \ k)$
$+ \sum S0 \ p \ j \ (\lambda k. \ s \ k \ \&\& \ e - z \ (l \ k))) \ t \ b$
$= nth\text{-}digit \ (\sum S+ \ p \ j \ (\lambda k. \ s \ k)) \ t \ b$
$+ \ nth\text{-}digit \ (\sum S0 \ p \ j \ (\lambda k. \ s \ k \ \&\& \ e - z \ (l \ k))) \ t \ b$
using *sum-sadd-fst-digit-zero sum-szero-fst-digit-zero block-additivity*
by (*auto simp*: *c-gt0 b-def*)
also have ... $= \sum S+ \ p \ j \ (\lambda k. \ nth\text{-}digit \ (s \ k) \ t \ b)$
$+ \sum S0 \ p \ j \ (\lambda k. \ nth\text{-}digit \ (s \ k \ \&\& \ e - z \ (l \ k)) \ t \ b)$
by (*simp add*: *sum-sadd-nth-digit-commute sum-szero-nth-digit-commute*)
also have ... $\leq \sum S+ \ p \ j \ (\lambda k. \ Seq \ k \ t) + \sum S0 \ p \ j \ (\lambda k. \ Seq \ k \ t)$
using *bitAND-nth-digit-commute aux-sum-zero*
unfolding *Seq-def* **by** (*simp add*: *b-def*)
also have ... $\leq (\sum k = 0..length \ p - 1. \ if \ isadd \ (p \ ! \ k) \ then \ Seq \ k \ t \ else \ 0) +$
$(\sum k = 0..length \ p - 1. \ if \ issub \ (p \ ! \ k) \ then \ Seq \ k \ t \ else \ 0)$
using *aux-sum-add-lt aux-sum-sub-lt* **by** *auto*
also have ... $= (\sum k \leq length \ p - 1. \ if \ (isadd \ (p \ ! \ k) \lor issub \ (p \ ! \ k))$
$then \ Seq \ k \ t \ else \ 0)$
using *aux-addsub-excl*
using *sum.distrib*[**where** $?g = \lambda k. \ if \ isadd \ (p \ ! \ k) \ then \ Seq \ k \ t \ else \ 0$
and $?h = \lambda k. \ if \ issub \ (p \ ! \ k) \ then \ Seq \ k \ t \ else \ 0$]
by (*auto simp*: *aux-addsub-excl atMost-atLeast0*)
also have ... $\leq (\sum k \leq length \ p - 1. \ Seq \ k \ t)$
by (*smt eq-iff le0 sum-mono*)

finally show *?thesis* **using** *sum-sk-bound*[*of t*] *Seq-def* **by** *auto*
qed
lemma *sum-state-special-fst-digit-zero*:
shows $(nth\text{-}digit \ (\sum S+ \ p \ j \ (\lambda k. \ s \ k)$
$+ \sum S0 \ p \ j \ (\lambda k. \ s \ k \ \&\& \ (e - z \ (modifies \ (p!k))))) \ t \ b)$ ¡ $c = 0$
using *sum-state-special-block-bound block-bound-impl-fst-digit-zero* **by** *auto*

Main three redution lemmas: Zero Indicators, Registers, States

lemma *Z*:
assumes $l<n$
shows $Zeq \ l \ t = (if \ Req \ l \ t > 0 \ then \ Suc \ 0 \ else \ 0)$
proof −
have *cond*: $2\hat{\ }c * (z \ l) = (r \ l + d) \ \&\& \ f$ **using** *m-eq mask-equations-def assms*

by *auto*
have *d-block*: ∀ *t*≤*q*. *nth-digit d t b* = *2^c* − *1* **using** *d-def b-def less-imp-diff-less nth-digit-gen-power-series*[*of* λ-. *2^c*−*1 c*] *c-gt0* **by** *auto*
have *rl-bound*: *t*≤*q* ⟶ *nth-digit* (*r l*) *t b* ¡ *c*= *0* **for** *t* **by** (*simp add*: *assms rl-fst-digit-zero*)
have *f-block*: ∀ *t*≤*q*. *nth-digit f t b* = *2^c*
using *f-def b-def less-imp-diff-less nth-digit-gen-power-series*[*of* λ-. *2^c c*] *c-gt0* **by** *auto*
then have ∀ *t*≤*q*.∀ *k*<*c*. *nth-digit f t b* ¡ *k* = *0* **by** (*simp add*: *aux-powertwo-digits*)

moreover have *AND-gen*: ∀ *t*≤*q*.∀ *k*≤*c*. *nth-digit* ((*r l* + *d*) && *f*) *t b* ¡ *k* = (*nth-digit* (*r l* + *d*) *t b* ¡ *k*) ∗ *nth-digit f t b* ¡ *k*
using *b-def digit-gen-pow2-reduct bitAND-digit-mult digit-gen-pow2-reduct le-imp-less-Suc* **by** *presburger*
ultimately have ∀ *t*≤*q*.∀ *k*<*c*. *nth-digit* ((*r l* + *d*) && *f*) *t b* ¡ *k* = *0* **using** *f-def* **by** *auto*
moreover have (*r l* + *d*) && *f* < *b* ^ *Suc q* **using** *lm0245*[*of r l* + *d f*] *masks-leq*[*of* (*r l* + *d*) && *f f*] *f-def*
proof−
have *2* < *b* **using** *b-def c-gt0 gr0-conv-Suc not-less-iff-gr-or-eq* **by** *fastforce*
then have *b^u* + *b^u* < *b*∗ *b^ u* **for** *u* **using** *zero-less-power*[*of b u*] *mult-less-mono1*[*of 2 b b^u*] **by** *linarith*
then have (∑ *t*∈{..<*q*}. *b* ^ *t*) < *b* ^ *q* **apply**(*induct q*, *auto*) **subgoal for** *q*
using *add-strict-right-mono*[*of sum* ((^) *b*) {..<*q*} *b^q b^q*] *less-trans* **by** *blast* **done**
then have (∑ *t*∈{..<*q*}.*2^c*∗ *b* ^ *t*) < *2^c* ∗ *b^ q* **using** *sum-distrib-left*[*of 2^c* λ*q*. *b^q* {..<*q*}]
zero-less-power[*of 2 c*] *mult-less-mono1*[*of sum* ((^) *b*) {..<*q*} *b* ^ *q 2^c*] **by** (*simp add*: *mult.commute*)
moreover have *2* ^ *c* ∗ *b* ^ *q* = *b* ^ *Suc q div 2* **using** *b-def* **by** *auto*
moreover have *f*= (∑ *t*∈{..<*q*}.*2^c*∗ *b* ^ *t*) + *2^c*∗*b^q*
using *f-def atLeastLessThanSuc-atLeastAtMost c-eq rm-constants-def gr0-conv-Suc lessThan-atLeast0* **by** *auto*
ultimately have *f* < *b* ^ *Suc q* **by** *linarith*
moreover have (*r l* + *d*) && *f* ≤ *f* **using** *lm0245*[*of r l* + *d f*] *masks-leq*[*of* (*r l* + *d*) && *f f*] **by** *auto*
ultimately show *?thesis* **by** *auto*
qed
then have *rldf0*: *t*>*q* ⟶ *nth-digit* ((*r l* + *d*) && *f*) *t b* = *0* **for** *t* **using** *nth-digit-def*[*of r l* + *d* && *f t b*]
div-less[*of r l* + *d* && *f b^t*] *b-def power-increasing*[*of Suc q t b*] **by** *auto*
moreover have ∀ *t*>*q*.∀ *k*<*c*. *nth-digit* ((*r l* + *d*) && *f*) *t b* ¡ *k* = *0* **using** *aux-lt-implies-mask rldf0* **by** *fastforce*
ultimately have *AND-zero*: ∀ *t*.∀ *k*<*c*. *nth-digit* ((*r l* + *d*) && *f*) *t b* ¡ *k* = *0* **using** *leI* **by** *blast*

have *0*<*k* ⟹ *k*< *Suc c* ⟹ *nth-digit* (*z l*) *t b* ¡ *k* = *nth-digit* ((*r l* + *d*) && *f*) (*Suc t*) *b* ¡ (*k* − *1*)
for *k* **using** *b-def nth-digit-bound digit-gen-pow2-reduct*[*of k Suc c z l t*] *aux-digit-shift*[*of*

z l c t + c ∗ t + k]
digit-gen-pow2-reduct[*of k−1 Suc c z l ∗ 2^c Suc t*] *cond* **by** (*simp add*: *add.commute add.left-commute mult.commute*)
then have *aux*: *0<k ⟹ k< Suc c ⟹ nth-digit* (*z l*) *t b* ¡ *k = 0* **for** *k* **using** *AND-zero* **by** *auto*
have *zl-formula*: *nth-digit* (*z l*) *t b* = *nth-digit* (*z l*) *t b* ¡ *0*
using *b-def digit-sum-repr*[*of nth-digit* (*z l*) *t b Suc c*]
proof −
have *nth-digit* (*z l*) *t b* < *2 ^ Suc c*
⟹ *nth-digit* (*z l*) *t b* = (∑ *k*∈{*0..<Suc c*}. *nth-digit* (*z l*) *t b* ¡ *k* ∗ *2 ^ k*)
using *b-def digit-sum-repr*[*of nth-digit* (*z l*) *t b Suc c*]
by (*simp add*: *atLeast0LessThan*)
hence *nth-digit* (*z l*) *t b* < *2 ^ Suc c*
⟹ *nth-digit* (*z l*) *t b* = *nth-digit* (*z l*) *t b* ¡ *0*
+ (∑ *k*∈{*0<..<Suc c*}. *nth-digit* (*z l*) *t b* ¡ *k* ∗ *2 ^ k*)
by (*metis One-nat-def atLeastSucLessThan-greaterThanLessThan mult-numeral-1-right numeral-1-eq-Suc-0 power-0 sum.atLeast-Suc-lessThan zero-less-Suc*)
thus *?thesis* **using** *aux nth-digit-bound b-def* **by** *auto*
qed

consider (*tleq*) *t≤q* |(*tgq*) *t>q* **by** *linarith*
then show *?thesis*
proof *cases*
case *tleq*
then have *t-bound*: *t ≤ q* **by** *auto*
have *nth-digit* ((*r l* + *d*) && *f*) *t b* ¡ *c* = (*nth-digit* (*r l* + *d*) *t b* ¡ *c*)
using *f-block bitAND-single-digit AND-gen t-bound* **by** *auto*
moreover have *nth-digit* (*r l* + *d* && *f*) *t b* < *2 ^ Suc c* **using** *nth-digit-def b-def* **by** *simp*
ultimately have *AND-all*:*nth-digit* ((*r l* + *d*) && *f*) *t b* = (*nth-digit* (*r l* + *d*) *t b* ¡ *c*) ∗ *2^c* **using** *AND-gen AND-zero*
using *digit-sum-repr*[*of nth-digit* ((*r l* + *d*) && *f*) *t b Suc c*] **by** *auto*

then have ∀ *k<c*. *nth-digit* (*2^c* ∗ (*z l*)) *t b* ¡ *k* = *0* **using** *cond AND-zero* **by** *metis*
moreover have *nth-digit* (*2^c* ∗ (*z l*)) *t b* ¡ *c* = *nth-digit* (*z l*) *t b* ¡ *0*
using *digit-gen-pow2-reduct*[*of c Suc c* (*2^c* ∗ (*z l*)) *t*]
digit-gen-pow2-reduct[*of 0 Suc c z l t*] *b-def* **by** (*simp add*: *aux-digit-shift mult.commute*)
ultimately have *zl0*: *nth-digit* (*2^c* ∗ (*z l*)) *t b* = *2^c* ∗ *nth-digit* (*z l*) *t b* ¡ *0*
using *digit-sum-repr*[*of nth-digit* (*2^c* ∗ (*z l*)) *t b Suc c*] *nth-digit-bound b-def* **by** *auto*

have *nth-digit* (*2^c* ∗ (*z l*)) *t b* = *2^c* ∗ *nth-digit* (*z l*) *t b* **using** *zl0 zl-formula* **by** *auto*
then have *zl-block*: *nth-digit* (*z l*) *t b* = *nth-digit* (*r l* + *d*) *t b* ¡ *c* **using** *AND-all cond* **by** *auto*

consider (*g0*) *Req l t* > *0* | (*e0*) *Req l t* = *0* **by** *auto*
then show *?thesis*
proof *cases*
case *e0*
show *?thesis* **using** *e0* **apply**(*auto simp add*: *Req-def Zeq-def*) **subgoal**
proof−
assume *asm*: *nth-digit* (*r l*) *t b* = *0*
have *add*:((*nth-digit d t b*) + (*nth-digit* (*r l*) *t b*)) ¡ *c* = *0* **by** (*simp add*: *asm d-block nth-bit-def t-bound*)
from *d-block t-bound* **have** *nth-digit d* (*t*−*1*) *b* ¡ *c* = *0*
using *add asm* **by** *auto*
then have (*nth-digit* (*r l* + *d*) *t b*) ¡ *c* = *0*
using *add digit-wise-block-additivity*[*of r l t c d c*] *rl-bound*[*of t*−*1*] *b-def asm t-bound c-gt0* **by** *auto*
then show *?thesis* **using** *zl-block* **by** *simp*
qed done
next
case *g0*
show *?thesis* **using** *g0* **apply**(*auto simp add*: *Req-def Zeq-def*) **subgoal**
proof −
assume *0* < *nth-digit* (*r l*) *t b*
then obtain *k0* **where** *k0-def*: *nth-digit* (*r l*) *t b* ¡ *k0* = *1* **using** *aux0-digit-wise-equiv* **by** *auto*
then have *k0*≤*c* **using** *nth-digit-bound*[*of r l t c*] *b-def aux-lt-implies-mask* **by** (*metis Suc-leI leI zero-neq-one*)
then have *k0bound*: *k0*<*c* **using** *rl-fst-digit-zero* **using** *k0-def le-less rl-bound t-bound* **by** *fastforce*
moreover have *d-dig*: ∀ *k*<*c*. *nth-digit d t b* ¡ *k* = *1* **using** *d-block t-bound nth-bit-def*[*of nth-digit d t b*]
by (*metis One-nat-def Suc-1 Suc-diff-Suc Suc-pred dmask-aux even-add even-power odd-iff-mod-2-eq-one*
one-mod-two-eq-one plus-1-eq-Suc zero-less-Suc zero-less-power)
ultimately have *nth-digit d t b* ¡ *k0* = *1* **by** *simp*
then have *bin-carry* (*nth-digit d t b*) (*nth-digit* (*r l*) *t b*) (*Suc k0*) = *1* **using**

k0-def sum-carry-formula carry-bounded less-eq-Suc-le **by** *simp*
moreover have ⋀*n*. *Suc k0* ≤ *n* ⟹ *n* < *c* ⟹ *bin-carry* (*nth-digit d t b*) (*nth-digit* (*r l*) *t b*) *n* =
Suc 0 ⟹ *bin-carry* (*nth-digit d t b*) (*nth-digit* (*r l*) *t b*) (*Suc n*) = *Suc 0* **subgoal for** *n*
proof−
assume *n*<*c bin-carry* (*nth-digit d t b*) (*nth-digit* (*r l*) *t b*) *n* = *Suc 0*
then show *?thesis* **using** *d-dig sum-carry-formula*
carry-bounded[*of* (*nth-digit d t b*) (*nth-digit* (*r l*) *t b*) *Suc n*] **by** *auto*
qed done
ultimately have *bin-carry* (*nth-digit d t b*) (*nth-digit* (*r l*) *t b*) *c* = *1* (**is** *?P c*)
using *dec-induct*[*of Suc k0 c ?P*] **by** (*simp add*: *Suc-le-eq k0bound*)

```
      then have add:((nth-digit d t b) + (nth-digit (r l) t b)) ¡ c = 1
        using sum-digit-formula[of nth-digit d t b nth-digit (r l) t b c]
          d-block nth-bit-def t-bound assms rl-fst-digit-zero by auto

      from d-block t-bound have nth-digit d (t−1) b ¡ c = 0
        by (smt aux-lt-implies-mask diff-le-self diff-less le-eq-less-or-eq le-trans
                zero-less-numeral zero-less-one zero-less-power)
      then have (nth-digit (r l + d) t b) ¡ c = 1 using add b-def t-bound
                block-additivity assms rl-fst-digit-zero c-gt0 d-block by (simp add:
add.commute)
      then show ?thesis using zl-block by simp
    qed done
  qed
next
  case tgq
  then have t-bound: q<t by auto

  have r l < b ^ q using reg-equations-def assms r-eq by auto
   then have rl0: nth-digit (r l) t b = 0 using t-bound nth-digit-def[of r l t b]
b-gt1
      power-strict-increasing[of q t b] by fastforce
  then have ∀k<c. nth-digit (2^c ∗ (z l)) t b ¡ k = 0 using cond AND-zero by
simp

  moreover have nth-digit (2^c ∗ (z l)) t b ¡ c = nth-digit (z l) t b ¡ 0
    using digit-gen-pow2-reduct[of c Suc c (2^c ∗ (z l)) t]
        digit-gen-pow2-reduct[of 0 Suc c z l t] b-def by (simp add: aux-digit-shift
mult.commute)
  ultimately have zl0: nth-digit (2^c ∗ (z l)) t b = 2^c ∗ nth-digit (z l) t b ¡ 0
    using digit-sum-repr[of nth-digit (2^c ∗ (z l)) t b Suc c] nth-digit-bound b-def
by auto
  have 0<k =⇒ k< Suc c =⇒ nth-digit (z l) t b ¡ k = nth-digit ((r l + d) &&
f) (Suc t) b ¡ (k − 1)
      for k using b-def nth-digit-bound digit-gen-pow2-reduct[of k Suc c z l t]
aux-digit-shift[of z l c t + c ∗ t + k]
      digit-gen-pow2-reduct[of k−1 Suc c z l ∗ 2^c Suc t] cond by (simp add:
add.commute add.left-commute mult.commute)

  then show ?thesis using Zeq-def Req-def cond rl0 zl0 rldf0 zl-formula t-bound
by auto
 qed
qed

lemma zl-le-rl: l<n =⇒ z l ≤ r l for l
proof −
 assume l: l < n
 have Zeq l t ≤ Req l t for t using Z l by auto
 hence nth-digit (z l) t b ≤ nth-digit (r l) t b for t
   using Zeq-def Req-def by auto
```

thus *?thesis* **using** *digitwise-leq b-gt1* **by** *auto*
qed

lemma *modifies-valid*: ∀ *k*≤*m*. *modifies* (*p*!*k*) < *n*
proof −
  have *reg-check*: *program-register-check p n*
    using *is-val* **by** (*cases ic*, *auto simp*: *is-valid-initial-def n-def is-valid-def*)
  {
    fix *k*
    assume *k* ≤ *m*
    then have *p* ! *k* ∈ *set p*
      by (*metis* ‹*k* ≤ *m*› *add-eq-if diff-le-self is-val le-antisym le-trans m-def*
          n-not-Suc-n not-less not-less0 nth-mem p-contains)
    then have *instruction-register-check n* (*p* ! *k*)
      using *reg-check* **by** (*auto simp*: *list-all-def*)
    then have *modifies* (*p*!*k*) < *n* **by** (*cases p* ! *k*, *auto simp*: *n-gt0*)
  }
  thus *?thesis* **by** *auto*
qed

lemma *seq-bound*: *k* ≤ *length p* − *1* ⟹ *Seq k t* ≤ *1*
  using *sk-bound Seq-def* **by** *blast*

lemma *skzl-bitAND-to-mult*:
  assumes *k* ≤ *length p* − *1*
  assumes *l* < *n*
  shows *nth-digit* (*z l*) *t b* && *nth-digit* (*s k*) *t b* = (*Zeq l t*) ∗ *Seq k t*
proof −
  have *nth-digit* (*z l*) *t b* && *nth-digit* (*s k*) *t b* = (*Zeq l t*) && *Seq k t*
    using *Zeq-def Seq-def* **by** *simp*
  also have ... = (*Zeq l t*) ∗ *Seq k t*
    using *bitAND-single-bit-mult-equiv*[*of* (*Zeq l t*) *Seq k t*] *seq-bound Z assms* **by** *auto*
  finally show *?thesis* **by** *auto*
qed

lemma *skzl-bitAND-to-mult2*:
  assumes *k* ≤ *length p* − *1*
  assumes ∀ *k* ≤ *length p* − *1*. *l k* < *n*
  shows (*1* − *nth-digit* (*z* (*l k*)) *t b*) && *nth-digit* (*s k*) *t b*
      = (*1* − *Zeq* (*l k*) *t*) ∗ *Seq k t*
proof −
  have (*1* − *nth-digit* (*z* (*l k*)) *t b*) && *nth-digit* (*s k*) *t b*
      = (*1* − *Zeq* (*l k*) *t*) && *Seq k t*
    using *Zeq-def Seq-def* **by** *simp*
  also have ... = (*1* − *Zeq* (*l k*) *t*) ∗ *Seq k t*
    using *bitAND-single-bit-mult-equiv*[*of* (*1* − *Zeq* (*l k*) *t*) *Seq k t*] *seq-bound Z assms* **by** *auto*

```
  finally show ?thesis by auto
qed

lemma state-equations-digit-commute:
 assumes t < q and j ≤ m
 defines l ≡ λk. modifies (p!k)
 shows nth-digit (s j) (Suc t) b =
           (∑ S+ p j (λk. Seq k t))
         + (∑ S− p j (λk. Zeq (l k) t ∗ Seq k t))
         + (∑ S0 p j (λk. (1 − Zeq (l k) t) ∗ Seq k t))
proof −
  define o′ :: nat where o′ ≡ if j = 0 then 1 else 0
  have o′-div: o′ div b = 0 using b-gt1 by (auto simp: o′-def)

  have l: ∀ k≤length p − 1. (l k) < n
    using l-def by (auto simp: m-def modifies-valid)

  have ∀ k. Suc 0 < 2^c using c-gt0 h0 by auto
  hence e-aux: ∀ tt. nth-digit e tt b = (if tt≤q then Suc 0 else 0)
    using e-def b-def c-gt0 nth-digit-gen-power-series[of λk. Suc 0 c q] by auto
  have zl-bounded: k≤m ⟹ ∀ t′. nth-digit (z (l k)) t′ b ≤ nth-digit e t′ b for k
  proof −
    assume k≤m
    from m-eq have ∀ l<n. z l ⪯ e using mask-equations-def by auto
    then have ∀ l<n. ∀ t′. (z l) ¡ t′ ≤ e ¡ t′ using masks-leq-equiv by auto
    then have k-lt-c: ∀ l<n. ∀ t′. ∀ k′<Suc c. nth-digit (z l) t′ b ¡ k′
                                  ≤ nth-digit e t′ b ¡ k′
      using digit-gen-pow2-reduct by (auto simp: b-def) (metis power-Suc)

    have k≥Suc c ⟹ x mod (2 ^ Suc c) div 2 ^ k = 0 for k x::nat
      by (simp only: drop-bit-take-bit flip: take-bit-eq-mod drop-bit-eq-div) simp
    then have ∀ k≥Suc c. nth-digit x y b ¡ k = 0 for x y
      using b-def nth-bit-def nth-digit-def by auto
    then have k-gt-c: ∀ l<n. ∀ t′. ∀ k′≥Suc c. nth-digit (z l) t′ b ¡ k′
                                       ≤ nth-digit e t′ b ¡ k′
      by auto
    from k-lt-c k-gt-c have ∀ l<n. ∀ t′. nth-digit (z l) t′ b ≤ nth-digit e t′ b
      using bitwise-leq by (meson not-le)
    thus ?thesis by (auto simp: modifies-valid l-def ‹k≤m›)
  qed

  have ∀ t k. k≤m ⟶ nth-digit (e − z (l k)) t b =
               nth-digit e t b − nth-digit (z (l k)) t b
    using zl-bounded block-subtractivity by (auto simp: c-gt0 b-def l-def)
  then have sum-szero-aux:
   ∀ t k. t<q ⟶ k≤m ⟶ nth-digit (e − z (l k)) t b = 1−nth-digit (z (l k)) t b
    using e-aux by auto

  have skzl-bound2: ∀ k≤length p − 1. (l k) < n ⟹
```

∀ *t*. ∀ *k*≤*length p* − *1*. *nth-digit* (*s k* && (*e* − *z* (*l k*))) *t b* < *2*^*c*
proof −
assume *l*: ∀ *k*≤*length p* − *1*. (*l k*) < *n*
have ∀ *t*. ∀ *k*≤*length p* − *1*. *nth-digit* (*s k* && (*e* − *z* (*l k*))) *t b*
= *nth-digit* (*s k*) *t b* && *nth-digit* (*e* − *z* (*l k*)) *t b*
using *bitAND-nth-digit-commute Zeq-def b-def* **by** *auto*

moreover have ∀ *t*<*q*. ∀ *k*≤*length p* − *1*.
nth-digit (*s k*) *t b* && *nth-digit* (*e* − *z* (*l k*)) *t b*
= *nth-digit* (*s k*) *t b* && (*1* − *nth-digit* (*z* (*l k*)) *t b*)
using *sum-szero-aux* **by** (*simp add*: *m-def*)

moreover have ∀ *t*. ∀ *k*≤*length p* − *1*.
nth-digit (*s k*) *t b* && (*1* − *nth-digit* (*z* (*l k*)) *t b*)
≤ *nth-digit* (*s k*) *t b*
using *Z l* **using** *lm0245 masks-leq* **by** (*simp add*: *lm0244*)

moreover have ∀ *t*. ∀ *k*≤*length p* − *1*. *nth-digit* (*s k*) *t b* < *2*^*c*
using *sk-bound h0* **by** (*meson le-less-trans*)

ultimately show *?thesis*
using *le-less-trans* **by** (*metis lm0244 masks-leq*)
qed

have *s j* = *o*′ + *b*∗∑ *S*+ *p j* (λ*k*. *s k*) + *b*∗∑ *S*− *p j* (λ*k*. *s k* && *z* (*modifies* (*p*!*k*)))
+ *b*∗∑ *S0 p j* (λ*k*. *s k* && (*e* − *z* (*modifies* (*p*!*k*))))
using *s-eq state-equations-def* ‹*j*≤*m*› **by** (*auto simp*: *o*′*-def*)
then have *s j div b*^*Suc t mod b* =
(*o*′ + *b*∗∑ *S*+ *p j* (λ*k*. *s k*)
+ *b*∗∑ *S*− *p j* (λ*k*. *s k* && *z* (*modifies* (*p*!*k*)))
+ *b*∗∑ *S0 p j* (λ*k*. *s k* && (*e* − *z* (*modifies* (*p*!*k*))))) *div b div b*^*t mod b*
by (*auto simp*: *algebra-simps div-mult2-eq*)
also have ... = (∑ *S*+ *p j* (λ*k*. *s k*)
+ ∑ *S*− *p j* (λ*k*. *s k* && *z* (*modifies* (*p*!*k*)))
+ ∑ *S0 p j* (λ*k*. *s k* && (*e* − *z* (*modifies* (*p*!*k*))))) *div b*^*t mod b*
using *o*′*-div*
by (*auto simp*: *algebra-simps div-mult2-eq*)
(*smt Nat.add-0-right add-mult-distrib2 b-gt1 div-mult-self2 gr-implies-not0*)

also have ... = *nth-digit* (∑ *S*− *p j* (λ*k*. *s k* && *z* (*l k*))
+ ∑ *S*+ *p j* (λ*k*. *s k*)
+ ∑ *S0 p j* (λ*k*. *s k* && (*e* − *z* (*l k*)))) *t b*
by (*auto simp*: *nth-digit-def l-def add.commute*)

also have ... = *nth-digit* (∑ *S*− *p j* (λ*k*. *s k* && *z* (*l k*))) *t b*

+ *nth-digit* (∑ *S+ p j* (λ*k*. *s k*)
+ ∑ *S0 p j* (λ*k*. *s k* && (*e* − *z* (*l k*)))) *t b*
using *block-additivity sum-ssub-fst-digit-zero sum-state-special-fst-digit-zero*
by (*auto simp*: *l-def c-gt0 b-def add.assoc*)
also have ... = *nth-digit* (∑ *S+ p j* (λ*k*. *s k*)) *t b*
+ *nth-digit* (∑ *S− p j* (λ*k*. *s k* && *z* (*l k*))) *t b*
+ *nth-digit* (∑ *S0 p j* (λ*k*. *s k* && (*e* − *z* (*l k*)))) *t b*
using *block-additivity sum-sadd-fst-digit-zero sum-szero-fst-digit-zero*
by (*auto simp*: *l-def c-gt0 b-def*)
also have ... = *nth-digit* (∑ *S+ p j* (λ*k*. *s k*)) *t b*
+ (∑ *S− p j* (λ*k*. *nth-digit* (*s k* && *z* (*l k*)) *t b*))
+ *nth-digit* (∑ *S0 p j* (λ*k*. *s k* && (*e* − *z* (*l k*)))) *t b*
using *sum-ssub-nth-digit-commute* **by** *auto*
also have ... = *nth-digit* (∑ *S+ p j* (λ*k*. *s k*)) *t b*
+ ∑ *S− p j* (λ*k*. *nth-digit* (*s k* && *z* (*l k*)) *t b*)
+ ∑ *S0 p j* (λ*k*. *nth-digit* (*s k* && (*e* − *z* (*l k*))) *t b*)
using *l-def l skzl-bound2 sum-szero-nth-digit-commute* **by** (*auto*)
also have ... = ∑ *S+ p j* (λ*k*. *nth-digit* (*s k*) *t b*)
+ ∑ *S− p j* (λ*k*. *nth-digit* (*s k* && *z* (*l k*)) *t b*)
+ ∑ *S0 p j* (λ*k*. *nth-digit* (*s k* && (*e* − *z* (*l k*))) *t b*)
using *sum-sadd-nth-digit-commute* **by** *auto*
also have ... = ∑ *S+ p j* (λ*k*. *nth-digit* (*s k*) *t b*)
+ ∑ *S− p j* (λ*k*. *nth-digit* (*z* (*l k*)) *t b* && *nth-digit* (*s k*) *t b*)
+ ∑ *S0 p j* (λ*k*. (*nth-digit* (*e* − *z* (*l k*)) *t b*) && *nth-digit* (*s k*) *t*
b)
using *bitAND-nth-digit-commute b-def* **by** (*auto simp*: *bitAND-commutes*)
also have ... = (∑ *S+ p j* (λ*k*. *nth-digit* (*s k*) *t b*))
+ (∑ *S− p j* (λ*k*. *nth-digit* (*z* (*l k*)) *t b* && *nth-digit* (*s k*) *t b*))
+ (∑ *S0 p j* (λ*k*. (*1* − *nth-digit* (*z* (*l k*)) *t b*) && *nth-digit* (*s k*) *t b*))
using *sum-szero-aux assms sum-ssub-zero.simps m-def* ‹*t*<*q*›
apply (*auto*) **using** *sum.cong atLeastAtMost-iff* **by** *smt*

ultimately have *s j div* (*b* ^ *Suc t*) *mod b* =
(∑ *S+ p j* (λ*k*. *nth-digit* (*s k*) *t b*))
+ (∑ *S− p j* (λ*k*. *nth-digit* (*z* (*l k*)) *t b* && *nth-digit* (*s k*) *t b*))
+ (∑ *S0 p j* (λ*k*. (*1* − *nth-digit* (*z* (*l k*)) *t b*) && *nth-digit* (*s k*) *t b*))
by *auto*

moreover have (∑ *S− p j* (λ*k*. *nth-digit* (*z* (*l k*)) *t b* && *nth-digit* (*s k*) *t b*))
= (∑ *S− p j* (λ*k*. *Zeq* (*l k*) *t* ∗ *Seq k t*))
using *skzl-bitAND-to-mult sum-ssub-nzero.simps l*
by (*smt atLeastAtMost-iff sum.cong*)

moreover have (∑ *S0 p j* (λ*k*. (*1* − *nth-digit* (*z* (*l k*)) *t b*) && *nth-digit* (*s k*)
t b))
= (∑ *S0 p j* (λ*k*. (*1* − *Zeq* (*l k*) *t*) ∗ *Seq k t*))
using *skzl-bitAND-to-mult2 sum-ssub-zero.simps l*
by (*auto*) (*smt atLeastAtMost-iff sum.cong*)

ultimately have *nth-digit* (*s j*) (*Suc t*) *b* =
($\sum$ *S+ p j* (λ*k*. *Seq k t*))
\+ ($\sum$ *S− p j* (λ*k*. *Zeq* (*l k*) *t* ∗ *Seq k t*))
\+ ($\sum$ *S0 p j* (λ*k*. (*1* − *Zeq* (*l k*) *t*) ∗ *Seq k t*))
using *Seq-def nth-digit-def* **by** *auto*

thus *?thesis* **by** *auto*
qed

lemma *aux-nocarry-sk*:
assumes *t*≤*q*
shows *i* ≠ *j* ⟶ *i*≤*m* ⟶ *j*≤*m* ⟶ *nth-digit* (*s i*) *t b* ∗ *nth-digit* (*s j*) *t b* = *0*
proof (*cases t*=*q*)
case *True*
have *j* < *m* ⟶ *Seq j q* = *0* **for** *j* **using** *s-bound Seq-def nth-digit-def* **by** *auto*
then show *?thesis* **using** *True Seq-def* **apply** *auto* **by** (*metis le-less less-nat-zero-code*)
next
case *False*
hence *k*≤*m* ∧ *nth-digit* (*s k*) *t b* = *1* ⟶
(∀ *j*≤*m*. *j* ≠ *k* ⟶ *nth-digit* (*s j*) *t b* = *0*) **for** *k*
using *states-unique-RAW*[*of t*] *Seq-def assms* **by** *auto*
thus *?thesis*
by (*auto*) (*metis One-nat-def le-neq-implies-less m-def not-less-eq sk-bound*)
qed

lemma *nocarry-sk*:
assumes *i* ≠ *j* **and** *i*≤*m* **and** *j*≤*m*
shows (*s i*) ¡ *k* ∗ (*s j*) ¡ *k* = *0*
proof −
have *reduct*: (*s i*) ¡ *k* = *nth-digit* (*s i*) (*k div Suc c*) *b* ¡ (*k mod Suc c*) **for** *i*
using *digit-gen-pow2-reduct*[*of k mod Suc c Suc c s i k div Suc c*] *b-def*
using *mod-less-divisor zero-less-Suc* **by** *presburger*
have *k div Suc c* ≤ *q* ⟶
nth-digit (*s i*) (*k div Suc c*) *b* ∗ *nth-digit* (*s j*) (*k div Suc c*) *b* = *0*
using *aux-nocarry-sk assms* **by** *auto*
moreover have *k div Suc c* > *q* ⟶
nth-digit (*s i*) (*k div Suc c*) *b* ∗ *nth-digit* (*s j*) (*k div Suc c*) *b* = *0*
using *nth-digit-def s-bound* **apply** *auto*
using *b-gt1 div-greater-zero-iff leD le-less less-trans mod-less neq0-conv power-increasing-iff*
by (*smt assms*)
ultimately have *nth-digit* (*s i*) (*k div Suc c*) *b* ¡ (*k mod Suc c*)
∗ *nth-digit* (*s j*) (*k div Suc c*) *b* ¡ (*k mod Suc c*) = *0*
using *nth-bit-def* **by** *auto*
thus *?thesis* **using** *reduct*[*of i*] *reduct*[*of j*] **by** *auto*
qed

lemma *commute-sum-rsub-bitAND*: $\sum$ *R− p l* (λ*k*. *s k* && *z l*) = $\sum$ *R− p l* (λ*k*. *s k*) && *z l*

proof −
  show *?thesis* **apply** (*auto simp*: *sum-rsub.simps*)
    using *m-def nocarry-sk aux-commute-bitAND-sum-if*[*of m*
      s λk. *issub* (*p* ! *k*) $\wedge$ *l* = *modifies* (*p* ! *k*) *z l*]
    by (*auto simp add*: *atMost-atLeast0*)
qed

lemma *sum-rsub-bound*: $l<n \Longrightarrow \sum R- \ p \ l \ (\lambda k. \ s \ k \ \&\& \ z \ l) \leq r \ l + \sum R+ \ p \ l \ s$
proof −
  assume $l<n$
  have $\sum R- \ p \ l \ (\lambda k. \ s \ k) \ \&\& \ z \ l \leq z \ l$ **by** (*auto simp*: *lm0245 masks-leq*)
  also have ... $\leq r \ l$ **using** *zl-le-rl* ‹$l<n$› **by** *auto*
  ultimately show *?thesis*
    using *commute-sum-rsub-bitAND* **by** (*simp add*: *trans-le-add1*)
qed

Obtaining single step register relations from multiple step register relations

lemma *mult-to-single-reg*:
  $c>0 \Longrightarrow l<n \Longrightarrow Req \ l \ (Suc \ t) = Req \ l \ t + (\sum R+ \ p \ l \ (\lambda k. \ Seq \ k \ t))$
    $- (\sum R- \ p \ l \ (\lambda k. \ (Zeq \ l \ t) * Seq \ k \ t))$ **for** *l t*
proof −
  assume *l*: $l<n$
  assume *c*: $c > 0$

  have *a-div*: $a \ div \ b = 0$ **using** *c-eq rm-constants-def B-def* **by** *auto*

  have *subtract-bound*: $\forall t'. \ nth\text{-}digit \ (\sum R- \ p \ l \ (\lambda k. \ s \ k \ \&\& \ z \ l)) \ t' \ b$
    $\leq nth\text{-}digit \ (r \ l + \sum R+ \ p \ l \ (\lambda k. \ s \ k)) \ t' \ b$
  proof −
    {
      fix t'
      have $nth\text{-}digit \ (z \ l) \ t' \ b \leq nth\text{-}digit \ (r \ l) \ t' \ b$
        using *Zeq-def Req-def Z l* **by** *auto*
      then have $nth\text{-}digit \ (\sum R- \ p \ l \ (\lambda k. \ s \ k)) \ t' \ b \ \&\& \ nth\text{-}digit \ (z \ l) \ t' \ b$
          $\leq nth\text{-}digit \ (r \ l) \ t' \ b$
        using *sum-rsub-special-block-bound*
        by (*meson dual-order.trans lm0245 masks-leq*)
      then have $nth\text{-}digit \ (\sum R- \ p \ l \ (\lambda k. \ s \ k \ \&\& \ z \ l)) \ t' \ b$
          $\leq nth\text{-}digit \ (r \ l) \ t' \ b$
        using *commute-sum-rsub-bitAND bitAND-nth-digit-commute b-def* **by** *auto*
      then have $nth\text{-}digit \ (\sum R- \ p \ l \ (\lambda k. \ s \ k \ \&\& \ z \ l)) \ t' \ b$
          $\leq nth\text{-}digit \ (r \ l) \ t' \ b + nth\text{-}digit \ (\sum R+ \ p \ l \ (\lambda k. \ s \ k)) \ t' \ b$
        by *auto*
      then have $nth\text{-}digit \ (\sum R- \ p \ l \ (\lambda k. \ s \ k \ \&\& \ z \ l)) \ t' \ b$
          $\leq nth\text{-}digit \ (r \ l + \sum R+ \ p \ l \ (\lambda k. \ s \ k)) \ t' \ b$
        using *block-additivity rl-fst-digit-zero sum-radd-fst-digit-zero*
        by (*auto simp*: *b-def c l*)
    }
    then show *?thesis* **by** *auto*

qed

define a' **where** $a' \equiv (\mathit{if}\ l = 0\ \mathit{then}\ a\ \mathit{else}\ 0)$
have a'-*div*: a' *div* $b = 0$ **using** *a-div* a'-*def* **by** *auto*

have *r l div* $(b * b \,\widehat{}\, t)$ *mod* $b =$
$(a' + b{*}r\ l + b{*}\sum R{+}\ p\ l\ (\lambda k.\ s\ k) - b{*}\sum R{-}\ p\ l\ (\lambda k.\ s\ k\ \&\&\ z\ l))$ *div* $(b * b \,\widehat{}\, t)$ *mod* b
using *r-eq reg-equations-def* **by** (*auto simp*: a'-*def l*)
also have ... =
$(a' + b * (r\ l + \sum R{+}\ p\ l\ (\lambda k.\ s\ k) - \sum R{-}\ p\ l\ (\lambda k.\ s\ k\ \&\&\ z\ l)))$ *div b div* $b\widehat{}t$ *mod b*
by (*auto simp*: *algebra-simps div-mult2-eq*)
(*metis Nat.add-diff-assoc add-mult-distrib2 mult-le-mono2 sum-rsub-bound l*)
also have ... =
$((r\ l + \sum R{+}\ p\ l\ (\lambda k.\ s\ k) - \sum R{-}\ p\ l\ (\lambda k.\ s\ k\ \&\&\ z\ l)) + a'\ \mathit{div}\ b)$ *div b* $\widehat{}\, t$ *mod b*
using *b-gt1* **by** *auto*
also have ... = $(r\ l + \sum R{+}\ p\ l\ (\lambda k.\ s\ k) - \sum R{-}\ p\ l\ (\lambda k.\ s\ k\ \&\&\ z\ l))$ *div b* $\widehat{}\, t$ *mod b*
using a'-*div* **by** *auto*
also have ... = *nth-digit* $(r\ l + \sum R{+}\ p\ l\ (\lambda k.\ s\ k) - \sum R{-}\ p\ l\ (\lambda k.\ s\ k\ \&\&\ z\ l))$ *t b*
using *nth-digit-def* **by** *auto*

also have ... = *nth-digit* $(r\ l + \sum R{+}\ p\ l\ (\lambda k.\ s\ k))$ *t b*
− *nth-digit* $(\sum R{-}\ p\ l\ (\lambda k.\ s\ k\ \&\&\ z\ l))$ *t b*
using *block-subtractivity subtract-bound*
by (*auto simp*: *c b-def*)
also have ... = *nth-digit* (*r l*) *t b*
+ *nth-digit* $(\sum R{+}\ p\ l\ (\lambda k.\ s\ k))$ *t b*
− *nth-digit* $(\sum R{-}\ p\ l\ (\lambda k.\ s\ k\ \&\&\ z\ l))$ *t b*
using *block-additivity rl-fst-digit-zero sum-radd-fst-digit-zero*
by (*auto simp*: *l b-def c*)
also have ... = *nth-digit* (*r l*) *t b*
+ $\sum R{+}\ p\ l\ (\lambda k.\ \mathit{nth\text{-}digit}\ (s\ k)\ t\ b)$
− $\sum R{-}\ p\ l\ (\lambda k.\ \mathit{nth\text{-}digit}\ (s\ k\ \&\&\ z\ l)\ t\ b)$
using *sum-radd-nth-digit-commute*
using *sum-rsub-nth-digit-commute*
by *auto*
ultimately have *r l div* $(b * b \,\widehat{}\, t)$ *mod* $b =$
(*nth-digit* (*r l*) *t b*)
+ $\sum R{+}\ p\ l\ (\lambda k.\ \mathit{nth\text{-}digit}\ (s\ k)\ t\ b)$
− $\sum R{-}\ p\ l\ (\lambda k.\ \mathit{nth\text{-}digit}\ (z\ l)\ t\ b\ \&\&\ \mathit{nth\text{-}digit}\ (s\ k)\ t\ b)$
using *bitAND-nth-digit-commute b-def* **by** (*simp add*: *bitAND-commutes*)

then show *?thesis* **using** *Req-def Seq-def nth-digit-def skzl-bitAND-to-mult l*
by (*auto simp*: *sum-rsub.simps*) (*smt atLeastAtMost-iff sum.cong*)
qed

Obtaining single step state relations from multiple step state relations

lemma *mult-to-single-state*:
fixes *t j* :: *nat*
defines $l \equiv \lambda k.\ modifies\ (p!k)$
shows $j \leq m \Longrightarrow t < q \Longrightarrow Seq\ j\ (Suc\ t) = (\sum S{+}\ p\ j\ (\lambda k.\ Seq\ k\ t))$
$+ (\sum S{-}\ p\ j\ (\lambda k.\ Zeq\ (l\ k)\ t * Seq\ k\ t))$
$+ (\sum S0\ p\ j\ (\lambda k.\ (1 - Zeq\ (l\ k)\ t) * Seq\ k\ t))$
proof −
assume $j \leq m$
assume $t < q$

have $nth\text{-}digit\ (s\ j)\ (Suc\ t)\ b =$
$(\sum S{+}\ p\ j\ (\lambda k.\ Seq\ k\ t))$
$+ (\sum S{-}\ p\ j\ (\lambda k.\ Zeq\ (l\ k)\ t * Seq\ k\ t))$
$+ (\sum S0\ p\ j\ (\lambda k.\ (1 - Zeq\ (l\ k)\ t) * Seq\ k\ t))$
using *state-equations-digit-commute* ‹$j \leq m$› ‹$t < q$› *l-def* **by** *auto*

then show *?thesis* **using** *nth-digit-def l-def Seq-def* **by** *auto*
qed

Conclusion: The central equivalence showing that the cell entries obtained from r s z indeed coincide with the correct cell values when executing the register machine. This statement is proven by induction using the single step relations for Req and Seq as well as the statement for Zeq.

lemma *rzs-eq*:
$l < n \Longrightarrow j \leq m \Longrightarrow t \leq q \Longrightarrow R\ ic\ p\ l\ t = Req\ l\ t \wedge Z\ ic\ p\ l\ t = Zeq\ l\ t \wedge S\ ic\ p\ j\ t = Seq\ j\ t$
proof (*induction t arbitrary*: *j l*)
have $m > 0$ **using** *m-def is-val is-valid-initial-def*[*of ic p*] *is-valid-def*[*of ic p*] **by** *auto*

case *0*

have *mod-aux0*: $Suc\ (b * k)\ mod\ b = 1$ **for** *k*
using *euclidean-semiring-cancel-class.mod-mult-self2*[*of 1 b k*] *b-gt1* **by** *auto*
have *step-state0*: $s\ 0 = 1 + b * \sum S{+}\ p\ 0\ (\lambda k.\ s\ k) + b * \sum S{-}\ p\ 0\ (\lambda k.\ s\ k\ \&\&\ z\ (modifies\ (p!k)))$
$+ b * \sum S0\ p\ 0\ (\lambda k.\ s\ k\ \&\&\ (e - z\ (modifies\ (p!k))))$
using *s-eq state-equations-def* **by** *auto*
hence $Seq\ 0\ 0 = 1$ **using** *Seq-def* **by** (*auto simp*: *nth-digit-def mod-aux0*)
hence *S00*: $Seq\ 0\ 0 = S\ ic\ p\ 0\ 0$ **using** *S-def is-val is-valid-initial-def*[*of ic*] **by** *auto*

have $s\ m = b\hat{\ }q$ **using** *s-eq state-equations-def* **by** *auto*
hence $Seq\ m\ 0 = 0$ **using** *Seq-def nth-digit-def c-eq rm-constants-def* **by** *auto*

hence *Sm0*: *S ic p m 0 = Seq m 0*
using *is-val is-valid-initial-def*[*of ic p a*] *S-def* ‹*m*>*0*› **by** *auto*

have *step-states*: $\forall$ *d*>*0*. *d*<*m* $\longrightarrow$ *s d* = *b*$*\sum$ *S*+ *p d* (λ*k*. *s k*)
+ *b*$*\sum$ *S*− *p d* (λ*k*. *s k* && *z* (*modifies* (*p*!*k*)))
+ *b*$*\sum$ *S0 p d* (λ*k*. *s k* && (*e* − *z* (*modifies* (*p*!*k*))))
using *s-eq state-equations-def* **by** *auto*
hence $\forall$ *k*>*0*. *k*<*m* $\longrightarrow$ *Seq k 0 = 0* **using** *Seq-def* **by** (*auto simp*: *nth-digit-def*)
hence *k*>*0* $\longrightarrow$ *k*<*m* $\longrightarrow$ *Seq k 0 = S ic p k 0* **for** *k*
using *S-def is-val is-valid-initial-def*[*of ic*] **by** *auto*

with *S00 Sm0* **have** *Sid*: *k*$\leq$*m* $\longrightarrow$ *Seq k 0 = S ic p k 0* **for** *k*
by (*cases k*=*0*; *cases k*=*m*; *auto*)

have *b* $*$ (*r 0* + $\sum$ *R*+ *p 0 s* − $\sum$ *R*− *p 0* (λ*k*. *s k* && *z 0*))
= *b* $*$ (*r 0* + $\sum$ *R*+ *p 0 s*) − *b* $*$ $\sum$ *R*− *p 0* (λ*k*. *s k* && *z 0*)
using *Nat.diff-mult-distrib2*[*of b r 0* + $\sum$ *R*+ *p 0 s* $\sum$ *R*− *p 0* (λ*k*. *s k* && *z 0*)] **by** *auto*
also have ... = *b* $*$ *r 0* + *b*$*$ $\sum$ *R*+ *p 0 s* − *b* $*$ $\sum$ *R*− *p 0* (λ*k*. *s k* && *z 0*)
using *Nat.add-mult-distrib2*[*of b r 0* $\sum$ *R*+ *p 0 s*] **by** *auto*
ultimately have *distrib*: *a* + *b* $*$ (*r 0* + $\sum$ *R*+ *p 0 s* − $\sum$ *R*− *p 0* (λ*k*. *s k* && *z 0*))
= *a* + *b* $*$ *r 0* + *b* $*$ $\sum$ *R*+ *p 0 s* − *b* $*$ $\sum$ *R*− *p 0* (λ*k*. *s k* && *z 0*)
by (*auto simp*: *algebra-simps*)
(*metis Nat.add-diff-assoc add-mult-distrib2 mult-le-mono2 n-gt0 sum-rsub-bound*)

hence *Req 0 0* = (*a* + *b* $*$ *r 0* + *b* $*$ $\sum$ *R*+ *p 0 s* − *b* $*$ $\sum$ *R*− *p 0* (λ*k*. *s k* && *z 0*)) *mod b*
using *Req-def nth-digit-def r-eq reg-equations-def* **by** *auto*
also have ... = (*a* + *b* $*$ (*r 0* + $\sum$ *R*+ *p 0 s* − $\sum$ *R*− *p 0* (λ*k*. *s k* && *z 0*))) *mod b*
using *distrib* **by** *auto*
finally have *Req 0 0 = a* **using** *c-eq rm-constants-def B-def* **by** *auto*
hence *R00*: *R ic p 0 0 = Req 0 0*
using *R-def is-val is-valid-initial-def*[*of ic p a*] **by** *auto*

have *rl-transform*: *l*>*0* $\longrightarrow$ *r l* = *b* $*$ *r l* + *b* $*$ $\sum$ *R*+ *p l s* − *b* $*$ $\sum$ *R*− *p l* (λ*k*. *s k* && *z l*)
using *reg-equations-def r-eq* ‹*l*<*n*› **by** *auto*

have *l*>*0* $\longrightarrow$ (*b* $*$ *r l* + *b* $*$ $\sum$ *R*+ *p l s* − *b* $*$ $\sum$ *R*− *p l* (λ*k*. *s k* && *z l*)) *mod b = 0*
using *Req-def nth-digit-def reg-equations-def r-eq* **by** *auto*
hence *l*>*0* $\longrightarrow$ *Req l 0 = 0*
using *Req-def rl-transform nth-digit-def* **by** *auto*
hence *l*>*0* $\Longrightarrow$ *Req l 0 = R ic p l 0* **using** *is-val is-valid-initial-def*[*of ic*] *R-def* **by** *auto*

```
  hence Rid: R ic p l 0 = Req l 0 using R00 by (cases l=0; auto)


  hence Zid: Z ic p l 0 = Zeq l 0 using Z Z-def 0 by auto

  show ?case using Sid Rid Zid ‹l<n› ‹j≤m› by auto
next
  case (Suc t)


  have Suc t≤q using Suc by auto
  then have t<q by auto

  have S-IH: k≤m ⟹ S ic p k t = Seq k t for k using Suc m-def by auto
  have Z-IH: ∀ l::nat. l<n ⟶ Z ic p l t = Zeq l t using Suc by auto


  from S-IH have S1: k≤m ⟹
        (if isadd (p ! k) ∧ l = modifies (p ! k) then Seq k t else 0)
      = (if isadd (p ! k) ∧ l = modifies (p ! k) then S ic p k t else 0) for k
    by auto
  have S2: k ∈ {0..length p−1} ⟹
       (if issub (p ! k) ∧ l = modifies (p ! k) then Zeq l t ∗ Seq k t else 0)
     = (if issub (p ! k) ∧ l = modifies (p ! k) then Zeq l t ∗ S ic p k t else 0) for k
    using Suc m-def by auto

  have Req l (Suc t) = Req l t + (∑R+ p l (λk. Seq k t)) − (∑R− p l (λk. (Zeq
l t) ∗ Seq k t))
    using mult-to-single-reg[of l] ‹l<n› by (auto simp: c-gt0)
  also have ... = R ic p l t + (∑R+ p l (λk. S ic p k t))
                        − (∑R− p l (λk. (Z ic p l t) ∗ S ic p k t))
    using Suc sum-radd.simps sum-rsub.simps S1 S2 m-def by auto
  finally have R: Req l (Suc t) = R ic p l (Suc t)
    using is-val ‹l<n› n-def lm04-06-one-step-relation-register[of ic p a l] by auto


  hence Z-suct: Zeq l (Suc t) = Z ic p l (Suc t) using Z Z-def ‹l<n› by auto

  have plength: length p ≤ Suc m by (simp add: m-def)


  have s m = b ^ q using s-eq state-equations-def by auto
  hence Seq m t = 0 using Seq-def ‹t<q› nth-digit-def apply auto
    using b-gt1 bx-aux by auto
  hence S ic p m t = 0 using Suc by auto
  hence fst (steps ic p t) ≠ m using S-def by auto
  hence fst (steps ic p t) < m using is-val m-def
    by (metis less-Suc-eq less-le-trans p-contains plength)
  hence nohalt: ¬ ishalt (p ! fst (steps ic p t)) using is-valid-def[of ic p]
```

is-valid-initial-def[*of ic p a*] *m-def is-val* **by** *auto*

have *j*<*length p* **using** ‹*j*≤*m*› *m-def*
by (*metis* (*full-types*) *diff-less is-val length-greater-0-conv less-imp-diff-less less-one*
list.size(*3*) *nat-less-le not-less not-less-zero p-contains*)
have *Seq j* (*Suc t*) = (∑ *S*+ *p j* (λ*k*. *Seq k t*))
+ (∑ *S*− *p j* (λ*k*. *Zeq* (*modifies* (*p*!*k*)) *t* ∗ *Seq k t*))
+ (∑ *S0 p j* (λ*k*. (*1* − *Zeq* (*modifies* (*p*!*k*)) *t*) ∗ *Seq k t*))
using *mult-to-single-state* ‹*j*≤*m*› ‹*t*<*q*› *c-gt0* **by** *auto*
also have ... = (∑ *S*+ *p j* (λ*k*. *Seq k t*))
+ (∑ *S*− *p j* (λ*k*. *Z ic p* (*modifies* (*p*!*k*)) *t* ∗ *Seq k t*))
+ (∑ *S0 p j* (λ*k*. (*1* − *Z ic p* (*modifies* (*p*!*k*)) *t*) ∗ *Seq k t*))
using *Z-IH modifies-valid sum-ssub-zero.simps sum-ssub-nzero.simps*
by (*auto simp*: *m-def*, *smt atLeastAtMost-iff sum.cong*)
also have ... = (∑ *S*+ *p j* (λ*k*. *S ic p k t*))
+ (∑ *S*− *p j* (λ*k*. *Z ic p* (*modifies* (*p*!*k*)) *t* ∗ *S ic p k t*))
+ (∑ *S0 p j* (λ*k*. (*1* − *Z ic p* (*modifies* (*p*!*k*)) *t*) ∗ *S ic p k t*))
using *S-IH sum-ssub-zero.simps sum-ssub-nzero.simps sum-sadd.simps*
by (*auto simp*: *m-def*, *smt atLeastAtMost-iff sum.cong*)
finally have *S*: *Seq j* (*Suc t*) = *S ic p j* (*Suc t*)
using *is-val lm04-07-one-step-relation-state*[*of ic p a j*] ‹*j*<*length p*› *nohalt* **by** *auto*

show *?case* **using** *R S Z-suct* **by** *auto*
qed

end

end

4.4 Arithmetizing equations are Diophantine

theory *Equation-Setup* **imports** *../Register-Machine/RegisterMachineSpecification*
../Diophantine/Diophantine-Relations

begin

locale *register-machine* =
fixes *p* :: *program*
and *n* :: *nat*
assumes *p-nonempty*: *length p* > *0*
and *valid-program*: *program-register-check p n*
assumes *n-gt-0*: *n* > *0*

begin

definition *m* :: *nat* **where**
m ≡ *length p* − *1*

lemma *modifies-yields-valid-register*:
  assumes $k < length\ p$
  shows *modifies* $(p!k) < n$
proof −
  have *instruction-register-check n* $(p!k)$
   using *valid-program assms list-all-length program-register-check.simps* **by** *auto*

  thus *?thesis* **by** (*cases p*!*k*, *auto simp*: *n-gt-0*)
qed

end

locale *rm-eq-fixes* = *register-machine* +
  fixes *a b c d e f* :: *nat*
    and *q* :: *nat*
    and *r z* :: *register* ⇒ *nat*
    and *s* :: *state* ⇒ *nat*

end

4.4.1 Preliminary: Register machine sums are Diophantine

theory *Register-Machine-Sums* **imports** *Diophantine-Relations*
  ../Register-Machine/RegisterMachineSimulation

begin

fun *sum-polynomial* :: (*nat* ⇒ *polynomial*) ⇒ *nat list* ⇒ *polynomial* **where**
  sum-polynomial f [] = *Const 0* |
  sum-polynomial f (*i*#*idxs*) = *f i* [+] *sum-polynomial f idxs*

lemma *sum-polynomial-eval*:
  peval (*sum-polynomial f idxs*) *a* = (∑ *k*=*0*..<*length idxs*. *peval* (*f* (*idxs*!*k*)) *a*)
proof (*induction idxs rule*: *List.rev-induct*)
  case *Nil*
  then show *?case* **by** *auto*
next
  case (*snoc x xs*)
  moreover have *suc*: *peval* (*sum-polynomial f* (*xs* @ [*x*])) *a* = *peval* (*sum-polynomial f* (*x* # *xs*)) *a*
    by (*induction xs*, *auto*)
  moreover have *list-property*: *xa* < *length xs* ⟹ (*xs* ! *xa*) = (*xs* @ [*x*]) ! *xa* **for** *xa*
    by (*simp add*: *nth-append*)
  ultimately show *?case* **by** *auto*
qed

definition *sum-program* :: *program* ⇒ (*nat* ⇒ *polynomial*) ⇒ *polynomial*
(‹[∑ -] -› [*100*, *100*] *100*) **where**
[∑ *p*] *f* ≡ *sum-polynomial f* [*0*..<*length p*]

lemma *sum-program-push*: *m* = *length ns* ⟹ *length l* = *length p* ⟹
peval ([∑ *p*] (λ*k*. *if g k then map* (λ*x*. *push-param x m*) *l* ! *k else h k*)) (*push-list a ns*)
= *peval* ([∑ *p*] (λ*k*. *if g k then l* ! *k else h k*)) *a*
unfolding *sum-program-def* **apply** (*induction p*, *auto*)
oops

definition *sum-radd-polynomial* :: *program* ⇒ *register* ⇒ (*nat* ⇒ *polynomial*) ⇒ *polynomial*
(‹[∑ *R*+] - - -›) **where**
[∑ *R*+] *p l f* ≡ [∑ *p*] (λ*k*. *if isadd* (*p*!*k*) ∧ *l* = *modifies* (*p*!*k*) *then f k else Const 0*)

lemma *sum-radd-polynomial-eval*[*defs*]:
assumes *length p* > *0*
shows *peval* ([∑ *R*+] *p l f*) *a* = (∑ *R*+ *p l* (λ*x*. *peval* (*f x*) *a*))
proof −
have *1*: *x* ≤ *length p* − *Suc 0* ⟹ *x* < *length p* **for** *x* **using** *assms* **by** *linarith*
have *2*: *x* ≤ *length p* − *Suc 0* ⟹ *peval* (*f* ([*0*..<*length p*] ! *x*)) *a* = *peval* (*f x*) *a* **for** *x*
using *assms*
by (*metis diff-Suc-less less-imp-diff-less less-le-not-le nat-neq-iff nth-upt plus-nat.add-0*)
show *?thesis*
unfolding *sum-radd-polynomial-def sum-program-def sum-radd.simps sum-polynomial-eval*
by (*auto*, *rule sum.cong*, *auto simp*: *1 2*)
qed

definition *sum-rsub-polynomial* :: *program* ⇒ *register* ⇒ (*nat* ⇒ *polynomial*) ⇒ *polynomial*
(‹[∑ *R*−] - - -›) **where**
[∑ *R*−] *p l f* ≡ [∑ *p*] (λ*k*. *if issub* (*p*!*k*) ∧ *l* = *modifies* (*p*!*k*) *then f k else Const 0*)

lemma *sum-rsub-polynomial-eval*[*defs*]:
assumes *length p* > *0*
shows *peval* ([∑ *R*−] *p l f*) *a* = (∑ *R*− *p l* (λ*x*. *peval* (*f x*) *a*))
proof −
have *1*: *x* ≤ *length p* − *Suc 0* ⟹ *x* < *length p* **for** *x* **using** *assms* **by** *linarith*
have *2*: *x* ≤ *length p* − *Suc 0* ⟹ *peval* (*f* ([*0*..<*length p*] ! *x*)) *a* = *peval* (*f x*) *a* **for** *x*
using *assms*
by (*metis diff-Suc-less less-imp-diff-less less-le-not-le nat-neq-iff nth-upt plus-nat.add-0*)
show *?thesis*
unfolding *sum-rsub-polynomial-def sum-program-def sum-rsub.simps sum-polynomial-eval*
by (*auto*, *rule sum.cong*, *auto simp*: *1 2*)

qed

definition *sum-sadd-polynomial* :: *program* ⇒ *state* ⇒ (*nat* ⇒ *polynomial*) ⇒ *polynomial*
(‹[$\sum$ *S*+] - - -›) **where**
[$\sum$ *S*+] *p d f* ≡ [$\sum$ *p*] (λ*k*. *if isadd* (*p*!*k*) ∧ *d* = *goes-to* (*p*!*k*) *then f k else Const 0*)

lemma *sum-sadd-polynomial-eval*[*defs*]:
assumes *length p* > *0*
shows *peval* ([$\sum$ *S*+] *p d f*) *a* = ($\sum$ *S*+ *p d* (λ*x*. *peval* (*f x*) *a*))
proof −
have *1*: *x* ≤ *length p* − *Suc 0* ⟹ *x* < *length p* **for** *x* **using** *assms* **by** *linarith*
have *2*: *x* ≤ *length p* − *Suc 0* ⟹ *peval* (*f* ([*0*..<*length p*] ! *x*)) *a* = *peval* (*f x*) *a* **for** *x*
using *assms*
by (*metis diff-Suc-less less-imp-diff-less less-le-not-le nat-neq-iff nth-upt plus-nat.add-0*)
show *?thesis*
unfolding *sum-sadd-polynomial-def sum-program-def sum-sadd.simps sum-polynomial-eval*
by (*auto*, *rule sum.cong*, *auto simp*: *1 2*)
qed

definition *sum-ssub-nzero-polynomial* :: *program* ⇒ *state* ⇒ (*nat* ⇒ *polynomial*) ⇒ *polynomial*
(‹[$\sum$ *S*−] - - -›) **where**
[$\sum$ *S*−] *p d f* ≡ [$\sum$ *p*] (λ*k*. *if issub* (*p*!*k*) ∧ *d* = *goes-to* (*p*!*k*) *then f k else Const 0*)

lemma *sum-ssub-nzero-polynomial-eval*[*defs*]:
assumes *length p* > *0*
shows *peval* ([$\sum$ *S*−] *p d f*) *a* = ($\sum$ *S*− *p d* (λ*x*. *peval* (*f x*) *a*))
proof −
have *1*: *x* ≤ *length p* − *Suc 0* ⟹ *x* < *length p* **for** *x* **using** *assms* **by** *linarith*
have *2*: *x* ≤ *length p* − *Suc 0* ⟹ *peval* (*f* ([*0*..<*length p*] ! *x*)) *a* = *peval* (*f x*) *a* **for** *x*
using *assms*
by (*metis diff-Suc-less less-imp-diff-less less-le-not-le nat-neq-iff nth-upt plus-nat.add-0*)
show *?thesis*
unfolding *sum-ssub-nzero-polynomial-def sum-program-def sum-ssub-nzero.simps sum-polynomial-eval*
by (*auto*, *rule sum.cong*, *auto simp*: *1 2*)
qed

definition *sum-ssub-zero-polynomial* :: *program* ⇒ *state* ⇒ (*nat* ⇒ *polynomial*) ⇒ *polynomial*
(‹[$\sum$ *S0*] - - -›) **where**
[$\sum$ *S0*] *p d f* ≡ [$\sum$ *p*] (λ*k*. *if issub* (*p*!*k*) ∧ *d* = *goes-to-alt* (*p*!*k*) *then f k else Const 0*)

lemma *sum-ssub-zero-polynomial-eval*[*defs*]:
assumes *length p* > *0*
shows *peval* ([$\sum$ *S0*] *p d f*) *a* = ($\sum$ *S0 p d* (λ*x*. *peval* (*f x*) *a*))
proof −
have *1*: *x* ≤ *length p* − *Suc 0* ⟹ *x* < *length p* **for** *x* **using** *assms* **by** *linarith*
have *2*: *x* ≤ *length p* − *Suc 0* ⟹ *peval* (*f* ([*0*..<*length p*] ! *x*)) *a* = *peval* (*f x*) *a* **for** *x*
using *assms*
by (*metis diff-Suc-less less-imp-diff-less less-le-not-le nat-neq-iff nth-upt plus-nat.add-0*)
show *?thesis*
unfolding *sum-ssub-zero-polynomial-def sum-program-def sum-ssub-zero.simps sum-polynomial-eval*
by (*auto*, *rule sum.cong*, *auto simp*: *1 2*)
qed

end
theory *RM-Sums-Diophantine* **imports** *Equation-Setup ../Diophantine/Register-Machine-Sums ../Diophantine/Binary-And*

begin

context *register-machine*
begin

definition *sum-ssub-nzero-of-bit-and* :: *polynomial* ⇒ *nat* ⇒ *polynomial list* ⇒ *polynomial list*
⇒ *relation*
(‹[- = $\sum$ *S*− - ′(- && -′)]›) **where**
[*x* = $\sum$ *S*− *d* (*s* && *z*)] ≡ *let x*′ = *push-param x* (*length p*);
s′ = *push-param-list s* (*length p*);
z′ = *push-param-list z* (*length p*)
in [∃ *length p*] [∀ <*length p*] (λ*i*. [*Param i* = *s*′!*i* && *z*′!*i*])
[∧] *x*′ [=] ([$\sum$ *S*−] *p d Param*)

lemma *sum-ssub-nzero-of-bit-and-dioph*[*dioph*]:
fixes *s z* :: *polynomial list* **and** *d* :: *nat* **and** *x*
shows *is-dioph-rel* [*x* = $\sum$ *S*− *d* (*s* && *z*)]
unfolding *sum-ssub-nzero-of-bit-and-def* **by** (*auto simp add*: *dioph*)

lemma *sum-rsub-nzero-of-bit-and-eval*:
fixes *z s* :: *polynomial list* **and** *d* :: *nat* **and** *x* :: *polynomial*
assumes *length s* = *Suc m length z* = *Suc m length p* > *0*
shows *eval* [*x* = $\sum$ *S*− *d* (*s* && *z*)] *a*
⟷ *peval x a* = $\sum$ *S*− *p d* (λ*k*. *peval* (*s*!*k*) *a* && *peval* (*z*!*k*) *a*) (**is** *?P* ⟷ *?Q*)
proof −
have *invariance*: ∀ *k*<*length p*. *y1 k* = *y2 k* ⟹ $\sum$ *S*− *p d y1* = $\sum$ *S*− *p d y2* **for** *y1 y2*

```
    unfolding sum-ssub-nzero.simps apply (intro sum.cong, simp)
   using ‹length p > 0› by auto (metis Suc-pred le-imp-less-Suc length-greater-0-conv)

  have len-ps: length s = length p
    using m-def ‹length s = Suc m› ‹length p > 0› by auto
  have len-pz: length z = length p
    using m-def ‹length z = Suc m› ‹length p > 0› by auto

  show ?thesis
  proof (rule)
    assume ?P
    thus ?Q
       using sum-ssub-nzero-of-bit-and-def ‹length p > 0› apply (auto simp add:
defs push-push)
      using push-push-map-i apply (simp add: push-param-list-def len-ps len-pz)
      unfolding list-eval-def apply (auto simp: assms len-ps len-pz invariance)
      apply (rule sum-ssub-nzero-cong) apply auto
      by (metis (no-types, lifting) One-nat-def assms(1) assms(2)
              le-imp-less-Suc len-ps m-def nth-map)

  next
    assume ?Q
    thus ?P
       using sum-ssub-nzero-of-bit-and-def ‹length p > 0› apply (auto simp add:
defs push-push)
      apply (rule exI[of - map (λk. peval (s ! k) a && peval (z ! k) a) [0..<length
p]], simp)
      using push-push push-push-map-i
      by (simp add: push-param-list-def invariance push-list-eval len-ps len-pz)
  qed
qed

definition sum-ssub-zero-of-bit-and :: polynomial ⇒ nat ⇒ polynomial list ⇒
polynomial list
                                  ⇒ relation
  (‹[- = ∑ S0 - '(- && -')]›) where
  [x = ∑ S0 d (s && z)] ≡ let x' = push-param x (length p);
                              s' = push-param-list s (length p);
                              z' = push-param-list z (length p)
                          in [∃ length p] [∀<length p] (λi. [Param i = s'!i && z'!i])
                                          [∧] x' [=] [∑ S0] p d Param

lemma sum-ssub-zero-of-bit-and-dioph[dioph]:
  fixes s z :: polynomial list and d :: nat and x
  shows is-dioph-rel [x = ∑ S0 d (s && z)]
  unfolding sum-ssub-zero-of-bit-and-def by (auto simp add: dioph)

lemma sum-rsub-zero-of-bit-and-eval:
  fixes z s :: polynomial list and d :: nat and x :: polynomial
```

```
  assumes length s = Suc m length z = Suc m length p > 0
  shows eval [x = ∑ S0 d (s && z)] a
        ⟷ peval x a = ∑ S0 p d (λk. peval (s!k) a && peval (z!k) a) (is ?P ⟷
?Q)
proof −
  have invariance: ∀ k<length p. y1 k = y2 k ⟹ ∑ S0 p d y1 = ∑ S0 p d y2 for
y1 y2
    unfolding sum-ssub-zero.simps apply (intro sum.cong, simp)
    using ‹length p > 0› by auto (metis Suc-pred le-imp-less-Suc length-greater-0-conv)

  have len-ps: length s = length p
    using m-def ‹length s = Suc m› ‹length p > 0› by auto
  have len-pz: length z = length p
    using m-def ‹length z = Suc m› ‹length p > 0› by auto

  show ?thesis
  proof (rule)
    assume ?P
    thus ?Q
     using sum-ssub-zero-of-bit-and-def ‹length p > 0› apply (auto simp add: defs
push-push)
      using push-push-map-i apply (simp add: push-param-list-def len-ps len-pz)
      unfolding list-eval-def apply (auto simp: assms len-ps len-pz invariance)
      apply (rule sum-ssub-zero-cong) apply auto
      by (metis (no-types, lifting) One-nat-def assms(1) assms(2)
              le-imp-less-Suc len-ps m-def nth-map)
  next
    assume ?Q
    thus ?P
     using sum-ssub-zero-of-bit-and-def ‹length p > 0› apply (auto simp add: defs
push-push)
      apply (rule exI[of - map (λk. peval (s ! k) a && peval (z!k) a) [0..<length
p]], simp)
      using push-push push-push-map-i
      by (simp add: push-param-list-def invariance push-list-eval len-ps len-pz)
  qed
qed

end

end
```

4.4.2 Register Equations

```
theory Register-Equations imports ../Register-Machine/MultipleStepRegister
                              Equation-Setup ../Diophantine/Register-Machine-Sums
                               ../Diophantine/Binary-And HOL−Library.Rewrite

begin
```

context *rm-eq-fixes*
begin

Equation 4.22

definition *register-0* :: *bool* **where**
register-0 ≡ *r 0* = *a* + *b*∗*r 0* + *b*∗∑ *R*+ *p 0 s* − *b*∗∑ *R*− *p 0* (λ*k*. *s k* && *z 0*)

Equation 4.23

definition *register-l* :: *bool* **where**
register-l ≡ ∀ *l*>*0*. *l* < *n* ⟶ *r l* = *b*∗*r l* + *b*∗∑ *R*+ *p l s* − *b*∗∑ *R*− *p l* (λ*k*. *s k* && *z l*)

Extra equation not in Matiyasevich's book

definition *register-bound* :: *bool* **where**
register-bound ≡ ∀ *l* < *n*. *r l* < *b* ^ *q*

definition *register-equations* :: *bool* **where**
register-equations ≡ *register-0* ∧ *register-l* ∧ *register-bound*

end

context *register-machine*
begin

definition *sum-rsub-of-bit-and* :: *polynomial* ⇒ *nat* ⇒ *polynomial list* ⇒ *polynomial*
⇒ *relation*
(‹[- = ∑ *R*− - ′(- && -′)]›) **where**
[*x* = ∑ *R*− *d* (*s* && *zl*)] ≡ *let x′* = *push-param x* (*length p*);
s′ = *push-param-list s* (*length p*);
zl′ = *push-param zl* (*length p*)
in [∃ *length p*] [∀ <*length p*] (λ*i*. [*Param i* = *s′*!*i* && *zl′*])
[∧] *x′* [=] [∑ *R*−] *p d Param*

lemma *sum-rsub-of-bit-and-dioph*[*dioph*]:
fixes *s* :: *polynomial list* **and** *d* :: *nat* **and** *x zl* :: *polynomial*
shows *is-dioph-rel* [*x* = ∑ *R*− *d* (*s* && *zl*)]
unfolding *sum-rsub-of-bit-and-def* **by** (*auto simp add*: *dioph*)

lemma *sum-rsub-of-bit-and-eval*:
fixes *z s* :: *polynomial list* **and** *d* :: *nat* **and** *x* :: *polynomial*
assumes *length s* = *Suc m length p* > *0*
shows *eval* [*x* = ∑ *R*− *d* (*s* && *zl*)] *a*
⟷ *peval x a* = ∑ *R*− *p d* (λ*k*. *peval* (*s*!*k*) *a* && *peval zl a*) (**is** *?P* ⟷ *?Q*)
proof −

have *invariance*: ∀ *k*<*length p*. *y1 k* = *y2 k* ⟹ ∑*R*− *p d y1* = ∑*R*− *p d y2*
for *y1 y2*
unfolding *sum-rsub.simps* **apply** (*intro sum.cong*, *simp*)
using ‹*length p* > *0*› **by** *auto* (*metis Suc-pred le-imp-less-Suc length-greater-0-conv*)

have *len-ps*: *length s* = *length p*
using *m-def* ‹*length s* = *Suc m*› ‹*length p* > *0*› **by** *auto*

have *aux1*: *peval* ([∑ *R*−] *p l f*) *a* = ∑*R*− *p l* (λ*x*. *peval* (*f x*) *a*) **for** *l f*
using *defs* ‹*length p* > *0*› **by** *auto*

show *?thesis*
proof (*rule*)
assume *?P*
thus *?Q*
unfolding *sum-rsub-of-bit-and-def*
using *aux1* **apply** *simp*
apply (*auto simp add*: *aux1 push-push defs*)
using *push-push-map-i* **apply** (*simp add*: *push-param-list-def len-ps*)
unfolding *list-eval-def* **apply** (*simp add*: *assms len-ps invariance*)
using *assms*(*2*) *invariance len-ps sum-rsub-polynomial-eval* **by** *force*
next
assume *?Q*
thus *?P*
unfolding *sum-rsub-of-bit-and-def* **apply** (*auto simp add*: *aux1 defs push-push*)
apply (*rule exI*[*of - map* (λ*k*. *peval* (*s* ! *k*) *a* && *peval zl a*) [*0*..<*length p*]], *simp*)
using *push-push push-push-map-i* **apply** (*simp add*: *push-param-list-def len-ps*)
using *invariance len-ps push-list-eval* ‹*length p* > *0*› *defs* **by** *simp*
qed
qed

lemma *register-0-dioph*[*dioph*]:
fixes *A b* :: *polynomial*
fixes *r z s* :: *polynomial list*
assumes *length r* = *n length z* = *n length s* = *Suc m*
defines *DR* ≡ *LARY* (λ*ll*. *rm-eq-fixes.register-0 p* (*ll*!*0*!*0*) (*ll*!*0*!*1*)
(*nth* (*ll*!*1*)) (*nth* (*ll*!*2*)) (*nth* (*ll*!*3*))) [[*A*, *b*], *r*, *z*, *s*]
shows *is-dioph-rel DR*
proof −
let *?N* = *1*
define *A′ b′ r′ z′ s′* **where** *pushed-def*: *A′* = *push-param A ?N b′* = *push-param b ?N*
r′ = *map* (λ*x*. *push-param x ?N*) *r z′* = *map* (λ*x*. *push-param x ?N*) *z*
s′ = *map* (λ*x*. *push-param x ?N*) *s*

define *DS* **where** *DS* ≡ [∃] ([*Param 0* = ∑*R*− *0* (*s′* && (*z′*!*0*))] [∧]

r'!0 [=] *A'* [+] *b'* [*] *r'!0* [+] *b'* [*] ([∑ *R*+] *p 0* (*nth s'*))
[−] *b'* [*] (*Param 0*))

have *length p* > *0* **using** *p-nonempty* **by** *auto*
have *n* > *0* **using** *n-gt-0* **by** *auto*

have *length p* = *length s*
using ‹*length s* = *Suc m*› *m-def* ‹*length p* > *0*› **by** *auto*
have *length s'* = *length s*
unfolding *pushed-def* **by** *auto*
have *length z* > *0*
using ‹*length z* = *n*› ‹*n* > *0*› **by** *simp*
have *length r* > *0*
using ‹*length r* = *n*› ‹*n* > *0*› **by** *simp*

have *eval DS a* = *eval DR a* **for** *a*
proof −

have *sum-radd-push*: ∑ *R*+ *p 0* (λ*x*. *peval* (*s'* ! *x*) (*push a k*)) = ∑ *R*+ *p 0* (*list-eval s a*) **for** *k*
unfolding *sum-radd.simps pushed-def* **apply** (*intro sum.cong*, *simp*)
using *push-push-map1* ‹*length p* = *length s*› ‹*length s* = *Suc m*› **by** *simp*

have *sum-rsub-push*: ∑ *R*− *p 0* (λ*x*. *peval* (*s'* ! *x*) (*push a k*) && *peval* (*z'* ! *0*) (*push a k*))
= ∑ *R*− *p 0* (λ*x*. *list-eval s a x* && *peval* (*z* ! *0*) *a*) **for** *k*
unfolding *sum-rsub.simps pushed-def* **apply** (*intro sum.cong*, *simp*)
using *push-push-map1* ‹*length p* = *length s*› ‹*length s* = *Suc m*› ‹*length z* > *0*›
by (*simp add*: *list-eval-def*)

have *1*: *peval* ([∑ *R*−] *p l f*) *a* = ∑ *R*− *p l* (λ*x*. *peval* (*f x*) *a*) **for** *f l*
using *defs* ‹*length p* > *0*› **by** *auto*

show *?thesis*
unfolding *DS-def rm-eq-fixes.register-0-def*
register-machine-axioms rm-eq-fixes-def **apply** (*simp add*: *defs*)
using ‹*length p* > *0*› **apply** (*simp add*: *sum-rsub-of-bit-and-eval* ‹*length s'* = *length s*›
‹*length s* = *Suc m*›)
apply (*simp add*: *sum-radd-push sum-rsub-push*)
unfolding *pushed-def* **using** *push-push1 push-push-map1* ‹*length r* > *0*› **apply** *simp*
unfolding *DR-def assms defs* ‹*length p* > *0*›
using *rm-eq-fixes-def rm-eq-fixes.register-0-def register-machine-axioms* **apply** (*simp*)
using ‹*length z* > *0*› *push-def list-eval-def 1* **apply** (*simp add*: *1 defs* ‹*length p* > *0*›)
using *One-nat-def sum-radd-push* **unfolding** *pushed-def*(*5*) *list-eval-def* **by**

presburger
  qed

  moreover have *is-dioph-rel DS*
    unfolding *DS-def* **by** (*simp add*: *dioph*)

  ultimately show *?thesis*
    by (*auto simp*: *is-dioph-rel-def*)
qed

lemma *register-l-dioph*[*dioph*]:
  fixes *b* :: *polynomial*
  fixes *r z s* :: *polynomial list*
  assumes *length r* = *n length z* = *n length s* = *Suc m*
  defines *DR* ≡ *LARY* (λ*ll*. *rm-eq-fixes.register-l p n* (*ll*!*0*!*0*)
                              (*nth* (*ll*!*1*)) (*nth* (*ll*!*2*)) (*nth* (*ll*!*3*))) [[*b*], *r*, *z*, *s*]
  shows *is-dioph-rel DR*
proof −
  define *indices* **where** *indices* ≡ [*Suc 0*..<*n*]

  let *?N* = *length indices* + *1*
  define *b′ r′ z′ s′* **where** *pushed-def*: *b′* = *push-param b ?N*
                                          r′ = *map* (λ*x*. *push-param x ?N*) *r*
                                          z′ = *map* (λ*x*. *push-param x ?N*) *z*
                                          s′ = *map* (λ*x*. *push-param x ?N*) *s*

  define *param-l-is-sum-rsub-of-bitand* **where**
    param-l-is-sum-rsub-of-bitand ≡ λ*l*. [*Param l* = ∑ *R*− *l* (*s′* && (*z′*!*l*))]
  define *params-are-sum-rsub-of-bitand* **where**
    params-are-sum-rsub-of-bitand ≡ [∀ *in indices*] *param-l-is-sum-rsub-of-bitand*
  define *single-register* **where**
    single-register ≡ λ*l*. *r′*!*l* [=] *b′* [∗] *r′*!*l* [+] *b′* [∗] ([∑ *R*+] *p l* (*nth s′*)) [−] *b′* [∗]
(*Param l*)

  define *DS* **where** *DS* ≡ [∃ *n*] *params-are-sum-rsub-of-bitand* [∧] [∀ *in indices*]
single-register

  have *length p* > *0* **using** *p-nonempty* **by** *auto*
  have *n* > *0* **using** *n-gt-0* **by** *auto*
  have *length p* = *length s*
    using ‹*length s* = *Suc m*› *m-def* ‹*length p* > *0*› **by** *auto*
  have *length s′* = *length s*
    unfolding *pushed-def* **by** *auto*
  have *length z* > *0*
    using ‹*length z* = *n*› ‹*n* > *0*› **by** *simp*
  have *length r* > *0*
    using ‹*length r* = *n*› ‹*n* > *0*› **by** *simp*
  have *length indices* + *1* = *n*
    unfolding *indices-def* ‹*n*>*0*›

using *Suc-pred′* ‹*n* > *0*› *length-upt* **by** *presburger*
have *length s′* = *Suc m*
using ‹*length s′* = *length s*› ‹*length s* = *Suc m*› **by** *auto*

have *eval DS a* = *eval DR a* **for** *a*
proof −

have *eval-to-peval*:
eval [*polynomial.Param* (*indices* ! *k*)
= $\sum$ *R*− *indices* ! *k* (*s′* && *z′* ! (*indices* ! *k*))] *y*
⟷(*peval* (*polynomial.Param* (*indices* ! *k*)) *y*
= $\sum$ *R*− *p* (*indices* ! *k*) (λ*ka*. *peval* (*s′* ! *ka*) *y* && *peval* (*z′* ! (*indices* ! *k*)) *y*)) **for** *k y*
using *sum-rsub-of-bit-and-eval* ‹*length p* > *0*› ‹*length s′* = *Suc m*› **by** *auto*

have *b′-unfold*: *peval b′* (*push-list a ks*) = *peval b a* **if** *length ks* = *n* **for** *ks*
unfolding *pushed-def* **using** *indices-def push-push that* ‹*length indices* + *1* = *n*› **by** *auto*

have *r′-unfold*: *peval* (*r′* ! (*indices* ! *k*)) (*push-list a ks*) = *peval* (*r*!(*indices*!*k*)) *a*
if *k* < *length indices* **and** *length ks* = *n* **for** *k ks*
using *indices-def push-push pushed-def that*(*1*) *that*(*2*) ‹*length r* = *n*› **by** *auto*

have *Param-unfold*: *peval* (*Param* (*indices* ! *k*)) (*push-list a ks*) = *ks*!(*indices*!*k*)

if *k* < *length indices* **and** *length ks* = *n* **for** *k ks*
using *One-nat-def Suc-pred indices-def length-upt nat-add-left-cancel-less*
nth-upt peval.simps(*2*) *plus-1-eq-Suc push-list-eval that*(*1*) *that*(*2*) **by** (*metis* ‹*0* < *n*›)

have *unfold-4*: *push-list a ks* (*indices* ! *k*) = *ks*!(*indices*!*k*)
if *k* < *length indices* **and** *length ks* = *n* **for** *k ks*
using *Param-unfold that*(*1*) *that*(*2*) **by** *force*

have *unfold-sum-radd*: $\sum$ *R*+ *p* (*indices* ! *k*) (λ*x*. *peval* (*s′* ! *x*) (*push-list a ks*))
= $\sum$ *R*+ *p* (*indices* ! *k*) (*list-eval s a*)
if *length ks* = *n* **for** *k ks*
apply (*rule sum-radd-cong*) **unfolding** *pushed-def*
using *push-push-map-i*[*of ks n - s a*] ‹*length indices* + *1* = *n*› *that*
using ‹*length p* = *length s*›
by (*metis* ‹*0* < *length p*› *add.left-neutral add-lessD1 le-neq-implies-less less-add-one*
less-diff-conv less-diff-conv2 nat-le-linear not-add-less1)

have *unfold-sum-rsub*: $\sum$ *R*− *p* (*indices* ! *k*) (λ*ka*. *peval* (*s′* ! *ka*) (*push-list a ks*)
&& *peval* (*z′* ! (*indices* ! *k*)) (*push-list a ks*))
= $\sum$ *R*− *p* (*indices* ! *k*) (λ*ka*. *list-eval s a ka*

&& *peval* (*z* ! (*indices* ! *k*)) *a*)
if *length ks* = *n* **for** *k ks*
apply (*rule sum-rsub-cong*) **unfolding** *pushed-def*
using *push-push-map-i*[*of ks n - s a*] **unfolding** ‹*length indices* + *1* = *n*›
using ‹*length p* = *length s*› *assms* **apply** *simp*
using *nth-map*[*of - z* λ*x*. *push-param x* (*Suc* (*length indices*))]
using *modifies-yields-valid-register* ‹*length z* = *n*›
by (*smt assms le-imp-less-Suc nth-map push-push-simp that*)

have *indices-unfold*: (∀ *k* < *length indices*. *P* (*indices*!*k*)) ⟷ (∀ *l*>*0*. *l*<*n* ⟶ *P l*) **for** *P*
unfolding *indices-def* **apply** *auto*
using ‹*n*>*0*› **by** (*metis Suc-diff-Suc diff-zero not-less-eq*)

have *alternative-sum-rsub*:
(∑*R*− *p l* (λ*ka*. *list-eval s a ka* && *peval* (*z* ! *l*) *a*))
=(∑*R*− *p l* (λ*k*. *map* (λ*P*. *peval P a*) *s* ! *k* && *map* (λ*P*. *peval P a*) *z* ! *l*)) **for** *l*
apply (*rule sum-rsub-cong*) **unfolding** *list-eval-def* **apply** *simp*
using *modifies-yields-valid-register*
One-nat-def assms(*3*) *nth-map* ‹*length z* = *n*› ‹*length s* = *Suc m*›
by (*metis* ‹*length p* = *length s*› *le-imp-less-Suc m-def*)

have (*eval DS a*) = (∃ *ks*. *n* = *length ks* ∧
(∀ *k*<*length indices*. *eval* [*Param* (*indices* ! *k*)
= ∑*R*− (*indices* ! *k*) (*s*′ && *z*′ ! (*indices* ! *k*))] (*push-list a ks*)) ∧
(∀ *k*<*length indices*. *eval* (*single-register* (*indices* ! *k*)) (*push-list a ks*)))
unfolding *DS-def params-are-sum-rsub-of-bitand-def param-l-is-sum-rsub-of-bitand-def*
by (*simp add*: *defs*)

also have ... = (∃ *ks*. *n* = *length ks* ∧
(∀ *k*<*length indices*.
peval (*Param* (*indices* ! *k*)) (*push-list a ks*)
= ∑*R*− *p* (*indices* ! *k*) (λ*ka*. *peval* (*s*′ ! *ka*) (*push-list a ks*)
&& *peval* (*z*′ ! (*indices* ! *k*)) (*push-list a ks*)) ∧
peval (*r*′ ! (*indices* ! *k*)) (*push-list a ks*)
= *peval b*′ (*push-list a ks*) ∗ *peval* (*r*′ ! (*indices* ! *k*)) (*push-list a ks*)
+ *peval b*′ (*push-list a ks*) ∗ ∑*R*+ *p* (*indices* ! *k*)
(λ*x*. *peval* (*s*′ ! *x*) (*push-list a ks*))
− *peval b*′ (*push-list a ks*) ∗ (*push-list a ks* (*indices* ! *k*))))
using *eval-to-peval* **unfolding** *single-register-def*
using *sum-radd-polynomial-eval* ‹*length p* > *0*› **by** (*simp add*: *defs*) (*blast*)

also have ... = (∃ *ks*. *n* = *length ks* ∧
(∀ *k*<*length indices*.
ks!(*indices*!*k*)
= ∑*R*− *p* (*indices* ! *k*) (λ*ka*. *peval* (*s*′ ! *ka*) (*push-list a ks*)

&& *peval* (z' ! (*indices* ! *k*)) (*push-list a ks*)) ∧
peval (*r*!(*indices*!*k*)) *a*
= *peval b a* ∗ *peval* (*r*!(*indices*!*k*)) *a*
+ *peval b a* ∗ $\sum$ *R+ p* (*indices* ! *k*) (λ*x*. *peval* (s' ! *x*) (*push-list a ks*))
− *peval b a* ∗ (*ks*!(*indices*!*k*))))
using b'*-unfold* r'*-unfold Param-unfold unfold-4* **by** (*smt* (*z3*))

also have ... = (∃ *ks*. *n* = *length ks* ∧
(∀ *k*<*length indices*.
ks!(*indices*!*k*)
= ($\sum$ *R− p* (*indices* ! *k*) (λ*ka*. *peval* (s' ! *ka*) (*push-list a ks*)
&& *peval* (z' ! (*indices* ! *k*)) (*push-list a ks*))) ∧
peval (*r*!(*indices*!*k*)) *a*
= *peval b a* ∗ *peval* (*r*!(*indices*!*k*)) *a*
+ *peval b a* ∗ ($\sum$ *R+ p* (*indices* ! *k*) (*list-eval s a*))
− *peval b a* ∗ (*ks*!(*indices*!*k*))))
using *unfold-sum-radd* **by** (*smt* (*z3*))

also have ... = (∃ *ks*. *n* = *length ks* ∧
(∀ *k*<*length indices*.
ks!(*indices*!*k*)
= $\sum$ *R− p* (*indices* ! *k*) (λ*ka*. *list-eval s a ka* && *peval* (*z* ! (*indices* !
k)) *a*)
∧ *peval* (*r*!(*indices*!*k*)) *a*
= *peval b a* ∗ *peval* (*r*!(*indices*!*k*)) *a*
+ *peval b a* ∗ ($\sum$ *R+ p* (*indices* ! *k*) (*list-eval s a*))
− *peval b a* ∗ (*ks*!(*indices*!*k*))))
using *unfold-sum-rsub* **by** *auto*

also have ... = (∃ *ks*. *n* = *length ks* ∧
(∀ *k*<*length indices*.
ks!(*indices*!*k*)
= $\sum$ *R− p* (*indices* ! *k*) (λ*ka*. *list-eval s a ka* && *peval* (*z* ! (*indices* !
k)) *a*)
∧ *peval* (*r*!(*indices*!*k*)) *a*
= *peval b a* ∗ *peval* (*r*!(*indices*!*k*)) *a*
+ *peval b a* ∗ ($\sum$ *R+ p* (*indices* ! *k*) (*list-eval s a*))
− *peval b a* ∗
($\sum$ *R− p* (*indices* ! *k*) (λ*ka*. *list-eval s a ka* && *peval* (*z* ! (*indices* !
k)) *a*))))
by *smt*

also have ... = (∀ *k*<*length indices*.
peval (*r*!(*indices*!*k*)) *a*
= *peval b a* ∗ *peval* (*r*!(*indices*!*k*)) *a*
+ *peval b a* ∗ ($\sum$ *R+ p* (*indices* ! *k*) (*list-eval s a*))
− *peval b a* ∗
($\sum$ *R− p* (*indices* ! *k*) (λ*ka*. *list-eval s a ka* && *peval* (*z* ! (*indices* !
k)) *a*)))

unfolding *indices-def* **apply** *auto*
apply (*rule exI*[*of* -
map (λ*k*. ∑ *R*− *p k* (λ*ka*. *list-eval s a ka* && *peval* (*z* ! *k*) *a*)) [*0*..<*n*]])
by *auto*

also have ... = (∀ *l*>*0*. *l* < *n* ⟶
peval (*r*!*l*) *a*
= *peval b a* ∗ *peval* (*r*!*l*) *a*
+ *peval b a* ∗ (∑ *R*+ *p l* (*list-eval s a*))
− *peval b a* ∗
(∑ *R*− *p l* (λ*ka*. *list-eval s a ka* && *peval* (*z* ! *l*) *a*)))
using *indices-unfold*[*of* λ*x*. *peval* (*r* ! *x*) *a* =
peval b a ∗ *peval* (*r* ! *x*) *a* + *peval b a* ∗ (∑ *R*+ *p x* (*list-eval s a*)) −
peval b a ∗ (∑ *R*− *p x* (λ*ka*. (*list-eval s a ka*) && *peval* (*z* ! *x*) *a*))]
by *auto*

also have ... = (∀ *l*>*0*. *l* < *n* ⟶
peval (*r*!*l*) *a* =
peval b a ∗ *map* (λ*P*. *peval P a*) *r* ! *l*
+ *peval b a* ∗ (∑ *R*+ *p l* ((!) (*map* (λ*P*. *peval P a*) *s*)))
− *peval b a* ∗ (∑ *R*− *p l* (λ*k*. *map* (λ*P*. *peval P a*) *s* ! *k* && *map* (λ*P*. *peval P a*) *z* ! *l*)))
using *nth-map*[*of* - *r* (λ*P*. *peval P a*)] **unfolding** ‹*length r* = *n*›
using *alternative-sum-rsub list-eval-def* **by** *auto*

also have ... = (*eval DR a*)
apply (*simp add*: *DR-def defs*) **using** *rm-eq-fixes-def rm-eq-fixes.register-l-def local.register-machine-axioms*
using *nth-map*[*of* - *r* λ*P*. *peval P a*] **unfolding** ‹*length r* = *n*› **by** *auto*

finally show *eval DS a* = *eval DR a* **by** *auto*
qed

moreover have *is-dioph-rel DS*
proof −
have *list-all* (*is-dioph-rel* ∘ *param-l-is-sum-rsub-of-bitand*) *indices*
unfolding *param-l-is-sum-rsub-of-bitand-def indices-def list-all-def* **by** (*auto simp*:*dioph*)
hence *is-dioph-rel params-are-sum-rsub-of-bitand*
unfolding *params-are-sum-rsub-of-bitand-def* **by** (*auto simp*: *dioph*)

have *list-all* (*is-dioph-rel* ∘ *single-register*) *indices*
unfolding *single-register-def list-all-def indices-def* **by** (*auto simp*: *dioph*)
thus *?thesis*
unfolding *DS-def* **using** ‹*is-dioph-rel params-are-sum-rsub-of-bitand*› **by** (*auto simp*: *dioph*)
qed

ultimately show *?thesis*

by (*auto simp*: *is-dioph-rel-def*)
qed

lemma *register-bound-dioph*:
fixes *b q* :: *polynomial*
fixes *r* :: *polynomial list*
assumes *length r* = *n*
defines *DR* ≡ *LARY* (*λll. rm-eq-fixes.register-bound n* (*ll*!*0*!*0*) (*ll*!*0*!*1*) (*nth* (*ll*!*1*)))
[[*b*, *q*], *r*]
shows *is-dioph-rel DR*
proof −

define *indices* **where** *indices* ≡ [*0*..<*n*]
hence *length indices* = *n* **by** *auto*

let *?N* = *length indices*
define *b′ q′ r′* **where** *pushed-def*: *b′* = *push-param b ?N*
q′ = *push-param q ?N*
r′ = *map* (*λx. push-param x ?N*) *r*

define *bound* **where**
bound ≡ *λl.* (*r′*!*l* [<] (*Param l*) [∧] [*Param l* = *b′* ^ *q′*])

define *DS* **where** *DS* ≡ [∃ *n*] [∀ *in indices*] *bound*

have *eval DS a* = *eval DR a* **for** *a*
proof −

have *r′-unfold*: *peval* (*r′* ! *k*) (*push-list a ks*) = *peval* (*r* ! *k*) *a*
if *length ks* = *n* **and** *k* < *length ks* **for** *k ks*
unfolding *pushed-def* ‹*length indices* = *n*›
using *push-push-map-i*[*of ks n k r*] *that* ‹*length r* = *n*› *list-eval-def* **by** *auto*

have *b′-unfold*: *peval b′* (*push-list a ks*) = *peval b a*
and *q′-unfold*: *peval q′* (*push-list a ks*) = *peval q a*
if *length ks* = *n* **and** *k* < *length ks* **for** *k ks*
unfolding *pushed-def* ‹*length indices* = *n*›
using *push-push-simp that* ‹*length r* = *n*› *list-eval-def* **by** *auto*

have *eval DS a* = (∃ *ks. n* = *length ks* ∧
(∀ *k*<*n. peval* (*r′* ! *k*) (*push-list a ks*) < *push-list a ks k* ∧
push-list a ks k = *peval b′* (*push-list a ks*) ^ *peval q′* (*push-list a ks*)))
unfolding *DS-def indices-def bound-def* **by** (*simp add*: *defs*)

also have ... = (∃ *ks. n* = *length ks* ∧
(∀ *k*<*n. peval* (*r* ! *k*) *a* < *peval b a* ^ *peval q a* ∧
push-list a ks k = *peval b a* ^ *peval q a*))
using *r′-unfold b′-unfold q′-unfold* **by** (*metis* (*full-types*))

also have ... = (∀ *k*<*n*. *peval* (*r* ! *k*) *a* < *peval b a* ^ *peval q a*)
apply *auto* **apply** (*rule exI*[*of* - *map* (λ*k*. *peval b a* ^ *peval q a*) [*0*..<*n*]])
unfolding *indices-def push-list-def* **by** *auto*

also have ... = (∀ *l*<*n*. *map* (λ*P*. *peval P a*) *r* ! *l* < *peval b a* ^ *peval q a*)
using *nth-map*[*of* - *r* λ*P*. *peval P a*] ‹*length r* = *n*› **by** *force*

finally show *?thesis* **unfolding** *DR-def*
using *rm-eq-fixes.register-bound-def rm-eq-fixes-def register-machine-def*
p-nonempty n-gt-0 valid-program **by** (*auto simp add*: *defs*)

qed

moreover have *is-dioph-rel DS*
proof −
have *list-all* (*is-dioph-rel* ∘ *bound*) *indices*
unfolding *bound-def indices-def list-all-def* **by** (*auto simp*:*dioph*)
thus *?thesis* **unfolding** *DS-def indices-def bound-def* **by** (*auto simp*: *dioph*)
qed

ultimately show *?thesis*
by (*auto simp*: *is-dioph-rel-def*)
qed

definition *register-equations-relation* :: *polynomial* ⇒ *polynomial* ⇒ *polynomial*
⇒ *polynomial list* ⇒ *polynomial list* ⇒ *polynomial list* ⇒ *relation* (‹[*REG*] - - -
- - -›) **where**
[*REG*] *a b q r z s* ≡ *LARY* (λ*ll*. *rm-eq-fixes.register-equations p n* (*ll*!*0*!*0*) (*ll*!*0*!*1*)
(*ll*!*0*!*2*)
(*nth* (*ll*!*1*)) (*nth* (*ll*!*2*)) (*nth* (*ll*!*3*))) [[*a*, *b*, *q*], *r*, *z*, *s*]

lemma *reg-dioph*:
fixes *A b q r z s*
assumes *length r* = *n length z* = *n length s* = *Suc m*
defines *DR* ≡ [*REG*] *A b q r z s*
shows *is-dioph-rel DR*
proof −

define *DS* **where** *DS* ≡ (*LARY* (λ*ll*. *rm-eq-fixes.register-0 p* (*ll*!*0*!*0*) (*ll*!*0*!*1*)
(*nth* (*ll*!*1*)) (*nth* (*ll*!*2*)) (*nth* (*ll*!*3*))) [[*A*, *b*], *r*, *z*, *s*])
[∧] (*LARY* (λ*ll*. *rm-eq-fixes.register-l p n* (*ll*!*0*!*0*)
(*nth* (*ll*!*1*)) (*nth* (*ll*!*2*)) (*nth* (*ll*!*3*))) [[*b*], *r*, *z*, *s*])
[∧] (*LARY* (λ*ll*. *rm-eq-fixes.register-bound n* (*ll*!*0*!*0*) (*ll*!*0*!*1*) (*nth*
(*ll*!*1*)))
[[*b*, *q*], *r*])

have *eval DS a* = *eval DR a* **for** *a*
unfolding *DS-def DR-def register-equations-relation-def rm-eq-fixes.register-equations-def*

apply (*simp add*: *defs*)
by (*simp add*: *register-machine-axioms rm-eq-fixes.intro rm-eq-fixes.register-equations-def*)

moreover have *is-dioph-rel DS*
unfolding *DS-def* **using** *assms register-0-dioph*[*of r z s*] *register-l-dioph*[*of r z s*]
register-bound-dioph **by** (*auto simp*: *dioph*)

ultimately show *?thesis* **by** (*auto simp*: *is-dioph-rel-def*)
qed

end

end

4.4.3 State 0 equation

theory *State-0-Equation* **imports** *../Register-Machine/MultipleStepState*
RM-Sums-Diophantine ../Diophantine/Binary-And

begin

context *rm-eq-fixes*
begin

Equation 4.24

definition *state-0* :: *bool* **where**
state-0 ≡ *s 0* = *1* + *b*∗∑ *S*+ *p 0 s* + *b*∗∑ *S*− *p 0* (λ*k*. *s k* && *z* (*modifies* (*p*!*k*)))
+ *b*∗∑ *S0 p 0* (λ*k*. *s k* && (*e* − *z* (*modifies* (*p*!*k*))))

end

context *register-machine*
begin

no-notation *ppolynomial.Sum* (**infixl** ‹+› *65*)
no-notation *ppolynomial.NatDiff* (**infixl** ‹−› *65*)
no-notation *ppolynomial.Prod* (**infixl** ‹∗› *70*)

lemma *state-0-dioph*:
fixes *b e* :: *polynomial*
fixes *z s* :: *polynomial list*
assumes *length z* = *n length s* = *Suc m*

defines *DR* ≡ *LARY* (λ*ll*. *rm-eq-fixes.state-0 p* (*ll*!*0*!*0*) (*ll*!*0*!*1*)
(*nth* (*ll*!*1*)) (*nth* (*ll*!*2*))) [[*b*, *e*], *z*, *s*]
shows *is-dioph-rel DR*
proof −
let *?N* = *2*
define *b*′ *e*′ *z*′ *s*′ **where** *pushed-def*: *b*′ = *push-param b ?N*
e′ = *push-param e ?N*
z′ = *map* (λ*x*. *push-param x ?N*) *z*
s′ = *map* (λ*x*. *push-param x ?N*) *s*

define *z0 z1* **where** *z-def*: *z0* ≡ *map* (λ*i*. *z*′ ! *modifies* (*p*!*i*)) [*0*..<*length p*]
z1 ≡ *map* (λ*i*. *e*′ [−] *z*′ ! *modifies* (*p*!*i*)) [*0*..<*length p*]

define *param-0-is-sum-sub-nzero-term* **where**
param-0-is-sum-sub-nzero-term ≡ [*Param 0* = ∑ *S*− *0* (*s*′ && *z0*)]

define *param-1-is-sum-sub-zero-term* **where**
param-1-is-sum-sub-zero-term ≡ [*Param 1* = ∑ *S0 0* (*s*′ && *z1*)]

define *step-relation* **where**
step-relation ≡ (*s*′!*0* [=] **1** [+] *b*′ [∗] ([∑ *S*+] *p 0* (*nth s*′))
[+] *b*′ [∗] *Param 0* [+] *b*′ [∗] *Param 1*)

define *DS* **where** *DS* ≡ [∃ *?N*] *step-relation*
[∧] *param-0-is-sum-sub-nzero-term*
[∧] *param-1-is-sum-sub-zero-term*

have *p* ≠ [] **using** *p-nonempty* **by** *auto*
have *ps-lengths*: *length p* = *length s*
using ‹*length s* = *Suc m*› *m-def* ‹*p* ≠ []› **by** *auto*
have *s-len*: *length s* > *0*
using *ps-lengths* ‹*p* ≠ []› **by** *auto*
have *p-len*: *length p* > *0*
using *ps-lengths s-len* **by** *auto*
have *p-len2*: *length p* = *Suc m*
using *ps-lengths* ‹*length s* = *Suc m*› **by** *auto*
have *len-s*′: *length s*′ = *Suc m*
unfolding *pushed-def* **using** ‹*length s* = *Suc m*› **by** *auto*
have *length z0* = *Suc m*
unfolding *z-def ps-lengths* ‹*length s* = *Suc m*› **by** *simp*
have *length z1* = *Suc m*
unfolding *z-def ps-lengths* ‹*length s* = *Suc m*› **by** *simp*

have *modifies-le-n*: *k* < *length p* ⟹ *modifies* (*p*!*k*) < *n* **for** *k*
using *modifies-yields-valid-register* ‹*length z* = *n*› **by** *auto*

have *eval DS a* = *eval DR a* **for** *a*
proof −

have *b′-unfold*: *peval b′* (*push-list a ks*) = *peval b a* **if** *length ks* = *2* **for** *ks*
using *push-push-simp* **unfolding** *pushed-def* **using** *that* **by** *metis*

have *s′-0-unfold*: *peval* (*s′* ! *0*) (*push-list a ks*) = *peval* (*s* ! *0*) *a* **if** *length ks* = *2* **for** *ks*
unfolding *pushed-def* **using** *push-push-map-i*[*of ks 2 0 s a*] *that* **unfolding** *list-eval-def*
‹*length s* > *0*› **using** *s-len* **by** *auto*

have *sum-nzero-unfold*:
eval [*polynomial.Param 0* = $\sum$ *S*− *0* (*s′* && *z0*)] (*push-list a ks*)
= (*peval* (*polynomial.Param 0*) (*push-list a ks*)
= $\sum$ *S*− *p 0* (λ*k*. *peval* (*s′* ! *k*) (*push-list a ks*) && *peval* (*z0* ! *k*) (*push-list a ks*))) **for** *ks*
using *sum-rsub-nzero-of-bit-and-eval*[*of s′ z0 Param 0 0 push-list a ks*]
‹*length p* > *0*› ‹*length s′* = *Suc m*› ‹*length z0* = *Suc m*› **by** *auto*

have *sum-zero-unfold*:
eval [*polynomial.Param 1* = $\sum$ *S0 0* (*s′* && *z1*)] (*push-list a ks*)
= (*peval* (*polynomial.Param 1*) (*push-list a ks*)
= $\sum$ *S0 p 0* (λ*k*. *peval* (*s′* ! *k*) (*push-list a ks*) && *peval* (*z1* ! *k*) (*push-list a ks*))) **for** *ks*
using *sum-rsub-zero-of-bit-and-eval*[*of s′ z1 Param 1 0 push-list a ks*]
‹*length p* > *0*› ‹*length s′* = *Suc m*› ‹*length z1* = *Suc m*› **by** *auto*

have *param-0-unfold*: *peval* (*Param 0*) (*push-list a ks*) = *ks* ! *0* **if** *length ks* = *2* **for** *ks*
unfolding *push-list-def* **using** *that* **by** *auto*

have *param-1-unfold*: *peval* (*Param 1*) (*push-list a ks*) = *ks* ! *1* **if** *length ks* = *2* **for** *ks*
unfolding *push-list-def* **using** *that* **by** *auto*

have *sum-sadd-unfold*:
peval ([$\sum$ *S*+] *p 0* ((!) *s′*)) (*push-list a ks*) = $\sum$ *S*+ *p 0* (λ*x*. *peval* (*s* ! *x*) *a*)
if *length ks* = *2* **for** *ks*
using *sum-sadd-polynomial-eval* ‹*length p* > *0*› **apply** *auto*
apply (*rule sum-sadd-cong*, *auto*)
unfolding *pushed-def* **using** *push-push-map-i*[*of ks 2 - s a*] *that*
unfolding ‹*length p* = *length s*› *list-eval-def*
by (*smt One-nat-def assms le-imp-less-Suc m-def nth-map p-len2*)

have *z0-unfold*:
peval (*s′* ! *k*) (*push-list a ks*) && *peval* (*z0* ! *k*) (*push-list a ks*)
= *peval* (*s* ! *k*) *a* && *peval* (*z* ! *modifies* (*p* ! *k*)) *a*
if *length ks* = *2* **and** *k* < *length p* **for** *k ks*
proof −
have *map*: *map* (λ*i*. *z′* ! *modifies* (*p* ! *i*)) [*0*..<*length p*] ! *k*
= *z′* ! *modifies* (*p* ! *k*)

unfolding *z-def* **using** *nth-map*[*of k* [*0..<length p*] λ*i. z′* ! *modifies* (*p* ! *i*)]
using ‹*k* < *length p*› **by** *auto*

have *s*: *peval* (*map* (λ*x. push-param x 2*) *s* ! *k*) (*push-list a ks*) = *peval* (*s* ! *k*) *a*
using *push-push-map-i*[*of ks 2 k s*] *that nth-map*[*of k s*]
unfolding ‹*length s* = *Suc m*› ‹*length p* = *Suc m*› *list-eval-def* **by** *auto*

have *z*: *peval* (*map* (λ*x. push-param x 2*) *z* ! *modifies* (*p* ! *k*)) (*push-list a ks*)
= *peval* (*z* ! *modifies* (*p* ! *k*)) *a*
using *push-push-map-i*[*of ks 2 modifies* (*p*!*k*) *z a*] *modifies-le-n*[*of k*] *that nth-map*[*of - z*]
unfolding ‹*length z* = *n*› *list-eval-def* **by** *auto*

show *?thesis*
unfolding *z-def map* **unfolding** *pushed-def s z* **by** *auto*
qed

have *z1-unfold*:
peval (*s′* ! *k*) (*push-list a ks*) && *peval* (*z1* ! *k*) (*push-list a ks*)
= *peval* (*s* ! *k*) *a* && (*peval e a* − *peval* (*z* ! *modifies* (*p* ! *k*)) *a*)
if *length ks* = *2* **and** *k* < *length p* **for** *k ks*
proof −
have *map*:
map (λ*i. e′* [−] (*z′* ! (*modifies* (*p* ! *i*)))) [*0..<length p*] ! *k*
= *e′* [−] (*z′* ! *modifies* (*p* ! *k*))
using *nth-map*[*of k* [*0..<length p*] λ*i. z′* ! *modifies* (*p* ! *i*)]
using ‹*k* < *length p*› **by** *auto*

have *s*: *peval* (*map* (λ*x. push-param x 2*) *s* ! *k*) (*push-list a ks*) = *peval* (*s* ! *k*) *a*
using *push-push-map-i*[*of ks 2 k s*] *that nth-map*[*of k s*]
unfolding ‹*length s* = *Suc m*› ‹*length p* = *Suc m*› *list-eval-def* **by** *auto*

have *z*: *peval* (*push-param e 2*) (*push-list a ks*)
− *peval* (*map* (λ*x. push-param x 2*) *z* ! *modifies* (*p* ! *k*)) (*push-list a ks*)
= *peval e a* − *peval* (*z* ! (*modifies* (*p*!*k*))) *a*
using *push-push-simp*[*of e ks a*] **unfolding** ‹*length ks* = *2*› **apply** *simp*
using *push-push-map-i*[*of ks 2 modifies* (*p*!*k*) *z a*] *modifies-le-n*[*of k*] *that*
nth-map[*of modifies* (*p*!*k*) *z* (λ*x. peval x a*)]
unfolding ‹*length z* = *n*› *list-eval-def* **by** *auto*

show *?thesis*
unfolding *z-def map* **unfolding** *pushed-def s* **using** *z* **by** *auto*
qed

have *z0sum-unfold*:
(∑*S*− *p 0* (λ*k. peval* (*s′* ! *k*) (*push-list a ks*) && *peval* (*z0* ! *k*) (*push-list a ks*)))

=(∑ *S− p 0* (λ*k*. *peval* (*s* ! *k*) *a* && *peval* (*z* ! *modifies* (*p* ! *k*)) *a*))
if *length ks* = *2* **for** *ks*
apply (*rule sum-ssub-nzero-cong*) **using** *z0-unfold*[*of ks*] *that*
by (*metis* ‹*length s* = *Suc m*› *le-imp-less-Suc m-def ps-lengths*)

have *z1sum-unfold*:
(∑ *S0 p 0* (λ*k*. *peval* (*s*′ ! *k*) (*push-list a ks*) && *peval* (*z1* ! *k*) (*push-list a ks*)))
=(∑ *S0 p 0* (λ*k*. *peval* (*s* ! *k*) *a* && *peval e a* − *peval* (*z* ! *modifies* (*p* ! *k*)) *a*))
if *length ks* = *2* **for** *ks*
apply (*rule sum-ssub-zero-cong*) **using** *z1-unfold*[*of ks*] *that*
by (*metis* ‹*length s* = *Suc m*› *le-imp-less-Suc m-def ps-lengths*)

have *sum-sadd-map*: ∑ *S+ p 0* ((!) (*map* (λ*P*. *peval P a*) *s*)) = ∑ *S+ p 0* (λ*x*. *peval* (*s* ! *x*) *a*)
apply (*rule sum-sadd-cong*, *auto*)
using *nth-map*[*of - s* (λ*P*. *peval P a*)] *m-def* ‹*length s* = *Suc m*› **by** *auto*

have *sum-ssub-nzero-map*:
(∑ *S− p 0* (λ*k*. *peval* (*s* ! *k*) *a* && *peval* (*z* ! *modifies* (*p* ! *k*)) *a*))
= (∑ *S− p 0* (λ*k*. *map* (λ*P*. *peval P a*) *s* ! *k* && *map* (λ*P*. *peval P a*) *z* ! *modifies* (*p* ! *k*)))
proof −
have *1*: *peval* (*s* ! *k*) *a* && *peval* (*z* ! *modifies* (*p* ! *k*)) *a* =
map (λ*P*. *peval P a*) *s* ! *k* && *map* (λ*P*. *peval P a*) *z* ! *modifies* (*p* ! *k*)
if *k* < *length p* **for** *k*
proof −
have *peval* (*s* ! *k*) *a* = *map* (λ*P*. *peval P a*) *s* ! *k*
using *nth-map that ps-lengths* **by** *auto*
moreover have *peval* (*z* ! *modifies* (*p* ! *k*)) *a*
= *map* (λ*P*. *peval P a*) *z* ! *modifies* (*p* ! *k*)
using *nth-map*[*of modifies* (*p*!*k*) *z* (λ*P*. *peval P a*)] *modifies-le-n*[*of k*] *that*
using ‹*length z* = *n*› **by** *auto*
ultimately show *?thesis* **by** *auto*
qed
show *?thesis* **apply** (*rule sum-ssub-nzero-cong*, *auto*)
using *1* **by** (*metis Suc-le-mono Suc-pred less-eq-Suc-le p-len*)
qed

have *sum-ssub-zero-map*:
(∑ *S0 p 0* (λ*k*. *peval* (*s* ! *k*) *a* && *peval e a* − *peval* (*z* ! *modifies* (*p* ! *k*)) *a*))
= (∑ *S0 p 0* (λ*k*. *map* (λ*P*. *peval P a*) *s* ! *k* && *peval e a*
− *map* (λ*P*. *peval P a*) *z* ! *modifies* (*p* ! *k*)))
proof −
have *1*: *peval* (*s* ! *k*) *a* && *peval e a* − *peval* (*z* ! *modifies* (*p* ! *k*)) *a* =
map (λ*P*. *peval P a*) *s* ! *k* && *peval e a* − *map* (λ*P*. *peval P a*) *z* ! *modifies* (*p* ! *k*)

```
      if k < length p for k
    proof -
      have peval (s ! k) a = map (λP. peval P a) s ! k
      using nth-map that ps-lengths by auto
      moreover have peval (z ! modifies (p ! k)) a
                   = map (λP. peval P a) z ! modifies (p ! k)
        using nth-map[of modifies (p!k) z (λP. peval P a)] modifies-le-n[of k] that
        using ‹length z = n› by auto
      ultimately show ?thesis by auto
    qed
    show ?thesis apply (rule sum-ssub-zero-cong, auto)
      using 1 by (metis Suc-le-mono Suc-pred less-eq-Suc-le p-len)
  qed


  have eval DS a =
          (∃ks. length ks = 2 ∧
              eval (s' ! 0 [=] 1 [+] b' [*] ([∑S+] p 0 (!) s') [+] b' [*] Param 0
                          [+] b' [*] Param (Suc 0)) (push-list a ks)
            ∧ eval [polynomial.Param 0 = ∑S− 0 (s' && z0)] (push-list a ks)
            ∧ eval [polynomial.Param 1 = ∑S0 0 (s' && z1)] (push-list a ks))
    unfolding DS-def step-relation-def param-0-is-sum-sub-nzero-term-def
      param-1-is-sum-sub-zero-term-def by (simp add: defs)

  also have ... = (∃ks. length ks = 2 ∧
        peval (s' ! 0) (push-list a ks) =
        Suc (peval b' (push-list a ks) ∗ peval ([∑S+] p 0 ((!) s')) (push-list a ks) +
             peval b' (push-list a ks) ∗ push-list a ks 0 +
             peval b' (push-list a ks) ∗ push-list a ks (Suc 0))
        ∧ (peval (Param 0) (push-list a ks)
             = ∑S− p 0 (λk. peval (s' ! k) (push-list a ks) && peval (z0 ! k)
(push-list a ks)))
        ∧ (peval (Param 1) (push-list a ks)
             = ∑S0 p 0 (λk. peval (s' ! k) (push-list a ks) && peval (z1 ! k)
(push-list a ks))))
    unfolding sum-nzero-unfold sum-zero-unfold by (simp add: defs )

  also have ... = (∃ks. length ks = 2 ∧
        peval (s ! 0) a =
        Suc (peval b a ∗ peval ([∑S+] p 0 ((!) s')) (push-list a ks) +
             peval b a ∗ push-list a ks 0 +
             peval b a ∗ push-list a ks (Suc 0))
        ∧ (ks!0
             = ∑S− p 0 (λk. peval (s' ! k) (push-list a ks) && peval (z0 ! k)
(push-list a ks)))
        ∧ (ks!1
             = ∑S0 p 0 (λk. peval (s' ! k) (push-list a ks) && peval (z1 ! k)
(push-list a ks))))
    using b'-unfold s'-0-unfold param-0-unfold param-1-unfold by auto
```

also have ... = (∃ *ks. length ks* = *2* ∧
peval (*s* ! *0*) *a* =
Suc (*peval b a* ∗ $\sum$ *S+ p 0* (λ*x. peval* (*s* ! *x*) *a*) +
peval b a ∗ (*ks*!*0*) + *peval b a* ∗ (*ks*!*1*))
∧ (*ks*!*0* = $\sum$ *S− p 0* (λ*k. peval* (*s′* ! *k*) (*push-list a ks*) && *peval* (*z0* ! *k*) (*push-list a ks*)))
∧ (*ks*!*1* = $\sum$ *S0 p 0* (λ*k. peval* (*s′* ! *k*) (*push-list a ks*) && *peval* (*z1* ! *k*) (*push-list a ks*))))
using *push-list-def sum-sadd-unfold* **by** *auto*

also have ... = (∃ *ks. length ks* = *2* ∧
peval (*s* ! *0*) *a* = *Suc* (*peval b a* ∗ $\sum$ *S+ p 0* (λ*x. peval* (*s* ! *x*) *a*)
+ *peval b a* ∗ (*ks*!*0*) + *peval b a* ∗ (*ks*!*1*))
∧ (*ks*!*0* = $\sum$ *S− p 0* (λ*k. peval* (*s* ! *k*) *a* && *peval* (*z* ! *modifies* (*p* ! *k*)) *a*))
∧ (*ks*!*1* = $\sum$ *S0 p 0* (λ*k. peval* (*s* ! *k*) *a* && *peval e a* − *peval* (*z* ! *modifies* (*p* ! *k*)) *a*)))
using *z0sum-unfold z1sum-unfold* **by** *auto*

also have ... = (∃ *ks. length ks* = *2* ∧
peval (*s* ! *0*) *a*
= *Suc* (*peval b a* ∗ $\sum$ *S+ p 0* (λ*x. peval* (*s* ! *x*) *a*)
+ *peval b a* ∗ $\sum$ *S− p 0* (λ*k. peval* (*s* ! *k*) *a* && *peval* (*z* ! *modifies* (*p* ! *k*)) *a*)
+ *peval b a* ∗ $\sum$ *S0 p 0* (λ*k. peval* (*s* ! *k*) *a* && *peval e a* − *peval* (*z* ! *modifies* (*p* ! *k*)) *a*))
∧ (*ks*!*0* = $\sum$ *S− p 0* (λ*k. peval* (*s* ! *k*) *a* && *peval* (*z* ! *modifies* (*p* ! *k*)) *a*))
∧ (*ks*!*1* = $\sum$ *S0 p 0* (λ*k. peval* (*s* ! *k*) *a* && *peval e a* − *peval* (*z* ! *modifies* (*p* ! *k*)) *a*)))
by *auto*

also have ... = (*peval* (*s* ! *0*) *a*
= *Suc* (*peval b a* ∗ $\sum$ *S+ p 0* (λ*x. peval* (*s* ! *x*) *a*)
+ *peval b a* ∗ $\sum$ *S− p 0* (λ*k. peval* (*s* ! *k*) *a* && *peval* (*z* ! *modifies* (*p* ! *k*)) *a*)
+ *peval b a* ∗ $\sum$ *S0 p 0* (λ*k. peval* (*s* ! *k*) *a* && *peval e a* − *peval* (*z* ! *modifies* (*p* ! *k*)) *a*)))
apply *auto*
apply (*rule exI*[*of* - [($\sum$ *S− p 0* (λ*k. peval* (*s* ! *k*) *a* && *peval* (*z* ! *modifies* (*p* ! *k*)) *a*)),
$\sum$ *S0 p 0* (λ*k. peval* (*s* ! *k*) *a* && *peval e a* − *peval* (*z* ! *modifies* (*p* ! *k*)) *a*)]])
by *auto*

also have ... = (*map* (λ*P. peval P a*) *s* ! *0* =
Suc (*peval b a* ∗ $\sum$ *S+ p 0* ((!) (*map* (λ*P. peval P a*) *s*)) +
peval b a ∗ $\sum$ *S− p 0* (λ*k. map* (λ*P. peval P a*) *s* ! *k*
&& *map* (λ*P. peval P a*) *z* ! *modifies* (*p* ! *k*)) +
peval b a ∗
$\sum$ *S0 p 0* (λ*k. map* (λ*P. peval P a*) *s* ! *k* && *peval e a*
− *map* (λ*P. peval P a*) *z* ! *modifies* (*p* !

k))))
 using *nth-map*[*of* - - (λ*P*. *peval P a*)] ‹*length s* > *0*›
 using *sum-ssub-zero-map sum-sadd-map sum-ssub-nzero-map* **by** *auto*

finally show *?thesis* **unfolding** *DR-def* **using** *rm-eq-fixes-def local.register-machine-axioms*

rm-eq-fixes.state-0-def **by** (*simp add*: *defs*)
qed

moreover have *is-dioph-rel DS*
unfolding *DS-def param-1-is-sum-sub-zero-term-def param-0-is-sum-sub-nzero-term-def step-relation-def* **by** (*auto simp add*: *dioph*)

ultimately show *?thesis*
by (*simp add*: *is-dioph-rel-def*)
qed

end

end

4.4.4 State d equation

theory *State-d-Equation* **imports** *State-0-Equation*

begin

context *rm-eq-fixes*
begin

Equation 4.25

definition *state-d* :: *bool* **where**
state-d ≡ ∀ *d*>*0*. *d*≤*m* ⟶ *s d* = *b*∗∑ *S*+ *p d s* + *b*∗∑ *S*− *p d* (λ*k*. *s k* && *z* (*modifies* (*p*!*k*)))
+ *b*∗∑ *S0 p d* (λ*k*. *s k* && (*e* − *z* (*modifies* (*p*!*k*))))

Combining the two

definition *state-relations-from-recursion* :: *bool* **where**
state-relations-from-recursion ≡ *state-0* ∧ *state-d*

end

context *register-machine*
begin

lemma *state-d-dioph*:
fixes *b e* :: *polynomial*
fixes *z s* :: *polynomial list*

assumes *length z = n length s = Suc m*
defines *DR* ≡ *LARY* (*λll. rm-eq-fixes.state-d p* (*ll*!*0*!*0*) (*ll*!*0*!*1*)
(*nth* (*ll*!*1*)) (*nth* (*ll*!*2*)))
[[*b*, *e*], *z*, *s*]
shows *is-dioph-rel DR*
proof −

define *d-domain* **where** *d-domain* ≡ [*1*..<*Suc m*]

define *number-of-ex-vars* **where** *number-of-ex-vars* = *2* ∗ *m*

have *length d-domain* = *m*
unfolding *d-domain-def* **by** *auto*

define *b′ e′ z′ s′* **where** *pushed-def*: *b′* = *push-param b number-of-ex-vars*
e′ = *push-param e number-of-ex-vars*
z′ = *map* (*λx. push-param x number-of-ex-vars*) *z*
s′ = *map* (*λx. push-param x number-of-ex-vars*) *s*

note *e′-def* = ‹*e′* = *push-param e number-of-ex-vars*›

define *z0 z1* **where** *z-def*: *z0* ≡ *map* (*λi. z′* ! *modifies* (*p*!*i*)) [*0*..<*Suc m*]
z1 ≡ *map* (*λi. e′* [−] *z′* ! *modifies* (*p*!*i*)) [*0*..<*Suc m*]

define *sum-ssub-nzero-param-of-state* **where**
sum-ssub-nzero-param-of-state ≡ (*λd. Param* (*d* − *Suc 0*))
write *sum-ssub-nzero-param-of-state* (‹∑ *S*−′*-Param* -›)

define *sum-ssub-zero-param-of-state* **where**
sum-ssub-zero-param-of-state ≡ (*λd. Param* (*m* + *d* − *Suc 0*))
write *sum-ssub-zero-param-of-state* (‹∑ *S0*′*-Param* -›)

define *param-is-sum-ssub-nzero-term* **where**
param-is-sum-ssub-nzero-term ≡ (*λd*::*nat*. [(∑ *S*−*-Param d*) = ∑ *S*− *d* (*s′* && *z0*)])

define *param-is-sum-ssub-zero-term* **where**
param-is-sum-ssub-zero-term ≡ (*λd*. [(∑ *S0-Param d*) = ∑ *S0 d* (*s′* && *z1*)])

define *params-are-sum-terms* **where**
params-are-sum-terms ≡ [∀ *in d-domain*] (*λd. param-is-sum-ssub-nzero-term d*

[∧] *param-is-sum-ssub-zero-term d*)

define *step-relation* **where**
step-relation ≡ (*λd*. (*s′*!*d*) [=] *b′* [∗] ([∑ *S*+] *p d* (*nth s′*))
[+] *b′* [∗] (∑ *S*−*-Param d*)
[+] *b′* [∗] (∑ *S0-Param d*))

define *DS* **where** *DS* ≡ [∃ *number-of-ex-vars*] (([∀ *in d-domain*] (*λd. step-relation d*))
[∧] *params-are-sum-terms*)

have *length p* > *0*
using *p-nonempty* **by** *auto*
hence *m* ≥ *0*
unfolding *m-def* **by** *auto*
have *length s′* = *Suc m* **and** *length z0* = *Suc m* **and** *length z1* = *Suc m*
unfolding *pushed-def z-def* **using** ‹*length s* = *Suc m*› *m-def* ‹*length p* > *0*› **by** *auto*

have *eval DS a* = *eval DR a* **for** *a*
proof −

have *b′-unfold*: *peval b′* (*push-list a ks*) = *peval b a*
if *length ks* = *number-of-ex-vars* **for** *ks*
unfolding *pushed-def* **using** *push-push-simp* ‹*length d-domain* = *m*› **by** (*metis that*)

have *h0*: *k* < *m* ⟹ *d-domain* ! *k* < *Suc m* **for** *k*
unfolding *d-domain-def* **apply** *simp*
using *One-nat-def Suc-pred* ‹*0* < *length p*› *add.commute*
assms(*3*) *d-domain-def less-diff-conv m-def nth-upt upt-Suc-append*
by (*smt* ‹*length d-domain* = *m*› *less-nat-zero-code list.size*(*3*) *upt-Suc*)

have *s′-unfold*: *peval* (*s′* ! (*d-domain* ! *k*)) (*push-list a ks*)
= *peval* (*s* ! (*d-domain* ! *k*)) *a*
if *length ks* = *number-of-ex-vars* **and** *k* < *m* **for** *k ks*
proof −
from ‹*k* < *m*› **have** *d-domain* ! *k* < *length s* **unfolding** ‹*length s* = *Suc m*›
using *h0* **by** *blast*

have *suc-k*: ([*Suc 0..*<*Suc m*]) ! *k* = *Suc k*
by (*metis Suc-leI Suc-pred add-less-cancel-left diff-Suc-1 le-add-diff-inverse nth-upt*
zero-less-Suc ‹*k* < *m*›)

have *peval* (*map* (*λx. push-param x number-of-ex-vars*) *s* ! (*d-domain* ! *k*)) (*push-list a ks*)
= *list-eval s a* (*d-domain* ! *k*)
using *push-push-map-i*[*of ks number-of-ex-vars d-domain*!*k s a*]
using ‹*length ks* = *number-of-ex-vars*› ‹*k* < *m*› *h0* ‹*length s* = *Suc m*› **by** *auto*
also have ... = *peval* (*s* ! (*d-domain* ! *k*)) *a*
unfolding *list-eval-def*
using *nth-map* [*of d-domain* ! *k s* (*λx. peval x a*)] ‹*d-domain* ! *k* < *length s*›
unfolding *d-domain-def* **using** ‹*m* ≥ *0*› ‹*k* < *m*› *suc-k* **by** *auto*

finally show *?thesis* **unfolding** *pushed-def* **by** *auto*
qed

have *sum-sadd-unfold*: ($\sum S+$ *p* (*d-domain* ! *k*) (λx. *peval* (*s'* ! *x*) (*push-list a ks*)))
= ($\sum S+$ *p* (*d-domain* ! *k*) ((!) (*map* (λP. *peval P a*) *s*)))
if *length ks* = *number-of-ex-vars* **for** *k ks*
apply (*rule sum-sadd-cong*, *auto*) **unfolding** *pushed-def*
using *push-push-map-i*[*of ks number-of-ex-vars - s a*] ‹*length ks* = *number-of-ex-vars*›
unfolding *list-eval-def* **by** (*simp add*: ‹*length s* = *Suc m*› *m-def*)

have *s*: *peval* (*s'* ! *ka*) (*push-list a ks*) = *map* (λP. *peval P a*) *s* ! *ka*
if *ka* < *Suc m* **and** *length ks* = *number-of-ex-vars* **for** *ka ks*
unfolding *pushed-def*
using *push-push-map-i*[*of ks number-of-ex-vars ka s a*] ‹*length ks* = *number-of-ex-vars*›
using *list-eval-def* ‹*length s* = *Suc m*› ‹*ka* < *Suc m*› **by** *auto*

have *modifies-valid*: *modifies* (*p* ! *ka*) < *length z* **if** *ka* < *Suc m* **for** *ka*
using *modifies-yields-valid-register that* **unfolding** ‹*length z* = *n*› *m-def*
using *p-nonempty* **by** *auto*

have *sum-ssub-nzero-unfold*:
($\sum S-$ *p* (*d-domain* ! *k*) (λk. *peval* (*s'* ! *k*) (*push-list a ks*)
&& *peval* (*z0* ! *k*) (*push-list a ks*)))
= ($\sum S-$ *p* (*d-domain* ! *k*) (λk. *map* (λP. *peval P a*) *s* ! *k*
&& *map* (λP. *peval P a*) *z* ! *modifies* (*p* ! *k*)))
if *length ks* = *number-of-ex-vars* **for** *k ks*
proof−
have *z0*: *peval* (*z0* ! *ka*) (*push-list a ks*) = *map* (λP. *peval P a*) *z* ! *modifies* (*p* ! *ka*)
if *ka* < *Suc m* **for** *ka*
unfolding *z-def pushed-def*
using *push-push-map-i*[*of ks number-of-ex-vars modifies* (*p*!*ka*) *z a*]
‹*length ks* = *number-of-ex-vars*› **unfolding** *list-eval-def*
using ‹*length z0* = *Suc m*› ‹*ka* < *Suc m*› *modifies-valid* ‹*0* < *length p*› *m-def map-nth* **by** *force*

show *?thesis* **apply** (*rule sum-ssub-nzero-cong*) **using** *s z0 le-imp-less-Suc m-def that*
by *presburger*
qed

have *sum-ssub-zero-unfold*:
($\sum S0$ *p* (*d-domain* ! *k*) (λk. *peval* (*s'* ! *k*) (*push-list a ks*)
&& *peval* (*z1* ! *k*) (*push-list a ks*)))
= ($\sum S0$ *p* (*d-domain* ! *k*) (λk. *map* (λP. *peval P a*) *s* ! *k*
&& *peval e a* − *map* (λP. *peval P a*) *z* ! *modifies* (*p* ! *k*)))

if *length ks* = *number-of-ex-vars* **and** *k* < *Suc m* **for** *k ks*
proof−

have *map*:
map (λ*i*. *e*′ [−] (*z*′ ! (*modifies* (*p* ! *i*)))) [*0*..<*Suc m*] ! *ka*
= *e*′ [−] (*z*′ ! *modifies* (*p* ! *ka*)) **if** *ka* < *Suc m* **for** *ka*
using *nth-map*[*of ka* [*0*..<*Suc m*] λ*i*. *e*′ [−] *z*′ ! *modifies* (*p* ! *i*)] ‹*ka* < *Suc m*›
by (*smt* (*z3*) *One-nat-def Suc-pred* ‹*0* < *length p*› ‹*m* ≥ *0*› *le-trans length-map m-def map-nth*
nth-map upt-Suc-append zero-le-one)

have *peval* (*e*′ [−] (*z*′ ! *modifies* (*p* ! *ka*))) (*push-list a ks*)
= *peval e a* − *map* (λ*P*. *peval P a*) *z* ! *modifies* (*p* ! *ka*)
if *ka* < *Suc m* **for** *ka*
unfolding *z-def pushed-def* **apply** (*simp add*: *defs*)
using *push-push-simp* ‹*length ks* = *number-of-ex-vars*› **apply** *auto*
using *push-push-map-i*[*of ks number-of-ex-vars modifies* (*p*!*ka*) *z a*]
‹*length ks* = *number-of-ex-vars*› *modifies-valid* ‹*ka* < *Suc m*›
unfolding *list-eval-def* **using** ‹*length z0* = *Suc m*› ‹*0* < *length p*› *m-def map-nth* **by** *auto*

hence *z1*: *peval* (*z1* ! *ka*) (*push-list a ks*)
= *peval e a* − *map* (λ*P*. *peval P a*) *z* ! *modifies* (*p* ! *ka*) **if** *ka* < *Suc m* **for** *ka*
unfolding *z-def* **using** *map that* **by** *auto*

show *?thesis* **apply** (*rule sum-ssub-zero-cong*) **using** *s z1 le-imp-less-Suc m-def that*
by *presburger*

qed

define *sum-ssub-nzero-map* **where**
sum-ssub-nzero-map ≡ λ*j*. $\sum$ *S*− *p j* (λ*k*. *map* (λ*P*. *peval P a*) *s* ! *k*
&& *map* (λ*P*. *peval P a*) *z* ! *modifies* (*p* ! *k*))
define *sum-ssub-zero-map* **where**
sum-ssub-zero-map ≡ λ*j*. $\sum$ *S0 p j* (λ*k*. *map* (λ*P*. *peval P a*) *s* ! *k*
&& *peval e a* − *map* (λ*P*. *peval P a*) *z* ! *modifies* (*p* ! *k*))

define *ks-ex* **where**
ks-ex ≡ *map sum-ssub-nzero-map d-domain* @ *map sum-ssub-zero-map d-domain*

have *length ks-ex* = *number-of-ex-vars*
unfolding *ks-ex-def number-of-ex-vars-def* **using** ‹*length d-domain* = *m*› **by** *auto*

have *ks-ex1*:

peval ($\sum$ *S−-Param* (*d-domain* ! *k*)) (*push-list a ks-ex*)
= $\sum$ *S− p* (*d-domain* ! *k*) (λ*k*. *map* (λ*P*. *peval P a*) *s* ! *k*
&& *map* (λ*P*. *peval P a*) *z* ! *modifies* (*p* ! *k*))
if *k* < *m* **for** *k*
proof −
have *domain-at-k-bound*:
d-domain ! *k* − *Suc 0* < *length ks-ex* **using** *that* ‹*length ks-ex* = *number-of-ex-vars*›
unfolding *number-of-ex-vars-def* **using** *h0* **by** *fastforce*

have *peval* ($\sum$ *S−-Param* (*d-domain* ! *k*)) (*push-list a ks-ex*) = *ks-ex* ! *k*
unfolding *push-list-def sum-ssub-nzero-param-of-state-def* **using** *that domain-at-k-bound*
apply *auto*
using *One-nat-def Suc-mono d-domain-def diff-Suc-1 nth-upt plus-1-eq-Suc* **by** *presburger*

also have ... = $\sum$ *S− p* (*d-domain* ! *k*) (λ*k*. *map* (λ*P*. *peval P a*) *s* ! *k*
&& *map* (λ*P*. *peval P a*) *z* ! *modifies* (*p* ! *k*))
unfolding *ks-ex-def*
unfolding *nth-append*[*of map sum-ssub-nzero-map d-domain map sum-ssub-zero-map d-domain k*]
using ‹*length d-domain* = *m*› *that* **unfolding** *sum-ssub-nzero-map-def* **by** *auto*
finally show *?thesis* **by** *auto*
qed

have *ks-ex2*:
peval ($\sum$ *S0-Param* (*d-domain* ! *k*)) (*push-list a ks-ex*)
= $\sum$ *S0 p* (*d-domain* ! *k*) (λ*k*. *map* (λ*P*. *peval P a*) *s* ! *k*
&& *peval e a* − *map* (λ*P*. *peval P a*) *z* ! *modifies* (*p* ! *k*))
if *k* < *m* **for** *k*
proof −
have *domain-at-k-bound*:
m + *d-domain* ! *k* − *Suc 0* < *length ks-ex* **using** *that* ‹*length ks-ex* = *number-of-ex-vars*›
unfolding *number-of-ex-vars-def* **using** *h0* **by** *fastforce*

have *d-domain* ! *k* ≥ *1*
unfolding *d-domain-def* ‹*k* < *m*›
using *m-def p-nonempty that* **by** *auto*

hence *index-calculation*: (*m* + *d-domain* ! *k* − *Suc 0*) = *k* + *m*
unfolding *d-domain-def*
by (*metis* (*no-types*, *lifting*) *Nat.add-diff-assoc One-nat-def Suc-pred add.commute*

less-diff-conv m-def nth-upt ordered-cancel-comm-monoid-diff-class.le-imp-diff-is-add *p-nonempty that*)

have *peval* ($\sum$ *S0-Param* (*d-domain* ! *k*)) (*push-list a ks-ex*) = *ks-ex* ! (*k* + *m*)
unfolding *push-list-def sum-ssub-zero-param-of-state-def* **using** *that domain-at-k-bound*
by (*auto simp*: *index-calculation*)

also have ... = $\sum$ *S0 p* (*d-domain* ! *k*) (λ*k*. *map* (λ*P*. *peval P a*) *s* ! *k*
&& *peval e a* − *map* (λ*P*. *peval P a*) *z* ! *modifies* (*p* ! *k*))
unfolding *ks-ex-def*
unfolding *nth-append*[*of map sum-ssub-nzero-map d-domain map sum-ssub-zero-map d-domain*]
using ‹*length d-domain* = *m*› *that* **unfolding** *sum-ssub-zero-map-def* **by** *auto*
finally show *?thesis* **by** *auto*
qed

have *all-quantifier-switch*: (∀ *k*<*length d-domain*. *Property* (*d-domain* ! *k*))
= (∀ *d*>*0*. *d* ≤ *m* ⟶ *Property d*) **for** *Property*
proof (*rule*)
assume *asm*: ∀ *k*<*length d-domain*. *Property* (*d-domain* ! *k*)
show ∀ *d*>*0*. *d* ≤ *m* ⟶ *Property d*
proof (*auto*)
fix *d*
assume *d* > *0 d* ≤ *m*
hence *d* − *Suc 0* < *length d-domain*
by (*metis Suc-le-eq Suc-pred* ‹*length d-domain* = *m*›)
hence *Property* (*d-domain* ! (*d* − *Suc 0*))
using *asm* **by** *auto*
thus *Property d*
unfolding *d-domain-def*
by (*metis One-nat-def Suc-diff-1* ‹*0* < *d*› ‹*d* ≤ *m*› *le-imp-less-Suc nth-upt plus-1-eq-Suc*)
qed
next
assume *asm*: ∀ *d*>*0*. *d* ≤ *m* ⟶ *Property d*
show ∀ *k*<*length d-domain*. *Property* (*d-domain* ! *k*)
proof (*auto*)
fix *k*
assume *k* < *length d-domain*
hence *d-domain* ! *k* > *0*
unfolding *d-domain-def*
by (*smt* (*z3*) *One-nat-def Suc-leI Suc-pred* ‹*0* < *length p*› ‹*length d-domain* = *m*›
add-less-cancel-left d-domain-def diff-is-0-eq′ gr-zeroI le-add-diff-inverse

less-nat-zero-code less-numeral-extra(*1*) *m-def nth-upt*)

moreover have *d-domain* ! $k \leq m$
unfolding *d-domain-def* **using** ‹$k <$ *length d-domain*› **unfolding** ‹*length d-domain* $= m$›
using *d-domain-def h0 less-Suc-eq-le* **by** *auto*
ultimately show *Property* (*d-domain* ! k)
using *asm* **by** *auto*
qed
qed

have *peval* ($s!d$) a = *map* (λP. *peval* P a) s ! d **if** $d > 0$ **and** $d \leq m$ **for** d
using *nth-map*[*of d s* λP. *peval P a*] *that* ‹*length* s = *Suc m*› **by** *simp*

have *eval DS a* = ($\exists$ *ks*. *number-of-ex-vars* = *length ks*
$\wedge$ ($\forall$ *k*<*length d-domain*. *eval* (*step-relation* (*d-domain* ! k)) (*push-list a ks*))
$\wedge$ *eval params-are-sum-terms* (*push-list a ks*))
unfolding *DS-def* **by** (*simp add*: *defs*)

also have ... = ($\exists$ *ks*. *number-of-ex-vars* = *length ks* $\wedge$
($\forall$ *k*<*m*.
peval (s ! (*d-domain* ! k)) a =
peval b a $*$ *peval* ([$\sum$ *S*+] p (*d-domain* ! k) ((!) s')) (*push-list a ks*) +
peval b a $*$ *peval* ($\sum$ *S−-Param* (*d-domain* ! k)) (*push-list a ks*) +
peval b a $*$ *peval* ($\sum$ *S0-Param* (*d-domain* ! k)) (*push-list a ks*)) $\wedge$
eval params-are-sum-terms (*push-list a ks*))
unfolding *step-relation-def* ‹*length d-domain* = m›
using b'*-unfold* s'*-unfold* **by** (*auto simp*: *defs*)

also have ... = ($\exists$ *ks*. *number-of-ex-vars* = *length ks* $\wedge$
($\forall$ *k*<*m*.
peval (s ! (*d-domain* ! k)) a =
peval b a $*$ ($\sum$ *S*+ p (*d-domain* ! k) (λx. *peval* (s' ! x) (*push-list a ks*)))
+
peval b a $*$ (*peval* ($\sum$ *S−-Param* (*d-domain* ! k)) (*push-list a ks*)) +
peval b a $*$ (*peval* ($\sum$ *S0-Param* (*d-domain* ! k)) (*push-list a ks*)))
$\wedge$ ($\forall$ *k*<*m*.
peval ($\sum$ *S−-Param* (*d-domain* ! k)) (*push-list a ks*)
= $\sum$ *S−* p (*d-domain* ! k) (λk. *peval* (s' ! k) (*push-list a ks*)
&& *peval* (*z0* ! k) (*push-list a ks*))
$\wedge$ *peval* ($\sum$ *S0-Param* (*d-domain* ! k)) (*push-list a ks*)
= $\sum$ *S0* p (*d-domain* ! k) (λk. *peval* (s' ! k) (*push-list a ks*)
&& *peval* (*z1* ! k) (*push-list a ks*))))
unfolding *params-are-sum-terms-def param-is-sum-ssub-nzero-term-def*
param-is-sum-ssub-zero-term-def **apply** (*simp add*: *defs*)
using *sum-rsub-nzero-of-bit-and-eval*[*of* s' *z0*] *sum-rsub-zero-of-bit-and-eval*[*of* s' *z1*]
‹*length* $p > 0$› ‹*length* s' = *Suc m*› ‹*length z0* = *Suc m*› ‹*length z1* = *Suc m*›

unfolding ‹*length d-domain* = *m*› **by** (*simp add*: *defs*)

also have ... = (∃ *ks*. *number-of-ex-vars* = *length ks* ∧
(∀ *k*<*m*.
peval (*s* ! (*d-domain* ! *k*)) *a* =
peval b a ∗ (∑ *S*+ *p* (*d-domain* ! *k*) ((!) (*map* (λ*P*. *peval P a*) *s*)))
+ *peval b a* ∗ (∑ *S*− *p* (*d-domain* ! *k*) (λ*k*. *map* (λ*P*. *peval P a*) *s* ! *k*
&& *map* (λ*P*. *peval P a*) *z* ! *modifies* (*p* ! *k*)))
+ *peval b a* ∗ (∑ *S0 p* (*d-domain* ! *k*) (λ*k*. *map* (λ*P*. *peval P a*) *s* ! *k*
&& *peval e a* − *map* (λ*P*. *peval P a*) *z* ! *modifies* (*p* !
k))))
∧ (∀ *k*<*m*.
peval (∑ *S*−-*Param* (*d-domain* ! *k*)) (*push-list a ks*)
= ∑ *S*− *p* (*d-domain* ! *k*) (λ*k*. *map* (λ*P*. *peval P a*) *s* ! *k*
&& *map* (λ*P*. *peval P a*) *z* ! *modifies* (*p* ! *k*))
∧ *peval* (∑ *S0-Param* (*d-domain* ! *k*)) (*push-list a ks*)
= ∑ *S0 p* (*d-domain* ! *k*) (λ*k*. *map* (λ*P*. *peval P a*) *s* ! *k*
&& *peval e a* − *map* (λ*P*. *peval P a*) *z* ! *modifies* (*p* !
k))))
using *sum-sadd-unfold sum-ssub-nzero-unfold sum-ssub-zero-unfold* **by** *auto*

also have ... = (∀ *k*<*m*.
peval (*s* ! (*d-domain* ! *k*)) *a* =
peval b a ∗ (∑ *S*+ *p* (*d-domain* ! *k*) ((!) (*map* (λ*P*. *peval P a*) *s*)))
+ *peval b a* ∗ (∑ *S*− *p* (*d-domain* ! *k*)
(λ*k*. *map* (λ*P*. *peval P a*) *s* ! *k*
&& *map* (λ*P*. *peval P a*) *z* ! *modifies* (*p* ! *k*)))
+ *peval b a* ∗ (∑ *S0 p* (*d-domain* ! *k*)
(λ*k*. *map* (λ*P*. *peval P a*) *s* ! *k*
&& *peval e a* − *map* (λ*P*. *peval P a*) *z* ! *modifies* (*p* !
k))))
apply *auto*
apply (*rule exI*[*of* - *ks-ex*])
using ‹*length ks-ex* = *number-of-ex-vars*› *ks-ex1 ks-ex2* **by** *auto*

also have ... = (∀ *d*>*0*. *d* ≤ *m* ⟶
peval (*s* ! *d*) *a*
= *peval b a* ∗ ∑ *S*+ *p d* ((!) (*map* (λ*P*. *peval P a*) *s*))
+ *peval b a* ∗ ∑ *S*− *p d* (λ*k*. *map* (λ*P*. *peval P a*) *s* ! *k*
&& *map* (λ*P*. *peval P a*) *z* ! *modifies* (*p* ! *k*))
+ *peval b a* ∗ ∑ *S0 p d* (λ*k*. *map* (λ*P*. *peval P a*) *s* ! *k*
&& *peval e a* − *map* (λ*P*. *peval P a*) *z* ! *modifies* (*p*
! *k*)))
using *all-quantifier-switch*[*of* λ*d*. *peval* (*s* ! *d*) *a* =
peval b a ∗ ∑ *S*+ *p d* ((!) (*map* (λ*P*. *peval P a*) *s*)) +
peval b a ∗ ∑ *S*− *p d* (λ*k*. *map* (λ*P*. *peval P a*) *s* ! *k*
&& *map* (λ*P*. *peval P a*) *z* ! *modifies* (*p* ! *k*)) +
peval b a ∗ ∑ *S0 p d* (λ*k*. *map* (λ*P*. *peval P a*) *s* ! *k*
&& *peval e a* − *map* (λ*P*. *peval P a*) *z* ! *modifies* (*p* ! *k*))]

unfolding ‹*length d-domain* = *m*› **by** *auto*

finally show *?thesis*
unfolding *DR-def*
using *local.register-machine-axioms rm-eq-fixes-def*[*of p n*] *rm-eq-fixes.state-d-def*[*of p n*]
apply (*simp add*: *defs*)
using *nth-map*[*of - s λP. peval P a*] ‹*length s* = *Suc m*›
by *auto*
qed

moreover have *is-dioph-rel DS*
proof −
have *is-dioph-rel* (*param-is-sum-ssub-nzero-term d* [∧] *param-is-sum-ssub-zero-term d*) **for** *d*
unfolding *param-is-sum-ssub-nzero-term-def param-is-sum-ssub-zero-term-def*

by (*auto simp*: *dioph*)
hence *1*: *list-all* (*is-dioph-rel* ∘ (*λd. param-is-sum-ssub-nzero-term d*
[∧] *param-is-sum-ssub-zero-term d*)) *d-domain*
by (*simp add*: *list.inducts*)

have *is-dioph-rel* (*step-relation d*) **for** *d*
unfolding *step-relation-def* **by** (*auto simp*: *dioph*)
hence *2*: *list-all* (*is-dioph-rel* ∘ *step-relation*) *d-domain*
by (*simp add*: *list.inducts*)

show *?thesis*
unfolding *DS-def params-are-sum-terms-def* **by** (*auto simp*: *dioph 1 2*)
qed

ultimately show *?thesis* **using** *is-dioph-rel-def* **by** *auto*
qed

lemma *state-relations-from-recursion-dioph*:
fixes *b e* :: *polynomial*
fixes *z s* :: *polynomial list*
assumes *length z* = *n length s* = *Suc m*
defines *DR* ≡ *LARY* (*λll. rm-eq-fixes.state-relations-from-recursion p* (*ll*!*0*!*0*) (*ll*!*0*!*1*)
(*nth* (*ll*!*1*)) (*nth* (*ll*!*2*)))
[[*b*, *e*], *z*, *s*]
shows *is-dioph-rel DR*
proof −

define *DS* **where** *DS* ≡ (*LARY* (*λll. rm-eq-fixes.state-0 p* (*ll*!*0*!*0*) (*ll*!*0*!*1*)
(*nth* (*ll*!*1*)) (*nth* (*ll*!*2*))) [[*b*, *e*], *z*, *s*])
[∧](*LARY* (*λll. rm-eq-fixes.state-d p* (*ll*!*0*!*0*) (*ll*!*0*!*1*) (*nth* (*ll*!*1*))

(*nth* (*ll*!*2*))) [[*b*, *e*], *z*, *s*])

have *eval DS a* = *eval DR a* **for** *a*
unfolding *DS-def DR-def*
using *local.register-machine-axioms rm-eq-fixes-def*
rm-eq-fixes.state-relations-from-recursion-def
using *assms* **by** (*simp add*: *defs*)

moreover have *is-dioph-rel DS*
unfolding *DS-def* **apply** (*rule and-dioph*) **using** *assms state-0-dioph state-d-dioph*
by *blast+*

ultimately show *?thesis* **using** *is-dioph-rel-def* **by** *auto*
qed

end

end

4.4.5 State unique equations

theory *State-Unique-Equations* **imports** *../Register-Machine/MultipleStepState*
Equation-Setup ../Diophantine/Register-Machine-Sums

../Diophantine/Binary-And

begin

context *rm-eq-fixes*
begin

Equations not in the book:

definition *state-mask* :: *bool* **where**
state-mask ≡ ∀ *k*≤*m*. *s k* ⪯ *e*

definition *state-bound* :: *bool* **where**
state-bound ≡ ∀ *k*<*m*. *s k* < *b* ^ *q*

definition *state-unique-equations* :: *bool* **where**
state-unique-equations ≡ *state-mask* ∧ *state-bound*

end

context *register-machine*
begin

lemma *state-mask-dioph*:

fixes *e* :: *polynomial*
fixes *s* :: *polynomial list*
assumes *length s* = *Suc m*
defines *DR* ≡ *LARY* (λ*ll*. *rm-eq-fixes.state-mask p* (*ll*!*0*!*0*) (*nth* (*ll*!*1*))) [[*e*], *s*]
shows *is-dioph-rel DR*
proof −
define *mask* **where** *mask* ≡ (λ*l*. (*s*!*l* [⪯] *e*))
define *DS* **where** *DS* ≡ [∀<*Suc m*] *mask*

have *eval DS a* = *eval DR a* **for** *a*
proof −
have *eval DS a* = (∀ *k*≤*m*. *peval* (*s* ! *k*) *a* ⪯ *peval e a*)
unfolding *DS-def mask-def* **by** (*simp add*: *less-Suc-eq-le defs*)

also have ... = (∀ *k*≤*m*. *map* (λ*P*. *peval P a*) *s* ! *k* ⪯ *peval e a*)
using *nth-map*[*of - s* (λ*P*. *peval P a*)] ‹*length s* = *Suc m*› **by** *auto*

finally show *?thesis*
unfolding *DR-def* **using** *rm-eq-fixes-def local.register-machine-axioms*
rm-eq-fixes.state-mask-def **by** (*simp add*: *defs*)
qed

moreover have *is-dioph-rel DS*
proof −
have *list-all* (*is-dioph-rel* ∘ *mask*) [*0*..<*Suc m*]
unfolding *mask-def list-all-def* **by** (*auto simp*: *dioph*)
thus *?thesis* **unfolding** *DS-def mask-def* **by** (*auto simp*: *dioph*)
qed

ultimately show *?thesis*
by (*auto simp*: *is-dioph-rel-def*)
qed

lemma *state-bound-dioph*:
fixes *b q* :: *polynomial*
fixes *s* :: *polynomial list*
assumes *length s* = *Suc m*
defines *DR* ≡ *LARY* (λ*ll*. *rm-eq-fixes.state-bound p* (*ll*!*0*!*0*) (*ll*!*0*!*1*) (*nth* (*ll*!*1*)))
[[*b*, *q*], *s*]
shows *is-dioph-rel DR*
proof −
let *?N* = *m*
define *b*′ *q*′ *s*′ **where** *pushed-def*: *b*′ = *push-param b ?N*
q′ = *push-param q ?N*
s′ = *map* (λ*x*. *push-param x ?N*) *s*

define *bound* **where**
bound ≡ λ*l*. *s*′!*l* [<] (*Param l*) [∧] [*Param l* = *b*′ ⌢ *q*′]

```
define DS where DS ≡ [∃ m] [∀ <m] bound

have eval DS a = eval DR a for a
proof −

  have s′-unfold: peval (s′ ! k) (push-list a ks) = peval (s ! k) a
    if length ks = m and k < length ks for k ks
    unfolding pushed-def
    using push-push-map-i[of ks n k s] that ‹length s = Suc m› list-eval-def
    by (metis less-SucI nth-map push-push)

  have b′-unfold: peval b′ (push-list a ks) = peval b a
   and q′-unfold: peval q′ (push-list a ks) = peval q a
    if length ks = m and k < length ks for k ks
    unfolding pushed-def
    using push-push-simp that ‹length s = Suc m› list-eval-def by metis+

  have eval DS a = (∃ ks. m = length ks ∧
        (∀ k<m. peval (s′ ! k) (push-list a ks) < push-list a ks k ∧
              push-list a ks k = peval b′ (push-list a ks) ^ peval q′ (push-list a ks)))
    unfolding DS-def bound-def by (simp add: defs)

  also have ... = (∃ ks. m = length ks ∧
        (∀ k<m. peval (s ! k) a < peval b a ^ peval q a ∧
              push-list a ks k = peval b a ^ peval q a))
    using s′-unfold b′-unfold q′-unfold by metis

  also have ... = (∀ k<m. peval (s ! k) a < peval b a ^ peval q a)
    apply auto apply (rule exI[of - map (λk. peval b a ^ peval q a) [0..<m]])
    unfolding push-list-def by auto

  also have ... = (∀ l<m. map (λP. peval P a) s ! l < peval b a ^ peval q a)
    using nth-map[of - s λP. peval P a] ‹length s = Suc m› by force

  finally show ?thesis unfolding DR-def
   using rm-eq-fixes-def local.register-machine-axioms rm-eq-fixes.state-bound-def

    by (simp add: defs)

qed

moreover have is-dioph-rel DS
proof −
  have list-all (is-dioph-rel ∘ bound) [0..<Suc m]
    unfolding bound-def list-all-def by (auto simp:dioph)
  thus ?thesis unfolding DS-def bound-def by (auto simp: dioph)
qed

ultimately show ?thesis
```

by (*auto simp*: *is-dioph-rel-def*)
qed

lemma *state-unique-equations-dioph*:
fixes *b q e* :: *polynomial*
fixes *s* :: *polynomial list*
assumes *length s* = *Suc m*
defines *DR* ≡ *LARY*
(*λll. rm-eq-fixes.state-unique-equations p* (*ll*!*0*!*0*) (*ll*!*0*!*1*) (*ll*!*0*!*2*) (*nth* (*ll*!*1*)))
[[*b*, *e*, *q*], *s*]
shows *is-dioph-rel DR*
proof −

define *DS* **where** *DS* ≡ *LARY* (*λll. rm-eq-fixes.state-mask p* (*ll*!*0*!*0*) (*nth* (*ll*!*1*))) [[*e*], *s*]
[∧] *LARY* (*λll. rm-eq-fixes.state-bound p* (*ll*!*0*!*0*) (*ll*!*0*!*1*) (*nth* (*ll*!*1*)))
[[*b*, *q*], *s*]

have *eval DS a* = *eval DR a* **for** *a*
unfolding *DR-def DS-def* **using** *rm-eq-fixes.state-unique-equations-def rm-eq-fixes-def local.register-machine-axioms*
by (*auto simp*: *defs*)

moreover have *is-dioph-rel DS*
unfolding *DS-def* **using** *state-bound-dioph state-mask-dioph assms dioph* **by** *auto*

ultimately show *?thesis* **using** *is-dioph-rel-def* **by** *auto*
qed

end

end

4.4.6 Wrap-up: Combining all state equations

theory *All-State-Equations* **imports** *State-Unique-Equations State-d-Equation*

begin

The remaining equations:

context *rm-eq-fixes*
begin

Equation 4.27

definition *state-m* :: *bool* **where**
state-m ≡ *s m* = *b* ⌒ *q*

Equation not in the book

definition *state-partial-sum-mask* :: *bool* **where**
state-partial-sum-mask ≡ ∀ *M*≤*m*. ($\sum$ *k*≤*M*. *s k*) ⪯ *e*

Wrapping it all up

definition *state-equations* :: *bool* **where**
state-equations ≡ *state-relations-from-recursion* ∧ *state-unique-equations*
∧ *state-partial-sum-mask* ∧ *state-m*

end

context *register-machine*
begin

lemma *state-m-dioph*:
fixes *b q* :: *polynomial*
fixes *s* :: *polynomial list*
assumes *length s* = *Suc m*
defines *DR* ≡ *LARY* (λ*ll*. *rm-eq-fixes.state-m p* (*ll*!*0*!*0*) (*ll*!*0*!*1*) (*nth* (*ll*!*1*)))
[[*b*, *q*], *s*]
shows *is-dioph-rel DR*
proof −
define *DS* **where** *DS* ≡ [(*s*!*m*) = *b* ^ *q*]

have *eval DS a* = *eval DR a* **for** *a*
using *DS-def DR-def rm-eq-fixes.state-m-def rm-eq-fixes-def local.register-machine-axioms*

using *assms* **by** (*simp add*: *defs*)

moreover have *is-dioph-rel DS*
unfolding *DS-def* **by** (*auto simp*: *dioph*)

ultimately show *?thesis* **using** *is-dioph-rel-def* **by** *auto*
qed

lemma *state-partial-sum-mask-dioph*:
fixes *e* :: *polynomial*
fixes *s* :: *polynomial list*
assumes *length s* = *Suc m*
defines *DR* ≡ *LARY* (λ*ll*. *rm-eq-fixes.state-partial-sum-mask p* (*ll*!*0*!*0*) (*nth*
(*ll*!*1*))) [[*e*], *s*]
shows *is-dioph-rel DR*
proof −

define *partial-sum-mask* **where** *partial-sum-mask* ≡ (λ*m*. (*sum-polynomial* (*nth*
s) [*0*..<*Suc m*] [⪯] *e*))
define *DS* **where** *DS* ≡ [∀<*Suc m*] *partial-sum-mask*

have *eval DS a* = *eval DR a* **for** *a*

proof −

have *aux*: $((\sum j = 0..<k.\ peval\ (s\ !\ (([0..<Suc\ k])\ !\ j))\ a)$
$+ peval\ (s\ !\ (([0..<Suc\ k])\ !\ k))\ a \preceq peval\ e\ a)$
$= ((\sum j = 0..<k.\ peval\ (s\ !\ j)\ a)$
$+ peval\ (s\ !\ k)\ a \preceq peval\ e\ a)$ **for** *k*
proof −
have $[0..<Suc\ k]\ !\ k = 0 + k$
using *nth-upt*[*of 0 k Suc k*] **by** *simp*

moreover have $(\sum j = 0..<k.\ peval\ (s\ !\ (([0..<Suc\ k])\ !\ j))\ a)$
$= (\sum j = 0..<k.\ peval\ (s\ !\ j)\ a)$
apply (*rule sum.cong*, *simp*) **using** *nth-upt*[*of 0 - Suc k*]
by (*metis Suc-lessD add-cancel-right-left ex-nat-less-eq not-less-eq*)
ultimately show *?thesis*
by *auto*
qed

have *aux2*: $(\sum j = 0..<Suc\ k.\ peval\ (s\ !\ j)\ a) =$
$(sum\ ((!)\ (map\ (\lambda P.\ peval\ P\ a)\ s))\ \{..k\})$ **if** $k \le m$ **for** *k*
apply (*rule sum.cong*, *auto*)
by (*metis assms*(*1*) *dual-order.strict-trans le-imp-less-Suc nth-map*
order.not-eq-order-implies-strict that)

have *eval DS a* = $(\forall k<Suc\ m.$
$(\sum j = 0..<k.\ peval\ (s\ !\ j)\ a) + peval\ (s\ !\ k)\ a \preceq peval\ e\ a)$
unfolding *DS-def partial-sum-mask-def* **using** *aux*
by (*simp add*: *defs* ‹*length s = Suc m*› *sum-polynomial-eval*)

also have ... = $(\forall k \le m.$
$(\sum j = 0..<k.\ peval\ (s\ !\ j)\ a) + peval\ (s\ !\ k)\ a \preceq peval\ e\ a)$
by (*simp add*: *less-Suc-eq-le*)

finally show *?thesis* **using** *rm-eq-fixes-def local.register-machine-axioms DR-def*

rm-eq-fixes.state-partial-sum-mask-def aux2 **by** (*simp add*: *defs*)
qed

moreover have *is-dioph-rel DS*
unfolding *DS-def partial-sum-mask-def* **by** (*auto simp*: *dioph*)

ultimately show *?thesis* **using** *is-dioph-rel-def* **by** *auto*
qed

definition *state-equations-relation* :: *polynomial* ⇒ *polynomial* ⇒ *polynomial* ⇒ *polynomial list*
⇒ *polynomial list* ⇒ *relation* (‹[*STATE*] - - - - -›)**where**
[*STATE*] *b e q z s* ≡ *LARY* (λ*ll*. *rm-eq-fixes.state-equations p* (*ll*!*0*!*0*) (*ll*!*0*!*1*) (*ll*!*0*!*2*)

(*nth* (*ll*!*1*)) (*nth* (*ll*!*2*)))
[[*b*, *e*, *q*], *z*, *s*]

lemma *state-equations-dioph*:
fixes *b e q* :: *polynomial*
fixes *s z* :: *polynomial list*
assumes *length s* = *Suc m length z* = *n*
defines *DR* ≡ [*STATE*] *b e q z s*
shows *is-dioph-rel DR*
proof −

define *DS* **where**
DS ≡ (*LARY* (λ*ll*. *rm-eq-fixes.state-relations-from-recursion p* (*ll*!*0*!*0*) (*ll*!*0*!*1*)
(*nth* (*ll*!*1*)) (*nth* (*ll*!*2*))) [[*b*, *e*], *z*, *s*])
[∧] (*LARY* (λ*ll*. *rm-eq-fixes.state-unique-equations p* (*ll*!*0*!*0*) (*ll*!*0*!*1*) (*ll*!*0*!*2*)
(*nth* (*ll*!*1*)))
[[*b*, *e*, *q*], *s*])
[∧] (*LARY* (λ*ll*. *rm-eq-fixes.state-partial-sum-mask p* (*ll*!*0*!*0*) (*nth* (*ll*!*1*)))
[[*e*], *s*])
[∧] (*LARY* (λ*ll*. *rm-eq-fixes.state-m p* (*ll*!*0*!*0*) (*ll*!*0*!*1*) (*nth* (*ll*!*1*))) [[*b*, *q*],
s])

have *eval DS a* = *eval DR a* **for** *a*
using *DS-def DR-def rm-eq-fixes.state-equations-def*
state-equations-relation-def rm-eq-fixes-def local.register-machine-axioms **by** (*auto*
simp: *defs*)

moreover have *is-dioph-rel DS*
unfolding *DS-def* **using** *assms state-relations-from-recursion-dioph*[*of z s*]
state-m-dioph[*of s*]
state-partial-sum-mask-dioph state-unique-equations-dioph and-dioph
by (*auto simp*: *dioph*)

ultimately show *?thesis* **using** *is-dioph-rel-def* **by** *auto*
qed

end

end

4.4.7 Equations for masking relations

theory *Mask-Equations*
imports ../*Register-Machine*/*MachineMasking Equation-Setup* ../*Diophantine*/*Binary-And*

abbrevs *mb* = ≼

begin

context *rm-eq-fixes*
begin

Equation 4.15

definition *register-mask* :: *bool* **where**
register-mask ≡ ∀ *l* < *n*. *r l* ⪯ *d*

Equation 4.17

definition *zero-indicator-mask* :: *bool* **where**
zero-indicator-mask ≡ ∀ *l* < *n*. *z l* ⪯ *e*

Equation 4.20

definition *zero-indicator-0-or-1* :: *bool* **where**
zero-indicator-0-or-1 ≡ ∀ *l*<*n*. *2^c* ∗ *z l* = (*r l* + *d*) && *f*

definition *mask-equations* :: *bool* **where**
mask-equations ≡ *register-mask* ∧ *zero-indicator-mask* ∧ *zero-indicator-0-or-1*

end

context *register-machine*
begin

lemma *register-mask-dioph*:
fixes *d r*
assumes *n* = *length r*
defines *DR* ≡ (*NARY* (λ*l*. *rm-eq-fixes.register-mask n* (*l*!*0*) (*shift l 1*)) ([*d*] @ *r*))
shows *is-dioph-rel DR*
proof −
define *DS* **where** *DS* ≡ [∀<*n*] (λ*i*. ((*r*!*i*) [⪯] *d*))

have *eval DS a* = *eval DR a* **for** *a*
proof −
have *eval DR a* = *rm-eq-fixes.register-mask n* (*peval d a*) (*list-eval r a*)
unfolding *DR-def* **by** (*auto simp add*: *shift-def list-eval-def*)
also have ... = (∀ *l* < *n*. (*peval* (*r*!*l*) *a*) ⪯ *peval d a*)
using *rm-eq-fixes.register-mask-def* ‹*n* = *length r*› *rm-eq-fixes-def*
local.register-machine-axioms **by** (*auto simp*: *list-eval-def*)
finally show *?thesis*
unfolding *DS-def defs* **by** *simp*
qed

moreover have *is-dioph-rel DS*
unfolding *DS-def* **by** (*auto simp add*: *dioph*)

ultimately show *?thesis*
by (*simp add*: *is-dioph-rel-def*)
qed

lemma *zero-indicator-mask-dioph*:
  fixes *e z*
  assumes *n = length z*
  defines *DR ≡ (NARY (λl. rm-eq-fixes.zero-indicator-mask n (l!0) (shift l 1)) ([e] @ z))*
  shows *is-dioph-rel DR*
proof −
  define *DS* **where** *DS ≡ [∀<n] (λi. ((z!i) [⪯] e))*

  have *eval DS a = eval DR a* **for** *a*
  proof −
    have *eval DR a = rm-eq-fixes.zero-indicator-mask n (peval e a) (list-eval z a)*
      unfolding *DR-def* **by** *(auto simp add: shift-def list-eval-def)*
    also have ... *= (∀ l < n. (peval (z!l) a) ⪯ peval e a)*
      using *rm-eq-fixes.zero-indicator-mask-def ‹n = length z›*
      rm-eq-fixes-def local.register-machine-axioms **by** *(auto simp: list-eval-def)*
    finally show *?thesis*
      unfolding *DS-def defs* **by** *simp*
  qed

  moreover have *is-dioph-rel DS*
    unfolding *DS-def* **by** *(auto simp add: dioph)*

  ultimately show *?thesis*
    by *(simp add: is-dioph-rel-def)*
qed

lemma *zero-indicator-0-or-1-dioph*:
  fixes *c d f r z*
  assumes *n = length r* **and** *n = length z*
  defines *DR ≡ LARY (λll. rm-eq-fixes.zero-indicator-0-or-1 n (ll!0!0) (ll!0!1) (ll!0!2)*
                          (nth (ll!1)) (nth (ll!2))) [[c, d, f], r, z]
  shows *is-dioph-rel DR*
proof −
  let *?N = 2*
  define *c′ d′ f′ r′ z′* **where** *pushed-def*: *c′ = push-param c ?N d′ = push-param d ?N*
                          f′ = push-param f ?N r′ = map (λx. push-param x ?N) r
                          z′ = map (λx. push-param x ?N) z
  define *DS* **where** *DS ≡ [∀<n] (λi. ([∃2] [Param 0 = (Const 2) ^ c′]*
                                   [∧] [Param 1 = (r′!i) [+] d′ && f′]
                                   [∧] Param 0 [*] (z′!i) [=] Param 1))

  have *eval DS a = eval DR a* **for** *a*
  proof −
    have *eval DR a = rm-eq-fixes.zero-indicator-0-or-1 n (peval c a) (peval d a) (peval f a)*

(*list-eval r a*) (*list-eval z a*)
unfolding *DR-def defs* **by** (*auto simp add*: *assms shift-def list-eval-def*)
also have ... = (∀ *l* < *n*. *2*^(*peval c a*) ∗ (*peval* (*z*!*l*) *a*)
= (*peval* (*r*!*l*) *a* + *peval d a*) && *peval f a*)
using *rm-eq-fixes.zero-indicator-0-or-1-def* ‹*n* = *length r*› **using** *assms*
rm-eq-fixes-def local.register-machine-axioms **by** (*auto simp*: *list-eval-def*)
finally show *?thesis*
unfolding *DS-def defs pushed-def* **using** *push-push* **apply** (*auto*)
subgoal for *k*
apply (*rule exI*[*of* - [*2*^*peval c a*, *peval* (*r*!*k*) *a* + *peval d a* && *peval f a*]])
apply (*auto simp*: *push-list-def assms*(*1*−*2*))
by (*metis assms*(*1*) *assms*(*2*) *length-Cons list.size*(*3*) *nth-map numeral-2-eq-2*)
subgoal
using *assms* **by** *auto*
done
qed

moreover have *is-dioph-rel DS*
unfolding *DS-def* **by** (*auto simp add*: *dioph*)

ultimately show *?thesis*
by (*simp add*: *is-dioph-rel-def*)
qed

definition *mask-equations-relation* (‹[*MASK*] - - - - - -›) **where**
[*MASK*] *c d e f r z* ≡ *LARY* (λ*ll*. *rm-eq-fixes.mask-equations n*
(*ll*!*0*!*0*) (*ll*!*0*!*1*) (*ll*!*0*!*2*) (*ll*!*0*!*3*) (*nth* (*ll*!*1*)) (*nth* (*ll*!*2*)))
[[*c*, *d*, *e*, *f*], *r*, *z*]

lemma *mask-equations-relation-dioph*:
fixes *c d e f r z*
assumes *n* = *length r* **and** *n* = *length z*
defines *DR* ≡ [*MASK*] *c d e f r z*
shows *is-dioph-rel DR*
proof −
define *DS* **where** *DS* ≡ *NARY* (λ*l*. *rm-eq-fixes.register-mask n* (*l*!*0*) (*shift l 1*))
([*d*] @ *r*)
[∧] *NARY* (λ*l*. *rm-eq-fixes.zero-indicator-mask n* (*l*!*0*) (*shift l 1*)) ([*e*] @ *z*)
[∧] *LARY* (λ*ll*. *rm-eq-fixes.zero-indicator-0-or-1 n* (*ll*!*0*!*0*) (*ll*!*0*!*1*) (*ll*!*0*!*2*)
(*nth* (*ll*!*1*)) (*nth* (*ll*!*2*))) [[*c*, *d*, *f*], *r*, *z*]

have *eval DS a* = *eval DR a* **for** *a*
using *DS-def DR-def mask-equations-relation-def rm-eq-fixes.mask-equations-def*
rm-eq-fixes-def local.register-machine-axioms **by** (*simp add*: *defs shift-def*)

moreover have *is-dioph-rel DS*
unfolding *DS-def* **using** *assms dioph*
using *register-mask-dioph zero-indicator-mask-dioph zero-indicator-0-or-1-dioph*

by (*metis* (*no-types*, *lifting*))

ultimately show *?thesis*
by (*simp add*: *is-dioph-rel-def*)
qed

end

end

4.4.8 Equations for arithmetization constants

theory *Constants-Equations* **imports** *Equation-Setup ../Register-Machine/MachineMasking ../Diophantine/Binary-And*

begin

context *rm-eq-fixes*
begin

Equation 4.14

definition *constant-b* :: *bool* **where**
constant-b ≡ *b* = *B c*

Equation 4.16

definition *constant-d* :: *bool* **where**
constant-d ≡ *d* = *D q c b*

Equation 4.18

definition *constant-e* :: *bool* **where**
constant-e ≡ *e* = *E q b*

Equation 4.21

definition *constant-f* :: *bool* **where**
constant-f ≡ *f* = *F q c b*

Equation not in the book

definition *c-gt-0* :: *bool* **where**
c-gt-0 ≡ *c* > *0*

Equation 4.26

definition *a-bound* :: *bool* **where**
a-bound ≡ *a* < *2* ^ *c*

Equation not in the book

definition *q-gt-0* :: *bool* **where**
q-gt-0 ≡ *q* > *0*

definition *constants-equations* :: *bool* **where**
constants-equations ≡ *constant-b* ∧ *constant-d* ∧ *constant-e* ∧ *constant-f*

definition *miscellaneous-equations* :: *bool* **where**
miscellaneous-equations ≡ *c-gt-0* ∧ *a-bound* ∧ *q-gt-0*

end

context *register-machine*
begin

definition *rm-constant-equations* ::
polynomial ⇒ *polynomial* ⇒ *polynomial* ⇒ *polynomial* ⇒ *polynomial* ⇒ *polynomial* ⇒ *relation*
(‹[*CONST*] - - - - - -›) **where**
[*CONST*] *b c d e f q* ≡ *NARY* (λ*l*. *rm-eq-fixes.constants-equations*
(*l*!*0*) (*l*!*1*) (*l*!*2*) (*l*!*3*) (*l*!*4*) (*l*!*5*)) [*b*, *c*, *d*, *e*, *f*, *q*]

definition *rm-miscellaneous-equations* ::
polynomial ⇒ *polynomial* ⇒ *polynomial* ⇒ *relation*
(‹[*MISC*] - - -›) **where**
[*MISC*] *c a q* ≡ *NARY* (λ*l*. *rm-eq-fixes.miscellaneous-equations*
(*l*!*0*) (*l*!*1*) (*l*!*2*)) [*c*, *a*, *q*]

lemma *rm-constant-equations-dioph*:
fixes *b c d e f q*
defines *DR* ≡ [*CONST*] *b c d e f q*
shows *is-dioph-rel DR*
proof−
have *fx*: *rm-eq-fixes p n*
using *rm-eq-fixes-def local.register-machine-axioms* **by** *auto*

define *b′ c′ d′ e′ f′ q′* **where** *pushed-defs*:
b′ = (*push-param b 2*) *c′* = (*push-param c 2*) *d′* = (*push-param d 2*)
e′ = (*push-param e 2*) *f′* = (*push-param f 2*) *q′* = (*push-param q 2*)

define *s t* **where** *params-def*: *s* = *Param 0 t* = *Param 1*

define *DS1* **where** *DS1* ≡ [*b′* = *Const 2* ^ (*c′* [+] **1**)] [∧]
[*s* = *Const 2* ^ *c′*] [∧] [*t* = *b′* ^ (*q′* [+] **1**)] [∧]
(*b′* [−] **1**) [∗] *d′* [=] (*s* [−] **1**) [∗] (*t* [−] **1**)

define *DS2* **where** *DS2* ≡ (*b′* [−] **1**) [∗] *e′* [=] *t* [−] **1** [∧]
(*b′* [−] **1**) [∗] *f′* [=] *s* [∗] (*t* [−] **1**)

define *DS* **where** *DS* ≡ [∃ *2*] *DS1* [∧] *DS2*

have *eval DS a* = *eval DR a* **for** *a*
unfolding *DR-def DS-def DS1-def DS2-def rm-constant-equations-def defs*
apply (*auto simp add*: *fx rm-eq-fixes.constants-equations-def*[*of p n*])
unfolding *pushed-defs params-def push-push* **apply** (*auto simp add*: *push-list-eval*)
apply (*auto simp add*: *fx rm-eq-fixes.constant-b-def*[*of p n*] *B-def*
rm-eq-fixes.constant-d-def[*of p n*] *rm-eq-fixes.constant-e-def*[*of p n*]
rm-eq-fixes.constant-f-def[*of p n*])
using *d-geom-series*[*of 2 ∗ 2 ^ peval c a peval c a* (*peval q a*) *peval d a*]
using *e-geom-series*[*of* (*2 ∗ 2 ^ peval c a*) *peval q a peval e a*]
using *f-geom-series*[*of 2 ∗ 2 ^ peval c a peval c a* (*peval q a*) *peval f a*]
apply (*auto*)
apply (*rule exI*[*of - [2 ^ peval c a*, *peval b a ∗ peval b a ^ peval q a*]])
using *push-list-def push-push* **by** *auto*

moreover have *is-dioph-rel DS* **unfolding** *DS-def DS1-def DS2-def* **by** (*simp add*: *dioph*)

ultimately show *?thesis*
by (*simp add*: *is-dioph-rel-def*)
qed

lemma *rm-miscellaneous-equations-dioph*:
fixes *c a q*
defines *DR* ≡ [*MISC*] *a c q*
shows *is-dioph-rel DR*
proof−
define *c′ a′ q′* **where** *pushed-defs*:
c′ == (*push-param c 1*) *a′* == (*push-param a 1*) *q′* = (*push-param q 1*)

define *DS* **where** *DS* ≡ [∃] *c′* [>] **0**
[∧] [(*Param 0*) = (*Const 2*) ^ *c′*] [∧] *a′*[<] *Param 0*
[∧] *q′* [>] **0**

have *eval DS a* = *eval DR a* **for** *a* **unfolding** *DS-def defs DR-def*
using *rm-miscellaneous-equations-def*
rm-eq-fixes.miscellaneous-equations-def rm-eq-fixes.c-gt-0-def rm-eq-fixes.a-bound-def

rm-eq-fixes.q-gt-0-def rm-eq-fixes-def local.register-machine-axioms **apply** *auto*
unfolding *pushed-defs push-push1*
apply (*auto*, *rule exI*[*of - 2 ^ peval c a*]) **unfolding** *push0* **by** *auto*

moreover have *is-dioph-rel DS* **unfolding** *DS-def* **by** (*simp add*: *dioph*)

ultimately show *?thesis*
by (*simp add*: *is-dioph-rel-def*)
qed

end

end

4.4.9 Invariance of equations

theory *All-Equations-Invariance*
 imports *Register-Equations All-State-Equations Mask-Equations Constants-Equations*

begin

context *register-machine*
begin

definition *all-equations* **where**
 all-equations a q b c d e f r z s
 ≡ *rm-eq-fixes.register-equations p n a b q r z s*
 ∧ *rm-eq-fixes.state-equations p b e q z s*
 ∧ *rm-eq-fixes.mask-equations n c d e f r z*
 ∧ *rm-eq-fixes.constants-equations b c d e f q*
 ∧ *rm-eq-fixes.miscellaneous-equations a c q*

lemma *all-equations-invariance*:
 fixes *r z s* :: *nat* ⇒ *nat*
 and *r′ z′ s′* :: *nat* ⇒ *nat*
 assumes ∀ *i*<*n*. *r i* = *r′ i* **and** ∀ *i*<*n*. *z i* = *z′ i* **and** ∀ *i*<*Suc m*. *s i* = *s′ i*
 shows *all-equations a q b c d e f r z s* = *all-equations a q b c d e f r′ z′ s′*
proof −
 have *r*: *i*<*n* ⟶ *r i* = *r′ i* **for** *i*
 using *assms* **by** *auto*
 have *z*: *i*<*n* ⟶ *z i* = *z′ i* **for** *i*
 using *assms* **by** *auto*
 have *s*: *i*<*Suc m* ⟶ *s i* = *s′ i* **for** *i*
 using *assms* **by** *auto*

 have *length p* > *0* **using** *p-nonempty* **by** *auto*
 have *n* > *0* **using** *n-gt-0* **by** *auto*

 have *z-at-modifies*: *z* (*modifies* (*p* ! *k*)) = *z′* (*modifies* (*p* ! *k*)) **if** *k* < *length p* **for**
k
 using *z*[*of modifies* (*p*!*k*)] *m-def modifies-yields-valid-register that* **by** *auto*

 have *rm-eq-fixes.register-equations p n a b q r z s*
 = *rm-eq-fixes.register-equations p n a b q r′ z′ s′*
 proof −

 have *sum-radd*: ∑ *R*+ *p d s* = ∑ *R*+ *p d s′* **for** *d*
 by (*rule sum-radd-cong*, *auto simp*: *s m-def*)

have *sum-rsub*: $\sum R-\ p\ d\ (\lambda k.\ s\ k\ \&\&\ z\ d) = \sum R-\ p\ d\ (\lambda k.\ s'\ k\ \&\&\ z'\ d)$ **for** *d*
apply (*rule sum-rsub-cong*) **using** *s z m-def z-at-modifies* ‹*length p* > *0*›
by (*auto*, *metis Suc-pred* ‹*0* < *length p*› *le-imp-less-Suc*)

have *rm-eq-fixes.register-0 p a b r z s* = *rm-eq-fixes.register-0 p a b r′ z′ s′*
using *rm-eq-fixes-def local.register-machine-axioms rm-eq-fixes.register-0-def sum-radd*[*of 0*]
sum-rsub[*of 0*] **using** *r* ‹*n* > *0*› **by** *auto*

moreover have *rm-eq-fixes.register-l p n b r z s* = *rm-eq-fixes.register-l p n b r′ z′ s′*
using *rm-eq-fixes.register-l-def sum-radd sum-rsub rm-eq-fixes-def*
local.register-machine-axioms **using** *r* ‹*n* > *0*› **by** *auto*

moreover have *rm-eq-fixes.register-bound n b q r* = *rm-eq-fixes.register-bound n b q r′*
using *rm-eq-fixes-def local.register-machine-axioms rm-eq-fixes.register-bound-def*

using *r* **by** *auto*

ultimately show *?thesis*
using *rm-eq-fixes-def local.register-machine-axioms rm-eq-fixes.register-equations-def*

by *auto*
qed

moreover have *rm-eq-fixes.state-equations p b e q z s*
= *rm-eq-fixes.state-equations p b e q z′ s′*
proof −
have *rm-eq-fixes.state-relations-from-recursion p b e z s*
= *rm-eq-fixes.state-relations-from-recursion p b e z′ s′*
proof −

have *sum-sadd*: $\sum S+\ p\ d\ s = \sum S+\ p\ d\ s'$ **for** *d*
by (*rule sum-sadd-cong*, *auto simp*: *s m-def*)

have *sum-ssub-nzero*: $\sum S-\ p\ d\ (\lambda k.\ s\ k\ \&\&\ z\ (modifies\ (p\ !\ k)))$
$= \sum S-\ p\ d\ (\lambda k.\ s'\ k\ \&\&\ z'\ (modifies\ (p\ !\ k)))$ **for** *d*
apply (*rule sum-ssub-nzero-cong*) **using** *z-at-modifies z s*
by (*metis One-nat-def Suc-pred* ‹*0* < *length p*› *le-imp-less-Suc m-def*)

have *sum-ssub-zero*: $\sum S0\ p\ d\ (\lambda k.\ s\ k\ \&\&\ e - z\ (modifies\ (p\ !\ k)))$
$= \sum S0\ p\ d\ (\lambda k.\ s'\ k\ \&\&\ e - z'\ (modifies\ (p\ !\ k)))$ **for** *d*
apply (*rule sum-ssub-zero-cong*) **using** *z-at-modifies z s*
by (*metis One-nat-def Suc-pred* ‹*0* < *length p*› *le-imp-less-Suc m-def*)

have *rm-eq-fixes.state-0 p b e z s* = *rm-eq-fixes.state-0 p b e z′ s′*

using *rm-eq-fixes.state-0-def sum-sadd sum-ssub-nzero sum-ssub-zero rm-eq-fixes-def local.register-machine-axioms*
using *s* **by** *auto*

moreover have *rm-eq-fixes.state-d p b e z s = rm-eq-fixes.state-d p b e z′ s′*
using *rm-eq-fixes.state-d-def sum-sadd sum-ssub-nzero sum-ssub-zero rm-eq-fixes-def local.register-machine-axioms*
using *s* **by** *auto*

ultimately show *?thesis*
using *rm-eq-fixes-def local.register-machine-axioms rm-eq-fixes.state-relations-from-recursion-def* **by** *auto*
qed

moreover have *rm-eq-fixes.state-unique-equations p b e q s = rm-eq-fixes.state-unique-equations p b e q s′*
using *rm-eq-fixes.state-unique-equations-def rm-eq-fixes-def local.register-machine-axioms rm-eq-fixes.state-mask-def rm-eq-fixes.state-bound-def*
using *s* **by** *force*

ultimately show *?thesis*
using *rm-eq-fixes-def local.register-machine-axioms rm-eq-fixes.state-equations-def*

rm-eq-fixes.state-mask-def rm-eq-fixes.state-bound-def rm-eq-fixes.state-m-def rm-eq-fixes.state-partial-sum-mask-def **using** *s z* **by** *auto*
qed

moreover have *rm-eq-fixes.mask-equations n c d e f r z = rm-eq-fixes.mask-equations n c d e f r′ z′*
proof −
have *rm-eq-fixes.register-mask n d r = rm-eq-fixes.register-mask n d r′*
using *rm-eq-fixes-def local.register-machine-axioms rm-eq-fixes.register-mask-def r* **by** *auto*

moreover have *rm-eq-fixes.zero-indicator-mask n e z = rm-eq-fixes.zero-indicator-mask n e z′*
using *rm-eq-fixes.zero-indicator-mask-def rm-eq-fixes-def local.register-machine-axioms z*
by *auto*

moreover have *rm-eq-fixes.zero-indicator-0-or-1 n c d f r z = rm-eq-fixes.zero-indicator-0-or-1 n c d f r′ z′*
using *rm-eq-fixes-def local.register-machine-axioms rm-eq-fixes.zero-indicator-0-or-1-def*

using *r z* **by** *auto*

ultimately show *?thesis*
using *rm-eq-fixes-def local.register-machine-axioms rm-eq-fixes.mask-equations-def*

by *auto*
 qed

 ultimately show *?thesis*
  unfolding *all-equations-def* **by** *auto*
qed

end

end

4.4.10 Wrap-Up: Combining all equations

theory *All-Equations*
 imports *All-Equations-Invariance*

begin

context *register-machine*
begin

definition *all-equations-relation* :: *polynomial* ⇒ *polynomial* ⇒ *polynomial* ⇒ *polynomial*
 ⇒ *polynomial* ⇒ *polynomial* ⇒ *polynomial* ⇒ *polynomial list* ⇒ *polynomial list* ⇒ *polynomial list*
 ⇒ *relation* (‹[*ALLEQ*] - - - - - - - - - -›) **where**
 [*ALLEQ*] *a q b c d e f r z s*
  ≡ *LARY* (*λll. all-equations* (*ll*!*0*!*0*) (*ll*!*0*!*1*) (*ll*!*0*!*2*) (*ll*!*0*!*3*) (*ll*!*0*!*4*) (*ll*!*0*!*5*) (*ll*!*0*!*6*)
   (*nth* (*ll*!*1*)) (*nth* (*ll*!*2*)) (*nth* (*ll*!*3*)))
   [[*a*, *q*, *b*, *c*, *d*, *e*, *f*], *r*, *z*, *s*]

lemma *all-equations-dioph*:
 fixes *A f e d c b q* :: *polynomial*
 fixes *r z s* :: *polynomial list*
 assumes *length r* = *n length z* = *n length s* = *Suc m*
 defines *DR* ≡ [*ALLEQ*] *A q b c d e f r z s*
 shows *is-dioph-rel DR*
proof −
 define *DS* **where** *DS* ≡ ([*REG*] *A b q r z s*)
   [∧] ([*STATE*] *b e q z s*)
   [∧] ([*MASK*] *c d e f r z*)
   [∧] ([*CONST*] *b c d e f q*)
   [∧] [*MISC*] *A c q*

 have *eval DS a* = *eval DR a* **for** *a*
  unfolding *DR-def DS-def all-equations-relation-def all-equations-def*
  unfolding *register-equations-relation-def state-equations-relation-def*

```
    mask-equations-relation-def rm-constant-equations-def rm-miscellaneous-equations-def
    by (simp add: defs)

  moreover have is-dioph-rel DS
    unfolding DS-def apply (rule and-dioph)+
   apply (simp-all add: rm-constant-equations-dioph rm-miscellaneous-equations-dioph)
    using assms reg-dioph[of r z s A b q] state-equations-dioph[of s z b e q]
          mask-equations-relation-dioph[of r z c d e f] by metis+

  ultimately show ?thesis using is-dioph-rel-def by auto
qed

definition rm-equations :: nat ⇒ bool where
  rm-equations a ≡ ∃ q :: nat.
                   ∃ b c d e f :: nat.
                   ∃ r z :: register ⇒ nat.
                   ∃ s :: state ⇒ nat.
                   all-equations a q b c d e f r z s

definition rm-equations-relation :: polynomial ⇒ relation (‹[RM] -›) where
  [RM] A ≡ UNARY (rm-equations) A



lemma rm-dioph:
  fixes A
  fixes ic :: configuration
  defines DR ≡ [RM] A
  shows is-dioph-rel DR
proof −
  define q b c d e f where q ≡ Param 0 and
                           b ≡ Param 1 and
                           c ≡ Param 2 and
                           d ≡ Param 3 and
                           e ≡ Param 4 and
                           f ≡ Param 5

  define r where r ≡ map Param [6..<n + 6]
  define z where z ≡ map Param [n+6..<2∗n + 6]
  define s where s ≡ map Param [2∗n + 6..<2∗n + 6 + m + 1]

  define number-of-ex-vars where number-of-ex-vars ≡ 2∗n + 6 + m + 1

  define A′ where A′ ≡ push-param A number-of-ex-vars

  define DS where DS ≡ [∃ number-of-ex-vars] [ALLEQ] A′ q b c d e f r z s

  have length r = n and length z = n and length s = Suc m
    unfolding r-def z-def s-def by auto
```

have *eval DS a* = *eval DR a* **for** *a*
proof (*rule*)
assume *eval DS a*
then obtain *ks* **where**
ks-length: *number-of-ex-vars* = *length ks* **and**
ALLEQ: *eval* ([*ALLEQ*] *A′ q b c d e f r z s*) (*push-list a ks*)
unfolding *DS-def* **by** (*auto simp add*: *defs*)

define *q′ b′ c′ d′ e′ f′* **where** *q′* ≡ *ks*!*0* **and**
b′ ≡ *ks*!*1* **and**
c′ ≡ *ks*!*2* **and**
d′ ≡ *ks*!*3* **and**
e′ ≡ *ks*!*4* **and**
f′ ≡ *ks*!*5*

define *r-list* **where** *r-list* ≡ (*take n* (*drop 6 ks*))
define *z-list* **where** *z-list* ≡ (*take n* (*drop* (*6*+*n*) *ks*))
define *s-list* **where** *s-list* ≡ (*drop* (*6* + *2*∗*n*) *ks*)

define *r′* **where** *r′* ≡ (!) *r-list*
define *z′* **where** *z′* ≡ (!) *z-list*
define *s′* **where** *s′* ≡ (!) *s-list*

have *A*: *peval A′* (*push-list a ks*) = *peval A a* **for** *a*
using *ks-length push-push-simp* **unfolding** *A′-def* **by** *auto*

have *q*: *peval q* (*push-list a ks*) = *q′*
unfolding *q-def q′-def push-list-def* **using** *ks-length* **unfolding** *number-of-ex-vars-def* **by** *auto*
have *b*: *peval b* (*push-list a ks*) = *b′*
unfolding *b-def b′-def push-list-def* **using** *ks-length* **unfolding** *number-of-ex-vars-def* **by** *auto*
have *c*: *peval c* (*push-list a ks*) = *c′*
unfolding *c-def c′-def push-list-def* **using** *ks-length* **unfolding** *number-of-ex-vars-def* **by** *auto*
have *d*: *peval d* (*push-list a ks*) = *d′*
unfolding *d-def d′-def push-list-def* **using** *ks-length* **unfolding** *number-of-ex-vars-def* **by** *auto*
have *e*: *peval e* (*push-list a ks*) = *e′*
unfolding *e-def e′-def push-list-def* **using** *ks-length* **unfolding** *number-of-ex-vars-def* **by** *auto*
have *f*: *peval f* (*push-list a ks*) = *f′*
unfolding *f-def f′-def push-list-def* **using** *ks-length* **unfolding** *number-of-ex-vars-def* **by** *auto*

have *r*: (!) (*map* (*λP*. *peval P* (*push-list a ks*)) *r*) *x* = (!) *r-list x* **if** *x* < *n* **for** *x*
unfolding *r-def r-list-def* **using** *that* **unfolding** *push-list-def*

using *ks-length* **unfolding** *number-of-ex-vars-def* **by** *auto*

have *z*: (*map* (*λP*. *peval P* (*push-list a ks*)) *z*) ! *x* = *z-list* ! *x* **if** *x* < *n* **for** *x*
unfolding *z-def z-list-def* **using** *that* **unfolding** *push-list-def*
using *ks-length* **unfolding** *number-of-ex-vars-def* **by** (*auto simp add*: *add.commute*)

have *s*: (*map* (*λP*. *peval P* (*push-list a ks*)) *s*) ! *x* = *s-list* ! *x* **if** *x* < *Suc m* **for** *x*
unfolding *s-def s-list-def* **using** *that* **unfolding** *push-list-def*
using *ks-length* **unfolding** *number-of-ex-vars-def* **by** (*auto simp add*: *add.commute*)

have *all-equations* (*peval A a*) *q′ b′ c′ d′ e′ f′ r′ z′ s′*
using *ALLEQ* **unfolding** *all-equations-relation-def* **apply** (*simp add*: *defs*)
unfolding *A q b c d e f*
using *all-equations-invariance*[*of*
(!) (*map* (*λP*. *peval P* (*push-list a ks*)) *r*) *r′*
(!) (*map* (*λP*. *peval P* (*push-list a ks*)) *z*) *z′*
(!) (*map* (*λP*. *peval P* (*push-list a ks*)) *s*) *s′*
peval A a q′ b′ c′ d′ e′ f′] *r z s*
using *r′-def s′-def z′-def* **by** *fastforce*

thus *eval DR a*
unfolding *DR-def rm-equations-def rm-equations-relation-def* **by** (*auto simp*: *defs*) (*blast*)
next
assume *eval DR a*
then obtain *q′ b′ c′ d′ e′ f′ r′ z′ s′* **where**
all-eq: *all-equations* (*peval A a*) *q′ b′ c′ d′ e′ f′ r′ z′ s′*
unfolding *DR-def rm-equations-def rm-equations-relation-def* **by** (*auto simp*: *defs*)

define *r-list* **where** *r-list* ≡ *map r′* [*0*..<*n*]
define *z-list* **where** *z-list* ≡ *map z′* [*0*..<*n*]
define *s-list* **where** *s-list* ≡ *map s′* [*0*..<*Suc m*]

define *ks* **where** *ks* ≡ [*q′*, *b′*, *c′*, *d′*, *e′*, *f′*] @ *r-list* @ *z-list* @ *s-list*

have *number-of-ex-vars* = *length ks*
unfolding *number-of-ex-vars-def ks-def r-list-def z-list-def s-list-def* **by** *auto*

have *A*: *peval A′* (*push-list a ks*) = *peval A a* **for** *a*
unfolding *A′-def*
using *push-push-simp*[*of A ks a*] **unfolding** ‹*number-of-ex-vars* = *length ks*› **by** *auto*

have *q*: *peval q* (*push-list a ks*) = *q′*
unfolding *q-def ks-def push-list-def* **by** *auto*
have *b*: *peval b* (*push-list a ks*) = *b′*

unfolding *b-def ks-def push-list-def* **by** *auto*
have *c*: *peval c* (*push-list a ks*) = *c′*
unfolding *c-def ks-def push-list-def* **by** *auto*
have *d*: *peval d* (*push-list a ks*) = *d′*
unfolding *d-def ks-def push-list-def* **by** *auto*
have *e*: *peval e* (*push-list a ks*) = *e′*
unfolding *e-def ks-def push-list-def* **by** *auto*
have *f*: *peval f* (*push-list a ks*) = *f′*
unfolding *f-def ks-def push-list-def* **by** *auto*

have *r*: (*map* (*λP. peval P* (*push-list a ks*)) *r*) ! *x* = *r′ x* **if** *x* < *n* **for** *x*
using *that* **unfolding** *ks-def r-list-def r-def push-list-def*
using *nth-append*[*of map r′* [*0..*<*n*] *z-list* @ *s-list*] **by** *auto*

have *z*: (*map* (*λP. peval P* (*push-list a ks*)) *z*) ! *x* = *z′ x* **if** *x* < *n* **for** *x*
using *that* **unfolding** *ks-def z-list-def r-list-def z-def push-list-def* **apply** *simp*
using *nth-append*[*of map r′* [*0..*<*n*] @ *map z′* [*0..*<*n*] *s-list*]
by (*metis add-diff-cancel-left′ gen-length-def length-code length-map length-upt*

not-add-less1 nth-append nth-map-upt)

have *s*: (*map* (*λP. peval P* (*push-list a ks*)) *s*) ! *x* = *s′ x* **if** *x* < *Suc m* **for** *x*
using *that* **unfolding** *ks-def r-list-def z-list-def s-list-def s-def push-list-def*
apply *simp*
using *nth-append*[*of map r′* [*0..*<*n*] @ *map z′* [*0..*<*n*] *map s′* [*0..*<*m*] @ [*s′*
m] (*2* ∗ *n* + *x*)]
by (*auto*) (*metis* (*mono-tags*, *lifting*) *add-cancel-left-left diff-zero length-map*
length-upt
less-antisym nth-append nth-append-length nth-map-upt)

have *eval* ([*ALLEQ*] *A′ q b c d e f r z s*) (*push-list a ks*)
using *all-eq* **unfolding** *all-equations-relation-def* **apply** (*simp add*: *defs*)
unfolding *A q b c d e f*
using *all-equations-invariance*[*of* (!) (*map* (*λP. peval P* (*push-list a ks*)) *r*)
r′
(!) (*map* (*λP. peval P* (*push-list a ks*)) *z*) *z′*
(!) (*map* (*λP. peval P* (*push-list a ks*)) *s*) *s′*
peval A a q′ b′ c′ d′ e′ f′] *r z s*
using *r-list-def s-list-def z-list-def* **by** *auto*

thus *eval DS a*
unfolding *DS-def* **using** ‹*number-of-ex-vars* = *length ks*› **by** (*auto*)
qed

moreover have *is-dioph-rel DS*
unfolding *DS-def*
using *all-equations-dioph* ‹*length r* = *n*› ‹*length z* = *n*› ‹*length s* = *Suc m*›
assms
by (*auto simp*: *dioph*)

ultimately show *?thesis*
  using *is-dioph-rel-def* **by** *auto*

qed

end

end

4.5 Equivalence of register machine and arithmetizing equations

theory *Machine-Equation-Equivalence* **imports** *All-Equations*
../Register-Machine/MachineEquations
../Register-Machine/MultipleToSingleSteps

begin

context *register-machine*
begin

lemma *conclusion-4-5*:
  assumes *is-val*: *is-valid-initial ic p a*
  and *n-def*: *n* ≡ *length* (*snd ic*)
  shows (∃ *q*. *terminates ic p q*) = *rm-equations a*
proof (*rule*)
  assume ∃ *q*. *terminates ic p q*
  then obtain *q*::*nat* **where** *terminates*: *terminates ic p q* **by** *auto*
  hence *q>0* **using** *terminates-def* **by** *auto*

  have ∃ *c>1*. *cells-bounded ic p c*
    using *terminate-c-exists terminates is-val is-valid-initial-def* **by** *blast*
  then obtain *c* **where** *c*: *cells-bounded ic p c* ∧ *c* > *1* **by** *auto*

  define *b* **where** *b* ≡ *B c*
  define *d* **where** *d* ≡ *D q c b*
  define *e* **where** *e* ≡ *E q b*
  define *f* **where** *f* ≡ *F q c b*

  have *c>1* **using** *c* **by** *auto*

  have *b>1* **using** *c b-def B-def*
    using *nat-neq-iff* **by** *fastforce*

  define *r* **where** *r* ≡ *RLe ic p b q*
  define *s* **where** *s* ≡ *SKe ic p b q*
  define *z* **where** *z* ≡ *ZLe ic p b q*

interpret *equations*: *rm-eq-fixes p n a b c d e f q r z s* **by** *unfold-locales*

have *equations.mask-equations*
proof −
have ∀ *l*<*n*. *r l* ⪯ *d*
using *lm04-15-register-masking*[*of ic p c - q*] *r-def n-def d-def b-def c* **by** *auto*
moreover have ∀ *l*<*n*. *z l* ⪯ *e*
using *lm04-15-zero-masking z-def n-def e-def b-def c* **by** *auto*
moreover have ∀ *l*<*n*. *2* ^ *c* ∗ *z l* = *r l* + *d* && *f*
using *lm04-20-zero-definition r-def z-def n-def d-def f-def b-def c* **by** *auto*
ultimately show *?thesis* **unfolding** *equations.mask-equations-def equations.register-mask-def*

equations.zero-indicator-mask-def equations.zero-indicator-0-or-1-def **by** *auto*
qed

moreover have *equations.register-equations*
proof −
have *r 0* = *a* + *b* ∗ *r 0* + *b* ∗ ∑ *R+ p 0 s* − *b* ∗ ∑ *R− p 0* (λ*k*. *s k* && *z 0*)
using *lm04-23-multiple-register1*[*of ic p a c 0 q*] *is-val c terminates* ‹*q*>*0*› *r-def*
s-def z-def b-def bitAND-commutes **by** *auto*
moreover have ∀ *l*>*0*. *l* < *n* ⟶ *r l* = *b* ∗ *r l* + *b* ∗ ∑ *R+ p l s* − *b* ∗ ∑ *R− p l* (λ*k*. *s k* && *z l*)
using *lm04-22-multiple-register*[*of ic p a c - q*]
b-def c terminates r-def s-def z-def is-val bitAND-commutes n-def ‹*q*>*0*› **by** *auto*
moreover have *l*<*n* ⟹ *r l* < *b*^*q* **for** *l*
proof −
assume *l*<*n*
hence *Rlq*: *R ic p l q* = *0*
using *terminates terminates-def correct-halt-def R-def n-def* **by** *auto*
have *c-ineq*: (*2*::*nat*)^*c* ≤ *2* ^ *Suc c* − *Suc 0* **using** ‹*c*>*1*› **by** *auto*
have ∀ *t*. *R ic p l t* < *2* ^ *c* **using** *c* ‹*l*<*n*› *n-def* **by** *auto*
hence *R-bound*: ∀ *t*. *R ic p l t* < *2* ^ *Suc c* − *Suc 0* **using** *c-ineq*
by (*metis dual-order.strict-trans linorder-neqE-nat not-less*)
have (∑ *t* = *0*..*q*. *b* ^ *t* ∗ *R ic p l t*) = (∑ *t* = *0*..(*Suc* (*q*−*1*)). *b* ^ *t* ∗ *R ic p l t*)
using ‹*q*>*0*› **by** *auto*
also have ... = (∑ *t* = *0*..*q*−*1*. *b* ^ *t* ∗ *R ic p l t*) + *b*^*q* ∗ *R ic p l q*
using *Set-Interval.comm-monoid-add-class.sum.atLeast0-atMost-Suc*[*of - q*−*1*] ‹*q*>*0*› **by** *auto*
also have ... = (∑ *t* = *0*..*q*−*1*. *b* ^ *t* ∗ *R ic p l t*) **using** *Rlq* **by** *auto*
also have ... < *b* ^ *q* **using** *b-def R-bound*
base-summation-bound[*of R ic p l c q*−*1*] ‹*q*>*0*› **by** (*auto simp*: *mult.commute*)
finally show *?thesis* **using** *r-def RLe-def* **by** *auto*
qed
ultimately show *?thesis* **unfolding** *equations.register-equations-def equations.register-0-def*

equations.register-l-def equations.register-bound-def **by** *auto*
qed

moreover have *equations.state-equations*
proof –
have *equations.state-relations-from-recursion*
proof –
have $\forall d{>}0.\ d{\le}m \longrightarrow s\ d = b{*}\sum S{+}\ p\ d\ (\lambda k.\ s\ k) + b{*}\sum S{-}\ p\ d\ (\lambda k.\ s\ k$ && $z\ (modifies\ (p!k)))$
$+\ b{*}\sum S0\ p\ d\ (\lambda k.\ s\ k$ && $(e - z\ (modifies\ (p!k))))$
apply (*auto simp*: *s-def z-def*)
using *lm04-24-multiple-step-states*[*of ic p a c - q*]
b-def c terminates s-def z-def is-val bitAND-commutes m-def ‹*q>0*› *e-def E-def* **by** *auto*
moreover have $s\ 0 = 1 + b{*}\sum S{+}\ p\ 0\ (\lambda k.\ s\ k) + b{*}\sum S{-}\ p\ 0\ (\lambda k.\ s\ k$ && $z\ (modifies\ (p!k)))$
$+\ b{*}\sum S0\ p\ 0\ (\lambda k.\ s\ k$ && $(e - z\ (modifies\ (p!k))))$
using *lm04-25-multiple-step-state1*[*of ic p a c - q*]
b-def c terminates s-def z-def is-val bitAND-commutes m-def ‹*q>0*› *e-def E-def* **by** *auto*
ultimately show *?thesis* **unfolding** *equations.state-relations-from-recursion-def*

equations.state-0-def equations.state-d-def equations.state-m-def **by** *auto*
qed

moreover have *equations.state-unique-equations*
proof –
have $k{<}m \longrightarrow s\ k < b\ \hat{}\ q$ **for** *k*
using *state-q-bound is-val terminates* ‹*q>0*› *b-def s-def m-def c* **by** *auto*
moreover have $k{\le}m \longrightarrow s\ k \preceq e$ **for** *k*
using *state-mask is-val terminates* ‹*q>0*› *b-def e-def s-def c* **by** *auto*
ultimately show *?thesis* **unfolding** *equations.state-unique-equations-def*
equations.state-mask-def equations.state-bound-def **by** *auto*
qed

moreover have $\forall M{\le}m.\ sum\ s\ \{..M\} \preceq e$
using *state-sum-mask is-val terminates* ‹*q>0*› *b-def e-def s-def c* ‹*b>1*› *m-def* **by** *auto*

moreover have $s\ m = b\hat{}q$
using *halting-condition-04-27*[*of ic p a q c*] *m-def b-def is-val* ‹*q>0*› *terminates*
s-def **by** *auto*

ultimately show *?thesis* **unfolding** *equations.state-equations-def*
equations.state-partial-sum-mask-def equations.state-m-def **by** *auto*
qed

moreover have *equations.constants-equations*

unfolding *equations.constants-equations-def equations.constant-b-def equations.constant-d-def equations.constant-e-def equations.constant-f-def*
using *b-def d-def e-def f-def* **by** *auto*

moreover have *equations.miscellaneous-equations*
proof −
have *tapelength*: *length* (*snd ic*) > *0*
using *is-val is-valid-initial-def*[*of ic p a*] **by** *auto*
have *R ic p 0 0* = *a* **using** *is-val is-valid-initial-def*[*of ic p a*]
R-def List.hd-conv-nth[*of snd ic*] **by** *auto*
moreover have *R ic p 0 0* < *2^c* **using** *c tapelength* **by** *auto*
ultimately have *a* < *2^c* **by** *auto*
thus *?thesis* **unfolding** *equations.miscellaneous-equations-def equations.c-gt-0-def*

equations.a-bound-def equations.q-gt-0-def
using ‹*q* > *0*› ‹*c* > *1*› **by** *auto*
qed

ultimately show *rm-equations a* **unfolding** *rm-equations-def all-equations-def*
by *blast*
next
assume *rm-equations a*

then obtain *q b c d e f r z s* **where**
reg: *rm-eq-fixes.register-equations p n a b q r z s* **and**
state: *rm-eq-fixes.state-equations p b e q z s* **and**
mask: *rm-eq-fixes.mask-equations n c d e f r z* **and**
const: *rm-eq-fixes.constants-equations b c d e f q* **and**
misc: *rm-eq-fixes.miscellaneous-equations a c q*
unfolding *rm-equations-def all-equations-def* **by** *auto*

have *fx*: *rm-eq-fixes p n*
unfolding *rm-eq-fixes-def* **using** *local.register-machine-axioms* **by** *auto*

have *q>0* **using** *misc fx rm-eq-fixes.miscellaneous-equations-def rm-eq-fixes.q-gt-0-def* **by** *auto*
have *b>1* **using** *B-def const rm-eq-fixes.constants-equations-def rm-eq-fixes.constant-b-def fx*
by (*metis One-nat-def Zero-not-Suc less-one n-not-Suc-n nat-neq-iff nat-power-eq-Suc-0-iff*

numeral-2-eq-2 of-nat-0 of-nat-power-eq-of-nat-cancel-iff of-nat-zero-less-power-iff pos2)
have *n>0* **using** *is-val is-valid-initial-def*[*of ic p a*] *n-def* **by** *auto*
have *m>0* **using** *m-def is-val is-valid-initial-def*[*of ic p*] *is-valid-def*[*of ic p*] **by** *auto*

define *Seq* **where** *Seq* ≡ (λ*k t*. *nth-digit* (*s k*) *t b*)
define *Req* **where** *Req* ≡ (λ*l t*. *nth-digit* (*r l*) *t b*)
define *Zeq* **where** *Zeq* ≡ (λ*l t*. *nth-digit* (*z l*) *t b*)

have *mask-old*: *mask-equations n r z c d e f* **and**
reg-old: *reg-equations p r z s b a* (*length* (*snd ic*)) *q* **and**
state-old: *state-equations p s z b e q* (*length p* − *1*) **and**
const-old: *rm-constants q c b d e f a*
subgoal
using *mask rm-eq-fixes.mask-equations-def rm-eq-fixes.register-mask-def fx mask-equations-def rm-eq-fixes.zero-indicator-0-or-1-def rm-eq-fixes.zero-indicator-mask-def*
by *simp*
subgoal
using *reg state mask const misc* **using** *rm-eq-fixes.register-equations-def rm-eq-fixes.register-0-def rm-eq-fixes.register-l-def rm-eq-fixes.register-bound-def reg-equations-def n-def fx* **by** *simp*
subgoal
using *state fx state-equations-def rm-eq-fixes.state-equations-def rm-eq-fixes.state-relations-from-recursion-def rm-eq-fixes.state-0-def rm-eq-fixes.state-m-def rm-eq-fixes.state-d-def rm-eq-fixes.state-unique-equations-def rm-eq-fixes.state-mask-def rm-eq-fixes.state-bound-def rm-eq-fixes.state-partial-sum-mask-def m-def* **by** *simp*
subgoal unfolding *rm-constants-def*
using *const misc fx rm-eq-fixes.constants-equations-def rm-eq-fixes.miscellaneous-equations-def rm-eq-fixes.constant-b-def rm-eq-fixes.constant-d-def rm-eq-fixes.constant-e-def rm-eq-fixes.constant-f-def rm-eq-fixes.c-gt-0-def rm-eq-fixes.q-gt-0-def rm-eq-fixes.a-bound-def* **by** *simp*
done

hence *RZS-eq*: $l<n \Longrightarrow j\leq m \Longrightarrow t\leq q \Longrightarrow$
$R\ ic\ p\ l\ t = Req\ l\ t \wedge Z\ ic\ p\ l\ t = Zeq\ l\ t \wedge S\ ic\ p\ j\ t = Seq\ j\ t$ **for** *l j t*
using *rzs-eq*[*of m p n ic a r z*] *mask-old reg-old state-old const-old m-def n-def is-val* ‹*q*>*0*› *Seq-def Req-def Zeq-def* **by** *auto*

have *R-eq*: $l<n \Longrightarrow t\leq q \Longrightarrow R\ ic\ p\ l\ t = Req\ l\ t$ **for** *l t* **using** *RZS-eq* **by** *auto*
have *Z-eq*: $l<n \Longrightarrow t\leq q \Longrightarrow Z\ ic\ p\ l\ t = Zeq\ l\ t$ **for** *l t* **using** *RZS-eq* **by** *auto*
have *S-eq*: $j\leq m \Longrightarrow t\leq q \Longrightarrow S\ ic\ p\ j\ t = Seq\ j\ t$ **for** *j t* **using** *RZS-eq*[*of 0*] ‹*n*>*0*› **by** *auto*

have *ishalt* (*p*!*m*) **using** *m-def is-val is-valid-initial-def*[*of ic p a*] *is-valid-def*[*of ic p*] **by** *auto*
have *Seq m q* = *1* **using** *state nth-digit-def Seq-def* ‹*b*>*1*›
using *fx rm-eq-fixes.state-equations-def rm-eq-fixes.state-relations-from-recursion-def rm-eq-fixes.state-m-def* **by** *auto*
hence *S ic p m q* = *1* **using** *S-eq* **by** *auto*
hence *fst* (*steps ic p q*) = *m* **using** *S-def* **by**(*cases fst* (*steps ic p q*) = *m*; *auto*)
hence *qhalt*: *ishalt* (*p* ! (*fst* (*steps ic p q*))) **using** *S-def* ‹*ishalt* (*p*!*m*)› **by** *auto*

hence *rempty*: *snd* (*steps ic p q*) ! *l* = *0* **if** *l* < *n* **for** *l*
unfolding *R-def*[*symmetric*]

```
    using R-eq[of l q] ‹l < n› apply auto
    using reg Req-def nth-digit-def
    using rm-eq-fixes.register-equations-def
          rm-eq-fixes.register-l-def
          rm-eq-fixes.register-0-def
          rm-eq-fixes.register-bound-def
    by auto (simp add: fx)

  have state-m-0: t<q ⟹ S ic p m t = 0 for t
    proof –
      assume t<q
      have b ^ q div b ^ t = b^(q−t)
        by (metis ‹1 < b› ‹t < q› less-imp-le not-one-le-zero power-diff)
      also have ... mod b = 0 using ‹1 < b› ‹t < q› by simp
      finally have mod: b^q div b^t mod b = 0 by auto
      have s m = b^q using state fx rm-eq-fixes.state-equations-def
        rm-eq-fixes.state-m-def
        rm-eq-fixes.state-relations-from-recursion-def by auto
      hence Seq m t = 0 using Seq-def nth-digit-def mod by auto
      with S-eq ‹t < q› show ?thesis by auto
    qed
  have ∀ k<m. ¬ ishalt (p!k)
    using is-val is-valid-initial-def[of ic p a] is-valid-def[of ic p] m-def by auto
  moreover have t<q ⟶ fst (steps ic p t) < length p − 1 for t
    proof (rule ccontr)
      assume asm: ¬ (t < q ⟶ fst (steps ic p t) < length p − 1)
      hence t<q by auto
      with asm have fst (steps ic p t) ≥ length p − 1 by auto
      moreover have fst (steps ic p t) ≤ length p − 1
        using p-contains[of ic p a t] is-val by auto
      ultimately have fst (steps ic p t) = m using m-def by auto
      hence S ic p m t = 1 using S-def by auto
      thus False using state-m-0[of t] ‹t<q› by auto
    qed
  ultimately have t<q ⟶ ¬ ishalt (p ! (fst (steps ic p t))) for t using m-def
by auto
  hence no-early-halt: t<q ⟶ ¬ ishalt (p ! (fst (steps ic p t))) for t using
state-m-0 by auto

  have correct-halt ic p q using qhalt rempty correct-halt-def n-def by auto
  thus (∃ q. terminates ic p q) using no-early-halt terminates-def ‹q>0› by auto
qed

end

end
```

5 Proof of the DPRM theorem

theory *DPRM*
  imports *Machine-Equations/Machine-Equation-Equivalence*
begin

definition *is-recenum* :: *nat set* ⇒ *bool* **where**
  is-recenum A =
    (∃ *p* :: *program*.
     ∃ *n* :: *nat*.
     ∀ *a* :: *nat*. ∃ *ic*. *ic* = *initial-config n a* ∧ *is-valid-initial ic p a* ∧
    (*a* ∈ *A*) = (∃ *q*::*nat*. *terminates ic p q*))

theorem *DPRM*: *is-recenum A* ⟹ *is-dioph-set A*
proof −
  assume *is-recenum A*
  hence (∃ *p* :: *program*.
        ∃ *n* :: *nat*. ∀ *a* :: *nat*.
        ∃ *ic*. *ic* = *initial-config n a* ∧ *is-valid-initial ic p a* ∧
          (*a* ∈ *A*) = (∃ *q*::*nat*. *terminates ic p q*)) **using** *is-recenum-def* **by** *auto*
  then obtain *p n* **where** *p*:
       ∀ *a* :: *nat*.
        ∃ *ic*. *ic* = *initial-config n a* ∧ *is-valid-initial ic p a* ∧
          (*a* ∈ *A*) = (∃ *q*::*nat*. *terminates ic p q*) **by** *auto*

  interpret *rm*: *register-machine p Suc n* **apply** (*unfold-locales*)
  proof −
    from *p* **have**
      ∃ *ic*. *ic* = *initial-config n 0* ∧ *is-valid-initial ic p 0* ∧
          (*0* ∈ *A*) = (∃ *q*::*nat*. *terminates ic p q*) **by** *auto*
    then obtain *ic* **where** *ic*: *ic* = *initial-config n 0* **and** *is-val*: *is-valid-initial ic p 0* **by** *auto*

    show *length p* > *0*
      using *is-val* **unfolding** *is-valid-initial-def is-valid-def* **by** *auto*

    have *length* (*snd ic*) = *Suc n*
      unfolding *ic initial-config-def* **by** *auto*

    moreover have *snd ic* ≠ []
      using *is-val* **unfolding** *is-valid-initial-def is-valid-def tape-check-initial.simps* **by** *auto*

    ultimately show *Suc n* > *0*
      by *auto*

    show *program-register-check p* (*Suc n*)
      using *is-val* **unfolding** *is-valid-initial-def is-valid-def* **using** ‹*length* (*snd ic*) = *Suc n*›

by *auto*
qed

have *equiv*: $a \in A \longleftrightarrow$ *register-machine.rm-equations p* (*Suc n*) *a* **for** *a*
proof −
from *p* **have** $\exists$ *ic. ic* = *initial-config n a* $\wedge$ *is-valid-initial ic p a* $\wedge$
($a \in A$) = ($\exists$ *q*::*nat. terminates ic p q*) **by** *auto*
then obtain *ic* **where** *ic*: *ic* = *initial-config n a* $\wedge$ *is-valid-initial ic p a* $\wedge$
($a \in A$) = ($\exists$ *q*::*nat. terminates ic p q*) **by** *blast*

have *ic-def*: *ic* = *initial-config n a* **using** *ic* **by** *auto*
hence *n-is-length*: *Suc n* = *length* (*snd ic*) **using** *initial-config-def*[*of n a*] **by** *auto*
have *is-val*: *is-valid-initial ic p a* **using** *ic* **by** *auto*
have ($a \in A$) = ($\exists$ *q. terminates ic p q*) **using** *ic* **by** *auto*
moreover have ($\exists$ *q. terminates ic p q*) = *register-machine.rm-equations p* (*Suc n*) *a*
using *is-val n-is-length rm.conclusion-4-5*
by *auto*

ultimately show *?thesis* **by** *auto*
qed

hence *A-characterization*: *A* = {*a*::*nat. register-machine.rm-equations p* (*Suc n*) *a*} **by** *auto*

have *eq-dioph*: $\exists P_1\ P_2.\ \forall a.$ *register-machine.rm-equations p* (*Suc n*) (*peval A a*)
= ($\exists$ *v. ppeval* P_1 *a v* = *ppeval* P_2 *a v*) **for** *A*
using *rm.rm-dioph*[*of A*] **using** *is-dioph-rel-def*[*of rm.rm-equations-relation A*]

unfolding *rm.rm-equations-relation-def* **by**(*auto simp*: *unary-eval*)

have $\exists P_1\ P_2.\ \forall b.$ *register-machine.rm-equations p* (*Suc n*) (*peval* (*Param 0*) ($\lambda x.\ b$))
= ($\exists$ *v. ppeval* P_1 ($\lambda x.\ b$) *v* = *ppeval* P_2 ($\lambda x.\ b$) *v*)
using *eq-dioph*[*of Param 0*] **by** *blast*

hence $\exists$ *P1 P2.* $\forall$ *a. register-machine.rm-equations p* (*Suc n*) *a*
= ($\exists$ *v. ppeval P1* ($\lambda x.\ a$) *v* = *ppeval P2* ($\lambda x.\ a$) *v*)
by *auto*

thus *?thesis*
unfolding *A-characterization is-dioph-set-def* **by** *simp*
qed

end

References

[1] J. Bayer, M. David, A. Pal, and B. Stock. Beginners' quest to formalize mathematics: A feasibility study in Isabelle. In C. Kaliszyk, E. Brady, A. Kohlhase, and C. Sacerdoti Coen, editors, *Intelligent Computer Mathematics*, pages 16–27, Cham, 2019. Springer International Publishing.

[2] 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 fuer Informatik.

[3] M. Carneiro. A Lean formalization of Matiyasevič's theorem. https://arxiv.org/abs/1802.01795v1, 02 2018.

[4] D. Larchey-Wendling and Y. Forster. Hilbert's Tenth Problem in Coq. In H. Geuvers, editor, *4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019)*, volume 131 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 27:1–27:20, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[5] Y. Matiyasevich. On Hilbert's tenth problem. In M. Lamoureux, editor, *PIMS Distinguished Chair Lectures*, volume 1. Pacific Institute for the Mathematical Sciences, 2000.

[6] K. Pak. Diophantine sets. preliminaries. *Formalized Mathematics*, 26(1):81–90, 2018.

[7] K. Pak. The Matiyasevich theorem. preliminaries. *Formalized Mathematics*, 25(4):315–322, 2018.

[8] J. Xu, X. Zhang, and C. Urban. Mechanising Turing machines and computability theory in Isabelle/HOL. In S. Blazy, C. Paulin-Mohring, and D. Pichardie, editors, *Interactive Theorem Proving. ITP 2013.*, volume 7998 of *Lecture Notes in Computer Science*, pages 147–162. Springer Berlin, Heidelberg, 2013.

[9] J. Xu, X. Zhang, C. Urban, and S. J. C. Joosten. Universal Turing machine. *Archive of Formal Proofs*, Feb. 2019. https://isa-afp.org/entries/Universal_Turing_Machine.html, Formal proof development.