Analysis of List Update Algorithms

Maximilian P.L. Haslbeck and Tobias Nipkow

August 16, 2018

Abstract

These theories formalize the quantitative analysis of a number of classical algorithms for the list update problem: 2-competitiveness of move-to-front, the lower bound of 2 for the competitiveness of deterministic list update algorithms and 1.6-competitiveness of the randomized COMB algorithm, the best randomized list update algorithm known to date.

An informal description is found in an accompanying report [HN16]. The material is based on the first two chapters of the book by Borodin and El-Yaniv [BEY98].

Contents

1 List Inversion 4
2 Swapping Adjacent Elements in a List 5
3 Deterministic Online and Offline Algorithms 8
4 Probability Theory 12
  4.1 function $E$  12
  4.2 function $bv$  14
  4.3 function $flip$  17
  4.4 Example for pmf  21
  4.5 Sum Distribution  21
5 Randomized Online and Offline Algorithms 26
  5.1 Competitive Analysis Formalized  26
  5.2 embedding of deterministic into randomized algorithms  30
6 Deterministic List Update 31
  6.1 Function $mtf$  31
  6.2 Function $mtf2$  32
  6.3 Function $Lxy$  33
  6.4 List Update as Online/Offline Algorithm  36
15.5 TS is pairwise .................................................. 277
15.6 TS is 2-compet .................................................. 317

16 BIT is pairwise ................................................. 319

17 BIT is 1.75 competitive on lists of length 2 ................. 335
  17.1 auxiliary lemmas ............................................ 335
  17.2 Analysis of the four phase forms ......................... 342
  17.3 Phase Partitioning .......................................... 361

18 COMB .......................................................... 362
  18.1 Definition of COMB ......................................... 362
  18.2 Comb 1.6-competitive on 2 elements ...................... 363
  18.3 COMB pairwise .............................................. 370
  18.4 COMB 1.6-competitive ...................................... 372
1 List Inversion

theory Inversion
imports List – Index.List_Index
begin

abbreviation dist_perm xs ys ≡ distinct xs ∧ distinct ys ∧ set xs = set ys

definition before_in :: 'a ⇒ 'a ⇒ 'a list ⇒ bool
  ((_, _/ _) in _) [55,55,55] 55) where
x < y in xs = (index xs x < index xs y ∧ y ∈ set xs)

definition Inv :: 'a list ⇒ 'a list ⇒ ('a × 'a) set where
Inv xs ys = {(x,y). x < y in xs ∧ y < x in ys}

lemma before_in_setD1: x < y in xs → x : set xs
by (metis index_conv_size_if_notin index_less before_in_def less_asym order_refl)

lemma before_in_setD2: x < y in xs → y : set xs
by (simp add: before_in_def)

lemma not_before_in:
  x : set xs → y : set xs → ¬ x < y in xs ↔ y < x in xs ∨ x = y
by (metis index_eq_index_conv before_in_def less_asym linorder_neqE_nat)

lemma before_in_irefl: x < x in xs = False
by (meson before_in_setD2 not_before_in)

lemma no_before_in[simp]: x < y in xs → (∼ y < x in xs) = True
by (metis before_in_setD1 not_before_in)

lemma finite.Inv[simp]: finite(Inv xs ys)
apply(rule finite_subset[where B = set xs × set ys])
apply(auto simp add: Inv_def before_in_def)
apply(metis index_conv_size_if_notin index_less_size_conv less_asym)+
done

lemma Inv_id[simp]: Inv xs xs = {}
by(auto simp add: Inv_def before_in_def)

lemma card.Inv_sym: card(Inv xs ys) = card(Inv ys xs)
proof –
  have Inv xs ys = (λ(x,y). (y,x)) ' Inv ys xs by(auto simp: Inv_def)
thus ?thesis by (metis card_image swap_inj_on)
qed

lemma Inv_tri_ineq:
  dist_perm xs ys ⇒ dist_perm ys zs ⇒
  Inv xs zs ⊆ Inv xs ys Un Inv ys zs
by (auto simp: Inv_def) (metis before_in_setD1 not_before_in)

lemma card_Inv_tri_ineq:
  dist_perm xs ys ⇒ dist_perm ys zs ⇒
  card (Inv xs zs) ≤ card (Inv xs ys) + card (Inv ys zs)
using card_mono[OF Inv_tri_ineq[of xs ys zs]]
by auto (metis card_Un_Int finite_Inv trans_le_add1)
end

2 Swapping Adjacent Elements in a List

theory Swaps
imports Inversion
begin

Swap elements at index n and Suc n:

definition swap n xs =
  (if Suc n < size xs then xs[n := xs!Suc n, Suc n := xs!n] else xs)

lemma length_swap[simp]: length (swap i xs) = length xs
by (simp add: swap_def)

lemma swap_id[simp]: Suc n ≥ size xs ⇒ swap n xs = xs
by (simp add: swap_def)

lemma distinct_swap[simp]:
  distinct (swap i xs) = distinct xs
by (simp add: swap_def)

lemma swap_Suc[simp]: swap (Suc n) (a # xs) = a # swap n xs
by (induction xs) (auto simp: swap_def)

lemma index_swap_distinct:
  distinct xs ⇒ Suc n < length xs ⇒
  index (swap n xs) x =
  (if x = xs!n then Suc n else if x = xs!Suc n then n else index xs x)
by (auto simp add: swap_def index_swap_if_distinct)
lemma set_swap[simp]: set(swaps n xs) = set xs
by(auto simp add: swap_def set_conv_nth nth_list_update) metis

lemma nth_swap_id[simp]: Suc i < length xs ⟹ swap i xs ! i = xs!(i+1)
by(simp add: swap_def)

lemma before_in_swap:
  dist_perm xs ys ⟹ Suc n < size xs ⟹
    x < y in (swap n xs) ⟷
    x < y in xs ∧ ¬(x = xs!n ∧ y = xs!Suc n) ∨ x = xs!Suc n ∧ y = xs!n
by(simp add:before_in_def index_swap_distinct)
  (metis Suc_lessD Suc_lessI index_less_size_conv index_nth_id less_Suc_eq
   n_not_Suc_n nth_index)

lemma Inv_swap: assumes dist_perm xs ys
shows Inv xs (swap n ys) =
  (if Suc n < size xs
    then if ys!n < ys!Suc n in xs
      then Inv xs ys ∪ {(ys!n, ys!Suc n)}
      else Inv xs ys − {(ys!Suc n, ys!n)}
    else Inv xs ys)
proof
  have length xs = length ys using assms by (metis distinct_card)
  with assms show ?thesis
    by(simp add: Inv_def set_eq_iff)
    (metis before_in_def not_before_in before_swap)
qed

Perform a list of swaps, from right to left:

abbreviation swaps where swaps == foldr swap

lemma swaps_inv[simp]:
  set (swaps sws xs) = set xs ∧
  size(swaps sws xs) = size xs ∧
  distinct(swaps sws xs) = distinct xs
by (induct sws arbitrary: xs) (simp_all add: swap_def)

lemma swaps_eq Nil_iff[simp]: swaps acts xs = [] ⟷ xs = []
by(induction acts)(auto simp: swap_def)

lemma swaps_map Suc[simp]:
  swaps (map Suc sws) (a # xs) = a # swaps sws xs
by(induction sus arbitrary: xs) auto
lemma card_Inv_swaps_le:
  distinct xs ⇒ card (Inv xs (swaps sws xs)) ≤ length sws
by (induction sws) (auto simp: Inv_swap card_insert_if card_Diff_singleton_if)

lemma nth_swaps: ∀ i ∈ set is. j < i ⇒ swaps is xs ! j = xs ! j
by (induction is) (simp_all add: swap_def)

lemma not_before0[simp]: ¬ x < xs ! 0 in xs
apply (cases xs = [])
by (auto simp: before_def neq_Nil_conv)

lemma before_id[simp]: [ distinct xs; i < size xs; j < size xs ] ⇒
  xs ! i < xs ! j in xs ↔ i < j
by (simp add: before_def index_nth_id)

lemma before_swaps:
  [ distinct is; ∀ i ∈ set is. Suc i < size xs; distinct xs; i /∈ set is; i < j; j < size xs ] ⇒
  swaps is xs ! i < swaps is xs ! j in xs
apply (induction is arbitrary: i j)
apply simp
apply (auto simp: swap_def nth_list_update)
done

lemma card_Inv_swaps:
  [ distinct is; ∀ i ∈ set is. Suc i < size xs; distinct xs ] ⇒
card (Inv xs (swaps is xs)) = length is
apply (induction is)
apply simp
apply (simp add: Inv_swap before_swaps card_insert_if)
apply (simp add: Inv_def)
done

lemma swaps_eq_nth_take_drop: i < length xs ⇒
  swaps [0..<i] xs = xs!i # take i xs @ drop (Suc i) xs
apply (induction i arbitrary: xs)
apply (auto simp add: neq_Nil_conv swap_def drop_update_swap
take_Suc_conv_app_nth Cons_nth_drop_Suc[symmetric])
done

lemma index_swaps_size: distinct s ⇒
  index s q ≤ index (swaps sws s) q + length sws
apply (induction sws arbitrary: s)
apply simp
apply (fastforce simp: swap_def index_swap_if_distinct index_nth_id)
done

lemma index_swaps_last_size: distinct s \implies
  size s \leq index \text{ (swaps sws s) \ (last s)} + \text{length sws} + 1
apply (cases s = [])
apply simp
using index_swaps_last_size[of s last s sws] by simp

end

3 Deterministic Online and Offline Algorithms

theory On_Off
imports Complex_Main
begin

   type_synonym ('s,'r,'a) \text{alg\_off} = 's \Rightarrow 'r list \Rightarrow 'a list
   type_synonym ('s,'is,'r,'a) \text{alg\_on} = ('s \Rightarrow 'is) * ('s * 'is \Rightarrow 'r \Rightarrow 'a * 'is)

locale On_Off =
  fixes step :: 'state \Rightarrow 'request \Rightarrow 'answer \Rightarrow 'state
  fixes t :: 'state \Rightarrow 'request \Rightarrow 'answer \Rightarrow nat
  fixes wf :: 'state \Rightarrow 'request list \Rightarrow bool
begin

  fun T :: 'state \Rightarrow 'request list \Rightarrow 'answer list \Rightarrow nat where
  T s [] [] = 0 |
  T s (r#rs) (a#as) = t s r a + T (step s r a) rs as

  definition Step ::
  (\text{'state , 'istate, 'request, 'answer})\text{alg\_on}
  \Rightarrow \text{'state * 'istate \Rightarrow 'request \Rightarrow 'state * 'istate}
  where
  Step A s r = (let (a,is') = snd A s r in (step (fst s) r a, is'))

  fun config' :: ('state,'is,'request,'answer) \text{alg\_on} \Rightarrow ('state*is) \Rightarrow 'request list
  \Rightarrow ('state * 'is) where
  config' A s [] = s |
  config' A s (r#rs) = config' A (Step A s r) rs

end
lemma config\'_snoc: config\' A s (rs@[r]) = Step A (config\' A s rs) r
apply(induct rs arbitrary: s) by simp_all

lemma config\'_append2: config\' A s (xs@ys) = config\' A (config\' A s xs) ys
apply(induct xs arbitrary: s) by simp_all

lemma config\'_induct: P (fst init) \implies (\forall s q a. P s \implies P (step s q a))
\implies P (fst (config\' A init rs))
apply (induct rs arbitrary: init) by(simp_all add: Step_def split: prod.split)

abbreviation config where
config A s0 rs == config\' A (s0, fst A s0) rs

lemma config\'_snoc: config A s (rs@[r]) = Step A (config A s rs) r
using config\'_snoc by metis

lemma config\'_append: config A s (xs@ys) = config\' A (config A s xs) ys
using config\'_append2 by metis

lemma config\'_induct: P s0 \implies (\forall s q a. P s \implies P (step s q a)) \implies P
(fst (config A s0 qs))
using config\'_induct[of P (s0, fst A s0)] by auto

fun T_on' :: ('state,'is,'request,'answer) alg_on \Rightarrow ('state*'is) \Rightarrow 'request
list \Rightarrow nat where
T_on' A s [] = 0 | T_on' A s (r\#rs) = (t (fst s) r (fst (snd A s r))) + T_on' A (Step A s r) rs

lemma T_on'append: T_on' A s (xs@ys) = T_on' A s xs + T_on' A (config A s xs) ys
apply(induct xs arbitrary: s) by simp_all

abbreviation T_on'':: ('state,'is,'request,'answer) alg_on \Rightarrow 'state \Rightarrow 'request
list \Rightarrow nat where
T_on'' A s rs == T_on' A (s,fst A s) rs

lemma T_on''append: T_on'' A s (xs@ys) = T_on'' A s xs + T_on'' A (config A s xs) ys
by(rule T_on'append)
abbreviation \( T_{\text{on}_n} \) \((A \ s0 \ xs \ n) \equiv T_{\text{on}'} A \ (\text{config} \ A \ s0 \ (\text{take} \ n \ xs)) \) [xs!n]

lemma \( T_{\text{on}_{as\_sum}} \): \( T_{\text{on}''} \) \((A \ s0 \ rs) \equiv \text{sum} \ (T_{\text{on}_n} A \ s0 \ rs) \{..<\text{length} \ rs\} \)
apply (induct \( rs \) rule: \( \text{rev\_induct} \))
by (simp_all add: \( T_{\text{on}'} \_\text{append} \ \text{nth\_append} \))

fun \( \text{off2} \) :: ('state,'is,'request,'answer) \( \text{alg\_on} \Rightarrow ('\text{state} * 'is,'request,'answer) \)
\( \text{alg\_off} \) where
\( \text{off2} \ A \ s \ [\] = [] | \)
\( \text{off2} \ A \ s \ (r##rs) = \text{fst} \ (\text{snd} \ A \ s \ r) \ # \ \text{off2} \ A \ (\text{Step} \ A \ s \ r) \ rs \)

abbreviation \( \text{off} \) :: ('state,'is,'request,'answer) \( \text{alg\_on} \Rightarrow ('\text{state},'request,'answer) \)
\( \text{alg\_off} \) where
\( \text{off} \ A \ s0 \equiv \text{off2} \ A \ (s0, \ \text{fst} \ A \ s0) \)

abbreviation \( T_{\text{off}} \) :: ('state,'request,'answer) \( \text{alg\_off} \Rightarrow \) 'state \Rightarrow 'request
\( \text{list} \Rightarrow \text{nat} \) where
\( T_{\text{off}} \ A \ s0 \ rs \equiv T s0 \ rs \ (A \ s0 \ rs) \)

abbreviation \( T_{\text{on}} \) :: ('state,'is,'request,'answer) \( \text{alg\_on} \Rightarrow '\text{state} \Rightarrow '\text{request}
\( \text{list} \Rightarrow \text{nat} \) where
\( T_{\text{on}} \ A \equiv T_{\text{off}} \ (\text{off} \ A) \)

lemma \( T_{\text{on}_{on}'} \): \( T_{\text{off}} \ (\lambda s0. \ (\text{off2} \ A \ (s0, \ x))) \) \( s0 \ qs \ = T_{\text{on}'} A \ (s0, x) \ qs \)
apply (induct \( qs \) arbitrary: \( s0 \ x \))
by (simp_all add: \( \text{Step\_def} \ \text{split} : \text{prod\_split} \))

lemma \( T_{\text{on}_{on}''} \): \( T_{\text{on}} \ A \ s0 \ qs \ = T_{\text{on}''} A \ s0 \ qs \)
using \( T_{\text{on}_{on}'}[\text{where} \ x=\text{fst} \ A \ s0, \ \text{of} \ s0 \ qs \ A] \) by (auto)

lemma \( T_{\text{on}_{as\_sum}} \): \( T_{\text{on}} \ A \ s0 \ rs \ = \text{sum} \ (T_{\text{on}_n} A \ s0 \ rs) \{..<\text{length} \ rs\} \)
using \( T_{\text{on}_{as\_sum}} \) \( T_{\text{on}_{on}''} \) by metis

10
definition \( T_{\text{opt}} :: '\text{state} \Rightarrow '\text{request list} \Rightarrow \text{nat} \) where
\[
T_{\text{opt}} \ s \ rs = \text{Inf} \ \{ T \ s \ rs \ | \ \text{as. size as = size rs} \}
\]
definition \( \text{compt} :: (\text{state,}'\text{is,}'\text{request,}'\text{answer}) \ \text{alg on} \Rightarrow \text{real} \Rightarrow '\text{state set} \\
\Rightarrow \text{bool} \) where
\[
\text{compt} \ A \ c \ S = (\forall s \in S. \exists \ b \geq 0. \forall rs. \ \text{wf} s \ rs \rightarrow \ \text{real}(T\_\text{on} A \ s \ rs) \leq c \ast T_{\text{opt}} s \ rs + b)
\]
lemma \( \text{length}_\text{off}[\text{simp}]: \ \text{length}(\text{off2} A \ s \ rs) = \text{length} \ rs \) by (induction \( rs \) arbitrary: \( s \)) (auto split: prod.split)

lemma \( \text{compt}_\text{mono} \): assumes \( \text{compt} \ A \ c \ S_0 \) and \( c \leq c' \) shows \( \text{compt} \ A \ c' \ S_0 \) proof (unfold \( \text{compt}_\text{def} \), auto)
  let \( ?\text{compt} = \lambda s0 \ rs \ b \ (c::\text{real}). \ T\_\text{on} A \ s0 \ rs \leq c \ast T_{\text{opt}} s0 \ rs + b \)
  fix \( s0 \) assume \( s0 \in S_0 \)
  with \( \text{assms}(1) \) obtain \( b \) where \( b \geq 0 \) and \( 1: \forall \ rs. \ \text{wf} s0 \ rs \rightarrow ?\text{compt} s0 \ rs \ b \ c' \)
    by (auto simp: \( \text{compt}_\text{def} \))
  have \( \forall \ rs. \ \text{wf} s0 \ rs \rightarrow ?\text{compt} s0 \ rs \ b \ c' \)
    proof safe
      fix \( rs \)
      assume \( \text{wf}: \ \text{wf} s0 \ rs \)
      from \( 1 \ \text{wf} \) have \( ?\text{compt} s0 \ rs \ b \ c' \) by blast
      thus \( ?\text{compt} s0 \ rs \ b \ c' \)
        using \( 1 \ \text{mult\_right\_mono}[\text{OF} \ \text{assms}(2) \ \text{of\_nat\_0\_le\_iff}[\text{of} \ T_{\text{opt}} s0 \ rs]] \)
          by arith
    qed
  thus \( \exists b \geq 0. \forall \ rs. \ \text{wf} s0 \ rs \rightarrow ?\text{compt} s0 \ rs \ b \ c' \) using \( b \geq 0 \) by (auto)
  qed

lemma \( \text{comptE} \): fixes \( c :: \text{real} \)
assumes \( \text{compt} \ A \ c \ S_0 \ c \geq 0 \ \forall s0 \ rs. \ \text{size}(\text{aoff} s0 \ rs) = \text{length} \ rs \ s0 \in S_0 \)
shows \( \exists b \geq 0. \forall \ rs. \ \text{wf} s0 \ rs \rightarrow T\_\text{on} A \ s0 \ rs \leq c \ast T\_\text{off} \ \text{aoff} s0 \ rs + b \)
proof –
  from \( \text{assms}(1,4) \) obtain \( b \) where \( b \geq 0 \) and
    1: \( \forall \ rs. \ \text{wf} s0 \ rs \rightarrow T\_\text{on} A \ s0 \ rs \leq c \ast T_{\text{opt}} s0 \ rs + b \)
    by (auto simp add: \( \text{compt}_\text{def} \))
  { fix \( rs \)
    assume \( \text{wf} s0 \ rs \)
    then have \( 2: \ \text{real}(T\_\text{on} A \ s0 \ rs) \leq c \ast \text{Inf} \ \{ T \ s0 \ rs \ | \ \text{as. size as = size rs} \} + b \)
(is _ ≤ _ * real(Inf ?T) + _)
using 1 by (auto simp add: T_opt_def)
have Inf ?T ≤ T_off aoff s0 rs
  using assms(3) by (intro cInf_lower) auto
from mult_left_mono[OF of_nat_le_iff[THEN iffD2, OF this] assms(2)]
have T_on A s0 rs ≤ c * T_off aoff s0 rs + b using 2 by arith
}
thus ?thesis using (b≥0) by (auto simp: compet_def)
qed
end

4 Probability Theory

theory Prob_Theory
imports HOL—Probability.Probability
begin

lemma integral_map_pmf[simp]:
  fixes f :: real ⇒ real
  shows (∫ x. f x ∂ (map_pmf g M)) = (∫ x. f (g x) ∂ M)
  unfolding map_pmf_rep_eq
  using integral_distr[of g (measure_pmf M) (count_space UNIV) f] by auto

4.1 function E

definition E :: real pmf ⇒ real
  where
  E M = (∫ x. x ∂ measure_pmf M)

translations
∫ x. f ∂ M <= CONST lebesgue_integral M (λx. f)

notation (latex output) E (E[ ] [1] 100)

lemma E_const[simp]: E (return_pmf a) = a
  unfolding E_def
unfolding return_pmf_rep_eq
by (simp add: integral_return)

lemma E_null[simp]: E (return_pmf 0) = 0
by auto
lemma $E_{\text{finite \_sum}}$: finite $(\text{set \_pmf \ X}) \implies E \ X = (\sum x \in (\text{set \_pmf \ X}). \ \text{pmf \ X \ x} \times x)$. 
unfolding $E_{\text{def}}$ by $(\text{subst integral \_measure \_pmf})$ simp_all

lemma $E_{\text{of \_const}}$: $E(\text{map \_pmf} \ (\lambda x. y) \ (X::\text{real \ \text{pmf} })) = y$ by auto

lemma $E_{\text{nonneg}}$: 
  shows $(\forall x \in \text{set \_pmf \ X}. \ 0 \leq x) \implies 0 \leq E \ X$
unfolding $E_{\text{def}}$
using integral_nonneg by $(\text{simp add: AE\_measure\_pmf\_iff integral\_nonneg\_AE})$

lemma $E_{\text{nonneg\_fun}}$: fixes $f :: 'a \Rightarrow \text{real}$
  shows $(\forall x \in \text{set \_pmf \ X}. \ 0 \leq f x) \implies 0 \leq E \ (\text{map \_pmf \ f \ X})$
using $E_{\text{nonneg \ by \ auto}}$

lemma $E_{\text{cong}}$: 
  fixes $f :: 'a \Rightarrow \text{real}$
  shows finite $(\text{set \_pmf \ X}) \implies (\forall x \in \text{set \_pmf \ X}. \ (f x) = (u x)) \implies E \ (\text{map \_pmf \ f \ X}) = E \ (\text{map \_pmf \ u \ X})$
unfolding $E_{\text{def}}$ integral_map_pmf apply(rule integral_cong_AE)
apply(simp add: integrable_measure_pmf_finite)+
by $(\text{simp add: AE\_measure\_pmf\_iff})$

lemma $E_{\text{mono3}}$: 
  fixes $f :: 'a \Rightarrow \text{real}$
  shows integrable $(\text{measure \_pmf \ X}) \ f \implies \text{integrable} \ (\text{measure \_pmf \ X}) \ u \implies (\forall x \in \text{set \_pmf \ X}. \ (f x) \leq (u x)) \implies E \ (\text{map \_pmf \ f \ X}) \leq E \ (\text{map \_pmf \ u \ X})$
unfolding $E_{\text{def}}$ integral_map_pmf apply(rule integral_mono_AE)
by $(\text{auto simp add: AE\_measure\_pmf\_iff})$

lemma $E_{\text{mono2}}$: 
  fixes $f :: 'a \Rightarrow \text{real}$
  shows finite $(\text{set \_pmf \ X}) \implies (\forall x \in \text{set \_pmf \ X}. \ (f x) \leq (u x)) \implies E \ (\text{map \_pmf \ f \ X}) \leq E \ (\text{map \_pmf \ u \ X})$
unfolding $E_{\text{def}}$ integral_map_pmf apply(rule integral_mono_AE)
apply(simp add: integrable_measure_pmf_finite)+
by $(\text{simp add: AE\_measure\_pmf\_iff})$

lemma $E_{\text{linear\_diff2}}$: finite $(\text{set \_pmf \ A}) \implies E \ (\text{map \_pmf \ f \ A}) - E \ (\text{map \_pmf \ g \ A}) = E \ (\text{map \_pmf} \ (\lambda x. \ (f x) - (g x)) \ A)$
unfolding $E_{\text{def}}$ integral_map_pmf apply(rule Bochner_Integration.integral_diff[of measure_pmf A f g, symmetric])
by $(\text{simp\_all add: integrable\_measure\_pmf\_finite})$
lemma $E_{\text{linear + 2}}$:
finite (set pmf A) $\implies$ $E (\text{map pmf f A}) + E (\text{map pmf g A}) = E (\text{map pmf ($\lambda x. (f x) + (g x)$) A})$

unfolding $E_{\text{def}}$ integral_map pmf apply (rule Bochner_Integration.integral_add[of measure_pmf A f g, symmetric]) by (simp_all add: integrable_measure_pmf_finite)

lemma $E_{\text{linear sum 2}}$:
finite (set pmf D) $\implies$ $E (\text{map pmf ($\lambda x. (\sum i < \text{up}. f i x)$) D}) = (\sum i < (\text{up::nat}). E (\text{map pmf (f i) D}))$

unfolding $E_{\text{def}}$ integral_map pmf apply (rule Bochner_Integration.integral_sum) by (simp add: integrable_measure_pmf_finite)

lemma $E_{\text{linear sum ally}}$:
finite (set pmf D) $\implies$ $E (\text{map pmf ($\lambda x. (\sum i \in A. f i x)$) D}) = (\sum i \in (A::'a set). E (\text{map pmf (f i) D}))$

unfolding $E_{\text{def}}$ integral_map pmf apply (rule Bochner_Integration.integral_sum) by (simp add: integrable_measure_pmf_finite)

lemma $E_{\text{finite sum fun}}$:
finite (set pmf X) $\implies$ $E (\text{map pmf f X}) = (\sum x \in \text{set pmf X}. \text{pmf X x} * f x)$

proof -
assume finite: finite (set pmf X)
have $E (\text{map pmf f X}) = (\int x. f x \partial \text{measure pmf X})$
  unfolding $E_{\text{def}}$ by auto
also have ... = $E (\text{map pmf (f i) D})$
  unfolding $E_{\text{def}}$ integral_map pmf apply (rule Bochner_Integration.integral_sum) (auto simp add: integrable_measure_pmf_finite)
finally show $\theta$thesis.
qed

lemma $E_{\text{bernoulli}}$:
$\theta \leq p \implies p \leq 1 \implies$ $E (\text{map pmf f (bernoulli pmf p)}) = p * (f \text{ True}) + (1 - p) * (f \text{ False})$

unfolding $E_{\text{def}}$ by (auto)

4.2 function $bv$

fun $bv$: nat $\Rightarrow$ bool list pmf where
$bv 0 = \text{return pmf []}$
$bv (\text{Suc n}) = \text{do}
  (xs::bool list) $\leftarrow$ $bv n$;
  (x::bool) $\leftarrow$ (bernoulli_pmf 0.5);
  return pmf (x#xs)
}
lemma bv_finite: finite (bv n)
by (induct n) auto

lemma len_bv.n: \( \forall xs \in \text{set}_\text{pmf} (bv n). \) length \( xs = n \)
apply (induct n) by auto

lemma bv.set: \( \text{set}_\text{pmf} (bv n) = \{ x::\text{bool list. length } x = n \} \)
proof (induct n)
case (Suc n)
then have \( \text{set}_\text{pmf} (bv (Suc n)) = (\bigcup x \in \{ x. \) length \( x = n \}. \{ \text{True} \# x, \text{False} \# x \} \)
by (simp add: \text{set}_\text{pmf}_\text{bernoulli} \text{UNIV}_\text{bool})
also have \( \ldots = \{ x\#xs | x \in \text{set}_\text{pmf} (bv n) \} \) by auto
also have \( \ldots = \{ x. \) length \( x = Suc n \} \) using Suc \text{length_cong} by fastforce
finally show ?case .
qed (simp)

lemma len_not_in_bv: length \( xs \neq n \) \( \Rightarrow \) \( xs \notin \text{set}_\text{pmf} (bv n) \)
by (auto simp: len_bv n)

lemma not_n_bv.0: length \( xs \neq n \) \( \Rightarrow \) pmf \( (bv n) \) \( xs = 0 \)
by (simp add: len_not_in_bv pmf_eq_0_set_pmf)

lemma bv.comp_bernoulli: \( n < l \) \( \Rightarrow \) map_pmf \( (\lambda y . y!n) (bv l) = \text{bernoulli}_\text{pmf} (5 / 10) \)
proof (induct n arbitrary: l)
case 0
then obtain m where \( l = Suc m \) by (metis Suc_pred)
then show \( \text{map}_\text{pmf} (\lambda y . y!l) (bv l) = \text{bernoulli}_\text{pmf} (5 / 10) \) by (auto simp: \text{map}_\text{pmf}_\text{def} \text{bind}_\text{return}_\text{pmf} \text{bind}_\text{assoc}_\text{pmf} \text{bind}_\text{return}_\text{pmf}'
next
case (Suc n)
then have \( 0 < l \) by auto
then obtain m where lsm: \( l = Suc m \) by (metis Suc_pred)
with Suc(2) have nltm: \( n < m \) by auto
from lsm have \( map_pmf (\lambda y . y ! Suc n) (bv l) \)
\( = \text{map}_\text{pmf} (\lambda x . x!n) (\text{bind}_\text{pmf} (bv m) (\lambda t. (\text{return}_\text{pmf} t))) \) by (auto simp: map_bind_pmf)
also have \( \ldots = \text{map}_\text{pmf} (\lambda x . x!n) (bv m) \) by (auto simp: bind_return_pmf')
also have \( \ldots = \text{bernoulli}_\text{pmf} (5 / 10) \) by (auto simp add: Suc(1)[of m, OF nltm])
finally
    show ?case .
qed

lemma pmf_2elemlist: pmf (bv (Suc 0)) ([x]) = pmf (bv 0) [] * pmf (bernoulli_pmf (5 / 10)) x
    unfolding bv.simps(2)[where n=0] pmf_bind pmf_return
    apply (subst integral_measure_pmf[where A={[]}])
    apply (auto) by (cases x) auto

lemma pmf_moreelemlist: pmf (bv (Suc n)) (x#xs) = pmf (bv n) xs * pmf (bernoulli_pmf (5 / 10)) x
    unfolding bv.simps(2) pmf_bind pmf_return
    apply (subst integral_measure_pmf[where A={xs}])
    apply auto apply (cases x) apply(auto)
    apply (meson indicator.simps(2) list.inject singletonD)
    apply (meson indicator.simps(2) list.inject singletonD)
    apply (cases x) by(auto)

lemma list_pmf: length xs = n \implies pmf (bv n) xs = (1 / 2) ^ n
proof(induct n arbitrary: xs)
  case 0
  then have xs = [] by auto
  then show pmf (bv 0) xs = (1 / 2) ^ 0 by(auto)
next
  case (Suc n xs)
  then obtain a as where split: xs = a#as by (metis Suc_length_conv)
  have length as = n using Suc(2) split by auto
  with Suc(1) have 1: pmf (bv n) as = (1 / 2) ^ n by auto
  from split pmf_moreelemlist[where n=n and x=a and xs=as] have
    pmf (bv (Suc n)) xs = pmf (bv n) as * pmf (bernoulli_pmf (5 / 10)) a by auto
  then have pmf (bv (Suc n)) xs = (1 / 2) ^ n * 1 / 2 using 1 by auto
  then show pmf (bv (Suc n)) xs = (1 / 2) ^ Suc n by auto
qed

lemma bv_0_notlen: pmf (bv n) xs = 0 \implies length xs \neq n
by(auto simp: list_pmf)

lemma length xs > n \implies pmf (bv n) xs = 0
proof (induct n arbitrary: xs)
  case (Suc n xs)
  then obtain a as where split: xs = a#as by (metis Suc_length_conv

16
Suc_lessE)
  have length as > n using Suc(2) split by auto
  with Suc(1) have 1: pmf (bv n) as = 0 by auto
  from split pmf_moreelemlist[where n=n and x=a and xs=as] have
    pmf (bv (Suc n)) xs = pmf (bv n) as * pmf (bernoulli_pmf (5 / 10))
  a by auto
  then have pmf (bv (Suc n)) xs = 0 * 1 / 2 using 1 by auto
  then show pmf (bv (Suc n)) xs = 0 by auto
qed simp

lemma map_hd_list_pmf: map_pmf hd (bv (Suc n)) = bernoulli_pmf (5 / 10)
  by (simp add: map_pmf_def bind_assoc_pmf bind_return_pmf bind_return_pmf')

lemma map_tl_list_pmf: map_pmf tl (bv (Suc n)) = bv n
  by (simp add: map_pmf_def bind_assoc_pmf bind_return_pmf bind_return_pmf')

4.3 function flip

fun flip :: nat ⇒ bool list ⇒ bool list where
  flip [] = []
| flip 0 (x#xs) = (¬x)#xs
| flip (Suc n) (x#xs) = x#(flip n xs)

lemma flip_length[simp]: length (flip i xs) = length xs
  apply(induct xs arbitrary: i) apply(simp) apply(case_tac i) by(simp_all)

lemma flip_out_of_bounds: y ≥ length X ⇒ flip y X = X
  apply(induct X arbitrary: y)
  proof −
    case (Cons X Xs)
    hence y > 0 by auto
    with Cons obtain y' where y1: y = Suc y' and y2: y' ≥ length Xs by
      (metis Suc_pred' length.Cons not_less_eq_eq)
    then have flip y (X # Xs) = X #(flip y' Xs) by auto
    moreover from Cons y2 have flip y' Xs = Xs by auto
    ultimately show ?case by auto
  qed simp

lemma flip_other: y < length X ⇒ z < length X ⇒ z ≠ y ⇒ flip z X
  ! y = X ! y
  apply(induct y arbitrary: X z)
  apply(simp) apply (metis flip.elims neq0_conv nth.Cons.0)
proof (case_tac z, goal_cases)
  case (1 y X z)
  then obtain a as where X=a#as using length_greater_0_conv by (metis (full_types) flip.elims)
    with 1(5) show ?case by (simp)
next
  case (2 y X z z')
    from 2 have 3: z' ≠ y by auto
    from 2(2) have length X > 0 by auto
    then obtain a as where aas: X = a#as by (metis (full_types) flip.elims length_greater_0_conv)
    then have a: flip (Suc z') X ! Suc y = flip z' as ! y
      and b : (X ! Suc y) = (as ! y) by (rule a)
    also have ... = as ! y by (rule c)
    also have ... = (X ! Suc y) by (rule b[symmetric])
    finally show flip z X ! Suc y = (X ! Suc y) .
qed

lemma flip_itself: y < length X =⇒ flip y X ! y = (∃ y < length X)
apply(induct y arbitrary: X)
apply(simp) apply (metis flip.elims nth.Cons_0 old.nat.distinct(2))
proof –
fix y
fix X::bool list
assume iH: (∀ X. y < length X =⇒ flip y X ! y = (∃ X ! y))
assume len: Suc y < length X
from len have y < length X by auto
from len have length X > 0 by auto
then obtain z zs where zs: X = z#zs by (metis (full_types) flip.elims length_greater_0_conv)
  then have a: flip (Suc y) X ! Suc y = flip y zs ! y
    and b: (∃ X ! Suc y) = (∃ zs ! y) by auto
from len zs have y < length zs by auto
note c=iH[OF this]
from a b c show flip (Suc y) X ! Suc y = (∃ X ! Suc y) by auto
qed

lemma flip_twice: flip i (flip i b) = b
proof (cases i < length b)
case True
then have A: i < length (flip i b) by simp
show ?thesis apply(simp add: list_eq_iff_nth_eq) apply(clarify)
proof (goal_cases)
case (1 j)
then show ?case
  apply (cases i = j)
  using flip itself[OF A] flip itself[OF True] apply(simp)
  using flip other True 1 by auto
qed
qed (simp add: flip out_of_bounds)

lemma flipidiflip: y < length X =⇒ e < length X =⇒ flip e X ! y = (if e=y then ~ (X ! y) else X ! y)
apply(cases e = y)
apply(simp add: flip itself)
by(simp add: flip other)

lemma bernoulli_Not: map_pmf Not (bernoulli_pmf (1 / 2)) = (bernoulli_pmf (1 / 2))
apply(rule pmf_eqI)
proof (case_tac i, goal_cases)
case (1 i)
  then have pmf (map_pmf Not (bernoulli_pmf (1 / 2))) i =
    pmf (map_pmf Not (bernoulli_pmf (1 / 2))) (Not False) by auto
  also have ... = pmf (bernoulli_pmf (1 / 2)) False apply (rule pmf_map_inj')
  apply(rule injI) by auto
  also have ... = pmf (bernoulli_pmf (1 / 2)) i by auto
  finally show ?case.
next
case (2 i)
  then have pmf (map_pmf Not (bernoulli_pmf (1 / 2))) i =
    pmf (map_pmf Not (bernoulli_pmf (1 / 2))) (Not True) by auto
  also have ... = pmf (bernoulli_pmf (1 / 2)) True apply (rule pmf_map_inj')
  apply(rule injI) by auto
  also have ... = pmf (bernoulli_pmf (1 / 2)) i by auto
  finally show ?case.
qed

lemma inv_flip bv: map_pmf (flip i) (bv n) = (bv n)
proof (induct n arbitrary: i)
case (Suc n i)
  note iH = this
have \(\text{bind pmf} (bv n) (\lambda x. \text{bind pmf} (\text{bernoulli pmf} (1 / 2)) (\lambda x. \text{map pmf} (\text{flip i}) (\text{return pmf} (x \# x))))\)
\[= \text{bind pmf} (\text{bernoulli pmf} (1 / 2)) (\lambda x. \text{bind pmf} (bv n) (\lambda x. \text{map pmf} (\text{flip i}) (\text{return pmf} (x \# x))))\]
by (rule \(\text{bind commute pmf}\))
also have \(\ldots = \text{bind pmf} (\text{bernoulli pmf} (1 / 2)) (\lambda x. \text{bind pmf} (bv n) (\lambda x. \text{return pmf} (x \# x)))\)
proof (cases i)
case 0
then have \(\ldots = \text{bind pmf} (\text{bernoulli pmf} (1 / 2)) (\lambda x. \text{bind pmf} (bv n) (\lambda x. \text{return pmf} ((\neg x) \# x)))\)
by (rule \(\text{bind commute pmf}\))
also have \(\ldots = \text{bind pmf} (bv n) (\lambda x. \text{bind pmf} (\text{bernoulli pmf} (1 / 2)) (\lambda x. \text{return pmf} ((\neg x) \# x)))\)
by (rule \(\text{bind commute pmf}\))
finally show ?thesis.
next
case (Suc i)
have \(\text{bind pmf} (\text{bernoulli pmf} (1 / 2)) (\lambda x. \text{bind pmf} (bv n) (\lambda x. \text{map pmf} (\text{flip i}) (\text{return pmf} (x \# x))))\)
\[= \text{bind pmf} (\text{bernoulli pmf} (1 / 2)) (\lambda x. \text{bind pmf} (bv n) (\lambda x. \text{return pmf} (x \# \text{flip i'} x)))\]
unfolding Suc by (simp)
also have \(\ldots = \text{bind pmf} (\text{bernoulli pmf} (1 / 2)) (\lambda x. \text{bind pmf} (\text{map pmf} (\text{flip i'}) (bv n)) (\lambda x. \text{return pmf} (x \# x)))\)
by (auto simp add: \(\text{bind map pmf}\))
also have \(\ldots = \text{bind pmf} (\text{bernoulli pmf} (1 / 2)) (\lambda x. \text{bind pmf} (bv n) (\lambda x. \text{return pmf} (x \# x)))\)
using \(iH\) of i' by simp
finally show ?thesis.
qed
also have \(\ldots = \text{bind pmf} (bv n) (\lambda x. \text{bind pmf} (\text{bernoulli pmf} (1 / 2)) (\lambda x. \text{return pmf} (x \# x)))\)
by (rule \(\text{bind commute pmf}\))
finally show ?case by (simp add: \(\text{map pmf def bind assoc pmf}\))
4.4 Example for pmf

definition twocoins =
  do { x ← (bernoulli_pmf 0.4); y ← (bernoulli_pmf 0.5); return_pmf (x ∨ y) }

lemma experiment0_7: pmf twocoins True = 0.7
unfolding twocoins_def
  unfolding pmf_bind pmf_return
  apply (subst integral_measure_pmf [where A={True, False}])
  by auto

4.5 Sum Distribution

definition Sum_pmf p Da Db = (bernoulli_pmf p) >>= (% b. if b then map_pmf Inl Da else map_pmf Inr Db)

lemma b0: bernoulli_pmf 0 = return_pmf False
apply (rule pmf_eqI) apply (case_tac i)
by (simp_all)

lemma b1: bernoulli_pmf 1 = return_pmf True
apply (rule pmf_eqI) apply (case_tac i)
by (simp_all)

lemma Sum_pmf_0: Sum_pmf 0 Da Db = map_pmf Inr Db
unfolding Sum_pmf_def
apply (rule pmf_eqI)
  by (simp add: b0 bind_return_pmf)

lemma Sum_pmf_1: Sum_pmf 1 Da Db = map_pmf Inl Da
unfolding Sum_pmf_def
apply (rule pmf_eqI)
  by (simp add: b1 bind_return_pmf)

definition Proj1_pmf D = map_pmf (%a. case a of Inl e ⇒ e) (cond_pmf D {f. (∃ e. Inl e = f)})
lemma $A$: $(\text{case\_sum} (\lambda e. e) (\lambda a. \text{undefined})) (\text{Inl} e) = e$
by(simp)

lemma $B$: $\text{inj} \ (\text{case\_sum} (\lambda e. e) (\lambda a. \text{undefined}))$
oops

lemma none: $p > 0 \implies p < 1 \implies (\text{set\_pmf} \ \text{(bernoulli\_pmf} p \gg (\lambda b. \text{if} b \ \text{then} \ \text{map\_pmf} \ \text{Inl} Da \ \text{else} \ \text{map\_pmf} \ \text{Inr} Db))$
\cap \{f. \ (\exists e. \ \text{Inl} e = f)\} \neq \{\}
apply(simp add: UNIV_bool)
using set\_pmf\_not\_empty by fast

lemma none2: $p > 0 \implies p < 1 \implies (\text{set\_pmf} \ \text{(bernoulli\_pmf} p \gg (\lambda b. \text{if} b \ \text{then} \ \text{map\_pmf} \ \text{Inl} Da \ \text{else} \ \text{map\_pmf} \ \text{Inr} Db))$
\cap \{f. \ (\exists e. \ \text{Inr} e = f)\} \neq \{\}
apply(simp add: UNIV_bool)
using set\_pmf\_not\_empty by fast

lemma $C$: $\text{set\_pmf} \ (\text{Proj1\_pmf} \ (\text{Sum\_pmf} 0.5 Da Db)) = \text{set\_pmf} Da$
proof –
show ?thesis
unfolding Sum\_pmf\_def Proj1\_pmf\_def
apply(simp add: )
using none[of 0.5 Da Db] apply(simp add: set\_cond\_pmf UNIV_bool)
by force
qed

thm integral\_measure\_pmf

thm pmf\_cond pmf\_cond[OF none]

lemma proj1\_pmf: assumes $p>0 \ p<1$ shows $\text{Proj1\_pmf} \ (\text{Sum\_pmf} p Da Db) = Da$
proof –

have kl: $\forall e. \ \text{pmf} \ (\text{map\_pmf} \ \text{Inr} Db) (\text{Inl} e) = 0$
apply(simp only: pmf\_eq\_0\_set\_pmf)
apply(simp) by blast

have ll: measure\_pmf\_prob
(bernoulli\_pmf p \gg (\lambda b. \text{if} b \ \text{then} \ \text{map\_pmf} \ \text{Inl} Da \ \text{else} \ \text{map\_pmf} \ \text{Inr} Db))
\{f. \ (\exists e. \ \text{Inl} e = f)\} = p
using assms

22
apply(simp add: integral_pmf[symmetric] pmf_bind)
apply(subst Bochner_Integration.integral_add)
using integrable_pmf apply fast
using integrable_pmf apply fast
  by(simp add: integral_pmf)

have E: (cond_pmf
  (bernoulli_pmf p >>=
   (λb. if b then map_pmf Inl Da else map_pmf Inr Db))
  {f. ∃e. Inl e = f}) =
  map_pmf Inl Da
apply(rule pmf_eqI)
apply(subst pmf_cond)
using none[of p Da Db] assms apply (simp)
using assms apply(auto)
  apply(subst pmf_bind)
  apply(simp add: kl ll)
  apply(simp only: pmf_eq 0 set_pmf) by auto

have ID: case_sum (λe. e) (λa. undefined) ∘ Inl = id
  by fastforce
show ?thesis
  unfolding Sum_pmf_def Proj1_pmf_def
  apply(simp only: E)
  apply(simp add: pmf_map_comp ID)
done

qed

definition Proj2_pmf D = map_pmf (%a. case a of Inr e ⇒ e) (cond_pmf D {f. (∃e. Inr e = f)})

lemma proj2_pmf: assumes p>0 p<1 shows Proj2_pmf (Sum_pmf p Da Db) = Db
proof –

have kl: ∀e. pmf (map_pmf Inl Da) (Inr e) = 0
  apply(simp only: pmf_eq 0_set_pmf)
  apply(simp) by blast

have ll: measure_pmf.prob
  (bernoulli_pmf p >>=
   (λb. if b then map_pmf Inl Da else map_pmf Inr Db))
\{f \exists e. \text{Inr } e = f\} = 1 - p

using assms
apply(simp add: integral_pmf[symmetric] pmf_bind)
apply(subst Bochner_Integration.integral_add)
using integrable_pmf apply fast
using integrable_pmf apply fast
by(simp add: integral_pmf)

have E: (cond_pmf
(bernoulli_pmf p \Rightarrow
(\lambda b. if b then map_pmf \text{Inl } Da else map_pmf \text{Inr } Db))
{f. \exists e. \text{Inr } e = f}) =
map_pmf \text{Inr } Db
apply(rule pmf_eqI)
apply(subst pmf_cond)
using none2[of p Da Db] assms apply (simp)
using assms apply(auto)
apply(subst pmf_bind)
apply(simp add: kl ll)
apply(simp only: pmf_eq 0 set_pmf) by auto

have ID: case_sum (\lambda e. \text{undefined}) (\lambda a. a) \circ \text{Inr} = id
by fastforce

show \ ?thesis
unfolding Sum_pmf_def Proj2_pmf_def
apply(simp only: E)
apply(simp add: pmf_map_comp ID)
done

qed

definition invSum invA invB D x i == invA (Proj1_pmf D) x i \land invB (Proj2_pmf D) x i

lemma invSum_split: \text{invSum}_p > 0 \Rightarrow p < 1 \Rightarrow \text{invA} Da x i \Rightarrow \text{invB} Db x i \Rightarrow
invSum invA invB (Sum_pmf p Da Db) x i
by(simp add: invSum_def proj1_pmf proj2_pmf)

term (%a. case a of \text{Inl } e \Rightarrow \text{Inl } (fa e) | \text{Inr } e \Rightarrow \text{Inr } (fb e))
definition f_on2 fa fb = (%a. case a of \text{Inl } e \Rightarrow \text{map_pmf Inl } (fa e) | \text{Inr } e
⇒ map_mpf Inr (fb e))

term bind_mpf

lemma Sum_bind_mpf: assumes a: bind_mpf Da fa = Da' and b: bind_mpf Db fb = Db'
  shows bind_mpf (Sum_mpf p Da Db) (f_on2 fa fb)
  = Sum_mpf p Da' Db'
proof -
  { fix x
    have (if x then map_mpf Inl Da else map_mpf Inr Db) ≅
      case_sum (λe. map_mpf Inl (fa e))
      (λe. map_mpf Inr (fb e))
    =
      (if x then map_mpf Inl Da ≅ case_sum (λe. map_mpf Inl (fa e))
      (λe. map_mpf Inr (fb e))
      else map_mpf Inr Db ≅ case_sum (λe. map_mpf Inl (fa e))
      (λe. map_mpf Inr (fb e)))
      apply(simp) done
  also
  have ... = (if x then map_mpf Inl (bind_mpf Da fa) else map_mpf Inr
      (bind_mpf Db fb))
    by(auto simp add: map_mpf_def bind_assoc_mpf bind_return_mpf)
  also
  have ... = (if x then map_mpf Inl Da' else map_mpf Inr Db')
    using a b by simp
  finally
  have (if x then map_mpf Inl Da else map_mpf Inr Db) ≅
    case_sum (λe. map_mpf Inl (fa e))
    (λe. map_mpf Inr (fb e)) = (if x then map_mpf Inl Da' else
      map_mpf Inr Db').
  } note gr=this

show ?thesis
  unfolding Sum_mpf_def f_on2_def
  apply(rule pmf_eqI)
  apply(case_tac i)
  by(simp_all add: bind_return_mpf bind_assoc_mpf gr)
qed

definition sum_map_mpf fa fb = (%a. case a of Inl e ⇒ Inl (fa e) | Inr e
\[
\Rightarrow \text{Inr }(fb\ e)
\]

**lemma** \(\text{Sum}_{\text{map}_{\text{pmf}}}:\) **assumes** \(a: \text{map}_{\text{pmf}} fa\ Da = Da'\) and \(b: \text{map}_{\text{pmf}} fb\ Db = Db'\)

**shows** \(\text{map}_{\text{pmf}} (\text{sum}_{\text{map}_{\text{pmf}} fa fb}) (\text{Sum}_{\text{pmf}} p Da Db) = \text{Sum}_{\text{pmf}} p Da' Db'\)

**proof**

- **have** \(\text{map}_{\text{pmf}} (\text{sum}_{\text{map}_{\text{pmf}} fa fb}) (\text{Sum}_{\text{pmf}} p Da Db) = \text{bind}_{\text{pmf}} (\text{Sum}_{\text{pmf}} p Da Db) (f_{\text{on2}} (\lambda x. \text{return}_{\text{pmf}} (fa\ x)) (\lambda x. \text{return}_{\text{pmf}} (fb\ x)))\)
  using \(a\ b\)
- unfolding \(\text{map}_{\text{pmf}}\_\text{def}\ \text{sum}_{\text{map}_{\text{pmf}}}\_\text{def}\ f_{\text{on2}}\_\text{def}\)
  by (auto simp add: \text{bind}_{\text{return}_{\text{pmf}}} \text{sum}_{\text{case_distrib}})
also
- **have** \(\ldots = \text{Sum}_{\text{pmf}} p Da' Db'\)
  using \(\text{assms[unfolded map}_{\text{pmf}}\_\text{def}]\)
  by (rule \text{Sum}_{\text{bind}_{\text{pmf}}} )
finally
- **show** \(\text{?thesis} .\)
qed

end

\section{Randomized Online and Offline Algorithms}

theory *Competitive\_Analysis*

imports
  *Prob\_Theory*
  *On\_Off*
begin

5.1 Competitive Analysis Formalized

type\_synonym \(\langle s, is, r, a\rangle\)\_alg\_on\_step = \(\langle s \ast \langle s \ast \text{is} \Rightarrow \langle r \Rightarrow (\langle a \ast \langle s \ast \text{is} \rangle) \text{ pmf}\rangle\rangle\rangle\)
type\_synonym \(\langle s, is\rangle\)\_alg\_on\_init = \(\langle s \Rightarrow \langle is \text{ pmf}\rangle\rangle\)
type\_synonym \(\langle s, is, q, a\rangle\)\_alg\_on\_rand = \(\langle s, is\rangle\)\_alg\_on\_init \ast \(\langle s, is, q, a\rangle\)\_alg\_on\_step

5.1.1 classes of algorithms

definition deterministic\_init :: \(\langle s, is\rangle\)\_alg\_on\_init \Rightarrow \text{bool}\) where
deterministic\_init \(I \longleftrightarrow (\forall \text{ init. card}(\text{ set}_{\text{pmf}} (I \text{ init})) = 1)\)

26
**definition** deterministic_step :: ('s,'is,'q,'a)alg_on_step ⇒ bool where
deterministic_step S ←→ (∀ i q. card(set_pmf(S (i, is) q))) = 1

**definition** random_step :: ('s,'is,'q,'a)alg_on_step ⇒ bool where
random_step S ←→ ∼ deterministic_step S

5.1.2 Randomized Online and Offline Algorithms

context On_Off
begin

fun steps where
steps s [] [] = s
| steps s (q#qs) (a#as) = steps (step s q a) qs as

lemma steps_append: length qs = length as ⇒ steps s (qs@qs') (as@as')
= steps (steps s qs as) qs' as'
apply(induct qs as arbitrary: s rule: list_induct2)
  by simp_all

lemma T_append: length qs = length as ⇒ T s (qs@[q]) (as@[a]) = T s
qs as + t (steps s qs as) q a
apply(induct qs as arbitrary: s rule: list_induct2)
  by simp_all

lemma T_append2: length qs = length as ⇒ T s (qs@qs') (as@as') = T
s qs as + T (steps s qs as) qs' as'
apply(induct qs as arbitrary: s rule: list_induct2)
  by simp_all

abbreviation Step_rand :: ('state,'is,'request,'answer) alg_on_rand ⇒ 'request
⇒ 'state * 'is ⇒ ('state * 'is) pmf where
Step_rand A r s ≡ bind_pmf ((snd A) s r) (λ(a,is'). return_pmf (step (fst s) r a, is'))

fun config'_rand :: ('state,'is,'request,'answer) alg_on_rand ⇒ ('state*'is)
pmf ⇒ 'request list
⇒ ('state * 'is) pmf where
config'_rand A s [] = s |
config'_rand A s (r#rs) = config'_rand A (s >>= Step_rand A r) rs
lemma config'\_rand\_snoč: config'\_rand A (rs@[r]) = config'\_rand A rs \Rightarrow Step\_rand A r
apply(induct rs arbitrary: s) by(simp\_all)

lemma config'\_rand\_append: config'\_rand A s (xs@ys) = config'\_rand A (config'\_rand A s xs) ys
apply(induct xs arbitrary: s) by(simp\_all)

abbreviation config\_rand where
cfg\_rand A s0 rs == config'\_rand A ((fst A s0) \Rightarrow (\lambda is. return\_pmf (s0, is))) rs

lemma config'\_rand\_induct: (\forall x \in set\_pmf init. P (fst x)) \Rightarrow (\forall s q a. P s \Rightarrow P (step s q a))
\Rightarrow \forall x \in set\_pmf (config'\_rand A init qs). P (fst x)
proof (induct qs arbitrary: init)
case (Cons r rs)
show ?case apply(simp)
apply(rule Cons(1))
apply(subst Set\_ball\_simp\_s(9)[where P=P, symmetric])
apply(subst set\_map\_pmf[symmetric])
apply(simp only: map\_bind\_pmf)
apply(simp add: bind\_assoc\_pmf bind\_return\_pmf split\_def)
using Cons(2,3) apply blast
by fact
qed (simp)

lemma config\_rand\_induct: P s0 \Rightarrow (\forall s q a. P s \Rightarrow P (step s q a)) \Rightarrow
\forall x \in set\_pmf (config\_rand A s0 qs). P (fst x)
using config'\_rand\_induct[of ((fst A s0) \Rightarrow (\lambda is. return\_pmf (s0, is))) P]
by auto

fun T\_on\_rand' :: ('state,'is,'request,'answer) alg\_on\_rand \Rightarrow ('state\_is) pmf \Rightarrow 'request list \Rightarrow real where
T\_on\_rand' A s [] = 0 |
T\_on\_rand' A s (r#rs) = E ( s \Rightarrow (\lambda s. bind\_pmf (snd A s r) (\lambda(a, is'). return\_pmf (real (t (fst s) r a)))) )
+ T\_on\_rand' A (s \Rightarrow Step\_rand A r) rs

lemma T\_on\_rand'\_append: T\_on\_rand' A s (xs@ys) = T\_on\_rand' A s xs
+ $T_{on\_rand}' A \ (\text{config}'_{\text{rand}} A s xs) ys$

**apply** (induct $xs$ arbitrary: $s$) by simp_all

**abbreviation** $T_{on\_rand} :: (\text{state}'is,\text{request}'answer) \text{ alg}_{on\_rand} \Rightarrow \text{'state} \Rightarrow \text{real list} \Rightarrow \text{real} \ where$

$T_{on\_rand} A s rs == T_{on\_rand}' A \ (\text{fst} A s \Rightarrow (\lambda is. \text{return}_{\text{pmf}} (s,is))) \ rs$

**lemma** $T_{on\_rand}_{\text{append}}$: $T_{on\_rand} A s (xs@ys) = T_{on\_rand} A s xs + T_{on\_rand}' A \ (\text{config}_{\text{rand}} A s xs) ys$

**by** (rule $T_{on\_rand}'_{\text{append}}$)

**abbreviation** $T_{on\_rand}'_{n} A s0 n xs n == T_{on\_rand}' A \ (\text{config}_{\text{rand}} A s0 \ (\text{take} n xs)) \ [xs!n]$

**lemma** $T_{on\_rand}'_{\text{as\_sum}}$: $T_{on\_rand}' A s0 rs = \text{sum} (T_{on\_rand}'_{n} A s0 rs) \{..<\text{length} rs\}$

**apply** (induct $rs$ rule: rev_induct)

**by** (simp_all add: $T_{on\_rand}'_{\text{append nth append}}$)

**abbreviation** $T_{on\_rand}_{n} A s0 n xs n == T_{on\_rand}' A \ (\text{config}_{\text{rand}} A s0 \ (\text{take} n xs)) \ [xs!n]$

**lemma** $T_{on\_rand}_{\text{as\_sum}}$: $T_{on\_rand} A s0 rs = \text{sum} (T_{on\_rand}_{n} A s0 rs) \{..<\text{length} rs\}$

**apply** (induct $rs$ rule: rev_induct)

**by** (simp_all add: $T_{on\_rand}'_{\text{append nth append}}$)

**lemma** $T_{on\_rand}'_{nn}$: $T_{on\_rand}' A s qs \geq 0$

**apply** (induct $qs$ arbitrary: $s$)

**apply** (simp_all add: bind_return_pmf)

**apply** (rule add_nonneg_nonneg)

**apply** (rule E_nonneg)

**by** (simp_all add: split_def)

**lemma** $T_{on\_rand}_{nn}$: $T_{on\_rand} (I,S) s0 qs \geq 0$

**by** (rule $T_{on\_rand}'_{nn}$)

**definition** $\text{compet}_{\text{rand}} :: (\text{state}'is,\text{request}'answer) \text{ alg}_{on\_rand} \Rightarrow \text{real} \Rightarrow \text{state set} \Rightarrow \text{bool} \ where$

$\text{compet}_{\text{rand}} A c S0 = (\forall s \in S0. \exists b \geq 0. \forall rs. \text{wf} s rs \rightarrow T_{on\_rand} A s$
\[rs \leq c \times T_{\text{opt}} s rs + b)\]

5.2 embedding of deterministic into randomized algorithms

fun embed :: ('state,'is,'request,'answer) alg_on ⇒ ('state,'is,'request,'answer) alg_on_rand
where
embed A = ( \( \lambda s. \) return_pmf (fst A s) \) ,
\( \lambda s r. \) return_pmf (snd A s r) )

lemma T_deter_rand: T_off (\( \lambda s0. \) (off2 A (s0, x))) s0 qs = T_on_rand' (embed A) (return_pmf (s0,x)) qs
apply(induct qs arbitrary: s0 x)
  by(simp_all add: Step_def bind_return_pmf split: prod.split)

lemma config_embed: config_rand (embed A) (return_pmf s0) qs = return_pmf (config A s0 qs)
apply(induct qs arbitrary: s0)
  apply(simp_all add: Step_def split_def bind_return_pmf) by metis

lemma config_embed: config_rand (embed A) s0 qs = return_pmf (config A s0 qs)
apply(simp add: bind_return_pmf)
  apply(subst config_embed[unfolded embed.simps])
  by simp

lemma T_on_embed: T_on A s0 qs = T_on_rand (embed A) s0 qs
using T_deter_rand[where x=fst A s0, of s0 qs A] by(auto simp: bind_return_pmf)

lemma T_on_embed': T_on' A (s0,x) qs = T_on_rand'(embed A) (return_pmf (s0,x)) qs
using T_deter_rand T_on_on' by metis

lemma compet_embed: compet A c S0 = compet_rand (embed A) c S0
unfolding compet_def compet_rand_def using T_on_embed by metis

end
6 Deterministic List Update

theory Move_to_Front
imports
  Swaps
  On_Off
  Competitive_Analysis
begin

declare Let_def[simp]

6.1 Function mtf

definition mtf :: 'a ⇒ 'a list ⇒ 'a list where
mtf x xs = (if x ∈ set xs then x # (take (index xs x) xs) @ drop (index xs x + 1) xs else xs)

lemma mtf_id[simp]: x ∉ set xs ⇒ mtf x xs = xs
by(simp add: mtf_def)

lemma mtf0[simp]: x ∈ set xs ⇒ mtf x xs ! 0 = x
by(auto simp: mtf_def)

lemma before_in_mtf: assumes z ∈ set xs
shows x < y in mtf z xs ←→ (y ≠ z ∧ (if x=z then y ∈ set xs else x < y in xs))
proof−
  have 0: index xs z < size xs by (metis assms index_less_size_conv)
  let ?xs = take (index xs z) xs @ xs ! index xs z # drop (Suc (index xs z)) xs
  have x < y in mtf z xs = (y ≠ z ∧ (if x=z then y ∈ set ?xs else x < y in ?xs))
  using assms
  by(auto simp add: mtf_def before_in_def index_append)
  (metis add_lessD1 index_less_size_conv length_take less_Suc_eq not_less_eq)
with id_take_nth_drop[OF 0, symmetric] show ?thesis by(simp)
qed

lemma Inv_mtf: set xs = set ys ⇒ z : set ys ⇒ Inv xs (mtf z ys) = Inv xs ys ∪ \{(x,z) | x. x < z in xs ∧ x < z in ys\} − \{(z,x) | x. z < x in xs ∧ x < z in ys\}
by (auto simp add: Inv_def before_in_mtf not_before_in dest: before_in_setD1)

lemma set_mtf[simp]: set (mtf x xs) = set xs
by (simp add: mtf_def)
  (metis append_take_drop_id Cons_nth_drop Suc index_le_refl Un_insert_right nth_index set_append set.simps(2))

lemma length_mtf[simp]: size (mtf x xs) = size xs
by (auto simp add: mtf_def)
  (metis index_less leD)

lemma distinct_mtf[simp]: distinct (mtf x xs) = distinct xs
by (metis length_mtf set_mtf card distinct)

6.2 Function mtf2

definition mtf2 :: nat ⇒ 'a ⇒ 'a list ⇒ 'a list where
  mtf2 n x xs =
  (if x : set xs then swaps [index xs x − n..<index xs x] xs else xs)

lemma mtf_eq_mtf2: mtf x xs = mtf2 (length xs − 1) x xs
proof −
  have x : set xs ⇒ index xs x − (size xs − Suc 0) = 0
  by (auto simp: less_Suc_eq [symmetric])
  thus ?thesis
  by (auto simp: mtf_def mtf2_def swaps_eq_nth_take_drop)
qed

lemma mtf20[simp]: mtf2 0 x xs = xs
by (auto simp add: mtf2_def)

lemma length_mtf2[simp]: length (mtf2 n x xs) = length xs
by (auto simp: mtf2_def index_less_size_conv [symmetric]
  simp del: index_conv_size_if_notin)

lemma set_mtf2[simp]: set (mtf2 n x xs) = set xs
by (auto simp: mtf2_def index_less_size_conv [symmetric]
  simp del: index_conv_size_if_notin)

lemma distinct_mtf2[simp]: distinct (mtf2 n x xs) = distinct xs
by (metis length_mtf2 set_mtf2 card distinct)

lemma card_Inv_mtf2: xs!j = ys!0 ⇒ j < length xs ⇒ dist_perm xs ys
⇒
  card (Inv (swaps [i..<j] xs) ys) = card (Inv xs ys) − int(j − i)
proof (induction j arbitrary: xs)
  case (Suc j)
  show ?case
  proof (cases)
    assume i > j thus ?thesis by simp
  qed

next
  assume [arith]: ¬ i > j
  have 0: Suc j < length ys by (metis Suc.prems(2,3) distinct_card)
  have 1: (ys ! 0, xs ! j) : Inv ys xs
  proof (auto simp: Inv_def)
    show ys ! 0 < xs ! j in ys using Suc.prems
      by (metis Suc_lessD n_not_Suc_n not before0 not before in nth_eq_iff_index_eq nth_mem)
  qed
  have 2: card (Inv ys xs) ≠ 0 using 1 by auto
  have int (card (Inv (swaps [i..<Suc j] xs) ys)) =
    card (Inv (swap j xs) ys) − int (j − i) using Suc by simp
  also have ... = card (Inv ys (swap j xs)) − int (j − i)
    by (simp add: card_Inv_sym)
  also have ... = card (Inv ys xs − {(ys ! 0, xs ! j)}) − int (j − i)
    using Suc.prems 0 by (simp add: Inv_swap)
  also have ... = int (card (Inv ys xs) − 1) − (j − i)
    using 1 by (simp add: card_Diff_singleton)
  also have ... = card (Inv ys xs) − int (Suc j − i) using 2 by arith
  also have ... = card (Inv xs ys) − int (Suc j − i) by (simp add: card_Inv_sym)
  finally show ?thesis .
  qed
  qed simp

6.3 Function Lxy

definition Lxy :: 'a list ⇒ 'a set ⇒ 'a list where
  Lxy xs S = filter (λz. z ∈ S) xs
thm inter_set_filter

lemma Lxy_length_cons: length (Lxy xs S) ≤ length (Lxy (x # xs) S)
unfolding Lxy_def by (simp)

lemma Lxy_empty [simp]: Lxy [] S = []
unfolding Lxy_def by simp
lemma Lxy_set_filter: set (Lxy xs S) = S ∩ set xs
by (simp add: Lxy_def inter_set_filter)

lemma Lxy_distinct: distinct xs ==> distinct (Lxy xs S)
by (simp add: Lxy_def)

lemma Lxy_append: Lxy (xs@ys) S = Lxy xs S @ Lxy ys S
by (simp add: Lxy_def)

lemma Lxy_snoc: Lxy (xs@[x]) S = (if x ∈ S then Lxy xs S @ [x] else Lxy xs S)
by (simp add: Lxy_def)

lemma Lxy_not: S ∩ set xs = {} ==> Lxy xs S = []
unfolding Lxy_def apply (induct xs) by simp_all

lemma Lxy_notin: set xs ∩ S = {} ==> Lxy xs S = []
apply (induct xs) by (simp_all add: Lxy_def)

lemma Lxy_in: x ∈ S ==> Lxy [x] S = [x]
by (simp add: Lxy_def)

lemma Lxy_project:
  assumes x≠y x ∈ set xs y ∈ set xs distinct xs
  and x < y in xs
  shows Lxy xs {x,y} = [x,y]
proof
  from assms have ij: index xs x < index xs y
  and xinx: index xs x < length xs
  and yinx: index xs y < length xs unfolding before_in_def by auto
  from xinx obtain a as where dec1: a @ [xs!index xs x] @ as = xs
    and a = take (index xs x) xs and as = drop (Suc (index xs x)) xs
    and length_a: length a = index xs x and length_as: length as = length xs - index xs x - 1
    using id_take_nth_drop by fastforce
  then have index xs y≥length (a @ [xs!index xs x]) using length_a ij by auto
  then have ((a @ [xs!index xs x]) @ as) ! index xs y = as ! (index xs y-length (a @ [xs ! index xs x])) using nth_append[where xs=a @ [xs!index xs x] and ys=as]
    by (simp)
then have \( \text{xsj}: \text{xs}!\) index \( \text{xs} = \text{as}! (\text{index} \ \text{xs} \ y - \text{index} \ \text{xs} \ x - 1) \) using
dcl length_a by auto
have las: (\text{index} \ \text{xs} \ y - \text{index} \ \text{xs} \ x - 1) < length as using length_as yinxs
ij by simp
obtain b c where dec2: b @ [\text{xs}!\text{index} \ \text{xs} \ y] @ c = as
and b = take (\text{index} \ \text{xs} \ y - \text{index} \ \text{xs} \ x - 1) as c = drop (Suc (\text{index} \ \text{xs} \ y - \text{index} \ \text{xs} \ x - 1))
using id_take_nth_drop[OF las] xs by force
have x.dec: a @ [\text{xs}!\text{index} \ \text{xs} \ x] @ b @ [\text{xs}!\text{index} \ \text{xs} \ y] @ c = \text{xs} using
dec1 dec2 by auto

from x.dec assms(4) have distinct ((a @ [\text{xs}!\text{index} \ \text{xs} \ x] @ b @ [\text{xs}!\text{index} \ \text{xs} \ y]) @ c) by simp
then have c.empty: set c \cap \{x,y\} = \{}
and b.empty: set b \cap \{x,y\} = \{\} and a.empty: set a \cap \{x,y\} = \{}
by(auto simp add: assms(2,3))

have Lxy (a @ [\text{xs}!\text{index} \ \text{xs} \ x] @ b @ [\text{xs}!\text{index} \ \text{xs} \ y] @ c) \{x,y\} = [x,y]
apply(simp only: Lxy_append)
apply(simp add: assms(2,3))
using a.empty b.empty c.empty by(simp add: Lxy_notin Lxy_in)

with x.dec show \( ?\)thesis by auto
qed

lemma Lxy_mono: \{x,y\} \subseteq set \text{xs} \implies \text{distinct} \text{xs} \implies x < y in \text{xs} = x < y in Lxy \text{xs} \{x,y\}
apply(cases x=y)
apply(simp add: before_in_irefl)
proof –
assume xyset: \{x,y\} \subseteq set \text{xs}
assume dxs: \text{distinct} \text{xs}
assume xy: x\#y
{
  fix x y
  assume 1: \{x,y\} \subseteq set \text{xs}
  assume xny: x\#y
  assume 3: x < y in \text{xs}
  have Lxy \{x,y\} = [x,y] apply(rule Lxy_project)
    using xny 1 3 dxs by(auto)
  then have x < y in Lxy \{x,y\} using xny by(simp add: before_in_def)
} note aha=this
have \( a: x < y \) in \( xs \) \( \Rightarrow \) \( x < y \) in \( Lxy \) \( xs \) \( \{ x, y \} \)
apply(subst \( Lxy_project \))
using \( xy \) \( xyset \) \( dxs \) by(simp_all add: before_in_def)
have \( t: \{ x, y \} = \{ y, x \} \) by(auto)
have \( f: \sim x < y \) in \( xs \) \( \Rightarrow \) y < x in \( Lxy \) \( xs \) \( \{ x, y \} \)
unfolding \( t \)
apply(rule aha)
using \( xy \) \( xyset \) apply(simp)
using \( xy \) apply(simp)
using \( xy \) \( xyset \) by(simp add: not_before_in)
have \( b: \sim x < y \) in \( xs \) \( \Rightarrow \) \( \sim x < y \) in \( Lxy \) \( xs \) \( \{ x, y \} \)
proof
assume \( \sim x < y \) in \( xs \)
then have \( y < x \) in \( Lxy \) \( xs \) \( \{ x, y \} \)
using \( f \) by auto
then have \( \sim x < y \) in \( Lxy \) \( xs \) \( \{ x, y \} \)
using \( xy \) by(simp add: not_before_in)
then show \( \text{thesis} \).
qed
from \( a \) \( b \)
show \( \text{thesis} \) by metis
qed

6.4 List Update as Online/Offline Algorithm

type_synonym \( 'a \) state = \( 'a \) list
type_synonym answer = nat * nat list
definition step :: \( 'a \) state \( \Rightarrow \) \( 'a \) \( \Rightarrow \) answer \( \Rightarrow \) \( 'a \) state where
step s r a =
  (let \( (k,sws) \) = a in mtf2 k r (swaps sws s))
definition t :: \( 'a \) state \( \Rightarrow \) \( 'a \) \( \Rightarrow \) answer \( \Rightarrow \) nat where
t s r a = (let \( (mf,sws) \) = a in index (swaps sws s) r + 1 + size sws)
definition static where static s rs = (set rs \( \subseteq \) set s)
interpretation On_Off step t static .
type_synonym \( 'a \) alg_off = \( 'a \) state \( \Rightarrow \) \( 'a \) list \( \Rightarrow \) answer list
type_synonym ('a,'is) alg_on = ('a state,'is,'a,answer) alg_on

lemma T_ge_len: length as = length rs \( \Rightarrow \) T s rs as \( \geq \) length rs
by(induction arbitrary: s rule: list_induct2)
  (auto simp: l_def trans_le_add2)
lemma \( \text{T\_off\_neq} \): \( \forall rs s0. \text{size} (\text{alg} s0 rs) = \text{length} rs \iff rs \neq [] \Rightarrow \text{T\_off} \text{alg} s0 rs \neq 0 \)

apply (erule_tac \( x=rs \) in meta_allE)
apply (erule_tac \( x=s0 \) in meta_allE)
apply (auto simp: neq Nil_conv length Suc_conv t_def)
done

lemma \text{length\_step}[simp]: \( \text{length} (\text{step} s r as) = \text{length} s \)
by (simp add: step_def split_def)

lemma \text{step\_Nil\_iff}[simp]: \( \text{step} xs r act = [] \iff xs = [] \)
by (auto simp add: step_def mtf2_def split: prod.splits)

lemma \text{set\_step2}: \( \text{set} (\text{step} s r (\text{mf}, \text{sws})) = \text{set} s \)
by (auto simp add: step_def)

lemma \text{set\_step}: \( \text{set} (\text{step} s r \text{ act}) = \text{set} s \)
by (cases act) (simp add: set_step2)

lemma \text{distinct\_step}: \( \text{distinct} (\text{step} s r as) = \text{distinct} s \)
by (auto simp: step_def split_def)

6.5 Online Algorithm Move-to-Front is 2-Competitive

definition \text{MTF} :: \( \mathcal{O} (a, \text{unit}) \) \text{ alg\_on} \text{ where}
\( \text{MTF} = (\lambda(). (), \lambda s r. ((\text{size} (\text{fst} s) - 1, []), ())) \)

It was first proved by Sleator and Tarjan [?] that the Move-to-Front algorithm is 2-competitive.

lemma \text{potential}:
fixes \( t \) :: nat \Rightarrow \( a::\text{linordered\_ab\_group\_add} \) \text{ and} \( p \) :: nat \Rightarrow \( a \)
assumes \( p0 : p\ 0 = 0 \) \text{ and} \( \text{ppos} : \forall n. \ p\ n \geq 0 \)
and \( \text{ub} : \forall n. \ t\ n + p(n+1) - p\ n \leq u\ n \)
shows \( \sum i<n. \ t\ i \leq (\sum i<n. \ u\ i) \)
proof
  let \( ?a = \lambda n. \ t\ n + p(n+1) - p\ n \)
  have \( 1: (\sum i<n. \ t\ i) = (\sum i<n. \ ?a\ i) - p(n) \)
  by (induction \( n \)) (simp\_all add: \( p0 \))
  thus \( ?\text{thesis} \)
  by (metis (erased, lifting) add\_commute \text{diff\_add\_cancel} \text{le\_add\_same\_cancel2} \text{order\_trans} \text{ppos} \text{sum\_mono} \text{ub})
qed

lemma \text{potential2}:
fixes $t :: \text{nat} \Rightarrow 'a :: \text{linordered_ab_group_add}$ and $p :: \text{nat} \Rightarrow 'a$
assumes $p0: p \ 0 = 0$ and $\text{ppos} : \forall n. p \ n \geq 0$
and $\text{ub}: \forall m. m < n \implies t \ m + p(m+1) - p \ m \leq u \ m$
shows $(\sum i < n. t \ i) \leq (\sum i < n. u \ i)$

proof –
  let $\ ?a = \lambda n. t \ n + p(n+1) - p \ n$
  have $(\sum i < n. t \ i) = (\sum i < n. \ ?a \ i) - p(n)$ by (induction n) (simp_all add: p0)
  also have $\ldots \leq (\sum i < n. \ ?a \ i)$ using $\text{ppos}$ by auto
  also have $\ldots \leq (\sum i < n. u \ i)$ apply (rule sum_mono) apply (rule $\text{ub}$)
by auto
  finally show $\ ?\text{thesis}$. qed

abbreviation $\text{before} \ x \ xs \equiv \{ y. y < x \ \text{in} \ xs \}$
abbreviation $\text{after} \ x \ xs \equiv \{ y. x < y \ \text{in} \ xs \}$

lemma $\text{finite}\_\text{before}[\text{simp}]$: finite ($\text{before} \ x \ xs$)
apply (rule $\text{finite}\_\text{subset}[\text{where} \ B = \text{set} \ xs]$)
apply (auto dest: before_in_setD1)
done

lemma $\text{finite}\_\text{after}[\text{simp}]$: finite ($\text{after} \ x \ xs$)
apply (rule $\text{finite}\_\text{subset}[\text{where} \ B = \text{set} \ xs]$)
apply (auto dest: before_in_setD2)
done

lemma $\text{before}\_\text{conv}_\text{take}$:
$x : \text{set} \ xs \implies \text{before} \ x \ xs = \text{set} (\text{take} (\text{index} \ xs \ x) \ xs)$
by (auto simp add: before_in_def set_take_if_index index_le_size) (metis index_take leI)

lemma $\text{card}\_\text{before}$: distinct $xs \implies x : \text{set} \ xs \implies \text{card} (\text{before} \ x \ xs) = \text{index} \ xs \ x$
using index_le_size[of $xs$ $x$]
by (simp add: before_conv_take distinct_card[of $\text{distinct}\_\text{take}$] min_def)

lemma $\text{before}\_\text{Un}$: $\text{set} \ xs = \text{set} \ ys \implies x : \text{set} \ xs \implies$
before $x \ ys = \text{before} \ x \ xs \ \cap \text{before} \ x \ ys \ \text{Un} \ \text{after} \ x \ xs \ \cap \text{before} \ x \ ys$
by(auto)(metis before_in_setD1 not_before_in)

lemma $\text{phi}\_\text{diff}_\text{aux}$:
card ($\text{Inv} \ xs \ ys \ \cup$
\{(y, x) \mid y < x \text{ in } xs \land y < x \text{ in } ys\} - \\
\{(x, y) \mid y < x \text{ in } xs \land y < x \text{ in } ys\} = \\
card(Inv xs ys) + \card(before x xs \cap before x ys) \\
- \card(int(after x xs \cap before x ys)) \\
(is \ \card(?I \cup ?B - ?A) = \card ?I + \card ?b - \int(\card ?a)) \\

proof— \\
\textbf{have 1:} \ \?I \cap ?B = \{} \text{ by (auto simp: Inv_def, metis no_before_inI)} \\
\textbf{have 2:} \ ?A \subseteq ?I \cup ?B \text{ by (auto simp: Inv_def)} \\
\textbf{have 3:} \ ?A \subseteq ?I \text{ by (auto simp: Inv_def)} \\
\textbf{have \ int}(\card(?I \cup ?B - ?A)) = \int(\card ?I + \card ?B) - \int(\card ?A) \\
\textbf{using} \ \textbf{card_mono[OF \_ 3]} \\
\textbf{by (simp add: card_Un_disjoint[OF \_ 1] card_Diff_subset[OF \_ 2])} \\
\textbf{also have} \ \\textbf{card} ?B = \textbf{card} \ \textbf{fst} ' ?B \text{ by (auto simp: card_image inj_on_def)} \\
\textbf{also have} \ \\textbf{fst} ' ?B = ?b \text{ by force} \\
\textbf{also have} \ \\textbf{card} ?A = \textbf{card} \ \textbf{snd} ' ?A \text{ by (auto simp: card_image inj_on_def)} \\
\textbf{also have} \ \\textbf{snd} ' ?A = ?a \text{ by force} \\
\textbf{finally show} \ ?thesis . \\
qxq

\textbf{lemma} \ not_before_Conssimp: \ \neg x < y \text{ in } y \# xs \\
\textbf{by (simp add: before_in_def)} \\

\textbf{lemma} \ before_Conssimp: \\
y \in \text{set } xs \Rightarrow y \neq x \Rightarrow \text{before} y (x\#xs) = \text{insert } x \ (\text{before } y \text{ xs}) \\
\textbf{by (auto simp: before_in_def)} \\

\textbf{lemma} \ card_before_le_index: \ \text{card} \ \text{before} x xs \leq \text{index } xs \ x \\
\textbf{apply (cases } x \in \text{set } xs) \\
\textbf{prefer 2 apply (simp add: before_in_def)} \\
\textbf{apply (induction } xs) \\
\textbf{apply (simp add: before_in_def)} \\
\textbf{apply (auto simp: card_insert_if)} \\
\textbf{done} \\

\textbf{lemma} \ config_config_length: \ \text{length} \ \text{fst (config A init qs)} = \text{length } \text{init} \\
\textbf{apply (induct rule: config_induct) by (simp_all)} \\

\textbf{lemma} \ config_config_distinct: \\
\textbf{shows} \ \text{distinct (fst (config A init qs)) = distinct } \text{init} \\
\textbf{apply (induct rule: config_induct) by (simp_all add: distinct_step)} \\

\textbf{lemma} \ config_config_set: \\
\textbf{shows} \ \text{set (fst (config A init qs)) = set } \text{init} \\
\textbf{apply (induct rule: config_induct) by (simp_all add: set_step)}
lemma config_config:
  set (fst (config A init qs)) = set init
  ∧ distinct (fst (config A init qs)) = distinct init
  ∧ length (fst (config A init qs)) = length init
using config_config_distinct config_config_set config_config_length by metis

lemma config_dist_perm:
  distinct init ⇒ dist_perm (fst (config A init qs)) init
using config_config_distinct config_config_set by metis

lemma config_rand_length: ∀ x ∈ set_pmf (config_rand A init qs). length (fst x) = length init
apply (induct rule: config_rand_induct) by (simp_all)

lemma config_rand_distinct:
  shows ∀ x ∈ (config_rand A init qs). distinct (fst x) = distinct init
apply (induct rule: config_rand_induct) by (simp_all add: distinct_step)

lemma config_rand_set:
  shows ∀ x ∈ (config_rand A init qs). set (fst x) = set init
apply (induct rule: config_rand_induct) by (simp_all add: set_step)

lemma config_rand:
  ∀ x ∈ (config_rand A init qs). set (fst x) = set init
  ∧ distinct (fst x) = distinct init ∧ length (fst x) = length init
using config_rand_distinct config_rand_set config_rand_length by metis

lemma config_rand_dist_perm:
  distinct init ⇒ ∀ x ∈ (config_rand A init qs). dist_perm (fst x) init
using config_rand_distinct config_rand_set by metis

lemma amor_mtf_ub: assumes x : set ys set xs = set ys
shows int(card(before x xs Int before x ys)) − card(after x xs Int before x ys)
  ≤ 2 * int(index xs x) − card (before x ys) (is ?m − ?n ≤ 2 * ?j − ?k)
proof−
have 
\( xx_s \): \( x \in \text{set} \; xx_s \) using assms\((1,2)\) by simp

let \(?xx_s = \text{before} \; x \; xx_s \) let \(?xx_y = \text{before} \; x \; xx_y \) let \(?xx_s = \text{after} \; xx_s \)

have 0: \(?xx_s \cap \?xx_s = \{\}\) by (auto simp: before in def)

hence 1: \((?xx_s \cap \?xx_y) \cap (?xx_s \cap \?xx_y) = \{\}\) by blast

have \((?xx_s \cap \?xx_y) \cup (?xx_s \cap \?xx_y) = \?xx_y\) using assms\((2)\) before\_Un xx_s by fastforce

hence \(?m + \?n = \?k\)

using card\_Un\_disjoint[OF _ _ 1] by simp

hence \(?m - \?n = 2 * \?m - \?k\) by arith

also have \(?m \leq \?j\)

using card\_before\_le\_index[of xx_s] card\_mono[of \?xx_s, OF _ Int\_lower1]

by(auto intro: order\_trans)

finally show \(?\text{thesis}\) by auto

qed

locale MTF\_Off =

fixes as :: answer list
fixes rs :: 'a list
fixes s0 :: 'a list
assumes dist\_s0[simp]: distinct s0
assumes len\_as: length as = length rs

begin

definition mtf\_A :: nat list where
  mtf\_A = map fst as

definition sw\_A :: nat list list where
  sw\_A = map snd as

fun s\_A :: nat ⇒ 'a list where
  s\_A 0 = s0 |
  s\_A(Suc n) = step (s\_A n) (rs!n) (mtf\_A!n, sw\_A!n)

lemma length\_s\_A[simp]: length(s\_A n) = length s0
  by (induction n) simp\_all

lemma dist\_s\_A[simp]: distinct(s\_A n)
  by (induction n) (simp\_all add: step\_def)

lemma set\_s\_A[simp]: set(s\_A n) = set s0
  by (induction n) (simp\_all add: step\_def)

fun s\_mtf :: nat ⇒ 'a list where
\[
s_{\text{mtf}} 0 = s0 \\
s_{\text{mtf}} (\text{Suc} \ n) = \text{mtf} (\text{rs!}n) (s_{\text{mtf}} n)
\]

**definition** \( t_{\text{mtf}} :: \text{nat} \Rightarrow \text{int} \) where
\[
t_{\text{mtf}} n = \text{index} (s_{\text{mtf}} n) (\text{rs!}n) + 1
\]

**definition** \( T_{\text{mtf}} :: \text{nat} \Rightarrow \text{int} \) where
\[
T_{\text{mtf}} n = (\sum i < n. \ t_{\text{mtf}} i)
\]

**definition** \( c_{\text{A}} :: \text{nat} \Rightarrow \text{int} \) where
\[
c_{\text{A}} n = \text{index} (\text{swaps} (\text{sw}_{\text{A}}!n) (s_{\text{A}} n)) (\text{rs!}n) + 1
\]

**definition** \( f_{\text{A}} :: \text{nat} \Rightarrow \text{int} \) where
\[
f_{\text{A}} n = \text{min} (\text{mtf}_{\text{A}}!n) (\text{index} (\text{swaps} (\text{sw}_{\text{A}}!n) (s_{\text{A}} n)) (\text{rs!}n))
\]

**definition** \( p_{\text{A}} :: \text{nat} \Rightarrow \text{int} \) where
\[
p_{\text{A}} n = \text{size}(\text{sw}_{\text{A}}!n)
\]

**definition** \( t_{\text{A}} :: \text{nat} \Rightarrow \text{int} \) where
\[
t_{\text{A}} n = c_{\text{A}} n + p_{\text{A}} n
\]

**definition** \( T_{\text{A}} :: \text{nat} \Rightarrow \text{int} \) where
\[
T_{\text{A}} n = (\sum i < n. \ t_{\text{A}} i)
\]

**lemma** \( \text{length}_{s_{\text{mtf}}}[\text{simp}]: \text{length}(s_{\text{mtf}} n) = \text{length} s0 \)
by \((\text{induction} n)\) \(\text{simp\_all}\)

**lemma** \( \text{dist}_{s_{\text{mtf}}}[\text{simp}]: \text{distinct}(s_{\text{mtf}} n) \)
apply \((\text{induction} n)\)
apply \((\text{simp})\)
apply \((\text{auto} \ \text{simp}: \text{mtf\_def} \ \text{index\_take} \ \text{set\_drop\_if\_index})\)
apply \((\text{metis} \ \text{set\_drop\_if\_index} \ \text{index\_take} \ \text{less\_Suc\_eq\_le} \ \text{linear})\)
done

**lemma** \( \text{set}_{s_{\text{mtf}}}[\text{simp}]: \text{set} (s_{\text{mtf}} n) = \text{set} s0 \)
by \((\text{induction} n)\) \(\text{simp\_all}\)

**lemma** \( \text{dperm\_inv}: \text{dist\_perm} (s_{\text{A}} n) (s_{\text{mtf}} n) \)
by \((\text{metis} \ \text{dist}_{s_{\text{mtf}}} \ \text{dist}_{s_{\text{A}}} \ \text{set}_{s_{\text{mtf}}} \ \text{set}_{s_{\text{A}}})\)

**definition** \( \Phi :: \text{nat} \Rightarrow \text{int} (\Phi) \) where
\[
\Phi n = \text{card} (\text{Inv} (s_{\text{A}} n) (s_{\text{mtf}} n))
\]

**lemma** \( \phi0: \Phi 0 = 0 \)
by (simp add: Phi_def)

lemma phi_pos: Phi n ≥ 0
by (simp add: Phi_def)

lemma mtf_ub: t_mtf n + Phi (n+1) - Phi n ≤ 2 * c_A n - 1 + p_A n
- f_A n
proof -
  let ?xs = s_A n let ?ys = s_mtf n let ?x = rs!n
  let ?xs' = swaps (sw_A!n) ?xs let ?ys' = mtf ?x ?ys
  show ?thesis
proof cases
assume xin: ?x ∈ set ?ys
let ?bb = before ?x ?xs ∩ before ?x ?ys
let ?ab = after ?x ?xs ∩ before ?x ?ys
have phi_mtf:
  card (Inv ?xs' ?ys') - int (card (Inv ?xs' ?ys))
  ≤ 2 * int (index ?xs' ?x) - int (card (before ?x ?ys))
  using xin by (simp add: Inv_mtf phi_diff_aux amor_mtf_ub)
have phi_sw: card (Inv ?xs' ?ys) ≤ Phi n + length(sw_A!n)
proof -
  have int (card (Inv ?xs' ?ys)) ≤ card (Inv ?xs' ?xs) + int (card (Inv ?xs
  ?ys))
  also have card (Inv ?xs' ?xs) = card (Inv ?xs' ?xs')
    by (rule cardInv_sym)
  also have card (Inv ?xs' ?xs') ≤ size(sw_A!n)
    by (metis cardInv_swaps_le dist_s_A)
  finally show ?thesis by (fastforce simp: Phi_def)
qed

have phi_free: card (Inv ?xs' ?ys) - Phi (Suc n) = f_A n using xin
  by (simp add: Phi_def mtf2_def step_def cardInv_mtf2 index_less_size_conv
  f_A_def)
show ?thesis using xin phi_sw phi_mtf phi_free card_before[of s_mtf n]
  by (simp add: t_mtf_def c_A_def p_A_def)
next
assume notin: ?x ∉ set ?ys
have int (card (Inv ?xs' ?ys)) - card (Inv ?xs ?ys) ≤ card (Inv ?xs ?xs')
  using cardInv_tri_ineq[OF _ dperm_inv, of ?xs' n]
  swaps_inv[of sw_A!n s_A n]
  by (simp add: cardInv_sym)
also have ... ≤ size(sw_A!n)
  by (simp add: cardInv_swaps_le dperm_inv)
finally show ?thesis using notin
by\,(simp\ add:\ t_mtf_def\ step_def\ c_A_def\ p_A_def\ f_A_def\ Phi_def\ mtf2_def)\n
qed

qed

**Theorem Sleator\_Tarjan:** \( T_mtf\,n\leq(\sum\,i<n\cdot\,2\cdot c_A\,i\,+\,p_A\,i\,-\,f_A\,i)\,\) - \( n \)

**Proof** —

have \( (\sum\,i<n\cdot\,t_mtf\,i)\leq(\sum\,i<n\cdot\,2\cdot c_A\,i\,-\,1\,+\,p_A\,i\,-\,f_A\,i) \)

by\,(rule\ potential[where\ p=Phi,OF\ phi0\ phi_pos\ mtf_ub])

also\ have\ \ldots\ =\,(\sum\,i<n\cdot\,(2\cdot c_A\,i\,+\,p_A\,i\,-\,f_A\,i)\,-\,1)\)

by\,(simp\ add: algebra_simps)

also\ have\ \ldots\ =\,(\sum\,i<n\cdot\,2\cdot c_A\,i\,+\,p_A\,i\,-\,f_A\,i)\,-\,n\)

by\,(simp\ add: sumr_diff_mult_const2[symmetric])

finally\ show\ ?thesis\ by\,(simp\ add: T_mtf_def)

**Corollary Sleator\_Tarjan':** \( T_mtf\,n\leq2\cdot T_A\,n\,-\,n \)

**Proof** —

have \( T_mtf\,n\leq(\sum\,i<n\cdot\,2\cdot c_A\,i\,+\,p_A\,i\,-\,f_A\,i)\) - \( n \) by\,(fact\ Sleator\_Tarjan)

also\ have\ \( (\sum\,i<n\cdot\,2\cdot c_A\,i\,+\,p_A\,i\,-\,f_A\,i)\leq(\sum\,i<n\cdot\,2\cdot(c_A\,i\,+\,p_A\,i)\))\)

by\,(intro\ sum_mono)\,(simp\ add: p_A_def f_A_def)

also\ have\ \ldots\ \leq\,2\cdot T_A\,n\ by\,(simp\ add: sum_distrib_left T_A_def t_A_def)

finally\ show\ T_mtf\,n\leq2\cdot T_A\,n\,-\,n\ by\ auto

**Lemma T_A\_nneg:** \( 0\leq T_A\,n \)

by\,(auto\ simp\ add: sum_nonneg T_A_def t_A_def c_A_def p_A_def)

**Lemma T_mtf\_ub:** \( \forall\,i<n.\,rs!i\in\text{set}\,s0\implies T_mtf\,n\leq n\,*\,size\,s0 \)

**Proof** (induction \( n \))

- **case 0** show \( ?case \) by\,(simp\ add: T_mtf_def)

- **next**

  - **case (Suc \( n \))**

  - **using**\,index_less_size_conv[of\ \text{smtf}\,n\,rs!n]

  - **by**\,(simp\ add: T_mtf_def t_mtf_def less_Suc_eq del: index_less)

**Qed**

**Corollary T_mtf\_competitive:** assumes \( s0\neq[] \) and \( \forall\,i<n.\,rs!i\in\text{set}\,s0 \)

shows \( T_mtf\,n\leq(2\,-\,1/(size\,s0))\,*\,T_A\,n \)

**Proof** cases

- **assume 0:** \( \text{real_of_int}(T_A\,n)\leq n\,*\,(size\,s0) \)

  - **have** \( T_mtf\,n\leq2\,*\,T_A\,n\,-\,n \)

44
proof 
have $T_{mtf} n \leq (\sum_{i<n} 2^*c_A i + p_A i - f_A i) - n$ by (rule Sleator_Tarjan)
also have $(\sum_{i<n} 2^*c_A i + p_A i - f_A i) \leq (\sum_{i<n} (2^*(c_A i + p_A i)))$
by (intro sum_mono) (simp add: p_A_def f_A_def)
also have $\ldots \leq 2^* T_A n$ by (simp add: sum_distrib_left T_A_def t_A_def)
finally show ?thesis by simp
qed

hence $\text{real of int}(T_{mtf} n) \leq 2^* \text{of int}(T_A n) - n$ by simp
also have $\ldots = 2^* \text{of int}(T_A n) - (n * \text{size s0}) / \text{size s0}$
using assms(1) by simp
also have $\ldots \leq 2^* \text{real of int}(T_A n) - T_A n / \text{size s0}$
by (rule diff_left_mono[OF divide_right_mono[OF 0]]) simp
also have $\ldots = (2^* - 1 / \text{size s0}) * T_A n$ by algebra
finally show ?thesis .

next
assume 0: $\neg \text{real of int}(T_A n) \leq n * (\text{size s0})$
have $2^* - 1 / \text{size s0} \geq 1$ using assms(1)
by (auto simp add: field_simps neq_Nil_conv)
have $\text{real of int}(T_{mtf} n) \leq n * \text{size s0}$ using T_mtf_ub[OF assms(2)]
by linarith
also have $\ldots < \text{of int}(T_A n)$ using 0 by simp
also have $\ldots \leq (2^* - 1 / \text{size s0}) * T_A n$ using assms(1) T_A_nneg[of n]
by (auto simp add: mult_le_cancel_right1 field_simps neq_Nil_conv)
finally show ?thesis by linarith
qed

lemma t_A_t: $n < \text{length rs} \implies t_A n = \text{int} (t (s_A n) (\text{rs} ! n) (\text{as} ! n))$
by (simp add: t_A_def t_def c_A_def p_A_def sw_A_def len_as split: prod.split)

lemma T_A_eq_cm: $(\sum_{i=0..<\text{length rs}} t_A i) =$
$T (s_A 0) \ (\text{drop 0 rs}) \ (\text{drop 0 as})$
proof (induction rule: zero_induct[of _ size rs])
case 1 thus ?case by (simp add: len_as)
next
case (2 n)
show ?case
proof cases
assume n < length rs
thus ?case using 2
by (simp add: Cons_nth_drop_Suc[symmetric, where i=n] len_as sum_head_upt_Suc)
t_A t mtf_A_def sw_A_def)
next
assume ¬ n < length rs thus ?case by (simp add: len_as)
qed

lemma T_A_eq: T_A (length rs) = T s0 rs as
using T_A_eq_lem by(simp add: T_A_def atLeast0LessThan)

lemma nth_off_MTF: n < length rs ⟹ off2 MTF s rs ! n = (size(fst s) − 1,[])
by(induction rs arbitrary: s n)(auto simp add: MTF_def nth_Cons' Step_def)

lemma t_mtf_MTF: n < length rs ⟹
t_mtf n = int (t (s_mtf n) (rs ! n) (off MTF s rs ! n))
by(simp add: t_mtf_def t_def nth_off_MTF split: prod.split)

lemma mtf_MTF: n < length rs ⟹ length s = length s0 ⟹ mtf (rs ! n) s =
step s (rs ! n) (off MTF s0 rs ! n)
by(auto simp add: nth_off_MTF step_def mtf_eq_mtf2)

lemma T_mtf_eq_lem: (∑ i=0..<length rs. t_mtf i) =
T (s_mtf 0) (drop 0 rs) (drop 0 (off MTF s0 rs))
proof(induction rule: zero_induct[of _ size rs])
case 1 thus ?case by (simp add: len_as)
next
case (2 n)
show ?case
proof cases
  assume n < length rs
  thus ?case using 2
  by(simp add: Cons_nth_drop_Suc[symmetric,where i=n] len_as sum_head_upt_Suc
          t_mtf_MTF[where s=s0] mtf_A_def sw_A_def mtf_MTF)
next
  assume ¬ n < length rs thus ?case by (simp add: len_as)
qed

lemma T_mtf_eq: T_mtf (length rs) = T_on MTF s0 rs
using T_mtf_eq_lem by(simp add: T_mtf_def atLeast0LessThan)

corollary MTF_competitive2: s0 ≠ [] ⟹ ∀ i<length rs. rs!i ∈ set s0 ⟹
T_on MTF s0 rs ≤ (2 − 1/(size s0)) * T s0 rs as
by (metis T_mtf_competitive T_A_eq T_mtf_eq of_int_of_nat_eq)

corollary MTF_competitive': T_on MTF s0 rs ≤ 2 * T s0 rs as
using Sleator_Tarjan[of length rs] T_A_eq T_mtf_eq
by auto

end

theorem compet_MTF: assumes s0 ≠ [] distinct s0 set rs ⊆ set s0
shows T_on MTF s0 rs ≤ (2 − 1/(size s0)) * T_opt s0 rs
proof−
  from assms(3) have 1: ∀ i < length rs. rs!i ∈ set s0 by auto
  { fix as :: answer list assume len: length as = length rs
    interpret MTF_Off as rs s0 proof qed (auto simp: assms(2) len)
    from MTF_competitive2[OF assms(1) 1] assms(1)
    have T_on MTF s0 rs / (2 − 1 / (length s0)) ≤ of_int(T s0 rs as)
      by (simp add: field_simps length_greater_0_conv[symmetric]
        del: length_greater_0_conv)
  } hence T_on MTF s0 rs / (2 − 1/(size s0)) ≤ T_opt s0 rs
  apply(simp add: T_opt_def Inf_def Nat_def)
  apply(rule LeastI2_wellorder)
  using length_replicate[of length rs undefined] apply fastforce
  apply auto
  done
  thus ?thesis using assms by (simp add: field_simps
    length_greater_0_conv[symmetric] del: length_greater_0_conv)
qed

theorem compet_MTF': assumes distinct s0
shows T_on MTF s0 rs ≤ (2::real) * T_opt s0 rs
proof−
  { fix as :: answer list assume len: length as = length rs
    interpret MTF_Off as rs s0 proof qed (auto simp: assms(1) len)
    from MTF_competitive'
    have T_on MTF s0 rs / 2 ≤ of_int(T s0 rs as)
      by (simp add: field_simps length_greater_0_conv[symmetric]
        del: length_greater_0_conv)
  } hence T_on MTF s0 rs / 2 ≤ T_opt s0 rs
  apply(simp add: T_opt_def Inf_def Nat_def)
  apply(rule LeastI2_wellorder)
  using length_replicate[of length rs undefined] apply fastforce
  apply auto
  done
  thus ?thesis using assms by (simp add: field_simps
  ...
**6.6 Lower Bound for Competitiveness**

This result is independent of MTF but is based on the list update problem defined in this theory.

**Lemma  rat_fun_lem:**

- **fixes** \( l, c :: \text{real} \)
- **assumes** \([\text{simp}]: F \neq \text{bot}\)
- **assumes** \( 0 < l \)
- **assumes** \( \text{ev}: \) \( \lambda n. l \leq f n / g n \) \( F \)
- **and** \( \text{g: LIM} n F. \ g n \triangleright \text{at_top} \)
- **shows** \( l \leq u \)

**Proof** (rule dense_le_bounded[OF \( 0 < l \)])

- **fix** \( x \) **assume** \( 0 < x < l \)
- **define** \( m \) **where** \( m = (x - l) / 2 \)
- **define** \( k \) **where** \( k = l / (x - m) \)
- **have** \( x = l / k + m \) \( 1 < k m < 0 \)
  - **unfolding** \( k\text{-def} m\text{-def} \) **using** \( x \) **by** (auto simp: divide_simps)
  - **from** \( 1 < k \) **have** \( \text{LIM} n F. (k - 1) * g n \triangleright \text{at_top} \)
    - **by** (intro filterlim_tends_to_pos_mult_at_top[OF filterlim_const_\( g \)]) (simp add: field_simps)
  - **then have** \( \text{eventually} (\lambda n. d \leq (k - 1) * g n) \) \( F \)
    - **by** (simp add: filterlim_at_top)
  - **moreover have** \( \text{eventually} (\lambda n. 1 \leq g n) \) \( F \) \( \text{eventually} (\lambda n. 1 - d \leq g n) \) \( F \)
    - **using** \( g \) **by** (auto simp add: filterlim_at_top)
  - **ultimately have** \( \text{eventually} (\lambda n. x \leq u) \) \( F \)
    - **using** \( \text{ev} \)
- **proof** \( \text{eventuallyelim} \)
  - **fix** \( n \) **assume** \( d: d \leq (k - 1) * g n \) \( 1 \leq g n 1 - d \leq g n c / m - d \)
    - **\leq g n**
      - **and** \( l: l \leq f n / g n \) **and** \( u: (f n + c) / (g n + d) \leq u \)
      - **from** \( d \) **have** \( g n + d \leq k * g n \)
        - **by** (simp add: field_simps)
from \(d\) have \(g\): \(0 < g n 0 < g n + d\)
by (auto simp: field_simps)
with \(0 < l\) have \(0 < f n\)
by (auto simp: field_simps intro: mult_pos_pos less_le_trans)

note \(x = l / k + m\)
also have \(l / k \leq f n / (k * g n)\)
using \(l \langle 1 < k\rangle\) by (simp add: field_simps)
also have \(\ldots \leq f n / (g n + d)\)
using \(d \langle 1 < k\rangle \langle 0 < f n\rangle\) by (intro divide_left_mono mult_pos_pos)
(also simp: field_simps intro: mult_pos_pos)
(also have \(m \leq c / (g n + d)\)
using \(\langle c / m - d \leq g n\rangle \langle 0 < g n + d\rangle \langle m < 0\rangle\) by (simp add: field_simps)
(also note \(u\)
finally show \(x \leq u\) by simp
qed
then show \(x \leq u\) by auto
qed

lemma compet_lb0:
fixes \(a\) \(A\) \(A'\) \(c\) \(S\)\(0\)
defines \(f s0 rs\) == \(\text{real}(T_{\text{on}} A s0 rs)\)
defines \(g s0 rs\) == \(\text{real}(T_{\text{off}} A' s0 rs)\)
assumes \(\forall rs s0. \text{size}(A' s0 rs) = \text{length}\ rs\) and \(\forall n. \text{cruel}\ n \neq []\)
assumes \(\text{compet}\ A\ c\ S\ 0\) and \(c \geq 0\) and \(s0 \in S\)
and \(l:\ \text{eventually}\ (\lambda n. f s0 (\text{cruel}\ n)) / (g s0 (\text{cruel}\ n) + a) \geq l)\) sequentially
and \(g:\ T_{\text{lim}} n\ \text{sequentially}.\ g s0 (\text{cruel}\ n) :> \text{at}\_\text{top}\)
and \(l > 0\) and \(\forall n. \text{static}\ s0 (\text{cruel}\ n)\)
shows \(l \leq c\)
proof
let \(\lambda h. \lambda b s0 rs. (f s0 rs - b) / g s0 rs\)
have \(g:\ T_{\text{lim}} n\ \text{sequentially}.\ g s0 (\text{cruel}\ n) + a :> \text{at}\_\text{top}\)
using filterlim_tendsto_add_at_top[OF tendsto_const g]
by (simp add: ac_simps)
from competE[OF assms(5) \(c \geq 0\) \((s0 \in S)\)\(0\)]\(assms(3)\) obtain \(b\) where
\(\forall rs. \text{static}\ s0 rs \land rs \neq []\) \(\rightarrow\ \lambda h. b s0 rs \leq c\)
by (fastforce simp del: neq0_conv simp: neq0_conv[symmetric]
field_simps intro: neq0_conv[where g=\(g\), OF assms(4)]
field_simps intro: neq0_conv[where g=\(g\), OF assms(3)]
)
\hence \(\forall n. (\lambda h. b s0 o \text{cruel}) n \leq c\) using assms(4,11) by simp
\(\forall rs. \text{static}\ s0 rs \land rs \neq []\)
\(\rightarrow\ \lambda h. b s0 rs \leq c\)
by (fastforce simp del: neq0_conv simp: neq0_conv[symmetric]
field_simps intro: neq0_conv[where g=\(g\), OF assms(4)]
field_simps intro: neq0_conv[where g=\(g\), OF assms(3)]
)
\hence \(\forall n. (\lambda h. b s0 o \text{cruel}) n \leq c\) using assms(4,11) by simp
\(\forall rs. \text{static}\ s0 rs \land rs \neq []\)
\(\rightarrow\ \lambda h. b s0 rs \leq c\)
by (fastforce simp del: neq0_conv simp: neq0_conv[symmetric]
field_simps intro: neq0_conv[where g=\(g\), OF assms(4)]
field_simps intro: neq0_conv[where g=\(g\), OF assms(3)]
)
Sorting

fun ins_sws where
  ins_sws k x [] = [] |
  ins_sws k x (y#ys) = (if k x ≤ k y then [] else map Suc (ins_sws k x ys) @ [0])

fun sort_sws where
  sort_sws k [] = [] |
  sort_sws k (x#xs) =
    ins_sws k x (sort_key k xs) @ map Suc (sort_sws k xs)

lemma length_ins_sws: length(ins_sws k x xs) ≤ length xs
by(induction xs) auto

lemma length_sort_sws_le: length(sort_sws k xs) ≤ length xs ^ 2
proof(induction xs)
  case (Cons x xs) thus ?case
  using length_ins_sws[of k x sort_key k xs] by (simp add: numeral_eq_Suc)
qed simp

lemma swaps_ins_sws:
  swaps (ins_sws k x xs) (x#xs) = insort_key k x xs
by(induction xs)(auto simp: swap_def[of 0])

lemma swaps_sort_sws[simp]:
  swaps (sort_sws k xs) xs = sort_key k xs
by(induction xs)(auto simp: swaps_ins_sws)

The cruel adversary:

fun cruel :: ('a,'is) alg_on => 'a state * 'is => nat => 'a list where
  cruel A s 0 = [] |
  cruel A s (Suc n) = last (fst s) # cruel A (Step A s (last s)) n

definition adv :: ('a,'is) alg_on => ('a::linorder) alg_off where
  adv A s rs = (if rs=[]) then [] else
    let crs = cruel A (Step A s (fst A s) (last s)) (size rs - 1)
    in (0,sort_sws (λx. size rs - 1 - count_list crs x) s) # replicate (size rs - 1) (0,[],])

lemma set_cruel: s ≠ [] => set(cruel A (s,is)) ≺ set s
apply (induction n arbitrary: s is)
apply (auto simp: step_def Step_def split: prod.split)
by (metis empty_iff swaps_inv last_in_set list.set(1) rev_subsetD set.mtf2)

lemma static_cruel: s ≠ []  ⇒  static s (cruel A (s,is) n)
by (simp add: set_cruel static_def)

lemma T_cruel:
  s ≠ []  ⇒  distinct s  ⇒
  T s (cruel A (s,is) n) (off2 A (s,is) (cruel A (s,is) n)) ≥ n*(length s)
apply (induction n arbitrary: s is)
apply (simp)
apply (erule_tac x = fst (Step A (s, is) (last s)) in meta_allE)
apply (erule_tac x = snd (Step A (s, is) (last s)) in meta_allE)
apply (erule_tac x = (snd (snd A (s, is) (last s))) in index_swaps_last_size)
apply (simp add: distinct_step t_def split_def Step_def length_greater_0_conv[symmetric] del: length_greater_0_conv)
done

lemma length_cruel[simp]: length (cruel A s n) = n
by (induction n arbitrary: s) (auto)

lemma t_sort_swss: t s r (mf, sort_swss k s) ≤ size s ^ 2 + size s + 1
using length_sort_swss_le[of k s] index_le_size[of sort_key k s r]
by (simp add: t_def add_mono index_le_size algebra_simps)

lemma T noop:
  n = length rs  ⇒  T s rs (replicate n (0, [])) = (∑ i→rs. index s r + 1)
by (induction rs arbitrary: s n)(auto simp: t_def step_def)

lemma sorted_asc: j≤i  ⇒  i<size ss  ⇒  ∀ x ∈ set ss. ∀ y ∈ set ss. k(x) ≤ k(y)  ⇒  f y ≤ f x
  ⇒  sorted (map k ss)  ⇒  f (ss ! i) ≤ f (ss ! j)
by (auto simp: sorted_iff_nth_mono)

lemma sorted_weighted_gauss_Ico_div2:
fixes f :: nat ⇒ nat
assumes ∃ i j. i ≤ j  ⇒  j < n  ⇒  f i ≥ f j
shows (∑ i=0..<n. (i + 1) * f i) ≤ (n + 1) * sum f {0..<n} div 2
proof (cases n)
case 0
then show \(?thesis\)  
  by simp

next
  case (Suc \(n\))
  with assms have Suc \(n\) \((\sum i=0..<Suc \(n\). Suc \(i\) \(* f \(i\)) \leq (\sum i=0..<Suc \(n\). Suc \(i\)) \(* \sum f \{0..<Suc \(n\}\}\))
    by (intro Chebyshev_sum_upper_nat \[\of Suc \(n\) Suc f\]) auto
  then have Suc \(n\) \((2 \(*(\sum i=0..\(n\). Suc \(i\) \(* f \(i\)) \leq (\sum i=0..\(n\). Suc \(i\)) \(* \sum f \{0..\(n\)\}\)\))
    by (simp add: atLeastLessThanSuc atLeastAtMost)
  also have Suc \(n\) \((2 \(*(\sum i=0..\(n\). Suc \(i\) \(* f \(i\)) \leq (n+2) \(* \sum f \{0..\(n\)\}\)\))
    by (simp only: ac_simps Suc mult le cancel1)
  finally have Suc \(n\) \((2 \(*(\sum i=0..\(n\). Suc \(i\) \(* f \(i\)) \leq (n+2) \(* \sum f \{0..\(n\)\}\)\))
    by (simp only: atLeastLessThanSuc atLeastAtMost)
  qed

lemma T_adv: assumes \(l \neq 0\)
  shows \(T_{off} \(\text{adv } A\) \([0..<\(l\)]\) \((\text{cruel } A \([0..<\(l\)]\), \text{fst } A \([0..<\(l\)]\) \((\text{Suc } n)\)) \leq l^2 + l + 1 + (l + 1) \(* n \div 2\) \((\text{is } ?l \leq ?r)\)

proof–
  let \(?s = [0..<\(l\)]\)
  let \(?r = \text{last } ?s\)
  let \(?S' = \text{Step } A \(\text{?s, fst } A \text{?s}\) \(?r\)\)
  let \(?s' = \text{fst } \?S'\)
  let \(?cr = \text{cruel } A \text{?S'} \text{?n}\)
  let \(?c = \text{count_list } ?cr\)
  let \(?k = \lambda x. n - \text{?c } x\)
  let \(?sort = \text{sort_key } ?k \text{?s}\)
  have 1: set \(?s' = \{0..<\(l\}\)\)
    by (simp add: set_step Step_def split: prod.split)
  have 3: \(\forall x. x < \(l\) \implies \text{?c } x \leq n\)
    by (simp) (metis count_le_length length_cruel)
  have \(?l = t \?s \text{ (last } ?s) \((\theta, \text{sort_swss } ?k \text{?s}) + (\sum x\in\text{set } ?s', \text{?c } x \*(\text{index } \text{?sort } x + 1))\)\)
    using assms
    apply (simp add: adv_def T_noop sum_list_map_eq_sum_count2[OF set_cruel]
    Step_def
      split: prod.split)
    apply (subst (3) step_def)
    apply (simp)
    done
  also have \((\sum x\in\text{set } ?s', \text{?c } x \*(\text{index } \text{?sort } x + 1)) = (\sum x\in\{0..<\(l\}\}. \text{?c}\)

52
\[
x \ast (\text{index } ?\text{sort } x + 1))
\]
by (simp add: 1)
also have \[
\ldots = (\sum x \in \{0..\lt l\}. \ ?c (\ ?\text{sort} ! x) \ast (\text{index } ?\text{sort} (\ ?\text{sort} ! x) + 1))
\]
by (rule sum.reindex_bij_betw [where \ ?h = nth \ ?\text{sort}, symmetric])
(simp add: bij_betw_imageI inj_on_nth nth_image)
also have \[
\ldots = (\sum x \in \{0..\lt l\}. \ ?c (\ ?\text{sort} ! x) \ast (x + 1))
\]
by (simp add: index_nth_id)
also have \[
\ldots \leq (\sum x \in \{0..\lt l\}. (x + 1) \ast ?c (\ ?\text{sort} ! x))
\]
by (simp add: algebra_simps)
also (ord_eq_le_subst) have \[
\ldots \leq (l + 1) \ast (\sum x \in \{0..\lt l\}. ?c (\ ?\text{sort} ! x))
\]
div 2
apply (rule sorted.weighted_gauss_ico_div2)
apply (erule sorted_asc [where \ k = \ λx. n - \ count_list (cruel A ?S' n) x])
apply (auto simp add: index_nth_id dest!: 3)
using assms [[linarith_split_limit = 20]] by simp
also have \[
(\sum x \in \{0..\lt l\}. ?c (\ ?\text{sort} ! x)) = (\sum x \in \{0..\lt l\}. ?c (\ ?\text{sort} ! x))
\]
by (rule sum.reindex_bij_betw [where \ ?h = index \ ?\text{sort}, symmetric])
(simp add: bij_betw_imageI inj_on_index2 index_image)
also have \[
\ldots = (\sum x \in \{0..\lt l\}. ?c x)
\]
by (simp)
also have \[
\ldots = \text{length } ?cr
\]
using set_cruel [of ?s' A n] assms 1
by (auto simp add: sum_count_set Step_def split: prod.split)
also have \[
\ldots = n
\]
by simp
also have \[
t \ ?s (\last \ ?s) (0, \ sort_sws \ ?k \ ?s) \leq (\text{length } ?s) \ast 2 + \text{length } ?s + 1
\]
by (rule t_sort_sw)
also have \[
l' \ast 2 + l + 1
\]
by simp
finally show \[
?l \leq l' \ast 2 + l + 1 + (l + 1) \ast n \div 2
\]
by auto
qed

The main theorem:

**Theorem** compet_lb2:

**Assumes**
- \text{compet} \ A \ c \ \{xs::nat list. \ size \ xs = \ l\} \ and \ l \neq \ 0 \ and \ c \geq \ 0

**Shows**
- \(c \geq 2 \ast l / (l+1)\)

**Proof**
- (rule \text{compet_lb0} [OF \ \_ \ assms(1) \ (c\geq0)])
- let \(\?S0 = \{xs::nat list. \ size \ xs = \ l\}\)
- let \(\?s0 = \{0..\lt l\}\)
- let \(\text{cruel} = \text{cruel} \ A \ (\?s0, \text{fst} \ A \ ?s0) \ o \ Suc\)
- let \(\text{on} = \lambda n. \ T_{on} \ A \ ?s0 \ (\text{cruel} \ n)\)
- let \(\text{off} = \lambda n. \ T_{off} (\text{adv} \ A) \ ?s0 \ (\text{cruel} \ n)\)
- show \(\forall \?s0 \ rs. \ \text{length} (\text{adv} \ A \ ?s0 \ rs) = \text{length} \ rs\) by (simp add: adv_def)
show \( \forall n. \neg \text{cruel } n \neq [] \) by auto

show \( \exists \theta \in \theta S0 \) by simp

\{ fix Z::real and n::nat assume n \geq \text{nat(ceiling } Z) \\
  have \theta \text{off } n \geq \text{length(} \neg \text{cruel } n \text{)} \text{by (rule T_ge_len) (simp add: adv_def)} \\
  hence \theta \text{off } n > n \text{ by simp} \\
  hence Z \leq \theta \text{off } n \text{ using } n \geq \text{nat(ceiling } Z) \text{ by linarith} \} \\
thus LIM n \text{ sequentially. real (} \theta \text{off } n \text{)} :> \text{ at_top} \\
  by (auto simp only: filterlim_at_top_eventually_sequentially)

let \( \alpha = - (l^2 + l + 1) \)

\{ fix n assume n \geq \lceil 2 \rceil + l + 1 \\
  have \( 2*l/(l+1) = 2*l*(n+1) / ((l+1)*(n+1)) \) \\
    by (simp del: One_nat_def) \\
  also have \( \ldots = 2*\text{real}(l*(n+1)) / ((l+1)*(n+1)) \) \text{ by simp} \\
  also have \( l * (n+1) \leq \alpha \text{on n} \) \\
    using T_cruel[of \( \exists \theta S0 \text{ Suc } n \) | \( l \neq 0 \)] \\
    by (simp add: ac_simps) \\
  also have \( 2*\text{real}(\alpha \text{on } n) / ((l+1)*(n+1)) \leq 2*\text{real}(\alpha \text{on } n)/(2*(\theta \text{off } n + \alpha)) \)

proof – \\
  have \( 0: \exists \text{real}(\alpha \text{on } n) \geq 0 \) \text{ by simp} \\
  have \( 1: 0 < \text{real } ((l + 1) * (n + 1)) \) \text{ by (simp del: of_nat_Suc)} \\
  have \( \theta \text{off } n \geq \text{length(} \neg \text{cruel } n \text{)} \) \\
    by (rule T_ge_len) (simp add: adv_def) \\
  hence \theta \text{off } n > n \text{ by simp} \\
  hence \( \theta \text{off } n + \alpha > 0 \text{ using } n \geq \lceil 2 \rceil + l + 1 \text{ by linarith} \) \\
  hence \( 2: \text{real of_int}(2*(\theta \text{off } n + \alpha)) > 0 \) \\
    by (simp only: of_int_0_less_iff zero_less_mult_iff zero_less_numeral simp_thms) \\
  have \( \theta \text{off } n + \alpha \leq (l+1)*(n) \text{ div } 2 \) \\
    using T_adv[OF \( |l|\neq0, \text{ of } A \text{n} \)] \\
    by (simp only: a_apply of_nat_add of_nat_le_iff) \\
  also have \( \ldots \leq (l+1)*(n+1) \text{ div } 2 \) \text{ by (simp)} \\
  finally have \( 2*(\theta \text{off } n + \alpha) \leq (l+1)*(n+1) \) \\
    by (simp add: zdiv_int) \\
  hence \( \text{of_int}(2*(\theta \text{off } n + \alpha)) \leq \text{real((l+1)*(n+1))) \) \text{ by (simp only: of_int_le_iff)} \\
  from divide_left_mono[OF this 0 mult_pos_pos[OF 1 2]] show \( \text{thesis} \).

qed \\
also have \( \ldots = \alpha \text{on } n / (\theta \text{off } n + \alpha) \) \\
    by (simp del: distrib_left_numeral One_nat_def cruel.simps) \\
  finally have \( 2*l/(l+1) \leq \alpha \text{on } n / (\text{real(} \theta \text{off } n + \alpha)) \) \\
    by (auto simp: divide_right_mono) \}

thus eventually (\( \lambda n. (2 * l) / (l + 1) \leq \alpha \text{on } n / (\text{real(} \theta \text{off } n + \alpha)) \))
sequentially
  by (auto simp add: filterlim_at_top eventually_sequentially)
show 0 < 2*l / (l+1) using l \neq 0 by (simp)
s
show \land. static ?s0 (?cruel n) using l \neq 0 by (simp add: static_cruel del: cruel.simps)
qed

theory Bit_Strings
imports Complex_Main
begin

7 Lemmas about BitStrings and sets thereof

7.1 the set of bitstring of length m is finite

lemma bitstrings_finite: finite {xs::bool list. length xs = m} using finite_lists_length_eq[where A=UNIV] by force

7.2 how to calculate the cardinality of the set of bitstrings with certain bits already set

lemma fbool: finite {xs. (\forall i\in X. xs ! i) \land (\forall i\in Y. \neg xs ! i) \land length xs = m \land f (xs!e)}
  by (rule finite_subset[where B={xs. length xs = m}])
       (auto simp: bitstrings_finite)

fun witness :: nat set \Rightarrow nat \Rightarrow bool list where

witness X 0 = []
|witness X (Suc n) = (witness X n) @ [n \in X]

lemma witness_length: length (witness X n) = n
apply (induct n) by auto

lemma iswitness: r<n \implies ((witness X n)!r) = (r\in X)
proof (induct n)
  case (Suc n)

  have witness X (Suc n) ! r = ((witness X n) @ [n \in X]) ! r by simp
  also have \ldots = (if r < length (witness X n) then (witness X n) ! r else [n \in X] ! (r - length (witness X n))) by (rule nth_append)
  also have \ldots = (if r < n then (witness X n) ! r else [n \in X] ! (r - n))
by (simp add: witness_length)

finally have \(1\): witness \(X \ (\text{Suc } n) \ ! \ r = (\text{if } r < n \text{ then } (\text{witness } X \ n) \ ! \ r \ \text{else} \ [n \in X] \ ! (r - n))\).

show \(?\)case
proof (cases \(r < n\))
  case True
  with \(1\) have \(a:\) witness \(X \ (\text{Suc } n) \ ! \ r = (\text{witness } X \ n) \ ! \ r\) by auto
  from Suc True have \(b:\) witness \(X \ n \ ! \ r = (r \in X)\) by auto
  from \(a\) \(b\) show \(?\)thesis by auto
next
  case False
  with Suc have \(r = n\) by auto
  with \(1\) show witness \(X \ (\text{Suc } n) \ ! \ r = (r \in X)\) by auto
qed
qed simp

lemma card1: finite \(S\) \(\implies\) finite \(X\) \(\implies\) finite \(Y\) \(\implies\) \(X \cap Y = \{\} \implies S \cap (X \cup Y) = \{\} \implies S \cup (X \cup Y) = \{\_..<\_\} \implies\)
  card \(\{xs. (\forall i \in X. xs ! i) \land (\forall i \in Y. \neg xs ! i) \land length xs = m\} = 2^m - \text{card } X - \text{card } Y\)
proof (induct arbitrary: \(X \ Y\) rule: finite_induct)
  case empty
  then have \(x:\) \(X \subseteq \{\_..<\_\}\) and \(y:\) \(Y \subseteq \{\_..<\_\}\) and \(xy: X \cup Y = \{\_..<\_\}\) by auto
  then have card \((X \cup Y) = m\) by auto
  with empty(3) have cardXY: card \(X + card Y = m\) using \(\text{card, Un, Int, OF empty(1) empty(2)}\) by auto
from empty have ents: \(\forall i < m. (i \in Y) = (i \notin X)\) by auto

have \((\exists! w. (\forall i \in X. w ! i) \land (\forall i \in Y. \neg w ! i) \land length w = m)\)
proof (rule ex1I, goal_cases)
  case 1
  show \((\forall i \in X. (\text{witness } X m) ! i) \land (\forall i \in Y. \neg (\text{witness } X m) ! i) \land length (\text{witness } X m) = m)\)
  proof (safe, goal_cases)
    case 2 \(i\)
    with \(y\) have \(a: i < m\) by auto
    with iswitness have \(\text{witness } X m \ ! i = (i \in X)\) by auto
    with \(a\) ents have \(~\text{witness } X m \ ! i\) by auto
    with 2(2) show \(False\) by auto
next

56
case (1 i)
  with x have a: i < m by auto
  with iswitness have witness X m ! i = (i ∈ X) by auto
  with a ents 1 show witness X m ! i by auto
qed (rule witness_length)

next
  case (2 w)
  show w = witness X m
  proof
    have (length w = length (witness X m) ∧ (∀i<length w. w ! i =
      (witness X m) ! i))
      using 2 apply(simp add: witness_length)
    proof
      fix i
      assume as: (∀i∈X. w ! i) ∧ (∀i∈Y. ¬ w ! i) ∧ length w = m
      have i < m −→ (witness X m) ! i = (i ∈ X) using iswitness by
        auto
      then show i < m −→ w ! i = (witness X m) ! i using ents as by
        auto
    qed
    then show ?thesis using list_eq_iff_nth_eq by auto
    qed
    qed
  then obtain w where {xs. Ball X ((!) xs) ∧ (∀i∈Y. ¬ xs ! i) ∧ length
    xs = m}
    = { w } using Nitpick.Ex1_unfold[where P=(λxs. Ball X ((!) xs)
    ∧ (∀i∈Y. ¬ xs ! i) ∧ length xs = m)]
    by auto
  then have card {xs. Ball X ((!) xs) ∧ (∀i∈Y. ¬ xs ! i) ∧ length xs =
    m} = card { w } by auto
  also have ... = 1 by auto
  also have ... = 2^(m − card X − card Y) using cardXY by auto
  finally show ?case .
next
  case (insert e S)
  then have eX: e ∉ X and eY: e ∉ Y by auto
  from insert(8) have insert e S ⊆ {0..<m} by auto
  then have ebetween0m: e∈{0..<m} by auto

  have fm: finite {0..<m} by auto
  have cardm: card {0..<m} =  m by auto
  from insert(8) eX eY ebetween0m have sub: X ∪ Y ⊆ {0..<m} by auto
  from insert have card (X ∩ Y) = 0 by auto
then have cardXY: card \((X \cup Y)\) = card \(X\) + card \(Y\) using card_Un_Int[\(OF insert(4) insert(5)\)] by auto

have \(m > card X + card Y\) using psubset_card_mono[\(OF fm sub\) cardm cardXY by(auto)
then have carde: \(1 + (m - card X - card Y - 1) = m - card X - card Y\) by auto

have is1: \(\{xs. Ball X ((!) xs) \land (\forall i \in Y. \neg xs ! i) \land length xs = m \land xs!e\}\)
= \(\{xs. Ball (insert e X) ((!) xs) \land (\forall i \in Y. \neg xs ! i) \land length xs = m\}\) by auto

have is2: \(\{xs. Ball X ((!) xs) \land (\forall i \in Y. \neg xs ! i) \land length xs = m \land \neg xs!e\}\)
= \(\{xs. Ball X ((!) xs) \land (\forall i \in (insert e Y). \neg xs ! i) \land length xs = m\}\) by auto

have 2: \(\{xs. Ball X ((!) xs) \land (\forall i \in Y. \neg xs ! i) \land length xs = m \land xs!e\}\)
\(\cup \{xs. Ball X ((!) xs) \land (\forall i \in Y. \neg xs ! i) \land length xs = m \land \neg xs!e\}\)
= \(\{xs. Ball X ((!) xs) \land (\forall i \in Y. \neg xs ! i) \land length xs = m\}\) by auto

have 3: \(\{xs. Ball X ((!) xs) \land (\forall i \in Y. \neg xs ! i) \land length xs = m \land xs!e\}\)
\(\cap \{xs. Ball X ((!) xs) \land (\forall i \in Y. \neg xs ! i) \land length xs = m \land \neg xs!e\}\)
= \(\{\}\) by auto

have fx: finite (insert e X)
and disjXY: insert e X \cap Y = \{\}
and cutx: \(S \cap (insert e X \cup Y) = \{\}\)
and uniX: \(S \cup insert e X \cup Y = \{0..<m\}\) using insert by auto

have fy: finite (insert e Y)
and disjXeY: \(X \cap (insert e Y) = \{\}\)
and cutY: \(S \cap (X \cup insert e Y) = \{\}\)
and uniY: \(S \cup X \cup insert e Y = \{0..<m\}\) using insert by auto

have card \(\{xs. Ball X ((!) xs) \land (\forall i \in Y. \neg xs ! i) \land length xs = m\}\)
= \(\{xs. Ball X ((!) xs) \land (\forall i \in Y. \neg xs ! i) \land length xs = m \land xs!e\}\)
\(+ \{xs. Ball X ((!) xs) \land (\forall i \in Y. \neg xs ! i) \land length xs = m \land \neg xs!e\}\)
apply(subst card_Un_Int)
apply(rule fbool) apply(rule fbool) using 2 3 by auto

58
also
have \ldots = \operatorname{card} \{xs. \operatorname{Ball} (\text{insert } e X) (\bang xs) \land (\forall i \in Y. \neg xs ! i) \land \text{length } xs = m\} + \operatorname{card} \{xs. \operatorname{Ball} X (\bang xs) \land (\forall i \in (\text{insert } e Y). \neg xs ! i) \land \text{length } xs = m\} \text{ by (simp only: is1 is2)}

also
have \ldots = 2 ^ \wedge (m - \operatorname{card} (\text{insert } e X) - \operatorname{card} Y)
+ 2 ^ \wedge (m - \operatorname{card} X - \operatorname{card} (\text{insert } e Y))
\text{ apply (simp only: insert(3)[of insert } e X Y, OF } fX \text{ insert(5) disj} e X y \text{ cut} X y \text{ uni} X)\)
by (simp only: insert(3)[of insert } e Y, OF insert(4) fY disj} e X y \text{ cut} Y y \text{ uni} Y)\)

also
have \ldots = 2 ^ \wedge (m - \operatorname{card} X - \operatorname{card} Y - 1)
+ 2 ^ \wedge (m - \operatorname{card} X - \operatorname{card} Y - 1) \text{ using insert(4,5) } e X e Y \text{ by auto}

also
have \ldots = 2 \times 2 ^ \wedge (m - \operatorname{card} X - \operatorname{card} Y - 1) \text{ by auto}
also have \ldots = 2 ^ \wedge (1 + (m - \operatorname{card} X - \operatorname{card} Y - 1)) \text{ by auto}
also have \ldots = 2 ^ \wedge (m - \operatorname{card} X - \operatorname{card} Y) \text{ using carde by auto}
finally show ?case .

qed

lemma \text{card2}: assumes finite X and finite Y and X \cap Y = \{\} and x: X \cup Y \subseteq \{0..<m\}
shows \operatorname{card} \{xs. (\forall i \in X. \neg xs ! i) \land (\forall i \in Y. \neg xs ! i) \land \text{length } xs = m\} = 2 ^ \wedge (m - \operatorname{card} X - \operatorname{card} Y)

proof –
  let ?S = \{0..<m\} - (X \cup Y)
  from x have a: ?S \subseteq X \cup Y = \{0..<m\} \text{ by auto}
  have b: ?S \cap (X \cup Y) = \{\} \text{ by auto}
  show ?thesis apply (rule card1[where ?S=?S]) \text{ by (simp_all add: assms a b)}

qed

7.3 Average out the second sum for free-absch

lemma \text{Expectation2or1}: finite S \Rightarrow finite Tr \Rightarrow finite Fa \Rightarrow \operatorname{card} Tr + \operatorname{card} Fa + \operatorname{card} S \leq l \Rightarrow
(S \cap (Tr \cup Fa) = \{\} \Rightarrow Tr \cap Fa = \{\} \Rightarrow S \cup Tr \cup Fa \subseteq \{0..<l\} \Rightarrow
(\sum x \in \{xs. (\forall i \in Tr. xs ! i) \land (\forall i \in Fa. \neg xs ! i) \land \text{length } xs = l\}. \sum j \in S. \text{ if } x ! j \text{ then } 2 \text{ else } 1) = 3 / 2 \times \operatorname{real} (\operatorname{card} S) \times 2 ^ \wedge (l - \operatorname{card} Tr - \operatorname{card} Fa)
proof (induct arbitrary: Tr Fa rule: finite_induct)
  case (insert e S)

  from insert(7) have e ∈ (insert e S) and eTr: e /∈ Tr and eFa: e /∈ Fa
  by auto
  from insert(9) have tra: Tr ⊆ {0..l} and trb: Fa ⊆ {0..l} and tre: e < l by auto

  have ntrFa: l > (card Tr + card Fa) using insert(6) card_insert_if_insert(1,2) by auto

  have myhelp2: 1 + (l - card Tr - card Fa - 1) = l - card Tr - card Fa using ntrFa by auto

  have juhuTr: {xs. (∀ i ∈ Tr. xs ! i) ∧ (∀ i ∈ Fa. ¬ xs ! i) ∧ length xs = l ∧ xs!e}
    = {xs. (∀ i ∈ (insert e Tr). xs ! i) ∧ (∀ i ∈ Fa. ¬ xs ! i) ∧ length xs = l}
  by auto
  have juhuFa: {xs. (∀ i ∈ Tr. xs ! i) ∧ (∀ i ∈ Fa. ¬ xs ! i) ∧ length xs = l ∧ ¬xs!e}
    = {xs. (∀ i ∈ Tr. xs ! i) ∧ (∀ i ∈ (insert e Fa). ¬ xs ! i) ∧ length xs = l}
  by auto

  let ?Tre = {xs. (∀ i ∈ (insert e Tr). xs ! i) ∧ (∀ i ∈ Fa. ¬ xs ! i) ∧ length xs = l}

  have card ?Tre = 2 ^ (l - card (insert e Tr) - card Fa)
    apply (rule card2) using insert by simp_all
  then have resi: card ?Tre = 2 ^ (l - card Tr - card Fa - 1) using insert(4) eTr by auto
  have yabaTr: (∑ x ∈ ?Tre. 2 ^ real) = 2 ^ 2 ^ (l - card Tr - card Fa - 1)
  using resi by (simp add: power_commutes)

  let ?Fae = {xs. (∀ i ∈ Tr. xs ! i) ∧ (∀ i ∈ (insert e Fa). ¬ xs ! i) ∧ length xs = l}

  have card ?Fae = 2 ^ (l - card Tr - card (insert e Fa))
    apply (rule card2) using insert by simp_all
  then have resi2: card ?Fae = 2 ^ (l - card Tr - card Fa - 1) using insert(5) eFa by auto
  have yabaFa: (∑ x ∈ ?Fae. 1 ^ real) = 1 ^ 2 ^ (l - card Tr - card Fa - 1)
  using resi2 by (simp add: power_commutes)
\{ \text{fix } X \ Y \\
\text{have } \{ \text{xs. } (\forall i \in X. \ x \neq i) \land (\forall i \in Y. \ x \neq i) \land \text{length xs} = 1 \land \text{xs!e} \} \\
\quad \cap \{ \text{xs. } (\forall i \in X. \ x \neq i) \land (\forall i \in Y. \ x \neq i) \land \text{length xs} = 1 \land \neg \text{xs!e} \} \\
= \{ \} \text{ by auto} \\
\} \text{ note } 3 = \text{this} \\

\text{have } \left( \sum_{x \in \{\text{xs. } (\forall i \in \text{Tr. } x \neq i) \land (\forall i \in \text{Fa. } x \neq i) \land \text{length xs} = 1 \}} \begin{array}{l}
\sum_{j \in (\text{insert } e \ S). \text{ if } x ! j \text{ then } (2::\text{real}) \text{ else } 1} \\
= (\sum_{x \in \{\text{xs. } (\forall i \in \text{Tr. } x \neq i) \land (\forall i \in \text{Fa. } x \neq i) \land \text{length xs} = 1 \land \text{xs!e} \}}. \sum_{j \in (\text{insert } e \ S). \text{ if } x ! j \text{ then } 2 \text{ else } 1} \\
+ (\sum_{x \in \{\text{xs. } (\forall i \in \text{Tr. } x \neq i) \land (\forall i \in \text{Fa. } x \neq i) \land \text{length xs} = 1 \land \neg \text{xs!e} \}}. \sum_{j \in (\text{insert } e \ S). \text{ if } x ! j \text{ then } 2 \text{ else } 1}) \\
\text{is } (\sum x \in \text{?all}. \text{ ? } x) = (\sum x \in \text{?allT}. \text{ ? } x) + (\sum x \in \text{allF}. \text{ ? } x) \\
\text{proof } - \\
\text{have } (\sum x \in \text{?all}. \sum j \in (\text{insert } e \ S). \text{ if } x ! j \text{ then } 2 \text{ else } 1) \\
= (\sum x \in \text{(?allT } \cup \text{ ?allF}). \sum j \in (\text{insert } e \ S). \text{ if } x ! j \text{ then } 2 \text{ else } 1) \\
\text{apply } (\text{rule sum.cong } \text{by } \text{auto}) \\
\text{also have } \ldots = (\sum x \in \text{(?allT}). \sum j \in (\text{insert } e \ S). \text{ if } x ! j \text{ then } 2 \text{ else } 1) \\
\quad + (\sum x \in \text{(?allF}). \sum j \in (\text{insert } e \ S). \text{ if } x ! j \text{ then } 2 \text{ else } 1) \\
\text{else } 1) \\
\quad - (\sum x \in \text{(?allT } \cap \text{ ?allF}). \sum j \in (\text{insert } e \ S). \text{ if } x ! j \text{ then } 2 \text{ else } 1) \\
\text{apply } (\text{rule sum.Un } \text{by } \text{auto}) \text{ apply } (\text{rule bool } \text{+ done}) \\
\text{also have } \ldots = (\sum x \in \text{(?allT}). \sum j \in (\text{insert } e \ S). \text{ if } x ! j \text{ then } 2 \text{ else } 1) \\
\text{by } (\text{simp add: 3}) \\
\text{finally show } \text{?thesis} . \\
\text{qed} \\
\text{also} \\
\text{have } \ldots = (\sum x \in \text{?Tre}. \sum j \in (\text{insert } e \ S). \text{ if } x ! j \text{ then } 2 \text{ else } 1) \\
\quad + (\sum x \in \text{?Fa}. \sum j \in (\text{insert } e \ S). \text{ if } x ! j \text{ then } 2 \text{ else } 1) \\
\text{using juhuTy juhuFa by } \text{auto} \\
\text{also} \\
\text{have } \ldots = (\sum x \in \text{?Tre}. (\lambda x. \ 2) x + (\lambda x. (\sum j \in S. \text{ if } x ! j \text{ then } 2 \text{ else } 1))) \\
\quad + (\sum x \in \text{?Fa}. (\lambda x. \ 1) x + (\lambda x. (\sum j \in S. \text{ if } x ! j \text{ then } 2 \text{ else } 1))) x) \\
\text{using insert(1,2) by } \text{auto} \\
\text{also} \\
\text{have } \ldots = (\sum x \in \text{?Tre. } 2) + (\sum x \in \text{?Tre. } (\sum j \in S. \text{ if } x ! j \text{ then } 2 \text{ else } 1)) \\
\quad + ((\sum x \in \text{?Fa. } 1) + (\sum x \in \text{?Fa. } (\sum j \in S. \text{ if } x ! j \text{ then } 2 \text{ else } 1))) $
by (simp add: Groups_Big.comm_monoid_add_class.sum.distrib)
also have ... = 2 * 2^*(l - card Tr - card Fa - 1) + (\(\sum x \in ?Tr. (\sum j \in S. \\
if x \land j \text{ then 2 else 1})\))
   + (1 * 2^*(l - card Tr - card Fa - 1) + (\(\sum x \in ?Fa. (\sum j \in S. \\
if x \land j \text{ then 2 else 1})\)))
by (simp only: yabaTr yabaFa)
also have ... = (2::real) * 2^*(l - card Tr - card Fa - 1) + (\(\sum x \in ?Tr. (\sum j \in S. \\
if x \land j \text{ then 2 else 1})\))
   + (1::real) * 2^*(l - card Tr - card Fa - 1) + (\(\sum x \in ?Fa. (\sum j \in S. \\
if x \land j \text{ then 2 else 1})\))
by auto
also have ... = (3::real) * 2^*(l - card Tr - card Fa - 1) + 
   3 / 2 * real (card S) * 2^*(l - card (insert e Tr) - card Fa) + 
   (\(\sum x \in ?Fa. (\sum j \in S. \\
if x \land j \text{ then 2 else 1})\))
apply (subst insert(3)) using insert by simp_all
also have ... = 3 * 2^*(l - card Tr - card Fa - 1) + 
   3 / 2 * real (card S) * 2^*(l - card Tr - card Fa) + 
   3 / 2 * real (card S) * 2^*(l - card Tr - card (insert e Fa))
apply (subst insert(3)) using insert by simp_all
also have ... = 3 * 2^*(l - card Tr - card Fa - 1) + 
   3 / 2 * real (card S) * 2^*(l - (card Tr + 1) - card Fa) + 
   3 / 2 * real (card S) * 2^*(l - card Tr - (card Fa + 1)) using 
card_insert_if insert(4,5) eTr eFa by auto
also have ... = 3 * 2^*(l - card Tr - card Fa - 1) + 
   3 / 2 * real (card S) * 2^*(l - card Tr - card Fa - 1) + 
   3 / 2 * real (card S) * 2^*(l - card Tr - card Fa - 1) by auto
also have ... = (3 / 2 * 2 + 2 * 3 / 2 * real (card S)) * 2^*(l - card Tr - 
   card Fa - 1) by algebra
also have ... = (3 / 2 * (1 + real (card S))) * 2^*(l - card Tr - 
   card Fa - 1) by simp
also
have \ldots = (3/2 \cdot (1 + real (card S))) \cdot 2^\cdot (Suc (1 - card Tr - card Fa - 1)) by simp
also have \ldots = (3/2 \cdot (1 + real (card S))) \cdot 2^\cdot (1 - card Tr - card Fa)
) using myhelp2 by auto
also have \ldots = (3/2 \cdot (real (1 + card S))) \cdot 2^\cdot (l - card Tr - card Fa)
using insert(1,2) by auto
finally show ?case .
qed simp
end

8 Effect of mtf2

theory MTF2_Effects
imports Move_to_Front
begin

lemma difind_difelem:
  i < length xs \implies distinct xs \implies xs ! j = a \implies j < length xs \implies i \neq j
  \implies \neg a = xs ! i
apply(rule ccontr) by (metis index nth id)

lemma fullchar: assumes index xs q < length xs
shows
  (i < length xs) =
  (index xs q < i \land i < length xs
  \lor index xs q = i
  \lor index xs q - n \leq i \land i < index xs q
  \lor i < index xs q - n)
using assms by auto

lemma mtf2_effect:
  q \in set xs \implies distinct xs \implies (index xs q < i \land i < length xs \implies)
index (mtf2 n q xs) (xs!i) = index xs (xs!i) \land index xs q < index (mtf2 n

\( q \, xs \) \((xs!i) \land \text{index} \,(\text{mtf2} \, n \, q \, xs) \,(xs!i) < \text{length} \, xs)\)
\[ \land \,(\text{index} \, xs \, q \,= \, i \rightarrow (\text{index} \,(\text{mtf2} \, n \, q \, xs) \,(xs!i) = \text{index} \, xs \, q - n \land \text{index} \,(\text{mtf2} \, n \, q \, xs) \,(xs!i) = \text{index} \, xs \, q - n))\]
\[ \land \,(\text{index} \, xs \, q - n \leq i \land i < \text{index} \, xs \, q \rightarrow (\text{index} \,(\text{mtf2} \, n \, q \, xs) \,(xs!i) = \text{Suc} \,(\text{index} \, xs \, (xs!i)) \land \text{index} \, xs \, q - n < \text{index} \,(\text{mtf2} \, n \, q \, xs) \,(xs!i) \land \text{index} \,(\text{mtf2} \, n \, q \, xs) \,(xs!i) \leq \text{index} \, xs \, q))\]
\[ \land \,(i < \text{index} \, xs \, q - n \rightarrow (\text{index} \,(\text{mtf2} \, n \, q \, xs) \,(xs!i) = \text{index} \, xs \,(xs!i) \land \text{index} \,(\text{mtf2} \, n \, q \, xs) \,(xs!i) < \text{index} \, xs \, q - n))\]

unfolding \text{mtf2_def}
apply \,(\text{induct} \, n)\)
proof -
\[ \text{case} \,(\text{Suc} \, n)\]

\text{note indH}=\text{Suc}(1)[\text{OF} \, \text{Suc}(2) \, \text{Suc}(3), \, \text{simplified} \, \text{Suc}(2) \, \text{if}_\text{True}]\]
\text{note qinxs}=\text{Suc}(2)[\text{simp}]
\text{note distrxs}=\text{Suc}(3)[\text{simp}]

show \,(?case \,(\text{is} \, ?\text{toshow}))\)

apply \,(\text{simp only} \,: \, \text{qinxs if}_\text{True})\)

proof \,(\text{cases} \, \text{index} \, xs \, q \geq \text{Suc} \, n)\)
  \[ \text{case} \, \text{True}\]
  \text{note True1}=this
  from \text{True have} \, \text{onemore:} \,[\text{index} \, xs \, q - \text{Suc} \, n..<\text{index} \, xs \, q] = (\text{index} \, xs \, q - \text{Suc} \, n) \, \# \,[\text{index} \, xs \, q - n..<\text{index} \, xs \, q]\)
  using \text{Suc_diff_Suc upt_rec by auto}

  from \text{onemore have} \, \text{yeah:} \, \text{swaps} \,[\text{index} \, xs \, q - \text{Suc} \, n..<\text{index} \, xs \, q] \, \text{xs}
  \,= \, \text{swap} \,(\text{index} \, xs \, q - \text{Suc} \, n) \,(\text{swaps} \,[\text{index} \, xs \, q - n..<\text{index} \, xs \, q] \, \text{xs}) \, \text{by auto}

  \text{have} \, \text{sis}: \, \text{Suc} \,(\text{index} \, xs \, q - \text{Suc} \, n) = \text{index} \, xs \, q - n \, \text{using True} \, \text{Suc_diff_Suc by auto}

  \text{have} \, \text{indq}: \, \text{index} \, xs \, q < \text{length} \, xs\)
  \text{apply} \,(\text{rule} \, \text{index_less}) \, \text{by auto}

  let \, ?i\,' = \text{index} \,(\text{swaps} \,[\text{index} \, xs \, q - \text{Suc} \, n..<\text{index} \, xs \, q] \, \text{xs}) \,(\text{xs} \, ! \, i)\)
  let \, \text{?x}=\, (\text{xs}!\text{i}) \, \text{and} \, \text{?xs}=\,(\text{swaps} \,[\text{index} \, xs \, q - n..<\text{index} \, xs \, q] \, \text{xs})\)
  \, \text{and} \, \text{?n}=\,(\text{index} \, xs \, q - \text{Suc} \, n)\)

  \text{have} \, \text{?i}\,' = \text{index} \,(\text{swap} \,(\text{index} \, xs \, q - \text{Suc} \, n) \,(\text{swaps} \,[\text{index} \, xs \, q - n..<\text{index} \, xs \, q] \, \text{xs})) \,(\text{xs}!\text{i}) \, \text{using yeah by auto}

  \text{also have} \, \ldots \, = \,(\text{if} \, \text{?x} = \, ?\text{xs} \, ! \, ?\text{n} \, \text{then} \, \text{Suc} \, ?\text{n} \, \text{else} \, \text{if} \, \text{?x} = \, ?\text{xs} \, ! \, \text{Suc} \, ?\text{n} \, \text{then} \, ?\text{n} \, \text{else} \, \text{index} \, ?\text{xs} \, ?\text{x})\)
  \text{apply} \,(\text{rule} \, \text{index_swap_distinct})\)
apply(simp)
apply(simp add: sis) using indq by linarith

finally have i': ?i' = (if ?x = ?xs ! ?n then Suc ?n else if ?x = ?xs ! Suc ?n then ?n else index ?xs ?x) .

let ?i''=index (swaps [index xs q − n..<index xs q] xs) (xs ! i)

show (index xs q < i ∧ i < length xs →
index (swaps [index xs q − Suc n..<index xs q] xs) (xs ! i) = index xs
(xs ! i) ∧
index xs q < index (swaps [index xs q − Suc n..<index xs q] xs) (xs ! i)
index (swaps [index xs q − Suc n..<index xs q] xs) (xs ! i) < length
xs) ∧
(index xs q = i →
index (swaps [index xs q − Suc n..<index xs q] xs) (xs ! i) = index xs
q − Suc n ∧
index (swaps [index xs q − Suc n..<index xs q] xs) (xs ! i) = index xs
q − Suc n) ∧
(index xs q − Suc n ≤ i ∧ i < index xs q →
index (swaps [index xs q − Suc n..<index xs q] xs) (xs ! i) = Suc (index
xs (xs ! i)) ∧
index xs q − Suc n < index (swaps [index xs q − Suc n..<index xs q]
x) (xs ! i) ∧
index (swaps [index xs q − Suc n..<index xs q] xs) (xs ! i) ≤ index xs
q) ∧
i < index xs q − Suc n →
index (swaps [index xs q − Suc n..<index xs q] xs) (xs ! i) = index xs
(xs ! i) ∧
index (swaps [index xs q − Suc n..<index xs q] xs) (xs ! i) < index xs
q − Suc n)
apply(intro conjI)
apply(intro impI) apply(elim conjE) prefer 4 apply(intro impI)
prefer 4 apply(intro impI) apply(elim conjE)
prefer 4 apply(intro impI) prefer 4
proof (goal_cases)
case 1
have indH1: (index xs q < i ∧ i < length xs →
?i'' = index xs (xs ! i)) using indH by auto
assume ass: index xs q < i and ass2:i < length xs
then have a: ?i'' = index xs (xs ! i) using indH1 by auto
also have a': ... = i apply(rule index_nth_id) using ass2 by(auto)
finally have ii: ?i'' = i .
have fstF: ∼ ?x = ?xs ! ?n apply(rule difind_difelem|where j=index (swaps [index xs q - n..<index xs q] xs) (xs!i))
  using indq apply (simp add: less_imp_diff_less)
  apply(simp)
  apply(rule nth_index) apply(simp) using ass2 apply(simp)
  apply(rule index_less)
    apply(simp) using ass2 apply(simp)
    apply(simp)
  using i ii ass by auto
have sndF: ∼ ?x = ?xs ! Suc ?n apply(rule difind_difelem|where j=index (swaps [index xs q - n..<index xs q] xs) (xs!i))
  using indq True apply (simp add: Suc_diff_Suc less_imp_diff_less)
  apply(simp)
  apply(rule nth_index) apply(simp) using ass2 apply(simp)
  apply(rule index_less)
    apply(simp) using ass2 apply(simp)
    apply(simp)
  using ii ass Suc by auto

have ?i' = index xs (xs ! i) unfolding i' using fstF sndF a by simp
then show ?case using a' ass ass2 by auto
next
case 2
have indH2: index xs q = i → ?i'' = index xs (xs ! i) - n using indH by auto
  assume index xs q = i
  then have ass: i = index xs q by auto
  with indH2 have a: i - n = ?i'' by auto
  from ass have c: index xs (xs ! i) = i by auto
  have Suc (index xs q - Suc n) = i - n using ass True Suc_diff_Suc by auto
  also have ... = ?i'' using a by auto
  finally have a: Suc ?n = ?i''.

have sndTrue: ?x = ?xs ! Suc ?n apply(simp add: a)
  apply(rule nth_index[symmetric]) by (simp add: ass)
have fstFalse: ∼ ?x = ?xs ! ?n apply(rule difind_difelem|where j=index (swaps [index xs q - n..<index xs q] xs) (xs!i))
  using indq True apply (simp add: Suc_diff_Suc less_imp_diff_less)
  apply(simp)
  apply(rule nth_index) apply(simp) using ass apply(simp)
  apply(rule index_less)
    apply(simp) using ass apply(simp)
    apply(simp)
  apply(simp)

66
using a by auto

have ?i' = index xs (xs ! index xs q) − Suc n
  unfolding i' using sndTrue fstFalse by simp
with ass show ?case by auto

next
  case 3
  have indH3: index xs q − n ≤ i ∧ i < index xs q
    → ?i'' = Suc (index xs (xs ! i)) using indH by auto
  assume ass: index xs q − Suc n ≤ i and
  ass2: i < index xs q
  from ass2 have ilen: i < length xs using indq dual_order.strict_trans
    by blast
  show ?case
  proof (cases index xs q − n ≤ i)
    case False
    then have i < index xs q − n by auto
    moreover have (i < index xs q − n → ?i'' = index xs (xs ! i))
      using indH by auto
    ultimately have d: ?i'' = index xs (xs ! i) by simp
    from False ass have b: index xs q − Suc n = i by auto
    have index xs q < length xs apply (rule index_less) by (auto)
    have c: index xs (xs ! i) = i
      apply (rule index_nth_id) apply (simp) using indq ass2 using
      less_trans by blast
    from b c d have f: ?i'' = index xs q − Suc n by auto
    have fstT: ?xs ! ?n = ?x
      apply (simp only: f[symmetric]) apply (rule nth_index)
      by (simp add: ilen)
  have ?i' = Suc (index xs q − Suc n)
    unfolding i' using fstT by simp
  also have ... = Suc (index xs (xs ! i)) by (simp only: b c)
  finally show ?thesis using c False ass by auto

next
  case True
  with ass2 indH3 have a: ?i'' = Suc (index xs (xs ! i)) by auto
  have jo: index xs (xs ! i) = i apply (rule index_nth_id) using ilen
    by (auto)
  have fstF: ~ ?x = ?xs ! ?n apply (rule difind_difelem[where j=index
    (swaps [index xs q − n..<index xs q] xs) (xs!i)])
    using indq apply (simp add: less_imp_diff_less)
    apply (simp)
    apply (rule nth_index) apply (simp) using ilen apply (simp)
apply\text{(rule \textit{index\_less})
  apply\text{(simp) using \textit{ilen apply\text{(simp)}}
  apply\text{(simp)}
apply\text{(simp only: a jo) using True by auto
have \textit{sndF}: \sim ?x = \sim xs ! Suc \sim n apply\text{(rule \textit{difind\_difelem}[where j=index (swaps [index xs q − n..<index xs q] xs) (xs!i)])}
  using True1 apply \text{(simp add: Suc\_diff\_Suc less\_imp\_diff\_less)}
  apply\text{(simp)}
apply\text{(rule nth\_index)} apply\text{(simp) using \textit{ilen apply\text{(simp)}}
apply\text{(rule \textit{index\_less})}
  apply\text{(simp)}
apply\text{(simp)}
apply\text{(simp only: a jo) using True1 apply \text{(simp add: Suc\_diff\_Suc less\_imp\_diff\_less)}
  using True by auto
have \sim i' = Suc (index xs (xs ! i)) unfolding \textit{i'} using \textit{fstF sndF a by simp}}

then show \textit{?thesis} using \textit{ass ass2 jo by auto
}
qed

next
case 4
  assume \textit{ass: i < index xs q − Suc n}
  then have \textit{ass2: i < index xs q − n by auto
  moreover have (i < index xs q − n → \sim i'' = index xs (xs ! i))
using \textit{indH by auto
ultimately have a: \sim i'' = index xs (xs ! i) by auto
from \textit{ass2 have i < index xs q by auto
then have \textit{ilen: i < length xs using \textit{indq dual\_order.strict\_trans by blast

have \textit{jo: index xs (xs ! i) = i apply\text{(rule \textit{index\_nth\_id}) using \textit{ilen by\text{(auto)}}
  have \textit{fstF: \sim ?x = \sim xs ! ?n apply\text{(rule \textit{difind\_difelem}[where j=index (swaps [index xs q − n..<index xs q] xs) (xs!i)])
  using \textit{indq apply (simp add: less\_imp\_diff\_less)}
  apply\text{(simp)}
apply\text{(rule nth\_index)} apply\text{(simp) using \textit{ilen apply\text{(simp)}}
apply\text{(rule \textit{index\_less})
  apply\text{(simp)} using \textit{ilen apply\text{(simp)}}
apply\text{(simp)}
apply\text{(simp only: a jo) using \textit{ass by auto
have \textit{sndF: \sim ?x = \sim xs ! Suc \sim n apply\text{(rule \textit{difind\_difelem}[where j=index (swaps [index xs q − n..<index xs q] xs) (xs!i)])

68
using True1 apply (simp add: Suc_diff_Suc less_imp_diff_less)
apply(simp)
apply(rule nth_index) apply(simp) using i len apply(simp)
apply(rule index_less)
apply(simp) using i len apply(simp)
apply(simp)
apply(simp only: a jo)
using True1 apply (simp add: Suc_diff_Suc less_imp_diff_less)
using asss by auto
have "i' = (index xs (xs ! i))" unfolding i' using fstF sndF a by simp
then show ?case using jo ass by auto
qed
next
case False
then have smalla: "index xs q - Suc n = index xs q - n by auto"
then have nomore: "swaps [index xs q - Suc n..<index xs q] xs = swaps [index xs q - n..<index xs q] xs by auto"
show "(index xs q < i ∧ i < length xs → index (swaps [index xs q - Suc n..<index xs q] xs) (xs ! i) = index xs (xs ! i) ∧ index xs q < index (swaps [index xs q - Suc n..<index xs q] xs) (xs ! i) ∧ index (swaps [index xs q - Suc n..<index xs q] xs) (xs ! i) < length xs) ∧ (index xs q = i → index (swaps [index xs q - Suc n..<index xs q] xs) (xs ! i) = index xs q - Suc n ∧ index (swaps [index xs q - Suc n..<index xs q] xs) (xs ! i) = index xs q - Suc n) ∧ (index xs q - Suc n ≤ i ∧ i < index xs q → index (swaps [index xs q - Suc n..<index xs q] xs) (xs ! i) = Suc (index xs (xs ! i)) ∧ index xs q - Suc n < index (swaps [index xs q - Suc n..<index xs q] xs) (xs ! i) ∧ index (swaps [index xs q - Suc n..<index xs q] xs) (xs ! i) ≤ index xs q) ∧ (i < index xs q - Suc n → index (swaps [index xs q - Suc n..<index xs q] xs) (xs ! i) = index xs (xs ! i) ∧ index (swaps [index xs q - Suc n..<index xs q] xs) (xs ! i) < index xs q - Suc n)
unfolding nomore smalla by (rule indH)
qed
next
case 0
then show \( \text{case apply}(\text{simp}) \)
proof (safe, goal_cases)
case 1
have \( \text{index } xs \ (xs!i) = i \) apply(rule \text{index_nth_id}) using 1 by auto
with 1 show \( ?\text{case by auto} \)
next
case 2
have \( xs! \text{index } xs q = q \) using 2 by(auto)
with 2 show \( ?\text{case by auto} \)
next
case 3
have \( a: \text{index } xs q < \text{length } xs \) apply(rule \text{index_less}) using 3 by(auto)
apply(rule \text{index_nth_id}) apply(rule \text{fact 3(2)})
with 3 show \( ?\text{case by auto} \)
qed

lemma \text{mtf2_forward_effect1}:
\[
\begin{align*}
q \in \text{set } xs & \implies \text{distinct } xs \implies \text{index } xs q < i \land i < \text{length } xs \\
& \implies \text{index } (\text{mtf2 } n q xs) (xs!i) = \text{index } xs (xs!i) \land \text{index } xs q < \\
& \text{index } (\text{mtf2 } n q xs) (xs!i) \land \text{index } (\text{mtf2 } n q xs) (xs!i) < \text{length } xs \quad \text{and}
\end{align*}
\]

\[
\begin{align*}
\text{mtf2_forward_effect2}: q \in \text{set } xs & \implies \text{distinct } xs \implies \text{index } xs q = i \\
& \implies \text{index } (\text{mtf2 } n q xs) (xs!i) = \text{index } xs q - n \land \text{index } xs q - n = \\
& \text{index } (\text{mtf2 } n q xs) (xs!i) \quad \text{and}
\end{align*}
\]

\[
\begin{align*}
\text{mtf2_forward_effect3}: q \in \text{set } xs & \implies \text{distinct } xs \implies \text{index } xs q - n \leq i \\
& \land i < \text{index } xs q \\
& \implies \text{index } (\text{mtf2 } n q xs) (xs!i) = \text{Suc} (\text{index } xs (xs!i)) \land \text{index } xs q - n < \\
& \text{index } (\text{mtf2 } n q xs) (xs!i) \land \text{index } (\text{mtf2 } n q xs) (xs!i) \leq \text{index } xs q \\
\end{align*}
\]

\[
\begin{align*}
\text{mtf2_forward_effect4}: q \in \text{set } xs & \implies \text{distinct } xs \implies i < \text{index } xs q - n \\
& \implies \text{index } (\text{mtf2 } n q xs) (xs!i) = \text{index } xs (xs!i) \land \text{index } (\text{mtf2 } n q xs) \\
\end{align*}
\]

apply(safe) using \text{mtf2_effect by metis+}

lemma \text{yes[simp]}: index xs x < length xs
\[
\begin{align*}
& \implies (xs!\text{index } xs x) = x \quad \text{apply}(\text{rule nth_index}) \quad \text{by} \quad \text{simp add: index_less_size_conv}
\end{align*}
\]

lemma \text{mtf2_forward_effect1'}:
\[
\begin{align*}
q \in \text{set } xs & \implies \text{distinct } xs \implies \text{index } xs q < \text{index } xs x \land \text{index } xs x <
\end{align*}
\]
\[ \text{length } xs \Rightarrow \text{index } (\text{mtf2 } n \ q \ xs) \ x = \text{index } xs \ x \land \text{index } xs \ q < \text{index } (\text{mtf2 } n \ q \ xs) \ x \land \text{index } (\text{mtf2 } n \ q \ xs) \ x < \text{length } xs \]

**using** \text{mtf2\_forward\_effectl[where } xs=xs \text{ and } i=\text{index } xs \ x \text{]} \text{ yes by(auto)}

**lemma**
\text{mtf2\_forward\_effect2': } q \in \text{set } xs \Rightarrow \text{distinct } xs \Rightarrow \text{index } xs \ q = \text{index } xs \ x
\Rightarrow \text{index } (\text{mtf2 } n \ q \ xs) \ (xs!\text{index } xs \ x) = \text{index } xs \ q - n \land \text{index } xs \ q - n = \text{index } (\text{mtf2 } n \ q \ xs) \ (xs!\text{index } xs \ x)
\text{using } \text{mtf2\_forward\_effect2[where } xs=xs \text{ and } i=\text{index } xs \ x] \text{ by fast}

**lemma**
\text{mtf2\_forward\_effect3': } q \in \text{set } xs \Rightarrow \text{distinct } xs \Rightarrow \text{index } xs \ x
\Rightarrow \text{index } (\text{mtf2 } n \ q \ xs) \ (xs!\text{index } xs \ x) = \text{Suc } (\text{index } xs \ (xs!\text{index } xs \ x)) \land \text{index } xs \ q - n < \text{index } (\text{mtf2 } n \ q \ xs) \ (xs!\text{index } xs \ x) \land \text{index } (\text{mtf2 } n \ q \ xs) \ (xs!\text{index } xs \ x) \leq \text{index } xs \ q
\text{using } \text{mtf2\_forward\_effect3[where } xs=xs \text{ and } i=\text{index } xs \ x] \text{ by fast}

**lemma**
\text{mtf2\_forward\_effect4': } q \in \text{set } xs \Rightarrow \text{distinct } xs \Rightarrow \text{index } xs \ x
\Rightarrow \text{index } (\text{mtf2 } n \ q \ xs) \ (xs!\text{index } xs \ x) = \text{index } xs \ (xs!\text{index } xs \ x) \land \text{index } (\text{mtf2 } n \ q \ xs) \ (xs!\text{index } xs \ x) < \text{index } xs \ q - n
\text{using } \text{mtf2\_forward\_effect4[where } xs=xs \text{ and } i=\text{index } xs \ x] \text{ by fast}

**lemma** \text{splitit: } (\text{index } xs \ q < i \land i < \text{length } xs \Rightarrow P)
\Rightarrow (\text{index } xs \ q = i \Rightarrow P)
\Rightarrow (\text{index } xs \ q - n \leq i \land i < \text{index } xs \ q \Rightarrow P)
\Rightarrow (i < \text{index } xs \ q - n \Rightarrow P)
\Rightarrow (i < \text{length } xs \Rightarrow P)
\text{by force}

**lemma** \text{mtf2\_forward\_beforeq: } q \in \text{set } xs \Rightarrow \text{distinct } xs \Rightarrow i < \text{index } xs \ q
\Rightarrow \text{index } (\text{mtf2 } n \ q \ xs) \ (xs!i) \leq \text{index } xs \ q
\text{apply } (\text{cases } i < \text{index } xs \ q - n)
using mtf2_forward_effect4 apply force
using mtf2_forward_effect3 using leI by metis

lemma x_stays_before_y_if_y_notMoved_to_front:
assumes q ∈ set xs distinct xs x ∈ set xs y ∈ set xs y ≠ q
and x < y in xs
shows x < y in (mtf2 n q xs)

proof –
from assms(3) obtain i where i = index xs x and i2: i < length xs
by auto
from assms(4) obtain j where j = index xs y and j2: j < length xs
by auto
have x < y in xs → x < y in (mtf2 n q xs)
apply (cases i xs rule: splitI[where q=q and n=n])
apply (cases j xs rule: splitI[where q=q and n=n])
apply (metis before_in_def assms(1−3) i j less_imp_diff_less mtf2_effect
nth_index set_mtf2)
apply (simp add: i j assms mtf2_forward_effect1' mtf2_forward_effect2'
before_in_def)
apply (simp add: i j assms mtf2_forward_effect1' mtf2_forward_effect2'
before_in_def)
apply (simp add: i j assms mtf2_forward_effect1' mtf2_forward_effect3'
before_in_def)
apply (rule j2)
apply (cases j xs rule: splitI[where q=q and n=n])
apply (smt before_in_def assms(1−3) i j le_less_trans mtf2_forward_effect1
mtf2_forward_effect3 nth_index set_mtf2)
using assms(4,5) j apply simp
apply (smt Suc_leI before_in_def assms(1−3) i j le_less_trans lessI
mtf2_forward_effect3 nth_index set_mtf2)
apply (simp add: before_in_def i j)
apply (rule j2)
apply (cases j xs rule: splitI[where q=q and n=n])
apply (smt before_in_def assms(1−3) i j le_less_trans mtf2_forward_effect1
mtf2_forward_effect4 nth_index set_mtf2)
using assms(4−5) j apply simp
apply (smt before_in_def assms(1−3) i j le_less_trans less_imp_le_nat
mtf2_forward_effect3 mtf2_forward_effect4 nth_index set_mtf2)
apply (metis before_in_def assms(1−3) i j mtf2_forward_effect4 nth_index
set_mtf2)
apply (rule j2)
apply (rule i2) done

72
corollary swapped_by_mtf2: q ∈ set xs ⇒ distinct xs ⇒ x ∈ set xs ⇒ y ∈ set xs ⇒
  x < y in xs ⇒ y < x in (mtf2 n q xs) ⇒ y = q
apply (rule ccontr) using x_stays_before_y_if_y_not Moved_to_front not before in
by (metis before_in_setD1)

lemma x_stays_before_y if_y_not Moved_to_front 2dir:
  q ∈ set xs ⇒ distinct xs ⇒ x ∈ set xs ⇒ y ∈ set xs ⇒ y ≠ q ⇒
  x < y in xs = x < y in (mtf2 n q xs)
oops

lemma mtf2.backwards_effect1:
  assumes index xs q < length xs q ∈ set xs distinct xs
  index xs q < index (mtf2 n q xs) (xs ! i) ∧ index (mtf2 n q xs) (xs ! i) < length xs
  i < length xs
  shows index xs q < i ∧ i < length xs
proof –
  from assms(4) have ~ (index xs q − n = index (mtf2 n q xs) (xs ! i))
  by auto
  with assms mtf2.forward_effect2 have 1: ~ (index xs q = i) by metis
  from assms(4) have ~ (index xs q − n < index (mtf2 n q xs) (xs ! i) ∧
  index (mtf2 n q xs) (xs ! i) ≤ index xs q) by auto
  with assms mtf2.forward_effect3 have 2: ~ (index xs q − n ≤ i ∧ i <
  index xs q) by metis
  from assms(4) have ~ (index (mtf2 n q xs) (xs ! i) < index xs q − n)
  by auto
  with assms mtf2.forward_effect4 have 3: ~ (i < index xs q − n) by metis
  from fullchar[OF assms(1)] assms(5) 1 2 3 show index xs q < i ∧ i <
  length xs by metis
qed

lemma mtf2.backwards_effect2:
  assumes index xs q < length xs q ∈ set xs distinct xs index (mtf2 n q xs)
  (xs ! i) = index xs q − n
  i < length xs
  shows index xs q = i
proof –
  from assms(4) have ~ (index xs q < index (mtf2 n q xs) (xs ! i) ∧ index
\[(\text{mtf2 } n \ q \ \text{xs}) (\text{xs} ! i) < \text{length xs}) \text{ by auto}\]

\text{with assms mtf2\_forward\_effect1 have 1: } ~ (\text{index xs } q < i \land i < \text{length xs}) \text{ by metis}

\text{from assms(4) have } ~ (\text{index xs } q - n < \text{index (mtf2 } n \ q \ \text{xs}) (\text{xs} ! i) \land \text{index (mtf2 } n \ q \ \text{xs}) (\text{xs} ! i) \leq \text{index xs } q) \text{ by auto}

\text{with assms mtf2\_forward\_effect3 have 2: } ~ (\text{index xs } q - n \leq i \land i < \text{index xs } q) \text{ by metis}

\text{from assms(4) have } ~ (\text{index (mtf2 } n \ q \ \text{xs}) (\text{xs} ! i) < \text{index xs } q - n) \text{ by auto}

\text{with assms mtf2\_forward\_effect4 have 3: } ~ (i < \text{index xs } q - n) \text{ by metis}

\text{from fullchar[OF assms(1)] assms(5) 1 2 3 show index xs } q = i \text{ by metis}

\text{qed}

\text{lemma mtf2\_backwards\_effect3:}
\text{assumes index xs } q < \text{length xs } q \in \text{set xs distinct xs}
\text{index xs } q - n < \text{index (mtf2 } n \ q \ \text{xs}) (\text{xs} ! i) \land \text{index (mtf2 } n \ q \ \text{xs}) (\text{xs} ! i) \leq \text{index xs } q
\text{i < length xs}
\text{shows index xs } q - n \leq i \land i < \text{index xs } q

\text{proof ~}
\text{from assms(4) have } ~ (\text{index xs } q < \text{index (mtf2 } n \ q \ \text{xs}) (\text{xs} ! i) \land \text{index (mtf2 } n \ q \ \text{xs}) (\text{xs} ! i) < \text{length xs}) \text{ by auto}
\text{with assms mtf2\_forward\_effect1 have 2: } ~ (\text{index xs } q < i \land i < \text{length xs}) \text{ by metis}
\text{from assms(4) have } ~ (\text{index (mtf2 } n \ q \ \text{xs}) (\text{xs} ! i) = \text{index (mtf2 } n \ q \ \text{xs}) (\text{xs} ! i)) \text{ by auto}
\text{with assms mtf2\_forward\_effect2 have 1: } ~ (\text{index xs } q = i) \text{ by metis}
\text{from assms(4) have } ~ (\text{index (mtf2 } n \ q \ \text{xs}) (\text{xs} ! i) < \text{index xs } q - n) \text{ by auto}
\text{with assms mtf2\_forward\_effect4 have 3: } ~ (i < \text{index xs } q - n) \text{ by metis}

\text{from fullchar[OF assms(1)] assms(5) 1 2 3 show index xs } q - n \leq i \land i < \text{index xs } q \text{ by metis}

\text{qed}

\text{lemma mtf2\_backwards\_effect4:}
\text{assumes index xs } q < \text{length xs } q \in \text{set xs distinct xs}
\text{index (mtf2 } n \ q \ \text{xs}) (\text{xs} ! i) < \text{index xs } q - n
\text{i < length xs}
\text{shows i < index xs } q - n

\text{proof ~}

74
from assms(4) have \( \sim (\text{index } xs \ q < \text{index } (\text{mtf2 } n \ q \ xs) \ (xs \ i)) \land \text{index } (\text{mtf2 } n \ q \ xs) \ (xs \ i) < \text{length } xs \) by auto

with assms mtf2\_forward\_effect1 have 2:\( \sim (\text{index } xs \ q < i \land i < \text{length } xs) \) by metis

from assms(4) have \( \sim (\text{index } xs \ q - n = \text{index } (\text{mtf2 } n \ q \ xs) \ (xs \ i)) \) by auto

with assms mtf2\_forward\_effect2 have 1:\( \sim (\text{index } xs \ q - n < \text{index } (\text{mtf2 } n \ q \ xs) \ (xs \ i) \land \text{index } (\text{mtf2 } n \ q \ xs) \ (xs \ i) \leq \text{index } xs \ q) \) by auto

from assms(4) have \( \sim (\text{index } xs \ q - n \leq i \land i < \text{index } xs \ q) \) by metis

from fullchar[\text{OF assms(1)}] assms(5) 1 2 3 show i < index xs q - n by metis

qed

lemma mtf2\_backwards\_effect4':
assumes \( \text{index } xs \ q < \text{length } xs \ q \in \text{set } xs \ \text{distinct } xs \)
\( \text{index } (\text{mtf2 } n \ q \ xs) \ x < \text{index } xs \ q - n \)
\( x \in \text{set } xs \)

shows \( \text{index } xs \ x < \text{index } xs \ q - n \)

using assms mtf2\_backwards\_effect4[\text{where } xs=xs \ \text{and } i=\text{index } xs \ x] \) yes by auto

lemma
assumes distA: \text{distinct } A \text{ and}
asm: \( q \in \text{set } A \)

shows
mtf2\_mono: \( q < x \ \text{in } A \Longrightarrow q < x \ \text{in } (\text{mtf2 } n \ q \ A) \) and
mtf2\_q\_after: \( \text{index } (\text{mtf2 } n \ q \ A) \ q = \text{index } A \ q - n \)

proof –

have lele: \( (q < x \ \text{in } A \longrightarrow q < x \ \text{in } \text{swaps } [\text{index } A \ q - n..<\text{index } A \ q] A) \land (\text{index } \ (\text{swaps } [\text{index } A \ q - n..<\text{index } A \ q] A) \ q = \text{index } A \ q - n) \)
apPLY(induct n) apPLY(simp)

proof –

fix n

assume ind: \( (q < x \ \text{in } A \longrightarrow q < x \ \text{in } \text{swaps } [\text{index } A \ q - n..<\text{index } A \ q] A) \land (\text{index } (\text{swaps } [\text{index } A \ q - n..<\text{index } A \ q] A) \ q = \text{index } A \ q - n) \)

then have iH: \( q < x \ \text{in } A \Longrightarrow q < x \ \text{in } \text{swaps } [\text{index } A \ q - n..<\text{index } A \ q] A \) by auto

from ind have indH2: \( \text{index } (\text{swaps } [\text{index } A \ q - n..<\text{index } A \ q] A) \)
\[ q = \text{index } A \ q - n \ \text{by auto} \]

show \((q < x \text{ in } A) \rightarrow q < x \text{ in swaps [index } A \ q - \text{Suc } n..<\text{index } A \ q] A) \land \text{index (swaps [index } A \ q - \text{Suc } n..<\text{index } A \ q] A) } q = \text{index } A \ q - \text{Suc } n \ (\text{is } ?\text{part1 } \land ?\text{part2}) \]

proof (cases index \( A \ q \geq \text{Suc } n \))

case True

then have onemore: \([\text{index } A \ q - \text{Suc } n..<\text{index } A \ q] = (\text{index } A \ q - \text{Suc } n) \# [\text{index } A \ q - n..<\text{index } A \ q] \]

usingSuc_diff_Suc upt_rec by auto

from onemore have yeah: \(\text{swaps [index } A \ q - \text{Suc } n..<\text{index } A \ q] A \)

= swap (\text{index } A \ q - \text{Suc } n) (\text{swaps [index } A \ q - n..<\text{index } A \ q] A) \ \text{by auto} \]

from indH2 have gr: \(\text{index (swaps [index } A \ q - n..<\text{index } A \ q] A) } q = \text{Suc (index } A \ q - \text{Suc } n) \ 	ext{using Suc_diff_Suc True by auto} \]

have whereisq: \(\text{swaps [index } A \ q - n..<\text{index } A \ q] A ! (\text{Suc (index } A \ q - \text{Suc } n) = q) \)

unfolding gr[symmetric] apply(rule nth_index) using asm by auto

have indSi: \(\text{index } A \ q < \text{length } A \ \text{using asm index_less by auto} \)

have 3: \(\text{Suc (index } A \ q - \text{Suc } n) < \text{length (swaps [index } A \ q - n..<\text{index } A \ q] A) \ \text{using True} \)

apply(auto simp: Suc_diff_Suc asm) using indSi by auto

have 1: \(q \neq \text{swaps [index } A \ q - n..<\text{index } A \ q] A ! (\text{index } A \ q - \text{Suc } n) \)

proof

assume as: \(q = \text{swaps [index } A \ q - n..<\text{index } A \ q] A ! (\text{index } A \ q - \text{Suc } n) \)

\{
fix \(xs \ x\)

have Suc \(x < \text{length } xs \Rightarrow xs ! x = q \Rightarrow xs ! \text{Suc } x = q \)

\(\Rightarrow \) distinct \(xs\)

by (metis Suc_lessD index_nth_id n_not_Suc_n)

\} note cool=this

have \(! \text{distinct (swaps [index } A \ q - n..<\text{index } A \ q] A)\)

apply(rule cool[of (index A q - Suc n)])

apply(simp only: 3)
apply(simp only: as[symmetric])
by(simp only: whereisq)
then show False using distA by auto
qed

have part1: ?part1
proof
assume qx: q < x in A
{
  fix q x B i
  assume a1: q < x in B
  assume a2: ~ q = B ! i
  assume a3: distinct B
  assume a4: Suc i < length B

  have dist_perm B B by(simp add: a3)
  moreover have Suc i < length B using a4 by auto
  moreover have q < x in B ∧ ¬(q = B ! i ∧ x = B ! Suc i)
  using a1 a2 by auto

  ultimately have q < x in swap i B
  using before_in_swap[of B B] by simp
} note grr = this

have 2: distinct (swaps [index A q − n..<index A q] A) using distA by auto

show q < x in swaps [index A q − Suc n..<index A q] A
apply(simp only: yeah)
apply(rule grr[OF iH[OF qx]]) using 1 2 3 by auto
qed

let ?xs = (swaps [index A q − n..<index A q] A)
let ?n = (index A q − Suc n)
have ?xs ! Suc ?n = swaps [index A q − n..<index A q] A !(index
(swaps [index A q − n..<index A q] A) q)
  using indH2 Suc_diff_Suc True by auto
also have ... = q apply(rule nth_index) using asm by auto
finally have sndTrue: ?xs ! Suc ?n = q .
have fstFalse: ~ q = ?xs ! ?n by (fact 1)

have index (swaps [index A q − Suc n..<index A q] A) q

77
= index (swap (index A q – Suc n) ?xs) q by (simp only: yeah)

also have … = (if q = ?xs ! ?n then Suc ?n else if q = ?xs ! Suc
?n then ?n else index ?xs q)
apply (rule index_swap_distinct)
apply (simp add: distA)
by (fact 3)
also have … = ?n using fstFalse sndTrue by auto
finally have part2: ?part2.

from part1 part2 show ?part1 ∧ ?part2 by simp
next
case False
then have a: index A q – Suc n = index A q – n by auto
then have b: [index A q – Suc n..<index A q] = [index A q – n..<index A q] by auto
show ?thesis apply (simp only: b a) by (fact ind)
qed
qed

show q < x in A ⇒ q < x in (mtf2 n q A)
(index (mtf2 n q A) q) = index A q – n
unfolding mtf2_def
using asm lele apply (simp)
using asm lele by (simp)
qed

8.1 effect of mtf2 on index

lemma swapsthrough: distinct xs ⇒ q ∈ set xs ⇒ index (swaps [index
xs q – entf..<index xs q] xs ) q = index xs q – entf
proof (induct entf)
case (Suc e)
note iH = this
show ?case
proof (cases index xs q – e)
case 0
then have [index xs q – Suc e..<index xs q]
= [index xs q – e..<index xs q] by force
then have index (swaps [index xs q – Suc e..<index xs q] xs ) q
= index xs q – e using Suc by auto
also have … = index xs q – (Suc e) using 0 by auto
finally show index (swaps [index xs q – Suc e..<index xs q] xs ) q =
index xs q – Suc e.

78
next
case (Suc f)

have gaa: Suc (index xs q - Suc e) = index xs q - e using Suc by auto

from Suc have index xs q - e ≤ index xs q by auto
also have ... < length xs by(simp add: index_less_size_conv iH)
finally have indle: index xs q - e < length xs.

have arg: Suc (index xs q - Suc e) < length (swaps [index xs q - e..<index xs q] xs)
  apply(auto) unfolding gaa using indle by simp
then have arg2: index xs q - Suc e < length (swaps [index xs q - e..<index xs q] xs) by auto
from Suc have nexter: index xs q - e = Suc (index xs q - (Suc e))
  by auto
then have aaa: [index xs q - Suc e..<index xs q]
  = (index xs q - Suc e)#[index xs q - e..<index xs q] using upt_rec
by auto

let ?i=index xs q - Suc e
let ?rest=swaps [index xs q - e..<index xs q] xs
from iH nexter have indj: index ?rest q = Suc ?i by auto
from iH(2) have distinct ?rest by auto

have ?rest ! (index ?rest q) = q apply(rule nth_index) by(simp add: iH)
  with indj have whichcase: q = ?rest ! Suc ?i by auto

  with (distinct ?rest) have whichcase2: ~ q = ?rest ! ?i
    by (metis Suc_lessD arg index_nth_id n_not_Suc_n)

from aaa have index (swaps [index xs q - Suc e..<index xs q] xs) q
  = index (swap (index xs q - Suc e) (swaps [index xs q - e..<index xs q] xs)) q
  by auto
also have ... = (if q = ?rest ! ?i then (Suc ?i) else if q = ?rest ! (Suc ?i) then ?i else index ?rest q)
  apply(simp only: swap_def arg if_True)
  apply(rule index_swap_if_distinct)
  apply(simp add: iH)

79
apply(simp only: arg2)
by(simp only: arg)
also have \ldots = ?i using whichcase whichcase2 by simp
finally show index (swaps [index xs q - Suc e..<index xs q] xs) q = index xs q - Suc e.
qed

next
case 0
show ?case by simp
qed

term mtf2
lemma mtf2_moves_to_front: distinct xs \implies q \in set xs \implies index (mtf2 (length xs) q xs) q = 0
unfolding mtf2_def
proof -
assume distxs: distinct xs
assume qinxs: q \in set xs
have index (if q \in set xs then swaps [index xs q - length xs..<index xs q] xs else xs) q
= index ( swaps [index xs q - length xs..<index xs q] xs) q using qinxs
by auto
also have \ldots = index xs q - (length xs) apply(rule swapsthrough) using distxs qinxs by auto
also have \ldots = 0 using index_less_size_conv qinxs by (simp add: index_le_size)
finally show index (if q \in set xs then swaps [index xs q - length xs..<index xs q] xs else xs) q = 0 .
qed

lemma xy_relativorder_mtf2:
assumes
q\neq x q\neq y distinct xs x\in set xs y\in set xs q\in set xs
shows x < y in mtf2 n q xs
= x < y in xs
using assms
by (metis before_in_setD2 not_before_in x_stays_before_y_if_y_not_moved_to_front)

lemma mtf2_moves_to_frontm1: distinct xs \implies q \in set xs \implies index (mtf2
\[(\text{length } xs - 1) \ q \xs) \ q = 0\]

unfolding mtf2_def

proof –

assume distxs: distinct xs

assume qinxs: q \in \text{set } xs

have \(\text{index } (\text{if } q \in \text{set } xs \text{ then } \text{swaps } [\text{index } xs \ q - (\text{length } xs - 1)..<\text{index } xs \ q] \ xs \text{ else } xs) \ q = \text{index } (\text{swaps } [\text{index } xs \ q - (\text{length } xs - 1)..<\text{index } xs \ q] \ xs) \ q)\)

using qinxs by auto

also have \ldots = \text{index } xs \ q - (\text{length } xs - 1)\)

apply (rule swapsthrough)

using distxs qinxs by auto

also have \ldots = 0\)

using index_less_size_conv qinxs

by (metis Suc_pred' gr0I length_pos_if_in_set less_irrefl less_trans Suc zero_less_diff)

finally show \(\text{index } (\text{if } q \in \text{set } xs \text{ then } \text{swaps } [\text{index } xs \ q - (\text{length } xs - 1)..<\text{index } xs \ q] \ xs \text{ else } xs) \ q = 0\)

qed

lemma mtf2_moves_to_front':

\[\text{distinct } xs \implies y \in \text{set } xs \implies x \in \text{set } xs \implies x \neq y \implies x < y \implies \text{mtf2 } (\text{length } xs - 1) \ x \ x s = \text{True}\]

using mtf2_moves_to_frontm1 by (metis before_in_def gr0I index_eq_index_conv set_mtf2)

lemma mtf2_moves_to_front'':

\[\text{distinct } xs \implies y \in \text{set } xs \implies x \in \text{set } xs \implies x \neq y \implies x < y \implies \text{mtf2 } (\text{length } xs) \ x \ x s = \text{True}\]

using mtf2_moves_to_front by (metis before_in_def gr0I index_eq_index_conv set_mtf2)

9 BIT: an Online Algorithm for the List Update Problem

theory BIT

imports

Bit_Strings

MTF2_Effects

begin

abbreviation config'' A qs init n == config_rand A init (take n qs)
lemma `sum_my`: fixes f g::'b ⇒ 'a::ab_group_add
  assumes finite A finite B
  shows \((\sum x \in A. f x) - (\sum x \in B. g x)\) \n  \qquad = \((\sum x \in (A \cap B). f x - g x) + (\sum x \in A - B. f x) - (\sum x \in B - A. g x)\) \nproof - 
  have finite (A - B) and finite (A ∩ B) and finite (B - A) and finite (B ∩ A) 
using assms by auto
  note finites=this 
  have \((A - B) \cap (A \cap B)\) = \(\{\}\) and \((B - A) \cap (B \cap A)\) = \(\{\}\) by auto 
  note inters=this 
  have commute: A ∩ B = B ∩ A by auto 
  have A = (A - B) ∪ (A ∩ B) and B = (B - A) ∪ (B ∩ A) by auto 
  then have \((\sum x \in A. f x) - (\sum x \in B. g x) = \sum x \in (A - B) ∪ (A ∩ B). f x\) \n- (\sum x \in (B - A) ∪ (B ∩ A). g x) by auto 
  also have \ldots = \((\sum x \in (A - B). f x) + (\sum x \in (A ∩ B). f x)\) 
- (\sum x \in (B - A). g x) + (\sum x \in (B ∩ A). g x) - (\sum x \in (B - A) ∩ (A ∩ B). g x) 
using sum_Un[where \(f=f, OF\) finites(1) finites(2)] sum_Un[where \(f=g, OF\) finites(3) finites(4)] by(simp) 
  also have \ldots = \((\sum x \in (A - B). f x) + (\sum x \in (A ∩ B). f x)\) 
- (\sum x \in (B - A). g x) - (\sum x \in (B ∩ A). g x) using inters by auto 
  also have \ldots = \((\sum x \in (A - B). f x) - (\sum x \in (A ∩ B). g x) + (\sum x \in (A ∩ B). f x)\) 
- (\sum x \in (B - A). g x) using commute by auto 
  also have \ldots = \((\sum x \in (A ∩ B). f x - g x) + (\sum x \in (A - B). f x)\) 
- (\sum x \in (B - A). g x) using sum_subtract[of f g (A ∩ B)] by auto 
  finally show \(?thesis\).
qed

lemma `sum_my2`: \(\forall x \in A. f x = g x\) \implies \((\sum x \in A. f x) = (\sum x \in A. g x)\) 
by auto 

9.1 Definition of BIT

definition `BIT_init`: \('a state, bool list * 'a list)alg_on_init where 
  BIT_init init = map_pmf (λl. (l,init)) (bv (length init))

lemma \sim determinantistic_init BIT_init 
unfolding determinantistic_init_def BIT_init_def apply(auto)
apply (intro exI [where \(x=a\)])
— comment in a proof
by (auto simp: UNIV_bool set_pmf_bernoulli)

definition BIT_step :: ('a state, bool list * 'a list, 'a, answer)alg_on_step where
BIT_step s q = (let a = ((if (fst (snd s)))![index (snd (snd s)) q) then 0 else (length (fst s)))[]) in
return_pmf (a, (flip (index (snd (snd s)) q) (fst (snd s)), snd (snd s)))

lemma deterministic_step BIT_step
unfolding deterministic_step_def BIT_step_def
by simp

abbreviation BIT :: ('a state, bool list * 'a list, 'a, answer)alg_on_rand where
BIT == (BIT_init, BIT_step)

9.2 Properties of BIT’s state distribution

lemma BIT_no_paid: \(\forall s q. (\text{paid}[]=\) BIT_step s q\)
unfolding BIT_step_def
by (auto)

9.2.1 About the Internal State

term (config’rand (BIT_init, BIT_step) s0 qs)
lemma config’n_init: fixes qs init n
shows map_pmf (snd \circ snd) (config’rand (BIT_init, BIT_step) init qs) = map_pmf (snd \circ snd) init
apply (induct qs arbitrary: init)
by (simp_all add: map_pmf_def bind_assoc_pmf BIT_step_def bind_return_pmf )

lemma config’n_init: map_pmf (snd \circ snd) (config_rand (BIT_init, BIT_step) s0 qs) = return_pmf s0
using config’n_init[of ((fst (BIT_init, BIT_step) s0) \Rightarrow (\lambda is. return_pmf (s0, is)))]
by (simp_all add: map_pmf_def bind_assoc_pmf bind_return_pmf BIT_init_def )
lemma config_n_init2: \forall (\langle (\_ , \_ ) \rangle) \in \text{set_pmf} \ (\text{config_rand} \ (\text{BIT_init}, \text{BIT_step}) \ \text{init} \ qs). \ x = \text{init}
proof (\text{rule, goal_cases})
  case (1 \ z)
  then have \mathrm{1}: \text{snd}(\text{snd} \ z) \in (\text{snd} \circ \text{snd}) \ ^{-1} \ \text{set_pmf} \ (\text{config_rand} \ (\text{BIT_init}, \text{BIT_step}) \ \text{init} \ qs)
    by force
  have (\text{snd} \circ \text{snd})^{-1} \ \text{set_pmf} \ (\text{config_rand} \ (\text{BIT_init}, \text{BIT_step}) \ \text{init} \ qs)
    = \text{set_pmf} \ (\text{map_pmf} \ (\text{snd} \circ \text{snd}) \ (\text{config_rand} \ (\text{BIT_init}, \text{BIT_step}) \ \text{init} \ qs)) \ by (\text{simp})
  also have \ldots = \{\text{init}\} \ apply (\text{simp only: config_n_init}) \ by \ simp
finally have \text{snd}(\text{snd} \ z) = \text{init} \ using 1 by auto
then show \ ?case by auto
qed
lemma config_n_init3: \forall x \in \text{set_pmf} \ (\text{config_rand} \ (\text{BIT_init}, \text{BIT_step}) \ \text{init} \ qs). \ \text{snd} \ (\text{snd} \ x) = \text{init}
using config_n_init2 by (simp add: split_def)

lemma config'_n_bv: \text{fixes} \ qs \ \text{init} \ n
  \text{shows} \ \text{map_pmf} \ (\text{snd} \circ \text{snd}) \ \text{init} = \text{return_pmf} \ s0
  \implies \text{map_pmf} \ (\text{fst} \circ \text{snd}) \ \text{init} = \text{bv} \ (\text{length} \ s0)
  \implies \text{map_pmf} \ (\text{snd} \circ \text{snd}) \ (\text{config'_rand} \ (\text{BIT_init}, \text{BIT_step}) \ \text{init} \ qs)
  = \text{return_pmf} \ s0
  \land \text{map_pmf} \ (\text{fst} \circ \text{snd}) \ (\text{config'_rand} \ (\text{BIT_init}, \text{BIT_step}) \ \text{init} \ qs) \ = \text{bv} \ (\text{length} \ s0)
proof (\text{induct} \ qs \ \text{arbitrary: init})
  case (\text{Cons} \ r \ rs)
  from \text{Cons}(2) \ have \ a: \ \text{map_pmf} \ (\text{snd} \circ \text{snd}) \ (\text{init} \ \implies (\lambda s. \ \text{snd} \ (\text{BIT_init}, \text{BIT_step}) \ s) \ r \ a, \ \text{is'}))
    = \text{return_pmf} \ s0 \ apply (\text{simp add: BIT_step_def})
    by (\text{simp_all add: map_pmf_def bind_assoc_pmf BIT_step_def bind_return_pmf})
  then have \ b: \ \forall z\in \text{set_pmf} \ (\text{init} \ \implies (\lambda s. \ \text{snd} \ (\text{BIT_init}, \text{BIT_step}) \ s) \ r \ \implies (\lambda (a, \ \text{is'}). \ \text{return_pmf} \ (\text{step} \ (\text{fst} \ s) \ r \ a, \ \text{is'})))). \ \text{snd} \ (\text{snd} \ z) = s0
    by (\text{metis} \ \text{mono_tags, lifting}) \ \text{comp_eq_dest_lhs map_pmf_eq_return_pmf_iff})
show \ ?case
  apply (\text{simp only: config'_rand.simps})
proof (\text{rule Cons}(1), goal_cases)
  case 2
have \( \text{map pmf} \ (\text{fst} \circ \text{snd}) \)

\((\text{init} \gg= \lambda. \ \text{snd} \ (\text{BIT} \_\text{init}, \ \text{BIT} \_\text{step}) \ s \ r \gg= \lambda(a, \ \text{is}'). \ \text{return pmf} \ (\text{step} \ (\text{fst} \ s \ r \ a, \ \text{is}')))) = \text{map pmf} \ (\text{flip} \ (\text{index} \ s0 \ r)) \ (\text{bv} \ (\text{length} \ s0)) \)

using \( b \)

apply (simp add: \( \text{BIT} \_\text{step_def} \) \( \text{Cons(3)[symmetric]} \) \( \text{bind_return_pm} \) \( \text{pmf_def} \) \( \text{bind_assoc_pm} \) \( \text{pmf} \) )

apply (rule \( \text{bind_pm}_n \) \( \text{cong} \) )

apply (simp)

by (simp add: \( \text{inv_flip_bv} \) )

also have \( \ldots = \text{bv} \ (\text{length} \ s0) \) using \( \text{inv_flip_bv} \) by auto

finally show \(?\mathrm{case} \).

qed (fact)

qed simp


lemma \( \text{config_n bv 2}: \text{map pmf} \ (\text{snd} \circ \text{snd}) \ (\text{config_rand} \ (\text{BIT} \_\text{init}, \ \text{BIT} \_\text{step}) \ s0 \ qs) = \text{return pmf} \ s0 \)

\( \land \ \text{map pmf} \ (\text{fst} \circ \text{snd}) \ (\text{config_rand} \ (\text{BIT} \_\text{init}, \ \text{BIT} \_\text{step}) \ s0 \ qs) = \text{bv} \ (\text{length} \ s0) \)

apply (rule \( \text{config'_n bv} \) )

by (simp_all add: \( \text{bind_return_pm} \) \( \text{map_pm} \) \( \text{def} \) \( \text{bind_assoc_pm} \) \( \text{pmf} \) \( \text{bind_return_pm} \) \( \text{pmf} \) \( \text{BIT} \_\text{init_def} \) )


lemma \( \text{config_n bv}: \text{map pmf} \ (\text{fst} \circ \text{snd}) \ (\text{config_rand} \ (\text{BIT} \_\text{init}, \ \text{BIT} \_\text{step}) \ s0 \ qs) = \text{bv} \ (\text{length} \ s0) \)

using \( \text{config_n bv 2} \) by auto


lemma \( \text{config_n fst init length}: \forall (\_ (x, \_)) \in \text{set pmf} \ (\text{config_rand} \ (\text{BIT} \_\text{init}, \ \text{BIT} \_\text{step}) \ s0 \ qs). \ \text{length} \ x = \text{length} \ s0 \)

proof

fix \( x::('a list \times (bool list \times 'a list)) \)

assume \( \text{ass} \): \( x \in \text{set pmf} \ (\text{config_rand} \ (\text{BIT} \_\text{init}, \ \text{BIT} \_\text{step}) \ s0 \ qs) \)

let \( ?a = \text{fst} \ (\text{snd} \ x) \)

from \( \text{ass} \) have \( (\text{fst} \ x, (?a, \text{snd} \ (\text{snd} \ x))) \in \text{set pmf} \ (\text{config_rand} \ (\text{BIT} \_\text{init}, \ \text{BIT} \_\text{step}) \ s0 \ qs) \) by auto

with \( \text{ass} \) have \( ?a \in (\text{fst} \circ \text{snd}) \ (\text{set pmf} \ (\text{config_rand} \ (\text{BIT} \_\text{init}, \ \text{BIT} \_\text{step}) \ s0 \ qs)) \) by force

then have \( ?a \in \text{set pmf} \ (\text{map pmf} \ (\text{fst} \circ \text{snd}) \ (\text{config_rand} \ (\text{BIT} \_\text{init}, \ \text{BIT} \_\text{step}) \ s0 \ qs)) \) by auto
then have \( ?a \in bv((\text{length } s0)) \) by simp only: \( \text{config}\_n\_bv \)
then have \( \text{length } ?a = \text{length } s0 \) by (auto simp: \( \text{len}\_bv\_n \))
then show case \( x \) of (\( \_ \), \( xa \), \( uua \)) \( \Rightarrow \) \( \text{length } xa = \text{length } s0 \) by simp
add: split_def
qed

lemma \( \text{config}\_n\_fst\_init\_length2: \forall x \in \text{set}\_pmf (\text{config}\_rand (\text{BIT}\_init, \text{BIT}\_step) s0 \text{ qs}), \text{length} (\text{fst} (\text{snd} x)) = \text{length } s0 \) using \( \text{config}\_n\_fst\_init\_length \) by (simp add: split_def)

lemma \( fperms: \text{finite} \{x: \text{a list}. \text{length} x = \text{length } \text{init} \wedge \text{distinct} x \wedge \text{set} x = \text{set } \text{init}\} \) apply (rule finite_subset[where \( B = \{xs. \text{set} xs \subseteq \text{set } \text{init} \wedge \text{length} xs \leq \text{length } \text{init}\}]])
apply (force) apply (rule finite_lists_length_le) by auto

lemma \( \text{finite}\_\text{config}\_\text{BIT}: \text{assumes} \) [simp]: \( \text{distinct } \text{init} \)
shows \( \text{finite} (\text{set}\_pmf (\text{config}\_rand (\text{BIT}\_init, \text{BIT}\_step) \text{init} \text{ qs})) \) (is finite ?\( D \))
proof –
have \( a: (\text{fst} \circ \text{snd}) ' ?D \subseteq \{x. \text{length} x = \text{length } \text{init}\} \) using \( \text{config}\_n\_fst\_init\_length2 \) by force
have \( c: (\text{snd} \circ \text{snd}) ' ?D = \{\text{init}\} \)
proof –
have \( (\text{snd} \circ \text{snd}) ' \text{ set}\_pmf (\text{config}\_rand (\text{BIT}\_init, \text{BIT}\_step) \text{init} \text{ qs}) = \text{set}\_pmf (\text{map}\_pmf (\text{snd} \circ \text{snd}) (\text{config}\_rand (\text{BIT}\_init, \text{BIT}\_step) \text{init} \text{ qs})) \) by simp
also have \( \ldots = \{\text{init}\} \) apply (subst \( \text{config}\_n\_init \)) by simp
finally show ?thesis .
qed
from \( a \) \( c \) have \( d: \text{snd} ' ?D \subseteq \{x. \text{length} x = \text{length } \text{init}\} \times \{\text{init}\} \) by force
have \( b: \text{fst} ' ?D \subseteq \{x. \text{length} x = \text{length } \text{init} \wedge \text{distinct} x \wedge \text{set} x = \text{set } \text{init}\} \)
using \( \text{config}\_rand \) by fastforce
from \( b \) \( d \) have \( ?D \subseteq \{x. \text{length} x = \text{length } \text{init} \wedge \text{distinct} x \wedge \text{set} x = \text{set } \text{init}\} \times (\{x. \text{length} x = \text{length } \text{init}\} \times \{\text{init}\}) \)
by auto
then show ?thesis
apply (rule finite_subset)
apply (rule finite_cartesian_product)
apply (rule fperms)
apply (rule finite_cartesian_product)
  apply (rule bitstrings_finite)
  by (simp)

qed

9.3 BIT is 1.75-competitive (a combinatorial proof)

9.3.1 Definition of the Locale and Helper Functions

locale BIT_Off = 
fixes acts :: 'a list
fixes qs :: 'a list
fixes init :: 'a list
assumes dist_init[simp]: distinct init
assumes lenActs: length acts = length qs

begin

lemma setinit: (index init) ' set init = {0..<length init}
using dist_init
proof (induct init)
  case (Cons a as)
  with Cons have iH: index as ' set as = {0..<length as} by auto
  from Cons have 1: (set as ∩ {x. (a ≠ x)}) = set as by fastforce
  have 2: (λa. Suc (index as a)) ' set as =
    (λa. Suc a) ' ((index as) ' set as) by auto
  show ?case
  apply (simp add: 1 2 iH) by auto
qed simp

definition free_A :: nat list where
free_A = map fst acts

definition paid_A' :: nat list list where
paid_A' = map snd acts

definition paid_A :: nat list list where
paid_A = map (filter (λx. Suc x < length init)) paid_A'

lemma len_paid_A[simp]: length paid_A = length qs
unfolding paid_A_def paid_A'_def using lenActs by auto
lemma len_paid_A'[simp]: length paid_A' = length qs

87
unfolding \( \text{paid}_A \)'\(_n\) using \( \text{len}_\text{acts} \) by auto

\[\text{lemma} \ \text{paid}_A\text{\_in\_b\_\_n}\text{\_m} : n < \text{length} \ \text{paid}_A \implies m < \text{length}(\text{paid}_A ! n) \implies (\text{Suc} ((\text{paid}_A ! n)!((\text{length} (\text{paid}_A ! n) \ - \ \text{Suc} m))) < \text{length} \ \text{init} \]

proof –
\[\text{assume} \ n < \text{length} \ \text{paid}_A\]
then have \( n < \text{length} \ \text{paid}_A' \) by auto
then have \( a : (\text{paid}_A ! n) \)
\[= \text{filter} \ (\lambda x. \text{Suc} x < \text{length} \ \text{init}) (\text{paid}_A' ! n) \]
unfolding \( \text{paid}_A \)'\(_n\) by auto

let \( \text{?filtered} = (\text{filter} \ (\lambda x. \text{Suc} x < \text{length} \ \text{init}) (\text{paid}_A' ! n)) \)
assume \( \text{mtt} : m < \text{length} \ (\text{paid}_A ! n) \)
with \( a \) have \( (\text{length} \ (\text{paid}_A ! n) \ - \ \text{Suc} m) < \text{length} \ \text{?filtered} \) by auto
with \text{nth\_mem} have \( \text{Suc}(\text{?filtered} ! (\text{length} (\text{paid}_A ! n) \ - \ \text{Suc} m)) < \text{length} \ \text{init} \) by force

show \( \text{Suc} ((\text{paid}_A ! n) \ - \ \text{Suc} m) < \text{length} \ \text{init} \)
using \( a \) \( b \) by auto
qed

fun \( s_A' :: \text{nat} \Rightarrow 'a \text{ list where} \)
\( s_A' 0 = \text{init} \mid \)
\( s_A'(\text{Suc} n) = \text{step} \ (s_A' n) \ (qs!n) \ (\text{free}_A!n, \ \text{paid}_A?n) \)

\[\text{lemma} \ \text{length}_\text{s}_A'\text{[simp]} : \text{length}(s_A' n) = \text{length} \ \text{init} \]
by (induction \( n \)) simp_all

\[\text{lemma} \ \text{dist}_\text{s}_A'\text{[simp]} : \text{distinct}(s_A' n) \]
by (induction \( n \)) (simp_all add: \text{step\_def})

\[\text{lemma} \ \text{set}_\text{s}_A'\text{[simp]} : \text{set}(s_A' n) = \text{set} \ \text{init} \]
by (induction \( n \)) (simp_all add: \text{step\_def})

fun \( s_A :: \text{nat} \Rightarrow 'a \text{ list where} \)
\( s_A 0 = \text{init} \mid \)
\( s_A(\text{Suc} n) = \text{step} \ (s_A n) \ (qs!n) \ (\text{free}_A!n, \ \text{paid}_A!n) \)

\[\text{lemma} \ \text{length}_\text{s}_A\text{[simp]} : \text{length}(s_A n) = \text{length} \ \text{init} \]
by (induction \( n \)) simp_all

\[\text{lemma} \ \text{dist}_\text{s}_A\text{[simp]} : \text{distinct}(s_A n) \]
by (induction \( n \)) (simp_all add: \text{step\_def})
lemma set_s_A[simp]: set(s_A n) = set init
by(induction n) (simp_all add: step_def)

lemma cost_paidAA': n < length paid_A' \implies length (paid_A!n) \leq length (paid_A'!n)
unfolding paid_A_def by simp

lemma swaps_filtered: swaps (filter (\lambda x. Suc x < length xs) ys) xs = swaps (ys) xs
apply (induct ys) by auto

lemma sAsA': n < length paid_A' \implies s_A' n = s_A n
proof (induct n)
  case (Suc m)
  have s_A'(Suc m)
    = mtf2 (free_A!m) (qs!m) (swaps (paid_A!m) (s_A' m)) by (simp add: step_def)
  also from Suc(2) have ... = mtf2 (free_A!m) (qs!m) (swaps (paid_A!m) (s_A m))
    unfolding paid_A_def
    by (simp only: nth_map swaps_filtered[where xs=s_A' m, simplified])
  also have ... = s_A (Suc m) by (simp add: step_def)
  using Suc by auto
  also have ... = s_A (Suc m) by (simp add: step_def)
  finally show ?case .
qed simp

lemma sAsA'': n < length qs \implies s_A n = s_A' n
using sAsA' by auto

definition t_BIT :: nat \Rightarrow real where
t_BIT n = T_on_rand_n BIT init qs n

definition T_BIT :: nat \Rightarrow real where
T_BIT n = (\sum i<n. t_BIT i)

definition c_A :: nat \Rightarrow int where
c_A n = index (swaps (paid_A!n) (s_A n)) (qs!n) + 1

definition f_A :: nat \Rightarrow int where
\[ f_A n = \min (\text{free}_A!n) (\text{index} (\text{swaps} (\text{paid}_A!n) (s_A!n)) (qs!n)) \]

**definition** \( p_A :: \text{nat} \Rightarrow \text{int} \) where  
\[ p_A n = \text{size} (\text{paid}_A!n) \]

**definition** \( t_A :: \text{nat} \Rightarrow \text{int} \) where  
\[ t_A n = c_A n + p_A n \]

**definition** \( c_A' :: \text{nat} \Rightarrow \text{int} \) where  
\[ c_A' n = \text{index} (\text{swaps} (\text{paid}_A'!n) (s_A'!n)) (qs!n) + 1 \]

**definition** \( p_A' :: \text{nat} \Rightarrow \text{int} \) where  
\[ p_A' n = \text{size} (\text{paid}_A'!n) \]

**definition** \( t_A' :: \text{nat} \Rightarrow \text{int} \) where  
\[ t_A' n = c_A' n + p_A' n \]

**lemma** \( t_A A' \leq n < \text{length paid}_A \Rightarrow t_A n \leq t_A' n \)

**unfolding** \( t_A \text{def} t_A' \text{def} c_A \text{def} c_A' \text{def} p_A \text{def} p_A' \text{def} \)

**apply** (simp add: sAsA')

**unfolding** \( \text{paid}_A \text{def} \)

**by** (simp add: \( \text{swaps}_\text{filtered}[\text{where} \ xs=(s_A!n), \ \text{simplified}] \))

**definition** \( T_A' :: \text{nat} \Rightarrow \text{int} \) where  
\[ T_A' n = \left( \sum_{i<n} t_A' i \right) \]

**definition** \( T_A :: \text{nat} \Rightarrow \text{int} \) where  
\[ T_A n = \left( \sum_{i<n} t_A i \right) \]

**lemma** \( T_A A' \leq n < \text{length paid}_A \Rightarrow T_A n \leq T_A' n \)

**unfolding** \( T_A' \text{def} T_A \text{def} \)

**apply** (rule sum_mono)

**by** (simp add: \( T_A A' \text{leq} \))

**lemma** \( T_A A' \leq n < \text{length qs} \Rightarrow T_A n \leq T_A' n \)

**using** \( T_A A' \leq \text{by} \ \text{auto} \)

**fun** \( s' A :: \text{nat} \Rightarrow \text{nat} \Rightarrow 'a \text{ list} \) where  
\[ s' A n 0 = s_A n \mid (s' A n \ (\text{Suc} m)) = \text{swap} ((\text{paid}_A \ n)!n)(\text{length} (\text{paid}_A \ n)!n - (\text{Suc} m)) \]

**lemma** \( \text{set} s' A[n \ m] \text{simp} : \text{set} (s' A n \ m) = \text{set init} \)

90
apply \((\text{induct } m)\) by \((\text{auto})\)

lemma \(\text{len}_s' A\)\[\text{simp}\]: \(\text{length} (s' A n m) = \text{length init}\)
apply \((\text{induct } m)\) by \((\text{auto})\)

lemma \(\text{distperm}_s' A\)\[\text{simp}\]: \(\text{dist_perm} (s' A n m)\) init
apply \((\text{induct } m)\) by auto

lemma \(s'_A m le\)\[m \leq (\text{length} (\text{paid}_A ! n)) \implies \text{swaps} (\text{drop} (\text{length} (\text{paid}_A ! n) - m) (\text{paid}_A ! n)) (s_A n) = s'_A n m\]
apply \((\text{induct } m)\)
apply \((\text{simp})\)
proof
−
fix \(m\)
assume \(iH): \(m \leq \text{length} (\text{paid}_A ! n) \implies \text{swaps} (\text{drop} (\text{length} (\text{paid}_A ! n) - m) (\text{paid}_A ! n)) (s_A n) = s'_A n m\)
assume \(\text{Suc}: \text{Suc } m \leq \text{length} (\text{paid}_A ! n)\)
then have \(m \leq \text{length} (\text{paid}_A ! n)\) by auto
with \(iH\) have \(x: \text{swaps} (\text{drop} (\text{length} (\text{paid}_A ! n) - m) (\text{paid}_A ! n)) (s_A n) = s'_A n m\) by auto

from \(\text{Suc}\) have \(m\text{len}: \text{length} (\text{paid}_A ! n) - \text{Suc } m < \text{length} (\text{paid}_A ! n)\) by auto

let \(?l=\text{length} (\text{paid}_A ! n) - \text{Suc } m\)
let \(?Sucl=\text{length} (\text{paid}_A ! n) - m\)
have \(\text{Sucl}\): \(\text{Suc } ?l = ?Sucl\) using \(\text{Suc}\) by auto

from \(m\text{len}\) have \(\text{yu}: ((\text{paid}_A ! n)! ?l ) \# (\text{drop} (\text{Suc } ?l) (\text{paid}_A ! n)) = (\text{drop} ?l (\text{paid}_A ! n))\)
by \((\text{rule Cons\_nth\_drop\_Suc})\)

from \(\text{Suc}\) have \(s'_A n (\text{Suc } m)\)
= swap \(((\text{paid}_A ! n)!(\text{length} (\text{paid}_A ! n) - (\text{Suc } m)) \) (s'_A n m)\)
by auto
also have \(\ldots = \text{swap} \(((\text{paid}_A ! n)!(\text{length} (\text{paid}_A ! n) - (\text{Suc } m)) \) (\text{swaps} (\text{drop} (\text{length} (\text{paid}_A ! n) - m) (\text{paid}_A ! n)) (s_A n))\)
by \((\text{simp only: } x)\)
also have \(\ldots = \text{(swaps} (((\text{paid}_A ! n)!((\text{length} (\text{paid}_A ! n) - (\text{Suc } m)) \) \# (\text{drop} (\text{length} (\text{paid}_A ! n) - m) (\text{paid}_A ! n))) (s_A n))\)
by auto
also have \(\ldots = \text{(swaps} (((\text{paid}_A ! n)! ?l ) \# (\text{drop} (\text{Suc } ?l) (\text{paid}_A ! n))) (s_A n))\)

91
using Suc by auto
also from mlen have ... = (swaps ((drop ?l (paid_A ! n))) (s_A n))
  by (simp only: yu)
finally have s'_A n (Suc m) = swaps (drop (length (paid_A ! n) - Suc m) (paid_A ! n)) (s_A n).
  then show \( \text{swaps} \left( \text{drop} \left( \text{length} \left( \text{paid}_A ! n \right) - \text{Suc} \ m \right) \left( \text{paid}_A ! n \right) \right) \left( s_A n \right) = s'_A n \left( \text{Suc} \ m \right) \) by auto
qed

lemma s'_A_m: \( \text{swaps} \left( \text{paid}_A ! n \right) \left( s_A n \right) = s'_A n \left( \text{length} \left( \text{paid}_A ! n \right) \right) \) using s'_A_m_len[of \( \text{length} \left( \text{paid}_A ! n \right) \)] n, simplified by auto

definition gebub :: nat ⇒ nat ⇒ nat where
  gebub n m = index init ((s'_A n m)!(Suc ((paid_A!n)!(length (paid_A ! n) - Suc m))))

lemma gebub_inBound: assumes 1: \( n < \text{length} \text{paid}_A \) and 2: \( m < \text{length} \left( \text{paid}_A ! n \right) \)
  shows gebub n m < length init
proof –
  have Suc (paid_A ! n ! (length (paid_A ! n) - Suc m)) < length (s'_A n m)
    using paidAnm_inbound[OF 1 2] by auto
  then have s'_A n m ! Suc (paid_A ! n ! (length (paid_A ! n) - Suc m)) 
    ∈ set (s'_A n m) by (rule nth_mem)
  then show ?thesis
    unfolding gebub_def using setinit by auto
qed

9.3.2 The Potential Function

fun phi :: nat ⇒ 'a list × (bool list × 'a list) ⇒ real (ϕ) where
phi n (c,(b,)) = (∑ (x,y)∈(Inv c (s_A n)). (if b!(index init y) then 2 else 1))

lemma phi': phi n z = (∑ (x,y)∈(Inv (fst z) (s_A n)). (if (fst (snd z))!(index init y) then 2 else 1))
proof –
  have phi n z = phi n (fst z, (fst (snd z),snd (snd z))) by (metis prod.collapse)
  also have ... = (∑ (x,y)∈(Inv (fst z) (s_A n)). (if (fst (snd z))!(index init y) then 2 else 1)) by(simp del: prod.collapse)
  finally show ?thesis .
qed
lemma Inv_empty2: length d = 0 \implies Inv c d = {}
unfolding Inv_def before_in_def by (auto)

corollary Inv_empty3: length init = 0 \implies Inv c (s_A n) = {}
apply (rule Inv_empty2) by (metis length s_A)

lemma phi_empty2: length init = 0 \implies phi n (c, (b, i)) = 0
apply (simp only: phi.simps Inv_empty3) by auto

lemma phi_nonzero: phi n (c, (b, i)) \geq 0
by (simp add: sum_nonneg split_def)

definition Phi :: nat \Rightarrow real (\Phi)
where
Phi n = E (map pmf (\varphi n) (config'' BIT qs init n))

definition PhiPlus :: nat \Rightarrow real (\Phi^+) where
PhiPlus n = (let
nextconfig = bind pmf (config'' BIT qs init n)
(\lambda (s, is). bind pmf (BIT_step (s, is) (qs!n)) (\lambda (a, nis). return_pmf
(step s (qs!n) a, nis)))

in
E (map pmf (phi (Suc n)) nextconfig))

lemma PhiPlus_is_Phi_Suc: n < length qs \implies PhiPlus n = Phi (Suc n)
unfolding PhiPlus_def Phi_def
apply (simp add: bind_return_pmf map_pmf_def bind_assoc_pmf split_def take_Suc_conv_app_nth)
apply (simp add: config'_rand_snoc)
by (simp add: bind_assoc_pmf split_def bind_return_pmf)

lemma phi0: Phi 0 = 0 unfolding Phi_def
by (simp add: bind_return_pmf map_pmf_def bind_assoc_pmf BIT_init_def)

lemma phi_pos: Phi n \geq 0
unfolding Phi_def
apply (rule E_nonneg_fun)
using phi_nonzero by auto

9.3.3 Helper lemmas

lemma swap_subs: dist_perm X Y \implies Inv X (swap z Y) \subseteq Inv X Y \cup
{(Y ! z, Y ! Suc z)}
proof –
\begin{verbatim}
assume dist_perm X Y
note aj = Inv_swap[OF this, of z]
show Inv X (swap z Y) ⊆ Inv X Y ∪ \{(Y ! z, Y ! Suc z)\}
proof cases
  assume c1: Suc z < length X
  show Inv X (swap z Y) ⊆ Inv X Y ∪ \{(Y ! z, Y ! Suc z)\}
  proof cases
    assume Y ! z < Y ! Suc z in X
    with c1 have Inv X (swap z Y) = Inv X Y ∪ \{(Y ! Suc z, Y ! z)\}
    using aj by auto
    then show Inv X (swap z Y) ⊆ Inv X Y ∪ \{(Y ! z, Y ! Suc z)\} by auto
  next
    assume ~ Y ! z < Y ! Suc z in X
    with c1 have Inv X (swap z Y) = Inv X Y - \{(Y ! Suc z, Y ! z)\}
    using aj by auto
    then show Inv X (swap z Y) ⊆ Inv X Y ∪ \{(Y ! z, Y ! Suc z)\} by auto
  qed
  qed

9.3.4 InvOf

term Inv
abbreviation InvOf y bits as \equiv \{(x,y) | x < y in bits \land y < x in as\}
lemma InvOf y xs ys = \{(x,y) | (x,y) ∈ Inv xs ys\}
unfolding Inv_def by auto

lemma InvOf y xs ys ⊆ Inv xs ys unfolding Inv_def by auto

lemma numberofIsbeschr: assumes
distxsys: dist_perm xs ys and
yinzs: y ∈ set xs
shows index xs y ≤ index ys y + \text{card} (InvOf y xs ys)
  (is \textnormal{?iB} ≤ \textnormal{?iA} + \text{card} \textnormal{?I})
proof
  from assms have distinctxs: distinct xs
...
\end{verbatim}
and distinctys: distinct ys 
and yinys: y ∈ set ys by auto

let ?A=fst ' ?I  
have aha: card ?A = card ?I apply(rule card_image)  
    unfolding inj_on_def by(auto)  

have ?A ⊆ (before y xs) by(auto)  
have ?A ⊆ (after y ys) by auto

have finite (before y ys) by auto

have bef: (before y xs) — ?A ⊆ before y ys apply(auto)  
proof —  
  fix x  
  assume a: x < y in xs  
  assume x ∉ fst ' {(x, y) | x, x < y in xs ∧ y < x in ys}  
  then have ∼ (x < y in xs ∧ y < x in ys) by force  
  with a have d: ∼ y < x in ys by auto  
  from a have x ∈ set xs by (rule before_in_setD1)  
  with distxsys have b: x ∈ set ys by auto  
  from a have y ∈ set xs by (rule before_in_setD2)  
  with distxsys have c: y ∈ set ys by auto  
  from a have e: ∼ x = y unfolding before_in_def by auto  
  have (∼ y < x in ys) = (x < y in ys ∨ y = x) apply(rule not_before_in)  
    using b c by auto  
  with d e show x < y in ys by auto  
qed

have (index xs y) — card (InvOf y xs ys) = card (before y xs) — card ?A  
    by(simp only: aha card_before[OF distinctxs yinxs])  
also have ... = card ((before y xs) — ?A)  
    apply(rule card_Diff_subset[symmetric]) by auto  
also have ... ≤ card (before y ys)  
    apply(rule card_mono)  
    apply(simp)  
    apply(rule bef)  
    done  
also have ... = (index ys y) by(simp only: card_before[OF distinctys yinys])  
finally have index xs y — card ?I ≤ index ys y .
then show index xs y ≤ index ys y + card ?I by auto
lemma length init = 0 ⇒ length xs = length init ⇒ t xs q (mf, sws) = 1 + length sws

unfolding t_def by(auto)

9.3.5 Upper Bound on the Cost of BIT

lemma t_BIT_ub2: (qs!n) ∉ set init ⇒ t_BIT n ≤ Suc(size init)
apply(simp add: t_BIT_def t_def BIT_step_def)
apply(simp add: bind_return_pmf)
proof (goal_cases)
case 1
  note qs=this
  let ?D = (config" (BIT_init, BIT_step) qs init n)
  have absch: (∀ x ∈ set_pmf ?D. ((λ(s,is). real (Suc (index s (qs ! n)))) x) ⪯ ((λ(is,s). Suc (length init)) x))
    proof (rule ballI, goal_cases)
      case 1 x
      from 1 config_rand_length have f1: length (fst x) = length init by fastforce
      from 1 config_rand_set have 2: set (fst x) = set init by fastforce
    from qs 2 have (qs!n) ∉ set (fst x) by auto
    then show ?case using f1 by (simp add: split_def)
  qed

have integrable (measure_pmf (config" (BIT_init, BIT_step) qs init n))
  (λ(s, is). Suc (length init)) by(simp)

have E(bind_pmf ?D (λ(s, is). return_pmf (real (Suc (index s (qs ! n)))))))
  = E(map_pmf (λ(s, is). real (Suc (index s (qs ! n)))) ?D)

96
by (simp add: split_def map_pmf_def)
also have \ldots \leq E (map_pmf (\lambda(s, is). Suc (length init)) ?D)
  apply (rule E_mono3)
  apply (fact integr_index)
  apply (simp)
  using absch by auto
also have \ldots = Suc (length init)
  by (simp add: split_def)
finally show \?case by (simp add: map_pmf_def bind_assoc_pmf bind_return_pmf
split_def)
qed

lemma t_BIT_ub: (qs!n) \in set init \implies t_BIT n \leq size init
apply (simp add: t_BIT_def t_def BIT_step_def)
apply (simp add: bind_return_pmf)
proof (goal_cases)
case 1
note qs\=this
let ?D = (config" (BIT_init, BIT_step) qs init n)

  have absch: (\forall x \in set_pmf ?D. ((\lambda(s, is). real (Suc (index s (qs ! n))))
x) \leq ((\lambda(s, is). length init) x))
    proof (rule ballI, goal_cases)
      case (1 x)
      from 1 config_rand_length have f1: length (fst x) = length init by fastforce
      from 1 config_rand_set have 2: set (fst x) = set init by fastforce

      from qs 2 have (qs!n) \in set (fst x) by auto
      then have (index (fst x) (qs ! n)) < length init apply (rule index_less)
using f1 by auto
      then show \?case by (simp add: split_def)
qed

have E (bind_pmf ?D (\lambda(s, is). return_pmf (real (Suc (index s (qs ! n)))))))
  = E (map_pmf (\lambda(s, is). real (Suc (index s (qs ! n)))) ?D)
  by (simp add: split_def map_pmf_def)
also have \ldots \leq E (map_pmf (\lambda(s, is). length init) ?D)
  apply (rule E_mono3)
  apply (fact integr_index)
  apply (simp)
  using absch by auto
also have \ldots = length init
  by (simp add: split_def)
finally show \( \text{?case by (simp add: map_pmf_def bind_assoc_pmf bind_return_pmf split_def)} \)
qed

lemma \( T_{\text{BIT}} \_ \text{ub} \): \( \forall i < n. \ qs!i \in \text{set init} \implies T_{\text{BIT}} n \leq n * \text{size init} \)
proof (induction n)
  case 0 show \( \text{?case by (simp add: T_{\text{BIT}} \_ \text{def})} \)
next
  case (Suc n) thus \( \text{?case} \)
  using \( t_{\text{BIT}} \_ \text{ub} \[ \text{where n=n} \] \) by (simp add: T_{\text{BIT}} \_ \text{def})
qed

9.3.6 Main Lemma

lemma myub: \( n < \text{length qs} \implies t_{\text{BIT}} n + \Phi(n + 1) - \Phi n \leq (7 / 4) * t_{\text{A}} n - 3 / 4 \)
proof –
  assume \( nqs: n < \text{length qs} \)
  have \( t_{\text{BIT}} n + \Phi (n+1) - \Phi n \leq (7 / 4) * t_{\text{A}} n - 3 / 4 \)
  proof (cases length init > 0)
    case False
    show \( \text{?thesis} \)
    proof –
      from False have \( qsn: (qs!n) \notin \text{set init by auto} \)
      from False have \( l0: \text{length init} = 0 \) by auto
      then have \( \text{length (swaps (paid_\text{A} ! n) (s_\text{A} n)) = 0 using length_s_\text{A}} \)
      by auto

      with \( l0 \) have \( 4: t_{\text{A}} n = 1 + \text{length (paid_\text{A} ! n)} \) unfolding t_{\text{A}} \_ \text{def}
      c_{\text{A}} \_ \text{def p}_{\text{A}} \_ \text{def by (simp)}

      have \( 1: t_{\text{BIT}} n \leq 1 \) using \( t_{\text{BIT}} \_ \text{ub} \_2 \[ \text{OF qsn} \] \) \( l0 \) by auto

      \{ fix \( m \)
      have \( \text{phi m} = (\lambda(b,(a,i)). \text{phi m} (b,(a,i))) \) by auto
      also have \( \ldots \) = \( (\lambda(b,(a,i)). \ 0) \) by (simp only: phi_empty2[OF \( l0 \)])
      finally have \( \text{phi m} = (\lambda(b,(a,i)). \ 0) \).
      \} note \( \text{phinull= \this} \)

      have \( 2: \Phi_{\text{Plus}} n = 0 \) unfolding \( \Phi_{\text{Plus}} \_ \text{def} \) apply (simp) apply (simp only: \( \text{phinull} \))
      apply (simp only: \( \text{phinull} \))
      by (auto simp: split_def)

      have \( 3: \Phi n = 0 \) unfolding \( \Phi \_ \text{def} \) apply (simp only: \( \text{phinull} \))
      by (auto simp: split_def)
have \( t \cdot A \ n \geq 1 \implies 1 \leq 7 / 4 \ast (t \cdot A \ n) - 3 / 4 \) by (simp)
with \( 4 \) have \( 5: 1 \leq 7 / 4 \ast (t \cdot A \ n) - 3 / 4 \) by auto

from \( 1 \ 2 \ 3 \) have \( t \cdot \text{BIT} \ n + \Phi i \text{Phi} + \text{Phi} n \leq 1 \) by auto
also from \( 5 \) have \( \ldots \leq 7 / 4 \ast (t \cdot A \ n) - 3 / 4 \) by auto

finally show \( ?\text{thesis} \) using \( \Phi i \text{Phi} + \text{Phi} \_\text{Suc} \) nqs by auto
qed

next
  case True
  let \(?l = \text{length init}\)
from True obtain \( l' \) where \( l\_\text{Suc} : ?l = \text{Suc} \_\text{l}' \) by (metis \( \text{Suc} \_\text{pred} \))

have \( 31: \ n < \text{length paid}_A \) using nqs by auto

define \( q \) where \( q = q \_s!n \)
define \( D \) where [simp]: \( D = (\text{config}'' (\text{BIT} \_\text{init}, \text{BIT} \_\text{step} \ q s \ \text{init} \ n)) \)
  define \( \text{cost} \) where [simp]: \( \text{cost} = (\lambda (s, \ is). (t \ s \ q \ (\text{if} (\text{fst} \ is) \ ! (\text{index} (\text{snd} \ is) \ q) \text{then} 0 \ \text{else} \ \text{length} \ s, [])))) \)
  define \( \Phi_2 \) where [simp]: \( \Phi_2 = (\lambda (s, \ is). ((\text{phi} (\text{Suc} \ n)) (\text{step} \ s \ q \ (\text{if} (\text{fst} \ is) \ ! (\text{index} (\text{snd} \ is) \ q) \text{then} 0 \ \text{else} \ \text{length} \ s, []))(\text{flip} (\text{index} (\text{snd} \ is) \ q) (\text{fst} \ is), \text{snd} is)))) \)
  define \( \Phi_0 \) where [simp]: \( \Phi_0 = \text{phi} \ n \)

have inEreinziehn: \( t \cdot \text{BIT} \ n + \Phi i (n+1) - \Phi n = E \ (\text{map}_\text{pmf} (\lambda x. (\text{cost} \ x) + (\Phi_2 \ x) - (\Phi_0 \ x))) \ D \)
proof
  have bind\_pmf \( D \)
  \( (\lambda (s, \ is). \ \text{bind}_\text{pmf} (\text{BIT} \_\text{step} \ s, \ is) (q)) \ (\lambda (a, \nis). \ \text{return}_\text{pmf} \ (\text{real}(t \ s \ (q) \ a)))) \)
  \( = \ \text{bind}_\text{pmf} \ D \)
  \( (\lambda (s, \ is). \ \text{return}_\text{pmf} \ (t \ s \ q \ (\text{if} (\text{fst} \ is) \ ! (\text{index} (\text{snd} \ is) \ q) \text{then} 0 \ \text{else} \ \text{length} \ s, [])))) \)
  unfolding \( \text{BIT} \_\text{step} \_\text{def} \) \( \text{apply} \) (auto simp: bind\_return\_pmf \( \text{split} \_\text{def} \))
  by (metis prod.collapse)
also have \( \ldots = \text{map}_\text{pmf} \text{cost} \ D \)
  by (auto simp: map\_pmf\_def \( \text{split} \_\text{def} \))
finally have right\_form\_1: \( \text{bind}_\text{pmf} \ D \)
  \( (\lambda (s, \ is). \ \text{bind}_\text{pmf} (\text{BIT} \_\text{step} \ s, \ is) (q)) \ (\lambda (a, \nis). \ \text{return}_\text{pmf} \ (\text{real}(t \ s \ (q) \ a)))) \)
  \( = \ \text{map}_\text{pmf} \text{cost} \ D \)
have rightform2: map pmf (phi (Suc n)) (bind pmf D
  (λ(s, is), bind pmf (BIT_step (s, is) (q)) (λ(a, nis). return pmf
  (step s (q) a, nis))))
  = map pmf \Phi_2 D apply(simp add: bind_return_pmf bind_assoc_pmff
map_pmf_def split_def BIT_step_def)
  by (metis prod.collapse)
have t_BIT n + Phi (n+1) - Phi n =
t_BIT n + PhiPlus n - Phi n using PhiPlus_is_Phi_Suc nqs by auto
also have ... =
  T_on_rand_n BIT init qs n
  + E (map pmf (phi (Suc n)) (bind pmf D
    (λ(s, is), bind pmf (BIT_step (s, is) (q)) (λ(a, nis). return pmf
  (step s (q) a, nis))))
  - E (map pmf (phi n) D)
  unfolding PhiPlus_def Phi_def t_BIT_def q_def by auto
also have ... = E (map pmf cost D)
  + E (map pmf \Phi_0 D)
  - E (map pmf \Phi_0 D) using rightform1 rightform2 split_def
by auto
  also have ... = E (map pmf (λx. (cost x) + (\Phi_2 x)) D) - E
(map pmf (λx. (\Phi_0 x)) D)
  unfolding D_def using E_linear_plus2[OF finite_config_BIT[OF
dist_init]] by auto
  also have ... = E (map pmf (λx. (cost x) + (\Phi_2 x) - (\Phi_0 x)) D)
  unfolding D_def by(simp only: E_linear_diff2[OF finite_config_BIT[OF
dist_init]] split_def)
finally show t_BIT n + Phi (n+1) - Phi n
  = E (map pmf (λx. (cost x) + (\Phi_2 x) - (\Phi_0 x)) D) by auto
qed

define xs where [simp]: xs = s_A n
define xs' where [simp]: xs' = swaps (paid_A!n) xs
define xs'' where [simp]: xs'' = mtf2 (free_A!n) (q) xs'
define k where [simp]: k = index xs' q
define k' where [simp]: k' = max 0 (k-free_A!n)
have [simp]: length xs = length init by auto

have dp_xs_init[simp]: dist_perm xs init by auto

The Transformation

have ub_cost: ∀x∈set_pmf D. (real (cost x)) + (Φ2 x) − (Φ0 x) ≤ k + 1 +
  (if (q) ∈ set init
   then (if (fst (snd x))!(index init q) then k−k'
     else (∑j<k'. (if (fst (snd x))!(index init
        (xs''j)) then 2::real 1))))
   else 0)
  + (∑i<(length (paid_A!n)). (if (fst (snd x))!(gebub n i) then 2
  else 1))
proof (rule, goal_cases)
  case (1 x)
  note xinD=1
  then have [simp]: snd (snd x) = init using D_def config_n_init3 by fast

  define b where b = fst (snd x)
  define ys where ys = fst x
  define aBIT where [simp]: aBIT = (if b ! (index (snd (snd x)) q)
  then 0 else length ys, ([:]::nat list))
  define ys' where ys' = step ys (q) aBIT
  define b' where b' = flip (index init q) b
  define Φ1 where Φ1 = (λx:: 'a list× (bool list × 'a list) . (∑(x,y)∈(Inv
    ys xs')). (if fst (snd z)!((index init y) then 2::real 1)))

  have xs''_step: xs'' = step xs (q) (free_A!n,paid_A!n)
  unfolding xs'_def xs''_def xs_def step_def free_A_def paid_A_def
  by(auto simp: split_def)

  have gis2: (Φ2 (ys,(b,init))) = (∑(x,y)∈(Inv ys' xs''). (if b!(index
    init y) then 2 else 1))
  apply(simp only: split_def)
  apply(simp only: xs''_step)
  apply(simp only: Φ2_def phi.simps)
  unfolding b'_def b_def ys'_def aBIT_def q_def
  unfolding s_A.simps apply(simp only: split_def) by auto
  then have gis: Φ2 x = (∑(x,y)∈(Inv ys' xs''). (if b!(index init y)
    then 2 else 1))
  unfolding ys_def b_def by (auto simp: split_def)
have his2: $(\Phi_0 (ys,(b,\text{init}))) = (\sum (x,y) \in (\text{Inv } ys \; xs). \text{ (if } b!(\text{index init } y) \text{ then 2 else } 1))$

apply (simp only: split_def)
apply (simp only: $\Phi_0$ def phi.simps) by (simp add: split_def)
then have his: $(\Phi_0 x) = (\sum (x,y) \in (\text{Inv } ys \; xs). \text{ (if } b!(\text{index init } y) \text{ then 2 else } 1))$

by (auto simp: ys_def b_def split_def phi')

have dis: $(\Phi_1 x) = (\sum (x,y) \in (\text{Inv } ys \; xs'). \text{ (if } b!(\text{index init } y) \text{ then 2 else } 1))$

unfolding $\Phi_1$ def b_def by auto

have ys': $\text{mtf2 (fst aBIT) (q) } ys$ by (simp add: step_def ys'_def)

from config_rand_distinct[of BIT] config_rand_set[of BIT] xinD
have dp_ys_init[simp]: dist_perm ys init unfolding D_def ys_def by force
have dp_ys'_init[simp]: dist_perm ys' init unfolding ys'_def step_def by (auto)
then have lenys'[simp]: $\text{length } ys' = \text{length } init$ by (metis distinct_card)
have dp_xs'_init[simp]: dist_perm xs' init by auto
have gra: dist_perm ys xs' by auto

have leninitb[simp]: $\text{length } b = \text{length } init$ using b_def config_n_fst_init_length2 xinD[unfolded] by auto
have leninitys[simp]: $\text{length } ys = \text{length } init$ using dp_ys_init by (metis distinct_card)

{ fix m have dist_perm ys (s' A n m) using dp_ys_init by auto
  } note dist=this

Upper bound of the inversions created by paid exchanges of A

let ?paidUB = $(\sum \text{i} < (\text{length } (\text{paid}_A!n)). \text{ (if } b!(\text{gebub } n \text{ i) then 2::real else } 1))$

have paid_ub: $(\Phi_1 x \leq \Phi_0 x + ?paidUB)$
proof -

have a: $\text{length } (\text{paid}_A ! n) \leq \text{length } (\text{paid}_A ! n)$ by auto
have b: $xs' = (s' A n \text{ (length } (\text{paid}_A ! n)))$ using s'A_m by auto


\[
\text{fix } m
\]
\[
\text{have } m \leq \text{length } (\text{paid}_A!n) \implies (\sum (x,y) \in (\text{Inv } ys \ (s'_A n \ m)). \ (\text{if } b! (\text{index init } y) \text{ then } 2::\text{real else } 1)) \leq (\sum (x,y) \in (\text{Inv } ys \ x \ m). \ (\text{if } b! (\text{index init } y) \text{ then } 2 \text{ else } 1))
\]
\[
+ (\sum i < m. \ (\text{if } b! (\text{gebub } n \ i) \text{ then } 2 \text{ else } 1))
\]
\[
\text{proof (induct } m)
\]
\[
\text{case } (\text{Suc } m)
\]
\[
\text{then have } m_{bd2} : m \leq \text{length } (\text{paid}_A \ n) \\
\text{and } m_{bd} : m < \text{length } (\text{paid}_A \ n) \text{ by auto}
\]
\[
\text{note yeah = Suc(1)[OF } m_{bd2}]
\]
\[
\text{let } ?\text{revm} = (\text{length } (\text{paid}_A \ n) - \text{Suc } m)
\]
\[
\text{note ah = Inv_swap[of } ys \ (s'_A n \ m) \ (\text{paid}_A \ n \ ?\text{revm}), \ \text{OF dist]}
\]
\[
\text{have } (\sum (x,a,y) \in \text{Inv } ys \ (s'_A n \ (\text{Suc } m)). \ (\text{if } b! (\text{index init } y) \text{ then } 2::\text{real else } 1))
\]
\[
= (\sum (x,a,y) \in \text{Inv } ys \ (\text{swap } (\text{paid}_A \ n \ ?\text{revm}) \ (s'_A n \ m)). \ (\text{if } b! (\text{index init } y) \text{ then } 2 \text{ else } 1)) \text{ using } s'_A.\text{.simps(2) by auto}
\]
\[
\text{also}
\]
\[
\text{have } \ldots = (\sum (x,a,y) \in (\text{if } \text{Suc } (\text{paid}_A \ n \ ?\text{revm}) < \text{length } ys \text{ then if } s'_A n \ m ! (\text{paid}_A \ n \ ?\text{revm}) < s'_A n \ m ! \text{Suc } (\text{paid}_A \ n \ ?\text{revm}) \ \text{in } ys \text{ then } \text{Inv } ys \ (s'_A n \ m) \cup \{(s'_A n \ m ! (\text{paid}_A \ n \ ?\text{revm}), s'_A n \ m ! \text{Suc } (\text{paid}_A \ n \ ?\text{revm})\})
\]
\[
\text{else } \text{Inv } ys \ (s'_A n \ m) - \{(s'_A n \ m ! \text{Suc } (\text{paid}_A \ n \ ?\text{revm}), s'_A n \ m ! (\text{paid}_A \ n \ ?\text{revm})\})
\]
\[
\text{else } \text{Inv } ys \ (s'_A n \ m)). \ (\text{if } b! (\text{index init } y) \text{ then } 2::\text{real else } 1) \text{ by (simp only: ah)}
\]
\[
\text{also}
\]
\[
\text{have } \ldots \leq (\sum (x,a,y) \in \text{Inv } ys \ (s'_A n \ m). \ (\text{if } b! (\text{index init } y) \text{ then } 2::\text{real else } 1))
\]
\[
+ (\text{if } (b) ! (\text{index init } (s'_A n \ m ! \text{Suc } (\text{paid}_A \ n \ ?\text{revm}))) \text{ then } 2::\text{real else } 1) \text{ (is } A \leq B)
\]
\[
\text{proof(cases Suc } (\text{paid}_A \ n \ ?\text{revm}) < \text{length } ys)
\]
\[
\text{case False}
\]
\[
\text{then have } ?A = (\sum (x,a,y) \in (\text{Inv } ys \ (s'_A n \ m))). \ (\text{if } b! (\text{index init } y) \text{ then } 2 \text{ else } 1) \text{ by auto}
\]
\[
\text{also have } \ldots \leq (\sum (x,a,y) \in (\text{Inv } ys \ (s'_A n \ m)). \ (\text{if } b! (\text{index init } y) \text{ then } 2 \text{ else } 1) +
\]
\[
(\text{if } b! (\text{index init } (s'_A n \ m ! \text{Suc } (\text{paid}_A \ n \ ?\text{revm}))))
\]
\[
\text{then } 2::\text{real else } 1) \text{ by auto}
\]
\[
\text{finally show } ?A \leq ?B.
\]
\[
\text{next}
\]
\[
\text{case True}
\]
\[
\text{then have } ?A = (\sum (x,a,y) \in (\text{if } s'_A n \ m ! (\text{paid}_A \ n \ !}
\]
\[
\text{\ldots )}
\]
\[
\text{103}
\]
\( \text{?revm} < s'_A n m \neq \text{Suc} (\text{paid}_A n \neq \text{?revm}) \text{ in ys} \)

then \( \text{Inv y} s' (s'_A n m) \cup \{ (s'_A n m \neq (\text{paid}_A n \neq \text{?revm}), s'_A n m \neq \text{Suc} (\text{paid}_A n \neq \text{?revm})) \}
\)

else \( \text{Inv y} s' (s'_A n m) = \{ (s'_A n m \neq \text{Suc} (\text{paid}_A n \neq \text{?revm}), s'_A n m \neq (\text{paid}_A n \neq \text{?revm})) \}
\)

\( \). if \( b \) \((\text{index init} y) \) \text{ then 2 else 1} \) \text{ by auto}

also have \( \ldots \leq \?B \) \((\text{is} \?A' \leq \?B)\)

\text{proof} \((\text{cases} s'_A n m \neq (\text{paid}_A n \neq \text{?revm}) < s'_A n m \neq \text{Suc} (\text{paid}_A n \neq \text{?revm}) \text{ in ys})
\)

case \( \text{True} \)

let \( ?\text{neurein}=s'_A n m \neq (\text{paid}_A n \neq \text{?revm}), s'_A n m \neq \text{Suc} (\text{paid}_A n \neq \text{?revm}) \)

\text{from} \( \text{True} \) \text{ have} \( \?A' = (\sum (x, y)\in (\text{Inv y} s' (s'_A n m) \cup \{ ?\text{neurein} \}) \)

\( \). if \( b \) \((\text{index init} y) \) \text{ then 2 else 1} \) \text{ by auto}

also have \( \ldots = (\sum (x, y)\in \text{insert} \ ?\text{neurein} \ (\text{Inv y} s' (s'_A n m)) \)

\( \). if \( b \) \((\text{index init} y) \) \text{ then 2 else 1} \) \text{ by auto}

also have \( \ldots \leq (if \ b \ (\text{index init} \ (\text{snd} \ ?\text{neurein})) \) \text{ then 2 else 1} \)

\( + (\sum (x, y)\in (\text{Inv y} s' (s'_A n m)) \). \text{ if} \ b \ (\text{index init y}) \text{ then 2 else 1} \) \text{ by auto}

\text{proof} \((\text{cases} \ ?\text{neurein} \in \text{Inv y} s' (s'_A n m))
\)

case \( \text{True} \)

then have \( \text{insert} \ ?\text{neurein} \ (\text{Inv y} s' (s'_A n m)) = (\text{Inv y} s' (s'_A n m)) \)
\text{by auto}

then have \( (\sum (x, y)\in \text{insert} \ ?\text{neurein} \ (\text{Inv y} s' (s'_A n m)). \text{if} \ b \ (\text{index init y}) \text{ then 2 else 1}) \)
\text{by auto}

also have \( \ldots \leq (\text{if} \ b \ (\text{index init} \ (\text{snd} \ ?\text{neurein})) \) \text{ then 2 else 1}) \)
\text{by auto}

finally show \( ?\text{thesis} \).

next

case \( \text{False} \)

have \( (\sum (x, y)\in \text{insert} \ ?\text{neurein} \ (\text{Inv y} s' (s'_A n m)). \text{if} \ b \ (\text{index init y}) \text{ then 2 else 1}) \)
\text{by auto simp: split_def}

also have \( \ldots = (\lambda i. \text{if} \ b \ (\text{index init} \ (\text{snd} i)) \) \text{ then 2 else 1} \) \text{ ?neurein}
\( + (\sum y\in (\text{Inv y} s' (s'_A n m)) \). \text{ if} \ b \)
apply (rule sum.insert_remove) by (auto)
also have ... = (if b ! (index init (snd ?neurein)) then 2 else 1)
+ (\( \sum_{y \in \text{Inv ys (s' A n m)}} (\lambda i. \text{if b ! (index init (snd i)) then 2::real else 1}) y \) 
using False by auto
also have ... \leq (if b ! (index init (snd ?neurein)) then 2 else 1)
+ (\( \sum_{(x a, y) \in \text{Inv ys (s' A n m)}} (\text{if b ! (index init y) then 2 else 1}) \) 
by (simp only: split_def)
finally show ?thesis.
qed
also have ...
= (\( \sum_{(x a, y) \in \text{Inv ys (s' A n m)}} (\text{if b ! (index init y) then 2 else 1}) \) 
by auto
finally show ?thesis.

next
\text{case False}
then have \( ?A' = (\sum_{(x a, y) \in \text{Inv ys (s' A n m)}} \{s' A n m ! \text{Suc (paid_A ! n ! ?revm)}\} \) 
). if b ! (index init y) then 2 else 1) by auto
also have ... \leq (\( \sum_{(x a, y) \in \text{Inv ys (s' A n m)}} \) if b ! (index init y) then 2 else 1) \is (\( \sum_{(x a, y) \in ?X - \{?x\}} (\text{?g y}) \) \leq (\( \sum_{(x a, y) \in ?X} (\text{?g y}) \) )
proof (cases ?x \in ?X)
\text{case True}
have \( (\sum_{(x a, y) \in ?X - \{?x\}} (\text{?g y}) \) \leq (\( \% (x a , y) . \ ?g y \) \ ?x + 
(\( \sum_{(x a, y) \in ?X - \{?x\}} (\text{?g y}) \) 
by simp
also have ... = (\( \sum_{(x a, y) \in ?X} (\text{?g y}) \) 
apply (rule sum.remove[symmetric])
apply simp apply (fact) done
finally show ?thesis.
qed simp
also have ... \leq ?B by auto
finally show ?thesis.
qed
finally show ?A \leq ?B.

also have ...
\leq (\( \sum_{(x a, y) \in \text{Inv ys (s A n)}} (\text{if b ! (index init y) then 2::real else 1}) \) + (\( \sum_{i<m. \text{if b ! gebub n i then 2::real else 1}) \)
+ (if (b ! (index init (s'_{A} n m ! Suc (paid_{A} ! n ! revm))) then 2::real else 1) using yeah by simp
also have ... = (\(\sum (x, y) \in \text{Inv} ys (s_{A} n)\). if b ! (index init y) then 2::real else 1) + (\(\sum i < m.\) if b ! gebub n i then 2 else 1)
+ (if (b ! gebub n m then 2 else 1) unfolding gebub_def
by simp
also have ... = (\(\sum (x, y) \in \text{Inv} ys (s_{A} n)\). if b ! (index init y) then 2::real else 1) + (\(\sum i < (Suc m)\). if b ! gebub n i then 2 else 1)
by auto
finally show ?case by simp
qed (simp add: split_def)
\}

note x = this [OF a]

show ?thesis
unfolding \(\Phi_{1}\).def his apply(simp only: b) using x b_def by auto
qed

Upper bound for the costs of BIT

\begin{align*}
\text{define } & \text{inI where } [\text{simp}]: \text{inI} = \text{InvOf (q) ys xs}' \\\\text{define } & \text{I where } [\text{simp}]: I = \text{card(InvOf (q) ys xs')}
\end{align*}

have ub_cost_BIT: (cost x) \leq k + 1 + I
proof (cases (q) \in set init)
\quad case False
\quad \quad from False have 4: I = 0 by (auto simp: before_in_def)
\quad have (cost x) = 1 + index ys (q) by (auto simp: ys_def t_def split_def)
\quad also have \ldots = 1 + length init using False by auto
\quad also have \ldots = 1 + k using False by auto
\quad finally show ?thesis using 4 by auto
\end{proof}
next
\quad case True
\quad \quad then have gra2: (q) \in set ys using dp_ys_init by auto
\quad have (cost x) = 1 + index ys (q) by (auto simp: ys_def t_def split_def)
\quad \quad also have \ldots \leq k + I + I using numberofIsbeschr[OF gra gra2]
by auto
\quad \quad finally show (cost x) \leq k + 1 + I .
\quad qed

Upper bound for inversions generated by free exchanges

\begin{align*}
\text{define } & \text{ub_free}
\quad \text{where } \text{ub_free} =
\quad \quad (if (q \in set init)
\quad \quad \quad then (if b!(index init q) then \(k-k'\) else (\(\sum j < k'\). (if (b)! (index init
\[(xs' \cdot j)) \text{ then } 2::\text{real else 1} \])
  \]
  \[
  \text{else 0)}
  \]
  \[
  \text{let } ?\text{ub2} = - I + \text{ub_free}
  \]
  \[
  \text{have free UB: } (\sum (x,y) \in (\text{Inv } y s' x s''). (\text{if } b' !(\text{index } \text{init } y) \text{ then } 2 \text{ else 1 })
  \]
  \[
  - (\sum (x,y) \in (\text{Inv } y s'). (\text{if } b!(\text{index } \text{init } y) \text{ then } 2 \text{ else 1 }) \leq
  \]
  \[
  ?\text{ub2}
  \]
  \[
  \text{proof} \ (\text{cases } (q) \in \text{set } \text{init})
  \]
  \[
  \text{case } \text{False}
  \]
  \[
  \text{from } \text{False have 1: } y s' = y s \text{ unfolding } y s' \text{ def step def mtf2 def}
  \]
  \[
  \text{by (simp)}
  \]
  \[
  \text{from } \text{False have 2: } x s' = x s'' \text{ unfolding } x s'' \text{ def mtf2 def by (simp)}
  \]
  \[
  \text{from } \text{False have } (\text{index } \text{init } q) \geq \text{ length } b \text{ using setinit by auto}
  \]
  \[
  \text{then have 3: } b' = b \text{ unfolding } b' \text{ def using flip out of bounds by auto}
  \]
  \[
  \text{from } \text{False have 4: } I = 0 \text{ unfolding } I \text{ def before in def by (auto)}
  \]
  \[
  \text{note } \text{ubnn}=\text{False}
  \]
  \[
  \text{have nn: } k - k' \geq 0 \text{ unfolding } k \text{ def } k' \text{ def by auto}
  \]
  \[
  \text{from 1 2 3 4 have } (\sum (x,y) \in (\text{Inv } y s' x s''). (\text{if } b' !(\text{index } \text{init } y) \text{ then 2::real else 1 }))
  \]
  \[
  - (\sum (x,y) \in (\text{Inv } y s'). (\text{if } b!(\text{index } \text{init } y) \text{ then } 2 \text{ else 1 })\) =
  \]
  \[
  - I \ \text{by auto}
  \]
  \[
  \text{with } \text{ubnn show } ?\text{thesis unfolding } \text{ub free def by auto}
  \]
  \[
  \text{next}
  \]
  \[
  \text{case } \text{True}
  \]
  \[
  \text{note } \text{queryinlist=theis}
  \]
  \[
  \text{then have } gra2: q \in \text{ set } y s \text{ using dp ys init by auto}
  \]
  \[
  \text{have k inbounds: } k < \text{ length } \text{init}
  \]
  \[
  \text{using index less size conv queryinlist}
  \]
  \[
  \text{by (simp)}
  \]
  \[
  \{ \text{fix } y \ e
  \]
  \[
  \text{fix } X::\text{bool list}
  \]
  \[
  \text{assume rd: } e < \text{ length } X
  \]
  \[
  \text{have } y < \text{ length } X \implies (\text{if flip } e X \ y \text{ then 2::real else 1}) - (\text{if } X \ y \text{ then 2 else 1})
  \]
  \[
  \text{107}
  \]
proof cases
  assume \( y < \text{length } X \) and \( e = y \)
  then have \( (\text{flip } e \, X ! y \text{ then } 2::\text{real else } 1) - (\text{if } X ! y \text{ then } 2 \text{ else } 1) \)
  \( = (\text{if } e = y \text{ then } (\text{if } X ! y \text{ then } -1 \text{ else } 1) \text{ else } 0) \) using \( \text{flip itself} \) by auto

next
  assume \( \text{len: } y < \text{length } X \) and \( e \neq y \)
  then have \( (\text{flip } e \, X ! y \text{ then } 2::\text{real else } 1) - (\text{if } X ! y \text{ then } 2 \text{ else } 1) \)
  \( = (\text{if } e = y \text{ then } (\text{if } X ! y \text{ then } -1 \text{ else } 1) \text{ else } 0) \) using \( \text{en} \) by auto

qed

\{ note flipstyle=this \}

from \text{queryinlist setinit} have \( \text{qsfst}: (\text{index init } q) < \text{length } b \) by simp

have \( fA: \text{finite } (\text{Inv } y \, s' \, x''') \) by auto
have \( fB: \text{finite } (\text{Inv } y \, x' \, s') \) by auto

\begin{align*}
\text{define } \Delta \text{ where } [\text{simp}]: \Delta &= (\sum (x,y) \in (\text{Inv } y \, s' \, x''') \cdot (\text{if } b \,!(\text{index init } y) \text{ then } 2::\text{real else } 1)) \\
&\quad - (\sum (x,y) \in (\text{Inv } y \, x') \cdot (\text{if } b \,!(\text{index init } y) \text{ then } 2 \text{ else } 1)) \\
\text{define } C \text{ where } [\text{simp}]: C &= (\sum (x,y) \in (\text{Inv } y \, s''') \cap (\text{Inv } y \, x') \cdot (\text{if } b \,!(\text{index init } y) \text{ then } 2::\text{real else } 1)) \\
&\quad - (\text{if } b \,!(\text{index init } y) \text{ then } 2 \text{ else } 1)) \\
\text{define } A \text{ where } [\text{simp}]: A &= (\sum (x,y) \in (\text{Inv } y \, s''') \setminus (\text{Inv } y \, x')) \cdot (\text{if } b \,!(\text{index init } y) \text{ then } 2::\text{real else } 1)) \\
\text{define } B \text{ where } [\text{simp}]: B &= (\sum (x,y) \in (\text{Inv } y \, x') \setminus (\text{Inv } y \, s''')) \cdot (\text{if } b \,!(\text{index init } y) \text{ then } 2::\text{real else } 1))
\end{align*}
have *teilen*: \( \Delta = C + A - B \)

unfolding \( \Delta \_\text{def} A \_\text{def} B \_\text{def} C \_\text{def} \)
using `sum.my[OF fA fB]` by (auto simp: `split_def`)
then have \( \Delta = A - B + C \) by `auto`
then have *teilen2*: \( \Phi_2 x - \Phi_1 x = A - B + C \)
unfolding \( \Delta \_\text{def} \)
using `dis_gis by auto`

have *setys1*: `(index init) \^` `(set ys\') = \{0..<length ys\}'
proof
  have `(index init) \^` `(set ys\') = `(index init) \^` `(set init)` by `auto`
  also have `...` = `{0..<length init}` using `setinit by auto`
  also have `...` = `{0..<length ys\}' using `lenys by auto`
  finally show `?thesis` .
qed

have *BC_absch*: \( C - B \leq -I \)

proof (cases `b!(index init q)`)
case `True`
then have *samesame*: `ys\' = ys`
unfolding `ys\'_def step_def` by `auto`
then have *puh*: `(Inv ys\' xs\') = (Inv ys xs\')` by `auto`

```
\{ 
  fix \alpha \beta 
  assume `(alpha, beta) \in (Inv ys\' xs\'' \cap (Inv ys\' xs\'))`
  then have `(alpha, beta) \in (Inv ys\' xs\'' \cap (Inv ys\' xs\'))` by `auto`
  then have `(alpha < beta in ys\')` unfolding `Inv_def by auto`
  then have `1 : beta \in set ys\'` by (simp only: before_in_setD2)
  then have `index init beta < length ys\'` using `setys\' by auto`
  then have `index init beta < length init` using `lenys\' by auto`
  then have *puzzel*: `index init beta < length b` using `leninitb by auto`
```

have *betainit*: `beta \in set init using 1 by auto`
have *aha*: `(q=beta) = (index init q = index init beta)`
  using `betainit by simp`

have `(if b!(index init beta) then 2::real else 1) - (if b!(index init beta) then 2 else 1)`
  = `(if (index init q) = (index init beta) then if b !(index init beta) then -1 else 1 else 0)`
  unfolding `b\'_def apply(rule flipstyle) by (fact)+
  also have `...` = `(if (index init q) = (index init beta) then if b !)`
\[(\text{index init } q) \text{ then } -1 \text{ else } 1 \text{ else } 0) \text{ by auto} \]

also have \( \ldots = (\text{if } q = \beta \text{ then } -1 \text{ else } 0) \) using aha True by auto

finally have \((\text{if } b!(\text{index init } \beta) \text{ then } 2::\text{real else } 1) - (\text{if } b!(\text{index init } \beta) \text{ then } 2 \text{ else } 1)\)  
\[= (\text{if } (q) = \beta \text{ then } -1::\text{real else } 0) \text{ by auto} \]

} then have \(\text{grreeeeaa: } \forall x \in (\text{Inv } yst' xst'') \cap (\text{Inv } yst' xst')\).
\[(\lambda x. \text{if } b! (\text{index init } (\text{snd } x)) \text{ then } 2::\text{real else } 1) - (\text{if } b! (\text{index init } (\text{snd } x)) \text{ then } 2 \text{ else } 1)) x \]
\[= (\lambda x. \text{if } (q) = \text{snd } x \text{ then } -1::\text{real else } 0)) x \text{ by force} \]

let \(\text{ttt} = (\text{Inv } yst' xst'') \cap (\text{Inv } yst' xst')\)

have \(\text{ttt}: \{(x,y). (x,y)\in (\text{Inv } yst' xst'') \cap (\text{Inv } yst' xst')\}
\[\wedge y = (q) \} \cup \{(x,y). (x,y)\in (\text{Inv } yst' xst'') \cap (\text{Inv } yst' xst')\]
\[\wedge y \neq (q) \} = (\text{Inv } yst' xst'') \cap (\text{Inv } yst' xst') \text{ is } \text{?split1} \]
\[\cup \text{?split2} = \text{?easy} \text{ by auto} \]

have \(\text{interem: } \text{?split1} \cap \text{?split2} = \{\} \text{ by auto} \]

have \(\text{split1subs: } \text{?split1} \subseteq \text{?fin} \text{ by auto} \]

have \(\text{split2subs: } \text{?split2} \subseteq \text{?fin} \text{ by auto} \]

have \(\text{fs1: finite } \text{?split1} \text{ apply}(\text{rule finite_subset}[\text{where } B=\text{fin}]) \)
\[\text{apply}(\text{rule split1subs}) \text{ by(auto)} \]

have \(\text{fs2: finite } \text{?split2} \text{ apply}(\text{rule finite_subset}[\text{where } B=\text{fin}]) \)
\[\text{apply}(\text{rule split2subs}) \text{ by(auto)} \]

have \(k - k' \leq (\text{free } A!n) \text{ by auto} \]

have \(\text{g: InvOf } (q) \text{ yst' xst'' } \supseteq \text{InvOf } (q) \text{ yst' xst'}\)
using True apply(auto) apply(rule mtf2_mono[of swaps (paid_A ! n) \(s_A n))] \)
by (auto simp: queryinlist)

have \(h: \text{?split1} = (\text{InvOf } (q) \text{ yst' xst''}) \cap (\text{InvOf } (q) \text{ yst' xst'})\)
unfolding Inv_def by auto
also from \(g\) have \(\ldots = \text{InvOf } (q) \text{ yst' xst'} \text{ by force} \]
also from \(\text{samesame}\) have \(\ldots = \text{InvOf } (q) \text{ yst' xst'} \text{ by simp} \]
finally have \(\text{?split1} = \text{inI} \text{ unfolding inI_def} \).
then have \(\text{cardsp1isI: card } \text{?split1} = 1 \text{ by auto} \)

\{ 
fix \(a b\)
assume \((a, b)\in \text{?split1}\)
then have \(b = (q) \text{ by auto} \)
then have \((\text{if } (q) = b \text{ then } -1::\text{real} \text{ else } 0) = (-1::\text{real}) \text{ by} \)
\begin{verbatim}
auto

} then have split1easy: \(\forall x \in \text{split1}. (\lambda x. (if (q) = \text{snd} x \text{ then } (-1::real) \text{ else } 0)) x = (\lambda x. (-1::real)) \) x by force

{ 
  fix a b
  assume \((a,b) \in \text{split2}\)
  then have \(\sim b = (q)\) by auto
  then have \((if (q) = b \text{ then } (-1::real) \text{ else } 0) = 0\) by auto
}

then have split2easy: \(\forall x \in \text{split2}. (\lambda x. (if (q) = \text{snd} x \text{ then } (-1::real) \text{ else } 0)) x = (\lambda x. 0::real)\) x by force

have E0: \( C = \\
(\sum (x,y) \in (\text{Inv ys'} xs') \cap (\text{Inv ys} xs'). \\
(if b!(\text{index init} y) \text{ then } 2::real \text{ else } 1) \) ) by auto
also from puh have E1: \( ... = \\
(\sum (x,y) \in (\text{Inv ys'} xs') \cap (\text{Inv ys} xs'). \\
(if b!(\text{index init} y) \text{ then } 2::real \text{ else } 1) \) ) by (auto simp: split_def)
also have E2: \( ... = (\sum (x,y) \in \text{split1} \cup \text{split2}. \\
(if (q) = y \text{ then } (-1::real) \text{ else } 0) \) ) by (simp only: ttt)
also have \(... = (\sum (x,y) \in \text{split1}. (if (q) = y \text{ then } (-1::real) \text{ else } 0)) \) 
+ (\(\sum (x,y) \in \text{split2}. (if (q) = y \text{ then } (-1::real) \text{ else } 0)\)) 
- (\(\sum (x,y) \in \text{split1} \cap \text{split2}. (if (q) = y \text{ then } (-1::real) \text{ else } 0)\))
else 0)))
by (rule sum_Un[OF fs1 fs2])
also have \(... = (\sum (x,y) \in \text{split1}. (if (q) = y \text{ then } (-1::real) \text{ else } 0)) \)
apply (simp only: interem) by auto
also have E4: \( ... = (\sum (x,y) \in \text{split1}. (-1::real) \) 
+ (\(\sum (x,y) \in \text{split2}. 0\))
using sum_my2[OF split1easy]sum_my2[OF split2easy] by (simp only: split_def)
also have \( ... = (\sum (x,y) \in \text{split1}. (-1::real) \) ) by auto
\end{verbatim}
also have $E_5$: \ldots $ = - \operatorname{card} \ ? \operatorname{split1}$ by auto
also have $E_6$: \ldots $ = - I$ using cardsp1is1 by auto
finally have $\operatorname{abschC}$: $C = -I$.

have $\operatorname{abschB}$: $B \geq (0::\text{real})$ unfolding $B_{\text{def}}$ apply(rule sum_nonneg)
by auto

from $\operatorname{abschB}$ $\operatorname{abschC}$ show $C - B \leq -I$ by simp

next
case False
from leninitys False have $\operatorname{ya}$: $\operatorname{ys}' = \operatorname{mtf2}$ (length $\operatorname{ys}$) $q$ $\operatorname{ys}$
unfolding $\operatorname{step}_{\text{def}}$ $\operatorname{ys}'_{\text{def}}$ by(auto)
have $\operatorname{index} \operatorname{ys}' q = 0$
unfolding $\operatorname{ya}$ apply(rule mtf2_moves_to_front)
using $\operatorname{gra2}$ by simp_all
then have $\operatorname{nixbefore}$: $\operatorname{before} q$ $\operatorname{ys}' = \{\}$ unfolding before_in_def by auto

{ fix $\alpha \beta$
assume $(\alpha, \beta) \in (\operatorname{Inv} \operatorname{ys}' \operatorname{xs}'') \cap (\operatorname{Inv} \operatorname{ys} \operatorname{xs}')$
then have $(\alpha, \beta) \in (\operatorname{Inv} \operatorname{ys}' \operatorname{xs}'')$ by auto
then have $(\alpha < \beta \text{ in } \operatorname{ys}')$ unfolding $\operatorname{Inv}_{\text{def}}$ by auto
then have $1$: $\beta \in \operatorname{set} \operatorname{ys}'$ by (simp only: before_in_setD2)
then have $(\operatorname{index} \operatorname{init}$ $\beta) < \operatorname{length} \operatorname{ys}'$ using setys' by auto
then have $(\operatorname{index} \operatorname{init} \beta) < \operatorname{length} \operatorname{init}$ using lenys' by auto
then have puzzel: $(\operatorname{index} \operatorname{init} \beta) < \operatorname{length} b$ using leninitb by auto

have betainit: $\beta \in \operatorname{set} \operatorname{init}$ using $1$ by auto
have aha: $(q = \beta) = (\operatorname{index} \operatorname{init} q = \operatorname{index} \operatorname{init} \beta)$
using betainit by simp

have (if $b'!(\operatorname{index} \operatorname{init} \beta)$ then $2::\text{real}$ else $1$) $-$ (if $b!(\operatorname{index} \operatorname{init} \beta)$ then $2$ else $1$)

= (if (index init $q$) = (index init $\beta$) then $b$ ! (index init $\beta$)
then $-1$ else $1$ else $0$)
unfolding $b'_{\text{def}}$ apply(rule flipstyle) by(fact)+
also have \ldots $ = (if (index init $q$) = (index init $\beta$) then $b$ !
(index init $q$) then $-1$ else $1$ else $0$) by auto
also have \ldots \ldots $ = (if $(q = \beta$ then $1$ else $0$) using False aha by auto

112
finally have (if b'(index init β) then 2::real else 1) − (if b!(index init β) then 2 else 1)
   = (if (q) = β then 1::real else 0) by auto

then have grreeea2: ∀ x∈(Inv ys' xs'' ∩ (Inv ys xs')).
   (λx. (if b!(index (snd x)) then 2::real else 1)) x
   = (λx. (if (q) = snd x then 1::real else 0)) x by force

let ?fin=(Inv ys' xs'') ∩ (Inv ys xs')

have ttt: {(x,y). (x,y)∈(Inv ys' xs'') ∩ (Inv ys xs')}
   ∧ y = (q) ∪ {(x,y). (x,y)∈(Inv ys' xs'') ∩ (Inv ys xs')}
   ∧ y ≠ (q) = (Inv ys' xs'') ∩ (Inv ys xs') (is ?split1
   ∪ ?split2 = ?easy) by auto

have interem: ?split1 ∩ ?split2 = {} by auto
have split1subs: ?split1 ⊆ ?fin by auto
have split2subs: ?split2 ⊆ ?fin by auto
have fs1: finite ?split1 apply(rule finite_subset[where B=?fin])
   apply(rule split1subs) by(auto)
have fs2: finite ?split2 apply(rule finite_subset[where B=?fin])
   apply(rule split2subs) by(auto)

have split1easy : ∀ x∈?split1.
   (λx. (if (q) = snd x then (1::real) else 0)) x = (λx. (1::real))
   x by force

have split2easy : ∀ x∈?split2.
   (λx. (if (q) = snd x then (1::real) else 0)) x = (λx. (0::real))
   x by force

from nixbefore have InvOfempty: InvOf q ys' xs'' = {} unfolding
   Inv_def by auto

have ?split1 = InvOf q ys' xs'' ∩ InvOf q ys xs'
   unfolding Inv_def by auto
also from InvOfempty have ... = {} by auto
finally have split1empty: ?split1 = {} .

have C = (∑ (x,y)∈?easy.
   (if (q) = y then (1::real) else 0)) unfolding C_def
   by(simp only: split_def sum_my2[OF grreeea2])
also have \( \ldots = (\sum (x,y) \in ?split1 \cup ?split2. \) \\
\((\text{if } (q) = y \text{ then } (1::\text{real}) \text{ else } 0)) \) by (simp only: ttt)
also have \( \ldots = (\sum (x,y) \in ?split1. \) (if \((q) = y \text{ then } (1::\text{real}) \text{ else } 0)) \\
+ (\sum (x,y) \in ?split2. \) (if \((q) = y \text{ then } (1::\text{real}) \text{ else } 0)) \\
- (\sum (x,y) \in ?split1 \cap ?split2. \) (if \((q) = y \text{ then } (1::\text{real}) \text{ else } 0))
else 0)) \)
by (rule sum_Un[OF fs1 fs2])
also have \( \ldots = (\sum (x,y) \in ?split1. \) (if \((q) = y \text{ then } (1::\text{real}) \text{ else } 0)) \\
+ (\sum (x,y) \in ?split2. \) (if \((q) = y \text{ then } (1::\text{real}) \text{ else } 0)) \\
apply (simp only: interem) by auto
also have \( \ldots = (\sum (x,y) \in ?split1. \) (1::\text{real}) \\
+ (\sum (x,y) \in ?split2. \) 0 \) using sum_my2[OF split1easy] \\
sum_my2[OF split2easy] by (simp only: split_def)
also have \( \ldots = (\sum (x,y) \in ?split1. \) (1::\text{real}) \) by auto
also have \( \ldots = \text{card } ?split1 \) by auto
also have \( \ldots = (0::\text{real}) \) apply (simp only: split1empty) by auto
finally have \( \text{absch}\text{C: } C = (0::\text{real}) \).

have \( \text{ttt2: } \{(x,y). \) (x,y) \in (\text{Inv } ys \ ) x\} - (\text{Inv } ys' \ ) x' \) \\
\( \wedge y = (q)\} \cup \{(x,y). \) (x,y) \in (\text{Inv } ys \ ) x\} - (\text{Inv } ys' \ ) x' \) \\
\( \wedge y \neq (q)\} = (\text{Inv } ys \ ) x\} - (\text{Inv } ys' \ ) x' \) \((\text{is } ?split1 \) \\
\( \cup \ ?split2 = ?easy2) \) by auto
have \( \text{interem: } ?split1 \cap ?split2 = \{\} \) by auto
have \( \text{split1subs: } ?split1 \subseteq ?easy2 \) by auto
have \( \text{split2subs: } ?split2 \subseteq ?easy2 \) by auto
have \( \text{fs1: finite } ?split1 \) apply (rule finite_subset [where \( B = \) ?easy2]) \\
apply (rule split1subs) by (auto)
have \( \text{fs2: finite } ?split2 \) apply (rule finite_subset [where \( B = \) ?easy2]) \\
apply (rule split2subs) by (auto)
from False have \( \text{split1easy2: } \forall x \in ?split1. \) \\
\( (\lambda x. \) (if \( b! \) (index init (snd x)) then 2::real else 1) \) \( x = (\lambda x. \) \\
(1::real)) \) \( x \) by force

have \( ?split1 = (\text{InvOf } q \ ) y\ ) x\} - (\text{InvOf } q \ ) y\ } x' \) \\
unfolding \text{Inv_def} by auto
also have \( \ldots = \text{inI} \) unfolding \text{InvOfempty} by auto
finally have \( \text{spI: } ?split1 = \text{inI} \).

have \( \text{abschaway: } \sum (x,y) \in ?split2. \) (if \( b!(\text{index init } y) \) then 2::real}
\[\text{else 1}) \geq 0\]

apply (rule sum_nonneg) by auto

have \(B = \left( \sum (x, y) \in ?\text{split1} \cup ?\text{split2}. \right)\) unfolding B_def
by (simp only: ttt2)
also have \(\ldots = \left( \sum (x, y) \in ?\text{split1}. \right.\) (if b!(index init y) then 2::real else 1)
\[\left. + \left( \sum (x, y) \in ?\text{split2}. \right.\] (if b!(index init y) then 2::real else 1))
by (rule sum_Un[OF fs1 fs2])
also have \(\ldots = \left( \sum (x, y) \in ?\text{split1}. \right.\) (if b!(index init y) then 2::real else 1)
\[\left. \left. - \left( \sum (x, y) \in ?\text{split1} \cap ?\text{split2}. \right.\] (if b!(index init y) then 2::real else 1))
by (simp only: interem) by auto

also have \(\ldots = I\)
+ \(\left( \sum (x, y) \in ?\text{split2}. \right.\) (if b!(index init y) then 2::real else 1))
by auto
also have \(\ldots \geq I\) using abschaway by auto
finally have abschB: \(B \geq I\).

from abschB abschC show \(C - B \leq -I\) by auto
qed

have \(A_{absch}: A\)
\[\leq \left( \text{if } b!(\text{index init } q) \text{ then } k-k' \text{ else } \left( \sum j < k'. \text{ if } b!(\text{index init } (x \# j)) \text{ then 2::real else 1}) \right) \right)\]
proof (cases b!(index init q))
case False

from leninities False have \(y>: y' = \text{mtf2 length } y\) \(q\ y\)
unfolding step_def ys'_def by (auto)

have index y's' q = 0 unfolding ya apply (rule mtf2_moves_to_front)

115
using \texttt{gra2 by(simp\_all)}
then have \texttt{nixbefore: before q ys' = \{} unfolding before\_in\_def by auto

have \texttt{A = (\sum (x,y) \in (Inv ys' xs'') - (Inv ys xs'). (if b!(index init y) then 2::real else 1)) by auto}

have \texttt{index (mtf2 (free_A ! n) (q) (swaps (paid_A ! n) (s_A n))) (q)}
\quad \texttt{= (index (swaps (paid_A ! n) (s_A n)) (q) - free_A ! n)}
avpl\texttt{(rule mtf2_q\_after) using queryinlist by auto}
then have \texttt{whatisk': k' = index xs'' q by auto}

have \texttt{ss: set ys' = set ys by auto}
have \texttt{ss2: set xs' = set xs'' by auto}

have \texttt{di: distinct init by auto}
have \texttt{dys: distinct ys by auto}

have \texttt{(Inv ys' xs'') - (Inv ys xs')} 
\quad \texttt{= \{(x,y). x < y in ys' \land y < x in xs'' \land (\sim x < y in ys \lor \sim y < x in xs')\}}
\quad \texttt{unfolding Inv\_def by auto}
\quad \texttt{also have \ldots = }
\quad \texttt{\{(x,y). y\neq q \land x < y in ys' \land y < x in xs'' \land (\sim x < y in ys \lor \sim y < x in xs')\}}
\quad \texttt{using nixbefore by blast}
\quad \texttt{also have \ldots = }
\quad \texttt{\{(x,y). x\neq y \land y\neq q \land x < y in ys' \land y < x in xs'' \land (\sim x < y in ys \lor \sim y < x in xs')\}}
\quad \texttt{unfolding before\_in\_def by auto}
\quad \texttt{also have \ldots = }
\quad \texttt{\{(x,y). x\neq y \land y\neq q \land x < y in ys' \land y < x in xs'' \land (\sim x < y in ys \lor \sim y < x in xs')\}}
\quad \texttt{by force}
\quad \texttt{also have \ldots = }
\quad \texttt{\{(x,y). x\neq y \land y\neq q \land x < y in ys' \land y < x in xs'' \land y < x in ys\}}
\quad \texttt{\cup \{(x,y). x\neq y \land y\neq q \land x < y in ys' \land y < x in xs'' \land (\sim x < y in xs')\}}
\quad \texttt{by force}
\quad \texttt{also have \ldots = }
\quad \texttt{\{(x,y). x\neq y \land y\neq q \land x < y in ys' \land y < x in xs'' \land y < x in ys\}}
\quad \texttt{\cup \{(x,y). x\neq y \land y\neq q \land x < y in ys' \land y < x in xs'' \land (\sim x < y in xs')\}}
\quad \texttt{by force}
\quad \texttt{by force}
\quad \texttt{also have \ldots = }
\quad \texttt{\{(x,y). x\neq y \land y\neq q \land x < y in ys' \land y < x in xs'' \land (\sim x < y in xs')\}}
\quad \texttt{\cup \{(x,y). x\neq y \land y\neq q \land x < y in ys' \land y < x in xs'' \land (\sim x < y in xs')\}}
\quad \texttt{by force}
\quad \texttt{by force}
\quad \texttt{by force}
\quad \texttt{using before\_in\_setD1[where xs=ys'] before\_in\_setD2[where xs=ys'] not\_before\_in ss by metis}
also have \( \cdots = \)
\[
\{(x, y). x \neq y \land y \neq q \land x < y \text{ in } ys' \land y < x \text{ in } xs'' \land y < x \text{ in } ys \} \\
\cup \{(x, y). x \neq y \land y \neq q \land x < y \text{ in } ys' \land y < x \text{ in } xs'' \land x < y \text{ in } xs' \}
\]
\((xs') \text{ (is } S_1 \cup S_2 = S_1 \cup S_2') \)

proof
- have \(S_2 = S_2' \text{ apply(safe)}\)
  proof (goal_cases)
  case (2\(a \ b\))
  from 2(5) have ~ \(b < a \text{ in } xs'\) by auto
  with 2(6) show False by auto
next
case (1\(a \ b\))
from 1(4) have \(a \in \text{set } xs' \ b \in \text{set } xs'\)
  using before_in_setD1[where \(xs=xs'\)]
  before_in_setD2[where \(xs=xs'\)] ss2 by auto
  with not_before_in 1(5) have \((a < b \text{ in } xs' \lor a = b)\) by metis

  with 1(1) show \(a < b \text{ in } xs'\) by auto
  qed
then show ?thesis by auto
qed
also have \(\cdots = \)
\[
\{(x, y). x \neq y \land y \neq q \land x < y \text{ in } ys' \land y < x \text{ in } xs'' \land y < x \text{ in } ys \} \\
\cup \{(x, y). x \neq y \land y \neq q \land x < y \text{ in } ys' \land y < x \text{ in } xs'' \land x < y \text{ in } xs' \}
\]
\((in xs') \text{ (is } S_1 \cup S_2 = S_1 \cup S_2') \)

proof
- have \(S_2 = S_2' \text{ apply(safe)}\)
  proof (goal_cases)
  case (1\(a \ b\))
  from 1(4) have ~ \(a < b \text{ in } xs''\) by auto
  with 1(6) show False by auto
next
case (2\(a \ b\))
from 2(5) have \(a \in \text{set } xs'' \ b \in \text{set } xs''\)
  using before_in_setD1[where \(xs=xs'\)]
  before_in_setD2[where \(xs=xs'\)] ss2 by auto
  with not_before_in 2(4) have \((b < a \text{ in } xs'' \lor a = b)\) by metis

  with 2(1) show \(b < a \text{ in } xs''\) by auto
  qed
then show ?thesis by auto
qed
also have \(\cdots = \)
\[
\{(x, y). x \neq y \land y \neq q \land x < y \text{ in } ys' \land y < x \text{ in } xs'' \land y < x \text{ in } xs \}
\]
\[ \text{ys} \]
\[ \bigcup \{ \text{using } x_{\text{stays before } y \text{ if } y \text{ not moved to front} | \text{where } x s=x s' \} \]
and \( q=q \)
\begin{align*}
\text{before in setD1} & \text{[where } x s=x s' \} \text{ before in setD2} \text{[where } x s=x s' \} \text{ by (auto simp: queryinlist)} \\
\text{also have } \ldots & = \\
\{ (x,y), x\neq y \land x=q \land y\neq q \land x < y \text{ in } y s' \land y < x \text{ in } x s'' \land y < x \text{ in } y s \} \text{ by force} \\
\text{also have } \ldots & = \\
\{ (x,y), x=q \land y\neq q \land q < y \text{ in } y s' \land y < q \text{ in } x s'' \land y \in \text{ set } x s'' \} \\
\text{using before in setD1 by metis} \\
\text{also have } \ldots & = \\
\{ (x,y), x=q \land y\neq q \land q < y \text{ in } y s' \land \text{ index } x s'' y < \text{ index } x s'' q \land q \in \text{ set } x s'' \land y \in \text{ set } x s'' \} \text{ unfolding before in def by auto} \\
\text{also have } \ldots & = \\
\{ (x,y), x=q \land y\neq q \land \text{ index } x s' y < \text{ index } x s' q - (\text{ free } A \! n) \land q \in \text{ set } x s'' \land y \in \text{ set } x s'' \} \\
\text{using mtf2 \_q \_after [where } A=x s' \text{ and } q=q \} \text{ by force} \\
\text{also have } \ldots & = \\
\{ (x,y), x=q \land y\neq q \land \text{ index } x s' y < \text{ index } x s' q - (\text{ free } A \! n) \land y \in \text{ set } x s'' \} \\
\text{using mtf2 \_backwards \_effect4 [where } x s=x s' \text{ and } q=q \text{ and } n=(\text{ free } A \! n), \text{ simplified } \} \\
\text{by auto} \\
\text{also have } \ldots & = \\
\{ (x,y), x=q \land y\neq q \land \text{ index } x s' y < k' \} \\
\text{using mtf2 \_q \_after [where } A=x s' \text{ and } q=q \} \text{ by auto} \\
\text{finally have subsa: (Inv } y s' x s'' \} - (\text{ Inv } y s x s') \\
\subseteq \{ (x,y), x=q \land y\neq q \land \text{ index } x s' y < k' \}. \\
\text{have } k' x s' ; k' < \text{ length } x s'' \text{ unfolding whatisk'} \\
\text{apply (rule index \_less) by (auto simp: queryinlist)} \\
\text{then have } k' x s' ; k' < \text{ length } x s' \text{ by auto} \\
\text{have } \{ (x,y), x=q \land \text{ index } x s' y < k' \} \\
\subseteq \{ (x,y), x=q \land \text{ index } x s' y < \text{ length } x s' \} \text{ using } k' x s' \text{ by auto} \]
also have \( \ldots = \{(x,y). x=q \land y \in \text{set } xs'\} \)

using \text{index_less_size_convs} by \text{fast}

finally have \( \{(x,y). x=q \land \text{index } xs' \land y < k'\} \subseteq \{(x,y). x=q \land y \in \text{set } xs'\} \).

then have \text{finia2: } \text{finite } \{(x,y). x=q \land \text{index } xs' \land y < k'\}

apply\(\text{rule finite_subset} \) by\(\text{simp}\)

have \text{lulae: } \{(a,b). a=q \land \text{index } xs' \land b < k'\}

= \{(q,b)|b. \text{index } xs' \land b < k'\} by \text{auto}

have \(k'b; k' < \text{length } b\) using \text{whatisk'} by (\text{auto simp: queryinlist})

have \text{asdasd: } \{(\alpha,\beta). \alpha=q \land \beta \neq q \land \text{index } xs' \land \beta < k'\}

= \{(\alpha,\beta). \alpha=q \land \beta \neq q \land \text{index } xs' \land \beta < k' \land (\text{index init } \beta) < \text{length } b\}

proof (\text{auto, goal\_cases})

case \(1\ b)\)

from \(1(2)\) have \(\text{index } xs' \land b < \text{index } xs' \land (q)\) by \text{auto}

also have \(\ldots < \text{length } xs'\) by (\text{auto simp: queryinlist})

finally have \(b \in \text{set } xs'\) using \text{index_less_size_convs} by

\text{metis}

then show \(\text{?case using setinit by auto}\)

\text{qed}

{\text{fix } \beta}

have \(\beta \neq q \implies (\text{index init } \beta) \neq (\text{index init } q)\)

using \text{queryinlist} by \text{auto}

} note \text{ij=this}

have \text{subsa2: } \{(\alpha,\beta). \alpha=q \land \beta \neq q \land \text{index } xs' \land \beta < k'\} \subseteq

\{(\alpha,\beta). \alpha=q \land \text{index } xs' \land \beta < k'\} by \text{auto}

then have \text{finia: } \text{finite } \{(x,y). x=q \land y \neq q \land \text{index } xs' \land y < k'\}

apply\(\text{rule finite_subset} \) using \text{finia2} by \text{auto}

have \text{E0: } A = (\sum (x,y) \in (\text{Inv } ys' \times xs')). (\text{if b'(index init y) then } 2::\text{real else } 1)) by \text{auto}

also have \text{E1: } \ldots \leq (\sum (x,y) \in \{(a,b). a=q \land b \neq q \land \text{index } xs' \land b < k'\}.) (\text{if b'(index init y) then } 2::\text{real else } 1))

unfolding \text{A_def apply\(\text{rule sum_mono}2[\text{OF finia subsa}] \)} by \text{auto}

also have \(\ldots = (\sum (x,y) \in \{(\alpha,\beta). \alpha=q \land \beta \neq q \land \text{index } xs' \land \beta < k' \land (\text{index init } \beta) < \text{length } b\}. (\text{if b'(index init y) then } 2::\text{real else } 1))\)

using \text{asdasd by auto}

also have \(\ldots = (\sum (x,y) \in \{(\alpha,\beta). \alpha=q \land \beta \neq q \land \text{index } xs' \land \beta < k' \land (\text{index init } \beta) < \text{length } b\}. (\text{if b'(index init y) then } 2::\text{real else } 1))\)

using \text{asdasd by auto}

also have \(\ldots = (\sum (x,y) \in \{(\alpha,\beta). \alpha=q \land \beta \neq q \land \text{index } xs' \land \beta < k' \land (\text{index init } \beta) < \text{length } b\}. (\text{if b'(index init y) then } 2::\text{real else } 1))\)

using \text{asdasd by auto}

also have \(\ldots = (\sum (x,y) \in \{(\alpha,\beta). \alpha=q \land \beta \neq q \land \text{index } xs' \land \beta < k' \land (\text{index init } \beta) < \text{length } b\}. (\text{if b'(index init y) then } 2::\text{real else } 1))\)

using \text{asdasd by auto}

also have \(\ldots = (\sum (x,y) \in \{(\alpha,\beta). \alpha=q \land \beta \neq q \land \text{index } xs' \land \beta < k' \land (\text{index init } \beta) < \text{length } b\}. (\text{if b'(index init y) then } 2::\text{real else } 1))\)

using \text{asdasd by auto}

also have \(\ldots = (\sum (x,y) \in \{(\alpha,\beta). \alpha=q \land \beta \neq q \land \text{index } xs' \land \beta < k' \land (\text{index init } \beta) < \text{length } b\}. (\text{if b'(index init y) then } 2::\text{real else } 1))\)

using \text{asdasd by auto}

also have \(\ldots = (\sum (x,y) \in \{(\alpha,\beta). \alpha=q \land \beta \neq q \land \text{index } xs' \land \beta < k' \land (\text{index init } \beta) < \text{length } b\}. (\text{if b'(index init y) then } 2::\text{real else } 1))\)
\[ 2::\text{real else 1} \]
\[
\text{proof (rule sum.cong, goal_cases)}
\]
\[
\text{case (2 z)}
\]
\[
\text{then obtain } \alpha \beta \text{ where } z = (\alpha, \beta) \text{ and } \alpha = q \text{ and } \text{diff: } \beta \\
\text{\# q and } \text{index} \xs' \beta < k' \text{ and } i: \text{index init } \beta < \text{length } b \text{ by auto}
\]
\[
\text{from diff ij have index init } \beta \neq \text{index init } q \text{ by auto}
\]
\[
\text{with flip_other qsfst i have } b'! \text{ index init } \beta = b! \text{ index init } \beta
\]
\[
\text{unfolding } b'_\text{def by auto}
\]
\[
\text{with zab show } \text{case by (auto simp add: split_def)}
\]
\[
\text{qed simp}
\]
\[
\text{also have } E1a: \ldots = (\sum (x,y) \in \{(a,b). a=q \land b\neq q \land \text{index} \xs' b < k'\}. (if } b!(\text{index init } y) \text{ then } \text{2::real else 1})
\]
\[
\text{using asdasd by auto}
\]
\[
\text{also have } \ldots \leq (\sum (x,y) \in \{(a,b). a=q \land \text{index} \xs' b < k'\}. (if } b!(\text{index init } y) \text{ then } \text{2::real else 1})
\]
\[
\text{apply (rule sum_mono2[OF finia2 subsa2]) by auto}
\]
\[
\text{also have } E2: \ldots = (\sum (x,y) \in \{(q,b)|b. \text{index} \xs' b < k'\}. (if } b!(\text{index init } y) \text{ then } \text{2::real else 1})
\]
\[
\text{by (simp only: lulae[symmetric])}
\]
\[
\text{finally have aa: } A \leq (\sum (x,y) \in \{(q,b)|b. \text{index} \xs' b < k'\}. (if } b!(\text{index init } y) \text{ then } \text{2::real else 1})
\]
\[
\text{have sameset: } \{y. \text{index} \xs' y < k'\} = \{xs'!i | i. i < k'\}
\]
\[
\text{proof (safe, goal_cases)}
\]
\[
\text{case (1 z)}
\]
\[
\text{show } \text{?case}
\]
\[
\text{proof}
\]
\[
\text{from 1(1) have index } \xs' z < \text{index } \text{(swaps (paid_A ! n) (s_A n))} (q)
\]
\[
\text{by auto}
\]
\[
\text{also have } \ldots < \text{length } \xs' \text{using index_less_size_conv by (auto simp: queryinlist)}
\]
\[
\text{finally have index } \xs' z < \text{length } \xs'.
\]
\[
\text{then have zset: } z \in \text{set } \xs' \text{using index_less_size_conv by metis}
\]
\[
\text{have f1: } \xs'! (\text{index } \xs' z) = z
\]
\[
\text{apply (rule nth_index) using zset by auto}
\]
\[
\text{show } z = \xs'! (\text{index } \xs' z) \land \text{(index } \xs' z) < k'
\]
\[
\text{using f1 1(1) by auto}
\]
\[
\text{qed}
\]
\[
\text{next}
\]
\[
\text{case (2 k i)}
\]
\[
\text{from 2(1) have } i < \text{index } \text{(swaps (paid_A ! n) (s_A n))} (q)
\]
\[
\text{by auto}
\]
also have \( \ldots \ < \text{length } xs' \) \textbf{using} \texttt{index_less_size_conv} by (auto
simp: queryinlist)

finally have iset: \( i < \text{length } xs' \).
have index \( xs' (xs'!i) = i \) \textbf{apply} (rule \texttt{nth_id})
using iset by (auto)
with 2 show \( \text{?case by auto} \)
qed

have \texttt{aaa23: inj_on } (\lambda i. \texttt{xs}!i) \{ i. i < k' \}
apply (rule \texttt{inj_on_nth})
apply (simp)
apply (simp) \textbf{proof} (safe, goal_cases)
case (1 i)
then have \( i < \text{index } xs' \ (q) \) by auto
also have \( \ldots < \text{length } xs' \) \textbf{using} \texttt{index_less_size_conv} by (auto
simp: queryinlist)
also have \( \ldots = \text{length init} \) by auto
finally show \( i < \text{length init} \).
qed

have \texttt{aaa3: } \{ xs!i \mid i. i < k' \} = (\lambda i. \texttt{xs}!i) \ { i. i < k' } \) by auto
have \texttt{aa4: } \{(q,b)| (b. \text{ index } xs' b < k')\} = (\lambda b. (q,b)) \ { b. \text{ index } xs' b < k' }\) by auto

have unbelievable: \{ i::nat. i < k' \} = \{..<k' \} by auto

have \texttt{aadad: inj_on } (\lambda b. (q,b)) \{ b. \text{ index } xs' b < k' \}
\textbf{unfolding} \texttt{inj_on_def by (simp)}

have \( \sum (x,y)\in\{(q,b)| (\text{b. index } xs' b < k')\}. (\text{if } b!(\text{index init } y) \text{ then } 2::\text{real else } 1)\)
\( = (\sum y\in\{y. \text{ index } xs' y < k'\}. (\text{if } b!(\text{index init } y) \text{ then } 2::\text{real else } 1))\)
proof
have \( \sum (x,y)\in\{(q,b)| (\text{b. index } xs' b < k')\}. (\text{if } b!(\text{index init } y) \text{ then } 2::\text{real else } 1)\)
\( = (\sum (x,y)\in (\lambda b. (q,b)) \ { (\text{b. index } xs' b < k')\}. (\text{if } b!(\text{index init } y) \text{ then } 2::\text{real else } 1)) \) \textbf{using} \texttt{aa4 by simp}
also have \( \ldots = (\sum z\in (\lambda b. (q,b)) \ { b. \text{ index } xs' b < k'\}. (\text{if } b!(\text{index init } (\text{snd } z)) \text{ then } 2::\text{real else } 1)) \) \textbf{by} \texttt{(simp add: split_def)}
also have \( \ldots = (\sum z\in \{b. \text{ index } xs' b < k'\}. (\text{if } b!(\text{index init } (\text{snd } ((\lambda b. (q,b)) z))) \text{ then } 2::\text{real else } 1))\)

121
apply(simp only: sum.reindex[OF aadad]) by auto
also have ... = (\(\sum y \in \{y. \text{index} \; \text{xs'} \; y < k'\}. \text{(if } b!(\text{index init} \; y) \text{ then } 2::\text{real} \; 1))\)) by auto
finally show \(\text{thesis} \).
qed
also have ... = (\(\sum y \in \{xs'!i | i. \; i < k'\}. \text{(if } b!(\text{index init} \; y) \text{ then } 2::\text{real} \; 1))\)) using sameset by auto
also have ... = (\(\sum y \in (\lambda i. \; xs'!i) \cdot \{i. \; i < k'\}. \text{(if } b!(\text{index init} \; (xs'!y)) \text{ then } 2::\text{real} \; 1))\))
using sum.reindex[OF aadad] by simp
also have \(E3: \; \ldots = (\sum j::\text{nat} < k'. \text{(if } b!(\text{index init} \; (xs'!j)) \text{ then } 2::\text{real} \; 1))\))
using unbelievable by auto
finally have \(bb: (\sum (x,y)\in\{(q,b)| b. \; \text{index} \; xs' \; b < k'\}. \text{(if } b!(\text{index init} \; y) \text{ then } 2::\text{real} \; 1))\)
= \((\sum j<k'. \text{(if } b!(\text{index init} \; (xs'!j)) \text{ then } 2::\text{real} \; 1))\).

have \(A \leq (\sum j<k'. \text{(if } b!(\text{index init} \; (xs'!j)) \text{ then } 2::\text{real} \; 1))\))
using aa bb by linarith
then show \(A \leq (\sum (x,y)\in\{(q,b)| b. \; \text{index} \; xs' \; b < k'\}. \text{(if } b!(\text{index init} \; q) \text{ then } k-k' \text{ else } (\sum j<k'. \text{(if } b!(\text{index init} \; (xs'!j)) \text{ then } 2::\text{real} \; 1))\))
using False by auto
next
case True
then have samesame: \(ys' = ys\) unfolding ys'_def step_def by auto

have setxsbleibt: set xs'' = set init by auto

have whatisk': \(k' = \text{index} \; xs'' \; q\) apply(simp)
apply(rule mtf2_q_after[symmetric]) using queryinlist by auto

have \((\text{Inv} \; ys' \; xs'') - (\text{Inv} \; ys \; xs')\)
= \{(x,y). \; x < y \; \text{in} \; ys \; \land \; y < x \; \text{in} \; xs'' \; \land \; \sim y < x \; \text{in} \; xs'\}
unfolding \text{Inv_def} using samesame by auto
also have \((xs'!i,q)|i. \; i\in\{k'..<k'\}\)
apply (clarify)

proof

fix a b

assume 1: a < b in ys

and 2: b < a in xs''

and 3: ¬ b < a in xs'

then have anb: a ≠ b

using no_before_inI by (force)

have a: a ∈ set init

and b: b ∈ set init

using before_in_setD1[OF 1] before_in_setD2[OF 1] by auto

with anb 3 have 3: a < b in xs'

by (simp add: not_before_in)

note all = anb 1 2 3 a b

have bq: b = q apply (rule swapped_by_mtf2 [where xs = xs' and x = a])

using queryinlist apply (simp_all add: all)

using all (4) apply (simp)

using all (3) apply (simp) done

note mine = mtf2_backwards_effect3[THEN conjunct1]

from bq have q < a in xs'' using 2 by auto

then have (k' < index xs'' a ∧ a ∈ set xs'')

unfolding before_in_def

using whatisk' by auto

then have low: k' ≤ index xs' a

unfolding whatisk'

unfolding xs''_def

apply (subst mtf2_q_after)

apply (simp)

using queryinlist apply (simp)

apply (rule mine)

apply (simp add: queryinlist)

using bq b apply (simp)

apply (simp)

apply (simp del: xs'_def)

apply (metis 3 a before_in_def bq xs'_init k'_def k_def max_0L mtf2_forward_beforeq nth_index whatisk' xs''_def)

using a by (simp)

from bq have a < q in xs' using 3 by auto

then have up: (index xs' a < k)

unfolding before_in_def by auto

123
from a have a ∈ set xs' by simp
then have aa: a = xs' ! index xs' a using nth_index by simp

have inset: index xs' a ∈ {k'..<k}
  using low up by fastforce

from bq aa show (a, b) = (xs' ! index xs' a, q) ∧ index xs' a ∈ {k'..<k}
  using inset by simp

qed

finally have a: (Inv ys' xs'') - (Inv ys xs') ⊆ {(xs''!i.q)| i ∈ {k'..<k}}
(is ?M ⊆ ?UB).

have card_of_UB: card {(xs''!i.q)| i ∈ {k'..<k}} = k - k'
proof
  have e: fst ' ?UB = (% i. xs' ! i) ' {k'..<k} by force
  have card ?UB = card (fst ' ?UB)
    apply (rule card_image[symmetric])
    using inj_on_def by fastforce
  also have ... = card ((%i. xs' ! i) ' {k'..<k})
    by (simp only: e)
  also
    have ... = card {k'..<k}
    apply (rule card_image)
    apply (rule inj_on_nth)
    using k_inbounds by simp_all
  also
    have ... = k - k' by auto
finally
  show ?thesis.
qed

have flipit: flip (index init q) b ! (index init q) = (~ (b) ! (index init q))
  apply (rule flip_itself)
  using queryinlist setinit by auto

have q: {x∈?UB. snd x=q} = ?UB by auto

have E0: A = (∑(x,y)∈(Inv ys' xs'') - (Inv ys xs')). (if b!(index init y) then 2::real else 1)) by auto
also have E1: ... ≤ (∑(z,y)∈?UB. if flip (index init q) (b) ! (index init q) then 2::real else 1)) by auto
init y) then 2::real else 1)
  unfolding b'_def apply (rule sum_mono2 [OF _ a])
  by (simp_all add: split_def)
also have ... = (∑(z,y)∈{x∈UB. snd x=q}, if flip (index init q) (b) ! (index init y) then 2::real else 1) by (simp only: q)
also have ... = (∑ z∈{x∈UB. snd x=q}, if flip (index init q) (b) ! (index init (snd z)) then 2::real else 1) by (simp add: split_def)
also have ... = (∑ z∈{x∈UB. snd x=q}, if flip (index init q) (b) ! (index init init q) then 2::real else 1) by simp
also have E2: ... = (∑ z∈{x∈UB. if flip (index init q) (b)! (index init init q) then 2::real else 1}) by simp
also have E3: ... = (∑ z∈{x∈UB. if flip (index init q) (b)! (index init init q) then 2::real else 1}) by simp
also have E4: ... = k − k'
finally have result: A ≤ k − k'.
with True show ?thesis by auto
qed

show (∑(x,y)∈{Inv ys' xs'}. (if b!(index init init q) then 2::real else 1)) − (∑(x,y)∈{Inv ys xs'}. (if b!(index init init q) then 2::real else 1)) ≤ ?ub2
  unfolding ub_free_def teilen[unfolded ∆_def A_def B_def C_def] using BC_absch A_absch using True
  by auto
qed
from paid_ub have kl: Φ_1 x ≤ Φ_0 x + ?paidUB by auto
from free_ub have kl2: Φ_2 x − ?ub2 ≤ Φ_1 x using gis dis by auto

have iub_free: I + ?ub2 = ub_free by auto
from kl kl2 have Φ_2 x − Φ_0 x ≤ ?ub2 + ?paidUB by auto

then have (cost x) + (Φ_2 x) − (Φ_0 x) ≤ k + 1 + I + ?ub2 + ?paidUB using ub_cost_BIT by auto

then show ?case unfolding ub_free_def b_def by auto
qed

Approximation of the Term for Free exchanges
have free_absch: E(map_pmf (λx. (if (q) ∈ set init then (if (fst (snd x))!(index init q) then k − k' else (∑ j<k' (if (fst (snd x))!(index init (xs^4 j)) then 2::real else 1))) else 0)) D)
\[ \leq 3/4 * k \quad \text{(is ?EA} \leq \text{?absche)} \]

**proof** (cases \((q) \in \text{set init})

**case** False

then have ?EA = 0 by auto
then show ?thesis by auto

next
**case** True

**note** queryinlist=this

**have** \(k-k' \leq k\) by auto
**have** \(k' \leq k\) by auto

Transformation of the first term

**have** qsn: \{index init q} ∪ \{\} ⊆ \(\{0..<?l}\) using setinit queryinlist by auto

**have** \(\{l::\text{bool list. length } l = ?l \land l!(\text{index init q})\}
= \{xs. \text{Ball } \{\{\text{index init q}\}) ((!) xs) \land (\forall i\in\{\}. \neg xs ! i) \land \text{length } xs = ?l\} by auto

then have card \(\{l::\text{bool list. length } l = ?l \land l!(\text{index init q})\}
= card \{xs. \text{Ball } \{\{\text{index init q}\}) ((!) xs) \land (\forall i\in\{\}. \neg xs ! i) \land \text{length } xs = \text{length init}\} by auto

also have ... = \(2^{\text{length init} - \text{card } \{\text{index init q}\} - \text{card } \{\}}
apply(subst card2[of \{\{\text{index init q}\} \{\} \{?l\}] using qsn by auto

finally have lulu: \(\{\{l::\text{bool list. length } l = ?l \land l!(\text{index init q})\}
= \(2^{\text{length init} - 1}\) by auto

**have** \((\sum x\in\{l::\text{bool list. length } l = ?l \land l!(\text{index init q})\}. \text{real}(k-k'))
= \((\sum x\in\{l::\text{bool list. length } l = ?l \land l!(\text{index init q})\}. k-k') by auto

also have ... = \((k-k')*2^{\text{length init} - 1}\) using lulu by simp

finally have absch1stterm: \((\sum x\in\{l::\text{bool list. length } l = ?l \land l!(\text{index init q})\}. \text{real}(k-k'))
= \text{real}((k-k')*2^{\text{length init} - 1}) .

Transformation of the second term

**let** \(?S=\{(xs^?j)|j. j<k'/\}

from queryinlist **have** \(q \in \text{set } \{\text{swaps } \text{(paid}_A \text{! n) (s}_A \text{ n)}\} by auto

126
then have \( \text{index} (\text{swaps} (\text{paid} \_A \! \ n) \ (s \_A \ n)) \) \( q < \text{length} \ xs' \) by auto

then have \( k' \) inbound: \( k' < \text{length} \ xs' \) by auto

\[
\begin{align*}
&\{ \text{fix} \ x \} \\
&\quad \text{have} \ a: \{..<k'\} = \{j. \ j<k'\} \text{ by auto} \\
&\quad \text{have} \ b: \ ?S = ((\%j. \ xs' \! j) \ {\{\ j. \ j<k'\}) \text{ by auto} \\
&\quad \text{have} \ (\sum j<k'. (\lambda t. (\text{if} \ x!(\text{index init} \ t) \text{ then } 2::\text{real else } 1)) (xs'!j))) \\
&\quad \quad \text{by (auto)} \\
&\quad \quad \text{also have} \ \ldots = \text{sum} ((\lambda t. (\text{if} \ x!(\text{index init} \ t) \text{ then } 2::\text{real else } 1)) o (\%j. \ xs'!j)) \\
&\quad \quad \quad \text{by (simp only: a)} \\
&\quad \quad \quad \text{also have} \ \ldots = \text{sum} (\lambda t. (\text{if} \ x!(\text{index init} \ t) \text{ then } 2::\text{real else } 1)) \ (\%j. \ xs'!j)) \\
&\quad \quad \quad \quad \text{apply (rule sum.reindex[ symmetric])} \\
&\quad \quad \quad \quad \text{apply (rule inj_on_nth)} \\
&\quad \quad \quad \quad \text{using} \ k' \text{ inbound by (simp_all)} \\
&\quad \quad \quad \text{finally have} \ (\sum j<k'. (\lambda t. (\text{if} \ x!(\text{index init} \ t) \text{ then } 2::\text{real else } 1)) (xs'!j)) \\
&\quad \quad \quad \quad \quad \text{using} \ b \text{ by simp} \\
&\quad \text{note} \ \text{reindex=this} \\
&\text{have} \ \text{identS}: \ ?S = \text{set} \ (\text{take} \ k' \ xs') \\
&\text{proof} - \\
&\quad \text{have} \ \text{index} (\text{swaps} (\text{paid} \_A \! \ n) \ (s \_A \ n)) \ (q) \leq \text{length} \ (\text{swaps} \ (\text{paid} \_A \! \ n) \ (s \_A \ n)) \\
&\quad \quad \text{by (rule index.le.size)} \\
&\quad \quad \text{then have} \ kxs': \ k' \leq \text{length} \ xs' \text{ by simp} \\
&\quad \quad \text{have} \ ?S = \ (!) \ xs' \ {\{0..<k'\} \text{ by force} \\
&\quad \quad \text{also have} \ \ldots = \text{set} \ (\text{take} \ k' \ xs') \ \text{apply (rule nth_image)} \ \text{by (rule kxs')} \\
&\quad \quad \quad \text{finally show} \ ?S = \text{set} \ (\text{take} \ k' \ xs') . \\
&\text{qed} \\
&\text{have} \ \text{distinctS}: \ \text{distinct} \ (\text{take} \ k' \ xs') \ \text{using} \ \text{distinct.take identS} \ \text{by simp} \\
&\text{have} \ \text{lengthS}: \ \text{length} \ (\text{take} \ k' \ xs') = k' \ \text{using} \ \text{length.take k' inbound} \ \text{by simp} \\
&\quad \text{from} \ \text{distinct.card[OF distinctS] lengthS} \ \text{have} \ \text{card} \ (\text{set} \ (\text{take} \ k' \ xs')) = k' \ \text{by simp} \\
&\quad \text{then have} \ \text{cardS}: \ \text{card} \ ?S = k' \ \text{using identS} \ \text{by simp}
\end{align*}
\]

127
have a: ?S ⊆ set xs' using set_take_subset identS by metis
then have Ssubso: (index init) ' ?S ⊆ {0..<?l} using setinit by auto

from a have s_subst_init: ?S ⊆ set init by auto

note index_inj_on_S = subset_inj_on[OF inj_on_index[of init] s_subst_init]

have l: xs'!k = q unfolding k_def apply(rule nth_index) using queryinlist by (auto)

have xsl!k ∉ set (take k' xs')
  apply(rule index_take) using l by simp
then have requestnotinS: (q) ∉ ?S using l identS by simp
then have indexnotin: index init q ∉ (index init) ' ?S
  using index_inj_on_S s_subst_init by auto

have lua: {l. length l = ?l ∧ ~l!(index init q)}
  = {xs. (∀ i∈{}. xs ! i) ∧ (∀ i∈{index init q}. ¬ xs ! i) ∧ length xs
  = ?l} by auto

from k' inbound have k' inbound2: Suc k' ≤ length init using Suc_le_eq by auto

have (∑ x∈{l::bool list. length l = ?l ∧ ~l!(index init q)}. (∑ j<k'.
  (if x!(index init (xs'!j)) then 2::real else 1))))
  = (∑ x∈{l. length l = ?l ∧ ~l!(index init q)}. (∑ j∈?S. (λt.
  (if x!(index init t) then 2 else 1)) j))
  using reindex by auto

also
have ... = (∑ x∈{xs. (∀ i∈{}. xs ! i) ∧ (∀ i∈{index init q}. ¬ xs ! i)
  ∧ length xs = ?l}. (∑ j∈?S. (λt. (if x!(index init t) then 2 else 1)) j))
  using lua by auto
also
have ... = (∑ x∈{xs. (∀ i∈{}. xs ! i) ∧ (∀ i∈{index init q}. ¬ xs ! i)
  ∧ length xs = ?l}. (∑ j∈{index init t} ' ?S. (λt. (if x!(index init t) then 2 else 1)) j))
proof –
  { fix x
  have (∑ j∈?S. (λt. (if x!(index init t) then 2 else 1)) j)
\[
\sum_{j \in \{ \text{index init} \}} \cdot ?S. \ (\lambda t. (if x!t then 2 else 1)) \ j
\]
apply(simp only: sum.reindex[OF index inj_on_S, where 
g=(%j. if x! j then 2 else 1)])
by(simp)

} note a=this
show thesis by(simp only: a)
qed

also
have \ldots = 3 / 2 * \text{real} (\text{card} \ ?S) * 2 ^ (\?l - \text{card} \ {\text{q}}) 
apply(subst Expactation2or1)
apply(simp)
apply(simp)
apply(simp)
apply(simp)
apply(simp only: card_image index inj_on_S cardS)
apply(simp add: k\'inbound2 del: k\'_def)
using indextoind apply(simp add: )
apply(simp)
using Subso queryinlist apply(simp)
apply(simp only: card_image[OF index inj_on_S]) by simp
finally have \((\sum x \in \{l. \text{length} \ l = ?l \land \neg \ l ! (\text{index init} \ q)\}. \ \sum j < k\').
if x ! (\text{index init} (xs'! j)) then 2 else 1)
= 3 / 2 * \text{real} (\text{card} \ ?S) * 2 ^ (\?l - \text{card} \ {\text{q}}) \cdot

also
have 3 / 2 * \text{real} (\text{card} \ ?S) * 2 ^ (\?l - \text{card} \ {\text{q}}) =
(3/2) * (\text{real} (k\')) * 2 ^ (\?l - 1) using cardS by auto

finally have absch2ndterm: \((\sum x \in \{l. \text{length} \ l = ?l \land \neg \ l ! (\text{index init} \ q)\}).
\sum j < k\'. \ if x !(\text{index init} (xs'! j)) then 2 else 1) =
3 / 2 * \text{real} (k\') * 2 ^ (\?l - 1) \cdot

Equational transformations to the goal
have cardonebitset: card \{l::bool list. \text{length} \ l = ?l \land \text{!(index init} \ q)\}
= 2 ^ (\?l - 1) using lulu by auto

have splitie: \{l::bool list. \text{length} \ l = ?l\}
= \{l::bool list. \text{length} \ l = ?l \land \text{!(index init} \ q)\} \cup \{l::bool list. \text{length} \ l = ?l \land \sim \text{!(index init} \ q)\}
by auto
have interempty: \{l::bool list. \text{length} \ l = ?l \land \text{!(index init} \ q)\} \cap \{l::bool
list. length \( l = \emptyset | \sim \emptyset \{\text{index init } q\}\)

\[ = \{\} \text{ by auto} \]

**have fa: finite \( \{l::\text{bool list. length } l = \emptyset | \sim \emptyset \{\text{index init } q\}\}\)** using

**bitstrings_finite by auto**

**have fb: finite \( \{l::\text{bool list. length } l = \emptyset | \sim \emptyset \{\text{index init } q\}\}\)** using

**bitstrings_finite by auto**

\[
\{\text{fix } f :: \text{bool list } \Rightarrow \text{real}\}
\]

**have** \((\sum x\in\{l::\text{bool list. length } l = \emptyset\}. f x)\)

\[ = (\sum x\in\{l::\text{bool list. length } l = \emptyset | \sim \emptyset \{\text{index init } q\}\}. f x) \text{ by (simp only: splitie)} \]

**also have...**

\[ = (\sum x\in\{l::\text{bool list. length } l = \emptyset \sim \emptyset \{\text{index init } q\}. f x) \]

\[ + (\sum x\in\{l::\text{bool list. length } l = \emptyset | \sim \emptyset \{\text{index init } q\}\}. f x) \] by (simp add: interempty)

**finally have** \(\text{sum } f \{l. \text{length } l = \text{length init}\} = \text{sum } f \{l. \text{length } l = \text{length init } \sim \sim ! \{\text{index init } q\}\} \)

\} note darfstspliten=thes

**have E1: E(map_pmf (λx. (if (fst (snd x)))\{\text{index init } q\} then real(k-k') else (\sum j<k'. (if (fst (snd x)))\{\text{index init } \{xs!\}j\} then 2::real else 1)))) D)\]

\[ = E(map_pmf (λx. (if x!(\text{index init } q) then real(k-k') else (\sum j<k'. (if x!(\text{index init } \{xs!\}j\}) then 2::real else 1)))\ (map_pmf (fst o snd) D)) \]

**proof –**

**have triv: \(\lambda x. (\text{fst o snd} x) = \text{fst (snd x)}\)** by simp

**have E((map_pmf (λx. (if (fst (snd x)))\{\text{index init } q\} then real(k-k') else (\sum j<k'. (if (fst (snd x)))\{\text{index init } \{xs!\}j\} then 2::real else 1)))) D)\]

\[ = E(map_pmf (λx. (if y!(\text{index init } q) then real(k-k') else (\sum j<k'. (if y!(\text{index init } \{xs!\}j\}) then 2::real else 1))) o (\text{fst o snd}) x) \]

**apply (auto simp: comp_assoc) by (simp only: triv)**

**also have... = E((map_pmf (λx. (if x!(\text{index init } q) then...
real(k – k') else (∑ j<k'. (if x!(index init (xs’j)) then 2::real else 1)))) ○
(map_pmf (fst ⨅ snd)) D)

using map_pmf_compose by metis

also have ... = E(map_pmf (λx. (if x!(index init q) then real(k – k')
else (∑ j<k'. (if x!(index init (xs’j)) then 2::real else 1)))) (map_pmf (fst ⨅ snd) D)) by auto

finally show ?thesis .

qed

also have E2: ... = E(map_pmf (λx. (if x!(index init q) then real(k – k')
else (∑ j<k'. (if x!(index init (xs’j)) then 2::real else 1)))) (bv ?l))

using config_n_bv[of init _] by auto

also let ?insf = (λx. (if x!(index init q) then k – k' else (∑ j<k'. (if x!(index
init (xs’j)) then 2::real else 1))))

have E3: ... = (∑ x∈iset_pmf (bv ?l)). (?insf x) * pmf (bv ?l) x

by (subst E_finite_sum_fun) (auto simp: bv_finite mult_ac)

also have ... = (∑ x∈{l::bool list. length l = ?l}. (?insf x) * pmf (bv ?l) x)

using bv_set by auto

also have E4: ... = (∑ x∈{l::bool list. length l = ?l}. (?insf x) * (1/2) ^?l)

using list_pmf by auto

also have ... = (∑ x∈{l::bool list. length l = ?l}. (?insf x)) * ((1/2) ^?l)

by (simp only: sum_distrib_right[where r=(1/2) ^?l])

also have E5: ... = ((1/2) ^?l) * (∑ x∈{l::bool list. length l = ?l}. (?insf x))

by (auto)

also

have E6: ... = ((1/2) ^?l) * (∑ x∈{l::bool list. length l = ?l ∧
l!(index init q)}. ?insf x)

+ (∑ x∈{l::bool list. length l = ?l ∧ ~l!(index init
q)}. ?insf x)

) using darfstsplitten by auto

also

have E7: ... = ((1/2) ^?l) * (∑ x∈{l::bool list. length l = ?l ∧
l!(index init q)}. ((λx. real(k – k'))) x)

+ (∑ x∈{l::bool list. length l = ?l ∧ ~l!(index init
q)}. ((λx. (∑ j<k’. (if x!(index init (xs’j)) then 2::real else 1)))) x)

) by auto

finally have E(map_pmf (λx. (if (fst (snd x))!(index init q) then
real(k′) \text{ else } (\sum_{j<k′} (if (fst (snd x))!(\text{index init } (xs^q_j)) \text{ then } 2::\text{real else } 1))) \text{ D}) \\
= (\frac{1}{2} \cdot \text{?l}) \cdot ( (\sum_{x \in \{l::\text{bool list. length } l = ?l \land \text{?l}!(\text{index init } q)\}} ((\lambda x. \text{real}(k-k'))(x) \\
+ (\sum_{x \in \{l::\text{bool list. length } l = ?l \land \sim \text{?l}!(\text{index init } q)\}} ((\lambda x. (\sum_{j<k′} (if x!(\text{index init } (xs^q_j)) \text{ then } 2::\text{real else } 1))))(x)) \\
) \\
\text{ also have } \ldots = ((1/2)^{-?l}) \cdot ( (\sum_{x \in \{l::\text{bool list. length } l = ?l \land \text{?l}!(\text{index init } q)\}} \text{real}(k-k')) \\
+ (\frac{3}{2} \cdot \text{real}(k′) \cdot 2^{-?l} \cdot ?l^{-1}) \\
\text{ by (simp only: absch2ndterm)} \\
\text{ also have } E8: \ldots = ((1/2)^{-?l}) \cdot ( \text{real}((k-k′) \cdot 2^{-?l} \cdot ?l^{-1}) + (\frac{3}{2} \cdot \text{real}(k′) \cdot 2^{-?l} \cdot ?l^{-1}) \\
\text{ by (simp only: absch1stterm)} \\
\text{ also have } \ldots = ((1/2)^{-?l}) \cdot ( (k-k′) + (k′ \cdot (\frac{3}{2}) \cdot 2^{-?l} \cdot ?l^{-1}) \\
\text{ by (simp only: distrib_right) by simp} \\
\text{ also have } \ldots = ((1/2)^{-?l}) \cdot 2^{-?l} \cdot ?l^{-1} \cdot ( (k-k′) + (k′ \cdot (\frac{3}{2}) \cdot 2^{-?l} \cdot ?l^{-1}) \\
\text{ by (simp only: lSuc by auto) \\
\text{ also have } E9: \ldots = (1/2) \cdot ( \text{real}(k-k′) + (k′ \cdot (\frac{3}{2}) \cdot 2^{-?l} \cdot ?l^{-1}) \\
\text{ proof – \\
\text{ have } ((1::\text{real})/2)^{-?l'} \cdot 2^{-?l'} = ((1::\text{real})/2 \cdot 2^{-?l'}) \cdot ?l' \text{ by (rule power_mult_distrib[ symmetric])} \\
\text{ also have } \ldots = 1 \text{ by auto} \\
\text{ finally have } (((1::\text{real})/2)^{-?l'}) \cdot 2^{-?l'} = (1/2) \text{ by auto} \\
\text{ then show } ?\text{thesis by auto \\
\text{ qed \\
\text{ also have } E10: \ldots \leq (1/2) \cdot ( (\frac{3}{2}) \cdot (k-k′) + (k′ \cdot (\frac{3}{2}) \cdot 2^{-?l} \cdot ?l^{-1}) \\
\text{ by (auto \text{ auto \\
\text{ also have } \ldots = (1/2) \cdot ( (\frac{3}{2}) \cdot (k-k′) + (k′ \cdot (\frac{3}{2}) \cdot 2^{-?l} \cdot ?l^{-1}) \text{ by (auto \text{ auto \\
\text{ also have } E11: \ldots = (3/4) \cdot (k \text{ by auto} \\
\text{ finally show } E(\text{map pmf}(\lambda x. (if q \in \text{set init then } (if (fst (snd x))!(\text{index init } q) \text{ then real} (k-k′) \text{ else } (\sum_{j<k′} (if x!(\text{index init } (xs^q_j)) \text{ then } 2::\text{real else } 1)))) \text{ else } 0 )) \text{ D}) \\
\leq 3/4 \cdot k \text{ using True by simp \\
\text{ qed \\

\begin{center}
\text{132}
\end{center}
Transformation of the Term for Paid Exchanges

\[ \text{have } \text{paid}\_\text{absch}: E(\text{map}\_\text{pmf} (\lambda x. (\sum i<(\text{length } (\text{paid}_A!n))). (\text{if } ((\text{fst} (\text{snd} x)))!(\text{gebub } n \ i) \ \text{then } 2::\text{real else } 1) ) ) D) = 3/2 * (\text{length } (\text{paid}_A!n) ) \]

\[ \text{proof --} \]

\{ \\
  \text{fix } i \\
  \text{assume inbound: } (\text{index init } i) < \text{length init} \\
  \text{have map}\_\text{pmf} (\lambda xx. \text{if } \text{fst} (\text{snd} \ xx) \ ! (\text{index init } i) \ \text{then } 2::\text{real else } 1) D = \\
  \quad \text{bind}\_\text{pmf} (\text{map}\_\text{pmf} (\text{fst } \circ \text{snd} ) D) (\lambda b. \text{return}\_\text{pmf} (\text{if } b! \text{index init } i \ \text{then } 2::\text{real else } 1)) \\
  \quad \text{unfolding map}\_\text{pmf}\_\text{def by}(\text{simp add: } \text{bind}\_\text{assoc}\_\text{pmf} \\
  \text{bind}\_\text{return}\_\text{pmf}) \\
  \quad \text{also have } \ldots = \text{bind}\_\text{pmf} (\text{bv } (\text{length init})) (\lambda b. \text{return}\_\text{pmf} (\text{if } b! \text{index init } i \ \text{then } 2::\text{real else } 1)) \\
  \quad \text{using config}\_n\_\text{bv}[\text{of init take } n \ qs] \ \text{by simp} \\
  \quad \text{also have } \ldots = \text{map}\_\text{pmf} (\lambda yy. (\text{if } yy \ \text{then } 2\text{ else } 1)) (\text{map}\_\text{pmf} (\lambda y. y! (\text{index init } i))) (\text{bv } (\text{length init})) \\
  \quad \text{by (simp add: } \text{map}\_\text{pmf}\_\text{def } \text{bind}\_\text{return}\_\text{pmf} \text{ bind}\_\text{assoc}\_\text{pmf}) \\
  \quad \text{also have } \ldots = \text{map}\_\text{pmf} (\lambda yy. (\text{if } yy \ \text{then } 2\text{ else } 1)) (\text{bernoulli}\_\text{pmf} (5/10)) \\
  \quad \text{by (auto simp add: } \text{bv}\_\text{comp}\_\text{bernoulli}[\text{OF inbound}]) \\
  \text{finally have } \text{map}\_\text{pmf} (\lambda xx. \text{if } \text{fst} (\text{snd} \ xx) \ ! (\text{index init } i) \ \text{then } 2::\text{real else } 1) D = \\
  \quad \text{map}\_\text{pmf} (\lambda yy. \text{if } yy \ \text{then } 2::\text{real else } 1) (\text{bernoulli}\_\text{pmf} (5/10)). \\
\} \text{ note unform } = \text{this} \\

\text{have } E(\text{map}\_\text{pmf} (\lambda x. (\sum i<(\text{length } (\text{paid}_A!n))). (\text{if } ((\text{fst} (\text{snd} x)))!(\text{gebub } n \ i) \ \text{then } 2::\text{real else } 1)))) D) = \\
\quad (\sum i<(\text{length } (\text{paid}_A!n))). E(\text{map}\_\text{pmf} ((\lambda xx. (\text{if } ((\text{fst} (\text{snd} xx)))!(\text{gebub } n \ i) \ \text{then } 2::\text{real else } 1)))) D) \\
  \text{apply}(\text{subst E_linear_sum2}) \\
  \quad \text{using finite_config\_BIT}[\text{OF dist_init}] \ \text{by}(\text{simp\_all}) \\
  \text{also have } \ldots = (\sum i<(\text{length } (\text{paid}_A!n))). E(\text{map}\_\text{pmf} (\lambda y. y! (\text{index init } i))) (\text{map}\_\text{pmf} ((\lambda xx. (\text{if } ((\text{fst} (\text{snd} xx)))!(\text{gebub } n \ i) \ \text{then } 2::\text{real else } 1)))) D) = 3/2 * (\text{length } (\text{paid}_A!n)) \\
  \text{using unform gebub_def gebub\_inBound[\text{OF 31}] by simp} \\
  \text{also have } \ldots = 3/2 * (\text{length } (\text{paid}_A!n)) \ \text{by}(\text{simp add: E_bernoulli}) \\
  \text{finally show } E(\text{map}\_\text{pmf} (\lambda x. (\sum i<(\text{length } (\text{paid}_A!n))). (\text{if } ((\text{fst} (\text{snd} x)))!(\text{gebub } n \ i) \ \text{then } 2::\text{real else } 1)))) D) = 3/2 * (\text{length } (\text{paid}_A!n)) . \\
\text{qed}
Combine the Results

have cost_A_absch: k+(length (paid_A!n)) + 1 = t_A n unfolding k_def q_def e_A_def p_A_def t_A_def by (auto)

let ?yo = (\lambda x. (cost x) + (\Phi_2 x) - (\Phi_0 x))
let ?yo2 = (\lambda x. (k+1) + (if (q) \in set init then (if (fst (snd x))!(index init q) then k-k' else (\sum j<k'). (if (fst (snd x))!(index init (xs!j)) then 2::real else 1)) ) else 0) + (\sum i<(length (paid_A!n)). (if (fst (snd x))!(gebub n i then 2 else 1)))

have E0: \text{\texttt{\_BIT n + Phi(n+1) - Phi n = E (map_pmf ?yo D)}}
using inEreinziehn by auto
also have \ldots \leq E(map_pmf ?yo2 D)
apply(rule E_mono2) unfolding D_def
apply(fact finite_config\_BIT[\text{\texttt{OF dist_init}}])
apply(fact ub_cost[unfolded D_def])
done

also have E2: \ldots = E(map_pmf (\lambda x. k + 1::real) D)
+ (E(map_pmf (\lambda x. (if (q) \in set init then (if (fst (snd x))!(index init q) then real(k-k') else (\sum j<k'). (if (fst (snd x))!(index init (xs!j)) then 2::real else 1)) ) else 0) ) D)
+ E(map_pmf (\lambda x. (\sum i<(length (paid_A!n)). (if (fst (snd x))!(gebub n i then 2::real else 1)))) D)
unfolding D_def apply(simp only: E_linear_plus2[\text{\texttt{OF finite_config\_BIT[\text{\texttt{OF dist_init}}]}} by(auto simp: add_assoc)

also have E3: \ldots \leq k + 1 + (3/4 * (real (k)) + (3/2 * real (length (paid_A!n)))) using paid_absch free_absch by auto

also have \ldots = k + (3/4 * (real k)) + 1 + 3/2 *(length (paid_A!n)) by auto
also have \ldots = (1+3/4) * (real k) + 1 + 3/2 *(length (paid_A!n)) by auto
also have E4: \ldots = 7/4*(real k) + 3/2 *(length (paid_A!n)) + 1 by auto
also have \ldots \leq 7/4*(real k) + 7/4 *(length (paid_A!n)) + 1 by auto
also have E5: \ldots = 7/4*(k+(length (paid_A!n))) + 1 by auto
also have E6: \ldots = 7/4*(t_A n - (1::real)) + 1 using cost_A_absch by auto

134
also have \( \ldots = 7/4 * t_A n \) \(- 7/4 + 1 \) by algebra
also have \( E7: \ldots = 7/4 * (t_A n) - 3/4 \) by auto
finally show \( t_{\text{BIT}} n + \Phi (n + 1) - \Phi n \leq (7/4) * t_A n - 3/4 \)

\[ \text{qed} \]
then show \( t_{\text{BIT}} n + \Phi (n + 1) - \Phi n \leq (7/4) * t_A n - 3/4 \).
\[ \text{qed} \]

9.3.7 Lift the Result to the Whole Request List

**lemma** \( T_{\text{BIT}}_{\text{absch}} \): assumes \( nqs: n \leq \text{length} \ qs \)
shows \( T_{\text{BIT}} n \leq (7/4) * T_A n - 3/4 * n \)
unfolding \( T_{\text{BIT}} \) def \( T_A \) def

**proof** –
from potential2[of Phi, OF phi0 phi_pos myub] nqs have
\[
\text{sum } t_{\text{BIT}} \{..<n\} \leq (\sum i<n. 7/4 * (t_A i)) - 3/4 by auto
\]
also have \( \ldots = (\sum i<n. 7/4 * \text{real_of_int} (t_A i)) - (\sum i<n. (3/4)) \)
by \( \text{rule sum_subtractf} \)
also have \( \ldots = (\sum i<n. 7/4 * \text{real_of_int} (t_A i)) - (3/4)*(\sum i<n. 1) \) by simp
also have \( \ldots = (\sum i<n. (7/4) * \text{real_of_int} (t_A i)) - (3/4)*n \) by simp
also have \( \ldots = (7/4) * (\sum i<n. \text{real_of_int} (t_A i)) - (3/4)*n \) by simp add: sum_distrib_left
also have \( \ldots = (7/4) * \text{real_of_int} (\sum i<n. (t_A i)) - (3/4)*n \) by auto
finally show \( \text{sum } t_{\text{BIT}} \{..<n\} \leq 7/4 * \text{real_of_int} (\text{sum } t_A \{..<n\}) \)
\(- (3/4)*n \) by auto
\[ \text{qed} \]

**lemma** \( T_{\text{BIT}}_{\text{absch}} \): assumes \( nqs: n \leq \text{length} \ qs \)
shows \( T_{\text{BIT}} n \leq (7/4) * T_A' n - 3/4*n \)
using \( nqs \) \( T_{\text{BIT}}_{\text{absch}} \) [of n] \( T_A' \) leq [of n] by auto

**lemma** \( T_A'_{\text{nneg}} \): \( 0 \leq T_A n \)
by (auto simp add: sum_nonneg \( T_A' \) def \( t_A \) def \( c_A \) def \( p_A \) def)

**lemma** \( T_{\text{BIT}}_{\text{eq}} \): \( T_{\text{BIT}} \) (length qs) \( = T_{\text{on_rand}} \) BIT init qs
unfolding \( T_{\text{BIT}} \) def \( T_{\text{on_rand}} \) as sum using \( t_{\text{BIT}} \) def by auto

135
corollary $T_{\text{BIT}}_{competitive}$: assumes $n \leq \text{length } qs$ and $\text{init} \neq []$ and $
exists i < n. \text{qs'}!i \in \text{set init}$

shows $T_{\text{BIT}} n \leq ((7/4) - 3/((4 \times \text{size init})) \times T_{\text{A'}} n$

proof cases

assume $0: \text{real_of_int}(T_{\text{A'}} n) \leq n \times (\text{size init})$

then have $1: 3/4 \times \text{real_of_int}(T_{\text{A'}} n) \leq 3/4 \times (n \times (\text{size init}))$ by auto

have $T_{\text{BIT}} n \leq (7/4) \times T_{\text{A'}} n - 3/4 \times n$ using $T_{\text{BIT}}_{absch}[OF \ \text{assms}(1)]$ by auto

also have $\ldots = ((7/4) \times \text{real_of_int}(T_{\text{A'}} n)) - (3/4 \times (n \times \text{size init}))$

/ size init

using $\text{assms}(2)$ by simp

also have $\ldots \leq ((7/4) \times \text{real_of_int}(T_{\text{A'}} n)) - 3/4 \times T_{\text{A'}} n / \text{size init}$

by (rule $\text{diff_left_mono}[OF \ \text{divide_right_mono}[OF \ 1]] \text{ simp}$)

also have $\ldots = ((7/4) - 3/4 / \text{size init}) \times T_{\text{A'}} n$ by algebra

also have $\ldots = ((7/4) - 3/(4 \times \text{size init})) \times T_{\text{A'}} n$ by simp

finally show $\_\text{thesis}$. next

assume $0: \neg \text{real_of_int}(T_{\text{A'}} n) \leq n \times (\text{size init})$

have $T_{\text{A'}}_{\text{nneg}}: 0 \leq T_{\text{A'}} n$ using $T_{\text{A}}_{\text{nneg}}[\text{of } n] \ T_{\text{A}}_{\text{A'}}_{\text{leq}}[\text{of } n]$ $\text{assms}(1)$ by auto

have $2 - 1 / \text{size init} \geq 1$ using $\text{assms}(2)$

by (auto simp add: field_simps $\text{neg Nil conv}$)

have $T_{\text{BIT}} n \leq n \times \text{size init}$ using $T_{\text{BIT}}_{\text{ub}}[OF \ \text{assms}(3)]$ by linarith

also have $\ldots < \text{real_of_int}(T_{\text{A'}} n)$ using $0$ by linarith

also have $\ldots \leq ((7/4) - 3/4 / \text{size init}) \times T_{\text{A'}} n$ using $\text{assms}(2)$ $T_{\text{A'}}_{\text{nneg}}$

by (auto simp add: $\text{mult_le_cancel_right1 field_simps neg Nil conv}$)

finally show $\_\text{thesis}$ by simp

qed

lemma $t_{\text{A'}} t: \ n < \text{length } qs \implies t_{\text{A'}} n = \text{int } (t \ (s_{\text{A'}} n) \ (qs!n) \ (\text{acts ! n}))$

by (simp add: $\text{t}_{\text{A'}}_{\text{def}} \ t_{\text{def}} c_{\text{A'}}_{\text{def}} p_{\text{A'}}_{\text{def}} \ \text{paid}_{\text{A'}}_{\text{def}} \ \text{lenActs}$ $\text{split: prod.split}$)

lemma $T_{\text{A'}}_{\text{eq lem}}$: ($\sum_{i=0..<\text{length } qs} t_{\text{A'}} i$) $=$ $T (s_{\text{A'}} 0) (\text{drop } 0 \text{ qs}) (\text{drop } 0 \text{ acts})$

proof (induction rule: $\text{zero_induct}[\text{of } \ _\text{size qs}]$

case 1 thus $\_\text{thesis}$ by (simp add: $\text{lenActs}$)

next

136
case (2 n)
show ?case
proof cases
  assume n < length qs
  thus ?case using 2
  by (simp add: Cons_nth_drop_Suc[symmetric, where i=n] lenActs sum_head_upt_Suc t_A'_t free_A_def paid_A'_def)
next
  assume ~ n < length qs thus ?case by (simp add: lenActs)
qed
qed

lemma T_A'_eq: T_A' (length qs) = T init qs acts
using T_A'_eq_lem by (simp add: T_A'_def atLeast0LessThan)

corollary BIT_competitive3: init # [] ==> \forall i < length qs. qs!i \in set init ==>
  T_Bit (length qs) \leq ( (7/4) - 3 / (4 * length init)) * T init qs acts
using order_refl T_BIT_competitive[of length qs] T_A'_eq by (simp add: of_int_of_nat_eq)

corollary BIT_competitive2: init # [] ==> \forall i < length qs. qs!i \in set init ==>
  T_on_rand BIT init qs \leq ( (7/4) - 3 / (4 * length init)) * T init qs acts
using BIT_competitive3 T_BIT_eq by auto

corollary BIT_absch_le: init # [] ==>
  T_on_rand BIT init qs \leq (7/4) * (T init qs acts) - 3/4 * length qs
using T_BIT_absch[of length qs, unfolded T_A'_eq T_BIT_eq] by auto

end

9.3.8 Generalize Competitiveness of BIT

lemma setdi: set xs = {0..<length xs} == distinct xs
apply (rule card_distinct) by auto

theorem compet_BIT: assumes init # [] distinct init set qs \subseteq set init
shows T_on_rand BIT init qs \leq ( (7/4) - 3 / (4 * length init)) * T_opt
init qs
proof
  from assms(3) have 1: \forall i < length qs. qs!i \in set init by auto
  { fix acts :: answer list
    assume len: length acts = length qs
    interpret BIT_Off acts qs init proof qed (auto simp: assms(2) len)
from BIT_competitive2[OF assms(1) 1] assms(1)
have T_on_rand BIT init qs / ( (7/4) − 3 / (4 * length ini) ) ≤ real(T init qs acts)
  by(simp add: field_simps length_greater_0_conv[symmetric]
         del: length_greater_0_conv)
  hence T_on_rand BIT init qs / ( (7/4) − 3 / (4 * length ini) ) ≤ T_opt
apply(simp add: T_opt_def Inf_nat_def)
apply(rule LeastI2_wellorder)
using length_replicate[of length qs undefined] apply fastforce
apply auto
done
thus ?thesis using assms by(simp add: field_simps length_greater_0_conv[symmetric] del: length_greater_0_conv)
qed

theorem compet_BIT4: assumes init ≠ [] distinct init
shows T_on_rand BIT init qs ≤ 7/4 * T_opt init qs
proof−
{ fix acts :: answer list
  assume len: length acts = length qs
  interpret BIT_OFF acts qs init proof qed (auto simp: assms(2) len)
  from BIT_absch_le[OF assms(1)] assms(1)
  have (T_on_rand BIT init qs + 3 / 4 * length qs) / (7/4) ≤ real(T init qs acts)
  by(simp add: field_simps length_greater_0_conv[symmetric]
         del: length_greater_0_conv)
  hence (T_on_rand BIT init qs + 3 / 4 * length qs) / (7/4) ≤ T_opt init qs
  apply(simp add: T_opt_def Inf_nat_def)
  apply(rule LeastI2_wellorder)
  using length_replicate[of length qs undefined] apply fastforce
  apply auto
done
thus ?thesis by(simp add: field_simps length_greater_0_conv[symmetric] del: length_greater_0_conv)
qed

theorem compet_BIT2:
  compet_rand BIT (7/4) {init. init ≠ [] ∧ distinct init}
unfolding compet_rand_def
proof
  fix init
  assume init ∈ {init. init ≠ [] ∧ distinct init }
then have \( \text{ne: init} \neq [] \text{ and } a: \text{distinct init by auto} \)

\{
  \begin{align*}
  \text{fix qs} \\
  \text{assume init} \neq [] \text{ and } a: \text{distinct init} \\
  \text{then have } T_{\text{on_rand}} \text{ BIT init qs} \leq 7/4 \ast T_{\text{opt init qs}} \\
  \text{using compet\_BIT4[of init qs] by simp}
  \end{align*}
\}

\text{with a ne show } \exists b \geq 0. \forall \text{ qs. static init qs } \implies T_{\text{on_rand}} \text{ BIT init qs} \leq (7/4) \ast (T_{\text{opt init qs}}) + b

by auto

qed

end

10 Partial cost model

theory Partial\_Cost\_Model

imports Move\_to\_Front

begin

\begin{itemize}
  \item \text{definition } t_p :: 'a \text{ state } \Rightarrow 'a \Rightarrow \text{ answer } \Rightarrow \text{ nat where}
    \begin{align*}
    t_p s q a &= (\text{let } (mf, sws) = a \text{ in index } (\text{swaps sws s}) q + \text{ size sws})
    \end{align*}
  \item \text{notation } (\text{latex}) \ t_p (t^*)
  \item \text{lemma } t_p t: t_p s q a + 1 = t s q a \text{ unfolding } t_p-def \ t-def \text{ by simp add: split_def}
  \item \text{interpretation } \text{On} \_\text{Off step } t_p \text{ static }.
  \item \text{abbreviation } T_p == T
  \item \text{abbreviation } T_p_{\text{opt}} == T_{\text{opt}}
  \item \text{abbreviation } T_p_{\text{on}} == T_{\text{on}}
  \item \text{abbreviation } T_p_{\text{on_rand'}} == T_{\text{on_rand'}}
  \item \text{abbreviation } T_p_{\text{on_rand}} == T_{\text{on_rand}}
  \item \text{abbreviation } T_p_{\text{on_rand} n} == T_{\text{on_rand} n}
  \item \text{abbreviation } \text{config}_p == \text{config}
  \item \text{abbreviation } \text{compet}_p == \text{compet}
\end{itemize}

end
11 Equivalence of Regular Expression with Variables

theory RExp_Var
imports Regular_Sets.Equivalence_Checking
begin

fun castdown :: nat rexp ⇒ nat rexp where
castdown Zero = Zero
| castdown One = One
| castdown (Plus a b) = Plus (castdown a) (castdown b)
| castdown (Times a b) = Times (castdown a) (castdown b)
| castdown (Star a) = Star (castdown a)
| castdown (Atom x) = (Atom (x div 2))

fun castup :: nat rexp ⇒ nat rexp where
castup Zero = Zero
| castup One = One
| castup (Plus a b) = Plus (castup a) (castup b)
| castup (Times a b) = Times (castup a) (castup b)
| castup (Star a) = Star (castup a)
| castup (Atom x) = Atom (2*x)

lemma castdown (castup r) = r
apply(induct r) by(auto)

fun substvar :: nat ⇒ (nat ⇒ ((nat rexp) option)) ⇒ nat rexp where
substvar i σ = (case σ i of Some x ⇒ x
| None ⇒ Atom (2*i+1))

fun w2rexp :: nat list ⇒ nat rexp where
w2rexp [] = One
| w2rexp (a#as) = Times (Atom a) (w2rexp as)

lemma lang (w2rexp as) = { as }
apply(induct as)
apply(simp)
by(simp add: conc_def)

140
fun subst :: nat rexp ⇒ (nat ⇒ nat rexp option) ⇒ nat rexp where
  subst Zero _ = Zero
| subst One _ = One
| subst (Atom i) σ = (if i mod 2 = 0 then Atom i else substvar (i div 2) σ)
| subst (Plus a b) σ = Plus (subst a σ) (subst b σ)
| subst (Times a b) σ = Times (subst a σ) (subst b σ)
| subst (Star a) σ = Star (subst a σ)

lemma subst_w2rexp: lang (subst (w2rexp (xs @ ys)) σ) = lang (subst (w2rexp xs) σ) @ @ lang (subst (w2rexp ys) σ)
proof (induct xs)
case (Cons x xs)
  have lang (subst (w2rexp ((x # xs) @ ys)) σ) = lang (Times (subst (Atom x) (w2rexp (xs @ ys))) σ) by simp
  also have ... = lang (Times (subst (Atom x) σ) (subst (w2rexp (xs @ ys))) σ) by simp
  also have ... = lang (subst (Atom x) σ) @ @ (lang (subst (w2rexp (xs @ ys))) σ)) by simp
  also have ... = lang (Times (subst (Atom x) σ) (subst (w2rexp xs) σ)) @ @ (lang (subst (w2rexp ys) σ) σ) by simp
  also have ... = lang (Times (subst (Atom x) σ) (subst (w2rexp xs) σ)) @ @ lang (subst (w2rexp ys) σ) σ) by simp
  finally show ?case .
qed simp
\[ \text{concS } S1 \; S2 = \{ \text{Times } a \; b \mid a, b \in S1 \land b \in S2 \} \]

**Lemma substL_conc:** \( L \ (\text{substL} \ (L1 \ @@ L2) \ \sigma) = L \ (\text{concS} \ (\text{substL} \ L1 \ \sigma) \ (\text{substL} \ L2 \ \sigma)) \)

**Apply (simp add: concS_def substL_def) Apply (auto)**

**Proof (goal_cases)**

**Case** (1 \( x \) \( xs \) \( ys \))

**Show ?case**

apply (rule exI [where \( x = \text{Times} \ (\text{subst} \ (\text{w2rexp} \ \mathit{xs}) \ \sigma) \ (\text{subst} \ (\text{w2rexp} \ \mathit{ys}) \ \sigma) \])

apply (simp)

apply (safe)

apply (rule exI [where \( x = \mathit{xs} \)])

apply (simp add: 1 (2))

apply (rule exI [where \( x = \mathit{ys} \)])

apply (simp add: 1 (3))

using 2 (1) subst_w2rexp by (auto)

**Next**

**Case** (2 \( x \) \( xs \) \( ys \))

**Show ?case**

apply (rule exI [where \( x = \text{subst} \ (\text{w2rexp} \ (\mathit{xs} @@ \mathit{ys})) \ \sigma) \])

apply (safe)

apply (rule exI [where \( x = \mathit{xs} @@ \mathit{ys} \)])

apply (simp)

apply (rule exI [where \( x = \mathit{xs} \)])

apply (rule exI [where \( x = \mathit{ys} \)])

using 2 (2, 3) apply (simp)

using 2 (1) subst_w2rexp by (auto)

**Qed**

**Lemma L_conc:** \( L(\text{concS} \ M1 \ M2) = (L \ M1) @@ (L \ M2) \)

**Proof –**

have \( L(\text{concS} \ M1 \ M2) = (\bigcup x \in \{ \text{Times } a \; b \mid a, b \in M1 \land b \in M2 \}. \ \text{lang } x) \) unfolding concS_def by (simp)

also have \( \ldots = (\bigcup \{ \text{lang} \ (\text{Times } a \; b) \mid a, b \in M1 \land b \in M2 \} ) \) by blast

also have \( \ldots = (\bigcup \{ \text{lang } a @@ \text{lang } b \mid a, b \in M1 \land b \in M2 \} ) \) by simp

also have \( \ldots = (\bigcup \{ \text{lang } a \; a @@ \text{lang } b \mid a, b \in M1 \land b \in M2 \} ) \) by blast

also have \( \ldots = (\bigcup \{ \text{lang } a \; a @@ \text{lang } b \mid a, b \in M1 \land b \in M2 \} ) \) by simp

finally show \( \ldots \)

**Qed**

**Lemma L(M1 \cup M2) = (L M1) \cup (L M2)**
fun verund :: 'b rexp list ⇒ 'b rexp where
verund [] = Zero
| verund [r] = r
| verund (r#rs) = Plus r (verund rs)

lemma lang_verund: r ∈ L (set rs) = (r ∈ lang (verund rs))
apply (induct rs)
apply (simp)
apply (case_tac rs) by auto

lemma obtainit:
  assumes r ∈ lang (verund rs)
  shows ∃ x∈ (set (rs::nat rexp list)). r ∈ lang x
proof –
from assms have r ∈ L (set rs) by (simp only: lang_verund)
then show ?thesis by (auto)
qed

lemma lang_verund4: L (set rs) = lang (verund rs)
apply (induct rs)
apply (simp)
apply (case_tac rs) by auto

lemma lang_verund1: r ∈ L (set rs) ⇒ r ∈ lang (verund rs)
apply (induct rs)
apply (simp)
apply (case_tac rs) by auto

lemma lang_verund2: r ∈ lang (verund rs) ⇒ r ∈ L (set rs)
apply (induct rs)
apply (simp)
apply (case_tac rs) by auto

definition starS :: 'b rexp set ⇒ 'b rexp set where
starS S = {Star (verund xs)|xs. set xs ⊆ S}

lemma [] ∈ L (starS S)
unfolding starS_def apply (simp)
apply (rule ext[where x=Star (verund [])])
apply (simp)
apply (rule ext[where x=[]])

143
lemma power_mono: \( L_1 \subseteq L_2 \implies (L_1::'a\ lang)^^n \subseteq L_2 ^^n \)
apply(auto) apply(induct n) by(auto simp: conc_def)

lemma star_mono: \( L_1 \subseteq L_2 \implies \star L_1 \subseteq \star L_2 \)
apply (simp add: star_def)
apply (rule UN_mono)
apply (auto simp: power_mono)
done

lemma Lstar: \( L (\star S M) = \star (L (M)) \)
unfolding starS_def apply(auto)
proof (goal_cases)
case (1 x xs)
  from 1(2) have \( L (set xs) \subseteq L (M) \) by(rule L_mono)
  then have \( a: \star (L (set xs)) \subseteq \star (L (M)) \) by (rule star_mono)
  from 1(1) obtain \( n \) where \( x \in (lang \ (verund xs))^n \)
unfolding star_def by(auto)
  thm lang_verund4
  then have \( x \in (L (set xs))^n \) by(simp only: lang_verund4)
  then have \( x \in \star (L (set xs)) \) unfolding star_def by auto
with a have \( x \in \star (L (M)) \) by auto
  then show \( x \in \star (\bigcup x \in M. \ lang x) \) unfolding starS_def by auto
next
  case (2 x)
  then obtain \( n \) where \( x \in (\bigcup x \in M. \ lang x)^n \)
unfolding star_def by auto
  then show \( \ ?case \)
proof (induct n arbitrary: x)
  case 0
  then have t: \( x=[[\] by(simp)
  show ?case
    apply(rule exI[where \( x=\star \ Zero \)])
    apply(auto simp: t) apply(rule exI[where \( x=[\]]) by(simp)
next
  case (Suc n)
  from Suc(2) have t: \( x \in (\bigcup a \in M. \ lang a) @@ (\bigcup a \in M. \ lang a)^n \)
by (simp)
  then obtain \( A \ B \) where \( x = A \ @ \ B \) and \( A: A \in (\bigcup a \in M. \ lang a) \)
and \( B: B \in (\bigcup a \in M. \ lang a)^n \) by(auto simp: conc_def)
  then obtain \( m \) where am: \( A \in lang m \) and \( mM: m \in M \) by(auto)
  from Suc(1)[OF B] obtain \( b bs \) where \( b = \star (\verund bs) \) and \( bsM: set bs \subseteq M \ B \in lang b \) by auto
then have \( \text{Bin: } B \in \text{lang} (\text{Star (verund bs)}) \) by simp

let \( \texttt{?c} = \text{Star (verund (m\#bs))} \)

thm \( \text{Bin am mM x} \)

have \( \text{ac: lang m \subseteq lang (Star (verund (m \# bs)))} \) by (auto)

have \( \text{ad: (lang (Star (verund bs))) \subseteq lang (Star (verund (m \# bs)))} \)

apply (simp add: star_def)

apply (rule UN_mono)

apply simp_all

proof –

fix \( n \)

have \( \text{t: (lang (verund bs) ^^ n) \subseteq (lang m \cup lang (verund bs)) ^^ n} \)

by (rule power_mono) simp

then show \( \text{lang (verund bs) ^^ n \subseteq lang (verund (m \# bs)) ^^ n by (cases bs) simp_all} \)

qed

from \( \text{Bin am mM x} \) have \( x \in \text{lang m @@ (lang (Star (verund bs)))} \) by auto

then have \( x \in \text{lang (Star (verund (m \# bs))) @@ lang (Star (verund (m \# bs)))} \) using \( \text{ac ad by blast} \)

then have \( \text{x_in: } x \in \text{lang (Star (verund (m \# bs))) by (auto)} \)

show \( \text{?case} \)

apply (rule exI[where \( x=%c \)])

apply (safe)

apply (rule exI[where \( x=m\#bs \)]) apply (simp add: bsM mM)

by (fact \( x\_in \))

qed

lemma \( \text{substL-star: } L (\text{substL (star L1) } \sigma) = L (\text{starS (substL L1 \sigma)}) \)

apply (simp add: concS_def conc_def starS_def star_def)

apply auto unfolding star_def

proof –

fix \( x \ a \ n \)

assume \( x \in \text{lang (subst (w2rexp a) } \sigma) \)

moreover assume \( a \in L1 ^^ n \)

ultimately show \( \exists x a. (\exists xs. xa = \text{Star (verund xs) \land set xs} \subseteq \{\text{subst (w2rexp a) } \sigma \mid a. a \in L1\}) \land x \in \text{lang xa} \)

proof (induct \( n \) arbitrary: \( x \ a \))

case \( 0 \)
then have \( a = [] \) by auto
with \( \emptyset \) show \( ?\text{case} \) apply(simp)
apply(rule exI[where \( x = \text{Star} (\text{Zero}) \)])
apply(simp)
apply(rule exI[where \( x = [] \)])
by(simp)

next

case (Suc \( n \))
then have \( a1: a \in L1 \#@ L1 ^* \) by auto
then obtain \( A B \) where \( a2: a = A \# B \) and \( A: A \in L1 \) and \( B: B \in L1 ^* \) by auto

thm subst_w2rexp
from Suc(2) have \( x \in \text{lang} (\text{subst} (\text{w2rexp} A) \sigma) \#@ \text{lang} (\text{subst} (\text{w2rexp} B) \sigma) \) unfolding a2
by(simp only: subst_w2rexp)
then obtain \( x1 x2 \) where \( x: x = x1 \# x2 \) and \( x1: x1 \in \text{lang} (\text{subst} (\text{w2rexp} A) \sigma) \) and \( x2: x2 \in \text{lang} (\text{subst} (\text{w2rexp} B) \sigma) \) by auto
from Suc(1)[OF \( x2 B \)] obtain \( R \) \( li \) where
\( R: R = \text{Star} (\text{verund} li) \) and \( li: \text{set} li \subseteq \{ \text{subst} (\text{w2rexp} a) \sigma \mid \text{a. a} \in L1 \} \) and \( x2R: x2 \in \text{lang} R \) by auto

show \( ?\text{case} \)
apply(rule exI[where \( x = \text{Star} (\text{verund} ((\text{subst} (\text{w2rexp} A) \sigma) \# li)) \)])
apply(simp)
apply(safe)
apply(rule exI[where \( x = ((\text{subst} (\text{w2rexp} A) \sigma) \# li)) \])
apply(simp add: li)
apply(rule exI[where \( x = A \)]) apply(simp add: A)
unfolding x
proof (goal_cases)
case 1
let \( ?L = (\text{lang} (\text{subst} (\text{w2rexp} A) \sigma) \cup \text{lang} (\text{verund} li)) \)

have \( t1: x1 \in ?L \) using \( x1 \) star_mono by blast
have \( t2: \) \( x2 \in \text{star} ?L \) using \( x2R \) \( R \) star_mono apply(simp) by blast
have \( x1 \# x2 \in (?L \#@ \text{star} ?L) \) using \( t1 \) \( t2 \) by auto
then show \( ?\text{case} \)
apply(cases \( li \)) by(auto)
qed
qed
next
fix x and xs :: nat rexp list
assume x ∈ (⋃ n. lang (verund xs) ^^ n)
then obtain n where x ∈ lang (verund xs) ^^ n by auto
moreover assume set xs ⊆ {subst (w2rexp a) σ | a. a ∈ L1}
ultimately show ∃ xa. (∃ a. xa = subst (w2rexp a) σ ∧
(∃ n. a ∈ L1 ^^ n)) ∧ x ∈ lang xa
proof (induct n arbitrary: x)
case 0 then have xe: x=[] by auto
show ?case
apply(rule exI[where x=One])
apply(simp add: xe)
apply(rule exI[where x=[]])
apply(simp)
apply(rule exI[where x=0])
by(simp)

next
case (Suc n)
then have x ∈ lang (verund xs) @@ (lang (verund xs) ^^ n) by auto
then obtain x1 x2 where x: x=x1@@x2 and x1: x1∈lang (verund xs)
and x2: x2 ∈ (lang (verund xs) ^^ n) by auto
from obtainit [OF x1] obtain el
where el ∈ set xs and x1 ∈ lang el by auto
with Suc.prems obtain elem
where x1elem: x1 ∈ lang (subst (w2rexp elem) σ)
and elemL1: elem ∈ L1 by auto
from Suc.hyps [OF x2 Suc.prems(2)] obtain R word n where
R: R = subst (w2rexp word) σ and word: word ∈ L1 ^^ n and x2:
x2 ∈ lang R by auto

show ?case
apply(rule exI[where x=subst (w2rexp (elem@@word)) σ])
apply(safe)
apply(rule exI[where x=elem@@word])
apply(simp)
apply(rule exI[where x=Suc n])
proof (goal_cases)
case 1
have elem ∈ L1 by(fact elemL1)
with word
show elem @ word ∈ L1 ^^ Suc n by simp
next
case 2
have \( x_1 \in \text{lang} (\text{subst}_{w2rexp} \ \text{elem}) \ \sigma \) by (fact \( x_1 \text{elem} \))
with \( x_2[\text{unfolded} \ R] \) show ?case unfolding \( x \) apply (simp only: \( \text{subst}_{w2rexp} \)) by blast
qed
qed
qed

lemma substitutionslemma:
fixes \( E :: \text{nat rexp} \)
shows \( \text{L} (\text{subst}_{L} (\text{lang}(E)) \ \sigma) = \text{lang} (\text{subst} \ E \ \sigma) \)
proof (induct \( E \))
case (\( \text{Star} \ e \))
have \( \text{L} (\text{subst}_{L} (\text{lang}(\text{Star} \ e)) \ \sigma) = \text{L} (\text{subst}_{L} (\text{star}(\text{lang} \ e)) \ \sigma) \) by auto
also have \( \ldots = \text{star} (\text{L} (\text{subst}_{L} (\text{lang} \ e)) \ \sigma) \) by (simp only: \( \text{L}_{\text{star}} \))
also have \( \ldots = \text{star} (\text{lang} (\text{subst} \ e \ \sigma)) \) by (simp only: \( \text{Star} \))
also have \( \ldots = \text{lang} ((\text{subst} (\text{Star} \ e) \ \sigma)) \) by auto
finally show ?case .
next
case (\( \text{Plus} \ e_1 \ e_2 \))
have \( \text{L} (\text{subst}_{L} (\text{lang}(\text{Plus} \ e_1 \ e_2)) \ \sigma) = \text{L} (\text{subst}_{L} (\text{lang} \ e_1 \cup \text{lang} \ e_2)) \ \sigma) \) by simp
also have \( \ldots = \text{L} (\text{subst}_{L} (\text{lang} \ e_1) \ \sigma \cup \text{subst}_{L} (\text{lang} \ e_2) \ \sigma) \) by auto
also have \( \ldots = \text{L} (\text{subst}_{L} (\text{lang} \ e_1) \ \sigma) \cup \text{L} (\text{subst}_{L} (\text{lang} \ e_2) \ \sigma) \) by auto
also have \( \ldots = \text{lang} (\text{subst} \ e_1 \ \sigma) \cup \text{lang} (\text{subst} \ e_2 \ \sigma) \) by (simp only: \( \text{Plus} \))
also have \( \ldots = \text{lang} (\text{subst} (\text{Plus} \ e_1 \ e_2) \ \sigma) \) by auto
finally show ?case .
next
case (\( \text{Times} \ e_1 \ e_2 \))
have \( \text{L} (\text{subst}_{L} (\text{lang}(\text{Times} \ e_1 \ e_2)) \ \sigma) = \text{L} (\text{subst}_{L} (\text{lang} \ e_1 \ ** \ \text{lang} \ e_2) \ \sigma) \) by simp
also have \( \ldots = \text{L} (\text{conc}_{S} (\text{subst}_{L} (\text{lang} \ e_1) \ \sigma) \ (\text{subst}_{L} (\text{lang} \ e_2) \ \sigma)) \) by (simp only: \( \text{subst}_{\text{L}_{\text{conc}}} \))
also have \( \ldots = \text{L} (\text{conc}_{S} (\text{lang} \ e_1) \ (\text{lang} \ e_2) \ \sigma) \) by (simp only: \( \text{L}_{\text{conc}} \))
also have \( \ldots = \text{lang} (\text{conc}_{S} (\text{lang} \ e_1) \ (\text{lang} \ e_2) \ \sigma) \) by (simp only: \( \text{Times} \))
also have \( \ldots = \text{lang} (\text{Times} (\text{subst} \ e_1 \ \sigma) \ (\text{subst} \ e_2 \ \sigma)) \) by auto
also have \( \ldots = \text{lang} (\text{Times} (\text{Times} \ e_1 \ e_2 \ \sigma)) \) by auto
finally show ?case .
qed simp_all
corollary lift: \( \text{lang } e1 = \text{lang } e2 \implies \text{lang } (\text{subst } e1 \sigma) = \text{lang } (\text{subst } e2 \sigma) \)

proof –
assume eq: \( \text{lang } e1 = \text{lang } e2 \)
thm substitutionslemma
have \( \text{lang } (\text{subst } e1 \sigma) = L (\text{substL } (\text{lang } e1) \sigma) \) by(simp only: substitutionslemma)
also have \( \ldots = L (\text{substL } (\text{lang } e2) \sigma) \) using eq by simp
also have \( \ldots = \text{lang } (\text{subst } e2 \sigma) \) by(simp only: substitutionslemma)
finally show \( \text{?thesis} \).
qed

11.1 Examples

lemma \( \text{lang } (\text{Plus } (\text{Atom } (x\text{::nat})) (\text{Atom } x)) = \text{lang } (\text{Atom } x) \)
proof –
let \( ?\sigma = (\lambda i. \text{if } i=0 \text{ then } \text{Some } (\text{Atom } x) \text{ else } \text{None}) \)
let \( ?e1 = \text{Plus } (\text{Atom } 1) (\text{Atom } 1) \)
let \( ?e2 = \text{Atom } 1 \)
have \( \text{lang } (\text{Plus } (\text{Atom } x) (\text{Atom } x)) = \text{lang } (\text{subst } ?e1 ?\sigma) \) by (simp)
thm soundness
also have \( \ldots = \text{lang } (\text{subst } ?e2 ?\sigma) \)
  apply(rule lift)
  apply(rule soundness)
  by eval
also have \( \ldots = \text{lang } (\text{Atom } x) \) by auto
finally show \( \text{?thesis} \).
qed

fun seq : 'a rexp list ⇒ 'a rexp where
seq [] = One |
seq [r] = r |
seq (r#rs) = Times r (seq rs)

abbreviation question where question x == Plus x One

definition L_abcases (x::nat) y=
  verund [seq[question (Atom x),(Atom y), (Atom y)],
           seq[question (Atom x),(Atom y),(Atom x),Star(Times (Atom y)(Atom x)),(Atom y),(Atom y)],
           seq[question (Atom x),(Atom y)(Atom x)],(Atom y),(Atom y)],[seq[question (Atom x),(Atom y)],(Atom y),(Atom y)],[seq[question (Atom x),(Atom y)],(Atom y)],
           seq[question (Atom x)],(Atom y)],
           seq[question (Atom x)],(Atom y)],[seq[question (Atom x)],(Atom y)],[seq[question (Atom x)],(Atom y)],[seq[question (Atom x)],(Atom y)]

149
\[
\text{seq[question (Atom } x, (Atom } y, (Atom } x), \text{Star(Times (Atom } y)(Atom } x)), \text{seq[(Atom } x, (Atom } x))]
\]

definition \(L_A \ x \ y = \text{seq[question (Atom } x, (Atom } y), (Atom } y)]\)
definition \(L_B \ x \ y = \text{seq[question (Atom } x, (Atom } y), (Atom } x), \text{Star(Times (Atom } y)(Atom } x)), (Atom } y), (Atom } y)]\)
definition \(L_C \ x \ y = \text{seq[question (Atom } x, (Atom } y), (Atom } x), \text{Star(Times (Atom } y)(Atom } x)), (Atom } x)]\)
definition \(L_D \ x \ y = \text{seq[(Atom } x, (Atom } x)]\)

lemma \(L_4 \text{cases } x \ y = \text{verund } [L_A \ x \ y, L_B \ x \ y, L_C \ x \ y, L_D \ x \ y]\)
unfolding \(L_A \_\text{def } L_B \_\text{def } L_C \_\text{def } L_D \_\text{def } L_4 \text{cases} \_\text{def} \) by auto

definition \(L_{\text{lasthasxx}} \ x \ y = (\text{Plus (seq[question (Atom } x), \text{Star(Times (Atom } y)(Atom } x))), (Atom } y), (Atom } y))\)
\text{seq[question (Atom } y), \text{Star(Times(Atom } x) (Atom } y)), (Atom } x)]\)

lemma \(\text{lastxx_com: } \text{lang } (L_{\text{lasthasxx}} (x::nat) \ y) = \text{lang } (L_{\text{lasthasxx}} \ y \ x)\)
is \(\text{lang } ?A = \text{lang } ?B\)

proof –
let \(\sigma = (\lambda i. \text{if } i = 0 \text{ then Some (Atom } x) \text{ else } (i = 1 \text{ then Some (Atom } y) \text{ else None}))\)

let \(?e1 = \text{Plus (seq[Plus (Atom } 1) \text{ One}, \text{Star(Times (Atom } 3) (Atom } 1))}, (Atom } 3), (Atom } 3)]\)
\text{seq[Plus (Atom } 3) \text{ One}, \text{Star(Times (Atom } 1) \text{(Atom } 3))}, (Atom } 1), (Atom } 1)]\)
let \(?e2 = \text{Plus (seq[Plus (Atom } 3) \text{ One}, \text{Star(Times (Atom } 1) \text{(Atom } 3))}, (Atom } 1), (Atom } 1)]\)
\text{seq[Plus (Atom } 1) \text{ One}, \text{Star(Times (Atom } 3) \text{(Atom } 1))}, (Atom } 3), (Atom } 3)]\)

have \(\text{lang } ?A = \text{lang (subst } ?e1 \ ?\sigma)\) by (simp add: \(L_{\text{lasthasxx}} \_\text{def}\))
thm soundness
also have \(\ldots = \text{lang (subst } ?e2 \ ?\sigma)\)
apply (rule lift)
apply (rule soundness)
by eval
also have \(\ldots = \text{lang } ?B\) by (simp add: \(L_{\text{lasthasxx}} \_\text{def}\))
finally show \(?\text{thesis} \).

150
qed

lemma lastxx_is_4cases: lang (L_4cases x y) = lang (L_lasthasxx x y) (is lang ?A = lang ?B)
proof
  let ?σ = (λi. (if i=0 then Some (Atom x) else (if i=1 then Some (Atom y) else None)))

  let ?e1 = (Plus (seq[Plus (Atom 1) One,(Atom 3), (Atom 3)]))
  (Plus (seq[Plus (Atom 1) One,(Atom 3), (Atom 1),Star(Times (Atom 3) (Atom 1)),(Atom 3),(Atom 3)]))
  (Plus (seq[Plus (Atom 1) One,(Atom 3), (Atom 1),Star(Times (Atom 3) (Atom 1)),(Atom 1)]))
  (seq[(Atom 1), (Atom 1)]))

  let ?e2 = (Plus (seq[Plus (Atom 1) One, Star(Times (Atom 3) (Atom 1)),(Atom 3),(Atom 3)]))
  (seq[Plus (Atom 3) One, Star(Times (Atom 1) (Atom 3)),(Atom 1),(Atom 1)]))

  have lang ?A = lang (subst ?e1 ?σ) by (simp add: L_4cases_def)
  thm soundness
  also have ... = lang (subst ?e2 ?σ)
    apply(rule lift)
    apply(rule soundness)
    by eval
  also have ... = lang ?B by (simp add: L_lasthasxx_def)
finally show ?thesis .
qed

definition myUNIV x y = Star (Plus (Atom x) (Atom y))

lemma myUNIV_alle: lang (myUNIV x y) = {xs. set xs ⊆ {x,y}}
proof
  have star {y, [x]} = {concat ws |ws. set ws ⊆ {[y], [x]}} by(simp only: star_conv_concat)
  also have ... = {xs. set xs ⊆ {x, y}} apply(auto) apply(cases x=y)
apply(simp)
  apply(case_tac ws)
  apply(simp)
  apply(auto)
proof (goal_cases)
  case (I as)
  then show ?case
proof (induct as)
  case (Cons a as)
  then have as: set as ⊆ {x,y} and axy: a ∈ {x,y} by auto
  from Cons(1)(OF as) obtain ws where asco: as = concat ws
  and ws: set ws ⊆ {[y],[x]} by auto
  show ?case
    apply (rule exI[where x=[a]#ws])
    using axy by (auto simp add: asco ws)
  qed (rule exI[where x=[], simp])
qed

finally show ?thesis by (simp add: myUNIV_def)
qed

definition nodouble x y = (Plus
  (seq[question (Atom x), Star(Times(Atom y)(Atom x)),(Atom y)])
  (seq[question (Atom y), Star(Times(Atom x) (Atom y)),(Atom x)]))

lemma myUNIV_char: lang (myUNIV (x::nat) y) = lang (Times (Star (L_lasthasxx x y)) (Plus One (nodouble x y))) (is lang ?A = lang ?B)
proof –
  let ?σ = (λi. (if i=0 then Some (Atom x) else (if i=1 then Some (Atom y) else None)))
  let ?e1 = Star (Plus (Atom 1) (Atom 3))
  let ?e2 = (Times (Star (Plus (seq [Plus (Atom 1) One, Star (Times (Atom 3) (Atom 1))], Atom 3, Atom 3)])
    (seq [Plus (Atom 3) One, Star (Times (Atom 1) (Atom 3)), Atom 1, Atom 1]))
  (Plus One
    (Plus
      (seq
        [Plus (Atom 1) One, Star (Times (Atom 3) (Atom 1)), Atom 3])
      (seq
        [Plus (Atom 3) One, Star (Times (Atom 1) (Atom 3))]
      (Atom 3))
    (seq
      [Plus (Atom 3) One, Star (Times (Atom 1) (Atom 3))
        (Atom 1)]))

152
(Atom 3),
(Atom 1)))))

have lang ?A = lang (subst ?e1 ?σ) by (simp add: myUNIV_def)
thm soundness
also have ... = lang (subst ?e2 ?σ)
  apply (rule lift)
  apply (rule soundness)
  by eval
also have ... = lang ?B by (simp add: L_lasthasxx_def nodouble_def)
finally show ?thesis .
qed

definition mycaseexy x y = Plus (seq [Star (Plus (Atom x) (Atom y)), Atom x, Atom x, Atom y])
  (seq [Star (Plus (Atom x) (Atom y)), Atom y, Atom y, Atom x])
definition mycaseexx x y = Plus (seq [Star (Plus (Atom x) (Atom y)), Atom x, Atom y, Atom x])
  (seq [Star (Plus (Atom x) (Atom y)), Atom y, Atom x, Atom y])
definition mycaseex x y = Plus (seq [Star (Plus (Atom x) (Atom y)), Atom x, Atom x])
  (seq [Star (Plus (Atom x) (Atom y)), Atom y, Atom y])
definition mycaseexy x y = Plus (seq [Atom x, Atom y]) (seq [Atom y, Atom x])
definition mycases x y = Plus (Atom y) (Atom x)

definition mycases x y = Plus
  (mycaseexy x y)
  (Plus (mycaseexx x y)
    (Plus (mycaseex x y)
      (Plus (mycaseexy x y) (Plus (mycases x y) (One)))))

lemma mycases_char: lang (myUNIV (x::nat) y) = lang (mycases x y) (is lang ?A = lang ?B)
proof -
  let ?σ = (λi. (if i=0 then Some (Atom x) else (if i=1 then Some (Atom y) else None)))

  let ?e1 = Star (Plus (Atom 1) (Atom 3))
  let ?e2 = Plus (seq [Star (Plus (Atom 1) (Atom 3)), Atom 1, Atom
have \( \text{lang } ?A = \text{lang } \left( \text{subst } ?e1 \ ?\sigma \right) \) by (simp add: myUNIV_def)

thm soundness

also have ... = \text{lang } \left( \text{subst } ?e2 \ ?\sigma \right)

apply (rule lift)

apply (rule soundness)

by eval

also have ... = \text{lang } ?B by (simp add: mycases_def mycaseyxy_def mycasexy_def mycasex_def mycasexy_def)

finally show \(?\text{thesis}\).

qed

end

12 OPT2

theory OPT2

imports

Partial_Cost_Model

RExp_Var

begin

lemma \((N::\text{nat set}) \neq \{\} \implies \text{Inf } N : N\)

unfolding Inf_nat_def using LeastI[of \(\%x. \ x : N\)] by force

lemma nn_contains_Inf:

fixes \(S::\text{nat set}\)

assumes \(\text{nn: } S \neq \{\}\)

shows \(\text{Inf } S \in S\)

using \(\text{assms Inf}_\text{nat_def LeastI}\) by force
12.1 Definition

fun OPT2 :: 'a list ⇒ 'a list ⇒ (nat * nat list) list where
  OPT2 [] [x,y] = []
| OPT2 [a] [x,y] = [(0,[])]
| OPT2 (a#b#σ') [x,y] = (if a=x then (0,[]) # (OPT2 (b#σ') [x,y])
  else (if b=x then (0,[]) # (OPT2 (b#σ') [x,y])
  else (1,[]) # (OPT2 (b#σ') [y,x]))

lemma OPT2_length: length (OPT2 σ [x, y]) = length σ
apply(induct σ arbitrary: x y)
apply(simp)
apply(case_tac σ) by(auto)

lemma OPT2x: OPT2 (x#σ') [x,y] = (0,[]) # (OPT2 σ' [x,y])
apply(cases σ') by(simp_all)

lemma swapOpt: T_p opt [x,y] σ ≤ 1 + T_p opt [y,x] σ
proof −
  show ?thesis
  proof (cases length σ > 0)
    case True
      have T_p opt [y,x] σ ∈ {T_p [y,x] σ as |as. length as = length σ}
      unfolding T_opt_def
      apply(rule nn_contains_Inf)
      apply(auto) by (rule Ex_list_of_length)
      then obtain asyx where costyx: T_p [y,x] σ asyx = T_p opt [y,x] σ
        and lenyx: length asyx = length σ
        unfolding T_opt_def by auto
      from True lenyx have length asyx > 0 by auto
      then obtain A asyx' where aa: asyx = A # asyx' using list.exhaust
        by blast
      then obtain m1 a1 where AA: A = (m1,a1) by fastforce
      let ?asxy = (m1,a1@[0]) # asyx'
      from True obtain q σ' where qq: σ = q # σ' using list.exhaust by blast

155
have \( t : t_p [x, y] q (m1, a1 @[0]) = \text{Suc} (t_p [y, x] q (m1, a1)) \)
unfolding \( t_p\_\text{def} \)
apply(simp) unfolding swap\_def by (simp)

have \( s : \text{step} [x, y] q (m1, a1 @ [0]) = \text{step} [y, x] q (m1, a1) \)
unfolding step\_def mtf2\_def by(simp add: swap\_def)

have \( T : T_p [x,y] \sigma ?\text{asxy} = T_p [y,x] \sigma \text{asxy} \text{ unfolding } \text{gq aa AA} \)
by(auto simp add: s t)

have \( l : 1 + T_p \text{opt [y,x]} \sigma ?\text{asxy} = T_p [x, y] \sigma ?\text{asxy} \text{ using } T \text{ costyx by simp} \)

then show \( ?\text{thesis} \text{ unfolding } l \text{ unfolding } T_{\text{opt}\_\text{def}} \)
apply(rule cInf\_lower) by simp
qed (simp add: T\_opt\_def)

lemma \( \text{tt} : a \in \{x,y\} \implies \text{OPT2 } \text{rest1} \text{ (step [x,y] \text{ a (hd (OPT2 (a \# rest1) [x, y])})} \)
= \( \text{tl (OPT2 (a \# rest1) [x, y])} \)
apply(cases rest1) by(auto simp add: step\_def mtf2\_def swap\_def)

lemma \( \text{splitqsallg} : \text{Strat} \neq [] \implies a \in \{x,y\} \implies \)
\( t_p [x, y] \text{ a (hd (Strat))} + \)
\( \text{let } \text{L=step [x,y] a (hd (Strat))} \)
\( \text{ in } T_p \text{ L (rest1) (tl Strat)} = T_p [x, y] \text{ a (rest1) Strat} \)

proof –
assume \( \text{ne} : \text{Strat} \neq [] \)
assume \( \text{axy} : a \in \{x,y\} \)

n \( T_p [x, y] \text{ (a \# rest1) (Strat)} \)
\( = T_p [x, y] \text{ (a \# rest1) (hd (Strat)) (a \# rest1) (tl (Strat))} \)
by(simp only: List.list.collapse[OF ne])
then show \( ?\text{thesis} \text{ by auto} \)
qed

lemma \( \text{splitqs} : a \in \{x,y\} \implies T_p [x, y] \text{ (a \# rest1) (OPT2 (a \# rest1) [x, y])} \)
\( = t_p [x, y] \text{ a (hd (OPT2 (a \# rest1) [x, y])} + \)
(let \( L = \text{step} [x,y] \ a \ (\text{hd} \ (\text{OPT2} \ (a \ # \ \text{rest1}) \ [x,y])) \)
\[ \text{in} \ T_p \ L \ (\text{rest1}) \ (\text{OPT2} \ (\text{rest1}) \ L) \]

proof –

assume \( \text{axy} : a \in \{x,y\} \)

have \( \text{ne} : \text{OPT2} \ (a \ # \ \text{rest1}) \ [x,y] \neq [] \) apply(cases \text{rest1}) by(simp_all)

have \( T_p \ [x,y] \ (a \ # \ \text{rest1}) \ (\text{OPT2} \ (a \ # \ \text{rest1}) \ [x,y]) \)
\[ = T_p \ [x,y] \ (a \ # \ \text{rest1}) \ ((\text{hd} \ (\text{OPT2} \ (a \ # \ \text{rest1}) \ [x,y])) \ # \ (\text{tl} \ (\text{OPT2} \ (a \ # \ \text{rest1}) \ [x,y]))) \]

by(simp only: List.list.collapse[OF \text{ne}])

also have \( \ldots = T_p \ [x,y] \ (a \ # \ \text{rest1}) \ ((\text{hd} \ (\text{OPT2} \ (a \ # \ \text{rest1}) \ [x,y])) \ # \ (\text{OPT2} \ (\text{rest1}) \ (\text{step} [x,y] \ a \ (\text{hd} \ (\text{OPT2} \ (a \ # \ \text{rest1}) \ [x,y])))))) \)

by(simp only: \text{tl}[OF \text{axy}])

also have \( \ldots = \) \( T_p \ [x,y] \ a \ (\text{hd} \ (\text{OPT2} \ (a \ # \ \text{rest1}) \ [x,y])) + \)
\[ (\text{let} \ L = \text{step} [x,y] \ a \ (\text{hd} \ (\text{OPT2} \ (a \ # \ \text{rest1}) \ [x,y])) \]
\[ \text{in} \ T_p \ L \ (\text{rest1}) \ (\text{OPT2} \ (\text{rest1}) \ L) \) by(simp)

finally show \( ?\text{thesis} \).

qed

lemma \( \text{tpx} : T_p \ [x,y] \ x \ (\text{hd} \ (\text{OPT2} \ (x \ # \ \text{rest1}) \ [x,y]))) = 0 \)

by (simp add: \text{OPT2}x \text{tpx_def})

lemma \( \text{yup} : T_p \ [x,y] \ (x \ # \ \text{rest1}) \ (\text{OPT2} \ (x \ # \ \text{rest1}) \ [x,y]) \)
\[ = (\text{let} \ L = \text{step} [x,y] \ x \ (\text{hd} \ (\text{OPT2} \ (x \ # \ \text{rest1}) \ [x,y])) \]
\[ \text{in} \ T_p \ L \ (\text{rest1}) \ (\text{OPT2} \ (\text{rest1}) \ L) \]

by (simp add: \text{splitqs} \text{tpx})

lemma \( \text{swapsxy} : A \in \{ [x,y], [y,x] \} \implies \text{swaps} \ 	ext{sws} \) \( \text{ws} \) \( A \in \{ [x,y], [y,x] \} \)

apply(induct \text{sws})

apply(simp)

apply(simp) unfolding \text{swap_def} by auto

lemma \( \text{mtf2xy} : A \in \{ [x,y], [y,x] \} \implies r \in \{x,y\} \implies \text{mtf2} \ a \ r \ A \in \{ [x,y], [y,x] \} \)

by (metis \text{mtf2_def} \text{swapsxy})

lemma \( \text{stepxy} : \) assumes \( q \in \{x,y\} \) \( A \in \{ [x,y], [y,x] \} \)

shows \( \text{step} \ A \ q \ a \in \{ [x,y], [y,x] \} \)

unfolding \( \text{step_def} \) apply(simp only: \text{split_def} \text{Let_def})

apply(rule \text{mtf2xy})

apply(rule \text{swapszy}) by fact+
12.2 Proof of Optimality

**Lemma** OPT2 ls_lb: set \( \sigma \subseteq \{x, y\} \implies x \neq y \implies T_p [x, y] \sigma (OPT2 \sigma [x, y]) \leq T_{p-opt} [x, y] \sigma \)

**Proof** (induct length \( \sigma \) arbitrary; \( x \ y \sigma \) rule: less_induct)

case (less)

show ?case

proof (cases \( \sigma \))

case (Cons a \( \sigma' \))

note Cons1 = Cons

show ?thesis unfolding Cons
proof (cases a = x)

case True

from True Cons have qsform: \( \sigma = x \# \sigma' \) by auto

have up: \( T_p [x, y] (x \# \sigma') (OPT2 (x \# \sigma') [x, y]) \leq T_{p-opt} [x, y] (x \# \sigma') \)

unfolding True

unfolding T_opt_def apply (rule cInf_greater)

apply (simp add: Ex_list_of_length)

proof –

fix el

assume el \( \in \{T_p [x, y] (x \# \sigma') \ as \ length \ as = \ length (x \# \sigma')\} \)

then obtain Strat where IStrat: length Strat = length (x \# \sigma')

and el: \( T_p [x, y] (x \# \sigma') Strat = el \) by auto

then have ne: Strat \( \neq [] \) by auto

let \( \gamma LA = \text{step} [x, y] x (hd (OPT2 (x \# \sigma') [x, y])) \)

have E0: \( T_p [x, y] (x \# \sigma') (OPT2 (x \# \sigma') [x, y]) = T_p \gamma LA (\sigma') (OPT2 (\sigma') \gamma LA) \) using yup by auto

also have E1: \( \ldots = T_p [x, y] (\sigma') (OPT2 (\sigma') [x, y]) \) by (simp add: OPT2x step_def)

also have E2: \( \ldots \leq T_{p-opt} [x, y] \sigma' \) apply (rule less (1)) using Cons less (2, 3) by auto

also have \( \ldots \leq T_p [x, y] (x \# \sigma') Strat \)

proof (cases \( \text{step} [x, y] x (hd Strat) = [x, y] \))

case True

have aha: \( T_{p-opt} [x, y] \sigma' \leq T_p [x, y] \sigma' (tl Strat) \)

unfolding T_opt_def apply (rule cInf_lower)

apply (auto) apply (rule exI [where \( x = \text{tl Strat} \)) using IStrat by auto

also have E4: \( \ldots \leq t_p [x, y] x (hd Strat) + T_p (\text{step} [x, y] (hd Strat)) \)
y|\ x \ (hd \ Strat)) \ \sigma' \ (tl \ Strat)

unfolding \ True \ by(simp)
also have E5: \ \ldots \ = \ T_p \ [x, \ y] \ (x \ # \ \sigma') \ Strat \ using
splitqsally[of \ Strat \ x \ x' \ \sigma', \ OF \ ne, \ simplified]
by \ (auto)
finally show \ ?thesis \ by \ auto

next
case False
have tp1: \ T_p \ [x, \ y] \ x \ (hd \ Strat) \ \geq \ 1
proof \ (rule \ ccontr)
let \ ?a = \ hd \ Strat
assume \ \neg \ 1 \ \leq \ T_p \ [x, \ y] \ x \ ?a
then have \ tp0: \ T_p \ [x, \ y] \ x \ ?a = \ 0 \ by \ auto
then have \ size \ (snd \ ?a) = \ 0 \ unfolding \ t_p \ def \ by(simp
add: \ split_def)
then have \ nopaid: \ (snd \ ?a) = \ [] \ by \ auto
have \ step \ [x, \ y] \ x \ ?a = \ [x, \ y]
unfolding \ step_def \ apply(simp \ add: \ split_def \ nopaid)
unfolding \ mtf2_def \ by(simp)
then show \ False \ using \ False \ by \ auto
qed

from \ False \ have \ yx: \ step \ [x, \ y] \ x \ (hd \ Strat) = \ [y, \ x]
using \ stepxy[where \ x=x \ and \ y=y \ and \ a=hd \ Strat]
by \ auto

have \ E3: \ T_{p-opt} \ [x, \ y] \ \sigma' \ \leq \ 1 \ + \ T_{p-opt} \ [y, \ x] \ \sigma' \ using
swapOpt \ by \ auto
also have \ E4: \ \ldots \ \leq \ 1 \ + \ T_p \ [y, \ x] \ \sigma' \ (tl \ Strat)
apply(simp) \ unfolding \ T_{opt-def} \ apply(rule \ cInf_lower)
apply(auto) \ apply(rule \ exI[where \ x=tl \ Strat]) \ using
lStrat \ by \ auto
also have \ E5: \ \ldots \ = \ 1 \ + \ T_p \ (step \ [x, \ y] \ x \ (hd \ Strat)) \ \sigma'
(tl \ Strat) \ using \ yx \ by \ auto
also have \ E6: \ \ldots \ \leq \ T_p \ [x, \ y] \ x \ (hd \ Strat) + \ T_p \ (step \ [x, \ y] \ x \ (hd \ Strat)) \ \sigma' \ (tl \ Strat) \ using \ tp1 \ by \ auto

also have \ E7: \ \ldots \ = \ T_p \ [x, \ y] \ (x \ # \ \sigma') \ Strat \ using
splitqsally[of \ Strat \ x \ x' \ \sigma', \ OF \ ne, \ simplified]
by \ (auto)
finally show \ ?thesis \ by \ auto
qed
also have \ \ldots \ = \ el \ using \ True \ el \ by \ simp
finally show \ T_p \ [x, \ y] \ (x \ # \ \sigma') \ (OPT2 \ (x \ # \ \sigma') \ [x, \ y]) \ \leq \ el

159
by auto

qed

then show \( T_p [x, y] (a \# \sigma') (OPT2 (a \# \sigma') [x, y]) \leq T_{p-opt} [x, y] (a \# \sigma') \)

using True by simp

next

case False

with less Cons have ay: \( a = y \) by auto

show \( T_p [x, y] (a \# \sigma') (OPT2 (a \# \sigma') [x, y]) \leq T_{p-opt} [x, y] (a \# \sigma') \)

unfolding ay

proof (cases \( \sigma' \))

case Nil

have up: \( T_{p-opt} [x, y] [y] \geq 1 \)

unfolding T_opt_def apply (rule cInf_greatest)

apply (simp add: Ex_list_of_length)

proof -

fix el

assume el \( \in \{ T_p [x, y] [y] \text{ as } |as. \text{ length as } = \text{ length } [y]| \} \)

then obtain Strat where Strat: length Strat = length [y] and

el: \( el = T_p [x, y] [y] \text{ Strat by auto} \)

from Strat obtain a where a: Strat = [a] by (metis Suc_length_conv length_0_conv)

show 1 \leq el unfolding el a apply (simp) unfolding t_p_def

apply (simp add: split_def)

apply (cases snd a)

apply (simp add: less(3))

by (simp)

qed

show \( T_p [x, y] (y \# \sigma') (OPT2 (y \# \sigma') [x, y]) \leq T_{p-opt} [x, y] (y \# \sigma') \)

unfolding Nil

apply (simp add: t_p_def) using less(3) apply (simp)

using up by (simp)

next

case (Cons b rest2)

show up: \( T_p [x, y] (y \# \sigma') (OPT2 (y \# \sigma') [x, y]) \leq T_{p-opt} [x, y] (y \# \sigma') \)

unfolding Cons

proof (cases b=x)

case True
\[
\text{show } T_p \ [x, y] (y \# b \# \text{rest2}) (\text{OPT2} (y \# b \# \text{rest2}) [x, y]) \\
\leq T_{p\text{-opt}} \ [x, y] (y \# b \# \text{rest2})
\]

unfolding True

unfolding \( T_{opt\text{\_def}} \) \( \text{apply} (\text{rule cInf\_greatest}) \)

apply (simp add: Ex_list_of_length)

proof –

fix \( el \)

assume \( el \in \{ T_p \ [x, y] (y \# x \# \text{rest2}) \text{ as as. length as = length (y \# x \# \text{rest2})} \}

then obtain Strat where lenStrat: length Strat = length (y \# x \# \text{rest2}) and

\[
\text{Strat}: el = T_p \ [x, y] (y \# x \# \text{rest2}) \text{ Strat by auto}
\]

have \( v \): \( \text{set rest2} \subseteq \{ x, y \} \text{ using less(2)unfolded Cons1 Cons} \) by auto

let \( ?L1 = (\text{step} [x, y] y (\text{hd Strat})) \)
let \( ?L2 = (\text{step} ?L1 x (\text{hd (tl Strat)})) \)

have \( ?a1 = \text{hd Strat} \)
let \( ?a2 = \text{hd (tl Strat)} \)
let \( ?r = \text{tl (tl Strat)} \)

have Strat = ?a1 \# ?a2 \# ?r by (metis Nitpick.size_list_simp(2) Suc_length_conv lenStrat list.collapse list.discI list.inject)

have 1: \( T_p \ [x, y] (y \# x \# \text{rest2}) \text{ Strat} \)
\[
= t_p \ [x, y] y (\text{hd Strat}) + t_p ?L1 x (\text{hd (tl Strat)}) + T_p ?L2 \text{ rest2 (tl (tl Strat))}
\]

proof –

have a: Strat \( \neq [] \) using lenStrat by auto

have b: (tl Strat) \( \neq [] \) using lenStrat by (metis Nitpick.size_list_simp(2) Suc_length_conv list.discI list.inject)

have 1: \( T_p \ [x, y] (y \# x \# \text{rest2}) \text{ Strat} \)
\[
= t_p \ [x, y] y (\text{hd Strat}) + T_p ?L1 \ (x \# \text{rest2}) (tl Strat)
\]

using splitqally[OF a, where a=y and x=x and y=y, simplified] by (simp)

have tt: \( \text{step} [x, y] y (\text{hd Strat}) \neq [x, y] \implies \text{step} [x, y] y (\text{hd Strat}) = [y,x] \)
using 

have 2: \( T_p \ |L1 \ (x \# \ rest2) \ (tl \ Strat)) \quad = \quad T_p \ |L1 \ x \ (hd \ (tl \ Strat)) \quad + \quad T_p \ |L2 \ (rest2) \ (tl \ (tl \ Strat)) \)

apply\(\text{cases } |L1=\langle x,y \rangle\) 
using splitqsally\(\text{[OF b, where } a=x \text{ and } x=x\text{ and } y=y, \text{ simplified]} \) apply\(\text{auto}\)
using tt splitqsally\(\text{[OF b, where } a=x \text{ and } x=y \text{ and } y=x, \text{ simplified]} \) by \text{auto}
from \(1 \ 2 \text{ show } \text{thesis by auto}\)
qed

have \( T_p \ [x, y] \ (y \# x \# rest2) \ (OPT2 \ (y \# x \# rest2)) \)

unfolding \text{True}\nusing less(3) by(simp add: t_p_def step_def OPT2x)
also have \( \ldots \leq 1 + T_{p-opt} \ [x, y] \ (rest2) \ \text{apply}(simp) \)
apply\(\text{rule less(1)}\)
apply\(\text{simp add: less(2) Cons1 Cons} \) 
apply\(\text{fact} \) by \text{fact}
also

have \( \ldots \leq T_p \ [x, y] \ (y \# x \# rest2) \ Strat \)

proof \(\text{(cases } |L2 = \langle x,y \rangle)\)
\text{case True}\n
have 2: \( t_p \ [x, y] \ y \ (hd \ Strat) \quad + \quad t_p \ |L1 \ x \ (hd \ (tl \ Strat)) \quad + \quad T_p \ [x,y] \ rest2 \ (tl \ (tl \ Strat)) \quad \geq \quad t_p \ [x, y] \ y \ (hd \ Strat) \quad + \quad t_p \ |L1 \ x \ (hd \ (tl \ Strat)) \quad + \quad T_p \opt \ [x,y] \ rest2 \)
unfolding \text{T_opt_def apply}(rule cInf_lower) 
apply\(\text{simp} \) apply\(\text{rule exI[where x=tl (tl Strat)]} \)
by \(\text{auto simp: lenStrat} \)

have 3: \( t_p \ [x, y] \ y \ (hd \ Strat) \quad + \quad t_p \ |L1 \ x \ (hd \ (tl \ Strat)) \quad + \quad T_{p-opt} \ [x,y] \ rest2 \ \geq \ 1 + T_{p-opt} \ [x,y] \ rest2 \)
apply\(\text{simp} \)

\text{proof –}\n
have \( t_p \ [x, y] \ y \ (hd \ Strat) \quad \geq \quad 1 \)
unfolding \text{t_p_def apply}(simp add: split_def) 
apply\(\text{cases snd (hd Strat)} \) by \(\text{simp_all add: less(3)}\)
then show \( Suc \ 0 \leq t_p \ [x, y] \ y \ (hd \ Strat) \quad + \quad t_p \ |L1 \ x \ (hd \ (tl \ Strat)) \ \text{by auto} \)
qed
from 1 2 3 True show ?thesis by auto
next
case False
note L2F=this
have L1: ?L1 ∈ {[x, y], [y, x]} apply(rule stepxy) by simp_all
have ?L2 ∈ {x, y} apply(rule stepxy) using L1
by simp_all
with False have 2: ?L2 = [y,x] by auto
have k: Tp [x, y] (y # x # rest2) Strat
= t_p [x, y] y (hd Strat) + t_p ?L1 x (hd (tl Strat)) + T_p [y,x] rest2 (tl (tl Strat)) using 1 2 by auto
have l: t_p [x, y] y (hd Strat) > 0
using less(3) unfolding t_p def apply(cases snd (hd Strat) = [])
by (simp_all add: split_def)
have r: T_p [x, y] (y # x # rest2) Strat ≥ 2 + T_p [y,x] rest2 (tl (tl Strat))
proof (cases ?L1 = [x,y])
  case True
  case T=this
  then have t_p ?L1 x (hd (tl Strat)) > 0 unfolding True
  proof (cases snd (hd (tl Strat)) = [])
    case True
    have ?L2 = [x,y] unfolding T apply(simp add: split_def step_def)
    unfolding True mtf2_def by(simp)
    with L2F have False by auto
    then show 0 < t_p [x, y] x (hd (tl Strat)) ..
  next
  case False
  then show 0 < t_p [x, y] x (hd (tl Strat))
  unfolding t_p def by(simp add: split_def)
  qed
  with l have t_p [x, y] y (hd Strat) + t_p ?L1 x (hd (tl Strat)) ≥ 2 by auto
  with k show ?thesis by auto
next
  case False
  from L1 False have 2: ?L1 = [y,x] by auto
  { fix k sws T

163
have \( T \in \{ [x,y],[y,x] \} \implies \text{mtf2} \) k x T = [y,x] \implies T = [y,x]

apply (rule ccontr) by (simp add: less(3) mtf2_def)

apply (simp add: split_def)

apply (cases (snd (hd Strat))) using (x \neq y) by auto

have \( t_2 : t_p \ [x, \ y ] \ ( \text{hd} \ ( \text{tl} \ Strat)) \geq 1 \) unfolding \( t_p \_\text{def} \)

apply (simp add: split_def)

apply (cases (snd (hd (tl Strat)))) using (x \neq y) by auto

have \( T_p \ [x, y] \ (y \ # \ x \ # \ \text{rest2} \) \ Strat

= \( t_p \ [x, \ y ] \ ( \text{hd} \ Strat) + t_p \ ( \text{step} [x, \ y] \ ( \text{hd} \ Strat)) \)

\( x \ ( \text{hd} \ ( \text{tl} \ Strat)) \) + \( T_p \ [y, x] \ \text{rest2} \ ( \text{tl} \ ( \text{tl} \ Strat)) \)

by (rule k)

with \( t_1 \ t_2 \ 2 \) show \( \text{thesis} \) by auto

qed

have \( t : T_p \ [y, x] \ \text{rest2} \ ( \text{tl} \ ( \text{tl} \ Strat)) \geq T_{\text{opt}} \ [y, x] \ \text{rest2} \)

unfolding \( T_{\text{opt}} \_\text{def} \) apply (rule cInf_lower)

apply (auto) apply (rule exI[where \( x = (\text{tl} \ ( \text{tl} \ Strat))])

by (simp add: lenStrat)

show \( \text{thesis} \)

proof –

have \( 1 + T_{\text{opt}} \ [x, y] \ \text{rest2} \leq 2 + T_{\text{opt}} \ [x, x] \ \text{rest2} \)

using swapOpt by auto

also have \( \ldots \leq 2 + T_p \ [y, x] \ \text{rest2} \ ( \text{tl} \ ( \text{tl} \ Strat)) \) using \( t \) by auto

also have \( \ldots \leq T_p \ [x, y] \ (y \ # \ x \ # \ \text{rest2} \) \ Strat \) using \( r \)

by auto

finally show \( \text{thesis} \).

qed

qed

also have \( \ldots = \text{el} \) using Strat by auto

finally show \( T_p \ [x, y] \ (y \ # \ x \ # \ \text{rest2} \) \ (OPT2 \ (y \ # \ x \ # \ \text{rest2}) \ [x, y]) \leq \text{el} \).

qed

def

case False

with Cons1 Cons less(2) have \( \text{bisy} : b = y \) by auto

with less(3) have \( \text{OPT2} \ (y \ # \ b \ # \ \text{rest2}) \ [x, y] = (1,[])# (OPT2 \ (b\#\text{rest2}) \ [y,x]) \) by simp

show \( T_p \ [x, y] \ (y \ # \ b \ # \ \text{rest2}) \ (OPT2 \ (y \ # \ b \ # \ \text{rest2}) \ [x, y]) 

164
\[ T_{p-opt} [x, y] (y \# b \# rest2) \]

unfolding bisy

unfolding \( T_{opt_def} \) apply (rule cInf_greatest)

apply (simp add: Ex_list_of_length)

proof -

fix \( el \)

assume \( el \in \{ T_p [x, y] (y \# y \# rest2) \ as \ as. \ length \ as = length (y \# y \# rest2) \} \)

then obtain \( Strat \) where lenStrat: \( length \ Strat = length (y \# y \# rest2) \) and

\( Strat: el = T_p [x, y] (y \# y \# rest2) Strat \) by auto

have \( v: set \ rest2 \subseteq \{ x, y \} \) using \( \text{less}(2) \)|unfolded Cons1

proof

let \(?L1 = (step [x, y] y (hd Strat))\)

let \(?L2 = (step ?L1 y (hd (tl Strat)))\)

let \(?a1 = hd Strat\)

let \(?a2 = hd (tl Strat)\)

let \(?r = tl (tl Strat)\)

have \( Strat = ?a1 \# ?a2 \# ?r \) by \( \text{metis \ Nitpick.size_list_simp}(2) \)

Suc_length_conv lenStrat list-collapse list.discI list.inject)

have \( 1: T_p [x, y] (y \# y \# rest2) Strat \)

= \( t_p [x, y] y (hd Strat) + t_p ?L1 y (hd (tl Strat)) \)

+ \( T_p ?L2 rest2 (tl (tl Strat)) \)

proof -

have \( a: Strat \neq [] \) using lenStrat by auto

have \( b: (tl Strat) \neq [] \) using lenStrat by \( \text{metis \ Nitpick.size_list_simp}(2) \)

Suc_length_conv list-collapse list.discI list.inject)

have \( 1: T_p [x, y] (y \# y \# rest2) Strat \)

= \( t_p [x, y] y (hd Strat) + T_p ?L1 (y \# rest2) (tl Strat) \)

using splitqsally [OF \( a, where \ a=y \) and \( x=x \)]

and \( y=y, \) simplified] by \( \text{simp} \)

have \( t: \) step \( [x, y] y (hd Strat) \neq [x, y] \implies \) step \( [x, y] y (hd Strat) = [y, x] \)

using stepxy [where \( A=[x,y] \)] by blast
have 2: \( T_p \ ?L1 (y \ # \ rest2) (tl \ Strat) = t_p \ ?L1 y (hd \ (tl \ Strat)) + T_p \ ?L2 (rest2) (tl \ (tl \ Strat)) \)
apply(cases \( ?L1=[x,y] \))
using splitqssally[OF \( b \), where \( a=y \) and \( x=x \)
and \( y=y \), simplified] apply(auto)
using tt splitqssally[OF \( b \), where \( a=y \) and \( x=y \) and \( y=x \), simplified] by auto

from 1 2 show \( \text{thesis} \) by auto
qed

have \( T_p \ [x, y] (y \ # \ y \ # \ rest2) \ (OPT2 (y \ # \ y \ # \ rest2) [x, y]) \)
= \( 1 + T_p \ [y, x] \ (rest2) \ (OPT2 \ (rest2) [y, x]) \)

using less(3) by(simp add: \( t_p\_def \ step \_def \ mtf2 \_def \ swap \_def \)

\( OPT2x \)
also have \( \ldots \leq 1 + T_{p\_opt} \ [y, x] \ (rest2) \) apply(simp)
apply(rule less(1))
apply(simp add: less(2) Cons1 Cons)
using v less(3) by(auto)
also

have \( \ldots \leq T_p \ [x, y] (y \ # \ y \ # \ rest2) \ Strat \)
proof (cases \( ?L2 = [y, x] \))
case True
have 2: \( t_p \ [x, y] y (hd \ Strat) + t_p \ ?L1 y (hd \ (tl \ Strat)) + T_p \ [y, x] \ rest2 \ (tl \ (tl \ Strat)) \geq t_p \ [x, y] y (hd \ Strat) + t_p \ ?L1 y (hd \ (tl \ Strat)) + T_{p\_opt} \ [y, x] \ rest2 \ (rest2) \)
apply(simp)
unfolding \( T_{opt\_def} \) apply(rule cInf\_lower)
apply(simp) apply(rule exI[where \( x=tl \ (tl \ Strat) \)])
by (auto simp: lenStrat)

have 3: \( t_p \ [x, y] y (hd \ Strat) + t_p \ ?L1 y (hd \ (tl \ Strat)) + T_{p\_opt} \ [y, x] \ rest2 \geq 1 + T_{p\_opt} \ [y, x] \ rest2 \ (rest2) \)
apply(simp)

proof –
have \( t_p \ [x, y] y (hd \ Strat) \geq 1 \)
unfolding \( t_p\_def \) apply(simp add: split\_def)
apply(cases snd (hd \ Strat)) by (simp\_all add: less(3))
then show \( Suc \ 0 \leq t_p \ [x, y] y (hd \ Strat) + t_p \ ?L1 y (hd \ (tl \ Strat)) \) by auto
qed

from 1 2 3 True show \( \text{thesis} \) by auto

166
next
  case False
  note L2F=this
  have L1: ?L1 ∈ \{[x, y], [y, x]\} apply(rule stepxy) by simp_all

by simp_all
  have ?L2 ∈ \{[x, y], [y, x]\} apply(rule stepxy) using L1
with False have 2: ?L2 = [x,y] by auto

have k: T_p [x, y] (y ≠ y ≠ rest2) Strat
  = t_p [x, y] y (hd Strat) + t_p ?L1 y (hd (tl Strat)) + T_p [x,y] rest2 (tl (tl Strat)) using 1 2 by auto

have l: t_p [x, y] y (hd Strat) > 0
  using less(3) unfolding t_p_def apply(cases snd (hd Strat) = [])
  by (simp_all add: split_def)

have r: T_p [x, y] (y ≠ y ≠ rest2) Strat ≥ 2 + T_p [x,y] rest2 (tl (tl Strat))

proof (cases ?L1 = [y,x])
  case False
  from L1 False have ?L1 = [x,y] by auto
  note T=this
  then have t_p ?L1 y (hd (tl Strat)) > 0 unfolding T
  unfolding t_p_def apply(simp add: split_def)
  apply(cases snd (hd (tl Strat)) = [])
  using |x ≠ y| by auto
  with l k show ?thesis by auto
next

  case True
  note T=this

  have t_p ?L1 y (hd (tl Strat)) > 0 unfolding T
  proof(cases snd (hd (tl Strat)) = [])
    case True
    have ?L2 = [y,x] unfolding T apply(simp add: split_def step_def)
    unfolding True mtf2_def by(simp)
    with L2F have False by auto
    then show 0 < t_p [y, x] y (hd (tl Strat)) ..
next
  case False
then show \( 0 < t_p[y, x] y (hd (tl Strat)) \)

unfolding \( t_p \_ def \) by (simp add: split \_ def)

qed

with \( l \) have \( t_p[x, y] y (hd Strat) + t_p \ ?L1 y (hd (tl Strat)) \geq 2 \) by auto

with \( k \) show \( \?thesis \) by auto

qed

have \( t: T_p[x, y] rest2 (tl (tl Strat)) \geq T_p \_ opt[x, y] rest2 \)

unfolding \( T_\_ opt \_ def \) apply (rule cInf \_ lower)

apply (auto) apply (rule exI [where \( x = (tl (tl Strat)) \)])

by (simp add: lenStrat)

show \( \?thesis \)

proof –

have \( 1 + T_p \_ opt[y, x] rest2 \leq 2 + T_p \_ opt[x, y] rest2 \)

using swapOpt by auto

also have \( \ldots \leq 2 + T_p[x, y] rest2 (tl (tl Strat)) \) using

\( t \) by auto

also have \( \ldots \leq T_p[x, y] (y \# y \# rest2) Strat \) using \( r \)

by auto

finally show \( \?thesis \).

qed

qed

also have \( \ldots = el \) using Strat by auto

finally show \( T_p[x, y] (y \# y \# rest2) (OPT2 (y \# y \# rest2)) [x, y]) \leq el \).

qed

qed

qed (simp add: T_\_ opt \_ def)

qed

lemma \( OPT2 \_ is \_ sub \): set \( qs \subseteq \{x, y\} \implies x \neq y \implies T_p[x, y] qs \ (OPT2 qs [x, y]) \geq T_p \_ opt[x, y] qs \)

unfolding \( T_\_ opt \_ def \) apply (rule cInf \_ lower)

apply (simp) apply (rule exI [where \( x = (OPT2 qs [x, y]) \)])

by (auto simp add: OPT2 \_ length)

lemma \( OPT2 \_ is \_ opt \): set \( qs \subseteq \{x, y\} \implies x \neq y \implies T_p[x, y] qs \ (OPT2 qs \)

168
\[ [x,y] = T_{p,\text{opt}} [x,y] \] 
by (simp add: OPT2_is_lb OPT2_is_ub antisym)

12.3 Performance on the four phase forms

**lemma** OPT2_A: assumes \( x \neq y \) \( qs \in \text{lang} (\text{seq} [\text{Plus} (\text{Atom} x) \text{ One, Atom} y, \text{Atom} y]) \)
shows \( T_p [x,y] \) \( qs (OPT2 qs [x,y]) = 1 \)

**proof** – 
from assms(2) obtain \( u v \) where \( qs = u @ v \) and \( u = [x] \lor u = [] \) and \( v = [y,y] \) by (auto simp: conc_def)
from \( u \) have \( \text{pref1}: T_p [x,y] (u @ v) (OPT2 (u @ v) [x,y]) = T_p [x,y] v \) (OPT2 v [x,y]) 
apply(cases \( u = [] \))
apply(simp)
by(simp add: OPT2x t_p_def step_def)

have \( \text{ende}: T_p [x,y] v (OPT2 v [x,y]) = 1 \) unfolding \( v \) using assms(1)
by(simp add: mtf2_def swap_def t_p_def step_def)

from \( \text{pref1} \) \( \text{ende} \) \( qs \) show \( ?\text{thesis} \) by auto
qed

**lemma** OPT2_A': assumes \( x \neq y \) \( qs \in \text{lang} (\text{seq} [\text{Plus} (\text{Atom} x) \text{ One, Atom} y, \text{Atom} y]) \)
shows \( \text{real} (T_p [x,y] \) \( qs (OPT2 qs [x,y])) = 1 \)
using OPT2_A[OF assms] by simp

**lemma** OPT2_B: assumes \( x \neq y \) \( qs = u @ v \) \( u = [] \lor u = [x] \) \( v \in \text{lang} (\text{seq} [\text{Times} (\text{Atom} y) (\text{Atom} x), \text{Star}(\text{Times} (\text{Atom} y) (\text{Atom} x)), \text{Atom} y, \text{Atom} y]) \)
shows \( T_p [x,y] \) \( qs (OPT2 qs [x,y]) = (\text{length} v \text{ div} 2) \)

**proof** – 
from assms(3) have \( \text{pref1}: T_p [x,y] (u @ v) (OPT2 (u @ v) [x,y]) = T_p [x,y] v \) (OPT2 v [x,y]) 
apply(cases \( u = [] \))
apply(simp)
by(simp add: OPT2x t_p_def step_def)

from assms(4) obtain \( a \) \( w \) where \( v = a @ w \) and \( a \in \text{lang} (\text{Times} (\text{Atom} y) (\text{Atom} x)) \) and \( w \in \text{lang} (\text{seq}[\text{Star}(\text{Times} (\text{Atom} y) (\text{Atom} x)), \text{Atom} y, \text{Atom} y]) \) by (auto)
from this(2) have \( aa: a = [y,x] \) by(simp add: conc_def)
from assms(1) this v have pref2: \( T_p \ [x,y] \ v \ (OPT2 \ v \ [x,y]) = 1 + T_p \ [x,y] \ w \ (OPT2 \ w \ [x,y]) \)
by(simp add: t_p_def step_def OPT2x)

from w obtain c d where w2: \( w = c \oplus d \) and \( c \in \text{lang} \ (\text{Star} \ (\text{Times} \ (\text{Atom} \ y) \ (\text{Atom} \ x))) \) and \( d \in \text{lang} \ (\text{Times} \ (\text{Atom} \ y) \ (\text{Atom} \ y)) \) by auto
then have dd: \( d = [y,y] \) by auto

from c[simplified] have star: \( T_p \ [x,y] \ (c \oplus d) \ (OPT2 \ (c \oplus d) \ [x,y]) = \)
(length c div 2) + \( T_p \ [x,y] \ d \ (OPT2 \ d \ [x,y]) \)
proof(induct c rule: star_induct)
case (append r s)
then have r: \( r = [y,x] \) by auto
then have \( T_p \ [x,y] \ ((r \oplus s) \oplus d) \ (OPT2 \ ((r \oplus s) \oplus d) \ [x,y]) = \)
\( T_p \ [x,y] \ ([y,x] \oplus (s \oplus d)) \ (OPT2 \ ([y,x] \oplus (s \oplus d)) \ [x,y]) \) by simp
also have \( \ldots = 1 + T_p \ [x,y] \ (s \oplus d) \ (OPT2 \ (s \oplus d) \ [x,y]) \)
using assms(1) by(simp add: t_p_def step_def OPT2x)
also have \( \ldots = 1 + \text{length} \ s \div 2 + T_p \ [x,y] \ d \ (OPT2 \ d \ [x,y]) \)
using append by simp
also have \( \ldots = \text{length} \ (r \oplus s) \div 2 + T_p \ [x,y] \ d \ (OPT2 \ d \ [x,y]) \)
using r by auto
finally show \ ?case .
qed simp

have ende: \( T_p \ [x,y] \ d \ (OPT2 \ d \ [x,y]) = 1 \) unfolding dd using assms(1)
by(simp add: mt2_def swap_def t_p_def step_def)

have vv: \( v = [y,x] \oplus c \oplus [y,y] \) using w2 dd v aa by auto

from pref1 pref2 star w2 ende have
\( T_p \ [x,y] \ qs \ (OPT2 \ qs \ [x,y]) = 1 + \text{length} \ c \div 2 + 1 \) unfolding assms(2) by auto
also have \( \ldots = (\text{length} \ v \div 2) \) using vv by auto
finally show \ ?thesis .
qed

lemma OPT2_B1: assumes \( x \neq y \) qs \( \in \text{lang} \ (\text{seq} \ [\text{Atom} \ y, \ \text{Atom} \ x, \ \text{Star} \ (\text{Times} \ (\text{Atom} \ y) \ (\text{Atom} \ x)), \ \text{Atom} \ y, \ \text{Atom} \ y]) \)
shows \( \text{real} \ (T_p \ [x,y] \ qs \ (OPT2 \ qs \ [x,y])) = \text{length} \ qs / 2 \)
proof
  from assms(2) have qs: \( qs \in \text{lang} \ (\text{seq} \ [\text{Times} \ (\text{Atom} \ y) \ (\text{Atom} \ x), \ \text{Star} \ (\text{Times} \ (\text{Atom} \ y) \ (\text{Atom} \ x)), \ \text{Atom} \ y, \ \text{Atom} \ y]) \)
  by(simp add: conc_assoc)

  170
have \((\text{length } \mathit{qs}) \mod 2 = 0\)

proof
- from \(\text{assms}(2)\) have \(\mathit{qs} \in (\{[\mathit{y}]\} \# \{[\mathit{x}]\} \# \star (\{[\mathit{y}]\} \# \{[\mathit{x}]\})\)
  \(\#\{[\mathit{y}]\} \# \{[\mathit{x}]\}\) by (simp add: \(\text{conc_assoc}\))
  then obtain \(p\) \(q\) \(r\) where \(\mathit{ppr} : \mathit{qs}=p@q@r\) and \(\mathit{p}\in(\{[\mathit{y}]\} \# \{[\mathit{x}]\})\)
    and \(\mathit{q} : \mathit{q} \in \star (\{[\mathit{y}]\} \# \{[\mathit{x}]\})\) and \(\mathit{r} \in\{[\mathit{y}]\} \# \{[\mathit{x}]\}\) by
    \((\text{metis concE})\)
    then have \(rr : p = [\mathit{y},\mathit{x}]\) \(r=[\mathit{y},\mathit{y}]\) by \(\text{auto}\)
  with \(\mathit{ppr}\) have \(a : \text{length } \mathit{qs} = 4 + \text{length } \mathit{q} \text{ by } \text{auto}\)
  from \(\mathit{q}\) have \(b : \text{length } \mathit{q} \mod 2 = 0\)
  apply(induct \(\mathit{q}\) rule: \(\star\_\text{induct}\)) by \(\text{auto}\)
  from \(\mathit{a}\) \(\mathit{b}\) show \(\#\text{thesis}\) by \(\text{auto}\)
qed

with \(\text{OPT2}_B[\text{where } u=[]], \text{OF assms}(1) \_\_ \mathit{qs}\) show \(\#\text{thesis}\) by \(\text{auto}\)

qed

lemma \(\text{OPT2}_B2\): assumes \(x \neq y \mathit{qs} \in \text{lang} (\text{seq}[\text{Atom } x, \text{Atom } y, \text{Atom } x, \star (\text{Times } (\text{Atom } y) (\text{Atom } x)), \text{Atom } y, \text{Atom } y)\)
shows \(\mathit{T}_p [x,y] \mathit{qs} (\text{OPT2 } \mathit{qs} [x,y]) = ((\text{length } \mathit{qs} - 1) / 2)\)
proof
- from \(\text{assms}(2)\) obtain \(v\) where
  \(\mathit{qsv} : \mathit{qs} = [\mathit{x}]@v\) and \(\mathit{vv} : v \in \text{lang} (\text{seq}[\text{Times } (\text{Atom } y) (\text{Atom } x), \star (\text{Times } (\text{Atom } y) (\text{Atom } x)), \text{Atom } y, \text{Atom } y])\) by
  \((\text{auto simp add: conc_def})\)
  have \(\text{(length } v) \mod 2 = 0\)
  proof
    - from \(\mathit{vv}\) have \(v \in (\{[\mathit{y}]\} \# \{[\mathit{x}]\} \# \star (\{[\mathit{y}]\} \# \{[\mathit{x}]\}) \# \{[\mathit{y}]\} \# \{[\mathit{y}]\}\) by
      \((\text{simp add: conc_def})\)
    then obtain \(p\) \(q\) \(r\) where \(\mathit{ppr} : v=p@q@r\) and \(p\in(\{[\mathit{y}]\} \# \{[\mathit{x}]\})\)
      and \(\mathit{q} : \mathit{q} \in \star (\{[\mathit{y}]\} \# \{[\mathit{x}]\})\) and \(\mathit{r} \in\{[\mathit{y}]\} \# \{[\mathit{x}]\}\) by
      \((\text{metis concE})\)
    then have \(rr : p = [\mathit{y},\mathit{x}]\) \(r=[\mathit{y},\mathit{y}]\) by \((\text{auto simp add: conc_def})\)
    with \(\mathit{ppr}\) have \(a : \text{length } v = 4 + \text{length } \mathit{q} \text{ by } \text{auto}\)
    from \(\mathit{q}\) have \(b : \text{length } \mathit{q} \mod 2 = 0\)
    apply(induct \(\mathit{q}\) rule: \(\star\_\text{induct}\)) by \(\text{auto}\)
    from \(\mathit{a}\) \(\mathit{b}\) show \(\#\text{thesis}\) by \(\text{auto}\)
qed

with \(\text{OPT2}_B[\text{where } u=[x], \text{OF assms}(1) \_\_ \mathit{qs\_v\_vv}] \text{qsv show } \#\text{thesis}\)
by\(\text{(auto)}\)

qed

lemma \(\text{OPT2}_C\): assumes \(x \neq y \mathit{qs}=u@v u=[] \lor u=[x]\)
  and \(v \in \text{lang} (\text{seq}[\text{Atom } y, \text{Atom } x, \star (\text{Times } (\text{Atom } y) (\text{Atom } x)), \text{Atom } x])\)

171
shows $T_p [x,y] \; QS \; (OPT2 \; QS \; [x,y]) = (\text{length} \; v \; \text{div} \; 2)$

proof –
from $\text{assms}(3)$ have prep1: $T_p [x,y] \; (u \@ v) \; (OPT2 \; (u \@ v) \; [x,y]) = T_p [x,y] \; v \; (OPT2 \; v \; [x,y])$
apply(cases $u=[]$)
apply(simp)
by(simp add: OPT2x $t_p$-def step_def)

from $\text{assms}(4)$ obtain $a \; w$ where $v = a @ w$ and $aa: a@[y,x]$ and $w: w @ lang \; (\text{seq} \; \text{Star} \; (\text{Times} \; (\text{Atom} \; y) \; (\text{Atom} \; x)), \; \text{Atom} \; x)$ by(auto simp: conc_def)

from $\text{assms}(1)$ this $v$ have prep2: $T_p [x,y] \; v \; (OPT2 \; v \; [x,y]) = 1 + T_p [x,y] \; w \; (OPT2 \; w \; [x,y])$
by(simp add: $t_p$-def step_def OPT2x)

from $w$ obtain $c \; d$ where $w2: w = c @ d$ and $c: c \in lang \; (\text{Star} \; (\text{Times} \; (\text{Atom} \; y) \; (\text{Atom} \; x)))$ and $d: d \in lang \; (\text{Atom} \; x)$ by auto
then have $dd: d = [x]$ by auto

from $c$[simplified] have star: $T_p [x,y] \; (c @ d) \; (OPT2 \; (c @ d) \; [x,y]) = (\text{length} \; c \; \text{div} \; 2) + T_p [x,y] \; d \; (OPT2 \; d \; [x,y]) \land (\text{length} \; c) \; \text{mod} \; 2 = 0$

proof(induct $c$ rule: star_induct)
case (append $r \; s$)
from append have mod: $\text{length} \; s \; \text{mod} \; 2 = 0$ by simp
from append have $r: r = [y,x]$ by auto
then have $T_p [x, y] \; ((r @ s) @ d) \; (OPT2 \; ((r @ s) @ d) \; [x, y]) = T_p [x, y] \; (\text{length} \; (r @ s) \; \text{div} \; 2) + T_p [x, y] \; (OPT2 \; (s @ d) \; [x, y])$ by simp
also have $\ldots = 1 + T_p [x, y] \; (s @ d) \; (OPT2 \; (s @ d) \; [x, y])$
using $\text{assms}(1)$ by(simp add: $t_p$-def step_def OPT2x)
also have $\ldots = 1 + \text{length} \; (r @ s) \; \text{div} \; 2 + T_p [x, y] \; d \; (OPT2 \; d \; [x, y])$
using append by simp
also have $\ldots = \text{length} \; (r @ s) \; \text{div} \; 2 + T_p [x, y] \; d \; (OPT2 \; d \; [x, y])$
using $r$ by auto
finally show ?case by(simp add: mod $r$)
qed simp

have ende: $T_p [x,y] \; d \; (OPT2 \; d \; [x,y]) = 0$ unfolding $dd$ using $\text{assms}(1)$
by(simp add: mtf2_def swap_def $t_p$-def step_def)

have vv: $v = [y,x]@[c@[x]]$ using $w2 \; dd \; v \; aa$ by auto

from prep1 prep2 star $w2$ ende have $T_p [x, y] \; QS \; (OPT2 \; QS \; [x, y]) = 1 + \text{length} \; c \; \text{div} \; 2$ unfolding $\text{assms}(2)$
by auto
also have \( \ldots = \text{(length } v \text{ div } 2) \) using \( vv \) star by auto
finally show \(!thesis \).
qed

lemma \( \text{OPT2}_C1 \): assumes \( x \neq y \), \( qs \in \text{lang} (\text{seq[Atom} y, \text{Atom} x, \text{Star(Times (Atom} y)) (\text{Atom} x)\text{]}), \text{Atom} x))\)
shows \( \text{real} (T_p [x,y] qs \ldots) = (\text{length } qs - 1)/2 \)
proof –
from assms(2) have \( qs \in \text{lang} (\text{seq[Atom} y, \text{Atom} x, \text{Star(Times (Atom} y)) (\text{Atom} x)\text{]}), \text{Atom} x))\)
by (simp add: conc_assoc)
have \( \text{length } qs \mod 2 = 1 \)
proof –
from assms(2) have \( qs \in ([y] \star [x]) \star [x] \)
by (simp add: conc_assoc)
then obtain \( p \), \( q \), \( r \) where \( \text{pqr: qs} = p \cdot q \cdot r \) and \( p \in ([y] \star [x]) \star [x] \)
and \( q \in \text{star} ([y] \star [x]) \) and \( r \in [x] \) by (metis concE)
then have \( \text{rr: } p = [y,x] r = [x] \) by auto
with \( \text{pqr have a: length } qs = 3\text{+length } q \) by auto
from \( q \) have \( b: \text{length } q \text{ mod } 2 = 0 \)
apply (induct \( q \) rule: star_induct) by (auto)
from \( ab \) show \(!thesis \) by auto
by (metis minus_mod_eq_div_mult [symmetric] of_nat_mult of_nat_numeral)
qed

lemma \( \text{OPT2}_C2 \): assumes \( x \neq y \), \( qs \in \text{lang} (\text{seq[Atom} x, \text{Atom} y, \text{Atom} x, \text{Star(Times (Atom} y)) (\text{Atom} x)\text{]}), \text{Atom} x))\)
shows \( T_p [x,y] qs \ldots = ((\text{length } qs - 2)/2) \)
proof –
from assms(2) obtain \( v \) where \( \text{qsv: qs} = [x] \cdot v \) and \( vv \cdot v \in \text{lang} (\text{seq[Atom} y, \text{Atom} x, \text{Star(Times (Atom} y)) (\text{Atom} x)\text{]}), \text{Atom} x))\)
by (auto simp add: conc_def)
have \( \text{length } v \text{ mod } 2 = 1 \)
proof –
from \( vv \) have \( v \in ([y] \star [x]) \star [x] \)
by (simp add: conc_assoc)
then obtain \( p \), \( q \), \( r \) where \( \text{pqr: v} = p \cdot q \cdot r \) and \( p \in ([y] \star [x]) \star [x] \)
and \( q \in \text{star} ([y] \star [x]) \) and \( r \in [x] \) by (metis concE)
then have \( \text{rr: } p = [y,x] r = [x] \) by (auto simp add: conc_def)
with \( pqr \) have \( a : \text{length } v = 3 + \text{length } q \) by auto
from \( q \) have \( b : \text{length } q \mod 2 = 0 \)
apply(induct \( q \) rule: star_induct) by (auto)
from \( a \) \( b \) show \(?\text{thesis}\) by auto
qed

with \( \text{OPT2}_C[\text{where } u=[x], \text{OF assms}(1) \] \( qsv \) \( vv \]) \( qsv \) show \(?\text{thesis}\)
apply(auto)
by (metis \( \text{minus}\_\text{mod}\_\text{eq}\_\text{div}\_\text{mult} \) [symmetric] \( \text{of\_nat\_mult} \) \( \text{of\_nat\_numeral} \))
qed

\begin{verbatim}
lemma OPT2_ub: set qs \subseteq \{x,y\} \implies T_p [x,y] qs (\text{OPT2} qs [x,y]) \leq \text{length } qs
proof(induct qs arbitrary: x y)
  case (Cons q qs)
  then have set qs \subseteq \{x,y\} q\in\{x,y\} by auto
  note Cons1=Cons this
  show ?case
    proof(cases qs)
    case Nil
    with Cons1 show \( T_p [x,y] (\text{OPT2} (q \# qs) [x,y]) \leq \text{length } (q \# qs) \)
      apply(simp add: t_p_def) by blast
    next
    case (Cons q' qs')
    with Cons1 have q'\in\{x,y\} by auto
    note Cons=Cons this

    from Cons1 have T: \( T_p [x,y] qs (\text{OPT2} qs [x,y]) \leq \text{length } qs \)
    \( T_p [y,x] qs (\text{OPT2} qs [y,x]) \leq \text{length } qs \) by auto
    show \( T_p [x,y] (q \# qs) (\text{OPT2} (q \# qs) [x,y]) \leq \text{length } (q \# qs) \)
      unfolding Cons apply(simp only: OPT2.simps)
      apply(split if_splits(1))
      apply(safe)
      proof (goal_cases)
        case 1
        have \( T_p [x,y] (x \# q' \# qs') ((0,[])) \# OPT2 (q' \# qs') [x,y] \)
          = \( t_p [x,y] x (0,[]) \) + \( T_p [x,y] qs (\text{OPT2} qs [x,y]) \)
          by(simp add: step_def Cons)
        also have \( \ldots \leq t_p [x,y] x (0,[]) + \text{length } qs \) using T by auto
        also have \( \ldots \leq \text{length } (x \# q' \# qs') \) using Cons by(simp add: t_p_def)
    qed
\end{verbatim}

174
finally show ?case.

next
case 2
with Cons1 Cons show ?case
  apply(split if_splits(1))
  apply(safe)
proof (goal_cases)
  case 1
  then have \( T_p [x, y] (y \# x \# qs') ((0, []]) \# OPT2 (x \# qs') [x, y] \)
    \( = t_p [x, y] y (0, []]) + T_p [x, y] qs (OPT2 qs [x, y]) \)
    by(simp add: step_def)
  also have \( \ldots \leq t_p [x, y] y (0, []]) + length qs \) using \( T \) by auto

  also have \( \ldots \leq length (y \# x \# qs') \) using Cons by(simp add: t_p_def)
  finally show ?case.

next
  case 2
  then have \( T_p [x, y] (y \# y \# qs') ((1, []]) \# OPT2 (y \# qs') [y, x] \)
    \( = t_p [x, y] y (1, []]) + T_p [y, x] qs (OPT2 qs [y, x]) \)
    by(simp add: step_def mtf2_def swap_def)
  also have \( \ldots \leq t_p [x, y] y (1, []]) + length qs \) using \( T \) by auto

  also have \( \ldots \leq length (y \# y \# qs') \) using Cons by(simp add: t_p_def)
  finally show ?case.
  qed
qed

lemma OPT2_padded: \( R \in \{[x,y],[y,x]\} \implies set qs \subseteq \{x,y\} \)
  \( \implies T_p R (qs@[x,x]) (OPT2 (qs@[x,x]) R) \)
  \( \leq T_p R (qs@[x]) (OPT2 (qs@[x]) R) + 1 \)
apply(induct qs arbitrary: R)
apply(simp)
  apply(case_tac R=[x,y])
  apply(simp add: step_def t_p_def)
  apply(simp add: step_def mtf2_def swap_def t_p_def)
proof (goal_cases)
  case (1 a qs R)
  then have a: \( a \in \{x,y\} \) by auto
with 1 show ?case
  apply (cases qs)
  apply (cases a = x)
  apply (simp add: step_def t_p_def)
  apply (simp add: step_def mtf2_def swap_def t_p_def)
  apply (cases R = [x, y])
  apply (simp add: step_def t_p_def)
  apply (simp add: step_def mtf2_def swap_def t_p_def)
proof (goal cases)
  case (1 p ps)
  show ?case
    apply (cases a = x)
    apply (cases R = [x, y])
    apply (simp add: OPT2 step_def)
    using 1 apply (simp)
    using 1 (2) apply (simp)
    apply (cases qs)
    apply (simp add: step_def mtf2_def swap_def t_p_def)
    using 1 by (auto simp add: swap_def mtf2_def step_def)
qed

lemma OPT2_split11:
  assumes xy : x # y
  shows R ∈ {[x, y], [y, x]} → set xs ⊆ {x, y} → set ys ⊆ {x, y} → OPT2
  (xs @ [x, x] @ ys) R = OPT2 (xs @ [x, x]) R @ OPT2 ys [x, y]
proof (induct xs arbitrary: R)
  case Nil
  then show ?case
    apply (simp)
  apply (cases ys)
  apply (simp)
  apply (cases R = [x, y])
  apply (simp)
  by (simp)
next
  case (Cons a as)
  note iH = this
  then have AS: set as ⊆ {x, y} and A: a ∈ {x, y} by auto
  note iH[Cons(1)] where R = [y, x], simplified, OF AS Cons(4)]
  note iH'[Cons(1)] where R = [x, y], simplified, OF AS Cons(4)]
  show ?case
  proof (cases R = [x, y])
case True
note $R=$this
from iH iH’ show ?thesis
apply(cases $a=x$
  apply(simp add: $R \ OPT2x$)
  using $A$ apply(simp)
  apply(cases as)
    apply(simp add: $R$
      using $AS$ apply(simp)
      apply(case_tac $aa=x$
        by(simp_all add: $R$
    )
  )
) next

case False
with Cons(2) have $R: R=[y,x]$ by auto
from iH iH’ show ?thesis
apply(cases $a=y$
  apply(simp add: $R \ OPT2x$)
  using $A$ apply(simp)
  apply(cases as)
    apply(simp add: $R$
      apply(case_tac $aa=y$
        by (simp_all add: $R$
      )
    )
  )
) qed

12.4 The function steps

lemma steps_append: length $qs = length as \implies steps s \ (qs@[q]) \ (as@[a])$
  = step (steps s qs as) q a
apply(induct qs as arbitrary: s rule: list_induct2) by simp_all

end

13 Phase Partitioning

theory Phase_Partitioning
imports OPT2
begin

13.1 Definition of Phases

definition other $a \ x \ y = (if \ a=x \ then \ y \ else \ x)$
definition \textit{Lxx} where
\[\text{Lxx} \ (x::\text{nat}) \ y = \text{lang} \ (L_{\text{lasthasxx}} \ x \ y)\]

lemma \textit{Lxx_not_nullable}: \[\not\in L_{\text{xx}} x \ y\]
unfolding \textit{Lxx_def} \textit{L_{lasthasxx_def}} by simp

lemma \textit{Lxx_ends_in_two_equal}: \[xs \in L_{\text{xx}} x \ y \implies \exists \text{pref} \ e. \ xs = \text{pref} @ [e,e]\]
by (auto simp: conc_def \textit{Lxx_def} \textit{L_{lasthasxx_def}})

lemma \textit{Lxx x y = Lxx y x unfolding Lxx_def} by (rule \textit{lastxx_com})

definition \textit{hideit} x y = (\text{Plus rexp.One (nodouble x y)})

lemma \textit{Lxx_othercase}: set \(qs \subseteq \{x,y\}\) \implies \(\exists \text{xs} \ ys. \ qs = \text{xs} @ \text{ys} \land \text{xs} \in L_{\text{xx}} x \ y\) \implies qs \in \text{lang} (\text{hideit} x y)

proof -
  assume set \(qs \subseteq \{x,y\}\)
  then have \(qs \in \text{lang} \ (\text{myUNIV} x y)\) using \textit{myUNIV_alle[of x y]} by blast
  then have \(qs \in \text{star} \ (\text{lang} \ (L_{\text{lasthasxx}} x y)) @@ \text{lang} \ (\text{hideit} x y)\)
  unfolding \textit{hideit_def}
  by (auto simp add: \textit{myUNIV_char})
  then have \(qs \in \text{lang} (\text{hideit} x y)\) by (simp add: \textit{Lxx_def})

assume notpos: \(\exists \text{xs} \ ys. \ qs = \text{xs} @ \text{ys} \land \text{xs} \in L_{\text{xx}} x \ y\)

proof -
  from qs obtain A B where qsAB: \(qs = A@B\) and A: \(A \in \text{star} \ (L_{\text{xx}} x y)\)
  and B: \(B \in \text{lang} \ (\text{hideit} x y)\) by auto
  with notpos have notin: \(A \notin (L_{\text{xx}} x y)\) by blast

from A have 1: \(A = [] \lor A \in (L_{\text{xx}} x y) @@ \text{star} \ (L_{\text{xx}} x y)\) using
\textit{Regular_Set.star_unfold_left} by auto

have 2: \(A \notin (L_{\text{xx}} x y) @@ \text{star} \ (L_{\text{xx}} x y)\)

proof (rule ccontr)
  assume \(\neg \ A \notin L_{\text{xx}} x y @@ \text{star} \ (L_{\text{xx}} x y)\)
  then have \(A \in L_{\text{xx}} x y @@ \text{star} \ (L_{\text{xx}} x y)\) by auto
  then obtain A1 A2 where A=A1@@A2 and A1: A1\(\in(L_{\text{xx}} x y)\) and A2\(\in \text{star} \ (L_{\text{xx}} x y)\) by auto
  with qsAB have qs=A1@@(A2@@B) A1\(\in(L_{\text{xx}} x y)\) by auto
  with notpos have A1 \(\notin (L_{\text{xx}} x y)\) by blast

178
with A1 show False by auto
qed
from 1 2 have A=[] by auto
with qsAB have qs=B by auto
with B show ?thesis by simp
qed

fun pad where pad xs x y = (if xs=[] then [x,x] else
 (if last xs = x then xs @ [x] else xs @ [y]))

lemma pad_adds2: qs ≠ [] ⟹ set qs ⊆ {x,y} ⟹ pad qs x y = qs @ [last qs]
apply (auto) by (metis insertE insert_absorb insert_not_empty last_in_set subset_iff)

lemma nodouble_padded: qs ≠ [] ⟹ qs ∈ lang (nodouble x y) ⟹ pad qs x y ∈ Lxx x y
proof –
  assume nn: qs ≠ []
  assume qs ∈ lang (nodouble x y)
  then have a: qs ∈ lang (seq
      [Plus (Atom x) rexp.One,
       Star (Times (Atom y) (Atom x)),
       Atom y]) ∨ qs ∈ lang
      (seq
       [Plus (Atom y) rexp.One,
       Star (Times (Atom x) (Atom y)),
       Atom x]) unfolding nodouble_def by auto

  show ?thesis
  proof (cases qs ∈ lang (seq [Plus (Atom x) One, Star (Times (Atom y)
 (Atom x))], Atom y)))
    case True
    then have qs ∈ lang (seq [Plus (Atom x) One, Star (Times (Atom y)
 (Atom x))]) @@[ {y}]
      by (simp add: conc_assoc)
    then have last qs = y by auto
      with nn have p: pad qs x y = qs @ [y] by auto
    have A: pad qs x y ∈ lang (seq [Plus (Atom x) One, Star (Times (Atom y)
 (Atom x))],}
Atom y) @@ \{[y]\} unfolding p
  apply(simp)
  apply(rule concI)
  using True by auto

have B: lang (seq [Plus (Atom x) One, Star (Times (Atom y) (Atom x))],
  Atom y) @@ \{[y]\} = lang (seq [Plus (Atom x) One, Star (Times (Atom y) (Atom x))],
  Atom y, Atom y)) by (simp add: conc_assoc)

show pad qs x y ∈ Lxx x y unfolding Lxx_def Llasthasxx_def
  using B A by auto

next
  case False
  with a have T: qs ∈ lang (seq [Plus (Atom y) One, Star (Times (Atom x) (Atom y)), Atom x]) by auto

then have qs ∈ lang (seq [Plus (Atom y) One, Star (Times (Atom y) (Atom x))]) @@ \{[x]\}
  by(simp add: conc_assoc)

then have last qs = x by auto

with nn have p: pad qs x y = qs @ [x] by auto

have A: pad qs x y ∈ lang (seq [Plus (Atom y) One, Star (Times (Atom x) (Atom y)),
  Atom x]) @@ \{[x]\} unfolding p
  apply(simp)
  apply(rule concI)
  using T by auto

have B: lang (seq [Plus (Atom y) One, Star (Times (Atom x) (Atom y))],
  Atom x) @@ \{[x]\} = lang (seq [Plus (Atom y) One, Star (Times (Atom x) (Atom y))],
  Atom x, Atom x)) by (simp add: conc_assoc)

show pad qs x y ∈ Lxx x y unfolding Lxx_def Llasthasxx_def
  using B A by auto

qed

thm UnE

lemma \(c \in A \cup B \implies P\)
  apply(erule UnE) oops

lemma LxxE: qs ∈ Lxx x y
  \(\implies (qs \in \text{lang} \ (\text{seq} \ [\text{Atom} \ x, \text{Atom} \ x])) \implies P \ x \ y \ qs)\)
  \(\implies (qs \in \text{lang} \ (\text{seq} \ [\text{Plus} \ (\text{Atom} \ x)] \ \text{rexp.} \text{One}, \text{Atom} \ y, \text{Atom} \ x, \text{Star})\)
\[(\text{Times} (\text{Atom } y) (\text{Atom } x)), \text{Atom } y, \text{Atom } y] \Rightarrow P x y qs\]
\[\Rightarrow (qs \in \text{lang} (\text{seq} [\text{Plus} (\text{Atom } x) \text{ reexp. One}, \text{Atom } y, \text{Atom } x, \text{Star} (\text{Times} (\text{Atom } y) (\text{Atom } x)), \text{Atom } y, \text{Atom } y]) \Rightarrow P x y qs\]
\[\Rightarrow (qs \in \text{lang} (\text{seq} [\text{Plus} (\text{Atom } x) \text{ reexp. One}, \text{Atom } y, \text{Atom } x]) \Rightarrow P x y qs)\]
\[\Rightarrow P x y qs\]

unfolding \text{Lxx_def} lastxx_is_4cases[symmetric] \text{L_4cases_def} apply(simp only: verund.simps lang.simps)

using \text{UnE} by blast

thm \text{UnE LxxE}

lemma \(qs \in Lxx x y \Rightarrow P\)
apply(erule \text{LxxE}) oops

lemma \text{LxxI}: \(qs \in \text{lang} (\text{seq} [\text{Atom } x, \text{Atom } x]) \Rightarrow P x y qs\)
\[\Rightarrow (qs \in \text{lang} (\text{seq} [\text{Plus} (\text{Atom } x) \text{ reexp. One}, \text{Atom } y, \text{Atom } x, \text{Star} (\text{Times} (\text{Atom } y) (\text{Atom } x)), \text{Atom } y, \text{Atom } y]) \Rightarrow P x y qs)\]
\[\Rightarrow (qs \in \text{lang} (\text{seq} [\text{Plus} (\text{Atom } x) \text{ reexp. One}, \text{Atom } y, \text{Atom } x]) \Rightarrow P x y qs)\]
\[\Rightarrow (qs \in \text{lang} (\text{seq} [\text{Plus} (\text{Atom } x) \text{ reexp. One}, \text{Atom } y, \text{Atom } x]) \Rightarrow P x y qs)\]
\[\Rightarrow (qs \in Lxx x y \Rightarrow P x y qs)\]

unfolding \text{Lxx_def} lastxx_is_4cases[symmetric] \text{L_4cases_def} apply(simp only: verund.simps lang.simps)

by blast

lemma \text{LxxI}: \(xs \in Lxx x y \Rightarrow \text{length } xs \geq 2\)
apply(rule \text{LxxI}[where \(P=(\lambda x y qs. \text{length } qs \geq 2)])\]
apply(auto) by(auto simp: cone_def)

13.2 OPT2 Splitting

lemma \text{ayay}: \(\text{length } qs = \text{length } as \Rightarrow T_p s (qs\[@[q]\] (as\[@[a]\]) = T_p s qs\)
as + t_p (steps s qs as) q a
apply(induct qs as arbitrary: s rule: list_induct2) by simp_all

lemma \text{tlofOPT2}: \(Q \in \{x,y\} \Rightarrow set QS \subseteq \{x,y\} \Rightarrow R \in \{[x, y], [y, x]\}\)
\[\Rightarrow tl (\text{OPT2} ((Q \# QS) @ [x, x]) R) = \text{OPT2} (QS @ [x, x]) (\text{step } R Q (\text{hd (OPT2 ((Q \# QS) @ [x, x]) R))))\]
apply(cases \(Q=x)\)
apply(cases \(R=[x,y])\)
apply(simp add: OPT2step_def)
apply (simp)
apply (cases QS)
  apply (simp add: step_def mtf2_def swap_def)
  apply (simp add: step_def mtf2_def swap_def)
apply (cases R = [x, y])
apply (simp)
apply (cases QS)
  apply (simp add: step_def mtf2_def swap_def)
  apply (simp add: step_def mtf2_def swap_def)

by (simp add: OPT2x step_def)

lemma Tp_split: length qs1 = length as1 \implies T_p s (qs1 @ qs2) (as1 @ as2) = T_p s qs1 as1 + T_p (steps s qs1 as1) qs2 as2
apply (induct qs1 as1 arbitrary: s rule: list_induct2) by (simp_all)

lemma Tp_spliting: \( x \neq y \implies \text{set } xs \subseteq \{x, y\} \implies \text{set } ys \subseteq \{x, y\} \implies R \in \{[x, y], [y, x]\} \implies T_p R (xs @ [x, x]) (OPT2 (xs @ [x, x]) R) + T_p [x, y] ys (OPT2 ys [x, y]) = T_p R (xs @ [x, x] @ ys) (OPT2 (xs @ [x, x] @ ys) R)

proof
  assume nxy: \( x \neq y \)
  assume XSxy: \text{set } xs \subseteq \{x, y\}
  assume YSxy: \text{set } ys \subseteq \{x, y\}
  assume R: \( R \in \{[x, y], [y, x]\} \)
  { fix R
    assume XSxy: \text{set } xs \subseteq \{x, y\}
    have \( R \in \{[x, y], [y, x]\} \implies \text{set } xs \subseteq \{x, y\} \implies \text{steps } R (xs @ [x, x]) (OPT2 (xs @ [x, x]) R) = [x, y] \)
    proof (induct xs arbitrary: R)
      case Nil
      then show ?case
        apply (cases R = [x, y])
    apply simp_all
    apply (simp add: step_def)
    by (simp add: step_def mtf2_def swap_def)
  next
    case (Cons Q QS)
    let ?R' = (step R Q (hd (OPT2 ((Q # QS) @ [x, x]) R)))
    have a: \( Q \in \{x, y\} \) and b: \text{set } QS \subseteq \{x, y\} using Cons by auto
    have t: \( ?R' \in \{[x, y], [y, x]\} \)
      apply (rule stepxy) using nxy Cons by auto
  }
then have \( \text{length}(\text{OPT2 } (\text{QS } @ [x, x]) ?R') > 0 \)
apply (cases \( ?R' = [x, y] \)) by (simp_all add: OPT2_length)
then have OPT2 (\( \text{QS } @ [x, x] \)) ?R' \( \neq [] \) by auto
then have \( \text{hdtl}: \text{OPT2 } (\text{QS } @ [x, x]) ?R' = \text{hd } (\text{OPT2 } (\text{QS } @ [x, x]) \) ?R' \)
by auto

have maa: (tl (\( \text{OPT2 } ((Q \# \text{QS}) @ [x, x]) \) R)) = \( \text{OPT2 } (\text{QS } @ [x, x]) \) ?R' using tlofOPT2[OF a b Cons(2)] by auto
from Cons(2) have \( \text{length}(\text{OPT2 } ((Q \# \text{QS}) @ [x, x]) \) R) > 0
apply (cases \( R = [x, y] \)) by (simp_all add: OPT2_length)
then have \( \text{nempty}: \text{OPT2 } ((Q \# \text{QS}) @ [x, x]) \) R \( \neq [] \) by auto
then have \( \text{steps } R ((Q \# \text{QS}) @ [x, x]) (\text{OPT2 } ((Q \# \text{QS}) @ [x, x]) \) R)
# tl (\( \text{OPT2 } ((Q \# \text{QS}) @ [x, x]) \) R)
by (simp)
also have ...
# steps ?R' (\( \text{QS } @ [x, x] \)) (tl (\( \text{OPT2 } ((Q \# \text{QS}) @ [x, x]) \) R))
unfolding maa by auto
also have ... = steps ?R' (\( \text{QS } @ [x, x] \)) (\( \text{OPT2 } (\text{QS } @ [x, x]) \) ?R') using maa by auto
also with Cons(1)[OF t b] have ... = [x, y] by auto

finally show ?case.
qed 
}

note aa=\this

from aa XSxy R have tl: \( \text{steps } R (xs@[x,x]) (\text{OPT2 } (xs@[x,x]) \) R)
= [x, y] by auto

have uer: \( \text{length } (xs @ [x, x]) = \text{length } (\text{OPT2 } (xs @ [x, x]) \) R)
using R by (auto simp: OPT2_length)

have OPT2 \( (xs @ [x, x] @ ys) R = \text{OPT2 } (xs @ [x, x]) \) R @ OPT2 ys
[x, y]
apPLY (rule OPT2_split11)
using nxy XSxy YSxy R by auto

183
then have \( T_p R (xs@[x,x]@ys) (OPT2 (xs@[x,x]@ys) R) = T_p R ((xs@[x,x]@ys) (OPT2 (xs@[x,x]) R @ OPT2 ys [x, y]) \)
by auto
also have \( \ldots = T_p R (xs@[x,x]) (OPT2 (xs@[x,x]) R) + T_p [x,y] ys (OPT2 ys [x, y]) \)
using \( T_p \_split[of xs@[x,x] OPT2 (xs@[x,x]) R ys OPT2 ys [x, y], OF ucr, unfolded ll] \)
by auto
finally show \(?thesis by simp\)
qed

lemma OPTauseinander: \( x\neq y \Rightarrow set xs \subseteq \{x,y\} \Rightarrow set ys \subseteq \{x,y\} \Rightarrow \)
\( LTS \in \{[x,y],[y,x]\} \Rightarrow \) \( \) \( \) \( hd LTS = \) \( last xs \Rightarrow \)
\( xs = (pref@[hd LTS, hd LTS]) \Rightarrow \)
\( T_p [x,y] xs (OPT2 xs [x,y]) + T_p LTS ys (OPT2 ys LTS) \)
\( = T_p [x,y] (xs@ys) (OPT2 (xs@ys) [x,y]) \)
proof –
assume \( nxy: x\neq y \)
assume \( xsxy: set xs \subseteq \{x,y\} \)
assume \( ysxy: set ys \subseteq \{x,y\} \)
assume \( L: LTS \in \{[x,y],[y,x]\} \)
assume \( hd LTS = \) \( last xs \)
assume \( prefix: xs = (pref@[hd LTS, hd LTS]) \)
show \(?thesis\)
proof (cases \( LTS = [x,y]\))
case True
show \(?thesis unfolding True prefix\)
apply(simp)
apply(rule \( T_p\_splitting\_\_simplified\))
using \( nxy xsxy ysxy prefix\) by auto
next
case False
with \( L\) have \( TT: LTS = [y,x]\) by auto
show \(?thesis unfolding TT prefix\)
apply(simp)
apply(rule \( T_p\_splitting\_\_simplified\))
using \( nxy xsxy ysxy prefix\) by auto
qed
qed

13.3 Phase Partitioning lemma

theorem Phase\_partitioning\_general:
fixes \( P :: (nat \ast \text{'is}) \text{prof} \Rightarrow nat \Rightarrow nat \text{list} \Rightarrow bool \)
and \( A :: (nat \ast \text{'is}, \text{nat, answer}) \text{alg\_on\_rand} \)
assumes \( \text{xny: (x0::nat)} \neq y0 \)
and \( \text{cpos: (c::real) \geq 0} \)
and static: set \( \sigma \subseteq \{x0, y0\} \)
and initial: \( P \ (\text{map\_pmf} (\%\text{is}. ([x0,y0],\text{is})) \ (\text{fst} A [x0,y0])) \ x0 \ [x0,y0] \)
and \( D: a \ b \ s. \ \sigma \in Lxx \ a \ b \ \Longrightarrow a \neq b \ \Longrightarrow \{a,b\} \{x0,y0\} \ \Longrightarrow P \ s \ a \)
\( [x0,y0] \ \Longrightarrow \ \text{set} \ \sigma \subseteq \{a,b\} \)
\( \Longrightarrow T\text{\_on\_rand}' \ A \ s \ \sigma \leq c \ast T_p [a,b] \ \sigma \ (\text{OPT2} \ \sigma [a,b]) \ \land P \)
\( (\text{config\_rand} A \ s \ \sigma) \ (\text{last} \ \sigma) \ [x0,y0] \)
shows \( T_p\text{\_on\_rand} A [x0,y0] \ \sigma \leq c \ast T_{p\text{-opt}} [x0,y0] \ \sigma + c \)
proof

\{
fix \( x \ y \ s \)
have \( x \neq y \ \Longrightarrow P \ s \ x \ [x0,y0] \ \Longrightarrow \ \text{set} \ \sigma \subseteq \{x,y\} \ \Longrightarrow \{x,y\}\{x0,y0\} \ \Longrightarrow T\text{\_on\_rand}' A \ s \ \sigma \leq c \ast T_p [x,y] \ \sigma \ (\text{OPT2} \ \sigma [x,y]) + c \)
proof (induction length \( \sigma \) arbitrary; \( \sigma \ \times \ y \ s \ \text{rule: less\_induct})
\begin{itemize}
\item case \( (\text{less} \ \sigma) \)
\end{itemize}
show ?case
proof (cases \( \exists x \ s. \ \sigma = x @ y @ s \land x \in Lxx \ x \ y \))
\begin{itemize}
\item case \( \text{True} \)
\end{itemize}
then obtain \( x \ s \ y \) where \( q\text{s: } \sigma = x @ y \ @ s \) and \( x \in Lxx \ x \ y \) by auto
with \( Lxx1 \)
\begin{itemize}
\item have \( \text{len: length} \ ys < \text{length} \ \sigma \) by fastforce
\end{itemize}
from \( q\text{s(1) less(4)} \)
\begin{itemize}
\item have \( \text{ysxy: set} \ ys \subseteq \{x,y\} \) by auto
\end{itemize}
\begin{itemize}
\item have \( x\text{sset: set} \ xs \subseteq \{x,y\} \) using \( \text{less(4)} \ \text{qs by auto}
\end{itemize}
from \( x\text{sLxx Lxx1} \)
\begin{itemize}
\item have \( \text{lsxtl: length} \ xs \geq 2 \) by auto
\end{itemize}
then have \( x\text{s\_not\_Nil: xs \neq []} \) by auto
from \( D[OF x\text{sLxx less(2) less(5) less(3)} \ x\text{sset}] \)
\begin{itemize}
\item have \( D1: T\text{\_on\_rand}' A \ s \ xs \leq c \ast T_p [x,y] \ xs \ (\text{OPT2} \ xs [x,y]) \)
\item and \( \text{inv: } P \ (\text{config\_rand} A \ s \ xs) \ (\text{last} \ xs) \ [x0,y0] \) by auto
\end{itemize}
from \( x\text{sLxx Lxx\_ends\_in\_two\_equal} \)
\begin{itemize}
\item obtain \( \text{pref e where} \ xs = \text{pref @} \)
\item \([e,e]\) by metis
\item then have \( \text{ends\_with\_same: } xs = \text{pref @} \ [\text{last} \ xs, \ \text{last} \ xs] \) by auto
\end{itemize}
let \( ?e' = [\text{last} \ xs, \ \text{other} \ (\text{last} \ xs) \ x \ y] \)

185
have setys: set ys ⊆ \{x, y\} using qs less by auto
have setxs: set xs ⊆ \{x, y\} using qs less by auto
have lxs: last xs ∈ set xs using xs_not_Nil by auto
from lxs setxs have lxsy: last xs ∈ \{x, y\} by auto
from lxs setxs have otherzy: other (last xs) x y ∈ \{x, y\} by (simp add: other_def)
from less(2) have other_diff: last xs ≠ other (last xs) x y by (simp add: other_def)

have lo: \{last xs, other (last xs) x y\} = \{x0, y0\}
using lxsy otherzy other_diff less(5) by force

have nextstate: [[last xs, other (last xs) x y], [other (last xs) x y, last xs]]
= \{[x, y], [y, x]\} using lxsy otherzy other_diff by fastforce
have setys": set ys ⊆ \{last xs, other (last xs) x y\}
using setys lxsy otherzy other_diff by fastforce

have c: T_on_rand' A (config'_rand A s xs) ys
≤ c * T_p ?c' ys (OPT2 ys ?c') + c
apply (rule less(1))
apply (fact len)
apply (fact other_diff)
apply (fact inv)
apply (fact setys')
by (fact lo)

have well: T_p [x, y] xs (OPT2 xs [x, y]) + T_p ?c' ys (OPT2 ys ?c')
= T_p [x, y] (xs @ ys) (OPT2 (xs @ ys) [x, y])
apply (rule OPTauseinander[where pref=pref])
apply (fact)+
using lxsy otherzy other_diff by fastforce
apply (simp)
using endswithsame by simp

have E0: T_on_rand' A s σ
= T_on_rand' A s (xs@ys) using qs by auto
also have E1: \ldots = T_on_rand' A s xs + T_on_rand' A (config'_rand A s xs) ys
by (rule T_on_rand' append)
also have E2: \ldots ≤ T_on_rand' A s xs + c * T_p ?c' ys (OPT2 ys ?c') + c

186
using \( c \) by simp
also have \( E3: \ldots \leq c \times T_p [x, y] \) \( xs \) (OPT2 \( xs [x, y] \) \( + c \times T_p \) ?c' \( ys \) (OPT2 \( ys \) ?c') \) + c

using D1 by simp
also have \( \ldots = c \times (T_p [x, y] \) \( xs \) (OPT2 \( xs [x, y] \) \( + T_p \) ?c' \( ys \) (OPT2 \( ys \) ?c')) \) + c

using cpos apply(auto) by algebra
also have \( \ldots = c \times (T_p [x, y] \) (xs@ys) (OPT2 \( (xs@ys) [x, y] \)) \) + c
using well by auto
also have \( E4: \ldots = c \times (T_p [x, y] \) \( \sigma \) (OPT2 \( \sigma [x, y] \)) \) + c
using qs by auto
finally show \( \{?thesis \} \).

next
case False
note \( f1=this \)
from Lxx_othercase[OF less(\( \{4\} \) this, unfolded hideit_def] have
nodouble: \( \sigma = [] \) \( \lor \sigma \in \text{lang}(\text{nodouble } x y) \) by auto
show \( \{?thesis \}
proof (cases \( \sigma = [] \))
case True
then show \( \{?thesis \text{ using cpos by simp} \)
next
case False

from False nodouble have qsnodouble: \( \sigma \in \text{lang}(\text{nodouble } x y) \) by auto
let \( \{padded = pad \sigma x y \)

have padset: set \( \{padded \subseteq \{x, y\} \) using less(\( \{4\} \) by(simp)

from False pad_adds2[of \( \sigma x y \) less(\( \{4\} \) obtain addum where \( \text{ui: pad} \sigma x y = \sigma \ @ \{\text{last } \sigma \} \) by auto
from nodouble_padded[OF False qsnodouble] have pLxx: \( \{\text{padded } \in Lxx x y \} \).

have E0: \( T_{on_rand'} A s \sigma \leq T_{on_rand'} A s \) ?padded
proof
have \( T_{on_rand'} A s \sigma = \text{sum} \ (T_{on_rand'} n A s \sigma ) \{..<\text{length } \sigma \} \)
by(rule T_{on_rand'} as_sum)
also have \( \ldots = \text{sum} \ (T_{on_rand'} n A s (\sigma @ \{\text{last } \sigma \}) ) \{..<\text{length } \sigma \} \)
proof(rule sum.cong, goal_cases)
case (2 t)
then have \( t < \text{length } \sigma \) by auto

187
then show \( \textit{thesis} \) \( \text{by } \text{auto} \)

qed

also have \( \ldots \leq T\_on\_rand' A \) \( \textit{thesis} \) \( \text{by } \text{auto} \)

qed

also have \( \ldots \leq c \ast T_p [x,y] \) \( \textit{thesis} \) \( \text{by } \text{auto} \)

using \( \text{simp add: nth_append} \)

also have \( \ldots \leq c \ast (T_p [x,y] \sigma (\text{OPT2} \sigma [x,y]) + 1) \)

proof

from \( \text{False less(2)} \) obtain \( \sigma' x^{'} y^{'} \) where \( q_3' \): \( \sigma = \sigma' @ [x'] \) and \( x' \):

\( x' = \text{last } \sigma \neq x' y' \in \{x,y\} \)

by \( \text{metis append butlast last_id insert_iff} \)

have \( \text{tlp: last } \sigma \in \{x,y\} \) \( \text{using less(4)} \) \( \text{False last_in_set} \) \( \text{by } \text{blast} \)

with \( x' \) have \( \text{grr: } \{x,y\} = \{x',y'\} \) \( \text{by } \text{auto} \)

then have \( x = x' \land y = y' \lor (x = y' \land y = x') \) \( \text{using less(2)} \) \( \text{by } \text{auto} \)

then have \( \text{tts: } [x, y] \in \{[x', y'], [y', x']\} \) \( \text{by } \text{blast} \)

from \( q_3' \) \( \text{ui have } \text{pd: } \textit{thesis} = \sigma' @ [x', x'] \) \( \text{by } \text{auto} \)

have \( T_p [x,y] (\textit{thesis}) (\text{OPT2} (\textit{thesis}) [x,y]) \)

\( = T_p [x,y] (\sigma' @ [x', x']) (\text{OPT2} (\sigma' @ [x', x']) [x,y]) \)

unfolding \( \text{pd} \) \( \text{by } \text{simp} \)

also have \( \text{gr: } \ldots \leq T_p [x,y] (\sigma' @ [x']) (\text{OPT2} (\sigma' @ [x']) [x,y]) + 1 \)

\( \text{apply(\text{rule OPT2_padded[where x=x' and y=y']} \)} \)

\( \text{apply(\text{fact})} \)

\( \text{using grgr q_3' \text{less(4)} \text{by } \text{auto} \)} \)

also have \( \ldots \leq T_p [x,y] (\sigma) (\text{OPT2} (\sigma) [x,y]) + 1 \)

unfolding \( q_3' \) \( \text{by } \text{simp} \)

finally show \( \textit{thesis} \) \( \text{using cpos by (meson mult_left_mono \text{ mono of nat le_iff})} \)

qed

also have \( \ldots = c \ast T_p [x,y] (\text{OPT2} \sigma [x,y]) + c \) \( \text{by (metis (no_types, lifting) mult.commute of nat 1 of nat add semiring_normalization_rules(2))} \)

finally show \( \textit{thesis} \).

qed

qed
have \( T_{\text{on}_\text{rand}} A [x0, y0] \sigma \leq c * \text{real} (T_p [x0, y0] \sigma (\text{OPT2} \sigma [x0, y0])) + c \)

apply (rule allg)
apply (fact)
using initial apply (simp add: map_pmf_def)
apply (fact assms(3))
by simp
also have \( \ldots = c * T_p_{\text{opt}} [x0, y0] \sigma + c \)
using \( \text{OPT2}_{\text{is_opt}}[\text{OF assms(3,1)}] \) by (simp)
finally show \( \text{thesis} \).
qed

term \( A :: (\text{nat,\'is}) \) alg_on

theorem \( \text{Phase\_partitioning\_general\_det} \):
fixes \( P :: (\text{nat state * \'is}) \Rightarrow \text{nat} \Rightarrow \text{nat list} \Rightarrow \text{bool} \)
and \( A :: (\text{nat,\'is}) \) alg_on
assumes \( \text{xny} : (x0 :: \text{nat}) \neq y0 \)
and \( \text{cpos} : (c :: \text{real}) \geq 0 \)
and \( \text{static} : \text{set} \ \sigma \subseteq \{x0, y0\} \)
and \( \text{initial} : P ([x0,y0], ([\text{fst} A [x0,y0]]) x0 [x0,y0] \)
and \( \text{D} : \forall a b \ \sigma. \ \sigma \in Lxx a b \implies a \neq b \implies \{a,b\} = \{x0,y0\} \implies P \ s \ a \)
\( [x0,y0] \implies \text{set} \ \sigma \subseteq \{a,b\} \)
\( \implies T_{\text{on}_\text{'}A} \ s \ \sigma \leq c * T_p [a,b] \sigma (\text{OPT2} \sigma [a,b]) \land P (\text{config}_\text{'} A \ s \ \sigma) (\text{last} \ \sigma) [x0,y0] \)
shows \( T_{p_{\text{opt}}} A [x0,y0] \sigma \leq c * T_p_{\text{opt}} [x0,y0] \sigma + c \)
proof -
thm Phase\_partitioning\_general

thm \( T_{\text{deter}_\text{rand}} \)
term \( T_{\text{on}_\text{'}A} \)
term embed
show \( \text{thesis} \) oops

end

14 List factoring technique

theory \( \text{List\_Factoring} \)
imports
\( \text{Partial\_Cost\_Model} \)

189
14.1 Helper functions

14.1.1 Helper lemmas

**Lemma baf**: assumes \( q \in \text{set} \ s \) distinct \( s \)
shows \( \text{before} \ q \ s \cup \{ q \} \cup \text{after} \ q \ s = \text{set} \ s \)
proof –

have \( \text{before} \ q \ s \cup \{ y. \text{index} \ s \ y = \text{index} \ s \ q \land q \in \text{set} \ s \} \)
  = \( \{ y. \text{index} \ s \ y \leq \text{index} \ s \ q \land q \in \text{set} \ s \} \)
unfolding before_in_def apply (auto) by (simp add: le_neq_implies_less)
also have \( \ldots = \{ y. \text{index} \ s \ y \leq \text{index} \ s \ q \land y \in \text{set} \ s \land q \in \text{set} \ s \} \)
apply (auto) by (metis index_conv_size_if_notin index_less_size_conv not_less)
also with \( \langle q \in \text{set} \ s \rangle \) have \( \ldots = \{ y. \text{index} \ s \ y \leq \text{index} \ s \ q \land y \in \text{set} \ s \} \)
by auto
finally have \( \text{before} \ q \ s \cup \{ y. \text{index} \ s \ y = \text{index} \ s \ q \land q \in \text{set} \ s \} \cup \text{after} \ q \ s \)
  = \( \{ y. \text{index} \ s \ y \leq \text{index} \ s \ q \land y \in \text{set} \ s \} \cup \{ y. \text{index} \ s \ y > \text{index} \ s \ q \land y \in \text{set} \ s \} \)
unfolding before_in_def by simp
also have \( \ldots = \text{set} \ s \) by auto
finally show ?thesis using assms by simp
qed

**Lemma index_sum**: assumes distinct \( s \) \( q \in \text{set} \ s \)
shows \( \text{index} \ s \ q = (\sum e \in \text{set} \ s. \text{if} \ e < q \text{ in } s \text{ then } 1 \text{ else } 0) \)
proof –

from assms have \( \text{bia_empty} : \text{before} \ q \ s \cap (\{ q \} \cup \text{after} \ q \ s) = \{ \} \)
  by (auto simp: before_in_def)
from baf [OF assms(2) assms(1)] have \( (\sum e \in \text{set} \ s. \text{if} \ e < q \text{ in } s \text{ then } 1::\text{nat} \text{ else } 0) \)
  = \( (\sum e \in (\text{before} \ q \ s \cup \{ q \} \cup \text{after} \ q \ s). \text{if} \ e < q \text{ in } s \text{ then } 1 \text{ else } 0) \) by auto
also have \( \ldots = (\sum e \in (\text{before} \ q \ s. \text{if} \ e < q \text{ in } s \text{ then } 1 \text{ else } 0) + (\sum e \in \{ q \}. \text{if} \ e < q \text{ in } s \text{ then } 1 \text{ else } 0) + (\sum e \in \text{after} \ q \ s. \text{if} \ e < q \text{ in } s \text{ then } 1 \text{ else } 0) \)
proof –

have \( (\sum e \in (\text{before} \ q \ s \cup \{ q \} \cup \text{after} \ q \ s). \text{if} \ e < q \text{ in } s \text{ then } 1::\text{nat} \text{ else } 0) \)
  = \( (\sum e \in (\text{before} \ q \ s \cup (\{ q \} \cup \text{after} \ q \ s)). \text{if} \ e < q \text{ in } s \text{ then } 1::\text{nat} \text{ else } 0) \)
also have ... = \((\sum_{e \in \text{before } q \ s} \text{if } e < q \text{ in } s \text{ then } 1 \text{ else } 0) + (\sum_{e \in \{q\} \cup \text{after } q \ s} \text{if } e < q \text{ in } s \text{ then } 1 \text{ else } 0)\) applying (rule sum_Un_nat) by (simp_all)
also have ... = \((\sum_{e \in \text{before } q \ s} \text{if } e < q \text{ in } s \text{ then } 1 \text{ else } 0) + (\sum_{e \in \{q\} \cup \text{after } q \ s} \text{if } e < q \text{ in } s \text{ then } 1 \text{ else } 0)\) using bia_empty by auto
also have ... = \((\sum_{e \in \text{before } q \ s} \text{if } e < q \text{ in } s \text{ then } 1 \text{ else } 0) + (\sum_{e \in \{q\}} \text{if } e < q \text{ in } s \text{ then } 1 \text{ else } 0) + (\sum_{e \in \text{after } q \ s} \text{if } e < q \text{ in } s \text{ then } 1 \text{ else } 0)\) by (simp add: before_in_def)
finally show ?thesis .

lemma t_p\_sumofALG: assumes distinct (fst s) \Longrightarrow snd a = [] \Longrightarrow (qs!i) \in \text{set} (fst s)

\Longrightarrow t_p (fst s) (qs!i) a = (\sum_{e \in \text{set} (fst s)} \text{ALG } e \text{ qs } i \text{ s})

unfolding t_p\_def apply(simp add: split_def )
using index_sum by metis

lemma t_p\_sumofALGreal: assumes distinct (fst s) snd a = [] qs!i \in \text{set}(fst s)
shows real(t_p (fst s) (qs!i) a) = (\sum_{e \in \text{set} (fst s)} \text{real(ALG } e \text{ qs } i \text{ s)})
proof –

14.1.2 ALG

fun ALG :: \'a \Rightarrow \'a list \Rightarrow nat \Rightarrow (\'a list \ast \'is) \Rightarrow nat where
ALG x qs i s = (if x < (qs!i) in fst s then 1::nat else 0)

lemma t_p\_sumofALGreal: assumes distinct (fst s) snd a = [] qs!i \in \text{set}(fst s)
shows real(t_p (fst s) (qs!i) a) = (\sum_{e \in \text{set} (fst s)} \text{real(ALG } e \text{ qs } i \text{ s)})
proof –

191
from assms have \[ \text{real}(t_p(fst\ s) (qs!i)) = \text{real}(\sum_{e \in \text{set}\ (fst\ s)} ALG\ e\ qs\ i\ s)\] using \(t_p,\text{sumofALG}\) by metis
also have \(\ldots = (\sum_{e \in \text{set}\ (fst\ s)} \text{real}(ALG\ e\ qs\ i\ s))\) by auto
finally show \(?\text{thesis}\).
qed

14.1.3 The function \(\text{steps}'\)

\[
\begin{align*}
\text{fun} \quad \text{steps}' \quad \text{where} \\
steps' s 0 = s \\
\mid steps' s \underbrace{(\text{Suc}\ n)} = s \\
\mid steps' s (q\#qs) (a/#as) (\text{Suc}\ n) = steps' (\text{step}\ s\ q\ a)\ qs\ as\ n
\end{align*}
\]

\textbf{lemma steps'_steps:} length as = length qs \(\implies\) steps' s as qs (length as) = steps s as qs
by (induct arbitrary: s rule: list_induct2, simp_all)

\textbf{lemma steps'_length:} length qs = length as \(\implies\) n \(\leq\) length as 
\(\implies\) length (steps' s qs as n) = length s
apply (induct qs as arbitrary: s n rule: list_induct2)
apply (simp)
apply (case_tac n)
by (auto)

\textbf{lemma steps'_set:} length qs = length as \(\implies\) n \(\leq\) length as 
\(\implies\) set (steps' s qs as n) = set s
apply (induct qs as arbitrary: s n rule: list_induct2)
apply (simp)
apply (case_tac n)
by (auto simp: set_step)

\textbf{lemma steps'_distinct2:} length qs = length as \(\implies\) n \(\leq\) length as 
\(\implies\) distinct s \(\implies\) distinct (steps' s qs as n)
apply (induct qs as arbitrary: s n rule: list_induct2)
apply (simp)
apply (case_tac n)
by (auto simp: distinct_step)

\textbf{lemma steps'_distinct:} length qs = length as \(\implies\) length as = n 
\(\implies\) distinct (steps' s qs as n) = distinct s
apply (induct qs as arbitrary: s n rule: list_induct2) by (auto simp: distinct_step)

lemma steps_dist_perm: length qs = length as \implies length as = n
\implies dist_perm s s \implies dist_perm (steps' s qs as n) (steps' s qs as n)
using steps_set_distinct by blast

lemma steps_rests: length qs = length as \implies n \le length as \implies steps' s qs as n = steps' s (qs@@r1) (as@@r2) n
apply (induct qs as arbitrary: s n rule: list_induct2)
apply (simp) apply (case_tac n) by auto

lemma steps_append: length qs = length as \implies length qs = n \implies steps' s (qs@@[q]) (as@@[a]) (Suc n) = step (steps' s qs as n) q a
apply (induct qs as arbitrary: s n rule: list_induct2) by auto

14.1.4 \textsc{Alg'\_det}

definition \texttt{Alg'\_det Strat qs init i x} = \texttt{Alg x qs i (swaps (snd (Strat|i))) (steps' init qs Strat i))}

lemma \texttt{Alg'\_det_append}: n < length Strat \implies n < length qs \implies \texttt{Alg'\_det Strat (qs@@a) init n x} = \texttt{Alg'\_det Strat qs init n x}

proof –
assume qs: n < length qs
assume S: n < length Strat

have tt: (qs @@ a) ! n = qs ! n
using qs by (simp add: nth_append)

have steps' init (take n qs) (take n Strat) n = steps' init (((take n qs) @@ drop n qs) (((take n Strat) @@ (drop n Strat)) n
apply (rule steps'_rests)
using S qs by auto
then have A: steps' init (take n qs) (take n Strat) n = steps' init qs Strat n by auto
have steps' init (take n qs) (take n Strat) n = steps' init (((take n qs) @@ ((drop n qs)@@a)) (((take n Strat) @@ (drop n Strat)@@[]))) n
apply (rule steps'_rests)
using S qs by auto
then have B: steps' init (take n qs) (take n Strat) n = steps' init (qs@@a) (Strat@@[]) n
by (metis append_assoc List.append_take_drop_id)
from A B have steps' init qs Strat n = steps' init (qs@a) (Strat@[]) n by auto
then have C: steps' init qs Strat n = steps' init (qs@a) Strat n by auto

show ?thesis unfolding ALG' det_def C
  unfolding ALG simps tt by auto
qed

14.1.5 ALG'

abbreviation config'' A qs init n == config_rand A init (take n qs)
definition ALG' A qs init i x = E( map pmf (ALG x qs i) (config'' A qs init i))

lemma ALG'.refl: qs!i = x ==> ALG' A qs init i x = 0
unfolding ALG'.def by(simp add: split_def before_in_def)

14.1.6 ALGxy_det

definition ALGxy_det where
  ALGxy_det A qs init x y = (\sum i \in{1..<length qs}. (if (qs!i \in \{y,x\}) then
  ALG' det A qs init i y + ALG' det A qs init i x
  else 0::nat))

lemma ALGxy_det_alternative: ALGxy_det A qs init x y
  = (\sum i \in{i. i<length qs \& (qs!i \in \{y,x\})}. ALG' det A qs init i y +
  ALG' det A qs init i x)
proof
  have f: {i. i<length qs} = {..<length qs} by(auto)

  have e: {i. i<length qs \& (qs!i \in \{y,x\})} = {i. i<length qs} \cap {i. (qs!i
  \in \{y,x\})}
    by auto
  have (\sum i \in{i. i<length qs \& (qs!i \in \{y,x\})}. ALG' det A qs init i y +
  ALG' det A qs init i x)
    = (\sum i \in{i. i<length qs} \cap {i. (qs!i \in \{y,x\})}. ALG' det A qs init i y
    + ALG' det A qs init i x)
    unfolding e by simp
  also have \ldots = (\sum i \in{1..<length qs}. (if i \in \{i. (qs!i \in \{y,x\}) then
  ALG' det A qs init i y + ALG' det A qs init i x
  else 0))
    apply(rule sum.inter_restrict) by auto
  also have \ldots = (\sum i \in{..<length qs}. (if i \in \{i. (qs!i \in \{y,x\}) then
ALG′ det A qs init i y + ALG′ det A qs init i x

unfolding f by auto
also have ... = ALGxy det A qs init x y
unfolding ALGxy det def by auto
finally show ?thesis by simp
qed

14.1.7 ALGxy

definition ALGxy where
ALGxy A qs init x y = (∑ i∈{..<length qs} ∩ {i. (qs!i ∈ {y,x})}. ALG′ A qs init i y + ALG′ A qs init i x)

lemma ALGxy_def2:
ALGxy A qs init x y = (∑ i∈{i. i<length qs ∧ (qs!i ∈ {y,x})}. ALG′ A qs init i y + ALG′ A qs init i x)
proof
have a: {i. i<length qs ∧ (qs!i ∈ {y,x})} = {..<length qs} ∩ {i. (qs!i ∈ {y,x})} by auto
show ?thesis unfolding ALGxy def a by simp
qed

lemma ALGxy_append: ALGxy A (rs@[r]) init x y =
    ALGxy A rs init x y + (if (r ∈ {y,x}) then ALG′ A (rs@[r]) init (length rs) y + ALG′ A (rs@[r]) init (length rs) x else 0 )
proof
have ALGxy A (rs@[r]) init x y = (∑ i∈{..<Suc (length rs)}. (if i∈{i. (rs@[r]) ! i ∈ {y,x}} then
    ALG′ A (rs@[r]) init i y +
    ALG′ A (rs@[r]) init i x else 0 )
apply(rule sum.inter_restrict) by simp
also have ... = (∑ i∈{..<Suc (length rs)}. (if i∈{i. (rs@[r]) ! i ∈ {y}, x}}
then
    ALG′ A (rs@[r]) init l y +
    ALG′ A (rs@[r]) init l x else 0 ) + (if length rs∈{i. (rs@[r]) ! i ∈ {y}, x}}
then
    ALG′ A (rs@[r]) init (length rs) y +
    ALG′ A (rs@[r]) init (length rs) x else 0 ) by simp
also have ... = ALGxy A rs init x y + (if r ∈ {y, x} then
    ALG′ A (rs@[r]) init (length rs) y +
\[ \text{ALG}' \ A \ (rs @ [r]) \ \text{init}(\text{length } rs) \ x \ \text{else } 0 \]apply(simp add: ALGxy_def sum.inter_restr nth_append)unfolding ALG'_defapply(rule sum.cong)apply(simp) by(auto simp: nth_append)finally show \text{thesis}.

\text{qed}

\text{lemma} ALGxy_wholerange: ALGxy A qs init x y
= (\sum_i < (\text{length } qs). \ (\text{if } qs ! i \in \{y, x\} \\
\text{then } \text{ALG}' A \ qs \ init \ i \ y + \text{ALG}' A \ qs \ init \ i \ x \\
\text{else } 0 ))
\text{proof} –
\text{have} ALGxy A qs init x y
= (\sum_i \in \{i. \ i < \text{length } qs\} \cap \{i. \ qs ! i \in \{y, x\}\}.
\text{ALG}' A \ qs \ init \ i \ y + \text{ALG}' A \ qs \ init \ i \ x)
unfolding ALGxy_defapply(rule sum.cong)
apply(simp) apply(blast)
by simp
\text{also have} \ldots = (\sum_i \in \{i. \ i < \text{length } qs\}. \ \text{if } i \in \{i. \ qs ! i \in \{y, x\}\} \\
\text{then } \text{ALG}' A \ qs \ init \ i \ y + \text{ALG}' A \ qs \ init \ i \ x \\
\text{else } 0 )) \ \text{apply(rule sum.cong) by(auto)}
\text{finally show } \text{thesis}.
\text{qed}

14.2 Transformation to Blocking Cost
\text{lemma} umformung:
\text{fixes} A :: (('a:linorder) list,'is,'a,(nat * nat list)) \text{alg_on_rand}
\text{assumes} no_paid: \ls q \ \forall ((\text{free,paid})_s) \in (\snd A \ (s,is) \ q). \ \text{paid} = []
\text{assumes} inlist: set qs \subseteq set init
\text{assumes} dist: distinct init
\text{assumes} \ ext x. \ x < \text{length } qs \ \rightarrow \ \text{finite} (\text{set_pmf} \ (\text{config''} A \ qs \ init \ x))
\text{shows} T_{\text{on-std}} A \ init \ qs = 
(\sum (x,y) \in \{(x,y). \ x \in \text{set init} \land y \in \text{set init} \land x < y\}. \ \text{ALGxy} A \ qs \ init \ x \ y)
\text{proof} –
\text{have} config_dist: \ \forall n. \ \forall xa \in \text{set_pmf} \ (\text{config''} A \ qs \ init \ n). \ \text{distinct} (\fst xa)

196
using dist config_rand_distinct by metis

have E0: \( T_{p \cdot \text{on rand}} A \text{ init } qs = \) 
\( \left( \sum i \in \{..<\text{length } qs \} \cdot T_{p \cdot \text{on rand}_n} A \text{ init } qs i \right) \) unfolding T_on_rand_as_sum 
by auto
also have ... = 
\( \left( \sum i < \text{length } qs \cdot E (\text{bind pmf} (\text{config'' } A \text{ qs init } i) \right) 
\left( \lambda s. \text{bind pmf} (\text{snd } A s (qs ! i)) \right) 
\left( \lambda (a, nis). \text{return pmf} (\text{real } (\sum x \in \text{set init } \cdot \text{ALG } x \text{ qs } i s)) \right) \) )
apply (rule sum.cong)
apply (simp)
apply (simp add: bind_return_pmff bind_assoc_pmff)
apply (rule arg_cong[where f=E])
apply (rule bind_pmff_cong)
apply (simp)
apply (rule bind_pmff_cong)
apply (simp)
apply (simp add: split_def)
apply (subst t_p_sumofALGreal)
proof (goal_cases)
case 1
then show ?case using config_dist by (metis)
next
case (2 a b c)
then show ?case using no_paid[of fst b snd b] by (auto simp add: split_def)
next
case (3 a b c)
with config_rand_set have a: set (fst b) = set init by metis
with inlist have set qs \( \subseteq \) set (fst b) by auto
with 3 show ?case by auto
next
case (4 a b c)
with config_rand_set have a: set (fst b) = set init by metis
then show ?case by (simp)
qed

also have ... = \( \left( \sum i < \text{length } qs \right) 
E (\text{map pmf} (\lambda(is, s). (\text{real } (\sum x \in \text{set init } \cdot \text{ALG } x \text{ qs } (is, s)))) \right) 
\left( \text{config'' } A \text{ qs init } i \right) \) 
apply (simp only: map_pmff_def split_def) by simp
also have \( E_1: \ldots = (\sum i < \text{length } qs. (\sum x \in \text{set init}. \ ALG' A \ qs \ init \ i \ x)) \)

apply (rule sum.cong)
apply (simp)
apply (simp add: split_def ALG'_def)
apply (rule E_linear_sum_allg)
by (rule assms(4))

also have \( E_2: \ldots = (\sum x \in \text{set init}. (\sum i < \text{length } qs. \ ALG' A \ qs \ init \ i \ x)) \)
by (rule sum.swap)

also have \( E_3: \ldots = (\sum x \in \text{set init}. (\sum y \in \text{set init}. (\sum i \in \{i. \ i < \text{length } qs \land qs!i=y\}. \ ALG' A \ qs \ init \ i \ x))) \)

proof (rule sum.cong, goal_cases)
case (2 \( x \))
have \( (\sum i < \text{length } qs. \ ALG' A \ qs \ init \ i \ x) = \sum (\%i. \ ALG' A \ qs \ init \ i \ x) \ (\{i. \ i < \text{length } qs\} \ ALG' A \ qs \ init \ i \ x) \)
by (metis lessThan_def)
also have \( \ldots = \sum (\%i. \ ALG' A \ qs \ init \ i \ x) \)

\( \{y. \ y \in \text{set init} \} (\lambda y. \ (i. \ i < \text{length } qs \land qs!i=y) \) \)

apply (rule sum.cong)
apply (auto)
using inlist by auto

also have \( \ldots = \sum (\%t. \ \sum (\%i. \ ALG' A \ qs \ init \ i \ x) \ (\{i. \ i < \text{length } qs \land qs!i=t\} \ (y. \ y \in \text{set init}))) \)

apply (rule sum.UNION_disjoint)
apply (simp_all) by force

also have \( \ldots = (\sum y \in \text{set init}. \ \sum i \mid i < \text{length } qs \land qs!i=y \) \)
\( ALG' A \ qs \ init \ i \ x) \) by auto

finally show \( ?case. \)
qed (simp)

also have \( \ldots = (\sum (x, y) \in (\text{set init} \times \text{set init}). \)

\( (\sum i \in \{i. \ i < \text{length } qs \land qs!i=y\}. \ ALG' A \ qs \ init \ i \ x)) \)
by (rule sum.cartesian_product)

also have \( \ldots = (\sum (x, y) \in \{(x, y). \ x \in \text{set init} \land y \in \text{set init}\}. \)

\( (\sum i \in \{i. \ i < \text{length } qs \land qs!i=y\}. \ ALG' A \ qs \ init \ i \ x)) \)
by simp

also have \( E_4: \ldots = (\sum (x, y) \in \{(x, y). \ x \in \text{set init} \land y \in \text{set init} \land x \neq y\}. \)

\( (\sum i \in \{i. \ i < \text{length } qs \land qs!i=y\}. \ ALG' A \ qs \ init \ i \ x)) \) \( (\sum (x, y) \in ?L. \ ?f x y) = (\sum (x, y) \in ?R. \ ?f x y)) \)

proof –
let \( ?M = \{(x, y). \ x \in \text{set init} \land y \in \text{set init} \land x=y\} \)

have \( A: ?L = ?R \cup ?M \) by auto
have $B$: $\{\} = ?R \cap ?M$ by auto

have $(\sum (x,y) \in ?L. \ ?f x y) = (\sum (x,y) \in ?R \cup ?M. \ ?f x y)$
  by(simp only: A)
also have $\ldots = (\sum (x,y) \in ?R. \ ?f x y) + (\sum (x,y) \in ?M. \ ?f x y)$
  apply(rule finite_subset[where $B = \text{set init} \times \text{set init}$])
  \begin{itemize}
    \item apply(auto)
    \item apply(rule finite_subset[where $B = \text{set init} \times \text{set init}$])
  \end{itemize}
by(auto)
also have $(\sum (x,y) \in ?M. \ ?f x y) = 0$
  apply(rule sum.neutral)
by (auto simp add: $\text{ALG}'_\text{refl}$)
finally show ?thesis by simp
qed

also have $\ldots = (\sum (x,y) \in \{(x,y). \ x \in \text{set init} \land y \in \text{set init} \land x < y\}.
  (\sum i \in \{i. \ i < \text{length qs} \land qsi = y\}. \ \text{ALG}' A qsi init i \ x) +
  (\sum i \in \{i. \ i < \text{length qs} \land qsi = x\}. \ \text{ALG}' A qsi init i \ y))$
  (\text{is} (\sum (x,y) \in ?L. \ ?f x y) = (\sum (x,y) \in ?R. \ ?f x y + \ ?f y x))$
proof --
  let $?R' = \{(x,y). \ x \in \text{set init} \land y \in \text{set init} \land y < x\}$
  have $A$: $?L = ?R \cup ?R'$ by auto
  have $\{\} = ?R \cap ?R'$ by auto
  have $C$: $?R' = (%(x,y). \ (y, x)) \ ?R$ by auto

  have $D$: $(\sum (x,y) \in ?R'. \ ?f x y) = (\sum (x,y) \in ?R. \ ?f x y)$
  proof --
    have $(\sum (x,y) \in ?R'. \ ?f x y) = (\sum (x,y) \in (%(x,y). \ (y, x)))$
    $(\sum z \in ?R. \ ((%(x,y). \ ?f x y) \circ (%(x,y). \ (y, x))) z)$
    apply(rule sum.reindex)
    by(fact swap_inj_on)
  also have $\ldots = (\sum z \in ?R. \ (%(x,y). \ ?f y x) \circ (%(x,y). \ (y, x))) z)$
  apply(rule sum.cong)
  by(auto)
finally show ?thesis.
qed

have $(\sum (x,y) \in ?L. \ ?f x y) = (\sum (x,y) \in ?R \cup ?R'. \ ?f x y)$
  by(simp only: A)
also have $\ldots = (\sum (x,y) \in ?R. \ ?f x y) + (\sum (x,y) \in ?R'. \ ?f x y)$
apply (rule sum.union_disjoint)
apply (rule finite_subset [where B=set init x set init])
apply (auto)
apply (rule finite_subset [where B=set init x set init])
by (auto)
also have \[ \ldots = (\sum (x,y) \in ?R. \ ?f x y) + (\sum (x,y) \in ?R. \ ?f y x) \]
by (simp only: D)
also have \[ \ldots = (\sum (x,y) \in ?R. \ ?f x y + ?f y x) \]
by (simp add: split_def sum_distrib [symmetric])
finally show ?thesis .
qed

also have \[ E5: \ldots = (\sum (x,y) \in \{(x,y) \in \set init \cap y \in \set init \cap x < y\}. \ (\sum i \in \{i. \ i < \text{length } qs \land qs ! i = y \lor qs ! i = x\}). \ \text{ALG'} A \ \text{qs init } i \ y + \text{ALG'} A \ \text{qs init } i \ x) \]
apply (rule sum.cong)
apply (simp)
proof goal_cases
case (1 x)
then obtain a b where x = (a,b) and a: a \in \set init b \in \set init
a < b by auto
then have a \neq b by simp
then have \text{disj: } \{i. \ i < \text{length } qs \land qs ! i = b\} \cap \{i. \ i < \text{length } qs \land qs ! i = a\} = \{\}
by auto
have \text{unio: } \{i. \ i < \text{length } qs \land qs ! i = b\} \cup \{i. \ i < \text{length } qs \land qs ! i = a\} = \{a\} by auto
have \[ (\sum i \in \{i. \ i < \text{length } qs \land qs ! i = b\} \cup \{i. \ i < \text{length } qs \land qs ! i = a\}. \ \text{ALG'} A \ \text{qs init } i \ b + \text{ALG'} A \ \text{qs init } i \ a) = (\sum i \in \{i. \ i < \text{length } qs \land qs ! i = b\}. \ \text{ALG'} A \ \text{qs init } i \ b + \text{ALG'} A \ \text{qs init } i \ a) + (\sum i \in \{i. \ i < \text{length } qs \land qs ! i = a\}. \ \text{ALG'} A \ \text{qs init } i \ b + \text{ALG'} A \ \text{qs init } i \ a) - (\sum i \in \{i. \ i < \text{length } qs \land qs ! i = b\} \cap \{i. \ i < \text{length } qs \land qs ! i = a\}. \ \text{ALG'} A \ \text{qs init } i \ b + \text{ALG'} A \ \text{qs init } i \ a) \]
apply (rule sum_Un)
by (auto)
also have \[ \ldots = (\sum i \in \{i. \ i < \text{length } qs \land qs ! i = b\}. \ \text{ALG'} A \ \text{qs init } i \ b + \text{ALG'} A \ \text{qs init } i \ a) + (\sum i \in \{i. \ i < \text{length } qs \land qs ! i = a\}. \ \text{ALG'} A \ \text{qs init } i \ b + \text{ALG'} A \ \text{qs init } i \ a) \]
by (auto)
ALG' A qs init i a) using disj by auto
also have \( \ldots = (\sum i \in \{ i. \ i < \text{length} \ qs \land qs ! i = b \}. \ ALG' A qs \ init \ i \ a) \)
+ \( (\sum i \in \{ i. \ i < \text{length} \ qs \land qs ! i = a \}. \ ALG' A qs \ init \ i \ b) \)
by (auto simp: ALG'_refl)
finally
show ?case unfolding x apply(simp add: split_def)
unfolding unio by simp
qed
also have E6: \( \ldots = (\sum (x,y) \in \{(x,y). \ x \in \text{set init} \land y \in \text{set init} \land x < y \}. \ ALGxy A qs \ init \ x \ y) \)
unfolding ALGxy_def2 by simp
finally show ?thesis.

qed

lemma before_in_index1:
fixes l
assumes set l = \{x,y\} and length l = 2 and x\neq y
shows (if (x < y in l) then 0 else 1) = index l x
unfolding before_in_def
proof (auto, goal_cases)
  case 1
  from assms(1) have index l y < length l by simp
  with assms(2) 1(1) show index l x = 0 by auto
  next
  case 2
  from assms(1) have a: index l x < length l by simp
  from assms(1,3) have index l y \neq index l x by simp
  with assms(2) 2(1) a show Suc 0 = index l x by simp
  qed (simp add: assms)

lemma before_in_index2:
fixes l
assumes set l = \{x,y\} and length l = 2 and x\neq y
shows (if (x < y in l) then 1 else 0) = index l y
unfolding before_in_def
proof (auto, goal_cases)
  case 2
  from assms(1,3) have a: index l y \neq index l x by simp
  from assms(1) have index l x < length l by simp
  with assms(2) a 2(1) show index l y = 0 by auto
  next

201
case 1
from assms(1) have a: index l y < length l by simp
from assms(1,3) have index l y ≠ index l x by simp
with assms(2) l(1) a show Suc 0 = index l y by simp
qed (simp add: assms)

lemma before_in_index:
fixes l
assumes set l = {x,y} and length l = 2 and x≠y
shows (x < y in l) = (index l x = 0)
unfolding before_in_def
proof (safe, goal_cases)
  case 1
  from assms(1) have index l y < length l by simp
  with assms(2) l(1) show index l x = 0 by auto
next
  case 2
  from assms(1,3) have index l y ≠ index l x by simp
  with 2(1) show index l x < index l y by simp
qed (simp add: assms)

14.3 The pairwise property

definition pairwise where
pairwise A = (∀init. distinct init → (∀qs∈{xs. set xs ⊆ set init}. ∀(x::('a::linorder),y)∈{(x,y). x ∈ set init ∧ y∈set init ∧ x<y}. T_on_rand A (Lxy init {x,y}) (Lxy qs {x,y}) = ALGxy A qs init x y))

definition Pbefore_in x y A qs init = map_pmf (λp. x < y in fst p) (config_rand A init qs)

lemma T_on_n_no_paid:
assumes
  nopaid: ∀s n. map_pmf (λx. snd (fst x)) (snd A s n) = return_pmf []
schows T_on_rand_n A init qs i = E (config" A qs init i ≻= (λp. return_pmf (real(index (fst p) (qs ! i))))))
proof –

have (λs. snd A s (qs ! i) ≻= (λ(a, is'). return_pmf (real (t_p (fst s) (qs ! i) a))))
  = (λs. snd A s (qs ! i) ≻= (λx. return_pmf (snd (fst x)))))
\[
\Rightarrow (\lambda p. \text{return pmf} \\
(\text{real (index (swaps p (fst s)) (qs ! i))) } + \\
\text{real (length p)}))
\]

by (simp add: tpm_def split_def bind_return_pmf bind_assoc_pmf)

also

have \ldots = (\lambda p. \text{return pmf} (\text{real (index (fst p) (qs ! i))})))

using nopaid[unfolded map_pmf_def]

by (simp add: split_def bind_return_pmf)

finally

show ?thesis by simp

qed

lemma pairwise_property_lemma:

assumes

relativeorder: (\forall init qs. distinct init \Rightarrow qs \in \{xs. set xs \subseteq set init\} \\
\Rightarrow (\forall x y. (x,y) \in \{(x,y). x \in set init \land y \in set init \land x \neq y\} \\
\Rightarrow x \neq y \\
\Rightarrow \text{Pbefore_in x y A qs init = Pbefore_in x y A (Lxy qs \{x,y\})} \\
(Lxy init \{x,y\}) \\
))

and nopaid: \(\forall x a r. \forall z \in set_pmf(snd A xa r). \text{snd}(fst z) = []\)

shows pairwise A

unfolding pairwise_def

proof (clarify, goal_cases)

case (1 init rs x y)

then have xny: \(x \neq y\) by auto

note dinit=1(1)

then have dLxy: distinct (Lxy init \{x,y\}) by (rule Lxy_distinct)

from dinit have dLxy: distinct (Lxy init \{x,y\}) by (rule Lxy_distinct)

have setLxy: set (Lxy init \{x,y\}) = \{x,y\} apply (subst Lxy_set_filter)

using 1 by auto

have setLxy: set (Lxy init \{x,y\}) = \{x,y\} apply (subst Lxy_set_filter)

using 1 by auto

have lengthLxy: length (Lxy init \{y,x\}) = 2 using setLxy distinct_card[OF dLxy] xny by simp

have lengthLxy: length (Lxy init \{x,y\}) = 2 using setLxy distinct_card[OF dLxy] xny by simp

have aee: \(\{x,y\} = \{y,x\}\) by auto

from 1(2) show ?case

proof (induct rs rule: rev_induct)

case (snoc r rs)
have \( b : \text{Pbefore}_\text{in} x y A \text{ rs init} = \text{Pbefore}_\text{in} x y A \ (Lxy \text{ rs } \{x,y\}) \)

apply (rule relativeorder)

using snoc 1 xny by (simp_all)

show ?case (is \( ?L \ (\text{rs} @ [r]) = ?R \ (\text{rs} @ [r]) \))

proof (cases \( r \in \{x, y\} \))

  case True

  note xyrequest = this

  let \( ?\text{expr} = E \ (\text{Partial}_\text{Cost}_\text{Model}.\text{config}_\text{'} \text{rand} A \ (\text{fst} A \ (Lxy \text{ init } \{x, y\})) \gg \ (\lambda \text{is}. \text{return}_\text{pmf} \ (Lxy \text{ init } \{x, y\}, \text{is}))) \ (Lxy \text{ rs } \{x, y\}) \gg \ (\lambda s. \text{snd} A \ s \ r \gg \ (\lambda (a, is'). \text{return}_\text{pmf} \ (\text{real} (tp (\text{fst} s \ r \ a))))) \)

  let \( ?\text{expr}2 = \text{ALG}_\text{'} A \ (\text{rs} @ [r]) \text{ init } (\text{length} \text{ rs}) \ y + \text{ALG}_\text{'} A \ (\text{rs} @ [r]) \text{ init } (\text{length} \text{ rs}) \ x \)

  from \( xyrequest \) have \( ?L \ (\text{rs} @ [r]) = ?L \text{ rs} + ?\text{expr} \)

  by (simp add: Lxy_snoc T_on_rand'_append)

  also have \( \ldots = ?L \text{ rs} + ?\text{expr}2 \)

  proof (cases \( r = x \))

    case True

    let \( ?\text{projS} = \text{config}_\text{'} \text{rand} A \ (\text{fst} A \ (Lxy \text{ init } \{x, y\})) \gg \ (\lambda s. \text{snd} A \ s \ r \gg \ (\lambda (a, is'). \text{return}_\text{pmf} \ (\text{real} (\text{index} (\text{fst} s \ r))))) \)

    let \( ?S = (\text{config}_\text{'} \text{rand} A \ (\text{fst} A \text{ init} \gg \ (\lambda s. \text{return}_\text{pmf} \ (\text{init}, \text{is}))) \text{ rs}) \)

  have ?\text{projS} \gg \ (\lambda s. \text{snd} A \ s \ r \gg \ (\lambda (a, is'). \text{return}_\text{pmf} \ (\text{real} (tp (\text{fst} s \ r \ a))))) \)

  = ?\text{projS} \gg \ (\lambda s. \text{return}_\text{pmf} \ (\text{real} (\text{index} (\text{fst} s \ r)))))

  proof (rule bind_pmf_cong, goal_cases)

    case \( r \in z \)

    have \( \text{snd} A \ z \ r \gg \ (\lambda (a, is'). \text{return}_\text{pmf} \ (\text{real} (tp (\text{fst} z \ r \ a)))) \)

    = \( \text{snd} A \ z \ r \gg \ (\lambda x. \text{return}_\text{pmf} \ (\text{real} (\text{index} (\text{fst} z \ r))))) \)

    apply (rule bind_pmf_cong)

    apply (simp)

    using nopaid \{of z r\} by (simp add: split_def tp_def)

  then show ?case by (simp add: bind_return_pmf)

  qed simp
also have \ldots = \text{map\_pmf} (\%b. (if b then 0::real else 1)) \ (P\text{before\_in} \\
x\ y\ A\ (Lxy\ rs\ \{x,y\})\ (Lxy\ init\ \{x,y\}))

unfolding \ P\text{before\_in\_def}\ \text{map\_pmf\_def}
apply(simp add: \ bind\_return\_pmf\ \text{bind\_assoc\_pmf})
apply(rule \ bind\_pmf\_cong)
apply(simp add: \ ace)
proof goal_cases
  case (1 z)
  have (if \ x < \ y \ in \ fst\ z \ then 0 \ else 1) = (index (fst z) \ x)
  apply(rule \ \text{before\_in\_index1})
  using 1 \ \text{config\_rand\_set\ \text{setLxy}\ \text{apply\ fast}}
  using 1 \ \text{config\_rand\_length\ \text{lengthLxy}\ \text{apply\ metis}}

  using \text{nxy\ by\ simp}
  with True show ?case
  by(auto)
qed

also have \ldots = \text{map\_pmf} (\%b. (if b then 0::real else 1)) \ (P\text{before\_in} \\
x\ y\ A\ rs\ init) \ by(simp add: \ b)

also have \ldots = \text{map\_pmf} (\lambda xa. \ real\ (if \ y < \ x \ in \ fst\ xa \ then \ 1 \ else \ 0)) \ ?S
apply(simp add: P\text{before\_in\_def}\ \text{map\_pmf\_comp})
proof (rule \ text{map\_pmf\_cong, goal\_cases})
  case (2 z)
  then have set_z: set (fst z) = set init
  using \text{config\_rand\_set\ \text{by\ fast}}
  have (\neg \ x < \ y \ in \ fst\ z) = \ y < \ x \ in \ fst\ z
  apply(subst \ not\text{before\_in})
  using set_z 1(3,4) \text{nxy\ by(simp\_all)}
  then show ?case \ by(simp add: )
qed \text{simp}

finally have \ a: \ ?\text{projS} \gg (\lambda s. \ \text{snd\ A\ s\ x}
\gg (\lambda (a,\ is'). \ \text{return\_pmf} \ (\text{real\ (t\text{p} (fst\ s)\ x\ a))))
= \text{map\_pmf} (\lambda xa. \ real\ (if \ y < \ x \ in \ fst\ xa \ then \ 1\ else \ 0)) \ ?S
using True \ by \ simp
from True show \ ?thesis
apply(simp add: \text{ALG}\_\prime\_refl\ \text{nth\_append})
unfolding \text{ALG}\_\prime\_def
by(simp add: \ a)
next
  case False
  with \text{xyrequest\ have request: \ r=y\ by\ blast}

205
let projS = config'_rand A (fst A (Lxy init \{x, y\})) \Rightarrow (\lambda is. return_pmf (Lxy init \{x, y\}, is))) (Lxy rs \{x, y\})
let \(?S = (config'_rand A (fst A init \Rightarrow (\lambda is. return_pmf (init, is))) rs)

have projS \Rightarrow (\lambda s. snd A s r)
  \Rightarrow (\lambda is. return_pmf (real (tp (fst s) r a))))
proof (rule bind_pmf_cong, goal_cases)
  case (2 z)
    have snd A z r \Rightarrow (\lambda is. return_pmf (real (tp (fst z) r a)))
      apply (rule bind_pmf_cong)
      apply (simp)
      using nopaid[of z r] by (simp add: split_def tp_def)
    then show \(\text{case by (simp add: bind_return_pmf)}\)
    qed simp
  also have \(\ldots = map_pmf (\%b. (if b then 1::real else 0)) \text{ (Pbefore_in x y A (Lxy rs \{x,y\}) (Lxy init \{x,y\})})\)
    unfolding Pbefore_in_def map_pmf_def
    apply (simp add: bind_return_pmf bind_assoc_pmf)
    apply (rule bind_pmf_cong)
    apply (simp add: aee)
    proof goal_cases
      case (1 z)
      have \((if x < y in fst z then 1 else 0) = (index (fst z) y)\)
        apply (rule before_in_index2)
        using 1 config_rand_set setLxy apply fast
        using 1 config_rand_length lengthLxy apply metis
      using xny by simp
      with request show \(\text{case by (auto)}\)
      qed
  also have \(\ldots = map_pmf (\%b. (if b then 1::real else 0)) \text{ (Pbefore_in x y A rs init) by (simp add: b)}\)
also have \(\ldots = map_pmf (\lambda xa. real (if x < y in fst xa then 1 else 0)) \text{ ?S}\)
apply (simp add: Pbefore_in_def map_pmf_comp)
apply (rule map_pmf_cong) by simp_all
finally have a: projS \Rightarrow (\lambda s. snd A s y
  \Rightarrow (\lambda(a, is'). return_pmf (real (tp (fst s) y a))))
= map pmf (λxa. real (if x < y in fst xa then 1 else 0)) ?S

using request by simp
from request show ?thesis
apply(simp add: ALG’ refl nth_append)
unfolding ALG’ def
by(simp add: a)

qed
also have … = ?R rs + ?expr2 using snoc by simp
also from True have … = ?R (rs@[r])
apply(subst ALGxy_append) by(auto)
finally show ?thesis .

next

case False
then have ?L (rs@[r]) = ?L rs apply(subst Lxy_snoc) by simp
also have … = ?R rs using snoc by(simp)
also have … = ?R (rs@[r])
apply(subst ALGxy_append) using False by(simp)
finally show ?thesis .

qed

lemma umf_pair: assumes
  0: pairwise A
  assumes 1: ∀(is s q). ((free, paid),_) ∈ (snd A (s, is) q). paid=[]
  assumes 2: set qs ⊆ set init
  assumes 3: distinct init
  assumes 4: ∀x. x<length qs ⇒ finite (set pmf (config'' A qs init x))
  shows T_p_on_rand A init qs
  = (∑(x,y)∈{(x, y). x ∈ set init ∧ y ∈ set init ∧ x < y}. T_p_on_rand A (Lxy init {x,y}) (Lxy qs {x,y})))
proof –
  have T_p_on_rand A init qs = (∑(x,y)∈{(x, y). x ∈ set init ∧ y ∈ set init ∧ x < y}. ALGxy A qs init x y)
  by(simp only: umformung[OF 1 2 3 4])
  also have … = (∑(x,y)∈{(x, y). x ∈ set init ∧ y ∈ set init ∧ x < y}. T_p_on_rand A (Lxy init {x,y}) (Lxy qs {x,y})))
  apply(rule sum.cong)
  apply(simp)
  using 0[unfolded pairwise_def] 2 3 by auto
finally show ?thesis .

qed
14.4 List Factoring for OPT

fun \texttt{ALG\_P :: nat list }\Rightarrow\texttt{ (a \Rightarrow \texttt{ a list }\Rightarrow \texttt{ nat where}}
\texttt{ALG\_P [] x y xs = (0::nat)}
\texttt{| ALG\_P (s#ss) x y xs = (if Suc s < length (swaps ss xs)}
\texttt{then (if ((swaps ss xs)!s=x \land (swaps ss xs)!Suc s=y)}
\texttt{\lor ((swaps ss xs)!s=y \land (swaps ss xs)!Suc s=x)}
\texttt{then 1 \else 0)\else 0) + ALG\_P ss x y xs}}

lemma \texttt{ALG\_P\_erwischt\_alle:}
\texttt{ assumes dinit: distinct init}
\texttt{ shows \forall l< length sws. Suc (sws!l) < length init \implies length sws}
\texttt{ = (\sum (x,y)\in\{ (x,y) \cdot x \in set (init::(a::linorder) list) \land y\in set init \land x\land y\}. ALG\_P sws x y init)}
\texttt{ proof (induct sws)}
\texttt{ case (Cons s ss)}
\texttt{ then have isininit: Suc s < length init by auto}
\texttt{ from Cons have \forall l< length ss. Suc (ss ! l) < length init by auto}
\texttt{ note iH=Cons(1)|OF this}.
\texttt{ let \texttt{expr = (\lambda x y. (if Suc s < length (swaps ss init)}}
\texttt{ then (if ((swaps ss init)!s=x \land (swaps ss init)!Suc s=y)}
\texttt{ \lor ((swaps ss init)!s=y \land (swaps ss init)!Suc s=x)}
\texttt{ then 1::nat \else 0) \else 0))}.
\texttt{ let \texttt{expr2 = (\lambda x y. (if ((swaps ss init)!s=x \land (swaps ss init)!Suc s=y)}
\texttt{ \lor ((swaps ss init)!s=y \land (swaps ss init)!Suc s=x)}
\texttt{ then 1 \else 0))}.
\texttt{ let \texttt{expr3 = (%x y. ((swaps ss init)!s=x \land (swaps ss init)!Suc s=y)}
\texttt{ \lor ((swaps ss init)!s=y \land (swaps ss init)!Suc s=x)}
\texttt{ let \texttt{co’ = swaps ss init}}
\texttt{ from dinit have dco: distinct ?co’ by auto}
\texttt{ let \texttt{expr4 = (\lambda z. (if z\in\{(x,y). \texttt{expr3 x y}}
\texttt{ then 1}
have $sco\text{init}$: set $\exists co' = set\ init$ by auto
from $isin\text{init}$ have $isT$: $Suc\ s < length\ ?co'$ by auto
then have $isT2$: $Suc\ s < length\ init$ by auto
then have $isT3$: $s < length\ init$ by auto
then have $isT6$: $s < length\ ?co'$ by auto
from $isT2$ have $isT7$: $Suc\ s < length\ ?co'$ by auto
from $isT6$ have $a$: $?co'\ !s \in set \ ?co'$ by (rule nth_mem)
then have $a$: $?co'\ !s \in set\ init$ by auto
from $isT7$ have $?co'\ (! (Suc\ s) \in set \ ?co')$ by (rule nth_mem)
then have $b$: $?co'\ !((Suc\ s) \in set \ ?co')$ by auto

have $\{(x,y).\ x \in set\ init \land y \in set\ init \land x<y\}$
$\cap \{(x,y).\ ?expr3\ x\ y\}$
$= \{(x,y).\ x \in set\ init \land y \in set\ init \land x<y$
$\land (\ ?co'!s=x \land ?co'!(Suc\ s)=y$
$\lor \ ?co'!s=y \land ?co'!(Suc\ s)=x)\}$ by auto
also have $\ldots = \{(x,y).\ x \in set\ init \land y \in set\ init \land x<y$
$\land \ ?co'!s=x \land ?co'!(Suc\ s)=y\}$
$\cup \{(x,y).\ x \in set\ init \land y \in set\ init \land x<y$
$\land \ ?co'!s=y \land ?co'!(Suc\ s)=x\}$ by auto
also have $\ldots = \{(x,y).\ x \in set\ init \land y \in set\ init \land x<y$
$\land \ ?co'!s=x \land ?co'!(Suc\ s)=y\}$
$\cup \{(x,y).\ x \in set\ init \land y \in set\ init \land x<y$
$\land \ ?co'!s=y \land ?co'!(Suc\ s)=x\}$ by auto

using $a\ b$ by (auto)

finally have $c1$: $\{(x,y).\ x \in set\ init \land y \in set\ init \land x<y\}$
$\cap \{(x,y).\ ?expr3\ x\ y\}$
$= \{(x,y).\ x \in set\ init \land y \in set\ init \land x<y$
$\land \ ?co'!s=x \land ?co'!(Suc\ s)=y\}$
$\cup \{(x,y).\ x \in set\ init \land y \in set\ init \land x<y$
$\land \ ?co'!s=y \land ?co'!(Suc\ s)=x\}$. 

have $c2$: card $\{(x,y).\ x \in set\ init \land y \in set\ init \land x<y$
$\land \ ?co'!s=x \land ?co'!(Suc\ s)=y\}$
$\cup \{(x,y).\ x \in set\ init \land y \in set\ init \land x<y$
$\land \ ?co'!s=y \land ?co'!(Suc\ s)=x\} = 1$ (is card
$\forall A \cup \ ?B = 1$)
proof (cases $?co'!s < ?co'!(Suc\ s)$)
  case True
  then have $a$: $?A = \{(?co'!s, ?co'!(Suc\ s))\}$
  and $b$: $?B = \{\}$ by auto
  have $c$: $?A \cup ?B = \{(?co'!s, ?co'!(Suc\ s))\}$ by simp
  have $\text{card} (\forall A \cup \ ?B = 1$ unfolding $c$ by auto

209
then show \( ?\text{thesis} \).

next

\begin{enumerate}
\item \textit{case False}
\item have \( a: ?A = \{ \} \) by auto \\
\item have \( b: ?B = \{ (?co'! (Suc \ s), ?co'! s) \} \)
\item proof \\
\item from dco distinct_conv_nth[of \( ?co' \)] \\
\item have \( \text{swaps ss init} \neq \text{swaps ss init} \ (Suc \ s) \)
\item using isT2 isT3 by simp \\
\item with False show \( ?\text{thesis} \) by auto
\item qed \\
\item have \( c: ?A \cup ?B = \{ (?co'! (Suc \ s), ?co'! s) \} \) apply(simp only: \( a \ b \)) by simp \\
\item have \( \text{card} \ ( ?A \cup ?B ) = 1 \) unfolding \( c \) by auto \\
\item then show \( ?\text{thesis} \).
\item qed
\end{enumerate}

have \( \text{yeah}: ( \sum (x,y) \in \{ (x,y). x \in \text{set init} \land y \in \text{set init} \land x<y \} ). ?\text{expr} x y = (1::\text{nat}) \)

proof \\
\begin{enumerate}
\item have \( ( \sum (x,y) \in \{ (x,y). x \in \text{set init} \land y \in \text{set init} \land x<y \} ). ?\text{expr} x y \)
\item = \( ( \sum (x,y) \in \{ (x,y). x \in \text{set init} \land y \in \text{set init} \land x<y \} ). ?\text{expr} x y \)
\item using \( \text{isT} \) by auto \\
\item also have \( \ldots = ( \sum z \in \{ (x,y). x \in \text{set init} \land y \in \text{set init} \land x<y \} ). ?\text{expr} x y \)
\item by(simp add: split_def) \\
\item also have \( \ldots = ( \sum z \in \{ (x,y). x \in \text{set init} \land y \in \text{set init} \land x<y \} ). ?\text{expr} x y \)
\item by(simp add: split_def) \\
\item also have \( \ldots = ( \sum z \in \{ (x,y). x \in \text{set init} \land y \in \text{set init} \land x<y \} \) \\
\item \cap \{ (x,y). ?\text{expr} x y \} . 1 \)
\item apply(rule sum_inter_restrict[symmetric]) \\
\item apply(rule finite_subset[where \( B=\text{set init} \times \text{set init} \)]) \\
\item by(auto) \\
\item also have \( \ldots = \text{card} \ ( ( \sum (x,y). x \in \text{set init} \land y \in \text{set init} \land x<y ) \\
\item \cap \{ (x,y). ?\text{expr} x y \} ) \) by auto \\
\item also have \( \ldots = \text{card} \ ( ( \sum (x,y). x<y \land ?co'! s=x \land ?co'! (Suc \ s)=y ) \\
\item \cup \{ (x,y). x<y \land ?co'! s=x \land ?co'! (Suc \ s)=y \} ) \) by(simp only: \( c1 \)) \\
\item also have \( \ldots = (1::\text{nat}) \) using \( c2 \) by auto \\
\end{enumerate}
finally show ?thesis .

qed

have length (s ≠ ss) = 1 + length ss
  by auto
also have ... = 1 + (∑ (x,y)∈{(x,y). x ∈ set init ∧ y ∈ set init ∧ x<y}. ALG_P ss x y init)
    using iH by auto
also have ... = (∑ (x,y)∈{(x,y). x ∈ set init ∧ y ∈ set init ∧ x<y}. ?expr x y)
  + (∑ (x,y)∈{(x,y). x ∈ set init ∧ y ∈ set init ∧ x<y}. ALG_P ss x y init)
    by (simp only: yeah)
also have ... = (∑ (x,y)∈{(x,y). x ∈ set init ∧ y ∈ set init ∧ x<y}. ?expr x y + ALG_P ss x y init)
  (is ?A + ?B = ?C)
    by (simp add: sum.distrib split_def)
also have ... = (∑ (x,y)∈{(x,y). x ∈ set init ∧ y ∈ set init ∧ x<y}. ALG_P (s#ss) x y init)
    by auto
finally show ?case .

qed (simp)

lemma t_p_sumofALGALG_P:
assumes distinct s (qs!i) ∈ set s
  and ∀ l < length (snd a). Suc ((snd a)!l) < length s
shows t_p s (qs!i) a = (∑ e ∈ set s. ALG e qs i (swaps (snd a) s.()})
  + (∑ (x,y)∈{(x::('a::linorder),y). x ∈ set s ∧ y ∈ set s ∧ x<y}. ALG_P (snd a) x y s)
proof —

  have pe: length (snd a)
        = (∑ (x,y)∈{(x,y). x ∈ set s ∧ y ∈ set s ∧ x<y}. ALG_P (snd a) x y s)
      apply (rule ALG_P_erwischt_alle)
      by (fact+)

  have ac: index (swaps (snd a) s) (qs ! i) = (∑ e ∈ set s. ALG e qs i (swaps (snd a) s.()})

211
proof

  have index \((\text{swaps (snd a)} \ s) (qs \ i)\)
    \(= (\sum e \in \text{set (swaps (snd a)} s) \ . \ if \ e < (qs \ i) \ in \ (swaps (snd a)} s) \ then \ 1 \ else \ 0)\)
    apply(rule index_sum)
    using assms by(simp_all)
  also have \(\ldots = (\sum e \in \text{set s. ALG e qs i (swaps (snd a)} s,.))\) by auto
  finally show \(?thesis\).
qed

show \(?thesis\)
  unfolding t_p_def apply (simp add: split_def)
  unfolding ac pe by (simp add: split_def)
qed

definition \(\text{ALG}\_P'\ Strat qs init i x y = ALG\_P (\text{snd (Strat!i)}) x y (\text{steps'} init qs Strat i)\)

lemma \(\text{ALG}\_P'\ rest: n < \text{length qs} \implies n < \text{length Strat} \implies\)

  \(\text{ALG}\_P'\ Strat (\text{take n qs @ [qs ! n]}) init n x y =\)
  \(\text{ALG}\_P' (\text{take n Strat @ [Strat ! n]} (\text{take n qs @ [qs ! n]}) init n x y\)

proof

  assume qs: \(n < \text{length qs}\)
  assume S: \(n < \text{length Strat}\)

  then have lS: \(\text{length (take n Strat} = n\) by auto
  have \((\text{take n Strat @ [Strat ! n]} ! n =\)
    \((\text{take n Strat @ [Strat ! n]} \# [])! \text{length (take n Strat} \) using lS by auto
  also have \(\ldots = \text{Strat ! n by(rule nth_append_length}\)
  finally have it: \((\text{take n Strat @ [Strat ! n]} ! n = \text{Strat ! n} .\)

  obtain rest where rest: \(\text{Strat} = (\text{take n Strat @ [Strat ! n]} \# \text{rest}\)
    using S apply(auto) using id_take_nth_drop by blast

  have steps' init \((\text{take n qs @ [qs ! n]}\)
    \((\text{take n Strat @ [Strat ! n]} ! n =\)
    \((\text{take n Strat) n}
    \) apply(rule steps'_rests[symmetric])
    using S qs by auto

  212
also have \ldots =
\begin{align*}
steps' \init (\take n \qs @ [\qs ! n]) \\
& (\take n \Strat @ ([\Strat ! n] @ \rest)) n \\
& \apply (\rule steps'_rests) \\
& \using S \qs \by \auto
\end{align*}

finally show \thesis unfolding \textit{ALG\textsubscript{P}'\_def tt using rest by auto}

\begin{align*}
\textbf{lemma } \textit{ALG\textsubscript{P}'\_rest2}: n < \text{length} \qs \implies n < \text{length} \Strat \implies \\
& \textit{ALG\textsubscript{P}'(Strat@r1)} (qs@r2) \init n x y \\
\textbf{proof } -
\begin{align*}
& \textbf{assume} \ qs: n < \text{length} \qs \\
& \textbf{assume} \ S: n < \text{length} \Strat \\
& \textbf{have} \ tt: Strat ! n = (Strat @ r1) ! n \\
& \textbf{using} S \by (\simp \add: nth\_append)
\end{align*}

have steps' \init (\take n \qs) (\take n \Strat) n = steps' \init ((\take n \qs) @ \\
& \drop n \qs) ((\take n \Strat) @ (\drop n \Strat)) n \\
& \apply (\rule steps'_rests) \\
& \using S \qs \by \auto
\textbf{then have} A: steps' \init (\take n \qs) (\take n \Strat) n = steps' \init \qs \Strat \by \auto

have steps' \init (\take n \qs) (\take n \Strat) n = steps' \init ((\take n \qs) @ \\
& ((\drop n \qs)@r2)) ((\take n \Strat) @((\drop n \Strat)@r1)) n \\
& \apply (\rule steps'_rests) \\
& \using S \qs \by \auto
\textbf{then have} B: steps' \init (\take n \qs) (\take n \Strat) n = steps' \init (qs@r2) \\
& (Strat@r1) n \\
& \by (metis append\_assoc List.append\_take\_drop\_id) \\
\textbf{from} A B \textbf{have} C: steps' \init \qs \Strat \by \auto

show \thesis unfolding \textit{ALG\textsubscript{P}'\_def tt using C by auto}

\begin{align*}
\textbf{qed}
\end{align*}

\begin{align*}
\textbf{definition } \textit{ALG\textsubscript{P}xy } \textbf{where}
\textit{ALG\textsubscript{P}xy \ Strat qs init x y } = (\sum i < \text{length} \qs. \textit{ALG\textsubscript{P}' \ Strat qs init i x y})
\end{align*}

213
lemma wegdamit: length $A < \text{length} \ Strat \Rightarrow b \notin \{x,y\} \Rightarrow ALG_{xy\det} \\
Strat \ (A @ [b]) \ init \ x \ y \\
= ALG_{xy\det} \ Strat \ A \ init \ x \ y \\

proof – \\
assume bn: $b \notin \{x,y\}$ \\
have $(A @ [b]) ! (\text{length} \ A) = b$ by auto \\
assume l: length $A < \text{length} \ Strat \\

term $\exists i. \ ALG'_{\det} \ Strat \ (A @ [b]) \ init \ i \ y \\

have $e: \bigwedge i. \ i < \text{length} \ A \Rightarrow (A @ [b]) ! i = A ! i$ by(auto simp: nth_append) \\
have $(\sum i \in \{..< (\text{length} \ (A)\})$. \\
if $(A @ [b]) ! i \in \{y, x\}$ \\
then $ALG'_{\det} \ Strat \ (A @ [b]) \ init \ i \ y$ \\
$ALG'_{\det} \ Strat \ (A @ [b]) \ init \ i \ x$ \\
else $0 = (\sum i \in \{..< \text{Suc(\text{length} \ A)\}).$ \\
if $(A @ [b]) ! i \in \{y, x\}$ \\
then $ALG'_{\det} \ Strat \ (A @ [b]) \ init \ i \ y$ \\
$ALG'_{\det} \ Strat \ (A @ [b]) \ init \ i \ x$ \\
else $0)$ by auto \\
also have $\ldots = (\sum i \in \{..< (\text{length} \ (A)\}).$. \\
if $(A @ [b]) ! i \in \{y, x\}$ \\
then $ALG'_{\det} \ Strat \ (A @ [b]) \ init \ i \ y$ \\
$ALG'_{\det} \ Strat \ (A @ [b]) \ init \ i \ x$ \\
else $0)$ by simp \\
also have $\ldots = (\sum i \in \{..< (\text{length} \ (A)\}).$. \\
if $(A @ [b]) ! i \in \{y, x\}$ \\
then $ALG'_{\det} \ Strat \ (A @ [b]) \ init \ i \ y$ \\
$ALG'_{\det} \ Strat \ (A @ [b]) \ init \ i \ x$ \\
else $0)$ using bn by auto \\
also have $\ldots = (\sum i \in \{..< (\text{length} \ (A)\}).$. \\
if $A ! i \in \{y, x\}$ \\
then $ALG'_{\det} \ Strat \ A \ init \ i \ y$ \\
$ALG'_{\det} \ Strat \ A \ init \ i \ x$ \\
else $0)$ \\
apply(rule sum.cong) \\
apply(simp) \\
using $I \ ALG'_{\det} \ append$[where $qs=A] e$ by(simp) \\
finally show $\exists \text{thesis unfolding} \ ALG_{xy\det\_def \ by \ simp}$ \\

qed
lemma \textit{ALG\_P\_split}: \text{length } qs < \text{length } Strat \implies \text{ALG\_P\_xy} \text{ Strat} \ (qs \otimes [q])

\text{init} \ x \ y = \text{ALG\_P\_xy} \text{ Strat} \ qs \text{ init} \ x \ y
+ \text{ALG\_P'} \text{ Strat} \ (qs \otimes [q]) \text{ init} \ (\text{length } qs) \ x \ y

unfolding \text{ALG\_P\_xy} \text{ def} \ \text{apply} \ (\text{auto})
apply \ (\text{rule sum.cong})
apply \ (\text{simp})
using \text{ALG\_P'} \text{ rest2} \ (\text{symmetric, of } qs \text{ Strat} \ [] \ [q]) \ \text{by} \ (\text{simp})

lemma \textit{swap0in2}: \text{assumes} \ \text{set} \ l = \{x,y\} \ x \neq y \ \text{length} \ l = 2 \ \text{dist_perm} \ l \ l
\text{shows} \ x < y \ \text{in} \ \text{swap} \ 0 \ l = (\neg x < y \ \text{in} \ l)
proof \ (\text{cases} \ x < y \ \text{in} \ l)
\begin{itemize}
\item \text{case} \ True
\begin{itemize}
\item \text{then have} \ a: \text{index} \ l \ x < \text{index} \ l \ y \ \text{unfolding} \ \text{before\_in\_def} \ \text{by} \ \text{simp}
from \ \text{assms}(1) \ \text{have} \ \text{drin}: \ x \in \text{set} \ l \ y \in \text{set} \ l \ \text{by} \ \text{auto}
from \ \text{assms}(1,3) \ \text{have} \ b: \text{index} \ l \ y < 2 \ \text{by} \ \text{simp}
from \ a \ b \ \text{have} \ k: \text{index} \ l \ x = 0 \ \text{index} \ l \ y = 1 \ \text{by} \ \text{auto}
\end{itemize}
\item \text{have} \ g: x = l ! 0 \ y = l ! 1
\text{using} \ k \ \text{nth\_index} \ \text{assms}(1) \ \text{by} \ \text{force}+
\begin{itemize}
\item \text{have} \ x < y \ \text{in} \ \text{swap} \ 0 \ l
\begin{itemize}
\item \ (x < y \ \text{in} \ l \ \land \ \neg (x = l ! 0 \ \land \ y = l ! \ Suc 0)
\lor \ x = l ! \ Suc 0 \ \land \ y = l ! 0)
\item \ \text{apply} \ (\text{rule before\_in\_swap})
\item \ \text{apply} \ (\text{fact} \ \text{assms}(4))
\item \ \text{using} \ \text{assms}(3) \ \text{by} \ \text{simp}
\end{itemize}
\end{itemize}
\item \text{also have} \ \ldots = (\neg (x = l ! 0 \ \land \ y = l ! \ Suc 0)
\lor \ x = l ! \ Suc 0 \ \land \ y = l ! 0) \ \text{using} \ True \ \text{by} \ \text{simp}
\item \text{also have} \ \ldots = \text{False} \ \text{using} \ g \ \text{assms}(2) \ \text{by} \ \text{auto}
\end{itemize}
finally have \ \neg x < y \ \text{in} \ \text{swap} \ 0 \ l \ \text{by} \ \text{simp}
\text{then show} \ \neg \text{thesis} \ \text{using} \ True \ \text{by} \ \text{auto}
\end{itemize}
next
\begin{itemize}
\item \text{case} \ False
\item \text{from} \ \text{assms}(1,2) \ \text{have} \ \text{index} \ l \ y \neq \ \text{index} \ l \ x \ \text{by} \ \text{simp}
with \ \text{False} \ \text{assms}(1,2) \ \text{have} \ a: \ \text{index} \ l \ y < \ \text{index} \ l \ x
\text{by} \ (\text{metis} \ \text{before\_in\_def} \ \text{insert\_iff} \ \text{linorder\_neqE\_nat})
\item \text{from} \ \text{assms}(1) \ \text{have} \ \text{drin}: \ x \in \text{set} \ l \ y \in \text{set} \ l \ \text{by} \ \text{auto}
\item \text{from} \ \text{assms}(1,3) \ \text{have} \ b: \ \text{index} \ l \ x < 2 \ \text{by} \ \text{simp}
\item \text{from} \ a \ b \ \text{have} \ k: \ \text{index} \ l \ x = 1 \ \text{index} \ l \ y = 0 \ \text{by} \ \text{auto}
\item \text{then have} \ g: x = l ! 1 \ y = l ! 0
\item \ \text{using} \ k \ \text{nth\_index} \ \text{assms}(1) \ \text{by} \ \text{force}+
\item \text{have} \ x < y \ \text{in} \ \text{swap} \ 0 \ l
\end{itemize}
\[(x < y \text{ in } l \land \neg (x = l \cdot 0 \land y = l \cdot Suc 0)) \land (x = l \cdot Suc 0 \land y = l \cdot 0)\]

apply (rule before_in_swap)
apply (fact assms(4))
using assms(3) by simp
also have \ldots = (x = l \cdot Suc 0 \land y = l \cdot 0) using False by simp
also have \ldots = True using g by auto
finally have \ldots \quad \text{by simp}
then show \?thesis using False by auto
qed

lemma before_in_swap2:
\begin{align*}
dist \text{ _perm } xs \ y s \Longrightarrow & \quad Suc \ n < size \ x s \Longrightarrow x \neq y \Longrightarrow \\
x < y \ \text{ in } (swap \ n \ x s) \iff \\
(\sim x < y \ \text{ in } xs \land (y = xs!n \land x = xs!Suc \ n) \land \sim (y = xs!Suc \ n \land x = xs!n))
\end{align*}
apply (simp add: before_in_def index_swap_distinct)
by (metis Suc_lessD Suc_lessI index_nth_id less_Suc_eq nth_mem yes)

lemma projected_paid_same_effect:
assumes \(d::\text{dist _perm } s1 \ s1\)
and ee: \(x \neq y\)
and f: set \(s2 = \{x, y\}\)
and q: \(\text{length } s2 = 2\)
and h: \(\text{dist _perm } s2 \ s2\)
shows \(x < y \ \text{ in } s1 = x < y \ \text{ in } s2 \Longrightarrow \)
\(x < y \ \text{ in } \text{swaps acs } s1 = x < y \ \text{ in } (\text{swap } 0 \ ^{\text{ALG}_P} \text{acs } x \ y \ s1) \ s2\)
proof (induct acs)
case Nil
then show \?case by auto
next
case (Cons s ss)
from d have dd: \(\text{dist _perm } (\text{swaps } ss \ s1) (\text{swaps } ss \ s1)\) by simp
from f have ff: set \((\text{swap } 0 \ ^{\text{ALG}_P} \text{ss } x \ y \ s1) \ s2) = \{x, y\}\ by (metis foldr_replicate_swaps_inv)
from g have gg: \(\text{length } ((\text{swap } 0 \ ^{\text{ALG}_P} \text{ss } x \ y \ s1) \ s2) = 2\) by (metis foldr_replicate_swaps_inv)
from h have hh: \(\text{dist _perm } ((\text{swap } 0 \ ^{\text{ALG}_P} \text{ss } x \ y \ s1) \ s2) ((\text{swap } 0 \ ^{\text{ALG}_P} \text{ss } x \ y \ s1) \ s2)\) by (metis foldr_replicate_swaps_inv)
show \?case (is \?LHS \quad \text{?RHS})
proof (cases Suc \(s < \text{length } (\text{swaps } ss \ s1) \land ((\text{swaps } ss \ s1)!s=x \land (\text{swaps}}}
ss s1)!(Suc s)=y) ∨ ((swaps ss s1)s=y ∧ (swaps ss s1)!(Suc s)=x))

case True
  from True have 1: Suc s < length (swaps ss s1)
      and 2: (swaps ss s1 ! s = x ∧ swaps ss s1 ! Suc s = y
       ∨ swaps ss s1 ! s = y ∧ swaps ss s1 ! Suc s = x) by auto
  from True have ALG_P (s ≠ ss) x y s1 = 1 + ALG_P ss x y s1 by auto
    then have ?RHS = x < y in (swap 0) ((swap 0 ^^ ALG_P ss x y s1) s2)
      by auto
    also have ... = (~ x < y in swaps ss s1)
      by (rule swap0in2)
    also have ... = (swaps ss s1)
      using Cons by auto
    also have ... = x < y in (swap s) (swaps ss s1)
      using 1 2 before_in_swap
      by (metis Suc_lessD before_id dd lessI no_before_in1)
    also have ... = ?LHS by auto
    finally show ?thesis by simp
next
case False
  note F=this
  then have ALG_P (s ≠ ss) x y s1 = ALG_P ss x y s1 by auto
  then have ?RHS = x < y in ((swap 0 ^^ ALG_P ss x y s1) s2)
    by auto
  also have ... = x < y in swaps ss s1
    using Cons by auto
  also have ... = x < y in (swap s) (swaps ss s1)
proof (cases Suc s < length (swaps ss s1))
  case True
    with F have g: swaps ss s1 ! s ≠ x ∨
      swaps ss s1 ! Suc s ≠ y and
    h: swaps ss s1 ! s ≠ y ∨
      swaps ss s1 ! Suc s ≠ x by auto
    show ?thesis
      unfolding before_in_swap[OF dd True, of x y] apply(simp)
        using g h by auto
next
case False
  then show ?thesis unfolding swap_def by(simp)
qed
also have ... = ?LHS by auto
finally show ?thesis by simp
lemma steps\_steps':
  \[\text{length } qs = \text{length } as \implies \text{steps } s \text{ qs as } = \text{steps}' s \text{ qs as } (\text{length } as)\]
  by (induct qs as arbitrary: s rule: list\_induct2) (auto)

lemma T1\_7': \(T_p \text{ init } qs \text{ Strat} = T_p\_opt \text{ init } qs \implies \text{length Strat} = \text{length } qs\)
  \[\implies n \leq \text{length } qs \implies\]
  \[x \not= (y:(a::\text{linorder})) \implies\]
  \[x \in \text{set init} \implies y \in \text{set init} \implies \text{distinct init} \implies\]
  \[\text{set } qs \subseteq \text{set init} \implies\]
  \[
  \exists \text{Strat2 sws}. \quad T_p (Lxy init \{x,y\}) (Lxy (\text{take } n qs) \{x,y\}) \text{ Strat2 + length sws =}
  \]
  \[\text{ALG}xy_{\text{det Strat}} (\text{take } n qs) \text{ init } x y + \text{ALG}_{Pxy} \text{ Strat} (\text{take } n qs) \text{ init } x y\]

proof (induct n)
  case (Suc n)
  from Suc(3,4) have ns: \(n < \text{length } qs\) by simp
  then have n: \(n \leq \text{length } qs\) by simp
  from Suc(1)[OF Suc(2) Suc(3) n Suc(5) Suc(7) Suc(8) Suc(9)]
  obtain Strat2 sws where len: \(\text{length Strat2} = \text{length} (Lxy (\text{take } n qs) \{x,y\})\)
    and iff:
      \[x < y \text{ in steps}' \text{ init} (\text{take } n qs) (\text{take } n \text{ Strat}) n\]
      \[= x < y \text{ in swaps } \text{sws} (\text{steps}' (Lxy \text{ init } \{x,y\})) (Lxy (\text{take } n qs) \{x,y\})\]
    Strat2 (length Strat2))
    and \(T_{\text{Strat2}}: T_p (Lxy \text{ init } \{x,y\}) (Lxy (\text{take } n qs) \{x,y\}) \text{ Strat2 + length sws =}
    \]
    \[\text{ALG}xy_{\text{det Strat}} (\text{take } n qs) \text{ init } x y + \text{ALG}_{Pxy} \text{ Strat} (\text{take } n qs) \text{ init } x y\] by (auto)

qed

qed
from \(\text{Suc}(3, 4)\) have \(\text{nStrat}: n < \text{length Strat}\) by auto

from take.Suc_conv_app_nth[OF this] have tak2: take \((\text{Suc} \ n)\) \text{Strat} = take \ n \ \text{Strat} @ \text{[Strat ! n]}\) by auto

from take.Suc_conv_app_nth[OF \text{ns}] have tak: take \((\text{Suc} \ n)\) \text{qs} = take \ n \ \text{qs} @ \text{[qs ! n]}\) by auto

have aS: \text{length} \ (\text{take} \ n \ \text{Strat}) = n\) using \(\text{Suc}(3, 4)\) by auto

have aQ: \text{length} \ (\text{take} \ n \ \text{qs}) = n\) using \(\text{Suc}(4)\) by auto

from aS aQ have qQS: \text{length} \ (\text{take} \ n \ \text{qs}) = \text{length} \ (\text{take} \ n \ \text{Strat})\) by auto

have xyininit: \(x \in \text{set init}\) : \text{set init}\) by fact+

then have xysubs: \(\{x, y\} \subseteq \text{set init}\) by auto

have dl: \text{distinct init}\) by fact

have set qs \subseteq \text{set init}\) by fact

then have qsnset: \(\text{qs} ! n \in \text{set init}\) using \(\text{ns}\) by auto

from xyininit have ahjer: \(\text{set} \ (Lxy \ \text{init} \ \{x, y\}) = \{x, y\}\)

using xysubs by (simp add: Lxy_set_filter)

with \(\text{Suc}(5)\) have ah: \text{card} \ (\text{set} \ (Lxy \ \text{init} \ \{x, y\})) = 2\) by simp

have ahjer3: \text{distinct} \ (Lxy \ \text{init} \ \{x, y\})

apply(rule Lxy_distinct) by fact

from ah have ahjer2: \text{length} \ (Lxy \ \text{init} \ \{x, y\}) = 2

using distinct_card[OF ahjer3] by simp

show \(\text{?case}\)

proof (cases \(\text{qs}! n \in \{x, y\}\))

case False

with tak have nixzutun: \(Lxy \ (\text{take} \ (\text{Suc} \ n) \ \text{qs}) \ \{x, y\} = Lxy \ (\text{take} \ n \ \text{qs}) \ \{x, y\}\)

unfolding Lxy_def by simp

let \(m=\text{ALG}_P'(\text{take} \ n \ \text{Strat} \ @ \text{[Strat ! n]}) \ (\text{take} \ n \ \text{qs} @ [\text{qs ! n}])\) \text{init} \(n \ x \ y\)

let \(L=\text{replicate} \ ?m \ 0 \ @ \text{sws}\)

\{ fix \(xs::('a::linorder)\) list

fix \(m::\text{nat}\)

fix \(q::'a\)

assume \(q \notin \{x, y\}\)

then have \(5: y \neq q\) by auto

assume \(1: q \in \text{set} \ xs\)

219
assume 2: distinct xs
assume 3: x ∈ set xs
assume 4: y ∈ set xs
have (x < y in xs) = (x < y in (mtf2 m q xs))
  by (metis 1 2 3 4 (q ∈ {x, y}; insertCI not_before_in set_mtf2 swapped_by_mtf2)
} note f=this

have (x < y in steps’ init (take (Suc n) qs) (take (Suc n) Strat) (Suc n))
  = (x < y in mtf2 (fst (Strat ! n)) (qs ! n)
    (swaps (snd (Strat ! n)) (steps’ init (take n qs) (take n Strat) n))))
  unfolding tak2 tak apply(simp only: steps’_append[OF qQS aQ])
  by (simp add: step_def split_def)
also have ... = (x < y in (swaps (snd (Strat ! n)) (steps’ init (take n qs) (take n Strat) n)))
  unfolding tak2 tak apply(simp only: steps’_append[OF qQS aQ])
  by (simp add: step_def split_def)
also have ... = (x < y in (swaps (snd (Strat ! n)) (steps’ init (take n qs) (take n Strat) n)))
  unfolding tak2 tak apply(simp only: steps’_append[OF qQS aQ])
  by (simp add: step_def split_def)
also have ... = (x < y in (swaps (snd (Strat ! n)) (steps’ init (take n qs) (take n Strat) n)))
  unfolding tak2 tak apply(simp only: steps’_append[OF qQS aQ])
  by (simp add: step_def split_def)
also have ... = (x < y in (swaps (snd (Strat ! n)) (steps’ init (take n qs) (take n Strat) n)))
  unfolding tak2 tak apply(simp only: steps’_append[OF qQS aQ])
  by (simp add: step_def split_def)
also have ... = (x < y in (swaps (snd (Strat ! n)) (steps’ init (take n qs) (take n Strat) n)))
  unfolding tak2 tak apply(simp only: steps’_append[OF qQS aQ])
  by (simp add: step_def split_def)
also have ... = (x < y in (swaps (snd (Strat ! n)) (steps’ init (take n qs) (take n Strat) n)))
  unfolding tak2 tak apply(simp only: steps’_append[OF qQS aQ])
  by (simp add: step_def split_def)
also have ... = (x < y in (swaps (snd (Strat ! n)) (steps’ init (take n qs) (take n Strat) n)))
  unfolding tak2 tak apply(simp only: steps’_append[OF qQS aQ])
  by (simp add: step_def split_def)
also have ... = (x < y in (swaps (snd (Strat ! n)) (steps’ init (take n qs) (take n Strat) n)))
  unfolding tak2 tak apply(simp only: steps’_append[OF qQS aQ])
  by (simp add: step_def split_def)
also have ... = (x < y in (swaps (snd (Strat ! n)) (steps’ init (take n qs) (take n Strat) n)))
  unfolding tak2 tak apply(simp only: steps’_append[OF qQS aQ])
  by (simp add: step_def split_def)
also have ... = (x < y in (swaps (snd (Strat ! n)) (steps’ init (take n qs) (take n Strat) n)))
  unfolding tak2 tak apply(simp only: steps’_append[OF qQS aQ])
  by (simp add: step_def split_def)
also have ... = (x < y in (swaps (snd (Strat ! n)) (steps’ init (take n qs) (take n Strat) n)))
  unfolding tak2 tak apply(simp only: steps’_append[OF qQS aQ])
  by (simp add: step_def split_def)
also have ... = (x < y in (swaps (snd (Strat ! n)) (steps’ init (take n qs) (take n Strat) n)))
  unfolding tak2 tak apply(simp only: steps’_append[OF qQS aQ])
  by (simp add: step_def split_def)
also have ... = (x < y in (swaps (snd (Strat ! n)) (steps’ init (take n qs) (take n Strat) n)))
  unfolding tak2 tak apply(simp only: steps’_append[OF qQS aQ])
  by (simp add: step_def split_def)
also have ... = (x < y in (swaps (snd (Strat ! n)) (steps’ init (take n qs) (take n Strat) n)))
  unfolding tak2 tak apply(simp only: steps’_append[OF qQS aQ])
  by (simp add: step_def split_def)
also have ... = (x < y in (swaps (snd (Strat ! n)) (steps’ init (take n qs) (take n Strat) n)))
  unfolding tak2 tak apply(simp only: steps’_append[OF qQS aQ])
  by (simp add: step_def split_def)
also have ... = (x < y in (swaps (snd (Strat ! n)) (steps’ init (take n qs) (take n Strat) n)))
  unfolding tak2 tak apply(simp only: steps’_append[OF qQS aQ])
  by (simp add: step_def split_def)
also have ... = (x < y in (swaps (snd (Strat ! n)) (steps’ init (take n qs) (take n Strat) n)))
  unfolding tak2 tak apply(simp only: steps’_append[OF qQS aQ])
  by (simp add: step_def split_def)
also have ... = (x < y in (swaps (snd (Strat ! n)) (steps’ init (take n qs) (take n Strat) n)))
  unfolding tak2 tak apply(simp only: steps’_append[OF qQS aQ])
  by (simp add: step_def split_def)
also have ... = (x < y in (swaps (snd (Strat ! n)) (steps’ init (take n qs) (take n Strat) n)))
  unfolding tak2 tak apply(simp only: steps’_append[OF qQS aQ])
  by (simp add: step_def split_def)
also have ... = (x < y in (swaps (snd (Strat ! n)) (steps’ init (take n qs) (take n Strat) n)))
  unfolding tak2 tak apply(simp only: steps’_append[OF qQS aQ])
  by (simp add: step_def split_def)
apply(simp)
apply(fact)
using iff by auto

finally have umfa: x < y in steps' init (take (Suc n) qs) (take (Suc n) Strat) (Suc n) =
  x < y
  in (swap 0 ^^ ALG_P (snd (Strat ! n)) x y (steps' init (take n qs) (take n Strat) n))
  (swaps sus (steps' (Lxy init {x, y}) (Lxy (take n qs) {x, y}) Strat2 (length Strat2))).

from Suc(3,4) have ls: length (take n Strat) = n by auto
have (take n Strat @ [Strat ! n]) ! n =
  (take n Strat @ (Strat ! n) # []) ! length (take n Strat) using ls by auto
also have ... = Strat ! n by (rule nth_append_length)
finally have tt: (take n Strat @ [Strat ! n]) ! n = Strat ! n.

show ?thesis
apply(rule exI[where x=Strat2])
apply(rule exI[where x=?L])
unfolding nizzutun
apply(safe)
apply(fact)
proof goal_cases
  case 1
  show ?case
  unfolding tak2 tak
  apply(simp add: step_def split_def)
  unfolding ALG_P' def
  unfolding tt
  using aS apply(simp only: steps'_rests[OF qQS, symmetric])
  using 1(1) umfa by auto
next
  case 2
  then show ?case
  apply(simp add: step_def split_def)
  unfolding ALG_P' def
  unfolding tt
  using aS apply(simp only: steps'_rests[OF qQS, symmetric])
  using umfa[symmetric] by auto
next
  case 3
have \( ns2 \): \( n < \text{length} (\text{take} n \; qs @ [qs ! n]) \)

using \( ns \) by auto

have \( er \): \( \text{length} (\text{take} n \; qs) < \text{length} \; \text{Strat} \)

using Suc.prems(2) aQ \( ns \) by linarith

have \( T_p (Lxy \; \text{init} \; \{x,y\}) \; (Lxy \; (\text{take} n \; qs) \; \{x, y\}) \; \text{Strat2} \)

+ \( \text{length} (\text{replicate} \; (ALG_P' \; \text{Strat} (\text{take} n \; qs @ [qs ! n]) \; \text{init} \; n \; x \; y) \) 0 \)

@ sus)

\( = \; (T_p (Lxy \; \text{init} \; \{x,y\}) \; (Lxy \; (\text{take} n \; qs) \; \{x, y\}) \; \text{Strat2} + \text{length} \; \text{sws}) \)

+ \( ALG_P' \; \text{Strat} (\text{take} n \; qs @ [qs ! n]) \; \text{init} \; n \; x \; y \) by simp

also have \( \ldots = ALGxy_{\text{det}} \; \text{Strat} (\text{take} n \; qs) \; \text{init} \; x \; y + \)

\( ALG_{Pxy} \; \text{Strat} (\text{take} n \; qs) \; \text{init} \; x \; y + \)

\( ALG_{P'} \; \text{Strat} (\text{take} n \; qs @ [qs ! n]) \; \text{init} \; n \; x \; y \)

unfolding \( T_{\text{Strat2}} \) by simp

also have \( \ldots = ALGxy_{\text{det}} \; \text{Strat} (\text{take} (\text{Suc} \; n) \; qs) \; \text{init} \; x \; y + \)

\( ALG_{Pxy} \; \text{Strat} (\text{take} (\text{Suc} \; n) \; qs) \; \text{init} \; x \; y + \)

unfolding \( tak \) unfolding wegdamit[\( OF \) \( er \) \( False \)] apply(simp)

unfolding \( ALG_P'_{\text{split}}[\text{of} \; \text{take} \; n \; qs \; \text{Strat} \; qs ! n \; \text{init} \; x \; y, \text{unfolded} \) aQ, \( OF \) \( nStrat \)]

by(simp)

finally show \( ?\text{case} \) unfolding \( tak \) using \( ALG_P'_{\text{rest}}[OF \; ns \; nStrat] \) by auto

qed

next

case \( True \)

note \( qsinyx = \text{this} \)

then have \( yeh: Lxy (\text{take} (\text{Suc} \; n) \; qs) \; \{x, y\} = Lxy (\text{take} n \; qs) \; \{x,y\} \)

@ [qs!n]

unfolding \( tak \) \( Lxy_{\text{def}} \) by auto

from \( True \) have \( garar: (\text{take} \; n \; qs @ [qs ! n]) \; ! \; n \in \{y, x\} \)

using \( tak[\text{symmetric}] \) by(auto)

have \( aer: \forall \; i < n. \)

\( (((\text{take} \; n \; qs @ [qs ! n]) \; ! \; i \in \{y, x\}) \)

\( = (\text{take} \; n \; qs \; i \in \{y, x\}) \) using \( ns \) by \( \text{metis} \; \text{less_SucI} \; \text{nth_take} \; \text{tak} \)

let \( \?\text{Strat}_mft = \; \text{fst} \; (\text{Strat} \; ! \; n) \)

\( 222 \)
let \( ?\text{Strat}_{\text{sws}} = \text{snd} (\text{Strat} ! n) \)

let \( ?\text{x}_s = \text{steps}' \text{init} (\text{take} n \text{qs}) (\text{take} n \text{Strat}) n \)

let \( ?\text{x}_s' = (\text{swaps} (\text{snd} (\text{Strat} ! n)) ?\text{x}_s) \)

let \( ?\text{x}_s'' = \text{steps}' \text{init} (\text{take} (\text{Suc} n) \text{qs}) (\text{take} (\text{Suc} n) \text{Strat}) (\text{Suc} n) \)

let \( ?\text{x}_s''2 = \text{mtf2} ?\text{Strat}_{\text{mft}} (qs!n) ?\text{x}_s' \)

let \( ?\text{no_swap_occurs} = (x < y \text{ in } ?\text{x}_s') = (x < y \text{ in } ?\text{x}_s''2) \)

let \( ?\text{mtf} = (\text{if } ?\text{no_swap_occurs} \text{ then } 0 \text{ else } 1 \text{::nat}) \)

let \( ?m = \text{ALG}_{P'} \text{Strat} (\text{take} n \text{qs} \text{@} [qs ! n]) \text{init} n x y \)

let \( ?\text{L} = \text{replicate } ?m 0 \text{@} \text{sws} \)

let \( ?\text{newStrat} = \text{Strat2} \text{@} [(?\text{mtf}, ?\text{L})] \)

have \( ?\text{x}_s'' = \text{step} ?\text{x}_s (qs!n) (\text{Strat} ! n) \)

unfolding \( \text{tak} \) \( \text{tak2} \)

apply (rule \( \text{steps}'\_\text{append} \)) by fact+

also have \( \ldots = \text{mtf2} (\text{fst} (\text{Strat} ! n)) (qs!n) (\text{swaps} (\text{snd} (\text{Strat} ! n)) ?\text{x}_s) \)

unfolding \( \text{step\_def} \)

by (auto simp: \( \text{split\_def} \))

finally have \( A: ?\text{x}_s'' = \text{mtf2} (\text{fst} (\text{Strat} ! n)) (qs!n) ?\text{x}_s'. \)

let \( ?\text{ys} = (\text{steps}' (Lxy \text{init} \{x, y\})) (Lxy (\text{take} n \text{qs}) \{x, y\}) \text{Strat2} (\text{length} \text{Strat2})) \)

let \( ?\text{ys}' = (\text{swaps} \text{sws} (\text{steps}' (Lxy \text{init} \{x, y\}))) (Lxy (\text{take} n \text{qs}) \{x, y\}) \text{Strat2} (\text{length} \text{Strat2})) \)

let \( ?\text{ys}'' = (\text{swap} 0 \text{``} \text{ALG}_{P'} (\text{snd} (\text{Strat} ! n)) x y \text{?xs} ) ?\text{ys}' \)

let \( ?\text{ys}''' = (\text{steps}' (Lxy \text{init} \{x, y\}))) (Lxy (\text{take} (\text{Suc} n) \text{qs}) \{x, y\}) \text{?newStrat} (\text{length } ?\text{newStrat}) \)

have \( \text{gr}: Lxy (\text{take} n \text{qs} \text{@} [qs ! n]) \{x, y\} = \)

\( Lxy (\text{take} n \text{qs}) \{x, y\} \text{@} [qs ! n] \text{unfolding } Lxy\_\text{def} \text{using } \text{True} \)

by (simp)

have \( \text{steps'} \text{init} (\text{take} n \text{qs} \text{@} [qs ! n]) \text{Strat} n \)

= \( \text{steps'} \text{init} (\text{take} n \text{qs} \text{@} [qs ! n]) (\text{take} n \text{Strat} \text{@} \text{drop} n \text{Strat}) n \) by simp

also have \( \ldots = \text{steps'} \text{init} (\text{take} n \text{qs} \text{@} [qs ! n]) (\text{take} n \text{Strat} \text{@} \text{drop} n \text{Strat}) n \) by

apply (subst \( \text{steps'}\_\text{rests}[\text{symmetric}] \)) using \( aS \) \( qQS \) by (simp\_all)

finally have \( t: \text{steps'} \text{init} (\text{take} n \text{qs} \text{@} [qs ! n]) \text{Strat} n \)

= \( \text{steps'} \text{init} (\text{take} n \text{qs}) (\text{take} n \text{Strat}) n \).

have \( \text{gge}: \text{swaps} (\text{replicate } ?m 0) ?\text{ys}' \)
unfolding $\text{ALG}_P'$ def $t$ by simp

have $gg$: length $\newStrat = \text{Suc}(\text{length Strat2})$ by auto

have $\ys'''' = \text{step}\ ?y\ (qs!n)\ (\text{mtf},?L)$
  unfolding $\text{tak}\ \text{gr}$ unfolding $gg$
  apply (rule steps' append)
  using len by unfolding $gg$

also have $\ys'''''' = \text{step}\ ?ys\ (qs!n)\ (\text{mtf},?L)$
  by (simp)

also have $\ys'''''' = \text{step}\ ?ys\ (qs!n)\ ?ys''$
  unfolding $\text{step}$ def by (simp add: split_def)

also have $\ys'''''' = \text{m tf}\ ?mtf\ (qs!n)\ ?ys''$
  unfolding $\text{m tf}$ def by auto

also have $\ys'''''' = \text{m tf}\ ?mtf\ (qs!n)\ ?ys''''$
  using $\text{gge}$ by (simp)

finally have $B$: $\ys'''''' = \text{m tf}\ (qs!n)\ ?ys''''''$.

have 3: $\ys' = \{x,y\}$
  apply (simp add: swaps_inv)
apply (subst steps'_set) using $\text{ahjer}\ \text{len}$
by (simp_all)

have $k$: $\ys'''' = \text{swaps}\ (\text{replicate}\ \text{ALG}_P\ (\text{snd}\ (\text{Strat!n}))\ x\ y)\ 0)$
  $\ys''$
  by (auto)

have 6: $\ys'''' = \{x,y\}$ unfolding $k$ using 3 swaps_inv by metis
have 7: $\ys'''' = \{x,y\}$ unfolding $B$ using set_mtf2 6 by metis

have 22: $x \in \ys''''$ $y \in \ys''''$ using 6 by auto
have 23: $x \in \ys''''$ $y \in \ys''''$ using 7 by auto

have 26: $(qs!n) \in \ys''''$ using 6 True by auto

have distinct $\ys$ apply (rule steps'_distinct2)
  using $\text{len}\ \text{ahjer}$ by (simp)+
then have 9: distinct $\ys'$ using swaps_inv by metis
then have 27: distinct $\ys''''$ unfolding $k$ using swaps_inv by metis

from 3 Suc(5) have card $(\ys') = 2$ by auto
then have 4: length $\ys' = 2$ using distinct_card[OF 9] by simp
have length $\ys'''' = 2$ unfolding $k$ using 4 swaps_inv by metis
have 5: dist_perm $\ys' \ys''''$ using 9 by auto

have sx$: set $\xs = \text{set init}$ apply (rule steps'_set) using $qQS\ n\ Suc(3)$
by (auto)

have sx$: set $\xs' = \text{set init}$ using $\text{swaps_inv}$ by metis
have sx$: set $\xs'' = \text{set init}$ unfolding $A$ using set_mtf2 by metis
have 24: \( x \in \text{set } ?xs' \) \( y \in \text{set } ?xs' \) \((qs!n) \in \text{set } ?xs'\)
  using \text{ysubs True } sxs sxs' by auto
have 28: \( x \in \text{set } ?xs'' \) \( y \in \text{set } ?xs'' \) \((qs!n) \in \text{set } ?xs''\)
  using \text{xysubs True } sxs sxs's'' by auto

have 0: dist_perm init init using \text{dI} by auto
have 1: dist_perm ?xs ?xs apply (rule steps'_dist_perm) by \text{fact+}
then have 25: \text{distinct } ?xs' using \text{swaps_inv} by \text{metis}

from \text{projected_paid_same_effect[OF 1 Suc(5) 3 4 5, OF iff, where acs = snd (Strat ! n)]}
have aaa: \( x < y \) in \( ?xs' = x < y \) in \( ?ys'' \).

have \text{t: } \text{?mtf} = (\text{if } (x<y \text{ in } ?xs') = (x<y \text{ in } ?xs'')) \text{ then 0 else 1)}
  by (simp add: A)

have central: \( x < y \) in \( ?xs'' = x < y \) in \( ?ys'' \)
proof (cases (x<y in ?xs') = (x<y in ?xs''))
case True
  then have \( ?mtf = 0 \) using \text{t} by auto
  with \text{B} have \( ?ys''' = ?ys'' \) by auto
  with \text{aaa True} show \( \text{thesis} \) by auto
next
case False
  then have \( ?mtf = 1 \) using \text{t} by auto
from \text{False} have \( i: (x<y \text{ in } ?xs') = (~x<y \text{ in } ?xs') \) by auto

have gn: \( \land \ a \ b. a\in\{x,y\} \implies b\in\{x,y\} \implies \text{set } ?ys'' = \{x,y\} \implies a\neq b \implies \text{distinct } ?ys'' \implies a<b \text{ in } ?ys'' \implies ~a<b \text{ in mtf2 1 b } ?ys''

proof goal_cases
  case (1 a b)
  from \text{1} have \( f: \text{set } ?ys'' = \{a,b\} \) by auto
  with \text{1} have \( i: \text{card (set } ?ys'' = 2 \) by auto
  from \text{1(5)} have \( \text{dist_perm } ?ys'' \text{ ?ys'' by auto}
  from \text{i distinct_card 1(5)} have \( g: \text{length } ?ys'' = 2 \) by \text{metis}
  with \text{1(6)} have \( d: \text{index } ?ys'' b = 1 \)
    using \text{before_in_index2} f \text{1(4)} by \text{fastforce}
from 1(2,3) have e: b ∈ set ys'' by auto

from d e have p: mtf2 1 b ys'' = swap 0 ys''
  unfolding mtf2_def by auto
have q: a < b in swap 0 ys'' = (¬ a < b in ys'')
  apply(rule swap0in2) by(fact)+
from 1(6) p q show case by metis

qed

show thesis
proof (cases x<y in xs')
  case True
  with aaa have st: x < y in ys'' by auto
from True False have ~ x<y in xs'' by auto
  with Suc(5) 28 not_before_in A have y < x in xs'' by metis
with A have y < x in mtf2 (fst (Strat!n)) (qs!n) xs' by auto

have itsy: y = (qs!n)
  apply(rule swapped_by_mtf2[where xs= xs'])
    apply(fact)
    apply(fact)
    apply(fact 24)
    apply(fact 24)
    by(fact)+
have ~x<y in mtf2 1 y ys''
  apply(rule gn)
    apply(simp)
    apply(simp)
    apply(simp add: 6)
    by(fact)+
then have ts: ~x<y in ys'' using B itsy k by auto
have ii: (x<y in ys'') = (~x<y in ys'') using st ts by auto
from i ii aaa show thesis by metis

next
  case False
  with aaa have st: ~ x < y in ys'' by auto
with Suc(5) 22 not_before_in have st: y < x in ys'' by metis
from i False have kl: x<y in xs'' by auto
with A have x < y in mtf2 (fst (Strat!n)) (qs!n) xs' by auto
from False Suc(5) 24 not_before_in have y < x in xs' by metis
have itsx: x = (qs!n)
  apply(rule swapped_by_mtf2[where xs= xs'])
    apply(fact)
    apply(fact)

226
apply (fact 24 (2))
apply (fact 24)
by (fact) +

have "y < x in mtf 2 1 x ?ys"
apply (rule gn)
apply (simp)
apply (simp)
apply (simp add: 6)
apply (metis Suc 5)
by (fact) +

then have "y < x in ?ys" using itisz k B by auto
with Suc 5 not before in 23 have x < y in ?ys" by metis
with st have (x < y in ?ys") = ("x < y in ?ys") using B k by auto
with i aaa show ?thesis by metis
qed

show ?thesis
apply (rule exI [where x = ?newStrat])
apply (rule exI [where x = []])
proof (standard, goal_cases)
case 1
show ?case unfolding yeh using len by (simp)
next
case 2
show ?case
proof (standard, goal_cases)
case 1

from central show ?case by auto
next
case 2

have j: \( \text{ALG}_{\text{xy}} \_ \text{det} \_ \text{Strat} \_ (\text{take} \_ (\text{Suc} \_ n \_ ) \_ qs) \_ \text{init} \_ x \_ y = \) \( \text{ALG}_{\text{xy}} \_ \text{det} \_ \text{Strat} \_ (\text{take} \_ n \_ qs) \_ \text{init} \_ x \_ y \) \( + \_ (\text{ALG}' \_ \text{det} \_ \text{Strat} \_ qs \_ \text{init} \_ n \_ y \_ + \_ \text{ALG}' \_ \text{det} \_ \text{Strat} \_ qs \_ \text{init} \_ n \_ x) \)\)
proof
have \( \text{ALG}_{\text{xy}} \_ \text{det} \_ \text{Strat} \_ (\text{take} \_ (\text{Suc} \_ n \_ ) \_ qs) \_ \text{init} \_ x \_ y = \) \( (\sum i \in \ldots \_ \text{length} \_ (\text{take} \_ n \_ qs \_ @ \_ [qs \_ ! \_ n])) \).\)
if (take n qs @ [qs ! n]) ! i \in \{y, x\}
then \( \text{ALG}' \_ \text{det} \_ \text{Strat} \_ (\text{take} \_ n \_ qs \_ @ \_ [qs \_ ! \_ n]) \_ \text{init} \_ i \_ y \) \( + \_ \text{ALG}' \_ \text{det} \_ \text{Strat} \_ (\text{take} \_ n \_ qs \_ @ \_ [qs \_ ! \_ n]) \_ \text{init} \_ i \_ x \) else 0 \) unfolding \( \text{ALG}_{\text{xy}} \_ \text{det} \_ \text{def} \_ \text{tak} \) by auto
also have \ldots \)
= \left( \sum_{i \in \{\sim< \text{Suc } n\}} \right).

if (\text{take } n \text{ qs } @ [\text{qs } ! n]) \! i \in \{y, x\}
then \text{ALG'} \_ \text{det Strat} (\text{take } n \text{ qs } @ [\text{qs } ! n]) \text{ init } i \ y
+ \text{ALG'} \_ \text{det Strat} (\text{take } n \text{ qs } @ [\text{qs } ! n]) \text{ init } i \ x
else 0 \) \text{ using ns by simp}

also have \ldots = \left( \sum_{i \in \{\sim< n\}} \right).

if (\text{take } n \text{ qs } @ [\text{qs } ! n]) \! i \in \{y, x\}
then \text{ALG'} \_ \text{det Strat} (\text{take } n \text{ qs } @ [\text{qs } ! n]) \text{ init } i \ y
+ \text{ALG'} \_ \text{det Strat} (\text{take } n \text{ qs } @ [\text{qs } ! n]) \text{ init } i \ x
else 0 \) \text{ by simp}

also have \ldots = \left( \sum_{i \in \{\sim< n\}} \right).

if take n qs ! i \in \{y, x\}
then \text{ALG'} \_ \text{det Strat} (\text{take } n \text{ qs } @ [\text{qs } ! n]) \text{ init } i \ y
+ \text{ALG'} \_ \text{det Strat} (\text{take } n \text{ qs } @ [\text{qs } ! n]) \text{ init } i \ x
else 0 \) \text{ using aer using garar by simp}

also have \ldots = \left( \sum_{i \in \{\sim< n\}} \right).

if take n qs ! i \in \{y, x\}
then \text{ALG'} \_ \text{det Strat} (\text{take } n \text{ qs } @ [\text{qs } ! n]) \text{ init } i \ y
+ \text{ALG'} \_ \text{det Strat} (\text{take } n \text{ qs } @ [\text{qs } ! n]) \text{ init } i \ x
else 0 \) \text{ by simp}

also have \ldots = \left( \sum_{i \in \{\sim< n\}} \right).

if take n qs ! i \in \{y, x\}
then \text{ALG'} \_ \text{det Strat} (\text{take } n \text{ qs } @ [\text{qs } ! n]) \text{ init } i \ y
+ \text{ALG'} \_ \text{det Strat} (\text{take } n \text{ qs } @ [\text{qs } ! n]) \text{ init } i \ x
else 0 \) \text{ by simp}

proof –

have \text{ALG'} \_ \text{det Strat} qs \text{ init } n \ y
= \text{ALG'} \_ \text{det Strat} (\text{take } n \text{ qs } @ [\text{qs } ! n]) @ \text{drop } (\text{Suc } n) \text{ qs}

init n \ y

unfolding \text{tak}[\text{symmetric}] \text{ by auto}

also have \ldots = \text{ALG'} \_ \text{det Strat} (\text{take } n \text{ qs } @ [\text{qs } ! n]) \text{ init } n \ y
apply(\text{rule ALG'} \_ \text{det_append}) \text{ using nStrat ns by(auto)}
finally have 1: \text{ALG'} \_ \text{det Strat} qs \text{ init } n \ y = \text{ALG'} \_ \text{det Strat}
(take n qs @ [qs ! n]) \text{ init } n \ y.

have \text{ALG'} \_ \text{det Strat} qs \text{ init } n \ x
= \text{ALG'} \_ \text{det Strat} (\text{take } n \text{ qs } @ [\text{qs } ! n]) @ \text{drop } (\text{Suc } n) \text{ qs}

init n \ x

unfolding \text{tak}[\text{symmetric}] \text{ by auto}

also have \ldots = \text{ALG'} \_ \text{det Strat} (\text{take } n \text{ qs } @ [\text{qs } ! n]) \text{ init } n \ x
apply(\text{rule ALG'} \_ \text{det_append}) \text{ using nStrat ns by(auto)}
finally have 2: \text{ALG'} \_ \text{det Strat} qs \text{ init } n \ x = \text{ALG'} \_ \text{det Strat}
(take n qs @ [qs ! n]) init n x.

from 1 2 show ?thesis by auto

qed

also have ... = (∑ i ∈ {..< n}.
    if take n qs ! i ∈ {y, x} then
      ALG'_det Strat (take n qs) init i y
    + ALG'_det Strat (take n qs) init i x
    else 0)
  + ALG'_det Strat qs init n y + ALG'_det Strat qs init n x
apply(simp)
apply(rule sum.cong)
apply(simp)
apply(simp)
using ALG'_det_append[where qs=take n qs] Suc.prems(2) ns
by auto

also have ... = (∑ i ∈ {..< length(take n qs)}.
    if take n qs ! i ∈ {y, x} then
      ALG'_det Strat (take n qs) init i y
    + ALG'_det Strat (take n qs) init i x
    else 0)
  + ALG'_det Strat qs init n y + ALG'_det Strat qs init n x
using aQ by auto

also have ... = ALGxy_det Strat (take n qs) init x y
  + (ALG'_det Strat qs init n y + ALG'_det Strat qs init n x)
unfolding ALGxy_det_def by(simp)
finally show ?thesis .

qed

have list: ?ys' = swaps sws (steps (Lxy init {x, y}) (Lxy (take n qs) {x, y}) Strat2)
  unfolding steps_steps[OF len[symmetric], of (Lxy init {x, y})]
by simp

have j2: steps' init (take n qs @ [qs ! n]) Strat n
  = steps' init (take n qs) (take n Strat) n
proof —
  have steps' init (take n qs @ [qs ! n]) Strat n
    = steps' init (take n qs @ [qs ! n]) (take n Strat @ drop n
Strat) n
  by auto
  also have ... = steps' init (take n qs) (take n Strat) n
  apply(rule steps'_rests[symmetric]) apply fact using aS by
simp

finally show thesis.

qed

have arghschonwieder: steps' init (take n qs) (take n Strat) n
  = steps' init qs Strat n

proof
  have steps' init qs Strat n
    = steps' init (take n qs @ drop n qs) (take n Strat @ drop n Strat) n
  by auto
  also have ... = steps' init (take n qs) (take n Strat) n
    apply (rule steps' rests[symmetric]) apply fact using aS by

simp

finally show thesis by simp

qed

have indexe: ((swap 0 ^^ ?m) (swaps sws
  (steps (Lxy init {x,y}) (Lxy (take n qs) {x, y}) Strat2)))
  = ?ys'' unfolding ALG_P_def unfolding list using j2 by auto

have blocky: ALG'_det Strat qs init n y
  = (if y < qs ! n in ?xs' then 1 else 0)
unfolding ALG'_det_def ALG.simps by (auto simp: arghschonwieder
split_def)

have blockx: ALG'_det Strat qs init n x
  = (if x < qs ! n in ?xs' then 1 else 0)
unfolding ALG'_det_def ALG.simps by (auto simp: arghschonwieder
split_def)

have index_is_blocking_cost: index ((swap 0 ^^ ?m) (swaps sws
  (steps (Lxy init {x,y}) (Lxy (take n qs) {x, y}) Strat2)))
  (qs ! n)
  = ALG'_det Strat qs init n y + ALG'_det Strat qs init n x

proof (cases x= qs!n)
  case True
  then have ALG'_det Strat qs init n x = 0
    unfolding blockx apply (simp) using before_in_irefl by metis
  then have ALG'_det Strat qs init n y + ALG'_det Strat qs init n
    x
    = (if y < x in ?xs' then 1 else 0) unfolding blocky using
True by simp

also have ... = (if ~y < x in ?xs' then 0 else 1) by auto
also have ... = (if x < y in ?xs' then 0 else 1)

230
apply[(simp)] by (meson Suc.prems Suc.prems not_before_in)
also have ... = (if \( x < y \) then 0 else 1) using aaa by simp
also have ... = index ?ys'' \( x \)
apply[(rule before_in_index1)] by(fact)+
finally show ?thesis unfolding indexe using True by auto

next
  case False
  then have q: \( y = qs!n \) using qsinxy by auto
  then have \( ALG'_\text{det} \text{ Strat} qs \text{ init} n y = 0 \)
    unfolding blocky apply[(simp)] using before_in_irrefl by metis
  then have \( ALG'_\text{det} \text{ Strat} qs \text{ init} n y + ALG'_\text{det} \text{ Strat} qs \text{ init} n \)
    \( x = (if \( x < y \) \text{ in } ?xs' \text{ then } 1 \text{ else } 0) \) unfolding blockx using q
    by simp
also have ... = (if \( x < y \) \text{ in } ?ys'' \text{ then } 1 \text{ else } 0) using aaa by simp
also have ... = index ?ys'' y
apply[(rule before_in_index2)] by(fact)+
finally show ?thesis unfolding indexe using q by auto
qed

have jj: \( ALG_{Pxy} \text{ Strat (take (Suc n) qs) init} x y = \)
  \( ALG_{Pxy} \text{ Strat (take n qs) init} x y + ALG'_{P'} \text{ Strat (take (Suc n) qs) init} x y \)
proof
  have \( ALG_{Pxy} \text{ Strat (take (Suc n) qs) init} x y = \)
  \( (\sum i<\text{length (take (Suc n) qs).} \ ALG'_{P'} \text{ Strat (take (Suc n) qs) init} x y) \)
    unfolding \( ALG_{Pxy} \text{def} \) by simp
  also have ... = (\( \sum i<\text{Suc n.} \ ALG'_{P'} \text{ Strat (take (Suc n) qs) init} x y \)
    unfolding \( \text{take using ns} \) by simp
also have ... = (\( \sum i<\text{n.} \ ALG'_{P'} \text{ Strat (take (Suc n) qs) init} i x y \)
  + \( ALG'_{P'} \text{ Strat (take (Suc n) qs) init} n x y \)
  by simp
also have ... = (\( \sum i<\text{length (take n qs).} \ ALG'_{P'} \text{ Strat (take n qs @ [qs ! n]) init} i x y \)
    + \( ALG'_{P'} \text{ Strat (take n qs @ [qs ! n]) init} n x y \)
  unfolding \( \text{take using ns} \) by auto
also have ... = (\( \sum i<\text{length (take n qs).} \ ALG'_{P'} \text{ Strat (take n qs) init} i x y \)
  + \( ALG'_{P'} \text{ Strat (take n qs @ [qs ! n]) init} n x y \) (is ?A +
\( ?B = ?A' + ?B \)

proof
have \( ?A = ?A' \)
apply (rule sum.cong)
apply (simp)
proof goal_cases
case 1
show ?case
apply (rule ALG P' rest2[symmetric, where ?r1.0=[]], simplified)
using 1 apply (simp)
using 1 nStrat by (simp)
qed
then show ?thesis by auto
qed
also have \( \ldots = ALG_P xy Strat \) (take n qs) init x y
+ ALG_P' Strat (take n qs @ [qs ! n]) init n x y
unfolding ALG_Pxy_def by auto
finally show ?thesis .
qed

have \( tw: \text{length} (Lxy \text{ init} \{x, y\}) = \text{length} Strat2 \)
using len by auto
have \( T_p (Lxy \text{ init} \{x, y\}) (Lxy \text{ take} (Suc n) \text{ qs} \{x, y\}) \text{ newStrat} \)
+ \( \text{length} [] \)
= \( T_p (Lxy \text{ init} \{x, y\}) (Lxy \text{ take} n \text{ qs} \{x, y\}) Strat2 \)
+ \( t_p (\text{steps} (Lxy \text{ init} \{x, y\}) (Lxy \text{ take} n \text{ qs} \{x, y\}) Strat2) \)
\( (qs ! n) \) (?mtf, ?L)
unfolding yeh
by (simp add: T_APPEND[OF tw, of (Lxy init) \{x, y\}])
also have \( \ldots = T_p (Lxy \text{ init} \{x, y\}) (Lxy \text{ take} n \text{ qs} \{x, y\}) Strat2 \)
+ \( \text{length} \) sus
+ \( \text{index} ((\text{swap} 0 ^^ ?m) \text{ swaps} \) sus
\text{steps} (Lxy init \{x, y\}) (Lxy (take n qs \{x, y\}) Strat2)) \)
\( (qs ! n) \)
+ ALG_P' Strat (take n qs @ [qs ! n]) init n x y
by (simp add: t_p_def)
also have \( \ldots = ALG_{xy \text{ det} Strat} \) (take n qs) init x y
+ \( \text{index} ((\text{swap} 0 ^^ ?m) \text{ swaps} \) sus
\text{steps} (Lxy init \{x, y\}) (Lxy (take n qs \{x, y\}) Strat2)) \)
\( (qs ! n) \)
+ (ALG_Pxy Strat (take n qs) init x y

232
+ \text{ALG}_{P'}(\text{Strat} (\text{take } n \text{ qs @ } [\text{qs}! n]) \text{ init } n \text{ x y)}

\text{by (simp only: T_{Strat2})}

\text{also from index_is_blocking_cost have \ldots = (ALG_{xy} \text{ det Strat} (\text{take } n \text{ qs}) \text{ init } n \text x y)}
+ \text{ALG'}_{\text{det Strat}} \text{ qs init } n \text{ y} + \text{ALG'}_{\text{det Strat}} \text{ qs init } n \text{ x}
+ (\text{ALG}_{P_{xy}}(\text{Strat} (\text{take } n \text{ qs}) \text{ init } n \text x y)
+ \text{ALG}_{P'}(\text{Strat} (\text{take } n \text{ qs @ } [\text{qs}! n]) \text{ init } n \text x y) \text{ by auto}}

\text{also have \ldots = ALG_{xy} \text{ det Strat} (\text{take } (Suc n) \text{ qs}) \text{ init } n x y}
+ (\text{ALG}_{P_{xy}}(\text{Strat} (\text{take } n \text{ qs}) \text{ init } n \text x y)
+ \text{ALG}_{P'}(\text{Strat} (\text{take } n \text{ qs @ } [\text{qs}! n]) \text{ init } n \text x y) \text{ using j}}

\text{by auto}

\text{also have \ldots = ALG_{xy} \text{ det Strat} (\text{take } (Suc n) \text{ qs}) \text{ init } n x y}
+ \text{ALG}_{P_{xy}}(\text{Strat} (\text{take } (Suc n) \text{ qs}) \text{ init } n \text x y) \text{ using jj by auto}

\text{finally show ?case .}

\text{proof goal_cases}

\text{case 1}

\text{show ?case apply (rule Lxy_mono unfolded Lxy_def, simplified)}}

\text{using 1 by auto}

\text{qed}

\text{qed}

\text{next case 0}

\text{then show ?case}

\text{apply (simp add: Lxy_def ALG_{xy} \text{ def ALG}_{P_{xy}} \text{ def T_{opt} \text{ def})}}

\text{proof goal_cases}

\text{case 1}

\text{show ?case apply (rule Lxy_mono unfolded Lxy_def, simplified)}}

\text{using 1 by auto}

\text{qed}

\text{qed}

\text{lemma T1.7:}

\text{assumes T_{p} \text{ init qs Strat = T_{p-opt} init qs length Strat = length qs}}
\text{x \neq (y::('a::linorder)) x \in set init y \in set init distinct init}
\text{set qs \subseteq \text{ set init}}

\text{shows T_{p-opt} (\text{Lxy init \{x,y\}) (Lxy qs \{x,y\}) \leq ALG_{xy} \text{ det Strat} qs \text{ init } x y + ALG_{P_{xy}} \text{ Strat} qs \text{ init } x y}

\text{proof –}

\text{have A:length qs \leq length qs by auto}

\text{have B: x \neq y using assms by auto}

\text{from T1.7[OF assms(1,2), of length qs x y, OF A B assms(4-7)]}

\text{obtain Strat2 sws where}

\text{len: length Strat2 = length (Lxy qs \{x, y\})}

\text{and x < y in steps' init qs (take (length qs) Strat)}
\[(\text{length } qs) = x < y \text{ in swaps } \text{sws} (\text{steps}' (Lxy \text{ init } \{x,y\})) \]
\[(Lxy \text{ qs } \{x, y\}) \text{ Strat2 } (\text{length Strat2})\]
\[\text{and } T_p : T_p (Lxy \text{ init } \{x,y\}) (Lxy \text{ qs } \{x, y\}) \text{ Strat2 } + \text{length } \text{sws} = ALG_{xy, \text{det Strat qs init x y}} + ALG_{Pxy \text{ Strat qs init x y}} \text{ by auto}\]

\text{have } T_p, -\text{opt } (Lxy \text{ init } \{x,y\}) (Lxy \text{ qs } \{x, y\}) \leq T_p (Lxy \text{ init } \{x,y\}) (Lxy \text{ qs } \{x, y\}) \text{ Strat2} + \text{length } \text{sws} = ALG_{xy, \text{det Strat qs init x y}} + ALG_{Pxy \text{ Strat qs init x y}} \text{ by auto}\]

\text{unfolding } T_{\text{opt, def}} \text{ using len by auto}

\text{also have } \ldots \not\leq ALG_{xy, \text{det Strat qs init x y}} + ALG_{Pxy \text{ Strat qs init x y}} \text{ using } T_p \text{ by auto}

\text{finally show } \varnothing \text{thesis . } \text{qed}

\text{lemma } T_{\text{snoc}}: \text{length } rs = \text{length as} \implies T \text{ init } (rs@[r]) (as@[a]) = T \text{ init } rs \text{ as } + t_p (\text{steps}' \text{ init } rs \text{ as } (\text{length } rs)) r a \text{ apply(induct rs as arbitrary: init rule: list_induct2) by simp_all}

\text{lemma } \text{steps'}_{\text{snoc}}: \text{length } rs = \text{length as} \implies n = (\text{length as}) \implies \text{steps'} \text{ init } (rs@[r]) (as@[a]) (Suc n) = \text{step } (\text{steps'} \text{ init } rs \text{ as } n) r a \text{ apply(induct rs as arbitrary: init n r a rule: list_induct2)} \text{ by simp_all}

\text{lemma } \text{steps'}_{\text{take}}: \text{assumes } n < \text{length } qs \text{ length } qs = \text{length Strat} \text{ shows } \text{steps'} \text{ init } (\text{take } n \text{ qs}) (\text{take } n \text{ Strat}) n = \text{steps'} \text{ init } qs \text{ Strat } n \text{ proof} –

\text{have } \text{steps'} \text{ init } qs \text{ Strat } n = \text{steps'} \text{ init } (\text{take } n \text{ qs } @ \text{drop } n \text{ qs}) (\text{take } n \text{ Strat } @ \text{drop } n \text{ Strat}) n \text{ by simp}

\text{also have } \ldots = \text{steps'} \text{ init } (\text{take } n \text{ qs}) (\text{take } n \text{ Strat}) n \text{ apply(subst steps' rests[ symmetric]) using assms by auto}

\text{finally show } \varnothing \text{thesis by simp} \text{ qed}

\text{lemma } T_{p, \text{darstellung}}: \text{length } qs = \text{length Strat} \implies T_p \text{ init qs Strat } = (\sum_{i \in \{..< \text{length } qs\}}. t_p (\text{steps'} \text{ init } qs \text{ Strat } i) (qs!i) \text{ Strat!i})) \text{ proof –}
assume \( a \{ \text{simp} \}: \text{length } qs = \text{length } Strat \)

\{ fix \( n \) \}

have \( n \leq \text{length } qs \)
\[ \implies T_p \text{ init } (\text{take } n \text{ } qs) (\text{take } n \text{ } Strat) = \]
\[ (\sum i \in \{ .. < n \}. \ t_p \ (\text{steps'} \text{ init } qs \text{ Strat } i) \ (qs ! i) \ (Strat ! i)) \]

apply (induct \( n \))
apply (simp)
apply (simp add: take_Suc_conv_app_nth)
apply (subst T_snoc)
apply (simp)
by (simp add: min_def steps'

by auto
qed

lemma umformung_OPT':

assumes inlist: set \( qs \subseteq \text{set init} \)
assumes dist: distinct \( \text{init} \)
assumes qsStrat: \( \text{length } qs = \text{length } Strat \)
assumes noStupid: \( \forall x \ l. \ x < \text{length } Strat \implies l < \text{length } (\text{snd } (\text{Strat } ! x)) \)
\[ \implies \text{Suc } ((\text{snd } (\text{Strat } ! x))! l) < \text{length init} \]

shows \( T_p \text{ init } qs \text{ Strat } = \)
\[ (\sum (x,y) \in \{(x,y) :: 'a :: \text{linorder}). \ x \in \text{set init } \land y \in \text{set init } \land x < y}. \]
\[ \text{ALG}_{x y} \text{ det } \text{Strat } qs \text{ init } x \ y + \text{ALG}_{P x y} \text{ Strat } qs \text{ init } x \ y) \]

proof –

have \( (\sum i \in \{ .. < \text{length } qs \}. \)
\[ (\sum (x,y) \in \{(x,y). \ x \in \text{set init } \land y \in \text{set init } \land x < y}. \text{ALG}_P \ (\text{snd } (\text{Strat}! i)) \ x \ y \ (\text{steps'} \text{ init } qs \text{ Strat } i)) \)
\[ = (\sum i \in \{ .. < \text{length } qs \}. \)
\[ (\sum z \in \{(x,y). \ x \in \text{set init } \land y \in \text{set init } \land x < y}. \text{ALG}_P \ (\text{snd } (\text{Strat}! i)) \ (\text{fst } z) \ (\text{snd } z) \ (\text{steps'} \text{ init } qs \text{ Strat } i)) \)
by(auto simp: split_def)
also have …
\[ = (\sum z \in \{(x,y). \ x \in \text{set init } \land y \in \text{set init } \land x < y}. \]
\[ (\sum i \in \{ .. < \text{length } qs \}. \text{ALG}_P \ (\text{snd } (\text{Strat}! i)) \ (\text{fst } z) \ (\text{snd } z) \ (\text{steps'} \text{ init } qs \text{ Strat } i)) \)
by(rule sum.swap)
also have … = \( (\sum (x,y) \in \{(x,y). \ x \in \text{set init } \land y \in \text{set init } \land x < y}. \)
\[ (\sum i \in \{ .. < \text{length } qs \}. \text{ALG}_P \ (\text{snd } (\text{Strat}! i)) \ x \ y \ (\text{steps'} \text{ init } qs \)

235
let \( ?\text{config} = (\%i. \text{swaps} (\text{snd} \ (\text{Strat}!i)) \ (\text{steps}' \ \text{init} qs \ \text{Strat} i)) \)

also have \((\sum e \in \text{set init}. \ ALG e qs i \ (?\text{config} i, ()))\)

finally show \(?\text{case}\) apply(auto) using inlist by auto

qed simp

also have \((\sum y \in \text{set init}. \ (\sum i \in \{i. \ i < \text{length} qs \land qs!i = y\}. \ ALG x qs i \ (?\text{config} i, ()))\)

apply(rule sum.cong)

proof goal_cases

case 1

show ?case

apply(auto)

qed simp

also have \((\sum y \in \text{set init}. \ (\sum i \in \{i. \ i < \text{length} qs \land qs!i = y\}. \ ALG x qs i \ (?\text{config} i, ()))\)

apply(rule sum.UNION_disjoint)

apply(simp_all) by force

also have \((\sum y \in \text{set init}. \ \sum i \mid i < \text{length} qs \land qs!i = y. \ ALG x qs i \ (?\text{config} i, ()))\)

by(auto)

finally show ?case

qed (simp)
also have \[= (\sum (x, y) \in \{ (x, y) \mid x \in \text{set init} \land y \in \text{set init} \land x \neq y \}. \]
\[
(\sum i \in \{i \mid i < \text{length qs} \land qs!i = y\}. \ ALG x qs i (\text{?config i, ()}))
\]
by (rule sum.cartesian_product)
also have \[= (\sum (x, y) \in \{ (x, y) \mid x \in \text{set init} \land y \in \text{set init} \land x \neq y \}. \]
\[
(\sum i \in \{i \mid i < \text{length qs} \land qs!i = y\}. \ ALG x qs i (\text{?config i, ()}))
\]
by simp
also have \[E4: \[= (\sum (x, y) \in \{ (x, y) \mid x \in \text{set init} \land y \in \text{set init} \land x \neq y \}. \]
\[
(\sum i \in \{i \mid i < \text{length qs} \land qs!i = y\}. \ ALG x qs i (\text{?config i, ()}))
\]
(is \[(\sum (x, y) \in \text{?L. } \text{?f x y} = (\sum (x, y) \in \text{?R. } \text{?f x y}))\]
proof goal_cases

case 1

let \[?M = \{ (x, y) \mid x \in \text{set init} \land y \in \text{set init} \land x = y \}\]

have A: \[?L = ?R \cup ?M\] by auto

have B: \[\{} = ?R \cap ?M\] by auto

have \[(\sum (x, y) \in \text{?L. } \text{?f x y} = (\sum (x, y) \in ?R \cup ?M. \text{?f x y})\]

by (simp only: A)
also have \[= (\sum (x, y) \in ?R. \text{?f x y}) + (\sum (x, y) \in ?M. \text{?f x y})\]

apply (rule sum.union_disjoint)

apply (rule finite_subset [where B = \text{set init} \times \text{set init}])

apply (auto)
apply (rule finite_subset [where B = \text{set init} \times \text{set init}])

by (auto)
also have \[(\sum (x, y) \in \text{?M. } \text{?f x y} = 0)\]

apply (rule sum.neutral)

by (auto simp add: split_def before_in_def)

finally show \[\text{case by simp}\quad\text{qed}\]

also have \[= (\sum (x, y) \in \{ (x, y) \mid x \in \text{set init} \land y \in \text{set init} \land x < y \}. \]
\[
(\sum i \in \{i \mid i < \text{length qs} \land qs!i = y\}. \ ALG x qs i (\text{?config i, ()}))
\]
\[
+ (\sum i \in \{i \mid i < \text{length qs} \land qs!i = x\}. \ ALG y qs i (\text{?config i, ()}))
\]
(is \[(\sum (x, y) \in \text{?L. } \text{?f x y} = (\sum (x, y) \in ?R. \text{?f x y} + \text{?f y x}))\]

proof –

let \[?R' = \{ (x, y) \mid x \in \text{set init} \land y \in \text{set init} \land y < x \}\]

have A: \[?L = ?R \cup ?R'\] by auto

have \[\{} = ?R \cap ?R'\] by auto

have C: \[?R' = (\%(x, y). (y, x)) \cap ?R\] by auto

have D: \[(\sum (x, y) \in \text{?R'. } \text{?f x y} = (\sum (x, y) \in ?R. \text{?f y x})\]

proof –

have \[(\sum (x, y) \in \text{?R'. } \text{?f x y} = (\sum (x, y) \in \%(x, y). (y, x)) \cap ?R \cdot \text{?f x y})\]

by (simp only: C)
also have \[(\sum z \in \%(x, y). (y, x)) \cap ?R \cdot \%(x, y). \text{?f x y} z)\]
\[(\sum z \in \mathcal{R}. \ (\% (x, y). \ ?f x y \circ (\% (x, y). \ (y, x))) \ z)\]

apply\(\text{rule sum.reindex}\)
by\(\text{fact swap_inj}\)
also have \(\ldots = (\sum z \in \mathcal{R}. \ (\% (x, y). \ ?f y x) \ z)\)
apply\(\text{rule sum.cong}\)
by\(\text{auto}\)
finally show \(?\text{thesis}\) .

qed

have \((\sum (x, y) \in \mathcal{L}. \ ?f x y) = (\sum (x, y) \in \mathcal{R} \cup \mathcal{R}'. \ ?f x y)\)
by\(\text{simp only: A}\)
also have \(\ldots = (\sum (x, y) \in \mathcal{R}. \ ?f x y) + (\sum (x, y) \in \mathcal{R}'. \ ?f x y)\)

apply\(\text{rule sum.union_disjoint}\)
apply\(\text{rule finite_subset[where B=set init \times set init]}\)
apply\(\text{auto}\)
also have \(\ldots = (\sum (x, y) \in \mathcal{R}. \ ?f x y + ?f y x)\)
by\(\text{simp add: split_def sum.distrib[symmetric]}\)
finally show \(?\text{thesis}\) .

qed

also have \(E5: \ldots = (\sum (x, y) \in \{(x, y), \ x \in \text{set init} \land y \in \text{set init} \land x < y\}.\)
\((\sum i \in \{i. \ i < \text{length qs} \land (\text{qs!i=y} \lor \text{qs!i=x})\}. \ \text{ALG y qs i (\text{?config i, ()}}))\)
apply\(\text{rule sum.cong}\)
apply\(\text{simp}\)
proof \text{goal_cases}
case \(1 \ x\)
then obtain \(a \ b\) where \(x := (a, b)\) and \(a: a \in \text{set init} \ b \in \text{set init}\)
a < b by \text{auto}
then have \(a \neq b\) by \text{simp}
then have \text{disj: \{i. \ i < \text{length qs} \land \text{qs！i=a} \}} \cap \{i. \ i < \text{length qs} \land \text{qs！i=a} \}
by \text{auto}
also have \text{disj: \{i. \ i < \text{length qs} \land \text{qs！i=a} \}} \cap \{i. \ i < \text{length qs} \land \text{qs！i=a} \}
by \text{auto}
let \(?f=\%i. \ \text{ALG b qs i (\text{?config i, ()}}) + \text{ALG a qs i (\text{?config i, ()}}
let \(?B=\{i. \ i < \text{length qs} \land \text{qs！i=a} \}

238
let \( ?A = \{ i. \ i < \text{length} \ qs \land \ qs ! i = a \} \)

have \( (\sum \ i \in ?B \cup \ ?A. \ ?f \ i) \)
   \( = (\sum \ i \in ?B. \ ?f \ i) + (\sum \ i \in ?A. \ ?f \ i) - (\sum \ i \in ?B \cap \ ?A. \ ?f \ i) \)
apply(rule sum_UN_nat) by auto
also have \( \ldots = (\sum \ i \in ?B. \ \text{ALG} \ b \ qs \ i (\text{?config} \ i, ())) + \text{ALG} \ a \ qs \ i (\text{?config} \ i, ())) \)
   \( + (\sum \ i \in ?A. \ \text{ALG} \ b \ qs \ i (\text{?config} \ i, ())) + \text{ALG} \ a \ qs \ i (\text{?config} \ i, ())) \)
using disj by auto
also have \( \ldots = (\sum \ i \in ?B. \ \text{ALG} \ a \ qs \ i (\text{?config} \ i, ())) + (\sum \ i \in ?A. \ \text{ALG} \ b \ qs \ i (\text{?config} \ i, ())) \)
by (auto simp: split_def before_in_def)
finally
show \( ?\text{case} \ unfolding \ x \ apply(\text{simp add: split_def}) \)

unfolding unio by simp
qed
also have \( E6: \ldots = (\sum \ (x,y) \in \{(x,y). \ x \in \text{set init} \land y \in \text{set init} \land x < y\}. \ \text{ALGxy}_{\text{det Strat}} \ qs \ init \ x \ y) \)
apply(rule sum.cong)
unfolding \( \text{ALGxy}_{\text{det alternativ}} \ unfolding \ \text{ALG'}_{\text{det def}} \) by auto
finally have blockingpart: \( (\sum \ i \in \{..<\text{length} \ qs\}. \ \sum \ e \in \text{set init}. \ \text{ALG} \ e \ qs \ i (\text{?config} \ i, ())) \)
   \( = (\sum \ (x,y) \in \{(x,y). \ x \in \text{set init} \land y \in \text{set init} \land x < y\}. \ \text{ALGxy}_{\text{det Strat}} \ qs \ init \ x \ y) \)
.from \( T_p\text{darstellung}[OF qsStrat] \) have \( E0: T_p \ init \ qs \ Strat = \)
   \( (\sum \ i \in \{..<\text{length} \ qs\}. \ t_p \ (\text{steps' init qs Strat i}) \ (qsli) \ (Strat'!i)) \)
   by auto
also have \( \ldots = (\sum \ i \in \{..<\text{length} \ qs\}. \)
   \( (\sum \ e \in \text{set (steps' init qs Strat i)}. \ \text{ALG} \ e \ qs \ i (\text{swaps (snd (Strat'!i)) (steps' init qs Strat i)}, ()))) \)
   \( + (\sum \ (x,y) \in \{(x,y::('a::linorder)). \ x \in \text{set (steps' init qs Strat i}) \land y \in \text{set (steps' init qs Strat i}) \land x < y\}. \ \text{ALG}_{\text{P}} \ (\text{snd (Strat'!i)}) \ x \ y \ (\text{steps' init qs Strat i})) \)
apply(rule sum.cong)
apply(simp)
apply (rule \( t_p\text{sumofALGALGP} \))
apply (rule steps'_distinct2)
using dist qsStrat apply(simp_all)
apply(subst steps'_set)
using dist qsStrat inlist apply(simp_all)
apply fastforce
apply(subst steps'_length)
apply(simp_all)
using noStupid by auto
also have ... = (\(\sum_{i\in\{..<\text{length qs}\}}\))
  + (\(\sum_{(x,y)\in\{x,y\}. x \in \text{set init} \land y\in\text{set init} \land x<y}\). \(\text{ALG}_P (\text{snd (Strat!i)}) x y (\text{steps' init qs Strat i})\))
apply (rule sum.cong)
proof goal_cases
  case (1 x)
  then have set (steps' init qs Strat x) = set init
  apply (subst steps' set)
  using dist qsStrat 1 by (simp_all)
  then show ?case by simp
qed
also have ... = (\(\sum_{i\in\{..<\text{length qs}\}}\))
  + (\(\sum_{(x,y)\in\{x,y\}. x \in \text{set init} \land y\in\text{set init} \land x<y}\). \(\text{ALG}_P (\text{snd (Strat!i)}) x y (\text{steps' init qs Strat i})\))
by (simp only: blockingpart)
also have ... = (\(\sum_{(x,y)\in\{x,y\}. x \in \text{set init} \land y\in\text{set init} \land x<y}\). \(\text{ALG}_{xy} \text{det Strat qs init x y}\)
  + (\(\sum_{(x,y)\in\{x,y\}. x \in \text{set init} \land y\in\text{set init} \land x<y}\). \(\text{ALG}_P (\text{snd (Strat!i)}) x y (\text{steps' init qs Strat i})\))
by (simp only: paid_part)
also have ... = (\(\sum_{(x,y)\in\{x,y\}. x \in \text{set init} \land y\in\text{set init} \land x<y}\). \(\text{ALG}_{xy} \text{det Strat qs init x y}\)
  + (\(\sum_{(x,y)\in\{x,y\}. x \in \text{set init} \land y\in\text{set init} \land x<y}\). \(\text{ALG}_{Pxy} \text{Strat qs init x y}\))
by (simp add: sum.distrib split_def)
finally show ?thesis by auto
qed

lemma nn_contains_Inf:
fixes $S :: \text{nat set}$
assumes $nn: S \neq \emptyset$
shows $\inf S \in S$
using assms \text{Inf}_\text{nat_def} \text{LeastI} by force

lemma steps\_length: length $qs = \text{length as} \implies \text{length (steps s qs as)} = \text{length s}$
apply (induct $qs$ as arbitrary; s rule: list\_induct2)
by simp_all

lemma OPT\_noStupid:
fixes Strat
assumes [simp]: length Strat = length $qs$
assumes opt: $T_p \init qs \text{ Strat} = T_p\_\text{opt init qs}$
assumes init\_nempty: init $\neq \emptyset$
shows $\forall x \ l. x < \text{length Strat} \implies$
\[
 l < \text{length (snd (Strat \! x))} \implies 
\text{Suc ((snd (Strat \! x))!l)} < \text{length init}
\]
proof (rule ccontr, goal_cases)
case (1 $x$ $l$)

let $?sws' = \text{take l (snd (Strat!x)) @ drop (Suc l (snd (Strat!x))}$
let $?Strat' = \text{take x Strat @ (fst (Strat!x), }?sws' \# \text{ drop (Suc x) Strat}$

from 1(1) have valid: length $?Strat' = length $qs$ by simp
from valid have isin: $T_p \init qs \ ?Strat' \in \{ T_p \init qs \ as \ | as. \text{ length as} = \text{length qs} \}$ by blast

from 1(1,2) have lsws': length (snd (Strat!x)) = length $?sws' + 1
by (simp)

have a: (take $x$ $?Strat') = (take $x$ Strat)
  using 1(1) by (auto simp add: min_def takeSuc\_conv\_app\_nth)
have b: (drop (Suc $x$) Strat) = (drop (Suc $x$) $?Strat')
  using 1(1) by (auto simp add: min_def takeSuc\_conv\_app\_nth)

have aa: (take $l$ (snd (Strat!x))) = (take $l$ (snd (??Strat!x)))
  using 1(1,2) by (auto simp add: min_def takeSuc\_conv\_app\_nth\_nth\_append)
have bb: (drop (Suc $l$) (snd (Strat!x))) = (drop $l$ (snd (??Strat!x)))
  using 1(1,2) by (auto simp add: min_def takeSuc\_conv\_app\_nth\_nth\_append)

241
have \(\text{swaps} (\text{snd} (\text{Strat} ! x)) (\text{steps init} (\text{take} x \text{qs}) (\text{take} x \text{Strat})))\)
\[= (\text{swaps} (\text{take} l (\text{snd} (\text{Strat} ! x)) @ (\text{snd} (\text{Strat} ! x)))! l \# \text{drop} (\text{Suc} l) (\text{snd} (\text{Strat} ! x))) (\text{steps init} (\text{take} x \text{qs}) (\text{take} x \text{Strat})))\]

unfolding \(\text{id\_take\_nth\_drop}[\text{OF} 1(2), \text{symmetric}] \text{ by simp}\)
also have ...\[= (\text{swaps} (\text{take} l (\text{snd} (\text{Strat} ! x)) @ \text{drop} (\text{Suc} l) (\text{snd} (\text{Strat} ! x))) (\text{steps init} (\text{take} x \text{qs}) (\text{take} x \text{Strat})))\]

using \(1(3) \text{ by (simp add: swap\_def steps\_length)}\)
finally have noeffect: \(\text{swaps} (\text{snd} (\text{Strat} ! x)) (\text{steps init} (\text{take} x \text{qs}) (\text{take} x \text{Strat})))\)
\[= (\text{swaps} (\text{take} l (\text{snd} (\text{Strat} ! x)) @ \text{drop} (\text{Suc} l) (\text{snd} (\text{Strat} ! x))) (\text{steps init} (\text{take} x \text{qs}) (\text{take} x \text{Strat})))\]
.

have \(c: t_p (\text{steps init} (\text{take} x \text{qs}) (\text{take} x \text{Strat})) (\text{qs} ! x) (\text{Strat} ! x) = t_p (\text{steps init} (\text{take} x \text{qs}) (\text{take} x \text{Strat}')) (\text{qs} ! x) (\text{Strat}' ! x) + 1\)
unfolding a \(t_p\_def\) using \(1(1,2)\)
apply (simp add: min\_def split\_def nth\_append) unfolding noeffect
by (simp)

have \(T_p \text{ init} (\text{take} (\text{Suc} x) \text{qs}) (\text{take} (\text{Suc} x) \text{Strat})\)
\[= T_p \text{ init} (\text{take} x \text{qs}) (\text{take} x \text{Strat}') + t_p (\text{steps init} (\text{take} x \text{qs}) (\text{take} x \text{Strat})) (\text{qs} ! x) (\text{Strat} ! x)\]
using \(1(1) \text{ a by } (\text{simp add: take\_Suc\_conv\_app\_nth T\_append})\)
also have ... = \(T_p \text{ init} (\text{take} x \text{qs}) (\text{take} x \text{Strat}') + t_p (\text{steps init} (\text{take} x \text{qs}) (\text{take} x \text{Strat}')) (\text{qs} ! x) (\text{Strat}' ! x) + 1\)
unfolding c by (simp)
also have ... = \(T_p \text{ init} (\text{take} (\text{Suc} x) \text{qs}) (\text{take} (\text{Suc} x) \text{Strat}') + 1\)
using \(1(1) \text{ a by } (\text{simp add: min\_def take\_Suc\_conv\_app\_nth T\_append nth\_append})\)
finally have bef: \(T_p \text{ init} (\text{take} (\text{Suc} x) \text{qs}) (\text{take} (\text{Suc} x) \text{Strat})\)
\[= T_p \text{ init} (\text{take} (\text{Suc} x) \text{qs}) (\text{take} (\text{Suc} x) \text{Strat}') + 1\)

let \(?\text{interstate} = (\text{steps init} (\text{take} (\text{Suc} x) \text{qs}) (\text{take} (\text{Suc} x) \text{Strat}))\)
let \(?\text{interstate}' = (\text{steps init} (\text{take} (\text{Suc} x) \text{qs}) (\text{take} (\text{Suc} x) \text{Strat}'))\)

have state: \(?\text{interstate}' = ?\text{interstate}\)
using \(1(1) \text{ apply } (\text{simp add: take\_Suc\_conv\_app\_nth min\_def})\)
apply (simp add: steps\_append step\_def split\_def) using noeffect by simp
have \( T_p \) init qs Strat
   \( = T_p \) init \((\text{take} (\text{Suc} \ x) \ qs \ @ \ \text{drop} (\text{Suc} \ x) \ qs) \ \text{take} (\text{Suc} \ x) \ Strat \ @ \ \text{drop} (\text{Suc} \ x) \ Strat \)
   by simp
also have \( \ldots = T_p \) init \((\text{take} (\text{Suc} \ x) \ qs) \ \text{take} (\text{Suc} \ x) \ Strat \)
   \( + T_p \) ?interstate \((\text{drop} (\text{Suc} \ x) \ qs) \ \text{drop} (\text{Suc} \ x) \ Strat \)
   apply\(\text{(subst} \ T_{\text{append2}})\) by\(\text{(simp\_all)}\)
also have \( \ldots = T_p \) init \((\text{take} (\text{Suc} \ x) \ qs) \ \text{take} (\text{Suc} \ x) \ Strat \)'
   \( + T_p \) ?interstate' \((\text{drop} (\text{Suc} \ x) \ qs) \ \text{drop} (\text{Suc} \ x) \ Strat \)'
unfolding bef state using \(1(1)\) by\(\text{(simp\_add: min\_def nth\_append)}\)
also have \( \ldots = T_p \) init \((\text{take} (\text{Suc} \ x) \ qs \ @ \ \text{drop} (\text{Suc} \ x) \ qs) \ \text{take} (\text{Suc} \ x) \ Strat \)'
   \( @ \ \text{drop} (\text{Suc} \ x) \ Strat \)'
   + 1
   apply\(\text{(subst} \ T_{\text{append2}})\) using \(1(1)\) by\(\text{(simp\_all\_add: min\_def)}\)
also have \( \ldots = T_p \) init qs ?Strat' + 1 by simp
finally have better: \( T_p \) init qs ?Strat' + 1 = \( T_p \) init qs Strat by simp
have \( T_p \) init qs ?Strat' + 1 = \( T_p \) init qs Strat by \( \text{(fact better)}\)
also have \( \ldots = T_p\text{-opt init qs by} \ (\text{fact opt)}\)
also from \( \text{cInf\_lower[OF isin]}\) have \( \ldots \leq T_p \) init qs ?Strat' unfolding \(T_{\text{opt\_def}}\) by simp
finally show False using init\_nempty by auto
qed

lemma umformung\_OPT:
  assumes inlist: \( \text{set} \ qs \subseteq \text{set} \ init \)
  assumes dist: distinct init
  assumes a: \( T_{p\text{-opt init qs}} = T_p \) init qs Strat
  assumes b: length qs = length Strat
  assumes c: init\(\neq[]\)
shows \( T_{p\text{-opt init qs}} = \)
  \((\sum (x,y)\in\{(x,y::(a::\text{linorder}))\}. \ x \in \text{set} \ init \ \land \ y\in\text{set} \ init \ \land \ x<y\}).
  ALGxy\_det Strat qs init x y + ALG\_Pxy Strat qs init x y\)
proof –
  have \( T_{p\text{-opt init qs}} = T_p \) init qs Strat by\(\text{(fact a)}\)
also have \( \ldots = \)
  \((\sum (x,y)\in\{(x,y::(a::\text{linorder}))\}. \ x \in \text{set} \ init \ \land \ y\in\text{set} \ init \ \land \ x<y\}).
  ALGxy\_det Strat qs init x y + ALG\_Pxy Strat qs init x y\)
  apply\(\text{(rule umformung\_OPT)}\)
  apply\(\text{(fact)}\)+
    using \(\text{OPT\_noStupid[OF b|symmetric] a|symmetric} \ c\) apply\(\text{(simp)}\) done

243
corollary $OPT_{zerlegen}$:

assumes

- $dist: \text{distinct init}$
- $c: \text{init} \neq []$

and $\text{setqsinit: set qs} \subseteq \text{set init}$

shows $(\sum (x,y) \in \{(x,y):(a::\text{linorder})\}. x \in \text{set init} \land y \in \text{set init} \land x < y).$

$(T_{p, opt} (Lxy \text{ init } \{x,y\}) (Lxy \text{ qs } \{x,y\})) \leq T_{p, opt} \text{ init qs}$

proof

- have $T_{p, opt} \text{ init qs} \in \{T_p \text{ init qs as } | \text{as, length as} = \text{length qs}\}$

unfolding $T_{opt, def}$

apply (rule $\text{nn\_contains\_Inf}$)

apply (auto) by (rule $\text{Ex\_list\_of\_length}$)

then obtain $\text{Strat where a: T_p \text{ init qs Strat} = T_{p, opt} \text{ init qs}}$

and $b: \text{length Strat} = \text{length qs}$

unfolding $T_{opt, def}$ by auto

have $(\sum (x,y) \in \{(x,y). x \in \text{set init} \land y \in \text{set init} \land x < y\}. T_{p, opt} (Lxy \text{ init } \{x,y\}) (Lxy \text{ qs } \{x,y\})) \leq (\sum (x,y) \in \{(x,y). x \in \text{set init} \land y \in \text{set init} \land x < y\}. ALG_{xy, det} \text{ Strat qs init x y} + ALG_{Pxy} \text{ Strat qs init x y})$

apply (rule $\text{sum\_mono}$)

apply (auto)

proof goal_cases

- case (1 a b)

then have $a \neq b$ by auto

show ?case apply (rule $T1_7[OF \ a\ b]$) by (fact)+

qed

also from umformung $OPT[OF\ \text{setqsinit\ dist}] \ a\ b\ c$ have $\ldots = T_p \text{ init qs Strat by auto}$

also from $a$ have $\ldots = T_{p, opt} \text{ init qs by simp}$

finally show ?thesis.

qed

14.5 Factoring Lemma

lemma $\text{cardofpairs: S} \neq [] \implies \text{sorted S} \implies \text{distinct S} \implies \text{card } \{(x,y). x \in \text{set S} \land y \in \text{set S} \land x < y\} = ((\text{length S}) \ast (\text{length S} - 1)) / 2$
proof (induct \(S\) rule: list_nonempty_induct)
case (cons \(s\) \(ss\))
then have \(\text{sorted} \ ss\ \text{distinct} \ ss\ \text{by}\) auto
from \(\text{cons}(2)\)[OF \(\text{this}(1)\) \(\text{this}(2)\)] have \(\text{iH}\): card \(\{(x, y) . \ x \in \text{set} \ ss\ \land \ y \in \text{set} \ ss\ \land \ x < y\}\)
\(= (\text{length} \ ss \ast (\text{length} \ ss - 1)) / 2\)
by auto

from \(\text{cons}\) have \(\text{sss}\): \(s \notin \text{set} \ ss\) by auto

from \(\text{cons}\) have \(\text{tt}\): \((\forall y \in \text{set} \ (s \# \text{ss}). \ s \leq y)\) by auto
with \(\text{cons}\) have \(\text{tt}'\): \((\forall y \in \text{set} \ ss. \ s < y)\)
proof –
from \(\text{sss}\) have \((\forall y \in \text{set} \ ss. \ s \neq y)\) by auto
with \(\text{tt}\) show \(\text{thesis}\) by fastforce
qed

then have \(\{(x, y) . \ x = s \land y \in \text{set} \ ss\ \land \ x < y\}\)
\(= \{(x, y) . \ x = s \land y \in \text{set} \ ss\}\) by auto
also have \(\ldots = \{s\} \times (\text{set} \ ss)\) by auto
finally have \(\{(x, y) . \ x = s \land y \in \text{set} \ ss\ \land \ x < y\}\)
then have \(\text{card} \ \{(x, y). \ x = s \land y \in \text{set} \ ss\ \land \ x < y\}\)
\(= \text{card} \ (\text{set} \ ss)\) by(auto)
also from \(\text{cons}\) distinct_card have \(\ldots = \text{length} \ ss\) by auto
finally have step: \(\text{card} \ \{(x, y). \ x = s \land y \in \text{set} \ ss\ \land \ x < y\}\)
\(= \text{length} \ ss\).

have \(\text{uni}\): \(\{(x, y) . \ x \in \text{set} \ (s \# \text{ss})\ \land \ y \in \text{set} \ (s \# \text{ss})\ \land \ x < y\}\)
\(= \{(x, y) . \ x \in \text{set} \ ss\ \land \ y \in \text{set} \ ss\ \land \ x < y\}\)
\(\cup \{(x, y) . \ x = s \land y \in \text{set} \ ss\ \land \ x < y\}\)
using \(\text{tt}\) by auto

have \(\text{disj}\): \(\{(x, y) . \ x \in \text{set} \ ss\ \land \ y \in \text{set} \ ss\ \land \ x < y\}\)
\(\cap \{(x, y) . \ x = s \land y \in \text{set} \ ss\ \land \ x < y\}\) = \(
\)
using \(\text{sss}\) by(auto)
have \(\text{card}\): \(\{(x, y) . \ x \in \text{set} \ (s \# \text{ss})\ \land \ y \in \text{set} \ (s \# \text{ss})\ \land \ x < y\}\)
\(= \text{card} \ \{(x, y) . \ x \in \text{set} \ ss\ \land \ y \in \text{set} \ ss\ \land \ x < y\}\)
\(\cup \{(x, y) . \ x = s \land y \in \text{set} \ ss\ \land \ x < y\}\) using \(\text{uni}\) by auto
also have \(\ldots = \text{card}\): \(\{(x, y) . \ x \in \text{set} \ ss\ \land \ y \in \text{set} \ ss\ \land \ x < y\}\)
\(+ \text{card} \ \{(x, y) . \ x = s \land y \in \text{set} \ ss\ \land \ x < y\}\)
apply(rule card_Un_disjoint)
apply(rule finite_subset[where \(B=(\text{set} \ ss) \times (\text{set} \ ss)]\)
apply(force)
apply(simp)
apply (rule finite_subset [where $B = \{s\} \times (\text{set } ss)\])
apply (force)
apply (simp)
using disj apply (simp) done
also have \ldots = \frac{\text{length } ss \times (\text{length } ss - 1)}{2} + \text{length } ss \text{ using } \text{iH } \text{step by auto}
also have \ldots = \frac{\text{length } ss \times (\text{length } ss - 1) + 2 \times \text{length } ss}{2} \text{ by auto}
also have \ldots = \frac{\text{length } ss \times (\text{length } ss - 1) + \text{length } ss \times 2}{2} \text{ by auto}
also have \ldots = \frac{\text{length } ss \times (\text{length } ss - 1 + 2)}{2} \text{ by simp}
also have \ldots = \frac{\text{length } ss \times (\text{length } ss + 1)}{2} \text{ using } \text{cons}(1) \text{ by simp}
also have \ldots = \frac{(\text{length } ss + 1) \times \text{length } ss}{2} \text{ by auto}
also have \ldots = \frac{\text{length } (s\#ss) \times (\text{length } (s\#ss) - 1)}{2} \text{ by auto}
finally show ?case by auto
next
case single thus ?case by (simp cong: conj_cong)
qed

lemma factoringlemma_withconstant:
  fixes $A$
  and $b :: \text{real}$
  and $c :: \text{real}$
  assumes $c: c \geq 1$
  assumes dist: $\forall e \in S0. \ \text{distinct } e$
  assumes notempty: $\forall e \in S0. \ \text{length } e > 0$
  assumes pw: pairwise $A$
  assumes on2: $\forall s0 \in S0. \ \exists b \geq 0. \ \forall qs \subseteq \text{set } x. \ \text{set } x \subseteq \text{set } s0, \ \forall (x,y) \in \{(x,y). x \in \text{set } s0 \land y \in \text{set } s0 \land x < y\}. \ T_{p\cdot\text{rand}} A (Lxy s0 \{x,y\}) (Lxy qs \{x,y\}) \leq c \times (T_{p\cdot\text{opt}} (Lxy s0 \{x,y\}) (Lxy qs \{x,y\}) + b$
  assumes nopaid: $\forall is s q. \ \forall ((\text{free}, \text{paid}), \ldots) \in (\text{snd } A (s, is) q). \ \text{paid=}[]$
  assumes 4: $\forall \text{init } qs. \ \text{distinct } \text{init } \Longrightarrow \text{set } qs \subseteq \text{set } \text{init } \Longrightarrow (\forall x. \ x < \text{length } qs \Longrightarrow \text{finite } (\text{set pmf (config}'' A \ \text{qs init } x)))$
  shows $\forall s0 \in S0. \ \exists b \geq 0. \ \forall qs \subseteq \text{set } x. \ \text{set } x \subseteq \text{set } s0, \ T_{p\cdot\text{rand}} A s0 qs \leq c \times \text{real } (T_{p\cdot\text{opt}} s0 qs) + b$
proof (standard, goal_cases)
case (1 init)
  have $d$: distinct init using dist 1 by auto
  have $d2$: init $\neq []$ using notempty 1 by auto

246
obtain \( b \) where on3: \( \forall \, q_3 \in \{ x. \, \text{set } x \subseteq \text{set init} \}. \forall (x,y) \in \{(x,y). \, x \in \text{set init} \land y \in \text{set init} \land x < y \}. \, T_{p,\text{on_rand}} \, A \, (Lxy \, \text{init} \, \{x,y\}) \, (Lxy \, \text{qs} \, \{x,y\}) \leq c \ast (T_{p,\text{opt}} \, (Lxy \, \text{init} \, \{x,y\})) \, (Lxy \, \text{qs} \, \{x,y\}) + b \)

and \( b: b \geq 0 \)

using on2 1 by auto

\{

fix \( q_3 \)
assume drin: set \( q_3 \subseteq \text{set init} \)

have \( T_{p,\text{on_rand}} \, A \, \text{init} \, \text{qs} = \)
\((\sum (x,y) \in \{(x,y). \, x \in \text{set init} \land y \in \text{set init} \land x < y \}. \, T_{p,\text{on_rand}} \, A \, (Lxy \, \text{init} \, \{x,y\}) \, (Lxy \, \text{qs} \, \{x,y\})) \)

apply(rule umf_pair)
apply(fact)+
using \( \{\text{of init } q_3\} \, \text{drin} \, \text{d by(simp add: split_def)} \)

also have \( \ldots \leq (\sum (x,y) \in \{(x,y). \, x \in \text{set init} \land y \in \text{set init} \land x < y \}. \, c \ast \)
\((T_{p,\text{opt}} \, (Lxy \, \text{init} \, \{x,y\})) \, (Lxy \, \text{qs} \, \{x,y\})) + b) \)

apply(rule sum_mono)
using on3 \, \text{drin} \, \text{by(simp add: split_def)}
also have \( \ldots = c \ast (\sum (x,y) \in \{(x,y). \, x \in \text{set init} \land y \in \text{set init} \land x < y \}. \, T_{p,\text{opt}} \, (Lxy \, \text{init} \, \{x,y\}) \, (Lxy \, \text{qs} \, \{x,y\})) + b \ast (((\text{length init}) \ast (\text{length init} - 1)) \)

/ 2)
proof –

\{

fix \( S::\text{a list} \)
assume dis: distinct \( S \)
assume d2: \( S \neq [] \)
then have d3: \( \text{sort } S \neq [] \) by (metis length_0_conv length_sort)
have card \( \{ (x,y). \, x \in \text{set } S \land y \in \text{set } S \land x < y \} \)
\= card \( \{(x,y). \, x \in \text{set (sort } S \) \land y \in \text{set (sort } S \) \land x < y \} \)
by auto
also have \( \ldots = (\text{length (sort } S) \ast (\text{length (sort } S) - 1)) \) / 2
apply(rule cardofpairs) using dis d2 d3 by (simp_all)
finally have card \( \{(x,y). \, x \in \text{set } S \land y \in \text{set } S \land x < y \} = \)
\((\text{length (sort } S) \ast (\text{length (sort } S) - 1)) \) / 2 .
\}

with \( d \, d2 \) have e: \( \text{card } \{(x,y). \, x \in \text{set init} \land y \in \text{set init} \land x < y \} = \)
\(((\text{length init}) \ast (\text{length init} - 1)) \) / 2 by auto

247
show \( ?\text{thesis} = \left( \sum_{(x, y) \in S} c \ast (?T x y) + b \right) = c \ast \tau R + b \ast ?T2 \)

proof –
  have \( \left( \sum_{x, y \in S} c \ast (?T x y) + b \right) = \)
  \( c \ast \left( \sum_{x, y \in S} (?T x y) \right) + \left( \sum_{x, y \in S} b \right) \)
  by (simp add: split_def sum_distrib sum_distrib_left)
  also have \( \ldots = c \ast \left( \sum_{x, y \in S} (?T x y) \right) + b \ast ?T2 \)
  using \( e \) by (simp add: split_def)
  finally show \( ?\text{thesis} \) by (simp add: split_def)
  qed

qed

also have \( \ldots \leq c \ast \tau p_{\text{opt}} \text{ init q} + (b \ast ((\text{length init})^\ast(\text{length init}-1)) / 2) \)

proof –
  have \( \left( \sum_{x, y \in (x, y) . x \in \text{set init} \land y \in \text{set init} \land x < y} \right) \tau p_{\text{opt}} (Lxy \text{ init} \{x, y\}) (Lxy \text{ qs} \{x, y\}) \)
  \( \leq \tau p_{\text{opt}} \text{ init q} \)
  using OPT_zerlegen drin d d2 by auto
  then have \( \text{real} \left( \sum_{x, y \in (x, y) . x \in \text{set init} \land y \in \text{set init} \land x < y} \right) \tau p_{\text{opt}} (Lxy \text{ init} \{x, y\}) (Lxy \text{ qs} \{x, y\}) \)
  \( \leq \left( \tau p_{\text{opt}} \text{ init q} \right) \)
  by linarith
  with \( c \) show \( ?\text{thesis} \) by (auto simp: split_def)
  qed

finally have \( f : \tau p_{\text{on rand}} A \text{ init q} \leq c \ast \text{real} (\tau p_{\text{opt}} \text{ init q}) + (b \ast ((\text{length init})^\ast(\text{length init}-1)) / 2) \).

note all=this

show ?case unfolding compet_def
  apply (auto)
  apply (rule exI [where \( x=(b \ast ((\text{length init})^\ast(\text{length init}-1)) / 2) \)])
  apply (safe)
  using notempty 1 b apply simp
  using all b by simp
  qed

lemma factoringlemma_withconstant:*
  fixes \( A \)
  and \( b::\text{real} \)
  and \( c::\text{real} \)
  assumes \( c : c \geq 1 \)
  assumes dist: \( \forall e \in S0. \text{ distinct e} \)
  assumes notempty: \( \forall e \in S0. \text{ length e} > 0 \)
  assumes pw: pairwise A

248
**assumes on2**: ∀ s0 ∈ S0. ∃ b ≥ 0. ∀ qs ∈ {x. set x ⊆ set s0}. ∀ (x,y) ∈ {(x,y). x ∈ set s0 ∧ y ∈ set s0 ∧ x < y}. \( T_p\cdot\text{on}_2\cdot\text{rand} A (Lxy s0 \{x,y\}) (Lxy qs \{x,y\}) \leq c \ast (T_p\cdot\text{opt} (Lxy s0 \{x,y\}) (Lxy qs \{x,y\})) + b

**assumes nopaid**: \( \forall s q. \forall ((\text{free}, \text{paid}), \text{is}) \in (\text{snd} A (s, \text{is}) q). \text{paid} = [] \)

**assumes 4**: \( \forall s q. \forall ((\text{free}, \text{paid}), \text{is}) \in (\text{snd} A (s, \text{is}) q). \text{paid} = [] \)

**shows** compet_rand A c S0

unfolding compet_rand_def static_def using factoringlemma_withconstant[OF assms] by simp

end

15 TS: another 2-competitive Algorithm

theory TS
imports
OPT2
Phase_Partitioning
Move_to_Front
List_Factoring
RExp_Var
begin

15.1 Definition of TS

definition TS_step_d where
TS_step_d s q = ((
  let li = index (snd s) q in
  (if li = length (snd s) then 0 — requested for first time
    else (let sincelast = take li (snd s) in
      let S = {x. x < q in (fst s) ∧ count_list sincelast x ≤ 1} in
        (if S = {} then 0
          else (index (fst s) q) − Min ( (index (fst s)) ' S)))
  )
  )
),[], q#(snd s))
```plaintext
definition rTS :: nat list ⇒ (nat, nat list) alg_on  where rTS h = ((λs. h), TS_step_d)

fun TSstep where
TSstep qs n (is, s)
  = ((qs!n)#is, 
    step s (qs!n) ((
      let li = index is (qs!n) in 
      (if li = length is then 0 — requested for first time 
       else (let sincelast = take li is 
          in (let S={x. x < (qs!n) in s ∧ count_list sincelast x ≤ 1} 
            in 
            (if S={} then 0 
            else 
            (index s (qs!n)) − Min ( (index s) ‘ S)))))
    )
  ),[]))

lemma TSnopaid: (snd (fst (snd (rTS initH) is q))) = []
unfolding rTS_def by(simp add: TS_step_d_def)

abbreviation TSdet where
    TSdet init initH qs n == config (rTS initH) init (take n qs)

lemma TSdet_Suc: Suc n ≤ length qs ⇒ TSdet init initH qs (Suc n) = 
    Step (rTS initH) (TSdet init initH qs n) (qs!n)
by(simp add: take_Suc_conv_app_nth config_snoc)

definition s_TS where s_TS init initH qs n = fst (TSdet init initH qs n)

lemma sndTSdet: n≤length xs ⇒ snd (TSdet init initH xs n) = rev (take n xs) @ initH
apply(induct n)
apply(simp add: rTS_def)
by(simp add: split_def TS_step_d_def take_Suc_conv_app_nth config_snoc Step_def rTS_def)
```

250
15.2 Behaviour of TS on lists of length 2

lemma
  fixes hs x y
  assumes x ≠ y
  shows oneTS_step : TS_step_d ([x, y], x#y#hs) y = ((1, []), y # x # y # hs)
  and oneTS_stepyy: TS_step_d ([x, y], y#x#hs) y = ((Suc 0, []), y # x # y # hs)
  and oneTS_stepx: TS_step_d ([x, y], x#x#hs) y = ((0, []), y # x # x # hs)
  and oneTS_stepy: TS_step_d ([x, y], [x]) y = ((Suc 0, []), y # y)
  using assms by (auto simp add: step_def mtf2_def swap_def TS_step_d_def before_in_def)

lemmas oneTS_steps = oneTS_stepx oneTS_stepxy oneTS_stepyx oneTS_stepy oneTS_stepyy oneTS_stepxy oneTS_stepyy oneTS_stepx

15.3 Analysis of the Phases

definition TS_inv c x i ≡ (∃ hs. c = return_pmf (if x=hd i then i else rev i),[x,x]@hs)
  ∨ c = return_pmf (if x=hd i then i else rev i),[])

lemma TS_inv_sym: a ≠ b → {a,b}={x,y} → z ∈ {x,y} → TS_inv c z [a,b] = TS_inv c z [x,y]
  unfolding TS_inv_def by auto

abbreviation TS_inv' s x i == TS_inv (return_pmf s) x i

lemma TS_inv'_det: TS_inv' s x i = (∃ hs. s = ((if x=hd i then i else rev i),[x,x]@hs))
  ∨ s = ((if x=hd i then i else rev i),[])
  unfolding TS_inv_def by auto

lemma TS_inv'_det2: TS_inv' (s,h) x i = (∃ hs. (s,h) = ((if x=hd i then i else rev i),[x,x]@hs))
  ∨ (s,h) = ((if x=hd i then i else rev i),[])
  unfolding TS_inv_def by auto

251
15.3.1 \((yx)\) *

**lemma** \(TS_{yx}:\) \textit{assumes} \(x \neq y\) \(qs \in \text{lang} (\text{Star}(\text{Times} (\text{Atom} y) (\text{Atom} x)))\)

\[
\exists hs. h = [x, y] @ hs
\]

\textit{shows} \(T_{on'} (rTS h0) ([x, y], h) (qs @ r) = \text{length} \ v + T_{on'} (rTS h0) ([x, y], ((\text{rev} \ qs) @ h)) \ r
\]

\[
\land (\exists hs. ((\text{rev} \ qs) @ h) = [x, y] @ hs)
\]

\[
\land \text{config}' (rTS h0) ([x, y], h) \ qs = ([x, y], \text{rev} \ qs @ h)
\]

**proof** –

\textit{from} \textit{assms} \textit{have} \(qs \in \text{star} ([\{y\}] @ @[\{x\}])\) \textit{by} (\textit{simp})

\textit{from this \textit{assms}(3) show} \(?\text{thesis}\)

**proof** \((\text{induct} \ qs \ \text{arbitrary}; \ h \ \text{rule: star_induct})\)

\textit{case} \(\text{Nil}\)

\textit{then show} \(?\text{case by}(\text{simp add: rTS_def})\)

**next**

\textit{case} (\(\text{append} \ u \ v\))

\textit{then have} \(uyx: u = [y, x]\) \textit{by} \(\text{auto}\)

\textit{from} \(\text{append} \ \text{obtain} \ hs \ \text{where} \ a: h = [x, y] @ hs\) \textit{by} \(\text{blast}\)

\[
\text{have} \ T_{on'} (rTS h0) ([x, y], (\text{rev} \ u @ h)) (v @ r) = \text{length} \ v + T_{on'} (rTS h0) ([x, y], (\text{rev} \ u @ h)) \ r
\]

\[
\land (\exists hs. \text{rev} \ v @ (\text{rev} \ u @ h) = [x, y] @ hs)
\]

\[
\land \text{config}' (rTS h0) ([x, y], (\text{rev} \ u @ h)) \ v = ([x, y], \text{rev} \ v @ (\text{rev} \ u @ h))
\]

\textit{apply}(\textit{simp only: uyx}) \textit{apply}(\textit{rule append(3)}) \textit{by} \textit{simp}

\textit{then have} \(yy: T_{on'} (rTS h0) ([x, y], (\text{rev} \ u @ h)) (v @ r) = \text{length} \ v + T_{on'} (rTS h0) ([x, y], (\text{rev} \ u @ h)) \ r
\]

\[
\land \text{history:} (\exists hs. \text{rev} \ v @ (\text{rev} \ u @ h) = [x, y] @ hs)
\]

\[
\land \text{state:} \text{config}' (rTS h0) ([x, y], (\text{rev} \ u @ h)) \ v = ([x, y], \text{rev} \ v @ (\text{rev} \ u @ h))\) \textit{by} \textit{auto}
\]

\textit{have} \(s0: s_{TS} [x, y] h [y, x] 0 = [x, y]\) \textit{unfolding} \(s_{TS_{def}}\) \textit{by}(\textit{simp})

\textit{from} \textit{assms(1) have} \(hahah: \ \{xa. xa < y \in [x, y] \land \text{count_list} [x] xa \leq 1\} = \{x\}\)

\textit{unfolding} \(\text{before_in_def}\) \textit{by} \(\text{auto}\)

\textit{have} \(\text{config}' (rTS h0) ([x, y], h) \ u = ([x, y], x \# y \# x \# y \# hs)\)

\textit{apply}(\textit{simp add: split_def rTS_def uyx a})

\textit{using} \textit{assms(1) by}(\textit{auto simp add: Step_def oneTS_steps step_def mtf2_def swap_def})

252
then have s2: config’ (rTS h0) ([x, y], h) u = ([x, y], ((rev u) @} h)) unfolding a uyx by simp

have config’ (rTS h0) ([x, y], h) (u @} v) =
    config’ (rTS h0) (Partial_Cost_Model.config’ (rTS h0) ([x, y], h) u) v by (rule config’_append2)
also
have ... = config’ (rTS h0) ([x, y], ((rev u) @} h)) v by (simp only: s2)
also
have ... = ([x, y], rev (u @} v) @} h) by (simp add: state)
finally
have alles: config’ (rTS h0) ([x, y], h) (u @} v) = ([x, y], rev (u @} v) @} h).

have ta: T_on’ (rTS h0) ([x,y],h) u = 2 unfolding rTS_def uyx a apply(simp only: T_on’.simpss(2))
    using assms(1) apply(auto simp add: Step_def step_def mtf2_def swap_def oneTS_steps)
by(simp add: t_p_def)

have T_on’ (rTS h0) ([x,y],h) ((u @} v) @} r)
    = T_on’ (rTS h0) ([x,y],h) (u @} (v @} r)) by auto
also have ... = T_on’ (rTS h0) ([x,y],h) u
    + T_on’ (rTS h0) (config’ (rTS h0) ([x, y],h) u) (v @} r)
by(rule T_on’_append)
also have ... = T_on’ (rTS h0) ([x,y],h) u
    + T_on’ (rTS h0) ([x, y],(rev u @} h)) (v @} r) by(simp only: s2)
also have ... = T_on’ (rTS h0) ([x,y],h) u + length v + T_on’ (rTS h0) ([x, y], rev v @} (rev u @} h)) r by(simp only: yy)
also have ... = 2 + length v + T_on’ (rTS h0) ([x, y], rev v @} (rev u @} h)) r by(auto)
also have ... = length (u @} v) + T_on’ (rTS h0) ([x, y], rev v @} (rev u @} h)) r using uyx by auto
also have ... = length (u @} v) + T_on’ (rTS h0) ([x, y], (rev (u @} v) @} h)) r by auto
finally show ?case using history alles by simp
qed
qed
15.3.2 \( x \)

**Lemma** \( TS_{x^l} \): \( T_{on'} (rTS h0) ([x,y],h) [x] = 0 \land config' (rTS h0) ([x, y],h) [x] = ([x,y], rev [x] @ h) \)

by(auto simp add: t_p_def rTS_def TS_step_d_def Step_def step_def)

15.3.3 \( yy \)

**Lemma** \( TS_{yy} \): assumes \( x \neq y \exists hs. h = [x, y] @ hs \)

shows \( T_{on'} (rTS h0) ([x,y],h) [y, y] = 1 config' (rTS h0) ([x, y],h) [y,y] = ([y,x], rev [y,y] @ h) \)

**Proof**

from assms obtain hs where a: h = [x,y]@hs by blast

from a show \( T_{on'} (rTS h0) ([x,y],h) [y, y] = 1 \)

unfolding rTS_def

using assms(1) apply(auto simp add: oneTS_steps Step_def step_def mtf2_def swap_def)

by(simp add: t_p_def)

show config' (rTS h0) ([x,y],h) [y,y] = ([y,x], rev [y,y] @ h)

unfolding rTS_def a using assms(1)

by(simp add: Step_def oneTS_steps step_def mtf2_def swap_def)

qed

15.3.4 \( yx(yx)^* \)

**Lemma** \( TS_{yxxy} \): assumes \( simp\): \( x \neq y \) and \( qs \in lang \ (seq[ Times (Atom y) (Atom x)])) \)

\((\exists hs. h=[x,x]@hs) \lor \ index h y = length h \)

shows \( T_{on'} (rTS h0) ([x,y],h) (qs@r) = length qs - 1 + T_{on'} (rTS h0) ([x,y],rev qs @ h) r \)

\( \land (\exists hs. (rev qs @ h) = [x, y] @ hs) \)

\( \land config' (rTS h0) ([x, y],h) qs = ([x,y], rev qs @ h) \)

**Proof**

obtain u v where uu: u \( \in \) lang \( (Times (Atom y) (Atom x)) \)

and vv: v \( \in \) lang \( (seq[ Star(Times (Atom y) (Atom x))]) \)

and qswv: qs = u \@ v

using assms(2)

by (auto simp: conc_def)

from uu have uyx: u = [y,x] by(auto)

from qswv uyx have vqs: length v = length qs - 2 by auto

from qswv uyx have vqs2: length v + 1 = length qs - 1 by auto

254
have \texttt{firststep}: $\text{TS\_step\_d}$ ($[x, y], h$) $y = ((0, []), y \neq h)$

proof (cases index $h$ $y = \text{length } h$)

\begin{itemize}
  \item case \texttt{True}
    \begin{itemize}
      \item then show \texttt{thesis unfolding TS\_step\_d\_def by(simp)}
    \end{itemize}
  \item case \texttt{False}
    \begin{itemize}
      \item with \texttt{assms(3) obtain hs where a: h = [x,x]@hs by auto}
      \item then show \texttt{thesis by(simp add: oneTS\_steps)}
    \end{itemize}
\end{itemize}
qed

have \texttt{s2: config\_'} ($r\text{TS } h0$) ($[x,y], h$) $u = ([x, y], x \# y \# h)$
unfolding $r\text{TS\_def } wyx$ apply(simp add: )

unfolding $\text{Step\_def by(simp add: firststep step\_def oneTS\_steps}$

have \texttt{ta: T\_on\_'} ($r\text{TS } h0$) ($[x,y], h$) $u = 1$
unfolding $r\text{TS\_def } wyx$

apply(simp)

apply(simp add: firststep)

unfolding $\text{Step\_def}$

using \texttt{assms(1) by (simp add: firststep step\_def oneTS\_steps t\_p\_def)}

have \texttt{ttt: T\_on\_'} ($r\text{TS } h0$) ($[x,y], \text{rev } u \oplus h$) $v@r = \text{length } v + T\_on\_'$ ($r\text{TS } h0$)
($[x,y], ((\text{rev } v) \circ ((\text{rev } u) \circ h)))$ $r$

\begin{itemize}
  \item \texttt{and history: (\exists hs. (\text{rev } qs \oplus h) = [x, y] \oplus hs)}
  \item \texttt{and state: config\_'} ($r\text{TS } h0$) ($[x, y], x \# y \# h$) $v = ([x, y], \text{rev } v \oplus (\text{rev } u \oplus h))$
\end{itemize}

apply(rule $\text{TS\_yx'}$

apply(fact)

using \texttt{vv apply(simp)}

using \texttt{uxx by(simp)}

then have \texttt{tat: T\_on\_'} ($r\text{TS } h0$) ($[x,y], x \# y \# h$) $v@r = \text{length } v + T\_on\_'$ ($r\text{TS } h0$)
($[x,y], \text{rev } q\_s \oplus h$) $r$

\begin{itemize}
  \item \texttt{and history: (\exists hs. (\text{rev } q\_s \oplus h) = [x, y] \oplus hs)}
  \item \texttt{and state: config\_'} ($r\text{TS } h0$) ($[x, y], x \# y \# h$) $v = ([x, y], \text{rev } q\_s \oplus h)$
\end{itemize}

using \texttt{qsuv uxx}

by auto

have \texttt{config\_'} ($r\text{TS } h0$) ($[x, y], h$) $q\_s = \text{config\_'} ($r\text{TS } h0$) ($\text{config\_'} ($r\text{TS } h0$)
($[x, y], h$) $u$ $v$

unfolding \texttt{qsuv by (rule config\_append2)}

also

have \ldots = ($[x, y], \text{rev } q\_s \oplus h$) by(simp add: \texttt{s2 state})

finally

255
have his: config' (rTS h0) ([x, y], h) qs = ([x, y], rev qs @ h).

have T_on' (rTS h0) ([x,y],h) (qs@r) = T_on' (rTS h0) ([x,y],h) (u @ v @ r) using qsuv by auto
also have ...
    = T_on' (rTS h0) ([x,y],h) u + T_on' (rTS h0) (config' (rTS h0) ([x,y],h) u) (v @ r)
    by (rule T_on'.append)
also have ... = T_on' (rTS h0) ([x,y],h) u + T_on' (rTS h0) ([x, y], x # y # h) (v @ r) by (simp only: s2)
also have ... = T_on' (rTS h0) ([x,y],h) u + length v + T_on' (rTS h0) ([x,y],rev qs @ h) r
by (simp only: tat)
also have ... = 1 + length v + T_on' (rTS h0) ([x,y],rev qs @ h) r
using vqs2 by auto
finally show ?thesis
    apply (safe)
    using history apply (simp)
    using his by auto
qed

lemma TS_xr': assumes x ≠ y qs ∈ lang (Plus (Atom x) One)
    h = [] ∨ (∃hs. h = [x, x] @ hs)
shows T_on' (rTS h0) ([x,y],h) (qs@r) = T_on' (rTS h0) ([x,y],rev qs@h)
    r
    ((∃hs. (rev qs @ h) = [x, x] @ hs) ∨ (rev qs @ h) = [x] ∨ (rev qs @ h) = [])
config' (rTS h0) ([x,y],h) (qs@r) = config' (rTS h0) ([x,y],rev qs @ h) r
by (auto simp add: T_on'.append Step_def rTS_def TS_step_d_def step_def t_p_def)

15.3.5 (x+1)yx(yx)*yy

lemma ts_b': assumes x ≠ y
    v ∈ lang (seq[Times (Atom y) (Atom x), Star (Times (Atom y) (Atom x)), Atom y, Atom y])
    (∃hs. h = [x, x] @ hs) ∨ h = [x] ∨ h = []
shows $T_{on'}(rTS h\theta) ([x, y], h) \; v = (\text{length} \; v - 2)$
\[ \land (\exists \; h$. rev \; v \; @ \; h) = [y, y]@hs) \land \text{config'}(rTS h\theta) ([x, y], h) \; v = ([y, x], \text{rev} \; v \; @ \; h)$

proof –
from assms have lenvmod: length $v$ mod 2 = 0 apply(simp)

proof –
assume $v \in ([y]) \text{ @ } [x]) \text{ @ } star ([y]) \text{ @ } [x]) \text{ @ } ([y]) \text{ @ } ([y]) \text{ @ } ([y]) \text{ @ } ([y])$
then obtain $p \; q \; r$ where $pqr: v = p@q@r$ and $p \in ([y]) \text{ @ } [x])$
and $q: q \in star ([y]) \text{ @ } [x])$ and $r \in ([y]) \text{ @ } ([y])$ by

$(\text{metis concE})$

then have $p = [y, x] \; r = [y, y]$ by auto
with $pqr$ have $a$: length $v = 4 + \text{length} \; q$ by auto

from $q$ have $b$: length $q$ mod 2 = 0
apply(induct $q$ rule: star_induct) by (auto)
from $a \; b$ show $\text{?thesis}$ by auto

qed

with assms(1,3) have fall: $(\exists \; hs. \; h = [x, x] \text{ @ } hs) \lor \text{ index} \; h \; y = \text{length} \; h$

by (auto)

from assms(2) have $v \in \text{lang} \; (\text{seq}[\text{Times} \; (\text{Atom} \; y) \; (\text{Atom} \; x)])$

then obtain $a \; b$ where $aa$: $a \in \text{lang} \; (\text{seq}[\text{Times} \; (\text{Atom} \; y) \; (\text{Atom} \; x)])$

and $b \in \text{lang} \; (\text{seq}[\text{Atom} \; y, \; \text{Atom} \; y])$

and $vab$: $v = a @ b$

by (erule concE)
then have $bb$: $b = [y, y]$ by auto

from $aa$ have lena: length $a > 0$ by auto

from $TS_{xyz}''OF$ assms(1) $aa$ fall] have stars: $T_{on'}(rTS h\theta) ([x, y], h)$ $(a @ b) =$

length $a - 1 + T_{on'}(rTS h\theta) ([x, y], \text{rev} \; a \; @ \; h) \; b$
and history: $(\exists \; hs. \; \text{rev} \; a \; @ \; h = [x, y] \text{ @ } hs)$
and state: config’(rTS h\theta) ([x, y], h) $a = ([x, y], \text{rev} \; a \; @ \; h)$ by auto

have suffix: $T_{on'}(rTS h\theta) ([x, y], \text{rev} \; a \; @ \; h) \; b = 1$
and $jajajaj$: config’(rTS h\theta) ([x, y], rev a @ h) $b = ([y, x], \text{rev} \; b \; @ \; rev a @ h)$ unfolding $bb$
using $TS_{yy'}$ history assms(1) by auto

from stars suffix have $T_{on'}(rTS h0) ([x, y], h) (a @ b) = length a$
using lena by auto
then have whatineed: $T_{on'}(rTS h0) ([x, y], h) v = (length v - 2)$
using vab bb by auto

have grgr: $config'(rTS h0) ([x, y], h) v = ([y, x], rev v @ h)$
  unfolding vab
  apply(simp only: config'_append2 state jajajaj) by simp
from history obtain hs' where rev a @ h = [x, y] @ hs' by auto
then obtain hs2 where reva: rev a @ h = x # hs2 by auto

show ?thesis using whatineed grgr
  by(auto simp add: reva vab bb)
qed

lemma $TS_{b1}'$: assumes $x \neq y \land h = [] \lor (\exists hs. h = [x, x] @ hs)$
  $qs \in lang (seq [Atom y, Atom x, Star (Times (Atom y) (Atom x)), Atom y, Atom y])$
shows $T_{on'}(rTS h0) ([x, y], h) qs = (length qs - 2)$
  $\land TS_{inv'}(config'(rTS h0)) ([x, y], h) qs = \langle \text{last qs} \rangle [x, y]$
proof --
have f: $qs \in lang (seq [Times (Atom y) (Atom x), Star (Times (Atom y) (Atom x)), Atom y, Atom y])$
  using assms(3) by(simp add: conc_assoc)
from $ts_{b1}'[OF assms(1) f] \ \text{assms(2)}$ have
  $T_{star}: T_{on'} (rTS h0) ([x, y], h) qs = length qs - 2$
  \text{and inv1: config'(rTS h0)} ([x, y], h) qs = ([y, x], rev qs @ h)
  \text{and inv2: (}\exists hs. rev qs @ h = [y, y] @ hs) \ \text{by auto}$

from $T_{star}$ have TS: $T_{on'} (rTS h0) ([x, y], h) qs = (length qs - 2)$
  by metis

have lqs: last qs = y using assms(3) by force

from inv1 have inv: $TS_{inv'}(config'(rTS h0)) ([x, y], h) qs = (\text{last qs}) [x, y]$
  apply(simp add: lqs)
  apply(subst $TS_{inv'}\_\text{det}$)
using assms(2) inv2 by(simp)

show ?thesis unfolding TS
apply (safe)
by (fact inv)
qed

lemma TS_b1'': assumes
  \( x \neq y \{ x, y \} = \{ x0, y0 \} \) TS_inv s x [x0, y0]
  set qs \subseteq \{ x, y \}
  qs \in \text{lang} (seq [Atom y, Atom x, Star (Times (Atom y) (Atom x)), Atom y, Atom y])
shows TS_inv (config'_rand (embed (rTS h0))) s qs (last qs) [x0, y0]
  \( \land T_{on\_rand'} (embed (rTS h0)) \) s qs = (length qs - 2)
proof —
  from assms(1,2) have kas: \((x0=x \land y0=y) \lor (y0=x \land x0=y)\) by (auto)
  then obtain h where S: \( s = \text{return\_pmf} ([x,y],h) \) and h: h = [] \lor (\exists hs. h = [x, x] @ hs)
    apply (rule disjE) using assms(1,3) unfolding TS_inv_def by (auto)

  have l: qs \neq [] using assms by auto
  { fix x y qs h0
    fix h::nat list
    assume A: x \neq y
    and B: qs \in \text{lang} (seq[Times (Atom y) (Atom x), Star (Times (Atom y) (Atom x)), Atom y, Atom y])
    and C: h = [] \lor (\exists hs. h = [x, x] @ hs)
  }
  then have C': (\exists hs. h = [x, x] @ hs) \lor h = [x] \lor h = [] by blast
  from B have lqs: last qs = y using assms(5) by (auto simp add: conc_def)

  have TS_inv (config'_rand (embed (rTS h0))) (return\_pmf ([x, y], h)) qs (last qs) [x, y] \land
    T_{on\_rand'} (embed (rTS h0)) (return\_pmf ([x, y], h)) qs = length qs - 2
    apply (simp only: T_{on\_embed}[symmetric] config'_embed)
    using ts_b'\[OF A B C'] A lqs unfolding TS_inv'_det by auto
} note b1=this
show \( ? \text{thesis unfolding } S \)
using \( \text{kas apply(rule disjE)} \)
apply(simp only:)
apply(rule disjE)
  using assms apply(simp)
  using assms apply(simp add: conc_assoc)
  using h apply(simp)
apply(simp only:)
apply(subst TS_inv_sym[of y x x y])
using assms(1) apply(simp)
apply(blast)
defer
apply(rule disjE)
  using assms apply(simp)
  using assms apply(simp add: conc_assoc)
  using h apply(simp)
using last_in_set l assms(4) by blast

qed

lemma ts_b2': assumes \( x \neq y \)
  \( qs \in \text{lang (seq[Atom x, Times (Atom y) (Atom x), Star (Times (Atom y) (Atom x)), Atom y, Atom y])} \)
  \( (\exists hs. h = [x, x] @ hs) \lor h = [] \)
shows \( T \text{on}'(rTS h0) ([x, y], h) \) \( qs = (\text{length} qs - 3) \)
  \( \land \text{config}'(rTS h0) ([x,y], h) \) \( qs = ([y,x], \text{rev v @ h}) \land (\exists hs. (\text{rev q @ h}) = [y,y] @ hs) \)
proof
  from assms(2) obtain \( v \) where \( qs: qs = [x] @ v \)
    and \( V: v \in \text{lang (seq[Times (Atom y) (Atom x), Star (Times (Atom y) (Atom x)), Atom y, Atom y])} \)
    by(auto simp add: conc_assoc)
  from assms(3) have \( 3: (\exists hs. x\#h = [x, x] @ hs) \lor x\#h = [x] \lor x\#h = [] \) by auto
  from ts_b[OF assms(1) V 3]
  have \( T: T \text{on}'(rTS h0) ([x, y], x\#h) v = \text{length} v - 2 \)
    and \( C: \text{config}'(rTS h0) ([x, y], x\#h) v = ([y, x], \text{rev v @ x\#h}) \)
    and \( H: (\exists hs. \text{rev v @ x\#h} = [y, y] @ hs) \) by auto
  have \( t: t_p [x, y] x (\text{fst (snd (rTS h0) ([x, y], h) x))) = 0 \)

260
by (simp add: step_def rTS_def TS_step_d_def t_p_def)

have c: Partial_Cost_Model.Step (rTS h0) (\{x, y\}, h) x
  = ([x,y], x ≠ h) by (simp add: Step_def rTS_def TS_step_d_def
  step_def)

show ?thesis
  unfolding qs apply(safe)
  apply(simp add: T_on'_append T c t)
  apply(simp add: config'_rand_append C c)
  using H by simp

qed

lemma TS_b2''; assumes
  x ≠ y \{x, y\} = \{x0, y0\} TS_inv s x [x0, y0]
  set qs ⊆ \{x, y\}
  qs ∈ lang (seq [Atom x, Atom y, Atom x, Star (Times (Atom y) (Atom x)), Atom y, Atom y])
  shows TS_inv (config'_rand (embed (rTS h0))) s qs) (last qs) [x0, y0]
    ∧ T_on_rand' (embed (rTS h0)) s qs = (length qs - 3)

proof
  from assms(1,2) have kas: (x0=x ∧ y0=y) ∨ (y0=x ∧ x0=y) by(auto)
  then obtain h where S: s = return_pmf ([x,y],h) and h: h = [] ∨ (∃hs. h = [x, x] @ hs)
    apply(rule disjE) using assms(1,3) unfolding TS_inv_def by(auto)

  have l: qs ≠ [] using assms by auto
  |
  fix x y qs h0
  fix hs:: nat list
  assume A: x ≠ y
    and B: qs ∈ lang (seq[Atom x, Times (Atom y) (Atom x), Star (Times (Atom y) (Atom x)), Atom y, Atom y])
    and C: h = [] ∨ (∃hs. h = [x, x] @ hs)

  from B have lqs: last qs = y using assms(5) by(auto simp add: conc_def)

  from C have C': (∃hs. h = [x, x] @ hs) ∨ h = [] by blast

  have TS_inv (config'_rand (embed (rTS h0))) (return_pmf ([x, y], h))
    qs) (last qs) [x, y] ∧
    T_on_rand' (embed (rTS h0)) (return_pmf ([x, y], h)) qs = length
    qs - 3
apply(simp only: T_on'_embed[symmetric] config'_embed)
using ts_b2 "[OF A B C] A lqs unfolding TS_inv'_det by auto
}

show ?thesis unfolding S
using kas apply(rule disjE)
  apply(simp only:)
  apply(rule b2)
  using assms apply(simp)
  using assms apply(simp add: conc_assoc)
  using h apply(simp)
apply(simp only:)

apply(subst TS_inv_sym[of y x y])
using assms(1) apply(simp)
apply(blast)
defer
apply(rule b2)
  using assms apply(simp)
  using assms apply(simp add: conc_assoc)
  using h apply(simp)
using last_in_set l assms(4) by blast

qed

lemma TS_b': assumes x ≠ y ⚬ h = [] ∨ (∃hs. h = [x, x] ⚬ hs)
  qs ∈ lang(seq[Plus(Atom x) One, Atom y, Atom x, Star(Times
  Atom y)(Atom x)), Atom y, Atom y])
shows T_on'(rTS h0) ([x, y], h) qs ≤ 2 * T_pre (rTS h0) ([x, y], h) qs (OPT2 qs [x, y]) ∧ TS_inv' (config' (rTS h0) ([x, y], h) qs) (last qs) [x,y]

proof –
obtain u v where uu: u ∈ lang(Plus(Atom x) One)
  and vv: v ∈ lang(seq[Times(Atom y)(Atom x), Star(Times(Atom
  y)(Atom x)), Atom y, Atom y])
  and qsuv: qs = u ⚬ v
using assms(3)
by (auto simp: conc_def)

from TS_xr'[OF assms(1) uu assms(2)] have
  T_pre: T_on'(rTS h0) ([x, y], h) (u ⚬ v) =
        T_on'(rTS h0) ([x, y], rev u ⚬ h) v
\[
\begin{align*}
\text{and } \text{fall}' : (\exists h s. \text{rev } u @ h) = [x, x] @ h s) \lor (\text{rev } u @ h) = [x] \lor (\text{rev } u @ h) = [] \\
\text{and } \text{conf} : \text{config}' (rTS h0) ([x, y], h) (u@v) = \text{config}' (rTS h0) ([x, y], \text{rev } u @ h) v \\
\text{by } \text{auto} \\
\end{align*}
\]

with \text{assms uu have fall: (}\exists h s. (\text{rev } u @ h) = [x, x] @ h s) \lor \text{index (rev } u @ h) y = \text{length (rev } u @ h) \\
\text{by } \text{auto} \\

from \text{ts}_h[\text{OF } \text{assms(1) vv fall'}] \text{ have} \\
\text{T_star: } T_{\text{on'}} (rTS h0) ([x, y], \text{rev } u @ h) v = \text{length } v - 2 \\
\text{and } \text{inv1: } \text{config}' (rTS h0) ([x, y], \text{rev } u @ h) v = ([y, x], \text{rev } v @ \text{rev } u @ h) \\
\text{and } \text{inv2: } (\exists h s. \text{rev } v @ \text{rev } u @ h) = [y, y] @ h s) \text{ by } \text{auto} \\

from \text{T_pre } \text{T_star } \text{qsuv have } \text{TS: } T_{\text{on'}} (rTS h0) ([x, y], h) \text{ qs} = (\text{length } v - 2) \text{ by } \text{metis} \\

from \text{uu have } \text{uuu: } u = [] \lor u = [x] \text{ by } \text{auto} \\
from \text{vv have } \text{vvv: } v \in \text{lang (seq} \\
[\text{Atom y, Atom x, Star (Times (Atom y) (Atom x)), Atom y, Atom y}] \text{ by } (\text{auto simp: conc_def}) \\
\text{have } \text{OPT: } T_p [x, y] \text{ qs } (\text{OPT2 qs } [x, y]) = (\text{length } v) \text{ div } 2 \text{ apply } (\text{rule OPT2_B}) \text{ by } (\text{fact})+ \\
\text{have } \text{lqs: } \text{last qs} = y \text{ using } \text{assms(3) by } \text{force} \\
\text{have } \text{config}' (rTS h0) ([x, y], h) \text{ qs} = ([y, x], \text{rev } qs @ h) \\
\text{unfolding } \text{qsuv conf inv1 by } \text{simp} \\
\text{then have } \text{inv: } T_{\text{S_inv'}} (\text{config}' (rTS h0) ([x, y], h) \text{ qs}) \text{ (last qs } [x, y] \\
\text{apply } (\text{simp add: lqs}) \\
\text{apply } (\text{subst } T_{\text{S_inv'}} \text{'det}) \\
\text{using } \text{assms(2) inv2 qsuv by } (\text{simp}) \\
\text{show } \text{thesis unfolding } \text{TS OPT} \\
\text{apply } (\text{safe}) \\
\text{apply } (\text{simp}) \\
\text{by } (\text{fact inv}) \\
\text{qed}
15.3.6 \((x+1)yy\)

**Lemma ts.a': assumes** \(x \neq y\) \(qs \in \text{lang}\) \(\text{seq} [\text{Plus} (\text{Atom} x) \text{One}, \text{Atom} y, \text{Atom} y]\)

\[ h = \emptyset \lor (\exists hs. h = [x, x] @ hs) \]

**shows** \(\text{TS}^\text{inv'} (\text{config'} (rTS h0) ([x, y], h) \text{qs}) (\text{last} \text{qs}) [x, y] \]

\[ \land \text{T}_{\text{on'}} (rTS h0) ([x, y], h) \text{qs} = 2 \]

**Proof**

- **Obtain** \(u v\) where \(u: u \in \text{lang}\) \(\text{Plus} (\text{Atom} x) \text{One}\)
  - and \(v: v \in \text{lang}\) \(\text{seq}[\text{Atom} y, \text{Atom} y]\)
  - and \(qsuv: qs = u @ v\)
  - using \(\text{assms}(2)\)
    - by (auto simp: conc_def)

- **From** \(vv\) have \(vv2: v = [y, y]\) by auto

- From \(uu\) have \(TS_{\text{prefix}}: T_{\text{on'}} (rTS h0) ([x, y], h) u = 0\)
  - using \(\text{assms}(1)\) by (auto simp add: rTS_def oneTS_steps t_p_def)

- Have \(h_{\text{split}}: \text{rev} u @ h = \emptyset \lor \text{rev} u @ h = [x] \lor (\exists hs. \text{rev} u @ h = [x, x] @ hs)\)
  - using \(\text{assms}(3)\) \(uu\) by (auto)

- Then have \(e: T_{\text{on'}} (rTS h0) ([x, y], \text{rev} u @ h) [y, y] = 2\)
  - using \(\text{assms}(1)\)
    - apply (auto simp add: rTS_def oneTS_steps Step_def step_def t_p_def) done

- Have \(conf: \text{config'} (rTS h0) ([x, y], h) u = ([x, y], \text{rev} u @ h)\)
  - using \(uu\) by (auto simp add: Step_def rTS_def TS_step_d_def step_def)

- Have \(T_{\text{on'}} (rTS h0) ([x, y], h) \text{qs} = T_{\text{on'}} (rTS h0) ([x, y], h) (u @ v)\)
  - using \(qsuv\) by auto

  Also have . . .
  - \(= T_{\text{on'}} (rTS h0) ([x, y], h) u + T_{\text{on'}} (rTS h0) (\text{config'} (rTS h0) ([x, y], h) u @ v)\)
    - by(rule T_{on'}append)

  Also have . . .
  - \(= T_{\text{on'}} (rTS h0) ([x, y], h) u + T_{\text{on'}} (rTS h0) ([x, y], \text{rev} u @ h) [y, y]\)
    - by(simp add: conf vv2)

  Also have . . . \(= T_{\text{on'}} (rTS h0) ([x, y], h) u + 2\) by (simp only: e)

  Also have . . . \(= 2\) by (simp add: TS_prefix)
finally have \( TS: T_{on'} (rTS h0) ([x, y], h) qs = 2 \).

have \( lqs: \text{last } qs = y \) using assms(2) by force

from assms(1) have \( \text{config'} (rTS h0) ([x, y], h) qs = ([y, x], \text{rev } qs @ h) \)
  unfolding qsuv
  apply (simp only: config'_append2 conf vv2)
  using h_split
  apply (auto simp add: Step_def rTS_def oneTS_steps
    step_def)
  by (simp_all add: mtf2_def swap_def)

with assms(1) have \( \text{TS_inv'} (\text{config'} (rTS h0) ([x, y], h) qs) \) (last qs) \([x,y]\)
  apply (subst TS_inv'_det)
  by (simp add: qsuv vv2 lqs)

show \(?thesis unfolding TS apply(auto) by fact qed

lemma \( \text{TS}_{a'}\): assumes \( x \neq y \)
  \( h = [] \cup (\exists hs. h = [x, x] \# hs) \)
  and \( qs \in \text{lang } (\text{seq } [\text{Plus } (\text{Atom } x) \text{ One }, \text{Atom } y, \text{Atom } y]) \)
  shows \( T_{on'} (rTS h0) ([x, y], h) qs \leq 2 * T_p [x, y] qs (OPT2 qs [x, y]) \)
  \( \land \text{TS}_\text{inv'} (\text{config'} (rTS h0) ([x, y], h) qs) \) (last qs) \([x,y]\)
  \( \land \text{T}_{on'} (rTS h0) ([x, y], h) qs = 2 \)
  proof –
  have \( \text{OPT}: T_p [x,y] qs (OPT2 qs [x,y]) = 1 \) using \( \text{OPT2_A[OF assms(1,3)]} \)
  by auto
  show \(?thesis using OPT ts_{a'}[OF assms(1,3,2)] by auto qed

lemma \( \text{TS}_{a''}\): assumes \( x \neq y \) \( \{x, y\} = \{x0, y0\} \)
  \( \text{TS}_\text{inv } s x [x0, y0] \)
  set \( qs \subseteq \{x, y\} \) \( qs \in \text{lang } (\text{seq } [\text{Plus } (\text{Atom } x) \text{ One }, \text{Atom } y, \text{Atom } y]) \)
  shows \( \text{TS}_\text{inv } (\text{config'}_{\text{rand}} (\text{embed } (rTS h0))) s qs) \) (last qs) \([x0, y0]\)
  \( \land \text{T}_{p-on'_\text{rand'}} (\text{embed } (rTS h0))) s qs = 2 \)
  proof –
  from assms(1,2) have \( \text{kas}: (x0=x \land y0=y) \lor (y0=x \land x0=y) \) by(auto)
  then obtain \( h \) where \( S: s = \text{return_pmf } ([x,y], h) \) and \( h: h = [] \cup (\exists hs. \)
  265
\[ h = [x, x] @ hs \]

apply (rule disjE) using assms \((1, 3)\) unfolding \(TS_{\text{inv, def}}\) by (auto)

have \(l: qs \neq []\) using assms by auto

\{
  fix \(x y qs h0\)
  fix \(h::\text{nat list}\)
  assume \(A: x \neq y\)
  \(qs \in \text{lang (seq [\text{question (Atom }x\), \text{Atom }y, \text{Atom }y])}\)
  \(h = [] \lor (\exists hs. h = [x, x] @ hs)\)

  have \(TS_{\text{inv}} (\text{config'}_{\text{rand}} (\text{embed (rTS }h0))) (\text{return}_{\text{pmf}} ([x, y], h))\)
  \(qs) (\text{last } qs) [x, y] \land \quad \text{T}_{\text{on}_{\text{rand'}}} (\text{embed (rTS }h0)) (\text{return}_{\text{pmf}} ([x, y], h))\)
  \(qs = 2\)
  apply (simp only: \(T_{\text{on'}_{\text{embed[symmetric] config'}_{\text{embed}}}}\))
  using \(ts_a''[\text{OF } A]\) by auto
\}

note \(b = \text{this}\)

show \(?\text{thesis unfolding } S\)
using \(kas\) apply (rule disjE)
apply (simp only:)
apply (rule b)
  using assms apply (simp)
  using assms apply (simp)
  using \(h\) apply (simp)
apply (simp only:)

apply (subst \(TS_{\text{inv}} \_\text{sym[of y x x y]}\))
using assms \((1)\) apply (simp)
apply (blast)
defer
apply (rule b)
  using assms apply (simp)
  using assms apply (simp)
  using \(h\) apply (simp)
using \(\text{last_in_set l assms(4)}\) by blast

qed

15.3.7 \(x+y)(yx)^x\)

lemma \(ts\_c':\) assumes \(x \neq y\)
  \(v \in \text{lang (seq[Times (Atom y) (Atom x), Star (Times (Atom y) (Atom x))]}\)


\(x\)), \(\text{Atom } x)\)
\(\exists h. \ h = [x, x] \ @ h) \lor h = [x] \lor h = []\)
shows \(T_\text{on'} (rTS h0) ([x, y], h) \ v = (\text{length } v - 2)\)
\& \ \text{config'} (rTS h0) ([x, y], h) \ v = ([x, y], \text{rev } \ v \ @ h) \land (\exists h. \ (\text{rev } v \ @ h) = [x, x] \ @ h)\)

proof –
from assms have lenvmod: \(\text{length } v \mod 2 = 1\) apply(simp)
proof –
assume \(v \in (\{[y]\} \ @ ([x]) @ \text{ star} ([\{[y]\} @ ([x])}) @ ([x])\)
then obtain \(p q r\) where \(\text{pqr: } v = p @ q @ r\) and \(p \in ([\{y]\} @ ([x])\)
and \(q: q \in \text{ star } ([\{y]\} @ ([x])\) and \(r \in ([x])\) by (metis concE)
then have \(p = [y, x]\) \(r = [x]\) by auto
with \(\text{pqr}\) have \(a: \text{ length } v = 3 + \text{ length } q\) by auto

from \(q\) have \(b: \text{ length } q \mod 2 = 0\)
apply(induct \(q\) rule: star_induct) by (auto)
from \(a b\) show \(\text{ length } v \mod 2 = \text{ Suc } 0\) by auto
qed

with assms(1, 3) have fall: \(\exists h. \ h = [x, x] @ hs\) \(\lor \ \text{ index } h y = \text{ length } h\)
by(auto)

from assms(2) have \(v \in \text{ lang } (\text{ seq } [\text{ Times } (\text{ Atom } y) (\text{ Atom } x)]))\)
\(\at \text{ seq } (\text{ Atom } x)\) by (auto simp: conc_def)
then obtain \(a b\) where \(\text{aa: } a \in \text{ lang } (\text{ seq } [\text{ Times } (\text{ Atom } y) (\text{ Atom } x)]\)
\(\text{ Star}(\text{ Times } (\text{ Atom } y) (\text{ Atom } x)]))\)
and \(b \in \text{ lang } (\text{ seq } [\text{ Atom } x])\)
and \(\text{vab: } v = a \ @ b\)
by(erule concE)
then have \(\text{ bb: } b = [x]\) by auto
from \(\text{aa}\) have lena: \(\text{ length } a > 0\) by auto

from \(\text{TS}_\text{xyxyz'} [\text{OF assms(1)} \ \text{ aa \ fall}]\) have stars: \(T_\text{on'} (rTS h0) ([x, y], h) \ (a \ @ b) = \)
\(\text{length } a - 1 + T_\text{on'} (rTS h0) ([x, y], \text{rev } a \ @ h) b\)
and \(\text{ history: } (\exists h. \ \text{rev } a \ @ h = [x, y] @ hs)\)
and \(\text{ state: config'} (rTS h0) ([x, y], h) a = ([x, y], \text{rev } a \ @ h)\) by auto

267
have suffix: \( T_{on}'(rTS h0)([x, y], \text{rev } a \otimes h) b = 0 \)
  and suState: \( \text{config}'(rTS h0)([x, y], \text{rev } a \otimes h) b = ([x, y], \text{rev } b \otimes \text{rev } a \otimes h) \)
  unfolding \( bb \) using \( TS_x' \) by auto

from stars suffix have \( T_{on}'(rTS h0)([x, y], h) (a \otimes b) = \text{length } a - 1 \) by auto
  then have whatineed: \( T_{on}'(rTS h0)([x, y], h) v = (\text{length } v - 2) \)
  using \( vab \) by auto

have conf: \( \text{config}'(rTS h0)([x, y], h) v = ([x, y], \text{rev } v \otimes h) \)
  by (simp add: vab conf' append2 state suState)

from history obtain \( hs' \) where \( \text{rev } a \otimes h = [x, y] \otimes hs' \) by auto
  then obtain \( hs2 \) where \( \text{rev } a: \text{rev } a \otimes h = x \# hs2 \) by auto

show \( \text{thesis} \) using whatineed
  apply (auto)
  using conf apply (simp)
  by (simp add: reva vab bb)
qed

lemma \( TS_{c1''} \); assumes
  \( x \neq y \{x, y\} = \{x0, y0\} \) \( TS_{inv} s x [x0, y0] \)
  set \( qs \subseteq \{x, y\} \)
  \( qs \in \text{lang } (\text{seq } [\text{Atom } y, \text{Atom } x, \text{Star } (\text{Times } (\text{Atom } y) (\text{Atom } x)), \text{Atom } x]) \)
  shows \( TS_{inv} (\text{config'}\otimes\text{rand} (\text{embed } (rTS h0))) s qs) (\text{last } qs) [x0, y0] \)
  \( \land \ T_{on\_rand'} (\text{embed } (rTS h0))) s qs = (\text{length } qs - 2) \)
proof -
  from assms(1,2) have kas: (x0=x \land y0=y) \lor (y0=x \land x0=y) by (auto)
  then obtain \( h \) where \( S: s = \text{return}_{pmf} ([x, y], h) \) and \( h: h = [] \lor (\exists hs. h = [x, x] \otimes hs) \)
  apply (rule disjE) using assms(1,3) unfolding \( TS_{inv}_{def} \) by (auto)

have l: \( qs \neq [] \) using assms by auto
{
  fix \( x \ y \ qs \ h0 \)
  fix \( h:: \text{nat list} \)
  assume A: \( x \neq y \)
  and B: \( qs \in \text{lang } (\text{seq } [\text{Times } (\text{Atom } y) (\text{Atom } x)], \text{Star } (\text{Times } (\text{Atom }
y) \ (Atom x), Atom x)\\and C: h = [] \lor (\exists hs. h = [x, x] \ Assoc hs)\\then have C': (\exists hs. h = [x, x] \ Assoc hs) \lor h = [x] \lor h = [] \ by blast\\from B have lqs: last qs = x using assms(5) by (auto simp add: conc_def)\\have TS_inv (config'_rand (embed (rTS h0)) (return_pmf ([x, y], h)) qs) (last qs) [x, y] \\&\\T_on_rand' (embed (rTS h0)) (return_pmf ([x, y], h)) \ qs = length qs – 2\\apply (simp only: T_on_rand[embed[symmetric] config'_embed] using ts_c'[OF A B C'] A lqs unfolding TS_inv'det by auto)\\note c1=\this
from assms(2) obtain v where qs: qs = [x]@v
    and V: v∈lang (seq[Times (Atom y) (Atom x), Star (Times (Atom y) (Atom x))], Atom x))
    by(auto simp add: conc_assoc)

from assms(3) have 3: (∃hs. x#h = [x, x] @ hs) ∨ x#h = [x] ∨ x#h = [] by auto

from ts_c["OF assms(1) V 3"]
    have T: T_on' (rTS h0) ([x, y], x#h) v = length v − 2
    and C: config' (rTS h0) ([x, y], x#h) v = ([x, y], rev v @ x#h)
    and H: (∃hs. rev v @ x#h = [x, x] @ hs) by auto

have t: t_p [x, y] x (fst (snd (rTS h0) ([x, y], h) x)) = 0
    by (simp add: step_def rTS_def TS_step_d_def t_p_def)

have c: Partial_Cost_Model.Step (rTS h0) ([x, y], h) x
    = ([x,y], x#h) by (simp add: Step_def rTS_def TS_step_d_def step_def)

show ?thesis
    unfolding qs
    apply(auto simp add: T_on'append T c t)
    apply(auto simp add: config'_rand_append C c)
    using H by simp

qed

lemma TS_c2'': assumes
    x ≠ y {x, y} = {x0, y0} TS_inv s x [x0, y0]
    set qs ⊆ {x, y}
    qs ∈ lang (seq [Atom x, Atom y, Atom x, Star (Times (Atom y) (Atom x))], Atom x))
    shows TS_inv (config'_rand (embed (rTS h0))) s qs) (last qs) [x0, y0]
    ∧ T_on_rand' (embed (rTS h0)) s qs = (length qs − 3)

proof −
    from assms(1,2) have kas: (x0=x ∧ y0=y) ∨ (y0=x ∧ x0=y) by(auto)
    then obtain h where S: s = return_pmf ([x,y],h) and h: h = [] ∨ (∃hs.
        h = [x, x] @ hs)
        apply(rule disjE) using assms(1,3) unfolding TS_inv_def by(auto)

    have l: qs ≠ [] using assms by auto
    { fix x y qs h0
      fix h:: nat list
assume $A$: $x \neq y$

and $B$: $qs \in \text{lang}(\text{seq}[\text{Atom } x, \text{Times} (\text{Atom } y) (\text{Atom } x), \text{Star} (\text{Times} (\text{Atom } y) (\text{Atom } x)), \text{Atom } x])$

and $C$: $h = [] \lor (\exists hs. h = [x, x] @ hs)$

from $B$ have $lqs$: last $qs = x$ using $\text{assms}(5)$ by (auto simp add: conc_def)

from $C$ have $C'$: $(\exists hs. h = [x, x] @ hs) \lor h = []$ by blast

have $T_S_{\text{inv}} (\text{config'}_{\text{rand}} (\text{embed} (rTS h0))) \ (\mathit{return}_{\mathit{pmf}} ([x, y], h))$ $qs) \ (\text{last } qs) [x, y] \land$

$T_{\text{on}_{\text{rand'}}} (\text{embed} (rTS h0)) \ (\mathit{return}_{\mathit{pmf}} ([x, y], h))$ $qs = \mathit{length}$ $qs - 3$

apply(simp only: $T_{\text{on}_{\text{rand'}}} \mathit{embed}[\text{symmetric}] \text{config'}_{\text{embed}}$)

using $\text{ts}_{c2}'[\text{OF } A \ B \ C']$ $A$ $lqs$ unfolding $T_S_{\text{inv'}_{\text{det}}}$ by auto

\text{note} $c2$=this

show \text{thesis} unfolding $S$

using $\text{kas}$ apply(rule disjE)

apply(simp only:)

apply(rule $c2$)

using $\text{assms}$ apply(simp)

using $\text{assms}$ apply(simp add: conc_assoc)

using $h$ apply(simp)

apply(simp only:)

apply(subst $T_S_{\text{inv}} \mathit{sym}[\text{of } y \ x \ x \ y])$

using $\text{assms}(1)$ apply(simp)

apply(blast)

defer

apply(rule $c2$)

using $\text{assms}$ apply(simp)

using $\text{assms}$ apply(simp add: conc_assoc)

using $h$ apply(simp)

using $\text{last}_{\text{in}_{\text{set}}}$ $l$ $\text{assms}(4)$ by blast

qed

\textbf{lemma} $T_S_{c4'}$: assumes $x \neq y$ $h = [] \lor (\exists hs. h = [x, x] @ hs)$

$qs \in \text{lang} (\text{seq}[\text{Plus} (\text{Atom } x) \text{rexp}.\text{One}, \text{Atom } y, \text{Atom } x, \text{Star} (\text{Times} (\text{Atom } y) (\text{Atom } x)), \text{Atom } x])$

shows $T_{\text{on'}} (rTS h0) ([x, y], h)$ $qs$
\[ \leq 2 \times T_p [x, y] \text{qs } (\text{OPT2 qs } [x, y]) \land \text{TS_inv'} (\text{config'} (rTS h0) ([x, y], h) \text{qs}) (\text{last qs}) [x,y] \]

**proof**

obtain \( u \) \( v \) where \( uu : u \in \text{lang } (\text{Plus (Atom x) One}) \)

and vv: \( v : v \in \text{lang } (\text{seq[Times (Atom y) (Atom x), Star (Times (Atom y) (Atom x)) (Atom x)]}) \)

and qsuv: \( qs = u @ v \)

using assms(3)

by (auto simp: \text{conc_def})

from \( TS_{\text{xs'}}[OF \text{assms}(1) \text{ vv assms}(2)] \) have

\[ T_{\text{pre}}: T_{\text{on'}} (rTS h0) ([x, y], h) (u @ v) = T_{\text{on'}} (rTS h0) ([x, y], h) (u @ v) \]

and fall': \( (\exists \text{hs. } (\text{rev } u @ h) = [x, x] @ \text{hs}) \lor (\text{rev } u @ h) = [x] \lor (\text{rev } u @ h) = [] \)

and conf': \( \text{config'} (rTS h0) ([x, y], h) (u @ v) = \text{config'} (rTS h0) ([x, y], h) (u @ v) \)

by auto

with assms \( uu \) have fall: \( (\exists \text{hs. } (\text{rev } u @ h) = [x, x] @ \text{hs}) \lor \text{index } (\text{rev } u @ h) \)

by (auto)

from \( \text{ts_c'}[OF \text{assms}(1) \text{ vv fall'}] \) have

\[ T_{\text{star}}: T_{\text{on'}} (rTS h0) ([x, y], h) v = (\text{length } v - 2) \]

and inv1: \( \text{config'} (rTS h0) ([x, y], (\text{rev } u @ h)) v = ([x, y], \text{rev } v @ \text{rev } u @ h) \)

and inv2: \( (\exists \text{hs. } \text{rev } v @ \text{rev } u @ h = [x, x] @ \text{hs}) \) by auto

from \( T_{\text{pre}} T_{\text{star}} \) \( qsuv \) have \( TS: T_{\text{on'}} (rTS h0) ([x, y], h) qs = (\text{length } v - 2) \) by metis

from \( uu \) have \( uuu: u = [] \lor u = [x] \) by auto

from \( vv \) have rev: \( v \in \text{lang } (\text{seq} [\text{Atom y}, \text{Atom x}, \text{Star (Times (Atom y) (Atom x)), Atom x)]) \) by (auto simp: \text{conc_def})

have OPT: \( T_p [x,y] \text{qs } (\text{OPT2 qs } [x,y]) = (\text{length } v) \div 2 \) apply (rule OPT2,C) by (fact)+

have lqs: \( \text{last qs } = x \) using assms(3) by force

have conf: \( \text{config'} (rTS h0) ([x, y], h) qs = ([x, y], \text{rev } qs @ h) \)
by (simp add: qsv conf' inv1)
then have conff: \( TS_{inv'} (config' (rTS h0)) ([x, y], h) q) \) (last q) [x,y]
apply (simp add: q)
apply (subst TS_{inv'} det)
using inv2 qsv by (simp)

show \(?thesis unfoldig TS OPT
by (auto simp add: conf)
qed

15.3.8 xx

lemma request_first: \( x \neq y \implies \) Step (rTS h) ([x, y], is) x = ([x,y], x#is)
unfolding rTS_def Step_def by (simp add: split_def TS_step.d_def step_def)

lemma tsd': \( q \in Lxx x y \implies \)
x \( \neq y \implies \)
h = [] \lor (\exists hs. h = [x, x] @ hs) \implies 
q \in lang (seq [Atom x, Atom x]) \implies 
\( T_{on'} (rTS h0) ([x, y], h) q) = 0 \land 
TS_{inv'} (config' (rTS h0)) ([x, y], h) q) x [x,y]

proof –
assume xny: \( x \neq y \)
assume q \( \in lang (seq [Atom x, Atom x]) \)
then have xx: \( q = [x,x] \) by auto

from xny have TS: \( T_{on'} (rTS h0) ([x, y], h) q) = 0 \) unfolding xx
by (auto simp add: Step_def step_def oneTS_steps rTS_def t_p_def)

from xny have config' (rTS h0) ([x, y], h) q) = ([x, y], x # x # h)
by (auto simp add: xx Step_def rTS_def oneTS_steps step_def)

then have TS_{inv'} (config' (rTS h0)) ([x, y], h) q) x [x,y]
by (simp add: TS_{inv'} det)

with TS show \(?thesis by simp
qed

lemma TS_d': assumes xny: \( x \neq y \) and h = [] \lor (\exists hs. h = [x, x] @ hs)
and qsis: \( q \in lang (seq [Atom x, Atom x]) \)
shows \( T_{on'} (rTS h0) ([x],h) q) \leq 2 * T_p [x, y] q) (OPT2 q) [x, y])

273
and \( \text{TS}_{inv}'(\text{config}'(rTS\ h0)\ ([x,y], h)\ qs)\ (\text{last}\ qs)\ [x, y] \)
and \( T_{on}'(rTS\ h0)\ ([x,y], h)\ qs = 0 \)

proof –
from qsis have \( xx\): \( qs = [x,x] \) by auto

show \( \text{TS}: T_{on}'(rTS\ h0)\ ([x,y], h)\ qs = 0 \)
using assms(1) by (auto simp add: \( xx\) \( t_p\) \( \text{def}\) \( \text{Step}\) \( \text{def}\) \( \text{oneTS}\) \( \text{steps}\)

then show \( T_{on}'(rTS\ h0)\ ([x,y], h)\ qs \leq 2 \ast T_p\ [x, y]\ qs\ (\text{OPT2}\ qs\ [x, y]) \)
by simp

show \( \text{TS}_{inv}'(\text{config}'(rTS\ h0)\ ([x,y], h)\ qs)\ (\text{last}\ qs)\ [x, y] \)

unfolding \( \text{TS}_{inv}\) \( \text{def}\)
by(simp add: \( xx\) request \( \text{first}[OF}\ \text{xy}]\)

qed

lemma \( \text{TS}_{d''}\): assumes
\( x \neq y\ \{x, y\} = \{x0, y0\}\ \text{TS}_{inv}\ \{x0, y0\} \)
set \( qs \subseteq \{x, y\} \)
\( qs \in\ \text{lang}\ \{\text{seq}\ \{\text{Atom}\ x, \text{Atom}\ x\}\} \)
shows \( \text{TS}_{inv}'(\text{config}'\ \text{rand}\ (\text{embed}(rTS\ h0))\ s\ qs)\ (\text{last}\ qs)\ [x0, y0] \)
\( \land\ T_{on\ rand}'(\text{embed}(rTS\ h0))\ s\ qs = 0 \)

proof –
from assms(1,2) have \( \text{kas}:(x0=x\ \land\ y0=y)\ \lor\ (y0=x\ \land\ x0=y) \) by(auto)
then obtain \( h\) where \( S: s = \text{return}_{pmf}(\{x,y\}, h)\) and \( h: h = []\ \lor\ (\exists\ hs.\ h = [x, x] @ hs) \)

apply(rule disjE) using assms(1,3) unfolding \( \text{TS}_{inv}\) \( \text{def}\) by(auto)

have \( l: qs \neq []\) using assms by auto
{
fix \( x\ y\ qs\ h0\)
fix \( h::\ \text{nat}\ \text{list} \)
assume \( A:\ x \neq y \)
and \( B: qs \in:\ \text{lang}\ \{\text{seq}\ \{\text{Atom}\ x, \text{Atom}\ x\}\} \)
and \( C: h = []\ \lor\ (\exists\ hs.\ h = [x, x] @ hs) \)

from \( B\) have \( lqs: \text{last}\ qs = x\) using assms(5) by(auto simp add: \( \text{conc}\) \( \text{def}\) )

have \( \text{TS}_{inv}(\text{config}'\ \text{rand}\ (\text{embed}(rTS\ h0))\ \text{return}_{pmf}(\{x, y\}, h))\ qs)\ (\text{last}\ qs)\ [x, y] \land\ T_{on\ rand}'(\text{embed}(rTS\ h0))\ \text{return}_{pmf}(\{x, y\}, h))\ qs = 0 \)

274
apply(simp only: T_on'_embed[symmetric] config'_embed)
using TS_d'[OF A C B ] A lqs unfolding TS_inv'_det by auto
}

show ?thesis unfolding S
using kas apply(rule disjE)
apply(simp only:)
apply(rule d)
using assms apply(simp)
using assms apply(simp add: conc_assoc)
using h apply(simp)
apply(simp only:)
apply(subst TS_inv_sym[of y x y])
using assms(1) apply(simp)
defer
apply(rule d)
using assms apply(simp)
using assms apply(simp add: conc_assoc)
using h apply(simp)
using last_in_set l assms(4) by blast

qed

15.4 Phase Partitioning

lemma D': assumes σ' ∈ Lxx x y and x ≠ y and TS_inv' ([x, y], h) x [x, y]
shows T_on' (rTS h0) ([x, y], h) σ' ≤ 2 * T_p [x, y] σ' (OPT2 σ' [x, y])

∧ TS_inv (config'_rand (embed (rTS h0)) (return_pmf ([x, y], h)) σ')
(last σ') [x, y]

proof –

from config'_embed have config'_rand (embed (rTS h0)) (return_pmf ([x, y], h)) σ'
= return_pmf (Partial_Cost_Model.config' (rTS h0)) ([x, y], h) σ') by blast

then have L: TS_inv (config'_rand (embed (rTS h0)) (return_pmf ([x, y], h)) σ')
(last σ') [x, y]
= TS_inv' (config' (rTS h0)) ([x, y], h) σ') (last σ') [x, y] by auto
from \textit{assms}(3) have
\[ h' = \emptyset \lor (\exists h. h = [x, x] @ h) \]
by (auto simp add: \textit{TS\_inv\_det})

have \( T_{on'}(rTS h) [[x, y], h) \sigma' \leq 2 * T_p[x, y] \sigma'(OPT2 \sigma'[x, y]) \)
\& \( TS_{inv'}(\text{config'}(rTS h)) [[x, y], h) \sigma' (last \sigma') [x, y] \)
apply (rule \textit{LxxE}(OF \textit{assms}(1)))
using \( TS_d'(OF \textit{assms}(2), h, OF \sigma') \) apply (simp)
using \( TS_b'(OF \textit{assms}(2), h) \) apply (simp)
using \( TS_c'(OF \textit{assms}(2), h) \) apply (simp)
using \( TS_a'(OF \textit{assms}(2), h) \) apply fast
done

then show ?thesis using \( \textit{L} \) by (auto)
qed

theorem \textit{TS\_OPT2'}: \((x::nat) \neq y \implies \text{set} \sigma \subseteq \{x,y\} \)
\implies T_p.on (rTS []) [x,y] \sigma \leq 2 * \text{real}(T_p_opt [x,y] \sigma) + 2
apply (subst \textit{T\_on\_embed})
apply (rule \textit{Phase\_partitioning\_general}(where \( P=\textit{TS\_inv} \))
apply (simp)
apply (simp)
apply (simp)
apply (simp add: \textit{TS\_inv\_def} \textit{rTS\_def})
proof (goal_cases)
case (1 a b \sigma' s)
from 1 obtain \( h \) \( \text{hist}' \) where \( s: s = \text{return\_pmf} \([a, b], h) \)
\& \( h = \emptyset \lor h = [a,a'] h) \) unfolding \textit{TS\_inv\_def} apply (cases a=hd [x,y])
apply (simp) using 1 apply fast
apply (simp) using 1 by blast

from 1 have \( xyab: \textit{TS\_inv'} ([a, b], h) a [x, y] \)
\= \( \textit{TS\_inv'} ([a, b], h) a [a, b] \)
by (auto simp add: \textit{TS\_inv\_det})

with 1(6) s have \( \text{inv: \textit{TS\_inv'}} ([a, b], h) a [a, b] \) by (simp)

from \( \sigma' \in \text{Lxx a b} \) have \( \sigma' \neq [] \) using \textit{Lxx1} by fastforce
then have \( l: \text{last } \sigma' \in \{x,y\} \) using 1(5,7) last_in_set by blast

show ?case unfolding \( s: \textit{T\_on'}\_\text{embed}[\text{symmetric}] \)
using \( \textit{D'}(OF 1(3,4) \text{ inv, of []}) \)
apply (safe)

276
apply linarith
using TS_inv_sym[OF 1(4,5)] l apply blast
done

qed

15.5 TS is pairwise

lemma config'_distinct[simp]:
  shows distinct (fst (config' A S qs)) = distinct (fst S)
apply (induct qs rule: rev_induct) by(simp_all add: config'_snoc Step_def
split_def distinct_step)

lemma config'_set[simp]:
  shows set (fst (config' A S qs)) = set (fst S)
apply (induct qs rule: rev_induct) by(simp_all add: config'_snoc Step_def
split_def set_step)

lemma s_TS_append: i≤length as \implies s_TS init h (as@bs) i = s_TS init h
as i
by (simp add: s_TS_def)

lemma s_TS_distinct: distinct init \implies i<length qs \implies distinct (fst (TSdet
init h qs i))
by(simp_all add: config_config_distinct)

lemma othersdontinterfere: distinct init \implies i < length qs \implies a\in set init
\implies set qs \subseteq set init \implies qs!i\notin\{a,b\} \implies a < b in s_TS init h qs i \implies
a < b in s_TS init h qs (Suc i)
apply(simp add: s_TS_def split_def take_Suc_conv_app_nth config_append Step_def
step_def)
apply(subst x_stays_before_y_if_y_not_moved_to_front)
apply(simp_all add: config_config_distinct config_config_set)
by(auto simp: rTS_def TS_step_d_def)

lemma TS_mono:
  fixes l::nat
  assumes 1: x < y in s_TS init h xs (length xs)
  and Lin_cs: l ≤ length cs
  and firstocc: \\forall j<l. cs ! j \neq y
  and x \notin set cs
  and di: distinct init
  and inin: set (xs @ cs) \subseteq set init
  shows x < y in s_TS init h (xs@cs) (length (xs)+1)
proof -
  from before_in_setD2[OF 1] have y: y : set init unfolding s_TS_def
  by(simp add: config_config_set)
  from before_in_setD1[OF 1] have x: x : set init unfolding s_TS_def
  by(simp add: config_config_set)

  {
    fix n
    assume n ≤ l
    then have x < y in s_TS init h ((xs)@cs) (length (xs)+n)
    proof (induct n)
      case 0
      show ?case apply (simp only: s_TS_append) using 1 by(simp)
    next
      case (Suc n)
      then have n ≤ l: n < l by auto
      show ?case apply(simp)
      apply(rule othersdontinterfere)
      apply(rule di)
      using n ≤ l l_in_cs apply(simp)
      apply(fact x)
      apply(fact y)
      apply(fact inin)
      apply(simp add: nth_append) apply(safe)
      using assms(4) n ≤ l l_in_cs apply fastforce
      using firstocc n ≤ l apply blast
      using Suc(1) n ≤ l by(simp)
    qed
  }
  — before the request to y, x is in front of y
  then show x < y in s_TS init h (xs@cs) (length (xs)+l)
    by blast
  qed

lemma step_no_action: step s q (0,[]) = s
unfolding step_def mtf2_def by simp

lemma s_TS_set: i ≤ length qs ==> set (s_TS init h qs i) = set init
apply(induct i)
  apply(simp add: s_TS_def )
  apply(simp add: s_TS_def TSDet_Suc)
by(simp add: split_def rTS_def Step_def step_def)

lemma count_notin2: count_list xs x = 0 ==> x ∉ set xs
apply (induction xs) apply (auto del: count_notin)
apply(case_tac a=x) by(simp_all)+

lemma count_append: count_list (xs@ys) x = count_list xs x + count_list ys x
apply(induct xs) by(simp_all)

lemma count_rev: count_list (rev xs) x = count_list xs x
apply(induct xs) by(simp_all add: count_append)

lemma mtf2_q_passes: assumes q ∈ set xs distinct xs
and index xs q – n ≤ index xs x index xs x < index xs q
shows q < x in mtf2 n q xs
proof –
from assms have index xs q < length xs by auto
with assms(4) have ind_x: index xs x < length xs by auto
then have xinxs: x ∈ set xs using index_less_size_conv by metis

have B: index (mtf2 n q xs) q = index xs q – n
apply(rule mtf2_q_after)
by(fact)+
also from ind_x mtf2_forward_effect3[OF assms]
have A: ... < index (mtf2 n q xs) x by auto
finally show ?thesis unfolding before_in_def using xinxs by force
qed

lemma twotox:
assumes count_list bs y ≤ 1
and distinct init
and x ∈ set init
and y : set init
and x ≠ y
shows x < y in s_T S init h (as@[x]@bs@[x]) (length (as@[x]@bs@[x]))
proof –
have aa: snd (TSdet init h (as @ x # bs) @ [x]) (Suc (length as + length bs))
  = rev (take (Suc (length as + length bs)) ((as @ x # bs) @ [x])) @ h
apply(rule sndTSdet) by(simp)
then have aa': snd (TSdet init h (as @ x # bs @ [x]) (Suc (length as + length bs)))
  = rev (take (Suc (length as + length bs)) ((as @ x # bs @ [x])) @ h by auto
have lasocx: index (snd (TSdet init h (as @ x # bs) @ [x]) (Suc (length
as + length bs))) x = length bs

unfolding aa

apply(simp add: del: config\', simps)

using assms(5) by(simp add: index_append)

then have lasocex': (index (snd (TSdet init h (as @ x # bs @ [x]) (Suc
(length as + length bs)))) x) = length bs by auto

let ?sincelast = take (length bs)

(snd (TSdet init h ((as @ x # bs @ [x])
(Suc (length as + length bs))))

have sl: ?sincelast = rev bs unfolding aa by(simp)

let ?S = \{xa. xa < x in fst (TSdet init h (as @ x # bs @ [x])
(Suc (length as + length bs))) ∧
    count_list ?sincelast xa ≤ 1\}

have y: y ∈ ?S ∨ ~ y < x in s_TS init h (as @ x # bs @ [x]) (Suc
(length as + length bs))

unfolding sl unfolding s_TS_def using assms(1) by(simp add: count_rev
del: config\', simps)

have eklr: length (as@[x]@bs@[x]) = Suc (length (as@[x]@bs@x)) by simp

have 1: s_TS init h (as@[x]@bs@[x]) (length (as@[x]@bs@x))
    = fst (Partial_Cost_Model.Step (rTS h)
    (TSdet init h (as@[x]@bs@[x])
    (length (as@[x]@bs@x))
    ((as@[x]@bs@[x]) ! length (as@[x]@bs@x))) unfolding s_TS_def

unfolding eklr apply(subst TSdet_Suc)
    by(simp_all add: split_def)

have brrr: x∈ set (fst (TSdet init h (as @ x # bs @ [x]) (Suc (length as +
    length bs))))

    apply(subst s_TS_set[unfolded s_TS_def])
    apply(simp) by fact

have ydrin: y∈ set (fst (TSdet init h (as @ x # bs @ [x]) (Suc (length as +
    length bs))))

    apply(subst s_TS_set[unfolded s_TS_def]) apply(simp) by fact

have dbrrr: distinct (fst (TSdet init h (as @ x # bs @ [x]) (Suc (length as +
    length bs))))

    apply(subst s_TS_distinct[unfolded s_TS_def]) using assms(2) by(simp_all)

show ?thesis

proof (cases y < x in s_TS init h (as @ x # bs @ [x]) (Suc (length as +
    length bs)))

    case True

280
with \( y \) have \( yS: y \in S \) by auto

then have minsteps: \( \text{Min} (\text{index} (\text{fst} (\text{TSdet init} h (\text{as @ x # bs @ [x]})) (\text{Suc} (\text{length as + length bs})))) \) \( \leq \) \( \text{index} (\text{fst} (\text{TSdet init} h (\text{as @ x # bs @ [x]})) (\text{Suc} (\text{length as + length bs})))) \) \( y \)

by auto

let \( ?\text{entf} = \text{index} (\text{fst} (\text{TSdet init} h (\text{as @ x # bs @ [x]})) (\text{Suc} (\text{length as + length bs})))) \) \( \text{Min} (\text{index} (\text{fst} (\text{TSdet init} h (\text{as @ x # bs @ [x]})) (\text{Suc} (\text{length as + length bs})))) \) \( ?S \)

from minsteps have br: \( \text{index} (\text{fst} (\text{TSdet init} h (\text{as @ x # bs @ [x]})) (\text{Suc} (\text{length as + length bs})))) \) \( x - (\text{?entf}) \)

\( \leq \) \( \text{index} (\text{fst} (\text{TSdet init} h (\text{as @ x # bs @ [x]})) (\text{Suc} (\text{length as + length bs})))) \) \( y \)

by presburger

have brr: \( \text{index} (\text{fst} (\text{TSdet init} h (\text{as @ x # bs @ [x]})) (\text{Suc} (\text{length as + length bs})))) \) \( y \)

\( < \) \( \text{index} (\text{fst} (\text{TSdet init} h (\text{as @ x # bs @ [x]})) (\text{Suc} (\text{length as + length bs})))) \) \( x \)

using True unfolding before_in_def s_Ts_def by auto

from br brr have klo: \( \text{index} (\text{fst} (\text{TSdet init} h (\text{as @ x # bs @ [x]})) (\text{Suc} (\text{length as + length bs})))) \) \( x - (\text{?entf}) \)

\( \leq \) \( \text{index} (\text{fst} (\text{TSdet init} h (\text{as @ x # bs @ [x]})) (\text{Suc} (\text{length as + length bs})))) \) \( y \)

\( \land \) \( \text{index} (\text{fst} (\text{TSdet init} h (\text{as @ x # bs @ [x]})) (\text{Suc} (\text{length as + length bs})))) \) \( y \)

\( < \) \( \text{index} (\text{fst} (\text{TSdet init} h (\text{as @ x # bs @ [x]})) (\text{Suc} (\text{length as + length bs})))) \) \( x \) by metis

let \( ?\text{result} = (\text{mtf2} \text{?entf} x (\text{fst} (\text{TSdet init} h (\text{as @ x # bs @ [x]})) (\text{Suc} (\text{length as + length bs})))) \)

have whatthat: \( s_Ts \text{ init} h (\text{as @ [x] @ bs @ [x]}) (\text{length (as @ [x] @ bs @ [x])}) \)

= \( ?\text{result} \)

unfolding 1 apply(simp add: split_def step_def rTs_def Step_def Ts_step_d_def del: config.simps)

apply(simp add: nth_append del: config.simps)

using lasocc _[unfolded rTs_def] aa _[unfolded rTs_def]

apply(simp add: del: config.simps)

using yS[unfolded sl rTs_def] by auto

281
have ydrinee: y ∈ set (mtf2 ?entf x (fst (TSdet init h (as @ x # bs @ [x])) (Suc (length as + length bs)))))
apply subst_set_mtf2
apply subst s_TS_set [unfolded s_TS_def] apply simp by fact

show ?thesis unfolding whatthat apply (rule mtf2_q_passes) by (fact +

next
case False
then have 2: x < y in s_TS init h (as @ x # bs @ [x]) (Suc (length as + length bs))
using brrr ydrin not before in assms (6) unfolding s_TS_def by metis

{ fix e
have x < y in mtf2 e x (s_TS init h (as @ x # bs @ [x]) (Suc (length as + length bs)))
apply (rule x stays before y if y not moved to front)
unfolding s_TS_def
apply (fact +
using assms (6) apply simp
using 2 unfolding s_TS_def by simp
}

note bratz = this

show ?thesis unfolding 1 apply (simp add: TSnopaid split_def Step_def s_TS_def TS_step_d_def step_def nth_append del: config', simps)
using bratz [unfolded s_TS_def] by simp

qed

lemma count_drop: count_list (drop n cs) x ≤ count_list cs x
proof
  have count_list cs x = count_list (take n cs @ drop n cs) x by auto
  also have ... = count_list (take n cs) x + count_list (drop n cs) x by (rule count_append)
  also have ... ≥ count_list (drop n cs) x by auto
  finally show ?thesis .
qed

lemma count_take_less: assumes n ≤ m
shows count_list (take n cs) x ≤ count_list (take m cs) x
proof
  from assms have count_list (take n cs) x = count_list (take n (take m
also have \( \ldots \leq \text{count}\_\text{list} \left( \text{take} \ n \ (\text{take} \ m \ cs) \ @ \ \text{drop} \ n \ (\text{take} \ m \ cs) \right) x \)
by (simp only: count\_append)
also have \( \ldots = \text{count}\_\text{list} \ (\text{take} \ m \ cs) \) \( x \)
by (simp only: append\_take\_drop\_id)
finally show \( ?\text{thesis} \).
qed

**Lemma count\_take:** \( \text{count}\_\text{list} \ (\text{take} \ n \ cs) \ x \leq \text{count}\_\text{list} \ cs \ x \)

**Proof** –

have \( \text{count}\_\text{list} \ cs \ x = \text{count}\_\text{list} \ (\text{take} \ n \ cs \ @ \ \text{drop} \ n \ cs) \) \( x \) \( \text{by auto} \)
also have \( \ldots = \text{count}\_\text{list} \ (\text{take} \ n \ cs) \ + \ \text{count}\_\text{list} \ (\text{drop} \ n \ cs) \) \( x \)
by (rule count\_append)
also have \( \ldots \geq \text{count}\_\text{list} \ (\text{take} \ n \ cs) \) \( x \) \( \text{by auto} \)
finally show \( ?\text{thesis} \).
qed

**Lemma casexxy:** assumes \( \sigma = \text{as}[@x]@\text{bs}[@x]@cs \)
and \( x \notin \text{set} \ cs \)
and \( \text{set} \ cs \subseteq \text{set} \ \text{init} \)
and \( x \in \text{set} \ \text{init} \)
and \( \text{distinct} \ \text{init} \)
and \( x \notin \text{set} \ \text{bs} \)
and \( \text{set} \ \text{as} \subseteq \text{set} \ \text{init} \)
and \( \text{set} \ \text{bs} \subseteq \text{set} \ \text{init} \)
shows \( (\%i. \ i < \text{length} \ cs \longrightarrow (\forall j < i. \ cs!j \neq cs!i) \longrightarrow cs!i \neq x) \)
\( \longrightarrow (cs!i) \notin \text{set} \ \text{bs} \)
\( \longrightarrow x < (cs!i) \) in \( (\text{s}\_\text{TS} \ \text{init} \ h \ \sigma \ (\text{length} \ (\text{as}[@x]@\text{bs}[@x]) + i + 1)) \) \( i \)

**Proof** (rule infinite\_descent[where \( P = (\%i. \ i < \text{length} \ cs \longrightarrow (\forall j < i. \ cs!j \neq cs!i) \longrightarrow cs!i \neq x) \)
\( \longrightarrow (cs!i) \notin \text{set} \ \text{bs} \)
\( \longrightarrow x < (cs!i) \) in \( (\text{s}\_\text{TS} \ \text{init} \ h \ \sigma \ (\text{length} \ (\text{as}[@x]@\text{bs}[@x]) + i + 1)) \)],
**goal_cases**)

**Case** \( (1 \ i) \)

let \( ?y = cs!i \)
from \( I \) have \( i \_\text{in}_\text{cs}\): \( i < \text{length} \ cs \) \text{and}
firstocc: \( (\forall j < i. \ cs!j \neq cs!i) \)
and \( ynx: cs!i \neq x \)
and \( ynotinbs: cs!i \notin \text{set} \ \text{bs} \)
and \( y\_\text{before}_x': \neg x < cs!i \) in \( \text{s}\_\text{TS} \ \text{init} \ h \ \sigma \ (\text{length} \ (\text{as}@[x]@\text{bs}@[x]) + i + 1) \) \( \text{by auto} \)

have \( ss: \text{set} \ (\text{s}\_\text{TS} \ \text{init} \ h \ \sigma \ (\text{length} \ (\text{as}@[x]@\text{bs}@[x]) + i + 1)) = \text{set init using} \ \text{assms}(1) \) \( i \_\text{in}_\text{cs} \) \( \text{by (simp add: s_TS_set)} \)

283
then have \(cs!i \in \text{set} (s_{TS} \text{init} h \sigma (\text{length} (as \ @ \ [x] \ @ bs \ @ \ [x]) + i+1))\)

unfolding \(ss\) using \(\text{assms}(3)\) \(i\_\text{in} cs\) by fastforce

moreover have \(x : \text{set} (s_{TS} \text{init} h \sigma (\text{length} (as \ @ \ [x] \ @ bs \ @ \ [x]) + i+1))\)

unfolding \(ss\) using \(\text{assms}(4)\) by fastforce

— after the request to \(y\), \(y\) is in front of \(x\)

ultimately have \(y\_\text{before}_x Suct3: \ ?y < x \text{ in } s_{TS} \text{init} h \sigma (\text{length} (as \ @ \ [x] \ @ bs \ @ \ [x]) + i+1)\)

using \(y\_\text{before}_x' ynx \text{ not}_\text{before}_i \text{ in} \) by metis

from \(\text{ynotinbs}\) have \(\text{yatmostonceinbs} : \text{count}_\text{list} bs (cs!i) \leq 1\) by simp

let \(?y = cs!i\)

have \(y\_\text{ininit}: \ ?y \in \text{set init} \text{ using} \text{assms}(3) i\_\text{in} cs \text{ by fastforce}\)

\{ fix \(y\)
assume \(y \in \text{set init}\)
assume \(x \neq y\)
assume \(\text{count}_\text{list} bs y \leq 1\)
then have \(x < y \text{ in } s_{TS} \text{init} h (as@[x]@bs@[x]) (\text{length} (as@[x]@bs@[x]))\)
apply (rule \text{twotox}) by (fact)+
\} note \(xgostofront=\text{this}\)

with \(\text{yatmostonceinbs} ynx \text{ yininit} \) have \(\text{zeitpunktt2}: x < ?y \text{ in } s_{TS} \text{init} h (as@[x]@bs@[x]) (\text{length} (as@[x]@bs@[x]))\) by blast

have \(i \leq \text{length} cs \text{ using} i\_\text{in} cs \text{ by auto}\)

have \(x\_\text{before}_y \_\text{t3}: x < ?y \text{ in } s_{TS} \text{init} h ((as@[x]@bs@[x])@cs) (\text{length} (as@[x]@bs@[x])+i)\)
apply (rule \(\text{TS\_mono}\))
apply (fact)+
using \(\text{assms} \text{ by simp}\)

— so \(x\) and \(y\) swap positions when \(y\) is requested, that means that \(y\) was inserted infront of some elment \(z\) (which cannot be \(x\), has only been requested at most once since last request of \(y\) but is in front of \(x\))

— first show that \(y\) must have been requested in as

have \(\text{snd} (\text{TS}\_\text{det} \text{init} h (as \ @ [x] @ bs \ @ [x] @ cs) (\text{length} (as \ @ [x] @ bs \ @ [x]) + i)) =\)
\(\text{rev} (\text{take} (\text{length} (as \ @ [x] @ bs \ @ [x]) + i) (as \ @ [x] @ bs \ @ [x] @ cs)) \@ h\)
apply(rule sndTSdet) using i_in_cs by simp
also have \ldots = (rev (take i cs)) @ [x] @ (rev bs) @ [x] @ (rev as) @ h
by simp
finally have fstTS_i3: snd (TSdet init h (as @ [x] @ bs @ [x] @ cs)
(length (as @ [x] @ bs @ [x]) + i)) =
(rev (take i cs)) @ [x] @ (rev bs) @ [x] @ (rev as) @ h.
then have fstTS_i3': (snd (TSdet init h σ (Suc (Suc (length as + length bs + i)))) =
(rev (take i cs)) @ [x] @ (rev bs) @ [x] @ (rev as) @ h using
assms(1) by auto

let ?is = snd (TSdet init h (as @ [x] @ bs @ [x] @ cs) (length (as @ [x]
@ bs @ [x]) + i))
let ?is' = snd (config (rTS h) init (as @ [x] @ bs @ [x] @ (take i cs)))
let ?s = fst (TSdet init h (as @ [x] @ bs @ [x] @ cs) (length (as @ [x] @
bs @ [x]) + i))
let ?s' = fst (config (rTS h) init (as @ [x] @ bs @ [x] @ (take i cs)))
let ?s_Suc3'\_s_TS init h (as @ [x] @ bs @ [x] @ cs) (length (as @ [x] @
bs @ [x]) + i + 1)

let ?S = \{xa. (xa < (as @ [x] @ bs @ [x] @ cs) ! (length (as @ [x] @ bs
@ [x]) + i)) in ?s \} 
count_list (take (index ?is ((as @ [x] @ bs @ [x] @ cs) ! (length
(as @ [x] @ bs @ [x]) + i)) ?is) xa ≤ 1) \}
let ?S' = \{xa. (xa < (as @ [x] @ bs @ [x] @ cs) ! (length (as @ [x] @ bs
@ [x]) + i)) in ?s' \} 
count_list (take (index ?is' ((cs'!i))) ?is' xa ≤ 1) \}

have isis': ?is = ?is' by(simp)
have ss': ?s = ?s' by(simp)
then have SS': ?S = ?S' using i_in_cs by(simp add: nth_append)

have once: TSdet init h (as @ x # bs @ x # cs) (Suc (Suc (Suc (length
as + length bs + i))))
= Step (rTS h) (config_p (rTS h) init (as @ x # bs @ x # take i cs))
(cs ! i)
apply(subst TSdet_Suc)
using i_in_cs apply(simp)
by(simp add: nth_append)

have aha: (index ?is (cs ! i) ≠ length ?is)
∧ ?S ≠ {}
proof (rule ccontr, goal_cases)
  case 1
  then have (index ?is (cs ! i) = length ?is) ∨ ¬ ?S = {} by (simp)
  then have alters: (index ?is' (cs ! i) = length ?is') ∨ ¬ ?S' = {} 
  apply (simp only: SS') by (simp only: isis')
  — wenn (cs ! i) noch nie requested wurde, dann kann es gar nicht nach 
vorne gebracht werden! also widerspruch mit y_before_x'
  have ?s_Suc3 = fst (config (rTS h) init ((as @ [x] @ bs @ [x]) @ (take 
(i+1) cs)))
  unfolding s_Ts_def
  apply (simp only: length_append)
  apply (subst take_append)
  apply (subst take_append)
  apply (subst take_append)
  apply (simp)
  also have . . . = fst (config (rTS h) init (((as @ [x] @ bs @ [x]) @ (take 
i cs)) @ [cs!i]))
  using i_in_cs by (simp add: take_SucConv_app_nth)
  also have . . . = step ?s' ?y (0, [])
  proof (cases index ?is' (cs ! i) = length ?is')
  case True
  show ?thesis
  apply (subst config_append)
  using i_in_cs apply (simp add: rTS_def Step_def split_def nth_append)
  apply (subst TS_step_d_def)
  apply (simp only: True[unfolded rTS_def, simplified])
  by (simp)
  next
  case False
  with alters have S': ?S' = {} by simp

  have 1 : {xa. xa < cs ! i}
    in fst (Partial_Cost_Model.config' (λs. h, TS_step_d)
      (init, h)
      (as @ x # bs @ x # take i cs)) ∧
    count_list (take (index 
    (snd 
      (Partial_Cost_Model.config'
        (λs. h, TS_step_d) (init, h)
        (as @ x # bs @ x # take i cs)))
    (cs ! i))
    (snd 
      (Partial_Cost_Model.config'
        (λs. h, TS_step_d) (init, h)
        (as @ x # bs @ x # take i cs)))
(\lambda s. h, TS_{step_d}) \ (init, h) \\
(as \ @ x \ # bs \ @ x \ # \ take \ i \ cs))) \ xa \leq 1 = \{\} \ \textbf{using} \ S' \ \textbf{by}(simp \ add:\ rTS\_def nth\_append) \\

\textbf{show} \ \vartheta \ \textbf{thesis} \\
\ \textbf{apply}(subst \ config\_append) \\
\ \textbf{using} \ i\_in\_cs \ \textbf{apply}(simp \ add:\ rTS\_def \ Step\_def \ split\_def \ nth\_append) \\
\ \textbf{apply}(subst \ TS\_{step\_d}\_def) \\
\ \textbf{apply}(simp \ only:\ 1 \ Let\_def) \\
\ \textbf{by}(simp) \\
\ \textbf{qed} \\
\ \textbf{finally have} \ ?s\_{Suct3} = \ \textbf{step} \ ?s \ ?y \ (0, []) \ \textbf{using} \ ss' \ \textbf{by} \ \textbf{simp} \\
\ \textbf{then have} \ e: \ ?s\_{Suct3} = \ ?s \ \textbf{by}(simp \ only:\ \textbf{step\_no\_action}) \\
\ \textbf{from} \ x\_before\_y\_t3 \ \textbf{have} \ x < cs \ ! i \ \textbf{in} \ ?s\_{Suct3} \ \textbf{unfolding} \ e \ \textbf{unfolding} \ s\_{TS\_def} \ \textbf{by} \ \textbf{simp} \\
\ \ \textbf{with} \ y\_before\_x' \ \textbf{show} \ \textbf{False} \ \textbf{unfolding} \ \textbf{assms(1)} \ \textbf{by} \ \textbf{auto} \\
\ \textbf{qed} \\
\ \textbf{then have} \ aha': \ \textbf{index} \ (snd \ (TSdet \ init \ h \ (as \ @ x \ # bs \ @ x \ # cs) \ (Suc (length as + length bs + i)))) \\
\ \ \hspace{1em} (cs ! i) \neq \length \ (snd \ (TSdet \ init \ h \ (as \ @ x \ # bs \ @ x \ # cs) \ (Suc (length as + length bs + i)))) \\
\ \ \hspace{1em} \ \textbf{and} \\
\ \hspace{2em} \textbf{aha2}: \ ?S \neq \{\} \ \textbf{by} \ \textbf{auto} \\

\textbf{from} \ \textbf{fstTS\_t3}' \ \textbf{assms(1)} \ \textbf{have} \ ?is: \ ?is = (rev \ (take \ i \ cs)) \ @ [x] \ @ (rev \ bs) \ @ [x] \ @ (rev \ as) \ @ h \ \textbf{by} \ \textbf{auto} \\
\ \textbf{have} \ \textbf{minlencsi}: \ \textbf{min} \ (length \ cs) \ i = i \ \textbf{using} \ i\_in\_cs \ \textbf{by} \ \textbf{linarith} \\
\ \textbf{let} \ \textbf{?lastoccy}= (index \ (rev \ (take \ i \ cs)) \ @ x \ # \ rev \ bs \ @ x \ # \ rev \ as \ @ h) \\
\ \hspace{1em} \ (cs \ ! i)) \\
\ \textbf{have} \ ?y \ \notin \ \textbf{set} \ (rev \ (take \ i \ cs)) \ \textbf{using} \ \textbf{firstocce} \ \textbf{by}(simp \ add: \ \textbf{in\_set\_conv\_nth}) \\
\ \ \textbf{then have} \ \textbf{lastoccy}: \ \textbf{?lastoccy} \geq \\
\ \hspace{1em} \ i + 1 + \textbf{length} \ bs + 1 \ \textbf{using} \ \textbf{ynx} \ \textbf{ynotinbs} \ \textbf{minlencsi} \ \textbf{by}(simp \ add: \ \textbf{index\_append}) \\

\textbf{have} \ \textbf{x\_nin\_S}: \ x \notin ?S \\
\ \textbf{using} \ \textbf{is \_ apply}(simp \ add: \ \textbf{split\_def} \ \textbf{nth\_append} \ \textbf{del}: \ \textbf{config}', \textbf{simps}) \\
\ \textbf{proof} \ (\textbf{goal\_cases}) \\
\ \hspace{1em} \textbf{case} \ 1 \\
\ \hspace{2em} \textbf{have} \ \textbf{count\_list} \ (\textbf{take} \ \textbf{?lastoccy} \ (rev \ (take \ i \ cs))) \ x \leq
\begin{verbatim}
count_list (drop (length cs - i) (rev cs)) x by (simp add: count_take rev_take)
also have \ldots \leq count_list (rev cs) x by(simp add: count_drop)
also have \ldots = 0 using assms(2) by(simp add: count_rev)
finally have count_list (take \_lastoccy (rev (take i cs))) x = 0 by auto
have 2 \leq count_list ([x] @ rev bs @ [x]) x apply(simp only: count_append)
also have \ldots = count_list (take (1 + length bs + 1) (x # rev bs @ x # rev as @ h)) x by auto
also have \ldots \leq count_list (take (?lastoccy - i) (x # rev bs @ x # rev as @ h)) x
apply(rule count_take_less)
using lastoccy by linarith
also have \ldots \leq count_list (take (?lastoccy (rev (take i cs))) x
+ count_list (take (?lastoccy - i) (x # rev bs @ x # rev as @ h)) x
by(simp add: minlencsi count_append)
finally show \_case by presburger
qed

have Min (index ?s ' ?S) \in (index ?s ' ?S) apply(rule Min_in) using aha2 by (simp_all)
then obtain z where zminimal: index ?s z = Min (index ?s ' ?S) and
z_in_S: z \in ?S by auto
then have bef: z < (as @ [x] @ bs @ [x] @ cs) ! (length (as @ [x] @ bs @ [x]) + i) in ?s
and count_list (take (index ?is ((as @ [x] @ bs @ [x] @ cs) ! (length (as @ [x] @ bs @ [x]) + i))) ?is) z \leq 1 by(blast)+

with zminimal have zbefore: z < cs ! i in ?s
and zatmostonce: count_list (take (index ?is (cs ! i)) ?is) z \leq 1
and isminimal: index ?s z = Min (index ?s ' ?S) by(simp_all add: nth_append)
have elemns: z \in set ?s unfolding before_in_def by (meson zbefore before_in_setD1)
then have zinit: z \in set init
using i_in_cs by(simp add: s_TS_set[unfolded s_TS_def] del: config'.simps)
\end{verbatim}
from zbeforey have zbeforey_ind: index ?s z < index ?s ?y unfolding before_in_def by auto
then have el_n_y: z ≠ ?y by auto
have el_n_x: z ≠ x using x_in_S z_in_S by blast

{ fix s q
  have TS_step_d2: TS_step_d s q =
    (let V_r={x. x < q in fst s ∧ count_list (take (index (snd s) q) (snd s)) x ≤ l})
    in ((if index (snd s) q ≠ length (snd s) ∧ V_r ≠ {})
        then index (fst s) q − Min ( (index (fst s)) ' V_r)
        else 0,[],q#(snd s)))
    unfolding TS_step_d_def
        apply(cases index (snd s) q < length (snd s))
        using index_le_size apply(simp split: prod.split) apply blast
    by(auto simp add: index_less_size conv split: prod.split)
  note alt_chara=this
  have iF: (index (snd (config′ (λs. h, TS_step_d) (init, h) (as @ x # bs @ x # take i cs))) (cs ! i)
    ≠ length (snd (config′ (λs. h, TS_step_d) (init, h) (as @ x # bs @ x # take i cs))) ∧
    {xa. xa < cs ! i in fst (config′ (λs. h, TS_step_d) (init, h) (as @ x # bs @ x # take i cs)) ∧
        count_list
        (take (index (snd (config′ (λs. h, TS_step_d) (init, h) (as @ x # bs @ x # take i cs))) (cs ! i))
        (snd (Partial_Cost_Model.config′ (λs. h, TS_step_d) (init, h) (as @ x # bs @ x # take i cs))))
    xa
    ≤ l} ≠
    {()} = True using aha[unfolded rTS_def] ss' SS' by(simp add: nth_append)

  have ?S_Suct3 = fst (TSdet init h (as @ x # bs @ x # cs) (Suc (Suc (Suc (length as + length bs + i)))))
    by(auto simp add: s_TS_def)
  also have ... = step ?s ?y (index ?s ?y − Min (index ?s ' ?S), [])
    apply(simp only: once[unfolded assms(1)])
    apply(simp add: Step_def split_def rTS_def del: config'.simp)
    apply(subst alt_chara)
    apply(simp only: Let_def )
}
apply(simp only: iF)
  by(simp add: nth_append)
finally have \(\pi_S \text{Suct3} = \text{step} \ ?s \ ?y \ (\text{index} \ ?s \ ?y - \text{Min} (\text{index} \ ?s \ ' \ ?S), [])\).
with isminimal have state_dannach: \(\pi_S \text{Suct3} = \text{step} \ ?s \ ?y \ (\text{index} \ ?s \ ?y - \text{index} \ ?s \ ?z, [])\) by presburger

— so \(y\) is moved in front of \(z\), that means:

```plaintext
have yinfrontofz: \(?y < z \text{ in } s_{TS} \text{ init } h \sigma \ (\text{length } (as @ [x] @ bs @ [x]) + i + 1)\)
unfolding assms(1) state_dannach apply(simp add: step_def del: config'.simps)
apply(rule mtf2_q_passes)
using i_in_cs assms(5) apply(simp_all add: s_{TS}.distinct[unfolded s_{TS}.def] s_{TS}.set[unfolded s_{TS}.def])
using yininit apply(simp)
using zbeforey_end by simp
```

```plaintext
have yins: \(?y \in \text{ set } ?s\)
  using i_in_cs assms(3,5) apply(simp_all add: s_{TS}.set[unfolded s_{TS}.def] del: config'.simps)
  by fastforce
```

```plaintext
have index ?s_Suct3 ?y = index ?s z
  and index ?s_Suct3 z = Suc (index ?s z)
proof –
  let \(?xs = (\text{fst } (\text{TSdet init } h \ (\text{as @ x @ bs @ x @ cs}) \ (\text{Suc (Suc (length as + length bs + i))})))\)
  have setxs: \(?xs = \text{ set init}\)
    apply(rule s_{TS}.set[unfolded s_{TS}.def])
    using i_in_cs by auto
then have yinxs: cs ! i \in \text{ set } \?xs
  apply(simp add: setxs del: config'.simps)
  using assms(3) i_in_cs by fastforce
```

```plaintext
have distinctxs: distinct \(?xs\)
  apply(rule s_{TS}.distinct[unfolded s_{TS}.def])
  using i_in_cs assms(5) by auto
```

290
let \( n = (\text{index} (\text{fst} (\text{TSdet init } h \ (\text{as @ x # bs @ x # cs}) (\text{Suc (\text{length as + length bs + i)})))))) \)

\( (\text{cs ! i}) - \text{index} (\text{fst} (\text{TSdet init } h \ (\text{as @ x # bs @ x # cs}) (\text{Suc (\text{length as + length bs + i)})))) z) \)

have \( \text{index} (\text{mtf2 ?n ?y ?xs}) (\text{?xs ! index ?xs ?y}) = \text{index ?xs ?y} - \text{?n} \wedge \text{index ?xs ?y} - \text{?n} = \text{index} (\text{mtf2 ?n ?y ?xs}) (\text{?xs ! index ?xs ?y}) \)

apply (rule \text{mtf2\_forward\_effect2})
apply (fact)
apply (fact)
by simp

then have \( \text{index} (\text{mtf2 ?n ?y ?xs}) (\text{?xs ! index ?xs ?y}) = \text{index ?xs ?y} - \text{?n} \) by metis
also have \( \ldots = \text{index ?s z} \) using \text{zbeforey\_ind} by force
finally have A: \( \text{index} (\text{mtf2 ?n ?y ?xs}) (\text{?xs ! index ?xs ?y}) = \text{index ?s z} \).

have aa: \( \text{index ?xs ?y} - \text{?n} \leq \text{index ?xs z index ?xs z < index ?xs ?y} \)
apply (simp)
using \text{zbeforey\_ind} by fastforce

from \text{mtf2\_forward\_effect3} "OF yinxs distinctxs aa"
have B: \( \text{index} (\text{mtf2 ?n ?y ?xs}) z = \text{Suc (index ?xs z)} \)
using elemins yins by (simp add: nth_append split_def del: config'.simps)

show \( \text{index ?s\_Suct3 ?y = index ?s z} \)
unfolding state\_dannach apply (simp add: step_def nth_append del: config'.simps)
using A yins by (simp add: nth_append del: config'.simps)

show \( \text{index ?s\_Suct3 z = Suc (index ?s z)} \)
unfolding state\_dannach apply (simp add: step_def nth_append del: config'.simps)
using B yins by (simp add: nth_append del: config'.simps)

qed

then have are: \( \text{Suc (index ?s\_Suct3 ?y) = index ?s\_Suct3 z} \) by presburger
from are before_in_def y before_x_Suct3 el_n_x assms(1) have z before_x: 
z < x in ?s_Suct3
  by (metis Suc_lessI not_before_in y infrontofz)

have xSuct3: x ∈ set ?s_Suct3 using assms(4) i in cs by (simp add: s_TS_set)
have elSuct3: z ∈ set ?s_Suct3 using z ininit i in cs by (simp add: s_TS_set)

have xt3: x ∈ set ?s apply (subst config config_set) by fact

note elt3 = elms

have z_s: z < x in ?s
proof (rule ccontr, goal_cases)
  case 1
  then have x < z in ?s using not_before_in[OF xt3 elt3] el_n_x unfolding s_TS_def by blast
  then have x < z in ?s_Suct3
    apply (simp only: state dannach)
    apply (simp only: step_def)
    apply (simp add: nth_append del: config''.simps)
    apply (rule x stays before_y if_y not moved to_front)
    apply (subst config config_set) using i in cs assms(3) apply fastforce
    apply (subst config config_set distinct) using assms(5) apply fastforce
    apply (subst config config_set) using assms(4) apply fastforce
    apply (subst config config_set) using z ininit apply fastforce
    using el_n_y apply (simp)
    by (simp)

  then show False using z before_x not before_in[OF xSuct3 elSuct3] by blast
qed

have mind: (index ?is (cs ! i)) ≥ i + 1 + length bs + 1 using lastoccy
  using i in cs fstTS_t3'[unfolded assms(1)] by (simp add: split_def nth_append del: config''.simps)

have count_list (rev (take i cs) @ [x] @ rev bs @ [x]) z=
\[
\text{count_list} \ (\text{take} \ ((i + 1 + \text{length } bs + 1) \ ?is) \ z) \text{ unfolding is } \]
\[
\text{using el_n_x by (simp add: minlencsi count_append )}
\]
also from mind have …
\[
\leq \text{count_list} \ (\text{take} \ (\text{index } ?is \ (cs \ i)) \ ?is) \ z
\]
by (rule count_take_less)
also have ... ≤ 1 using zatmo1stnce by metis

finally have aaa: \text{count_list} \ (\text{rev} \ (\text{take} \ i \ cs \ @ \ [x] \ @ \ \text{rev } bs \ @ \ [x])) \ z ≤ 1 

with el_n_x have count_list bs z + count_list (take i cs) z ≤ 1
by (simp add: count_append count_rev)

moreover have count_list (take (Suc i) cs) z = count_list (take i cs) z
using i_in_cs el_n_y by (simp add: take_Suc_conv_app_nth count_append)

ultimately have aaaa: \text{count_list} \ bs z + \text{count_list} \ (\text{take} \ (Suc i) \ cs) \ z ≤ 1 by simp

have \text{z occurs once in cs: }\text{count_list} \ (\text{take} \ (Suc i) \ cs) \ z = 1
proof (rule ccontr, goal_cases)
case 1
with aaaa have atmost1: \text{count_list} \ bs z ≤ 1 \text{ and count_list} \ (\text{take} \ (Suc i) \ cs) \ z = 0 by force+

have yeah: \text{z} \notin \text{set} \ (\text{take} \ (Suc i) \ cs) \text{ apply (rule count_notin2) by fact}

— now we know that x is in front of z after 2nd request to x, and that z is not requested any more, that means it stays behind x, which leads to a contradiction with z_before_x

have \text{zin123: } x \in \text{set} \ s_{\text{TS}} \ \text{init h} \ (((\text{as } \circ [x] \ \circ bs \ @ \ [x]) \ @ \ (\text{take} \ (i+1) \ cs)) \ (\text{length} \ (\text{as } \circ [x] \ \circ bs \ @ \ [x]) + (i+1)))
using i_in_cs assms(4) by (simp add: s_{\text{TS set}})

have \text{zin123: } \text{z} \in \text{set} \ s_{\text{TS}} \ \text{init h} \ (((\text{as } \circ [x] \ \circ bs \ @ \ [x]) \ @ \ (\text{take} \ (i+1) \ cs)) \ (\text{length} \ (\text{as } \circ [x] \ \circ bs \ @ \ [x]) + (i+1)))
using i_in_cs elemin by (simp add: s_{\text{TS set}} del: config',simp)

have \text{x < z in s_{\text{TS init h}} (((as } \circ [x] \ \circ bs \ @ \ [x]) \ @ (\text{take} \ (i+1) \ cs)) \ (\text{length} \ (\text{as } @ [x] \ \circ bs \ @ [x]) + (i + 1))
apply (rule TS_mono)
apply (rule zgoestofront)
apply (fact) using el_n_x apply (simp) apply (fact)
using i_in_cs apply (simp)
using yeah i_in_cs length_take nth_mem
apply (metis Suc_eq_plus1 Suc_leI min_absorb2)
using set_take_subset assms(2) apply fast
using assms i_in_cs apply (simp_all) using set_take_subset by fast
then have ge: \text{¬ z < x in s_{\text{TS init h}} (((\text{as } @ [x] \ \circ bs \ @ [x]) \ @ (take}
(\(i+1\) \(cs\)) (\(\text{length} (\text{as} @ [x] @ \text{bs} @ [x]) + (i+1)\))

using not_before_in[\(OF \text{zin123 xin123}\) el \(n, x\) by blast

have \(s_{TS} \text{init h} ( ((\text{as} @ [x] @ \text{bs} @ [x]) @ cs)) (\text{length} (\text{as} @ [x] @ \text{bs} @ [x]) + (i+1))\)

\(= s_{TS} \text{init h} ( ((\text{as} @ [x] @ \text{bs} @ [x] @ (\text{take} (i+1) \text{cs})) @ (\text{drop} (i+1) \text{cs})) (\text{length} (\text{as} @ [x] @ \text{bs} @ [x]) + (i+1))\) by auto

also have ...

\(= s_{TS} \text{init h} ( ((\text{as} @ [x] @ \text{bs} @ [x] @ (\text{take} (i+1) \text{cs})) (\text{length} (\text{as} @ [x] @ \text{bs} @ [x]) + (i+1))\)

apply (rule \(s_{TS}\text{append}\))

using \(i\in cs\) by (simp)

finally have \(aaa: s_{TS} \text{init h} ( ((\text{as} @ [x] @ \text{bs} @ [x] @ cs)) (\text{length} (\text{as} @ [x] @ \text{bs} @ [x]) + (i+1))\)

\(= s_{TS} \text{init h} ( ((\text{as} @ [x] @ \text{bs} @ [x] @ (\text{take} (i+1) \text{cs})) (\text{length} (\text{as} @ [x] @ \text{bs} @ [x]) + (i+1))\).

from \(ge z_{before\_x}\) show False unfolding assms(1) using \(aaa\) by auto

qed

from \(z_{\text{occurs\_once\_in\_cs}}\) have \(kinSuci: z \in \text{set (take} (Suc i) \text{cs) by (metis One\_nat\_def count\_not in\_not Suc\_n)\)

then have \(zincs: z \in \text{set cs\ using set\_take\_subset\ by fast\)

from \(z_{\text{occurs\_once\_in\_cs}}\) obtain \(k\) where \(k_{\text{def}}: k=\text{index (take} (Suc i) \text{cs) z\ by blast\)

then have \(k=\text{index cs z}\ using kinSuci by (simp add: index\_take\_if\_set)\)

then have \(zcsk: z = cs[k\ using zincs\ by simp\)

have era: \(cs ! \text{index (take} (Suc i) \text{cs) z = z using kinSuci in\_set\_takeD index\_take\_if\_set\ by fastforce\)

have ki: \(k<i\ unfolding \(k_{\text{def}}\ using kinSuci el\_n\_y\)

by (metis i\_in\_cs\ index\_take\_index\_take\_if\_set\ le\_neg\_implies\_less\ not\_less\_eq\_eq\ yes\)

have \(zmustbebefore: cs[k < x in ?s\)

unfolding \(k_{\text{def}}\ era\ by (fact \(z\_s)\)

— before the request to \(z, x\) is in front of \(z,\) analog zu oben, vllt generell machen?
— element z does not occur between t1 and position k

have  \( z \notin \text{set bs} \)

proof –
from \( z \text{occurs\_once\_in\_cs} \) have \( \text{count\_list bs z} = 0 \) by auto
then show \( \text{thesis} \) using \( \text{zcsk count\_notin2} \) by metis

qed

have \( \text{count\_list bs z} \leq 1 \) using \( \text{aaaa} \) by linarith

with \( \text{zgoestofront[OF \text{init} e \oplus x[\text{symmetric}]]} \) have \( \text{xbeforez: } x < z \) in 
\( \text{s\_TS init h (as \oplus [x] \oplus bs \oplus [x]) (length (as \oplus [x] \oplus bs \oplus [x]))} \) by auto

obtain \( cs1 \) \( cs2 \) where \( \forall j < k. cs ! j \neq cs ! k \)
and \( \forall j < i - k - 1. cs2 ! j \neq cs ! k \)
proof (safe, goal\_cases)
case (1 j)
with \( ki \text{\_in\_cs} \) have \( 2: j < \text{length (take k cs)} \) by auto

have un1: \( \text{(take (Suc i) cs)!k = cs!k} \) apply (rule nth\_take) using \( ki \) by auto
have un2: \( \text{take k cs)!j = cs!j} \) apply (rule nth\_take) using 1(1) \( ki \) by auto

from \( i\_in\_cs \) \( ki \) have \( f1: k < \text{length (take (Suc i) cs)} \) by auto
then have \( \text{(take (Suc i) cs) = (take k (take (Suc i) cs)) \oplus (take (Suc i) cs)!k \oplus (drop (Suc k) (take (Suc i) cs))} \)
by (rule id\_take\_nth\_drop)
also have \( \text{(take k (Suc i) cs)) = take k cs \) using \( i\_in\_cs \) \( ki \) by (simp add: min\_def)
also have \( ... = (\text{take j (take k cs)}) \oplus (\text{take k cs})!j \oplus (\text{drop (Suc j) (take k cs)}) \)

using \( 2 \) by (rule id\_take\_nth\_drop)
finally have \( \text{take (Suc i) cs} \)
\( = (\text{take j (take k cs)}) \oplus [(\text{take k cs})!j] \oplus (\text{drop (Suc j) (take k cs)}) \)
\( \oplus [(\text{take (Suc i) cs})!k] \oplus (\text{drop (Suc k) (take (Suc i) cs)}) \)
by (simp)
then have \( A: \text{take (Suc i) cs} \)
\( = (\text{take j (take k cs)}) \oplus [cs!j] \oplus (\text{drop (Suc j) (take k cs)}) \)
\( \oplus [cs!k] \oplus (\text{drop (Suc k) (take (Suc i) cs)}) \)
unfolding un1 un2 by simp
have count_list ((take j (take k cs)) @ [cs] @ (drop (Suc j) (take k cs)))
   @ [cs] @ (drop (Suc k) (take (Suc i) cs))) z ≥ 2
apply(simp add: count_append)
using zesk 1(2) by(simp)
with A have count_list (take (Suc i) cs) z ≥ 2 by auto
with z_occurs_once_in_cs show False by auto
next
  case (2 j)
  then have 1: Suc k+j < i by auto
  then have f2: j < length (drop (Suc k) (take (Suc i) cs)) using i_in_cs
by simp
  have 3: (drop (Suc k) (take (Suc i) cs)) = take j (drop (Suc k) (take (Suc i) cs))
    @ (drop (Suc k) (take (Suc i) cs))! j
    # drop (Suc j) (drop (Suc k) (take (Suc i) cs))
    using f2 by(rule id_take_nth_drop)
  have (drop (Suc k) (take (Suc i) cs))! j = (take (Suc i) cs) ! (Suc k+j)
    apply(rule nth_drop) using i_in_cs 1 by auto
  also have ... = cs ! (Suc k+j) apply(rule nth_take) using 1 by auto
  finally have 4: (drop (Suc k) (take (Suc i) cs)) = take j (drop (Suc k) (take (Suc i) cs))
    @ cs! (Suc k+j)
    # drop (Suc j) (drop (Suc k) (take (Suc i) cs))
    using 3 by auto
  have 5: cs2 ! j = cs! (Suc k+j) unfolding cs2
    apply(rule nth_drop) using i_in_cs 1 by auto
  from 4 5 2(2) have 6: (drop (Suc k) (take (Suc i) cs)) = take j (drop (Suc k) (take (Suc i) cs))
    @ cs! k
    # drop (Suc j) (drop (Suc k) (take (Suc i) cs)) by auto
  from i_in_cs ki have 1: k < length (take (Suc i) cs) by auto
  then have 7: (take (Suc i) cs) = (take k (take (Suc i) cs)) @ (take (Suc i) cs)!k # (drop (Suc k) (take (Suc i) cs))
    by(rule id_take_nth_drop)
  have 9: (take (Suc i) cs)!k = z unfolding zesk apply(rule nth_take)
  using ki by auto
  from 6 7 have A: (take (Suc i) cs) = (take k (take (Suc i) cs)) @ z #
    take j (drop (Suc k) (take (Suc i) cs))

296
have \( k \text{ in } cs \): \( k < \text{length } cs \) using \( ki \text{ in } cs \) by auto

with \( cs1 \) have \( \text{lenkk: length } cs1 = k+1 \) by auto

from \( k \text{ in } cs \) have \( \text{mincsk: min (length } cs \text{) (Suc } k \) = Suc } k \) by auto

have \( s,TS \text{ init } h (((\text{as@}[x]@\text{bs}@[x])@cs1)@cs2) (\text{length } (\text{as@}[x]@\text{bs}@[x])+k+1) = s,TS \text{ init } h (((\text{as@}[x]@\text{bs}@[x])@cs1)) (\text{length } (\text{as@}[x]@\text{bs}@[x])+k+1) \) applying \( \text{rule } s,TS \text{ append} \)

using \( cs1 \text{ cs2 } k \text{ in } cs \) by(simp)

then have \( \text{splitter: } s,TS \text{ init } h (((\text{as@}[x]@\text{bs}@[x])@cs1)) (\text{length } (\text{as@}[x]@\text{bs}@[x]@@)(cs1))) = s,TS \text{ init } h (((\text{as@}[x]@\text{bs}@[x])@cs)) (\text{length } (\text{as@}[x]@\text{bs}@[x])+k+1) \) using \( \text{lenkk v cs1 apply(auto)} \) by (simp add: add.commute add.left.commute)

from \( cs2 \) have \( \text{length } cs2 = \text{length } cs - (\text{Suc } k) \) by auto

have \( \text{notxbeqz: } \sim x < z \text{ in } s,TS \text{ init } h \sigma \text{ (length } (\text{as @ } [x] @ \text{bs @ } [x]) + k + 1) \) proof (rule ccontr, goal_cases)

  case 1
  then have \( a: x < z \text{ in } s,TS \text{ init } h (((\text{as@}[x]@\text{bs}@[x])@cs1)) \text{ (length } (\text{as@}[x]@\text{bs}@[x]@@)(cs1))) \)

  unfolding \( \text{splitter } \text{assms(1)} \) by auto

  have \( 41: x \in \text{set}(s,TS \text{ init } h (((\text{as@}[x]@\text{bs}@[x])@cs)) (\text{length } (\text{as @ } [x] @ \text{bs @ } [x]) + i)) \)

  using \( i,\text{in } cs \text{ assms(4)} \) by(simp add: s,TS_set)

have \( 42: z \in \text{set}(s,TS \text{ init } h (((\text{as @ } [x] @ \text{bs @ } [x]) @ cs)) (\text{length } (as
have rew: $s_{\text{TS}} \text{init} \; h \; (\text{length} (as@x@bs@[x]@cs1)+i) = s_{\text{TS}} \text{init} \; h \; (\text{length} (as@x@bs@[x])+i)$

using $cs1 \; v \; ki$ apply(simp add: mincsk) by (simp add: add.commute add.left_commute)

have $x < z$ in $s_{\text{TS}} \text{init} \; h \; (\text{length} (as@x@bs@[x]@cs1)+i)$

apply(rule $TS_{\text{mono}}$)
using $a$ apply(simp)
using $cs2$ $i_{\text{in-cs}}$ $ki \; v \; cs1$ apply(simp)
using $z_{\text{lastocc}} \; zcsk$ apply(simp)
using $v \; assms(2)$ apply force
using $assms$ by(simp_all add: $cs1 \; cs2$)

from $\text{zmustbebefore}$ $\text{this[unfolded rew]}$ $\text{el}_n \; x \; zcsk$ $41 \; 42 \; \text{not before in}$

show False
unfolding $s_{\text{TS def}}$ by fastforce

from $\text{zmustbebefore}$ $\text{this[unfolded rew]}$ $\text{el}_n \; x \; zcsk$ $41 \; 42 \; \text{not before in}$

show $\text{False}$
unfolding $s_{\text{TS def}}$ by fastforce

have $1$: $k < \text{length} \; cs$

$(\forall j < k. \; cs ! j \neq cs ! k)$

$cs ! k \neq x \; cs ! k \notin \text{set} \; bs$

$\sim x < z$ in $s_{\text{TS init}} \; h \; \sigma \; (\text{length} (as@x@bs@[x])+k+1)$

apply(safe)
using $ki$ $i_{\text{in-cs}}$ apply(simp)
using $z_{\text{firstocc}}$ apply(simp)
using $assms(2)$ $ki \; i_{\text{in-cs}}$ apply(force)
using $z_{\text{notinbs}}$ apply(simp)
using $notzbeforez$ by auto

show $\text{?case}$ apply(simp only: $ex_{\text{nat less}}_{eq}$)
apply(rule bexI[where $x=k$])
using $1$ $zcsk$ apply(simp)
using $ki$ by simp

qed
lemma staysuntouched:
assumes d[simp]: distinct (fst S)
and x: x ∈ set (fst S)
and y: y ∈ set (fst S)
s shows set qs ⊆ set (fst S) ⇒ x ≠ set qs ⇒ y ≠ set qs
implies x < y in fst (config' (rTS [])) S qs = x < y in fst S
proof (induct qs rule: rev_induct)
case (snoc q qs)
have x < y in fst (config' (rTS []) S (qs @ [q])) =
   x < y in fst (config' (rTS []) S qs)
   apply (simp add: config'_snoc Step_def split_def step_def rTS_def nopaid)
   apply (rule xy relativorder mtf2)
   using snoc by (simp_all: x y)
also have ... = x < y in fst S
   apply (rule snoc)
   using snoc by simp_all
finally show ?case.
qed simp

lemma staysuntouched':
assumes d[simp]: distinct init
and x: x ∈ set init
and y: y ∈ set init
and set qs ⊆ set init
and x ≠ set qs and y ≠ set qs
shows x < y in fst (config (rTS [])) init qs = x < y in init
proof –
let ?S = (init, fst (rTS [])) init
have x < y in fst (config' (rTS []) ?S qs) = x < y in fst ?S
   apply (rule staysuntouched)
   using assms by (simp_all)
then show ?thesis by simp
qed

lemma projEmpty: Lxy qs S = [] ⇒ x ∈ S ⇒ x ≠ set qs
unfolding Lxy_def by (metis filter_empty_conv)

lemma Lxy_index_mono:
assumes x∈S y∈S
and index xs x < index xs y
and index xs y < length xs
and x≠y
shows index (Lxy xs S) x < index (Lxy xs S) y

proof –
from assms have ij: index xs x < index xs y
  and xinxs: index xs x < length xs
  and yinxs: index xs y < length xs by auto
then have inset: \( x \in \text{set xs} \) \( y \in \text{set xs} \) using index_less_size_conv by fast+
from xinxs obtain a as where \( \text{dec1: } a = \text{take (index xs x)} \) \( \text{xs and } as = \text{drop (Suc (index xs x))}\) \( \text{xs} \)
  and length_a: length a = index xs x and length_as: length as = length xs − index xs x − 1
  using id_take_nth_drop by fastforce
have index xs y ≥ length (a @ [xs!index xs x]) using length_a ij by auto
then have \( ((a @ [xs!index xs x]) @ as)! index xs y = as! (index xs y − index xs x)! xs) \( \text{using nth_append\[where } xs=a @ [xs!index xs x]\] \( \text{and } y\text{=}as\]\)
  by(simp)
then have xsj: xs! index xs y = as! (index xs y − index xs x − 1) using dec1 length_a by auto
have las: (index xs y − index xs x − 1) < length as using length_as yinxs
  ij by simp
obtain b c where \( \text{dec2: } b = \text{take (index xs y − index xs x − 1)} \) \( \text{as } c=\text{drop (Suc (index xs y − index xs x − 1))}\) \( \text{as} \)
  and length_b: length b = index xs y − index xs x − 1 using id_take_nth_drop[OF las] xsj by force

have xs_dec: a @ [xs!index xs x] @ b @ [xs!index xs y] @ c = xs using dec1 dec2 by auto

then have \( Lxy \) xs S = \( Lxy \) a @ [xs!index xs x] @ b @ [xs!index xs y] @ c S
  by(simp add: xs_dec)
also have \( \ldots = Lxy a S @ Lxy x S @ Lxy b S @ Lxy y S @ Lxy c S \)
  by(simp add: Lxy_append Lxy_def assms inset)
finally have gr: \( Lxy \) xs S = \( Lxy \) a S @ \( Lxy \) x S @ \( Lxy \) b S @ \( Lxy \) y S @ \( Lxy \) c S
  using assms by(simp add: Lxy_def)

have y \notin set (take (index xs x) xs)
  apply(rule index_take) using assms by simp
then have y \notin set (Lxy (take (index xs x) xs) S )
  apply(subst Lxy_set_filter) by blast
with a have ynot: y \notin set (Lxy a S) by simp
have index (Lxy xs S) y =

300
index \((Lxy\ a\ S \ @\ [x] \ @\ Lxy\ b\ S \ @\ [y] \ @\ Lxy\ c\ S)\ y\)
by(simp add: gr)
also have \(\ldots\ \geq\ length\ (Lxy\ a\ S) + 1\)
using assms(5) ynot by(simp add: index_append)
finally have 1: index \((Lxy\ xs\ S)\ y \geq\ length\ (Lxy\ a\ S) + 1\).

have index \((Lxy\ xs\ S)\ x = index\ (Lxy\ a\ S \ @\ [x] \ @\ Lxy\ b\ S \ @\ [y] \ @\ Lxy\ c\ S)\ x\)
by (simp add: gr)
also have \(\ldots\ \leq\ length\ (Lxy\ a\ S)\)
apply(simp add: index_append)
apply(subst index_less_size_conv[symmetric]) by simp
finally have 2: index \((Lxy\ xs\ S)\ x \leq\ length\ (Lxy\ a\ S)\).

from 1 2 show \(?\thesis\) by linarith

qed

lemma proj_Con:
assumes filterd_cons: \(Lxy\ qs\ S = a\#as\)
and a_filter: \(a \in S\)
obtains pre suf where \(qs = pre \# [a] \# suf\) and \(\forall x. x \in S \implies x \notin set\ pre\)
and \(Lxy\ suf\ S = as\)

proof –
have set \((Lxy\ qs\ S)\subseteq set\ qs\) using Lxy_set_filter by fast
with filterd_cons have a_inq: \(a \in set\ qs\) by simp
then have index qs a < length qs by(simp)

\{ fix e
assume eS: e\in S
assume e\neq a
have index qs a \leq index qs e
proof (rule ccontr)
assume \(\neg\ index\ qs\ a \leq index\ qs\ e\)
then have 1: index qs e < index qs a by simp
have 0: index \((Lxy\ qs\ S)\ a = 0\) unfolding filterd_cons by simp
have 2: index \((Lxy\ qs\ S)\ e < index\ (Lxy\ qs\ S)\ a\)
apply(rule Lxy_index mono)
by(fact)+
from 0 2 show False by linarith
qed
\}
note atfront=\this

let \(?lastInd=index\ qs\ a\)
have \( qs = \text{take } \# \text{lastInd } qs \) \& \( \# \text{Suc } \text{lastInd} \) \( qs \)
   apply\( \text{rule id}\_\text{take\_nth\_drop} \)
   using \( a\_\text{inq} \) by simp
also have \( \ldots = \text{take } \# \text{lastInd } qs \) \& \( \# \text{Suc } \text{lastInd} \) \( qs \)
   using \( a\_\text{inq} \) by simp
finally have \( \text{split}: \) \( qs = \text{take } \# \text{lastInd } qs \) \& \( \# \text{Suc } \text{lastInd} \) \( qs \).

have \( \text{nothingin}: \ldots \)
   apply\( \text{rule index\_take} \)
   apply\( \text{case\_tac } a=\_s \)
   apply\( \text{simp} \)
   by \( \text{(rule atfront) simp\_all} \)
then have \( \text{set} \) \( \text{Lxy} \) \( \text{take } \# \text{lastInd} \) \( qs \) \( S \) \( = \) \( \{ \} \)
   apply\( \text{subst Lxy\_set\_filter} \) by blast
then have \( \text{emptyPre} \) \( \text{Lxy} \) \( \text{take } \# \text{lastInd} \) \( qs \) \( S \)
   by \( \text{(simp add: Lxy\_append Lxy\_def)} \)
also have \( \ldots = a\# \text{Lxy} \) \( \text{drop } \# \text{Suc } \text{lastInd} \) \( qs \) \( S \)
   unfolding \( \text{emptyPre} \) by \( \text{(simp add: Lxy\_def a\_filter)} \)
finally have \( \text{suf} \) \( \text{Lxy} \) \( \text{drop } \# \text{Suc } \text{lastInd} \) \( qs \) \( S \) \( = \) \( \text{as} \) by simp

from \( \text{split nothingin suf show } \) \( ?\text{thesis} \).
qed

lemma \( \text{Lxy\_rev} \): \( \text{rev} \) \( \text{Lxy} \) \( qs \) \( S \) \( = \) \( \text{Lxy} \) \( \text{rev} \) \( qs \) \( S \)
apply\( \text{induct } qs \)
by\( \text{(simp\_all add: Lxy\_def)} \)

lemma \( \text{proj\_Snoc} \):
  assumes \( \text{filtered\_cons} : \text{Lxy} \) \( qs \) \( S \) \( = \) \( \text{as}\@[a] \)
  and \( a\_\text{filter} : a \in S \)
  obtains \( \text{pre suf where } \) \( qs = \) \( \text{pre} \) \( \& \) \( \# \) \( [a] \) \( \& \) \( \text{suf} \)
  \( \wedge x. \) \( x \in S \) \( \Rightarrow x \notin \) \( \text{set suf} \)
  and \( \text{Lxy pre S = as} \)
proof –
  have \( \text{Lxy} \) \( \text{rev} \) \( qs \) \( S \) \( = \) \( \text{rev} \) \( \text{Lxy} \) \( qs \) \( S \) by \( \text{(simp add: Lxy\_rev)} \)
also have \( \ldots = a\# \text{(rev as)} \) unfolding \( \text{filtered\_cons} \) by simp

302
finally have $L_{xy}$ (rev $qs$) $S = a \# (\text{rev as})$.

with $a_{\text{filter}}$

obtain pre' suf' where 1: rev $qs = \text{pre'} \@ [a] @ \text{suf'}$
and 2: $\forall x. x \in S \implies x \notin \text{set pre'}$
and 3: $L_{xy} \text{suf'} S = \text{rev as}$

using $\text{proj Cons}$ by metis

have $qs = \text{rev (rev qs)}$ by simp
also have $\ldots = \text{rev suf'} @ [a] @ \text{rev pre'}$ using 1 by simp
finally have a1: $qs = \text{rev suf'} @ [a] @ \text{rev pre'}$.

have $L_{xy} (\text{rev suf'}) S = \text{rev (Lxy suf'} S)$ by (simp add: Lxy_rev)
also have $\ldots = \text{as using 3 by simp}$
finally have a3: $L_{xy} (\text{rev suf'}) S = \text{as}$.

have a2: $\forall x. x \in S \implies x \notin \text{set (rev pre')}$ using 2 by simp

from a1 a2 a3 show ?thesis ..

qed

lemma sndTSconfig': snd (config' (rTS initH) (init,[])) $qs = \text{rev qs} @ []$

apply (induct $qs$ rule: rev_induct)
apply (simp add: rTS_def)
by (simp add: split_def TS_step_d_def config'_snoc Step_def rTS_def)

lemma projxx:
fixes $e$ $a$ $bs$
assumes axy: $a \in \{x,y\}$
assumes ane: $a \neq e$
assumes exy: $e \in \{x,y\}$
assumes add: $f \in \{[],[e]\}$
assumes bsaxy: set ($bs @ [a] @ f$) $\subseteq \{x,y\}$
assumes $Lxy$initxy: $Lxy$ init $\{x,y\}$ $\in \{[x,y],[y,x]\}$
shows $a < e$ in fst (configp (rTS [])) (Lxy init $\{x,y\}$) ($bs @ [a] @ f$ $@$ $[a]$))

proof –
have $a\text{exy} : \{a,e\} = \{x,y\}$ using exy axy ane by blast

let $?h = \text{snd (Partial_Cost_Model.config'} (\lambda s. []), \text{TS}_\text{step}_d)$
(Lxy init $\{x, y\}, []$) ($bs @ a \# f$))

have history: $?h = (\text{rev f}@a#(\text{rev bs})$

using sndTSdet[of length ($bs@a#f$) $bs@a#f$, unfolded rTS_def] by (simp)

{ fix $xs$ $s$ }
assume \( sinit: s:\{[a,e],[e,a]\} \)
assume \( set \ xs \subseteq \{a,e\} \)
then have \( \text{fst} (\text{config}' (\lambda s. []), TS\_step\_d) (s, []) \in \{[a,e],[e,a]\} \)
   apply (induct \(xs\) rule: rev_induct)
   using \( sinit \) apply(simp)
   apply(subst config'_append2)
   apply(simp only: Step_def config'.simps Let_def split_def fst_conv)
   apply(rule stepxy) by simp_all
\}

note staysae = this
have \( \text{opt: } \text{fst} (\text{config}' (\lambda s. []), TS\_step\_d) \)
   \( (Lxy \text{ init \{x, y\}, []}) (bs \@ [a] \@ f)) \in \{[a,e],[e,a]\} \)
   apply(rule staysae)
   using Lxyinitxy exy axy ane apply fast
   unfolding aexy by (fact bsaxy)

have \( \text{contr: } (\forall x. 0 < (if e = x then 0 else \text{index } [a] x + 1)) = \text{False} \)
proof (rule ccontr; goal_cases)
  case 1
  then have \( \text{\(\forall x. 0 < (if e = x then 0 else \text{index } [a] x + 1)\) by simp} \)
  then have \( 0 < (if e = e then 0 else \text{index } [a] e + 1) \) by blast
  then have \( 0 < 0 \) by simp
  then show \( \text{False by auto} \)
qed

show \( a < e \) in \( \text{fst} (\text{config}_p (rTS [])) (Lxy \text{ init \{x, y\}}) ((bs \@ [a] \@ f) @ [a])) \)
   apply(subst config_append)
   apply(simp add: rTS_def Step_def split_def)
   apply(subst TS_step_d_def)
   apply(simp only: history)
   using \( \text{opt ane add} \)
   apply(auto simp: step_def)
   apply(simp add: before_in_def)
   apply(simp add: before_in_def)
   apply(simp add: before_in_def contr)
   apply(simp add: mtf2_def swap_def before_in_def)
   apply(auto simp add: before_in_def contr)
   apply (metis One_nat_def add_is_1 count_list.simps(1) le_Suc_eq)
   by(simp add: mtf2_def swap_def)
qed
lemma oneposs:
  assumes set xs = {x,y}
  assumes x≠y
  assumes distinct xs
  assumes True: x<y in xs
  shows xs = [x,y]
proof –
  from assms have len2: length xs = 2 using distinct_card[OF assms(3)]
  by fastforce
  from True have index xs x < index xs y index xs y < length xs unfolding
  before_in_def using assms
  by simp_all
  then have f: index xs x = 0 ∧ index xs y = 1 using len2 by linarith
  have xs = take 1 xs @ xs!1 # drop (Suc 1) xs
    apply(rule id_take_nth_drop) using len2 by simp
  also have ... = take 1 xs @ [ys!1] using len2 by simp
  also have take 1 xs = take 0 (take 1 xs) @ (take 1 xs)!0 # drop (Suc 0)
    (take 1 xs)
    apply(rule id_take_nth_drop) using len2 by simp
  also have ... = [x,y] using f by simp
  finally have ... = [x!(index xs x), ys!index xs y] using f by simp
  also have ... = [x,y] using assms by(simp)
  finally show xs = [x,y].
qed

lemma twoposs:
  assumes set xs = {x,y}
  assumes x≠y
  assumes distinct xs
  shows xs ∈ {[x,y], [y,x]}
proof (cases x<y in xs)
  case True
  from assms have len2: length xs = 2 using distinct_card[OF assms(3)]
  by fastforce
  from True have index xs x < index xs y index xs y < length xs unfolding
  before_in_def using assms
  by simp_all
  then have f: index xs x = 0 ∧ index xs y = 1 using len2 by linarith
  have xs = take 1 xs @ xs!1 # drop (Suc 1) xs
    apply(rule id_take_nth_drop) using len2 by simp
  also have ... = take 1 xs @ [ys!1] using len2 by simp
  also have take 1 xs = take 0 (take 1 xs) @ (take 1 xs)!0 # drop (Suc 0)
    (take 1 xs)

apply(rule id_take_nth_drop) using len2 by simp
also have ... = [xs!0] by(simp)
finally have xs = [xs!0, xs!1] by simp
also have ... = [xs!(index xs x), xs!index xs y] using f by simp
also have ... = [x,y] using assms by(simp)
finally have xs = [x,y].
then show ?thesis by simp
next
case False
from assms have len2: length xs = 2 using distinct_card[OF assms(3)]
by fastforce
from False have y<x in xs using not_before_in assms(1,2) by fastforce
then have index xs y < index xs x index xs x < length xs unfolding before_in_def using assms
by simp_all
then have f: index xs y = 0 ∧ index xs x = 1 using len2 by linarith
have xs = take 1 xs @ xs!1 # drop (Suc 1) xs
apply(rule id_take_nth_drop) using len2 by simp
also have ... = take 1 xs @ [xs!1] using len2 by simp
also have take 1 xs = take 0 (take 1 xs) @ (take 1 xs)!0 # drop (Suc 0) (take 1 xs)
apply(rule id_take_nth_drop) using len2 by simp
also have ... = [xs!0] by(simp)
finally have xs = [xs!0, xs!1] by simp
also have ... = [xs!(index xs y), xs!index xs x] using f by simp
also have ... = [y,x] using assms by(simp)
finally have xs = [y,x].
then show ?thesis by simp
qed

lemma TS_pairwise': assumes qs ∈ {xs. set xs ⊆ set init}
(x, y) ∈ {(x, y), x ∈ set init ∧ y ∈ set init ∧ x ≠ y}
x ≠ y distinct init
shows Pbefore_in x y (embed (rTS [])) qs init =
Pbefore_in x y (embed (rTS [])) (Lxy qs {x, y}) (Lxy init {x, y})

proof –
from assms have xyininit: {x, y} ⊆ set init
and qsininit: set qs ⊆ set init by auto
note dinit=assms(4)
from assms have x≠y by simp
have Lxyinxy: Lxy init {x, y} ∈ {[x, y], [y, x]}
apply(rule twooposs)
apply(subst Lxy_set_filter) using xyininit apply fast
using xny Lxy_distinct[OF dinit] by simp_all
have \( lq.s \): set \( Lxy qs \{x, y\} \subseteq \{x, y\} \) by (simp add: Lxy_set_filter)

let ?pH = snd (config \_p (rTS []) \( Lxy init \{x, y\}\)) (Lxy qs \{x, y\}))
have \(?pH =\) snd (TSdet \( Lxy init \{x, y\}\) [] (Lxy qs \{x, y\})) (length (Lxy qs \{x, y\})))
  by (simp)
also have \( \ldots = \) rev (take (length (Lxy qs \{x, y\}))) (Lxy qs \{x, y\})) []
  apply (rule sndTSdet) by simp
finally have pH: \(?pH = rev \( Lxy qs \{x, y\}\)) by simp

let ?pQs = \( Lxy qs \{x, y\}\)
have \(?pH = rev \( Lxy qs \{x, y\}\)) by simp

proof (cases ?pQs rule: rev_cases)
  case Nil
    then have xqs: \( x \notin set qs \) and yqs: \( y \notin set qs \) by (simp_all add: projEmpty)
  also have \( \ldots = x < y \) in init (config \_p (rTS []) \( Lxy init \{x, y\}\)) (Lxy qs \{x, y\}))
    unfolding Nil apply (simp) apply (rule Lxy_mono) using xyininit
dinit by (simp_all)
  finally show \(?thesis\).
next
  case (snoc as a)
  then have a\( \in\)set \( Lxy qs \{x, y\}\) by (simp)
  then have axy: \( a \in \{x, y\}\) by (simp add: Lxy_set_filter)
  with xyininit have ainit: \( a \in set init \) by auto
  note a = snoc
  from a axy obtain pre suf where qs: qs = pre \@ [a] \@ suf
    and nosuf: \( \forall e. e \in \{x, y\} \rightarrow e \notin set suf \)
    and pre: Lxy pre \{x, y\} = as
    using proj_Snoc by metis
  show \(?thesis\).
  proof (cases as rule: rev_cases)
    case Nil
    from pre Nil have xqs: \( x \notin set pre \) and yqs: \( y \notin set pre \) by (simp_all
    add: projEmpty)
    from xqs yqs axy have a \( \notin\) set pre by blast
then have noocc: index (rev pre) a = length (rev pre) by simp
have \( x < y \) in \( \text{fst} (\text{config}_p (rTS [])) \) init qs
  \( = x < y \) in \( \text{fst} (\text{config}_p (rTS [])) \) init ((pre @ [a]) @ suf)) by(simp add: qs)
also have \( \ldots = x < y \) in \( \text{fst} (\text{config}_p (rTS [])) \) init (pre @ [a])
  apply(subst config_append)
apply(rule staysuntouched) using assms xqs yqs qs nosuf by(simp_all)
also have \( \ldots = x < y \) in \( \text{fst} (\text{config}_p (rTS [])) \) init pre
  apply(subst config_append)
apply(simp add: rTS_def Step_def split_def)
apply(simp only: TS_step_d_def)
apply(simp only: sndTSconfig[unfolded rTS_def])
by(simp add: noocc step_def)
also have \( \ldots = x < y \) in init
  apply(rule staysuntouched') using assms xqs yqs qs
  by(simp_all)
also have \( \ldots = x < y \) in \( \text{fst} (\text{config}_p (rTS [])) \) (Lxy init {x, y})
  (Lxy qs {x, y})
unfolding a Nil apply(simp add: Step_def split_def rTS_def TS_step_d_def step_def)
  apply(rule Lxy_mono) using xyininit dinit by(simp_all)
finally show \(?thesis\).
next
  case (snoc bs b)
  note \( b = \text{this} \)
with a have \( b \in \text{set} (Lxy qs \{x, y\}) \) by (simp)
then have \( bxy: b \in \{x, y\} \) by(simp add: Lxy_set_filter)
with xyininit have \( \text{binit}: b \in \text{set} \text{init} \) by auto
from \( b \) pre have \( Lxy \text{pre} \{x, y\} = b @ [b] \) by simp
with bxy obtain \( \text{pre2 suf2} \) where \( b: \text{pre} = \text{pre2} @ [b] \) @ suf2
  and \( \text{nosuf2}: \\forall e. e \in \{x, y\} \implies e \notin \text{set} \text{suf2} \)
  and \( \text{pre2}: Lxy \text{pre2} \{x, y\} = b \)
    using proj_Snoc by metis
from \( bs \) qs have \( \text{qs2}: \text{qs} = \text{pre2} @ [b] \) @ suf2 @ [a] @ suf by simp
show \(?thesis\)
proof (cases \( a = b \))
  case True
  note \( ab = \text{this} \)
  let \( ?qs = (\text{pre2} @ [a] \) @ suf2 @ [a]) @ suf
  \{ fix \ e 

308
assume $a \neq e$
assume $e \in \{x, y\}$
have $a < e$ in $\text{fst} \ (\text{config}_p \ (rTS \ [])) \ \text{init} \ qs$

$= a < e$ in $\text{fst} \ (\text{config}_p \ (rTS \ [])) \ \text{init} \ ?qs$ using $\text{True} \ \text{qs2}$
by$(\text{simp})$
also have ... $= a < e$ in $\text{fst} \ (\text{config}_p \ (rTS \ [])) \ \text{init} \ (\text{pre2} @ [a] @ \text{suf2} @ [a])$
apply(\text{rule staysuntouched}) using $\text{assms} \ \text{qs} \ \text{nosuf} \ \text{apply}(\text{simp\_all})$
using $\text{exy xyininit} \ \text{apply} \ \text{fast}$
using $\text{nosuf \ axy} \ \text{apply}(\text{simp})$
using $\text{nosuf \ exy} \ \text{by} \ \text{simp}$
also have ...
apply$(\text{simp})$
apply$(\text{rule \ twotox [unfolded \ s\_TS\_def, \ simplified]})$
using $\text{nosuf2 \ exy} \ \text{apply}(\text{simp})$
using $\text{assms} \ \text{apply}(\text{simp\_all})$
using $\text{axy \ xyininit} \ \text{apply} \ \text{fast}$
using $\text{axy \ xyininit} \ \text{apply} \ \text{fast}$
using $\text{nosuf2 \ axy} \ \text{apply}(\text{simp})$
using $\text{ane} \ \text{by} \ \text{simp}$
finally have $a < e$ in $\text{fst} \ (\text{config}_p \ (rTS \ [])) \ \text{init} \ \text{qs}$ by $(\text{simp})$

} note $\text{full}=$this

have set $(\text{bs} @ [a]) \subseteq \text{set} \ (\text{Lxy} \ \text{qs} \ \{x, y\})$ using $a \ b$ by $(\text{auto})$
also have ... $= \{x, y\} \cap \text{set} \ \text{qs}$ by $(\text{rule \ Lxy\_set\_filter})$
also have ... $\subseteq \{x, y\}$ by $(\text{simp})$
finally have $\text{bsaxy: set} \ (\text{bs} @ [a]) \subseteq \{x, y\}$.

with $\text{xny}$ show ?thesis
proof(cases $x=a$)
  case True
  have 1: $a < y$ in $\text{fst} \ (\text{config}_p \ (rTS \ [])) \ \text{init} \ \text{qs}$
   apply$(\text{rule \ full})$
   using $\text{True \ xny} \ \text{apply} \ \text{blast}$
   by $(\text{simp})$

  have $a < y$ in $\text{fst} \ (\text{config}_p \ (rTS \ [])) \ (\text{Lxy} \ \text{init} \ \{x, y\}) \ (\text{Lxy} \ \text{qs} \ \{x, y\})$
   $= a < y$ in $\text{fst} \ (\text{config}_p \ (rTS \ [])) \ (\text{Lxy} \ \text{init} \ \{x, y\}) \ ((\text{bs} @ [a] @ [])) @ [a])$
   using $a \ b \ b$ by $(\text{simp})$
also have ...
apply (rule projxx[where bs=bs and f=[]])
  using True apply blast
  using a b True ab xny Lxyinitxy bsaxy by(simp_all)
finally show \?thesis using True 1 by simp
next
case False
with axy have ay: a=y by blast
have 1: a < x in fst (config_p (rTS [])) init qs
  apply (rule full)
  using False xny apply blast
  by simp
have a < x in fst (config_p (rTS [])) (Lxy init {x, y}) (Lxy qs {x, y}))
  = a < x in fst (config_p (rTS [])) (Lxy init {x, y}) ((bs @ [a] @ [] @ [a]))
  using a b ab by simp
also have ...
  apply (rule projxx[where bs=bs and f=[]])
  using True axy apply blast
  using a b True ab xny Lxyinitxy ay bsaxy by(simp_all)
finally have 2: a < x in fst (config_p (rTS [])) (Lxy init {x, y}) (Lxy qs {x, y})

have x < y in fst (config_p (rTS [])) init qs =
  (¬ y < x in fst (config_p (rTS [])) init qs))
  apply (subst not_before_in)
  using assms by(simp_all)
also have ... = False using 1 ay by simp
also have ... = (¬ y < x in fst (config_p (rTS [])) (Lxy init {x, y}) (Lxy qs {x, y}))
  using 2 ay by simp
also have ... = x < y in fst (config_p (rTS [])) (Lxy init {x, y}) (Lxy qs {x, y})
  apply (subst not_before_in)
  using assms by(simp_all add: Lxy_set_filter)
finally show \?thesis .
qed
next
case False
note ab=this

show \?thesis
proof (cases bs rule: rev_cases)
case Nil

310
with a b have \( Lxy\) qs \( \{x, y\} = [b,a] \) by \( \text{simp} \)
from pre2 Nil have \( xqs: x \notin \text{set} \) pre2 and \( yqs: y \notin \text{set} \) pre2 by (\( \text{simp\ all add: projEmpty} \))
from xqs yqs bxy have \( b \notin \text{set} \) pre2 by blast
then have noocc2: index (rev pre2) \( b = \text{length} \) (rev pre2) by \( \text{simp} \)
from axy nosuf2 have \( a \notin \text{set} \) suf2 by blast
with xqs yqs axy False have \( a \notin \text{set} \) ((pre2 @ [b] @ [a]) @ [a]) @ [a]) by (\( \text{simp add: qs2} \))
also have \( \ldots = x < y \) in fst (config (rTS [])) init qs by (\( \text{simp add:} \) config append)
apply (rule staysuntouched) using assms xqs yqs qs nosuf2 by (\( \text{simp\ all} \))
also have \( \ldots = x < y \) in fst (config (rTS [])) init ((pre2 @ [b]) @ suf2) by (\( \text{simp add:} \) config append)
apply (rule staysuntouched) using assms xqs yqs qs2 nosuf2 by (\( \text{simp\ all} \))
also have \( \ldots = x < y \) in fst (config (rTS [])) init ((pre2 @ [b]) @ suf2) by (\( \text{simp add:} \) config append)
apply (rule staysuntouched) using assms xqs yqs qs2 nosuf2 by (\( \text{simp\ all} \))
also have \( \ldots = x < y \) in init by (\( \text{simp add:} \) config append)
also have \( \ldots = x < y \) in fst (config (rTS [])) init (pre2) by (\( \text{simp add:} \) config append)
also have \( \ldots = x < y \) in fst (config (rTS [])) init (pre2) by (\( \text{simp add:} \) config append)
also have \( \ldots = x < y \) in init by (\( \text{simp add:} \) config append)
also have \( \ldots = x < y \) in fst (config (rTS [])) (Lxy init \( \{x, y\} \)) unfolding a b Nil using False apply (simp add: Step_def split_def rTS_def TS_step_d_def step_def)
apply (rule Lxy_mono) using xyininit dinit by (simp_all)

finally show ?thesis.

next
case (snoc cs c)

note c = this

with a b have c ∈ set (Lxy qs {x, y}) by (simp)

then have cxy: c ∈ {x, y} by (simp add: Lxy_set_filter)

from c pre2 have Lxy pre2 {x, y} = cs @ [c] by simp

with cxy obtain pre3 suf3 where cs: pre2 = pre3 @ [c] @ suf3

and nosuf3: ∀ e. e ∈ {x, y} ⇒ e /∈ set suf3

and pre3: Lxy pre3 {x, y} = cs

using proj_Snoc by metis

let ?qs = pre3 @ [c] @ suf3 @ [b] @ suf2 @ [a] @ suf

from bs cs qs have qs2: qs = ?qs by simp

show ?thesis

proof (cases c = a)

case True

note ca = this

have a < b in fst (config_p (rTS [])) init qs

= a < b in fst (config_p (rTS [])) init ((pre3 @ a # (suf3 @ [b]

@ suf2) @ [a]) @ suf))

using qs2 True by simp

also have ... = a < b in fst (config_p (rTS [])) init (pre3 @ a #

(suf3 @ [b] @ suf2) @ [a]))

apply (subst config_append)

apply (rule staysuntouched) using assms qs nosuf apply (simp_all)

using bxy xyininit apply (fast)

using nosuf axy bxy by (simp_all)

also have ...

apply (rule twotox [unfolded s_TS_def, simplified])

using nosuf2 nosuf3 bxy apply (simp add: count_append)

using assms apply (simp_all)

using axy xyininit apply (fast)

using bxy xyininit apply (fast)

using ab nosuf2 nosuf3 axy apply (simp)

using ab by simp

finally have full: a < b in fst (config_p (rTS [])) init qs by simp

have set (cs @ [a] @ [b]) ⊆ set (Lxy qs {x, y}) using a b c by auto
also have \ldots \subseteq \{x, y\} \cap \text{set } \text{qs} \text{ by (rule } Lxy\text{-set\_filter)}
also have \ldots \subseteq \{x, y\} \text{ by simp}
finally have \text{csabxy: set } (\text{cs @ [a] @ [b]}) \subseteq \{x, y\}.

with \text{xny show } ?\text{thesis}
proof (cases \text{x}=\text{a})
case True
with \text{xny ab bxy have } bisy: \text{b}=\text{y} \text{ by blast}
have 1: \text{x} < \text{y} in fst (\text{config} p (rTS []) init \text{qs})
  using full True bisy by simp

  have \text{a} < \text{y} in fst (\text{config} p (rTS []) (Lxy init \{x, y\}) (Lxy \text{qs}
  \{x, y\}))
    = \text{a} < \text{y} in fst (\text{config} p (rTS []) (Lxy init \{x, y\}) ((\text{cs @ [a]}
    @ [b] @ [a]))
    using a b c ca ab by simp
also have \ldots
  apply (rule projxx)
    using True apply blast
  using a b True ab \text{xny Lxyinitxy csabxy by simp_all)
finally show \text{thesis using 1 True by simp}
next
case False
with \text{axy have } ay: \text{a}=\text{y} \text{ by blast}
with \text{xny ab bxy have } bisx: \text{b}=\text{x} \text{ by blast}
have 1: \text{y} < \text{x} in fst (\text{config} p (rTS []) init \text{qs})
  using full ay bisx by simp

  have \text{a} < \text{x} in fst (\text{config} p (rTS []) (Lxy init \{x, y\}) (Lxy \text{qs}
  \{x, y\}))
    = \text{a} < \text{x} in fst (\text{config} p (rTS []) (Lxy init \{x, y\}) ((\text{cs @ [a]}
    @ [b] @ [a]))
    using a b c ca ab by simp
also have \ldots
  apply (rule projxx)
    using a b True ab \text{xny Lxyinitxy csabxy False by simp_all)
finally have 2: \text{a} < \text{x} in fst (\text{config} p (rTS []) (Lxy init \{x, y\})
  (Lxy \text{qs} \{x, y\})).

  have \text{x} < \text{y} in fst (\text{config} p (rTS []) init \text{qs}) =
    (\neg \text{y} < \text{x} in fst (\text{config} p (rTS []) init \text{qs})
  apply (subst not_before_in)
    using assms by (simp_all)
also have \ldots = False using 1 ay by simp
also have \( \neg y < x \) in \( \text{fst} \ (\text{config}_p \ (r\text{TS} \ [])) \ (Lxy \ \text{init} \ \{x, y\}) \)
using 2 \( ay \) by simp
also have \( \ldots = x < y \) in \( \text{fst} \ (\text{config}_p \ (r\text{TS} \ [])) \ (Lxy \ \text{init} \ \{x, y\}) \)
\( (Lxy \ \text{qs} \ \{x, y\}) \)
apply (\( \text{subst not\_before\_in} \))
using assms by (simp\_all add: Lxy\_set\_filter)
finally show \( \ ?\text{thesis} \).
qed

next
case False
then have \( cb \): \( c = b \) using \( bxy \ cxy \ axy \ ab \) by blast

let \( ?cs = \text{suf2} @ [a] @ \text{suf} \)
let \( ?i = \text{index} \ ?cs \ a \)

have \( \text{aed:} \ (\forall j < \text{index} \ (\text{suf2} @ a \ # \ \text{suf}) \ a. (\text{suf2} @ a \ # \ \text{suf}) ! j \neq a) \)
by (metis add\_right\_neutral axy index\_Cons index\_append nosuf2 nth\_append nth\_mem)

have \( ?i < \text{length} \ ?cs \)
\( \rightarrow (\forall j < ?i. \ ?cs ! j \neq ?cs ! ?i) \rightarrow ?cs ! ?i \neq b \)
\( \rightarrow ?cs ! ?i \notin \text{set suf3} \)
\( \rightarrow b < ?cs ! ?i \text{ in s\_TS init [] qs (length (pre3 @ [b] @ suf3 @ [b]) + ?i + 1)} \)
apply (rule case\_xxy)
using \( cb \ qsf2 \ apply(\text{simp}) \)
using \( bxy \ ab \ nosuf2 \ nosuf \ apply(\text{simp}) \)
using \( bs \ qsf \ qsin\_init \ apply(\text{simp}) \)
using \( bxy \ xy\_in\_init \ apply(\text{blast}) \)
apply (fact)
using \( nosuf3 \ bxy \ apply(\text{simp}) \)
using \( cs \ bs \ qs \ qsin\_init \ by(\text{simp}\_all) \)

then have inner: \( b < a \text{ in s\_TS init [] qs (length (pre3 @ [b] @ suf3 @ [b]) + ?i + 1)} \)
using \( ab \ nosuf3 \ axy \ bxy \ aed \)
by (simp)
let \( ?n = (\text{length (pre3 @ [b] @ suf3 @ [b]) + ?i + 1}) \)
let \( ?inner = (\text{config}_p \ (r\text{TS} \ []) \ \text{init} \ (\text{take} \ (\text{length (pre3 @ [b] @ suf3 @ [b]) + ?i + 1}) \ ?qs)) \)
have \( b < a \) in \( \text{fst} (\text{config}_p (rTS \ [])) \) \( \text{init} \) \( qs \)

\[
= b < a \text{ in } \text{fst} (\text{config}_p (rTS \ [])) \text{ init } \langle \text{take } ?n \ ?qs \ @ \text{drop } ?n \ ?qs \rangle
\]

using \( qs2 \) by \( \text{simp} \)

also have \( \ldots = b < a \) in \( \text{fst} (\text{config}' (rTS \ [])) \) \( \text{?inner} \) \( \text{suf} \)

apply(\( \text{simp only: config_append} \) \( \text{drop_append} \))

using \( \text{nosuf2} \) \( \text{axy} \) by(\( \text{simp add: index} \) \( \text{append} \) \( \text{config_append} \))

also have \( \ldots = b < a \) in \( \text{fst} ?\text{inner} \)

apply(\( \text{rule staysuntouched} \)) \( \text{using} \) \( \text{assms} \) \( \text{bxy} \) \( \text{xyininit} \) \( \text{qs} \) \( \text{nosuf} \)

apply(\( \text{simp all} \)) \( \text{using} \) \( \text{bxy} \) \( \text{xyininit} \) \( \text{axy} \) \( \text{bxy} \) \( \text{ab} \) \( \text{csbxy} \) \( \text{Lxyinitxy} \)

also have \( \ldots = \text{True} \) using \( \text{inner} \) by(\( \text{simp add: s\_TS\_def} \) \( \text{qs2} \))

finally have \( \text{full}: b < a \) in \( \text{fst} (\text{config}_p (rTS \ [])) \) \( \text{init} \) \( qs \) by \( \text{simp} \)

have set \( (cs @ [b] @ []) \subseteq \{ x \text{, } y \} \) \( \text{using} \) \( \text{a} \) \( \text{b} \) \( \text{c} \) \( \text{by} \) auto

also have \( \ldots = \{ x \text{, } y \} \cap \text{set} \) \( \text{qs} \) by(\( \text{rule Lxy\_set\_filter} \))

also have \( \ldots \subseteq \{ x \text{, } y \} \) by \( \text{simp} \)

finally have \( \text{csbxy}: \text{set} \ (cs @ [b] @ []) \subseteq \{ x \text{, } y \} \).

have set \( (Lxy \text{ init} \ (\{ x \text{, } y \})) \) \( \subseteq \{ x \text{, } y \} \)

by(\( \text{rule Lxy\_set\_filter} \))

also have \( \ldots = \{ x \text{, } y \} \) using \( \text{xyininit} \) by fast

also have \( \ldots = \{ b \text{, } a \} \) using \( \text{axy bxy ab} \) by fast

finally have \( r: \text{set} \ (Lxy \text{ init} \ (\{ x \text{, } y \})) = \{ b \text{, } a \} \).

let \( \text{?confbef} = (\text{config}_p (rTS \ [])) \ (Lxy \text{ init} \ (\{ x \text{, } y \})) ((cs @ [b] @ [])) @ [b]) \)

have \( f1: b < a \) in \( \text{fst} \ ?\text{confbef} \)

apply(\( \text{rule projxx} \))

using \( \text{bxy ab axy a b c} \) \( \text{csbxy Lxyinitxy by(simp\_all)} \)

have \( 1: \text{fst} ?\text{confbef} = [b,a] \)

apply(\( \text{rule oneposs} \))

using \( \text{ab axy bxy xyininit Lxy\_distinct[OF dinit]} \) \( f1 \) \( \text{by(simp\_all)} \)

have \( 2: \text{snd} (\text{Partial\_Cost\_Model.config'}) \)

\( (\lambda s, [], \text{TS\_step}\_d) \)
\( (Lxy \text{ init} \ (\{ x \text{, } y \}), []) \)
\( (cs @ [b, b])) = [b,b](\text{rev cs}) \)

using \( \text{sndTSdet[of length} \ (cs @ [b, b]) \ (cs @ [b, b]), unfolded} \)

\( r\text{TS\_def} \) by(\( \text{simp} \))

have \( b < a \) in \( \text{fst} (\text{config}_p (rTS \ [])) \ (Lxy \text{ init} \ (\{ x \text{, } y \})) \ (Lxy \text{ qs} \ (\{ x, y \})) \)

\( = b < a \) in \( \text{fst} (\text{config}_p (rTS \ [])) \ (Lxy \text{ init} \ (\{ x, y \})) (((cs @ [b] @ [])) @ [b])@[a]) \)

\( 315 \)
using $a \ b \ c \ cb$ by(simp)
also have \ldots
apply(subst config_append)
using $1 \ 2 \ ab$ apply(simp add: step_def Step_def split_def rTS_def
$TS_{step_{d_def}}$
by(simp add: before_in_def)
finally have projected: $b < a$ in fst (config$_p$ (rTS [])) ($Lxy$ init 
{x, y}) ($Lxy$ qs {x, y})

have $1$: {x, y} = {a, b} using $ab$ axy bxy by fast
with $xny$ show ?thesis
proof(cases $x=a$
  case $True$
  with $1$ $xny$ have: $y=b$ by fast
  have $a < b$ in fst (config$_p$ (rTS [])) init qs =
  ($\neg \ b < a$ in fst (config$_p$ (rTS [])) init qs))
  apply(subst not_before_in)
  using binit ainit ab by(simp_all)
also have \ldots = $False$ using full by simp
  also have \ldots = ($\neg \ b < a$ in fst (config$_p$ (rTS [])) ($Lxy$ init 
{x, y}) ($Lxy$ qs {x, y})) ($Lxy$ qs {x, y}))
  using projected by simp
  also have \ldots = $a < b$ in fst (config$_p$ (rTS [])) ($Lxy$ init 
{x, y})
  ($Lxy$ qs {x, y}))
  apply(subst not_before_in)
  using binit ainit ab axy bxy by(simp_all add: $Lxy$ set_filter)
finally show ?thesis using $True$ y by simp
next
  case $False$
  with $1$ $xny$ have: $y=a \ x=b$ by fast+
  with full projected show ?thesis by fast
qed
qed
qed
qed
qed
show ?thesis unfolding $P$before_in_def
apply(subst config_embed)
apply (subst config_embed)
apply (simp) by (rule A)
qed

theorem TS_pairwise: pairwise (embed (rTS []))
apply (rule pairwise_property_lemma)
apply (rule TS_pairwise') by (simp_all add: rTS_def TS_step_d_def)

15.6 TS is 2-compet

lemma TS_compet': pairwise (embed (rTS [])) \implies
   \forall s0 \in \{\text{init}::(nat list). \text{distinct init} \land init \neq []\}. \exists b \geq 0. \forall qs \in \{x. \text{set} x \subseteq \text{set} s0\}. T_{p\_on\_rand} (\text{embed} (rTS [])) s0 qs \leq (2::real) * T_{p\_opt} s0 qs + b

unfolding rTS_def

proof (rule factoring lemma_withconstant, goal_cases)
  case 5
  show ?case
  proof (safe, goal_cases)
    case (1 init)
    note out = this
    show ?case
      apply (rule exI [where \(x = 2\)])
      apply (simp)
      proof (safe, goal_cases)
        case (1 qs a b)
        then have a: \(a \neq b\) by simp
        have twist: \(\{a, b\} = \{b, a\}\) by auto
        have b1: \(\text{set} (Lxy qs \{a, b\}) \subseteq \{a, b\}\) unfolding Lxy_def by auto
        with this [unfolded twist] have b2: \(\text{set} (Lxy qs \{b, a\}) \subseteq \{b, a\}\)
        by (auto)

        have set (Lxy init \{a, b\}) = \{a, b\} \cap (\text{set init}) apply (induct init)
        unfolding Lxy_def by (auto)
        with 1 have A: \(\text{set} (Lxy init \{a, b\}) = \{a, b\}\) by auto
        have finite \(\{a, b\}\) by auto
        from out have B: \(\text{distinct} (Lxy init \{a, b\})\) unfolding Lxy_def by auto

        have C: \(\text{length} (Lxy init \{a, b\}) = 2\)
        using distinct_card [OF B, unfolded A] using a by auto

        have \(\{xs. \text{set} xs = \{a, b\} \land \text{distinct} xs \land \text{length} xs = (2::nat)\}\)
          = \(\{[a, b], [b, a]\}\)
        apply (auto simp: a[symmetric])
proof (goal_cases)
  case (1 xs)
    from 1(4) obtain x xs' where r:xs=x#xs' by (metis Suc_length_conv add_2_eq_Suc' append_Nil length_append)
    with 1(4) have length xs' = 1 by auto
    then obtain y where s: [y] = xs' by (metis One_nat_def length_0_conv length_Suc_conv)
    from r s have t: [x,y] = xs by auto
    moreover from t 1(1) have x=b using doubleton_eq_iff
    ultimately show ?case by auto
    qed

with A B C have pos: (Lxy init {a, b}) = [a,b] 
  ∨ (Lxy init {a, b}) = [b,a] by auto

{  fix a::nat
  fix b::nat
  fix qs
  assume as: a ≠ b set qs ⊆ {a, b}
  have T_on_rand' (embed (rTS [])) (fst (embed (rTS []))) [a,b] 
  ≡ (λis. return_pmf ([a,b], is))) qs 
  = T_p-on (rTS []) [a, b] qs by (rule T_on_embed[symmetric])
  also from as have ... ≤ 2 * T_p_opt [a, b] qs + 2 using TS_OPT2' by fastforce
  finally have T_on_rand' (embed (rTS [])) (fst (embed (rTS []))) [a,b] 
  ≡ (λis. return_pmf ([a,b], is))) qs 
  ≤ 2 * T_p_opt [a, b] qs + 2 .
}  note ye=this

show ?case
  apply (case (Lxy init {a, b}) = [a,b])
  using ye[OF a b1, unfolded rTS_def] apply (simp)
  using pos ye[OF a[symmetric] b2, unfolded rTS_def] by (simp add: twist)
  qed
qed
next
  case 6
  show ?case unfolding TS_step_d_def by (simp add: split_def TS_step_d_def)
next
  case (7 init qs x)
then show ?case
  apply (induct x)
  by (simp_all add: rTS_def split_def take_Suc_convs_app_nth config'_rand_snoc

next
  case 4 then show ?case by simp
qed (simp_all)

lemma TS_compet: compet_rand (embed (rTS [])) 2 {init. distinct init ∧ init ≠ []}
unfolding compet_rand_def static_def
using TS_compet[OF TS_pairwise] by simp

end

16  BIT is pairwise

theory BIT_pairwise
imports List_Factoring BIT
begin

lemma L_nths: S ⊆ {..<length init}
  ⇒ map pmf (λl. nths l S) (Prob_Theory.bv (length init))
  = (Prob_Theory.bv (length (nths init S)))
proof (induct init arbitrary: S)
  case (Cons a as)
  then have passt: {j. Suc j ∈ S} ⊆ {..<length as} by auto

  have map pmf (λl. nths l S) (Prob_Theory.bv (length (a # as))) =
    Prob_Theory.bv (length as) ≫
    (λx. bernoulli_pmf (1 / 2)) ≫
    (λxa. return_pmf
      ((if 0 ∈ S then [xa] else []) @ nths x {j. Suc j ∈ S}))
    by (simp add: map pmf_def bind_return_pmf bind_assoc_pmf nths_Const)

  also have ... = (bernoulli_pmf (1 / 2)) ≫
    (λx. return_pmf ((if 0 ∈ S then [xa] else []) @ nths x {j. Suc j ∈ S}))
    by (rule bind_commutate_pmf)
  also have ... = (bernoulli_pmf (1 / 2)) ≫
    (λxa. (map pmf (λx. (nths x {j. Suc j ∈ S})) (Prob_Theory.bv
      (length as))))

  by (simp_all add: map pmf_def bind_return_pmf bind_assoc_pmf nths_Const)

end
\begin{center}
\begin{align*}
\Rightarrow & \quad (λxs. \text{return}_\mathsf{pmf} \ (\text{if } 0 \in S \text{ then } [xa] \ \text{else } []) \ @ \ xs)) \\
\text{by (simp add: bind\_return\_pmf bind\_assoc\_pmf map\_pmf\_def)} \\
\text{also have } & \quad (\lambda x. \text{Prob\_Theory\.bv \ (length \ (\text{nths \ {j. \ Suc \ j \in S}))}) \\
\Rightarrow & \quad (λxs. \text{return}_\mathsf{pmf} \ (\text{if } 0 \in S \text{ then } [xa] \ \text{else } []) \ @ \ xs)) \\
\text{using } & \quad \text{Cons}(1)[\text{OF passt}] \ \text{by auto} \\
\text{also have } & \quad = \quad \text{Prob\_Theory\.bv \ (length \ (\text{nths \ {a \# as} S}))} \\
\text{apply (auto simp add: nths\_Cons bind\_return\_pmf')} \\
\text{by (rule bind\_commute\_pmf)} \\
\text{finally show } & \quad \text{?case } .
\end{align*}
\end{center}

\textbf{qed (simp)}

\textbf{lemma} \ $L\text{-nths\_Lxy}$:
\textbf{assumes} \ $x\in\text{set \ x} \neq y \ \text{distinct \ set}$
\textbf{shows} \ $\text{map\_pmf} \ (λl. \text{nths \ {l \ \{index \ init \ x, index \ init \ y\}}}) \ (\text{Prob\_Theory\.bv \ (length \ init}))$
\hspace{1cm} $=$ \ $\text{Prob\_Theory\.bv \ (length \ (\text{nths \ init \ {index \ init \ x, index \ init \ y}}))}$
\textbf{proof} –
\textbf{from} \ \textbf{assms}(4) \ \textbf{have} \ set\_init: \ (\text{index \ init}) \ 1 \ \text{set \ init} \ = \ \{0..<\text{length \ init}\}$
\textbf{proof (induct \ init)}
\hspace{1cm} \textbf{case} \ \text{Cons a as}
\hspace{1cm} \textbf{with} \ \text{Cons have} \ iH: \ \text{index \ as} \ 1 \ \text{set \ as} \ = \ \{0..<\text{length \ as}\} \ \text{by auto}
\hspace{1cm} \textbf{from} \ \text{Cons have} \ 1: \ (\text{set \ as} \ \cap \ \{x. \ (a \neq x)\}) \ = \ \text{set \ as} \ \text{by fastforce}
\hspace{1cm} \textbf{have} \ 2: \ (λa. \text{Suc \ (index \ as \ a)}) \ 1 \ \text{set \ as} = \ (λa. \text{Suc \ a}) \ (\text{index \ as}) \ 1 \ \text{set \ as} \ \text{by auto}
\hspace{1cm} \textbf{show} \ \text{?case}
\hspace{1cm} \textbf{apply (simp add: 1 2 iH) by auto}
\textbf{qed simp}

\textbf{have} \ xy\_le: \ \text{index \ init} \ x < \text{length \ init} \ \text{index \ init} \ y < \text{length \ init} \ \text{using \ assms by auto}
\textbf{have} \ \text{map\_pmf} \ (λl. \text{nths \ {l \ \{index \ init \ x, index \ init \ y\}}}) \ (\text{Prob\_Theory\.bv \ (length \ init)})$
\hspace{1cm} $=$ \ $\text{Prob\_Theory\.bv \ (length \ (\text{nths \ init \ {index \ init \ x, index \ init \ y}}))}$
\hspace{1cm} \textbf{apply (rule L\text{-nths)}
\hspace{1cm} \textbf{using} \ \textbf{assms}(1,2) \ \text{by auto}
\hspace{1cm} \textbf{moreover have} \ \text{length} \ (Lxy \ init \ {x, y}) = \ \text{length} \ (\text{nths \ init \ {index \ init} \ x, index \ init \ y})$
\hspace{1cm} \textbf{proof –}
\hspace{1.5cm} \textbf{have} \ \text{set} \ (Lxy \ init \ {x, y}) = \ {x, y}$
\hspace{1.5cm} \textbf{using} \ \textbf{assms}(1,2) \ \textbf{by (simp add: Lxy\_set\_filter)}
\hspace{1.5cm} \textbf{moreover have} \ \text{card} \ {x, y} = 2 \ \textbf{using} \ \textbf{assms}(3) \ \textbf{by auto}
\hspace{1.5cm} \textbf{moreover have} \ \text{distinct} \ (Lxy \ init \ {x, y}) \ \textbf{using} \ \textbf{assms}(4) \ \textbf{by (simp add: Lxy\_distinct)}
ultimately have 1: \( \text{length}(Lxy \text{ init } \{x,y\}) = 2 \) by (simp add: distinct_card[symmetric])

have \( \text{set}(\text{nths init }\{\text{index init }x,\text{index init }y\}) = \{(\text{init }! i) | i. \ i < \text{length init} \land i \in \{\text{index init }x,\text{index init }y\}\} \)

using assms(1,2) by (simp add: set_nth)

moreover have \( \text{card}\{(\text{init }! i) | i. \ i < \text{length init} \land i \in \{\text{index init }x,\text{index init }y\}\} = 2 \)

proof 

have 1: \( \{(\text{init }! i) | i. \ i < \text{length init} \land i \in \{\text{index init }x,\text{index init }y\}\} \) using \( xy \leq \) by blast

also have \( \ldots = \{x,y\} \) using nth_index assms(1,2) by auto

finally show ?thesis using assms(3) by auto

qed

ultimately have 2: \( \text{length}(\text{nths init }\{\text{index init }x,\text{index init }y\}) = 2 \)

by (simp add: distinct_card[symmetric])

ultimately have 2: \( \text{length}(\text{nths init }\{\text{index init }x,\text{index init }y\}) = 2 \)

by (simp add: distinct_card[symmetric])

ultimately show ?thesis by simp

qed

lemma nths_map: \( \text{map }f(\text{nths }xs S) = \text{nths }\text{map }f \text{ xs }S \)

apply (induct xs arbitrary: S) by (simp_all add: nths_Cons)

lemma nths_empty: \( \forall i \in S. \ i \geq \text{length }xs \Rightarrow \text{nths }xs S = [] \)

proof 

assume \( \forall i \in S. \ i \geq \text{length }xs \)

then have \( \text{set}(\text{nths }xs S) = \{\} \) apply (simp add: set_nth) by force

then show \( \text{nths }xs S = [] \) by simp

qed

lemma nths_project': \( i < \text{length }xs \Rightarrow j < \text{length }xs \Rightarrow i < j \)

\( \Rightarrow \text{nths }xs \{i,j\} = [xs!i, \ xs!j]\)

proof 

assume il: \( i < \text{length }xs \) and jl: \( j < \text{length }xs \) and ij: \( i < j \)

from il obtain a as where \( \text{dec1}: a \ *@ [xs!i] @ as = xs \)

and a = take i xs as=drop (Suc i) xs

and length_a: \( \text{length }a = i \) and length_as: \( \text{length }as = \text{length }xs - i - 1 \)

using id_take_nth_drop by fastforce

have \( j \geq \text{length } (a @ [xs!i]) \) using length_a ij by auto

then have \( ((a @ [xs!i]) @ as) ! j = as ! (j - \text{length } (a @ [xs!i])) \) using
nth_append[where \( xs = a \ @ [xs!i] \) and \( ys = as \)]

by (simp)
then have \( xsj \): \( xs ! j = as \) \((j-i-1)\) using dec1 length_a by auto
have las: \((j-i-1) < length as\) using length_as jl ij by simp
obtain \( b \ c \) where \( dec2: b @ [xs!j] @ c = as \)
and \( b = \) take \((j-i-1)\) as \( c=\)drop \((Suc \ (j-i-1))\) as
and length_b: length \( b = j-i-1\) using id_take_nth_drop[OF las]
xsj by force
have \( xs_dec \): \( a @ [xs!i] @ b @ [xs!j] @ c = xs\) using dec1 dec2 by auto

have \( s2: \{ k. (k + i \in \{i, j\}) \} = \{0, j-i\} \) using ij by force
have \( s3: \{ k. (k + length \[xs!i\] \in \{0, j-i\}) \} = \{j-i-1\} \) using ij by force
have \( u: nths a \{i, j\} = [] \)
apply (rule nths_empty) using length_a ij by fastforce
have \( l2: nths b \{j - Suc i\} = [] \)
apply (rule nths_empty) using length_b ij by fastforce
have nths (\( a @ [xs!i] @ b @ [xs!j] @ c \) \( \{i, j\} = [xs!i, xs!j] \)
apply (simp only: nths_append length_a s2 s3 s4 s5)
by (simp add: \( l1 \ l2 \)
then show nths xs \( \{i, j\} = [xs!i, xs!j] \) unfolding \( xs_dec \)
qed

lemma nths_project:
assumes \( i < length xs \) \( j < length xs \) \( i < j \)
shows nths xs \( \{i, j\} ! 0 = xs ! i \) \( \land \) nths xs \( \{i, j\} ! 1 = xs ! j \)
proof –
from \( \text{assms} \) have nths xs \( \{i, j\} = [xs!i, xs!j] \) by (rule nths_project)'
then show ?thesis by simp
qed

lemma BIT_pairwise':
assumes set qs \( \subseteq \) set init
\((x,y) \in \{(x,y). x \in \text{set init} \land y \in \text{set init} \land x \neq y\}\)
and \( xny:k \neq y \) and \( \text{dinit: distinct init} \)
shows \( P\text{before_in} \ x \ y \) BIT qs init = \( P\text{before_in} \ x \ y \) BIT (\( Lxy \) qs \( \{x,y\} \))
(\( Lxy \) init \( \{x,y\} \))
proof –
from \( \text{assms} \) have \( xyninit: \{x, y\} \subseteq \text{set init} \)
and \( qsininit: \text{set qs} \subseteq \text{set init} \) by auto
have \( xyninit: \{y, x\} \subseteq \text{set init} \) using \( xyninit \) by auto

322
have a: \( x \in \text{set init} \) \( y \in \text{set init} \) using assms by auto

\[
\begin{align*}
\{ \text{fix } n \\
\text{have strong: set qs } &\subseteq \text{set init } \implies \\
\text{map_pmf } (\lambda(l,(w,i)). (Lxy l \{x,y\},(nths w \{\text{index init} x,\text{index init} y\},Lxy init \{x,y\}))) (\text{config_rand BIT init qs}) = \\
\text{config_rand BIT } (Lxy init \{x,y\}) (Lxy qs \{x,y\}) (\text{is inv } \implies \ ?L \ qs = \ ?R \ qs) \\
\text{proof } (\text{induct qs rule: rev_induct}) \\
\text{case Nil} \\
\text{have map_pmf } (\lambda(l,(w,i)). (Lxy l \{x,y\},(nths w \{\text{index init} x,\text{index init} y\},Lxy init \{x,y\}))) (\text{config_rand BIT init }[]) = \\
\text{map_pmf } (\lambda w. (Lxy init \{x,y\}, (w, Lxy init \{x,y\}))) (\text{map_pmf } (\lambda l. nths l \{\text{index init} x,\text{index init} y\}) (\text{Prob_Theory.bv } (\text{length init}))) \\
\text{by}(\text{simp add: bind_return_pmf map_pmf_def bind_assoc_pmf split_def BIT_init_def}) \\
\text{also have }\ldots = \text{map_pmf } (\lambda w. (Lxy init \{x,y\}, (w, Lxy init \{x,y\}))) (\text{Prob_Theory.bv } (\text{length } (Lxy init \{x,y\}))) \\
\text{using L_nths_Lxy[OF a xny dinit] by simp} \\
\text{also have }\ldots = \text{config_rand BIT } (Lxy init \{x,y\}) (Lxy [] \{x,y\}) \\
\text{by}(\text{simp add: BIT_init_def bind_return_pmf bind_assoc_pmf map_pmf_def}) \\
\text{finally show } ?\text{case .} \\
\text{next} \\
\text{case } (\text{snoc q qs}) \\
\text{then have qininit: } q \in \text{ set init} \\
\text{and qsininit: set qs } \subseteq \text{ set init using qsininit by auto} \\
\text{from } \text{snoc}(1)[OF qsininit] \text{ have iH: } ?L \ qs = ?R \ qs \text{ by } (\text{simp add: split_def}) \\
\text{show } ?\text{case} \\
\text{proof } (\text{cases } q \in \{x,y\}) \\
\text{case True} \\
\text{note whatisq=}\text{this} \\
\text{have } ?L \ (qs@[q]) = \\
\text{map_pmf } (\lambda(l,(w,i)). (Lxy l \{x,y\},(nths w \{\text{index init} x,\text{index init} y\},Lxy init \{x,y\}))) (\text{config_rand BIT init qs }\gg= \\
(\lambda s. \text{BIT_step s q }\gg= (\lambda(a, nis). \text{return_pmf } \text{step } \text{fst s} a, nis)))) \\
\text{by}(\text{simp add: split_def config_rand_snoc}) \\
\text{also have }\ldots =
\end{align*}
\]

323
map_pmf (λ(l,(w,i)). (Lxy l {x,y}, (nths w {index init x,index init y},Lxy init {x,y}))) (config_rand BIT init qs) ⊨
(λs.
  BIT_step s q ⊨
  (λ(a, nis). return_pmf (step (fst s) q a, nis)))
apply(simp add: map_pmf_def split_def bind_return_pmf bind_assoc_pmf)
apply(simp add: BIT_step_def bind_return_pmf)
proof (rule bind_pmf_cong, goal_cases)
case 2 z
let ?s = fst z
let ?b = fst (snd z)

from 2 have z: set (?s) = set init using config_rand_set[of BIT, simplified] by metis
  with True have qLxy: q ∈ set (Lxy (?s) {x, y}) using xyinitinit
by (simp add: Lxy_set_filter)
from 2 have dz: distinct (?s) using dinit config_rand_distinct[of BIT, simplified] by metis
  then have dLxy: distinct (Lxy (?s) {x, y}) using Lxy_distinct by auto

from 2 have [simp]: snd (snd z) = init using config_n_init3[simplified]
by metis

from 2 have [simp]: length (fst (snd z)) = length init using config_n_fst_init_length2[simplified] by metis

have indexinbounds: index init x < length init index init y < length init using a by auto
  from a xny have indnot: index init x ≠ index init y by auto

have f1: index init x < length (fst (snd z)) using xyinitinit by auto
have f2: index init y < length (fst (snd z)) using xyinitinit by auto
have 3: index init x ≠ index init y using xny xyinitinit by auto

from dinit have dfil: distinct (Lxy init {x,y}) by(rule Lxy_distinct)
  have Lxy_set: set (Lxy init {x, y}) = {x,y} apply(simp add: Lxy_set_filter) using xyinitinit by fast
  then have xLxy: x ∈ set (Lxy init {x, y}) by auto
  have Lxy_length: length (Lxy init {x, y}) = 2 using dfil Lxy_set
  xny distinct_card by fastforce
have 31: \( \text{index } (L_{xy \text{ init } \{x, y\}}) x < 2 \)
and 32: \( \text{index } (L_{xy \text{ init } \{x, y\}}) y < 2 \) using \( L_{xy \text{ set } xyininit} \)
\( L_{xy \text{ length by auto}} \)

have 33: \( \text{index } (L_{xy \text{ init } \{x, y\}}) x \neq \text{index } (L_{xy \text{ init } \{x, y\}}) y \)
using \( \text{xny xLxy by auto} \)

have a1: \( nths (\text{flip} (\text{index init } (q)) (\text{fst } (\text{snd } z))) \{\text{index init x}, \text{index init y}\} \)
\( = \text{flip} (\text{index } (L_{xy \text{ init } \{x,y\}}) (q)) (\text{nth s } (\text{fst } (\text{snd } z)) \{\text{index init x}, \text{index init y}\}) \) (is \( ?A=?B \))

proof (simp only: list_eq_iff_nth_eq, goal_cases)
case 1

{assume ass: \( \text{index init x} < \text{index init y} \)
then have \( \text{index } (L_{xy \text{ init } \{x,y\}}) x < \text{index } (L_{xy \text{ init } \{x,y\}}) y \)
using \( L_{xy \text{ mono}[OF xyininit dinit]} \) before_in_def a(2) by force

with 31 32 have ix: \( \text{index } (L_{xy \text{ init } \{x,y\}}) x = 0 \)
and iy: \( \text{index } (L_{xy \text{ init } \{x,y\}}) y = 1 \) by auto

have g1: \( nths (\text{flip} (\text{index init } (q)) (\text{fst } (\text{snd } z))) \{\text{index init x}, \text{index init y}\} \)
\( = [(\text{fst } (\text{snd } z)) \! \text{index init x}, (\text{fst } (\text{snd } z)) \! \text{index init y}] \)
apply (rule nth\_project')
using \( \text{xyininit apply(simp)} \)
using \( \text{xyininit apply(simp)} \)
by fact

have nths (\( \text{flip} (\text{index init } (q)) (\text{fst } (\text{snd } z))) \{\text{index init x}, \text{index init y}\} \)
\( = [\text{flip} (\text{index init } (q)) (\text{fst } (\text{snd } z))!\text{index init x}, \)
\( \text{flip} (\text{index init } (q)) (\text{fst } (\text{snd } z))!\text{index init y}] \)
apply (rule nth\_project')
using \( \text{xyininit apply(simp)} \)
using \( \text{xyininit apply(simp)} \)
by fact
also have \( . . . = \text{flip} (\text{index } (L_{xy \text{ init } \{x,y\}}) (q)) [(\text{fst } (\text{snd } z)) \! \text{index init x}, (\text{fst } (\text{snd } z)) \! \text{index init y}] \)
apply (cases \( q=x \))
apply (simp add: ix) using flip\_other[OF f2 f1 3] flip\_itself[OF f1] apply (simp)
using whatsisq apply (simp add: iy) using flip\_other[OF f1 f2 3[symmetric]] flip\_itself[OF f2] by (simp)
finally have nths (flip (index init (q)) (fst (snd z))) {index init x,index init y}
    = flip (index (Lxy init {x,y}) (q)) (nths (fst (snd z)) {index init x,index init y})
    by(simp add: g1)

}{note cas1=this
have man: \{x,y\} = \{y,x\} by auto
{assume " index init x < index init y
then have as: index init y < index init init x using 3 by auto
then have index (Lxy init \{x,y\}) y < index (Lxy init \{x,y\}) x
using Lxy_mono[OF xyininit!dinit] xyininit a(1) man by(simp add: before_in_def)
with 31 32 have ix: index (Lxy init \{x,y\}) x = 1
and iy: index (Lxy init \{x,y\}) y = 0 by auto

have g1: (nths (fst (snd z)) {index init y,index init init x})
    = [(fst (snd z)) ! index init y, (fst (snd z)) ! index init init x]
    apply(rule nthss_project')
    using xyininit apply(simp)
    using xyininit apply(simp)
    by fact

have man2: \{index init x,index init init y\} = \{index init y,index init init x\} by auto
have nths (flip (index init (q)) (fst (snd z))) \{index init y,index init init x\}
    = [flip (index init (q)) (fst (snd z))!index init y,
        flip (index init (q)) (fst (snd z))!index init init x]
    apply(rule nthss_project')
    using xyininit apply(simp)
    using xyininit apply(simp)
    by fact
also have \ldots = \{index init y, (fst (snd z)) ! index init x\}
apply(cases q=x)
    apply(simp add: ix) using flip_other[OF f2 f1 3] flip_itself[OF f1]
    apply(simp)
    using whatisq apply(simp add: iy) using flip_other[OF f1 f2 3[symmetric]] flip_itself[OF f2] by(simp)
finally have nths (flip (index init (q)) (fst (snd z))) \{index init y,index init init x\}
    = flip (index (Lxy init \{x,y\}) (q)) (nths (fst (snd z)) \{index

$\text{init } y, \text{index init } x \}$

by (simp add: g1)

then have $\text{nths (flip (index init } (q)) (fst (snd z))) \{\text{index init } x, \text{index init } y\}$

$= \text{flip (index (Lxy init } \{x,y\}) (q)) \{\text{nths (fst (snd z)) \{\text{index init } x, \text{index init } y\}\}$

using man2 by auto

} note cas2 = this

from cas1 cas2 3 show ?case by metis

qed

have $a: \text{nths (fst (snd z)) \{\text{index init } x, \text{index init } y\} ! (index (Lxy init } \{x,y\}) (q))$

$= \text{fst (snd z)} ! (index init (q))$

proof –

from 31 32 33 have $ca: (\text{index (Lxy init } \{x,y\}) x = 0 \land index (Lxy init } \{x,y\}) y = 1)$

$\lor (\text{index (Lxy init } \{x,y\}) x = 1 \land index (Lxy init } \{x,y\}) y = 0)$ by force

show ?thesis

proof (cases index (Lxy init } \{x,y\}) x = 0)

case True

from True $ca$ have $y1: \text{index (Lxy init } \{x,y\}) y = 1$ by auto

with True have $\text{index (Lxy init } \{x,y\}) x < \text{index (Lxy init } \{x,y\}) y$ by auto

then have $xy: \text{index init } x < \text{index init } y$ using dinit dfil Lxy_mono

using 32 before_in_def Lxy_length xyininit by fastforce

have $4: \{\text{index init } y, \text{index init } x\} = \{\text{index init } x, \text{index init } y\}$ by auto

have $\text{nths (fst (snd z)) \{\text{index init } x, \text{index init } y\} ! index (Lxy init } \{x,y\}) x = (fst (snd z)) ! \text{index init } x$

$nths (fst (snd z)) \{\text{index init } x, \text{index init } y\} ! index (Lxy init } \{x,y\}) y = (fst (snd z)) ! \text{index init } y$

unfolding True y1

by (simp_all only: nths_project[OF f1 f2 xy])

with whatisq show ?thesis by auto

next

case False

327
with \texttt{ca} have \( x1: \text{index} \ (Lxy \text{ init } \{x,y\}) \)? \( x = 1 \) by \textit{auto}
from \texttt{dinit} have \texttt{dfil}: distinct \ (Lxy \text{ init } \{x,y\}) \ by (rule \ Lxy\_distinct)

from \( x1 \) ca have \( y1: \text{index} \ (Lxy \text{ init } \{x,y\}) \)? \( y = 0 \) by \textit{auto}
with \( x1 \) have \( \text{index} \ (Lxy \text{ init } \{x,y\}) \)? \( y < \text{index} \ (Lxy \text{ init } \{x,y\}) \)
\( x \) by \textit{auto}
then have \( xy: \text{index} \) \( \text{init} \) \( y \) < \( \text{index} \) \( \text{init} \) \( x \) \textit{using} \( \text{dinit} \) \( \text{dfil} \)

have \( 4: \{\text{index} \ \text{init} \ y, \text{index} \ \text{init} \ x\} = \{\text{index} \ \text{init} \ x, \text{index} \ \text{init} \ y\} \) by \textit{auto}

have \( \text{nths} \ (?b) \{\text{index} \ \text{init} \ x, \text{index} \ \text{init} \ y\} \)? \( \text{index} \ (Lxy \text{ init } \{x,y\}) \)? \( x = (?b) \text{! index init x} \)
\( \text{nths} \ (?b) \{\text{index} \ \text{init} \ x, \text{index} \ \text{init} \ y\} = (?b) \text{! index init y} \)
\text{unfolding} \( x1 \) \( y1 \)

using \( 32 \) \textit{before_in_def} \( Lxy\_length \text{ xinininit} \ by \ (\text{metis} \ a(2) \ \text{indnot} \ \text{linorder} \neg \text{E} \ \text{nat not} \text{less0} \ y1) \)

have \( 4: \{\text{index} \ \text{init} \ y, \text{index} \ \text{init} \ x\} = \{\text{index} \ \text{init} \ x, \text{index} \ \text{init} \ y\} \) by \textit{auto}

have \( \text{nths} \ (?b) \{\text{index} \ \text{init} \ x, \text{index} \ \text{init} \ y\} = \{\text{index} \ \text{init} \ x, \text{index} \ \text{init} \ y\} \)
unfolding \( x1 \) \( y1 \)

by \textit{simp_all}
with \( \text{whatisq} \) \textit{show} ?\text{thesis} \ by \textit{auto}
\textbf{qed}
\textbf{qed}

have \( b: Lxy \ (\text{mtf2} \ (\text{length} \ ?s) \ (q) \ ?s) \{x, y\} = ?B) \)
proof
  have \( sA: \text{set} \ ?A = \{x,y\} \) \textit{using} \( z \) \( \text{xininit} \ by(\text{simp add:} \ Lxy\_set\_filter) \)
  then have \( \text{xlymA}: x \in \text{set} \ ?A \)
    and \( \text{ylymA}: y \in \text{set} \ ?A \) \textit{by} \textit{auto}
  have \( dA: \text{distinct} \ ?A \) \textit{apply} (rule \( Lxy\_distinct) \ by (\text{simp add:} \ dz)
  have \( lA: \text{length} \ ?A = 2 \) \textit{using} \( xny \) \( sA \) \( dA \) \textit{distinct_card} \textit{by} \textit{fastforce}
  from \( lA \) \( \text{ylxymA} \) have \( \text{yindA}: \text{index} \ ?A \ y < 2 \) \textit{by} \textit{auto}
  from \( lA \) \( \text{xlymA} \) have \( \text{xindA}: \text{index} \ ?A \ x < 2 \) \textit{by} \textit{auto}
  have \( \text{geA}: \{x,y\} \subseteq \text{set} \ (\text{mtf2} \ (\text{length} \ ?s) \ (q) \ ?s) \) \textit{using} \( \text{xyinininit} \) \( z \) \textit{by} \textit{auto}

328
have \( \text{gcA}': \{y,x\} \subseteq \text{set (mtf2 (length \ ?s) (q) (\?s)) using xyininit z by auto} \)

have \( \text{man}': \{y,x\} = \{x,y\} \) by auto

\[
\text{have sB: set } \?B = \{x,y\} \text{ using } z \text{ xyininit by (simp add: Lxy_set_filter)}
\]

then have \( xlxymB: x \in \text{set } \?B \)

and \( ylxymB: y \in \text{set } \?B \) by auto

have \( dB: \text{distinct } \?B \) apply (simp) apply (rule Lxy_distinct)

by (simp add: dz)

have \( IB: \text{length } \?B = 2 \) using xny sB dB distinct_card by fastforce

from \( IB ylxymB \) have \( yindB: \text{index } \?B y < 2 \) by auto

from \( IB xlxymB \) have \( xindB: \text{index } \?B x < 2 \) by auto

show ?thesis

proof (cases \( q = x \))

case True

then have \( xby: x < y \) in \( \text{mtf2 (length } \?s) (q) (\?s)) \)

apply (simp)

apply (rule mtf2_moves_to_front"[simplified])

using \( z \text{ xyininit xny by (simp_all add: dz)} \)

then have \( x < y \) in \( \?A \) using Lxy_mono \( OF \text{ gcA] dz by (auto) \)

then have \( \text{index } \?A x < \text{ index } \?A y \) unfolding before_in_def by auto

then have \( in1: \text{index } \?A x = 0 \)

and \( in2: \text{index } \?A y = 1 \) using yindA by auto

have \( \?A = [\text{?A}!0, \text{?A}!1] \)

apply (simp only: list_eq_iff_nth_eq)

apply (auto simp: \( \text{LA} \) apply (case_tac \( i \)) by (auto)

also have \( \ldots = [\text{?A}!\text{index } \?A x, \text{?A}!\text{index } \?A y] \) by (simp only: in1 in2)

also have \( \ldots = [x,y] \) using xlxymA ylxymA by simp

finally have \( end1: \?A = [x,y] \).

have \( x < y \) in \( \?B \)

using True apply (simp)

apply (rule mtf2_moves_to_front"[simplified])

using \( z \text{ xyininit xny by (simp_all add: Lxy_distinct} \)

dz Lxy_set_filter)

then have \( \text{index } \?B x < \text{ index } \?B y \)

unfolding before_in_def by auto

then have \( in1: \text{index } \?B x = 0 \)

and \( in2: \text{index } \?B y = 1 \)

329
using \texttt{yindB} by \texttt{auto}

have \(?B = [?B!0, ?B!1]\)
  apply (simp only: list\_eq\_iff\_nth\_eq)
  apply (simp only: \texttt{lB})
  apply (auto) apply (case\_tac \texttt{i}) by (auto)

also have \(\ldots = [?B!\text{index } ?B \ x, \ ?B!\text{index } ?B \ y]\)
  by (simp only: \texttt{in1 in2})

also have \(\ldots = [x, y]\) using \texttt{xlymB ylxymB} by simp

finally have \texttt{end2: ?B = [x, y]}.

then show \(?A = ?B\) using \texttt{end1 end2} by simp

next

\texttt{case False}

with \texttt{whatissq} have \texttt{qsy: q=y} by simp

then have \(\texttt{xyb: y < x}\) in \texttt{(mtf2 (length (?s)) (q) (?s))}
  apply (simp)
  apply (rule mtf2\_moves\_to\_front"[simplified!])
  using \(\texttt{z xyinit xny}\) by (simp\_all add: \texttt{dz})

then have \(y < x\) in \(?A\) using \texttt{Lxy\_mono [OF geA]} man \texttt{dz}
by \texttt{auto}

then have \(\texttt{index ?A y < index ?A x}\) unfolding \texttt{before}\_in\_def
by \texttt{auto}

then have \(\texttt{in1: index ?A y = 0}\)
  and \(\texttt{in2: index ?A x = 1}\) using \texttt{xindA} by \texttt{auto}

have \(?A = [?A!0, ?A!1]\)
  apply (simp only: list\_eq\_iff\_nth\_eq)
  apply (auto simp: \texttt{lA})
  apply (case\_tac \texttt{i}) by (auto)

also have \(\ldots = [?A!\text{index } ?A \ y, ?A!\text{index } ?A \ x]\) by (simp only: \texttt{in1 in2})

also have \(\ldots = [y, x]\) using \texttt{xlymA ylxmA} by simp

finally have \texttt{end1: ?A = [y, x]}.

have \(y < x\) in \(?B\)
  using \texttt{qsy} apply (simp)
  apply (rule mtf2\_moves\_to\_front"[simplified!])
  using \(\texttt{z xyinit xny}\) by (simp\_all add: \texttt{Lxy\_distinct})

\texttt{dz Lxy\_set\_filter})

then have \(\texttt{index ?B y < index ?B x}\)
  unfolding \texttt{before\_in\_def} by \texttt{auto}

then have \(\texttt{in1: index ?B y = 0}\)
  and \(\texttt{in2: index ?B x = 1}\)
  using \texttt{xindB} by \texttt{auto}
have $B = [B10, B11]$
   apply(simp only: list_eq_iff_nth_eq)
   apply(simp only: lB)
   apply(auto) apply(case_tac i) by(auto)
also have ... = $[B1index ?B y, B1index ?B x]$
   by(simp only: in1 in2)
also have ... = $[y, x]$ using $x$lxymB $y$lxymB by(simp )
finally have end2: $B = [y, x]$.

then show $A = B$ using end1 end2 by simp
qed

show $?case$ using a1 a2 by(simp)
qed simp
also have ... = $?R (qs \cdot[q])$
   using True apply(simp add: Lxy_snoc take_Suc_conv_app_nth
config'rand_snoc)
   using iH by(simp add: split_def )
finally show $?thesis$.
next
case False
then have qnx: $(q) \neq x$ and qny: $(q) \neq y$ by auto

let $?proj$=($\lambda$a. $(Lxy \ (fst \ a) \ {x, y}, \ (nths \ (fst \ (snd \ a)) \ {index \ init \ x, \ index \ init \ y})$, \ $Lxy \ init \ {x, y}$))

have map_pmf $?proj \ (config\_rand \ BIT \ init \ (qs@[q]))
   = map_pmf $?proj \ (config\_rand \ (BIT\_init, \ BIT\_step) \ init \ qs
   \Rightarrow (\lambda p. \ BIT\_step \ p \ (q) \Rightarrow (\lambda pa. \ return\_pmf \ (step \ (fst \ p) \ (q)
   (fst \ pa), \ snd \ pa)))
   by (simp add: split_def take_Suc_conv_app_nth config'rand_snoc)
also have ... = map_pmf $?proj \ (config\_rand \ (BIT\_init, \ BIT\_step)
init \ qs)
   apply(simp add: map_pmf_def bind_assoc_pmf bind_return_pmf
BIT_step_def)
proof (rule bind_pmf_cong, goal_cases)
case (2 z)
  let ?s = fst z
  let ?m = snd (snd z)
  let ?b = fst (snd z)

  from 2 have sf_init: ?m = init using config_n_init3 by auto

  from 2 have ff_len: length (?b) = length init using config_n_fst_init_length2 by auto

    have ff_ix: index init x < length (?b) unfolding ff_len using a(1) by auto
    have ff_iy: index init y < length (?b) unfolding ff_len using a(2) by auto
    have ff_q: index init (q) < length (?b) unfolding ff_len using qininit by auto

    have iq_ix: index init (q) ≠ index init x using a(1) qnx by auto
    have iq_iy: index init (q) ≠ index init y using a(2) qny by auto

    from 2 have s_set[simp]: set (?s) = set init using config_rand_set
      by force
    have s_xin: x∈set (?s) using a(1) by simp
    have s_yin: y∈set (?s) using a(2) by simp
      from 2 have s_dist: distinct (?s) using config_rand_distinct
    dinit by force
    have s_qin: q ∈ set (?s) using qininit by simp

    have fstfst: nths (flip (index ?m (q)) (?b))
      {index init x, index init y}
    = nths (?b) {index init x, index init y} (is nths ?A ?I = nths ?B ?I)

  proof (cases index init x < index init y)
    case True
      have nths ?A ?I = [?A!index init x, ?A!index init y]
        by (rule nths_project)"
also have \ldots = \text{nths question} \text{B question} \text{I}
  apply (rule \text{nths-project}[\text{symmetric}])
  by (simp_all add: \text{ff}_x \text{ff}_y \text{True})

finally show \text{?thesis}.

next
case \text{False}
then have \text{yx}: index \text{init y} < index \text{init x} using \text{ix}_y \text{iy} by auto

have \text{man}: \text{?I} \text{= \{index init y, index init x\}} by auto

have \text{\text{nths question} A \{index init y, index init x\} = \{?A!index init y, ?A!index init x\}}
  apply (rule \text{nths-project'})
  by (simp_all add: \text{ff}_x \text{ff}_y \text{yx})

also have \ldots = \{?B!index init y, ?B!index init x\}
  unfolding \text{sf_init using flip}\_{\text{other} }\text{ff}_x \text{ff}_y \text{ff}_q \text{iq}_x \text{iq}_y
  by auto

also have \ldots = \text{nths question} \text{B \{index init y, index init x\}}
  apply (rule \text{nths-project}[\text{symmetric}])
  by (simp_all add: \text{ff}_x \text{ff}_y \text{yx})

finally show \text{?thesis} by (simp add: \text{man})
qed

have \text{\text{snd: Lxy \{step ?s\} (q)}
  (if \text{\text{?b! index ?m} (q) then 0 else length\{?s\},}
    []) \text{\{x, y\} = Lxy \{?s\} \{x, y\} (is Lxy \text{?A \{x,y\} = Lxy question B \{x,y\}}}\}

\{x, y\})

\text{proof (cases x < y in question B)}
\text{case True}
\text{note B=this}

  then have A: x<y in question A \text{ apply (auto simp add: step_def split_def)}

  apply (rule x_stays_before_y if_y_not_moved_to_front)
  by (simp_all add: a s_dist qny[symmetric] qinin)

have \text{Lxy question A \{x,y\} = \{x,y\}}

apply (rule \text{Lxy_project})
by (simp_all add: xny set_step distinct_step A s_dist a)

also have \ldots = \text{Lxy question B \{x,y\}}

apply (rule \text{Lxy_project}[symmetric])
by (simp_all add: xny B s_dist a)

finally show \text{?thesis}.

next
case \text{False}
then have B: y < x in question B using not_before_in[\text{OF s_xin s_yin}]

333
xny by simp

then have A: y < x in ?A apply(auto simp add: step_def split_def)

apply(rule x_stays_before_y_if_y_not_moved_to_front)
by(simp_all add: a s_dist qnx[symmetric] qininit)

have man: \{x,y\} = \{y,x\} by auto

have Lxy ?A \{y,x\} = \[y,x\]
apply(rule Lxy_project)
by(simp_all add: xny[symmetric] set_step distinct_step A)

also have ... = Lxy ?B \{y,x\}
apply(rule Lxy_project[symmetric])
by(simp_all add: xny[symmetric] B s_dist a)

finally show ?thesis using man by auto

times

show ?case by(simp add: fstfst snd)

qed simp

also have ... = config_rand BIT (Lxy init \{x, y\}) (Lxy qs \{x, y\})
using iH by (auto simp: split_def)

also have ... = ?R (qs@[q])
using False by(simp add: Lxy_snoc)

finally show ?thesis by (simp add: split_def)

qed

qed

} note strong=\this

{
fix n::nat

have Pbefore_in x y BIT qs init =
map_pmf (\lambda. x < y in fst p)
(map_pmf (\lambda(l, w, i)). (Lxy l \{x, y\}, (nths w \{index init x, index init y\}, Lxy init \{x, y\})))
(config_rand BIT init qs))

unfolding Pbefore_in_def apply(simp add: map_pmf_def bind_return_pmf bind_assoc_pmf split_def)

apply(rule bind_pmf_cong)

apply(simp)

proof (goal_cases)

  case (1 z)
  let ?s = fst z
  from 1 have u: set (?s) = set init using config_rand[of BIT, simplified] by metis

  from 1 have v: distinct (?s) using dinit config_rand[of BIT, simplified] by metis

334
BIT, simplified] by metis
  have (x < y in ?s) = (x < y in Lxy (Lxy ?s) {x, y})
  apply(rule Lxy_mono)
     using u xyminit apply(simp)
     using v by simp
  then show ?case by simp
qed

also have ...
  map_pmf (λp. x < y in fst p) (config_rand BIT (Lxy init {x, y}))
  apply(subst strong) using assms by simp
also have ...
  Pbefore_in x y BIT (Lxy qs {x, y}) (Lxy init {x, y})
unfolding Pbefore_in_def by simp
finally have Pbefore_in x y BIT qs init =
  Pbefore_in x y BIT (Lxy qs {x, y}) (Lxy init {x, y}).

} note fine=this

with assms show ?thesis by simp
qed

theorem BIT_pairwise: pairwise BIT
  apply(rule pairwise_property_lemma)
  apply(rule BIT_pairwise')
  by(simp_all add: BIT_step_def)

end

17 BIT is 1.75 competitive on lists of length 2

theory BIT_2comp_on2
imports BIT Phase_Partitioning
begin

17.1 auxiliary lemmas

17.1.1 E_bernoulli3

lemma E_bernoulli3: assumes 0<p
  and p<1
  and finite (set_pmf (bind_pmf (bernoulli_pmf p) f))
  shows E (bind_pmf (bernoulli_pmf p) f) = E(f True)*p + E(f False)*(1-p)
(is ?L = ?R)
proof –

have \( T : (\sum_{a \in (\bigcup x. \text{set}\_\text{pmf}\ (f\ x))} (a \ast \text{pmf}\ (f\ True)\ a)) = E(f\ True) \)
  unfolding \( E\_\text{def} \)
  apply(subst integral\_measure\_pmf[of\ bind\_pmf\ (\text{bernoulli}\_\text{pmf}\ p)\ f])
  using assms apply(simp)
  using assms apply(simp add: set\_pmf\_bernoulli) apply blast
have \( F : (\sum_{a \in (\bigcup x. \text{set}\_\text{pmf}\ (f\ x))} (a \ast \text{pmf}\ (f\ False)\ a)) = E(f\ False) \)
  unfolding \( E\_\text{def} \)
  apply(subst integral\_measure\_pmf[of\ bind\_pmf\ (\text{bernoulli}\_\text{pmf}\ p)\ f])
  using assms apply(simp)
  using assms apply(simp add: set\_pmf\_bernoulli) apply blast
have \( \text{?L} = (\sum_{a \in (\bigcup x. \text{set}\_\text{pmf}\ (f\ x))} a \ast (\text{pmf}\ (f\ True)\ a \ast p + \text{pmf}\ (f\ False)\ a \ast (1 - p))) \)
  unfolding \( E\_\text{def} \)
  apply(subst integral\_measure\_pmf[of\ bind\_pmf\ (\text{bernoulli}\_\text{pmf}\ p)\ f])
  using assms apply(simp)
  apply(simp)
  using assms apply(simp add: set\_pmf\_bernoulli)
  by(simp add: pmf\_bind\_mult\_ac)
also have \do\ (\sum_{a \in (\bigcup x. \text{set}\_\text{pmf}\ (f\ x))} (a \ast \text{pmf}\ (f\ True)\ a \ast p + (a \ast \text{pmf}\ (f\ False)\ a \ast (1 - p)))
  apply(rule sum.cong) apply(simp) by algebra
also have \do\ (\sum_{a \in (\bigcup x. \text{set}\_\text{pmf}\ (f\ x))} (a \ast \text{pmf}\ (f\ True)\ a \ast p))
  + (\sum_{a \in (\bigcup x. \text{set}\_\text{pmf}\ (f\ x))} (a \ast \text{pmf}\ (f\ False)\ a \ast (1 - p)))
  by(simp add: pmf\_bind\_mult\_ac)
also have \do\ (\sum_{a \in (\bigcup x. \text{set}\_\text{pmf}\ (f\ x))} (a \ast \text{pmf}\ (f\ True)\ a) \ast p
  + (\sum_{a \in (\bigcup x. \text{set}\_\text{pmf}\ (f\ x))} (a \ast \text{pmf}\ (f\ False)\ a) \ast (1 - p))
  by(simp add: pmf\_bind\mult\_ac)
also have \do\ ?R unfolding \( T\ F \) by simp
finally show \( \text{thesis} \).
qed
17.1.2 types of configurations

definition type0 init x y = do {
  (a::bool) ← (bernoulli_pmf 0.5);
  (b::bool) ← (bernoulli_pmf 0.5);
  return pmf ([x,y], ([a,b],init))
}

definition type1 init x y = do {
  (a::bool) ← (bernoulli_pmf 0.5);
  (b::bool) ← (bernoulli_pmf 0.5);
  return pmf (if ~[a,b]!(index init x)∧[a,b]!(index init y) then
              ([y,x], ([a,b],init))
              else ([x,y], ([a,b],init)))
}

definition type3 init x y = do {
  (a::bool) ← (bernoulli_pmf 0.5);
  (b::bool) ← (bernoulli_pmf 0.5);
  return pmf (if [a,b]!(index init x)∧~[a,b]!(index init y) then
              ([x,y], ([a,b],init))
              else ([y,x], ([a,b],init)))
}

definition type4 init x y = do {
  (a::bool) ← (bernoulli_pmf 0.5);
  (b::bool) ← (bernoulli_pmf 0.5);
  return pmf (if ~[a,b]!(index init y)then
              ([x,y], ([a,b],init))
              else ([y,x], ([a,b],init)))
}

definition BIT\_inv s x i == (s = (type0 i x (hd (filter (λy. y≠x) i))))

lemma BIT\_inv2: x≠y ⇒ z∈\{x,y\} ⇒ BIT\_inv s z [x,y] = (s= type0 [x,y] z (other z x y))

  unfolding BIT\_inv\_def by(auto simp add: other\_def)

17.1.3 cost of BIT

lemma costBIT\_0x:
  assumes x≠y x : {x0,y0} y∈{x0,y0}
  shows E (type0 [x0, y0] x y ≫
       (λs. BIT\_step s x ≫
\[(\lambda(a, \text{is}'). \text{return}\_pmf \ (\text{real} \ (t_p \ (\text{fst} \ s) \ x \ a))) = 0\]

using assms apply(auto)
apply(simp_all add: type0_def BIT_step_def bind_assoc_pmf bind_return_pmf)
apply(simp_all add: E_bernoulli3 t_p_def)
done

lemma costBIT_0y:
  assumes \(x \neq y \in \{x0, y0\}\)
  shows 
  \(E\ (\lambda s. \text{BIT}\_step\ s \ y \Rightarrow \ (\lambda(a, \text{is}'). \text{return}\_pmf \ (\text{real} \ (t_p \ (\text{fst} \ s) \ y \ a))) = 1\)
using assms apply(auto)
apply(simp_all add: type0_def BIT_step_def bind_assoc_pmf bind_return_pmf)
apply(simp_all add: E_bernoulli3 t_p_def)
done

lemma costBIT_1x:
  assumes \(x \neq y \in \{x0, y0\}\)
  shows 
  \(E\ (\lambda s. \text{BIT}\_step\ s \ x \Rightarrow \ (\lambda(a, \text{is}'). \text{return}\_pmf \ (\text{real} \ (t_p \ (\text{fst} \ s) \ x \ a))) = 1/4\)
using assms apply(auto)
apply(simp_all add: type1_def BIT_step_def bind_assoc_pmf bind_return_pmf)
apply(simp_all add: E_bernoulli3 t_p_def)
done

lemma costBIT_1y:
  assumes \(x \neq y \in \{x0, y0\}\)
  shows 
  \(E\ (\lambda s. \text{BIT}\_step\ s \ y \Rightarrow \ (\lambda(a, \text{is}'). \text{return}\_pmf \ (\text{real} \ (t_p \ (\text{fst} \ s) \ y \ a))) = 3/4\)
using assms apply(auto)
apply(simp_all add: type1_def BIT_step_def bind_assoc_pmf bind_return_pmf)
apply(simp_all add: E_bernoulli3 t_p_def)
done

lemma costBIT_3x:
assumes $x \neq y$ : $\{x_0, y_0\}$, $y \in \{x_0, y_0\}$
shows
\[ E \left( \text{type3 } [x_0, y_0] \right) x y \quad \Rightarrow \]
\[ (\lambda s. \text{BIT\_step } s x \quad \Rightarrow \]
\[ (\lambda(a, is'). \text{return\_pmf } (\text{real } (t_p (\text{fst } s) x a)))) = 3/4 \]
using assms apply(auto)
apply(simp_all add: type3_def BIT\_step_def bind_assoc\_pmf bind\_return\_pmf)
apply(simp_all add: E\_bernoulli3 t\_p_def)
done

lemma costBIT\_3y:
assumes $x \neq y$ : $\{x_0, y_0\}$, $y \in \{x_0, y_0\}$
shows
\[ E \left( \text{type3 } [x_0, y_0] \right) x y \quad \Rightarrow \]
\[ (\lambda s. \text{BIT\_step } s y \quad \Rightarrow \]
\[ (\lambda(a, is'). \text{return\_pmf } (\text{real } (t_p (\text{fst } s) y a)))) = 1/4 \]
using assms apply(auto)
apply(simp_all add: type3_def BIT\_step_def bind_assoc\_pmf bind\_return\_pmf)
apply(simp_all add: E\_bernoulli3 t\_p_def)
done

lemma costBIT\_4x:
assumes $x \neq y$ : $\{x_0, y_0\}$, $y \in \{x_0, y_0\}$
shows
\[ E \left( \text{type4 } [x_0, y_0] \right) x y \quad \Rightarrow \]
\[ (\lambda s. \text{BIT\_step } s x \quad \Rightarrow \]
\[ (\lambda(a, is'). \text{return\_pmf } (\text{real } (t_p (\text{fst } s) x a)))) = 0.5 \]
using assms apply(auto)
apply(simp_all add: type4_def BIT\_step_def bind_assoc\_pmf bind\_return\_pmf)
apply(simp_all add: E\_bernoulli3 t\_p_def)
done

lemma costBIT\_4y:
assumes $x \neq y$ : $\{x_0, y_0\}$, $y \in \{x_0, y_0\}$
shows
\[ E \left( \text{type4 } [x_0, y_0] \right) x y \quad \Rightarrow \]
\[ (\lambda s. \text{BIT\_step } s y \quad \Rightarrow \]
\[ (\lambda(a, is'). \text{return\_pmf } (\text{real } (t_p (\text{fst } s) y a)))) = 0.5 \]
using assms apply(auto)
apply(simp_all add: type4_def BIT\_step_def bind_assoc\_pmf bind\_return\_pmf)
}
apply(simp_all add: E_bernoulli3 t_p_def)
done

lemmas costBIT = costBIT_0x costBIT_0y costBIT_1x costBIT_1y costBIT_3x costBIT_3y costBIT_4x costBIT_4y

17.1.4 state transformation of BIT
abbreviation BIT_step s x == (s >>= (λs. BIT_step s x >>= (λ(a, is'). return_pmf (step (fst s) x a, is'))))

lemma oneBIT_step0x:
  assumes x≠y x : {x0,y0} y∈{x0,y0}
  shows BIT_step (type0 [x0, y0] x y) x = type0 [x0, y0] x y
  using assms
  apply(simp add: type0_def BIT_step_def bind_assoc_pmf bind_return_pmf step_def mtf2_def)
  apply(safe)
  apply(rule pmf_eqI) apply(simp add: pmf_bind_swap_def type0_def)
  apply(rule pmf_eqI) apply(simp add: commutet pmf_bind swap_def type0_def)
  done

lemma oneBIT_step0y:
  assumes x≠y x : {x0,y0} y∈{x0,y0}
  shows BIT_step (type0 [x0, y0] x y) y = type4 [x0, y0] x y
  using assms
  apply(simp add: type0_def BIT_step_def bind_assoc_pmf bind_return_pmf step_def mtf2_def)
  apply(safe)
  apply(rule pmf_eqI) apply(simp add: commutet pmf_bind swap_def type4_def)
  apply(rule pmf_eqI) apply(simp add: commutet pmf_bind swap_def type4_def)
  done

lemma oneBIT_step1x:
  assumes x≠y x : {x0,y0} y∈{x0,y0}
  shows BIT_step (type1 [x0, y0] x y) x = type0 [x0, y0] x y
  using assms
  apply(simp add: type1_def BIT_step_def bind_assoc_pmf bind_return_pmf step_def mtf2_def)
  apply(safe)
  apply(rule pmf_eqI) apply(simp add: pmf_bind_swap_def type0_def)
  apply(rule pmf_eqI) apply(simp add: commutet pmf_bind swap_def
lemma oneBIT_step1y:
  assumes \( x \neq y \): \( \{x_0, y_0\} \) \( y \in \{x_0, y_0\} \)
  shows \( \text{BIT}_\text{Step} (\text{type1} \ [x_0, y_0] \ x \ y) \ y = \text{type3} \ [x_0, y_0] \ x \ y \)
  using assms
  apply(simp add: type1_def BIT_step_def bind_assoc_pmf bind_return_pmf step_def mtf2_def)
  apply(safe)
  apply(rule pmf_eqI) apply(simp add: add.commute pmf_bind_swap_def type3_def)
  apply(rule pmf_eqI) apply(simp add: pmf_bind_swap_def type3_def)
  done

lemma oneBIT_step3x:
  assumes \( x \neq y \): \( \{x_0, y_0\} \) \( y \in \{x_0, y_0\} \)
  shows \( \text{BIT}_\text{Step} (\text{type3} \ [x_0, y_0] \ x \ y) \ x = \text{type1} \ [x_0, y_0] \ x \ y \)
  using assms
  apply(simp add: type3_def BIT_step_def bind_assoc_pmf bind_return_pmf step_def mtf2_def)
  apply(safe)
  apply(rule pmf_eqI) apply(simp add: pmf_bind_swap_def type1_def)
  apply(rule pmf_eqI) apply(simp add: add.commute pmf_bind_swap_def type1_def)
  done

lemma oneBIT_step3y:
  assumes \( x \neq y \): \( \{x_0, y_0\} \) \( y \in \{x_0, y_0\} \)
  shows \( \text{BIT}_\text{Step} (\text{type3} \ [x_0, y_0] \ x \ y) \ y = \text{type0} \ [x_0, y_0] \ y \ x \)
  using assms
  apply(simp add: type3_def BIT_step_def bind_assoc_pmf bind_return_pmf step_def mtf2_def)
  apply(safe)
  apply(rule pmf_eqI) apply(simp add: add.commute pmf_bind_swap_def type0_def)
  apply(rule pmf_eqI) apply(simp add: pmf_bind_swap_def type0_def)
  done

lemma oneBIT_step4x:
  assumes \( x \neq y \): \( \{x_0, y_0\} \) \( y \in \{x_0, y_0\} \)
  shows \( \text{BIT}_\text{Step} (\text{type4} \ [x_0, y_0] \ x \ y) \ x = \text{type1} \ [x_0, y_0] \ x \ y \)
  using assms
  apply(simp add: type4_def BIT_step_def bind_assoc_pmf bind_return_pmf
17.2 Analysis of the four phase forms

17.2.1 yx

```
lemma oneBIT_step4y:
  assumes \( x \neq y \) \( x \in \{ x_0, y_0 \} \)
  shows \( \text{BIT}_4 \text{Step} (\text{type}_4 \{ x_0, y_0 \} \ y \ x) = 0.75 \times \text{length } qs + T \text{p}_{\text{on}(\text{rand}' \text{BIT}_4 \text{Step}(\text{type}_4 \text{init} \ x \ y) \ qs \ @ \ r)} = \text{type}_0 \{ x_0, y_0 \} \ y \ x \)
proof
  from assms have \( qs \in \text{star}(\{[y]\} @@ \ {[x]\}) \) by (simp)
  from this assms show \( \text{thesis} \)
  proof (induct qs rule: star_induct)
    case (append u v)
    then have uyx: \( u = [y, x] \) by auto
    have \( yy: T \text{p}_{\text{on}(\text{rand}' \text{BIT}_4 \text{Step}(\text{type}_4 \text{init} \ x \ y) \ (v @ r)} = 0.75 \times \text{length } v + T \text{p}_{\text{on}(\text{rand}' \text{BIT}_4 \text{Step}(\text{type}_4 \text{init} \ x \ y) \ r}
    \wedge \text{config}' \text{rand} \text{BIT}_4 \text{Step}(\text{type}_4 \text{init} \ x \ y) \ qs \ = \ (\text{type}_1 \text{init} \ x \ y) \)
    apply (rule append(3))
    apply (fact)
  done
```

lemmas oneBIT_step = oneBIT_step0x oneBIT_step0y oneBIT_step1x oneBIT_step1y oneBIT_step3x oneBIT_step3y oneBIT_step4x oneBIT_step4y
using \texttt{append}(2,6) \textbf{by}(simp\_all)

have \(s2\): \texttt{config\_rand BIT (type1 init x y)} \([y,x] = (type1 init x y)\) using \texttt{kas assms(1)} \textbf{by} (auto simp add: oneBIT\_step )

have \(ta\): \(T_p\_on\_rand' BIT (type1 init x y) u = 1.5\) using \texttt{kas assms(1)} by (auto simp add: uyx oneBIT step costBIT 1y costBIT 3x)

have \(config\): \texttt{config\_rand BIT (type1 init x y)} (u@v) = (type1 init x y) by (simp only: \texttt{config\_rand_append s2 uyx yy})

have \(T_p\_on\_rand' BIT (type1 init x y) (u@v@r)\) = \(T_p\_on\_rand' BIT (type1 init x y) u\) + \(T_p\_on\_rand' BIT (type1 init x y) (v@r)\) by (simp only: \texttt{T\_on\_rand'\_append})

also have \(\ldots = T_p\_on\_rand' BIT (type1 init x y) u + 0.75*\text{length} v + T_p\_on\_rand' BIT (type1 init x y) r\) by (simp only: yy)

also have \(\ldots = 2*0.75 + 0.75*\text{length} v + T_p\_on\_rand' BIT (type1 init x y) r\) by (simp add: \texttt{ring\_distribs del: add_eq\_Suc' add_eq\_Suc})

also have \(\ldots = 0.75 * (2+\text{length} v) + T_p\_on\_rand' BIT (type1 init x y) r\) by (simp add: \texttt{ring\_distribs del: add_eq\_Suc' add_eq\_Suc})

also have \(\ldots = 0.75 * \text{length} (u@v) + T_p\_on\_rand' BIT (type1 init x y) r\)

using uyx by simp

finally show \(?case using config by simp\)

qed simp

qed

17.2.2 (yx)*yx

\textbf{lemma bit\_yxyx: assumes x \neq y and kas: init \in \{[x,y],[y,x]\} and qs \in lang (seq[\texttt{Times} (Atom y) (Atom x), \texttt{Star}(\texttt{Times} (Atom y) (Atom x))])\}

\textbf{shows} \(T_p\_on\_rand' BIT (type0 init x y) (qs@r) = 0.75 * \text{length} qs + T_p\_on\_rand' BIT (type1 init x y) r\)

\& \texttt{config\_rand BIT (type0 init x y) qs = (type1 init x y)}

\textbf{proof}\ –

\textbf{obtain} u v where \(uu: u \in \text{lang} (\text{\texttt{Times}} (\text{Atom y}) (\text{Atom x}))\)
and \( vv: v \in \text{lang}(\text{seq}[\text{Star}(\text{Times}(\text{Atom} \ y) \ (\text{Atom} \ x))]) \)
and \( qsuv: qs = u \otimes v \)

using \text{assms}(3) by (auto simp: conc_def)

from \( uu \) have \( uyx: u = [y, x] \) by (auto)

from \( qsuv \ uyx \) have \( vqs: \text{length} \ v = \text{length} \ qs - 2 \) by auto
from \( qsuv \ uyx \) have \( vqs2: \text{length} \ v + 2 = \text{length} \ qs \) by auto

have \( s2: \text{config}' \text{rand} \ BIT \ (\text{type0 init} \ x \ y) \ [y, x] = (\text{type1 init} \ x \ y) \)
using \text{kas assms}(1) by (auto simp add: oneBIT_step)

have \( ta: T_p \text{-on_rand}' \ BIT \ (\text{type0 init} \ x \ y) \ u = 1.5 \)
using \text{kas assms}(1) by (auto simp add: uyx oneBIT_step costBIT)

have \( tat: T_p \text{-on_rand}' \ BIT \ (\text{type1 init} \ x \ y) \ (v @ r) = 0.75*\text{length} \ v + T_p \text{-on_rand}' \ BIT \ (\text{type1 init} \ x \ y) \ r \)
\( \land \) \( \text{config}' \text{rand} \ BIT \ (\text{type1 init} \ x \ y) \ v = (\text{type1 init} \ x \ y) \)
apply (rule bit_yx)
apply (fact)
using \( vv \) by (simp_all)

have \( \text{config}: \text{config}' \text{rand} \ BIT \ (\text{type0 init} \ x \ y) \ (u @ v) = \text{type1 init} \ x \ y \)
by (simp only: \text{config}'\text{rand_append} \ s2 \ uyx \ tat)

have \( T_p \text{-on_rand}' \ BIT \ (\text{type0 init} \ x \ y) \ (u @ (v @ r)) = T_p \text{-on_rand}' \ BIT \ (\text{type0 init} \ x \ y) \ u + T_p \text{-on_rand}' \ BIT \ (\text{config}'\text{rand BIT} \ (\text{type0 init} \ x \ y) \ u) \ (v @ r) \) by (simp only: \text{T_on_rand'}\text{append})
also have \( \ldots = T_p \text{-on_rand}' \ BIT \ (\text{type0 init} \ x \ y) \ u + T_p \text{-on_rand}' \ BIT \ (\text{type1 init} \ x \ y) \ (v \otimes r) \) by (simp only: uyx \( s2 \))
also have \( \ldots = 2*0.75 + 0.75*\text{length} \ v + T_p \text{-on_rand}' \ BIT \ (\text{type1 init} \ x \ y) \ r \) by (simp only: tat)
also have \( \ldots = 2*0.75 + 0.75*\text{length} \ v + T_p \text{-on_rand}' \ BIT \ (\text{type1 init} \ x \ y) \ r \) by (simp add: ta)
also have \( \ldots = 0.75 * (2+\text{length} \ v) + T_p \text{-on_rand}' \ BIT \ (\text{type1 init} \ x \ y) \ r \)
by (simp add: ring_distrib del: \_add_2_eq_Suc \_add_2_eq_Suc)
also have \( \ldots = 0.75 * \text{length} \ (u @ v) + T_p \text{-on_rand}' \ BIT \ (\text{type1 init} \ x \ y) \ r \)
using \( uyx \) by simp
finally
  \( \text{show thesis using gsuv config by simp} \)
  qed

17.2.3 \( x^+.. \)

\textbf{lemma BIT_a: assumes} \( x \neq y \)
  \( \text{init} \in \{[x,y],[y,x]\} \)
  \( qs \in \text{lang (Plus (Atom } x \text{) One)} \)
\( \text{shows } T_{p\cdot\text{on} \cdot \text{rand}'} \text{BIT (type0 init } x \text{ y) \ (qs}@r) = T_{p\cdot\text{on} \cdot \text{rand}'} \text{BIT (type0 init } x \text{ y) } \)
  \( \wedge \text{config'} \cdot \text{rand} \text{BIT (type0 init } x \text{ y) qs = (type0 init } x \text{ y)} \)
\( \text{proof --} \)
  \( \text{have } s: \text{config'} \cdot \text{rand} \text{BIT (type0 init } x \text{ y) qs = type0 init } x \text{ y) using assms by (auto simp add: oneBIT_step)} \)
  \( \text{have } t: T_{p\cdot\text{on} \cdot \text{rand}'} \text{BIT (type0 init } x \text{ y) qs = 0 using assms by (auto simp add: costBIT)} \)
\( \text{show thesis using } s \ t \text{ by(simp add: T_on_rand'} \cdot \text{append)} \)
  qed

17.2.4 \( \text{Phase Form A}\)

\textbf{lemma BIT_a: assumes} \( x \neq y \)
  \( \text{init} \in \{[x,y],[y,x]\} \)
  \( qs \in \text{lang (seq [Plus (Atom } x \text{) One, Atom y, Atom y])} \)
\( \text{shows } \text{config'} \cdot \text{rand} \text{BIT (type0 init } x \text{ y) qs = (type0 init } x \text{ y) (is } ?C)} \)
  \( \wedge \text{and b: } T_{p\cdot\text{on} \cdot \text{rand}'} \text{BIT (type0 init } x \text{ y) qs = 1.5 (is } ?T)} \)
\( \text{proof --} \)
  \( \text{from assms(3) have alt: qs = [x,y,y] } \lor \text{ qs = [y,y] apply(simp) by fastforce} \)
  \( \text{show } ?C \)
  \( \text{using assms(1,2) alt by (auto simp add: oneBIT_step)} \)
  \( \text{show } ?T \)
  \( \text{using assms(1,2) alt by(auto simp add: oneBIT_step costBIT)} \)
  qed

\textbf{lemma bit_a: assumes} \( x \neq y \}
  \( \{x, y\} = \{x0, y0\} \)
  \( \text{BIT_inv s x [x0, y0]} \)
  \( \text{set qs } \subseteq \{x, y\} \)
  \( qs \in \text{lang (seq [Plus (Atom } x \text{) One, Atom y, Atom y])} \)
\( \text{shows } \)
  \( T_{p\cdot\text{on} \cdot \text{rand}'} \text{BIT s qs } \leq \ 1.75 \ast T_{p \ [x,y]} \)
  \( \text{qs (OPT2 qs [x,y])} \)
  \( \wedge \text{BIT_inv (config'} \cdot \text{rand} \text{BIT s qs) (last qs) [x0, y0]} \)
  \( \wedge T_{p\cdot\text{on} \cdot \text{rand}'} \text{BIT s qs = 1.5} \)
proof

from assms have f: x0 ≠ y0 by auto
from assms(1,3) assms(2)[symmetric] have s: s = type0 [x0,y0] x y
apply(simp add: BIT_inv2[OF f] other_def) by fast

from assms(1,2) have kas: [x,y] = [x0,y0] ∨ [x,y] = [y0,x0] by auto

from assms have lqs: last qs = y by fastforce
from assms kas have p: Tp_on_rand' BIT s qs = 1.5
unfolding s
apply(safe)
apply(rule BIT_a)
apply(simp) apply(simp) using assms(5) apply(simp)
apply(rule BIT_a)
apply(simp) apply(simp) using assms(5) apply(simp)
done

with OPT2_A[OF assms(1,5)] have BIT: Tp_on_rand' BIT s qs ≤ 1.75
* Tp [x, y] qs (OPT2 qs [x, y]) by auto

from assms kas have config_rand BIT s qs = type0 [x0, y0] y x
unfolding s
apply(safe)
apply(rule BIT_a)
apply(simp) apply(simp) using assms(5) apply(simp)
apply(rule BIT_a)
apply(simp) apply(simp) using assms(5) apply(simp)
done

then have BIT_inv (config_rand BIT s qs) (last qs) [x0, y0]
apply(simp)
using assms(1) kas f lqs by(auto simp add: BIT_inv2 other_def)

then show ?thesis using BIT s p by simp
qed

lemma bit_a": a ≠ b ⟹
{a, b} = {x, y} ⟹
BIT_inv s a [x, y] ⟹
set qs ⊆ {a, b} ⟹
qs ∈ lang (seq [question (Atom a), Atom b, Atom b]) ⟹
BIT_inv (Partial_Cost_Model.config_rand BIT s qs) (last qs) [x, y]
∧ Tp_on_rand' BIT s qs = 1.5
using bit_a[of a b x y] by blast
17.2.5 Phase Form B

lemma BIT,b: assumes A: x ≠ y
  init ∈ {x,y,y,x}
  v ∈ lang (seq [Times (Atom y) (Atom x), Star (Times (Atom y) (Atom x)), Atom y, Atom y])
  shows T_{p\cdot on\cdot rand} BIT (type0 init x y) v = 0.75 * length v − 0.5 (is ?T)
  and config\',rand BIT (type0 init x y) v = (type0 init y x) (is ?C)
proof —
  have lenmod: length v mod 2 = 0
  proof —
    from assms(3) have v ∈ (\{\} \ @@ \ \{x\}) \ @@ \ star(\{\} \ @@ \ \{x\})
    \@@ \ \{\} \ @@ \ \{\} by(simp add: conc_assoc)
    then obtain p q r where pqr: v = p\@q\@r and p\in(\{\} \ @@ \ \{x\})
    and q: q\in star (\{\} \ @@ \ \{x\}) and r\in(\{\} \ @@ \ \{y\}) by (metis concE)
    then have p = [y,x] r = [y,y] by auto
    with pqr have a: length v = 4 + length q by auto
  from q have b: length q mod 2 = 0
  apply(induct q rule: star_induct) by (auto)
  from a b show length v mod 2 = 0 by auto
qed

from assms(3) have v ∈ lang (seq[Times (Atom y) (Atom x), Star(Times (Atom y) (Atom x))])
  \ @@ \ lang (seq[Atom y, Atom y]) by (auto simp: conc_def)
  then obtain a b where aa: a \in lang (seq[Times (Atom y) (Atom x),
  Star(Times (Atom y) (Atom x))])
  and b \in lang (seq[Atom y, Atom y])
  and vab: v = a \@ b
  by(erule concE)
  then have bb: b = [y,y] by auto
  from vab bb have lenv: length v = length a + 2 by auto

from bit_gxyz[OF assms(1,2) aa] have stars: T_{p\cdot on\cdot rand} BIT (type0 init x y) (a \@ b) = 0.75 * length a + T_{p\cdot on\cdot rand} BIT (type1 init x y) b
  and s2: config\',rand BIT (type0 init x y) a = type1
init x y by fast+

have t: T_{p\cdot on\cdot rand} BIT (type1 init x y) b = 1
  using assms(1,2) by (auto simp add: oneBIT_step costBIT bb)
have \( s : \text{config}'\text{rand} \text{BIT} \ (\text{type1 init} \ x \ y) \ [y, y] = \text{type0 init} \ y \ x \)
using \( \text{assms}(1, 2) \) by (auto simp add: oneBIT_step)

have \( \text{config} : \text{config}'\text{rand} \text{BIT} \ (\text{type0 init} \ y \ x) \ (a \ @ \ b) = \text{type0 init} \ y \ x \)
by (simp only: \text{config}'\text{rand_append} \ s2 \ vab \ bb \ s)

have \( \text{calc} : 3 \ast \text{Suc} \ (\text{Suc} \ (\text{length} \ a)) / 4 - 1 / 2 = 3 \ast (2+\text{length} \ a) / 4 - 1 / 2 \) by simp

from \( t \ \text{stars} \) have \( \text{T}_{p\text{-on_rand'}} \text{BIT} \ (\text{type0 init} \ x \ y) \ (a \ @ \ b) = 0.75 \ast \text{length} \ a + 1 \) by auto
then show \( \text{T}_{p\text{-on_rand'}} \text{BIT} \ (\text{type0 init} \ x \ y) \ v = 0.75 \ast \text{length} \ v - 0.5 \)
unfolding lenv by (simp add: vab ring_distrib del: add_2_eq_Suc)
from \( \text{config} \ vab \) show \( ?C \) by simp
qed

lemma \text{bitb''1}: assumes \( x \neq y \ {\{x, y\}} = \{x0, y0\} \) \( \text{BIT}_\text{inv} \ s \ x \ [x0, y0] \)
set \( qs \subseteq {\{x, y\}} \)
\( qs \in \text{lang} \ (\text{seq}\{\text{Atom} \ y, \text{Atom} \ x, \text{Star}(\text{Times} (\text{Atom} \ y) (\text{Atom} \ x)), \text{Atom} \ y, \text{Atom} \ y)\})
shows \( \text{BIT}_\text{inv} \ (\text{config}'\text{rand} \text{BIT} \ s \ qs) \ (\text{last} \ qs) \ [x0, y0] \land \\
\text{T}_{p\text{-on_rand'}} \text{BIT} \ s \ qs = 0.75 \ast \text{length} \ qs - 0.5 \)
proof –
from \( \text{assms} \) have \( f : x0 \neq y0 \) by auto
from \( \text{assms}(1, 3) \) \( \text{assms}(2)\) [symmetric] have \( s : s = \text{type0} \ [x0, y0] \ x \ y \)
apply (simp add: \( \text{BIT}_\text{inv2}(\text{OF} \ f) \) other_def) by fast
from \( \text{assms}(1, 2) \) have \( \text{kas} : [x, y] = [x0, y0] \lor [x, y] = [y0, x0] \) by auto

from \( \text{assms}(5) \) have \( \text{lqs} : \text{last} \ qs = y \) by fastforce
from \( \text{assms}(1, 2) \) \( \text{kas} \) have \( \text{BIT} : \text{T}_{p\text{-on_rand'}} \text{BIT} \ s \ qs = 0.75 \ast \text{length} \ qs - 0.5 \)
unfolding \( s \)
apply (safe)
apply (rule \( \text{BIT}_b \))
apply (simp) apply (simp) using \( \text{assms}(5) \) apply (simp add: conc_assoc)
apply (rule \( \text{BIT}_b \))
apply (simp) apply (simp) using \( \text{assms}(5) \) apply (simp add: conc_assoc)
done

from \( \text{assms}(1, 2) \) \( \text{kas} \) have \( \text{config}'\text{rand} \text{BIT} \ s \ qs = \text{type0} \ [x0, y0] \ y \ x \)
unfolding $s$
apply(safe)
  apply(rule BIT_b)
apply(simp) apply(simp) using assms(5) apply(simp add: conc_assoc)
apply(rule BIT_b)
apply(simp) apply(simp) using assms(5) apply(simp add: conc_assoc)
done

then have config: BIT_inv (config'_rand BIT s qs) (last qs) $[x_0, y_0]$ apply simp using assms(1) kas lqs by (auto simp add: BIT_inv2 other_def)

show ?thesis using BIT config by simp
qed

lemma BIT_b2: assumes A: $x \neq y$
  init $\in \{[x,y],[y,x]\}$
  $v \in$ lang (seq [Atom x, Times (Atom y) (Atom x), Star (Times (Atom y) (Atom x)), Atom y, Atom y])
  shows $T_{p'_on\_rand'}$ BIT (type0 init x y) $v = 0.75 \times (\text{length} \ v - 1) - 0.5$ (is ?T)
  and config'_rand BIT (type0 init x y) $v = (\text{type0 init y x})$ (is ?C)
proof –
  from assms(3) obtain $w$ where $vw: v = [x]@w$ and
    $w: w \in$ lang (seq [Times (Atom y) (Atom x), Star (Times (Atom y) (Atom x)), Atom y, Atom y])
    by (auto)
  have c1: config'_rand BIT (type0 init x y) $[x] = \text{type0 init x y}$
    using assms by (auto simp add: oneBIT_step)
  have t1: $T_{p'_on\_rand'}$ BIT (type0 init x y) $[x] = 0$
    using assms by (auto simp add: costBIT)
  show $T_{p'_on\_rand'}$ BIT (type0 init x y) $v$
    $= 0.75 \times (\text{length} \ v - 1) - 0.5$
    unfolding $vw$ apply(simp only: $T_{on\_rand'}$append c1 BIT_b[OF assms(1,2) w] t1)
    by (simp)
  show config'_rand BIT (type0 init x y) $v = (\text{type0 init y x})$
    unfolding $vw$ by (simp only: config'_rand_append c1 BIT_b[OF assms(1,2) w])
qed

lemma bit_b''2: assumes
  $x \neq y \\{x, y\} = \{x_0, y_0\}$ BIT_inv s $x \{x_0, y_0\}$
set qs ⊆ \{x, y\}
qs ∈ lang (seq[Atom x, Atom y, Atom x, Star(Times (Atom y) (Atom x)), Atom y, Atom y])

shows $BIT_{\text{inv}} (\text{config}'_{\text{rand}} BIT s qs) (\text{last } qs) [x0, y0] \land$

$T_{p,\text{on}_{\text{rand}'} BIT s qs} = 0.75 \times (\text{length } qs - 1) - 0.5$

proof –
from assms have \( f : x0 \neq y0 \) by auto
from assms(1,3) assms(2)[symmetric] have \( s : s = \text{type0} [x0,y0] x y \)
apply(simp add: BIT_inv2[OF \( f \)] other_def) by fast

from assms(1,2) have \( \text{kas} : [x,y] = [x0,y0] \lor [x,y] = [y0,x0] \) by auto

from assms(5) have \( \text{lqs} : \text{last } qs = y \) by fastforce
from assms(1,2) kas have \( \text{BIT: } T_{p,\text{on}_{\text{rand}'} \text{BIT} s qs} = 0.75 \times (\text{length } \text{qs-1}) - 0.5 \)

unfolding \( s \)
apply(safe)
apply(rule BIT_b2)
apply(simp) apply(simp) using assms(5) apply(simp add: conc_assoc)
apply(rule BIT_b2)
apply(simp) apply(simp) using assms(5) apply(simp add: conc_assoc)
done

from assms(1,2) kas have \( \text{config}'_{\text{rand}} \text{BIT s qs} = \text{type0} [x0, y0] y x \)

unfolding \( s \)
apply(safe)
apply(rule BIT_b2)
apply(simp) apply(simp) using assms(5) apply(simp add: conc_assoc)
apply(rule BIT_b2)
apply(simp) apply(simp) using assms(5) apply(simp add: conc_assoc)
done

then have \( \text{config: } BIT_{\text{inv}} (\text{config}'_{\text{rand}} BIT s qs) (\text{last } qs) [x0, y0] \)
apply(simp)
using assms(1) kas lqs by(auto simp add: BIT_inv2 other_def)

show \( \text{?thesis using } BIT \text{ config by simp} \)
qed

lemma bit_b: assumes \( x \neq y \)
init ∈ \{[x,y],[y,x]\}
qs ∈ lang (seq[Plus (Atom x) One, Atom y, Atom x, Star(Times (Atom y) (Atom x)), Atom y, Atom y])

shows \( T_{p,\text{on}_{\text{rand}'} BIT (\text{type0 init } x y) \text{ qs} \leq 1.75 \times T_{p} [x,y] \text{ qs} (\text{OPT2} \} \)

350
\(qs \ [x, y]\)

and  \texttt{config'\_rand} \texttt{BIT} (\texttt{type0 init} \ x \ y) \ qs = \texttt{type0 init} \ y \ x

proof –

obtain \(u \ v\) where \(uu: u \in \texttt{lang} (\texttt{Plus} (\texttt{Atom} \ x) \ \texttt{One})\)

and \(vv: v \in \texttt{lang} (\texttt{seq}[\texttt{Times} (\texttt{Atom} \ y) \ (\texttt{Atom} \ x), \ \texttt{Star} (\texttt{Times} (\texttt{Atom} y) \ (\texttt{Atom} \ x)), \ \texttt{Atom} \ y, \ \texttt{Atom} \ y])\)

and \(qsuv: qs = u \ @ \ v\)

using \texttt{assms}

by (\texttt{auto simp: conc\_def})

have \(\texttt{lenv} : \texttt{length} \ v \ \texttt{mod} \ 2 = 0 \ \land \ \texttt{last} \ v = y \ \land \ \texttt{v} \neq []\)

proof –

from \(vv\) have \(v \in ([\{\texttt{y}\}] \ @\ @ \{[\texttt{x}]\}) \ @\ @ \star \{([\texttt{y}] \ @\ @ \{[\texttt{x}]\}) \ @\ @ \{[\texttt{y}]\}\)

@\ @ \{[\texttt{y}]\}\) by \texttt{simp}

then obtain \(p \ q \ r\) where \(qpr: v = p \@ q \@ r\) and \(p \in (\{\texttt{y}\} \ @\ @ \{[\texttt{x}]\})\)

and \(q: q \in \texttt{star} (\{\texttt{y}\} \ @\ @ \{[\texttt{x}]\})\) and \(r \in \{\texttt{y}\} \ @\ @ \{[\texttt{y}]\}\) by (\texttt{metis concE})

then have \(rr: p = [\texttt{y}, \texttt{x}]\) \(r = [\texttt{y}, \texttt{y}]\) by \texttt{auto}

with \(qpr\) have \(a: \texttt{length} \ v = 4 + \texttt{length} \ q\) by \texttt{auto}

have \(\texttt{last} \ v = \texttt{last} \ r\) using \(qpr\) \(rr\) by \texttt{auto}

then have \(c: \texttt{last} \ v = y\) using \(rr\) \(by\) \texttt{auto}

from \(q\) have \(b: \texttt{length} \ q \ \texttt{mod} \ 2 = 0\)

apply(\texttt{induct} \ q \ \texttt{rule}: \texttt{star\_induct}) \texttt{by} \texttt{(auto)}

from \(a \ b \ c\) show \(\texttt{thesis}\) \texttt{by} \texttt{auto}

qed

from \(vv\) have \(v \in \texttt{lang} (\texttt{seq}[\texttt{Times} (\texttt{Atom} \ y) \ (\texttt{Atom} \ x), \ \texttt{Star} (\texttt{Times} (\texttt{Atom} y) \ (\texttt{Atom} \ x))]\)

@\ @ \texttt{lang} (\texttt{seq}[\texttt{Atom} \ y, \ \texttt{Atom} \ y]) \texttt{by} \texttt{(auto simp: conc\_def)}

then obtain \(a \ b\) where \(aa: a \in \texttt{lang} (\texttt{seq}[\texttt{Times} (\texttt{Atom} \ y) \ (\texttt{Atom} \ x), \ \texttt{Star} (\texttt{Times} (\texttt{Atom} y) \ (\texttt{Atom} \ x))]\)

and \(b \in \texttt{lang} (\texttt{seq}[\texttt{Atom} \ y, \ \texttt{Atom} \ y])\)

and \(vab: v = a \ @ \ b\)

by(\texttt{erule concE})

from \(\texttt{BIT}_x[\texttt{OF} \ \texttt{assms}(1, 2) \ uu]\) have \(u_{-t}: T_{p\_on\_rand'} \texttt{BIT} (\texttt{type0 init} \ x \ y) \ (a \ @ \ v) = T_{p\_on\_rand'} \texttt{BIT} (\texttt{type0 init} \ x \ y) \ v\)

and \(u_{-c}: \texttt{config'\_rand} \texttt{BIT} (\texttt{type0 init} \ x \ y) \ u = \texttt{type0 init} \ x \ y\) \texttt{by} \texttt{auto}

from \(\texttt{BIT}_b[\texttt{OF} \ \texttt{assms}(1, 2) \ vv]\) have \(b_{-t}: T_{p\_on\_rand'} \texttt{BIT} (\texttt{type0 init} \ x \ y) \ v = 0.75 \ast \texttt{length} \ v - 0.5\)

and \(b_{-c}: \texttt{config'\_rand} \texttt{BIT} (\texttt{type0 init} \ x \ y) \ v = (\texttt{type0 init} \ y \ x)\) \texttt{by} \texttt{auto}
have BIT: \( T_{p\cdot on\_rand'} BIT \) (type0 init \( x \, y \)) \( qs = 0.75 \times \text{length } v - 0.5 \)
by (simp add: qsuv u_t b_t)

from uu have uu: \( u = \[] \vee u = [x] \) by auto
have OPT: \( T_p [x, y] \) qs (OPT2 qs [x, y]) = \( (\text{length } v) \div 2 \)
apply (rule OPT2_B) by (fact)+

from lenv have \( v \neq \[] \) last \( v = y \) by auto
then have 1: last qs = y using last_appendR qsuv by simp
then have 2: other (last qs) \( x = y \) unfolding other_def by simp

show \( T_{p\cdot on\_rand'} BIT \) (type0 init \( x \, y \)) \( qs \leq 1.75 \times T_p [x, y] \) qs (OPT2 qs [x, y])
using BIT OPT lenv by auto

show config'rand BIT (type0 init \( x \, y \)) \( qs = \text{type0 init } y \) \( x \)
by (auto simp add: config'_rand_append qsuv u_c b_c)
qed

lemma bit_b': assumes
\( x \neq y \) \( \{x, y\} = \{x0, y0\} \) BIT_inv s \( x \) \( x0, y0 \)
set \( qs \subseteq \{x, y\} \)
qs \( \in \) lang (seq) Plus (Atom x) One, Atom y, Atom x, Star(Times (Atom y) (Atom x)), Atom y, Atom y)
shows \( T_{p\cdot on\_rand'} BIT \) qs \( s \leq 1.75 \times T_p [x, y] \) qs (OPT2 qs [x, y])
\( \wedge \) BIT_inv (config'_rand BIT s qs) (last qs) \( [x0, y0] \)
proof –
from assms have \( f: x0 \neq y0 \) by auto
from assms(1,3) assms(2)[symmetric] have s: \( s = \text{type0 } [x0, y0] \) \( x \) \( y \)
apply (simp add: BIT_inv2[OF f] other_def) by fast

from assms(1,2) have kas: \( [x, y] = [x0, y0] \vee [x, y] = [y0, x0] \) by auto

from assms(5) have lqs: last qs = \( y \) by fastforce
from assms(1,2) kas have BIT: \( T_{p\cdot on\_rand'} BIT \) s qs \( \leq 1.75 \times T_p [x, y] \) qs (OPT2 qs [x, y])
unfolding $s$
apply(safe)
  apply(rule bit_b)
  apply(simp) apply(simp) using assms(5) apply(simp)
apply(rule bit_b)
  apply(simp) apply(simp) using assms(5) apply(simp)
done

from assms(1,2) kas have config'_rand BIT $s$ qs = type0 $[x_0, y_0]$ $y$ $x$
unfolding $s$
apply(safe)
  apply(rule bit_b)
  apply(simp) apply(simp) using assms(5) apply(simp)
apply(rule bit_b)
  apply(simp) apply(simp) using assms(5) apply(simp)
done

then have BIT_inv (config'_rand BIT $s$ qs) (last qs) $[x_0, y_0]$
  apply(simp)
  using assms(1) kas lqs by (auto simp add: BIT_inv2 other_def)
then show ?thesis using BIT $s$ by simp
qed

lemma bit_b'': $a \neq b \implies$
  \{$a, b\} = \{x, y\} \implies$
  BIT_inv $s$ $a$ $[x, y] \implies$
  set $qs \subseteq \{a, b\} \implies$
  $qs \in$ lang $[\text{seq[Plus (Atom x) One, Atom y, Atom x, Star(Times (Atom y) (Atom x)), Atom y, Atom y]} \implies$
  BIT_inv $\text{Partial_Cost_Model.config'_rand BIT s qs}$ (last qs) $[x, y]$
\land T_{p\_on\_rand'} BIT $s$ qs = 1.5
using bit_a[of $a$ $b$ $x$ $y$] oops

17.2.6 Phase Form C

lemma BIT_c: assumes $x \neq y$
  init $\in \{[x,y],[y,x]\}$
  \(v \in \text{lang} [\text{seq[Times (Atom y) (Atom x), Star (Times (Atom y) (Atom x)), Atom x]}], \text{Atom x}]\)
  shows $T_{p\_on\_rand'} BIT$ (type0 init $x$ $y$) $v = 0.75 * \text{length} v - 0.5$
  and config'_rand BIT (type0 init $x$ $y$) $v = (\text{type0 init x y})$ (is ?C)
proof –
  have $A$: $x \neq y$ using assms by auto
from assms(3) have \( v \in \text{lang}(\text{seq}(\text{Times}(\text{Atom} y)(\text{Atom} x), \text{Star}(\text{Times}(\text{Atom} y)(\text{Atom} x)))) \)

@@ \text{lang}(\text{seq}(\text{Atom} x)) \text{ by (auto simp: conc_def)}

then obtain \( a \ b \) where \( aa: a \in \text{lang}(\text{seq}(\text{Times}(\text{Atom} y)(\text{Atom} x), \text{Star}(\text{Times}(\text{Atom} y)(\text{Atom} x)))) \)

and \( b \in \text{lang}(\text{seq}(\text{Atom} x)) \)

and \( vab: v = a @ b \)

by (erule concE)

then have \( bb: b = [x] \) by auto

from \( aa \) have \( lena: \text{length} \ a > 0 \) by auto

from \( vab \ bb \) have \( lenv: \text{length} \ v = \text{length} \ a + 1 \) by auto

from bit_yxxy assms(1,2) \( aa \) have stars: \( T_{p, \text{on\_rand}'} \text{BIT (type0 init} x y) \) \((a @ b) = 0.75 \ast \text{length} \ a + T_{p, \text{on\_rand}'} \text{BIT (type1 init} x y) \) \( b \)

and \( s2: \text{config'} \text{rand BIT (type0 init} x y) \) \( a = \text{type1 init} x y \)

init \( x y \) by fast+

have \( t: T_{p, \text{on\_rand}'} \text{BIT (type1 init} x y) \) \( b = 1/4 \)

using assms(1,2) by (auto simp add: bb costBIT)

have \( s: \text{config'} \text{rand BIT (type1 init} x y) \) \( b = \text{type0 init} x y \)

using assms(1,2) by (auto simp add: bb oneBIT_step1x)

have \( \text{config: config'} \text{rand BIT (type0 init} x y) \) \((a @ b) = \text{type0 init} x y \)

by (simp only: config'rand_append s2 vab s)

have \( \text{calc: 3} \ast \text{Suc} (\text{Suc} (\text{length} \ a))) / 4 - 1 / 2 = 3 \ast (2 + \text{length} \ a) / 4 - 1 / 2 \) by simp

from \( t \) stars have \( T_{p, \text{on\_rand}'} \text{BIT (type0 init} x y) \) \((a @ b) = 0.75 \ast \text{length} \ a + 1/4 \) by auto

then show \( T_{p, \text{on\_rand}'} \text{BIT (type0 init} x y) \) \( v = 0.75 \ast \text{length} \ v - 0.5 \)

unfolding \( lenv \)

by (simp add: vab ring_distrib del: add_2_eq_Suc')

from \( \text{config vab} \) show \(?C \) by simp

qed

lemma bit_c''1: assumes
\( x \neq y \ \{x, y\} = \{x0, y0\} \text{ BIT_inv} s \ [x0, y0] \)

set \( qs \subseteq \{x, y\} \)

\( qs \in \text{lang}(\text{seq}(\text{Atom} y, \text{Atom} x, \text{Star}(\text{Times}(\text{Atom} y)(\text{Atom} x)), \text{Atom}) \)

354
shows $BIT\_inv\ (config\_\'\ rand\ BIT\ s\ qs)\ (last\ qs)\ [x0, y0] \land T_{p\_on\_rand'}\ BIT\ s\ qs = 0.75 \ast length\ qs - 0.5$

proof –
from assms have $f: x0 \neq y0$ by auto
from assms(1,3) assms(2)[symmetric] have $s: s = type0\ [x0,y0]\ x\ y$
apply(simp add: $BIT\_inv2[OF\ f]$ other_def) by fast

from assms(1,2) have kas: $[x,y] = [x0,y0] \lor [x,y] = [y0,x0]$ by auto
from assms(5) have $qs: last\ qs = x$ by fastforce
from assms(1,2) kas have $BIT: T_{p\_on\_rand'}\ BIT\ s\ qs = 0.75 \ast length\ qs - 0.5$
unfolding $s$
apply(safe)
apply(rule $BIT\_c$)
apply(simp) apply(simp) using assms(5) apply(simp add: conc_assoc)
apply(rule $BIT\_c$)
apply(simp) apply(simp) using assms(5) apply(simp add: conc_assoc)
done

from assms(1,2) kas have config\_\'\ rand\ BIT\ s\ qs = type0\ [x0, y0]\ x\ y
unfolding $s$
apply(safe)
apply(rule $BIT\_c$)
apply(simp) apply(simp) using assms(5) apply(simp add: conc_assoc)
apply(rule $BIT\_c$)
apply(simp) apply(simp) using assms(5) apply(simp add: conc_assoc)
done

then have config: $BIT\_inv\ (config\_\'\ rand\ BIT\ s\ qs)\ (last\ qs)\ [x0, y0]$
apply(simp)
using assms(1) kas qs by(auto simp add: $BIT\_inv2\ other\_def$)

show $\theta$thesis using $BIT\ config$ by simp
qed

lemma $bit\_c$: assumes $x \neq y$
  \begin{itemize}
    \item $init \in \{[x,y],[y,x]\}$
    \item $qs \in lang\ (seq(Plus\ (Atom\ x)\ One,\ Atom\ y,\ Atom\ x,\ Star(Times\ (Atom\ y)\ (Atom\ x)),\ Atom\ x))$
  \end{itemize}
shows $T_{p\_on\_rand'}\ BIT\ (type0\ init\ x\ y)\ qs \leq 1.75 \ast T_p\ [x,y]\ qs\ (OPT2\ qs\ [x,y])$
and config\_\'\ rand\ BIT\ (type0\ init\ x\ y)\ qs = type0\ init\ x\ y
proof -

obtain \( u \, v \) where \( u \in \text{lang} (\text{Plus} (\text{Atom} x) \text{ One}) \)
and \( v \in \text{lang} (\text{seq} [\text{Times} (\text{Atom} y) (\text{Atom} x), \text{Star} (\text{Times} (\text{Atom} y) (\text{Atom} x))]) \)
and \( \text{qsuv} : \text{qs} = u \, @ \, v \)
using assms

by (auto simp:conc_def)

have \( \text{lenv} : \text{length} \, v \, \text{mod} \, 2 = 1 \, \land \, \text{length} \, v \geq 3 \, \land \, \text{last} \, v = x \)

proof -
from vv have \( v \in (([y] \ \text{@@} \ {x}) \ \text{@@} \ \text{star} ([y] \ \text{@@} \ {x}) \ \text{@@} \ {x}) \)
by auto

then obtain \( p \ q \ r \) where \( pqr : v = p \, @ \, q \, @ \, r \)
and \( q : q \in \text{star} ([y] \ \text{@@} \ {x}) \)
and \( r \in [x] \)
by (metis concE)

then have \( \text{rr} : p = \text{[y, x]} \ r = \text{[x]} \)
by auto

have \( \text{last} \, v = \text{last} \, r \)
using \( pqr \, \text{rr} \) by auto

then have \( \text{c} : \text{last} \, v = \text{x} \)

then have \( \text{c} : \text{last} \, v = \text{x} \)
using \( \text{rr} \) by auto

apply (induct \( q \) rule: star_induct) by (auto)

from \( a \ b \ c \) show \( \text{length} \, v \, \text{mod} \, 2 = 1 \, \land \, \text{length} \, v \geq 3 \, \land \, \text{last} \, v = x \)
by auto

qed
by (simp add: qsuv u_t b_t)

from uu have uu: u=[] \or u=[x] by auto
from vv have vv: v \in lang (seq
    Atom y, Atom x,
    Star (Times (Atom y) (Atom x)),
    Atom x) by (auto simp: conc_def)

have OPT: T_p [x,y] qs (OPT2 qs [x,y]) = (length v) div 2 apply (rule OPT2_C) by (fact)+

from lenv have v \neq [] last v = x by auto
then have 1: last qs = x using last_appendR qsuv by simp
then have 2: other (last qs) x y = y unfolding other_def by simp

have vgt3: length v \geq 3 using lenv by simp
have T_p_on_rand' BIT (type0 init x y) qs = 0.75 * length v - 0.5 using BIT by simp
also have ... \leq 1.75 * (length v - 1)/2
proof –
    have 10 + 6 * length v \leq 7 * Suc (length v)
        \iff 10 + 6 * length v \leq 7 * length v + 7 by auto
    also have ... \iff 3 \leq length v by auto
    also have ... \iff True using vgt3 by auto
finally have A: 6 * length v - 4 \leq 7 * (length v - 1) by simp
show ?thesis apply (simp) using A by linarith
qed
also
have ... = 1.75 * (length v div 2)
proof –
    from div_mult_mod_eq have length v = length v div 2 * 2 + length v mod 2 by auto
    with lenv have length v = length v div 2 * 2 + 1 by auto
    then have (length v - 1) / 2 = length v div 2 by simp
    then show ?thesis by simp
qed
also
have ... = 1.75 * T_p [x, y] qs (OPT2 qs [x, y]) using OPT by auto
finally
show \( T_{p\cdot on\cdot rand'} \ BIT \ (type0 \ init \ x \ y) \ qs \leq 1.75 * T_p \ [x,y] \ qs \ (OPT2 qs \ [x,y]) \)

using \( BIT \ OPT \ lenv \ 1 \ 2 \) by auto

show \( \text{config'}_\text{rand} \ BIT \ (type0 \ init \ x \ y) \ qs = type0 \ init \ x \ y \)
by (auto simp add: \( \text{config'}_\text{rand} \_\text{append} \) \( qsuv u_c b_c \))

qed

lemma \( \text{bit'}_c'' \): assumes \( x \neq y \ \{x, y\} = \{x_0, y_0\} \ BIT_{\text{inv}} s \ [x_0, y_0] \)
set \( qs \subseteq \{x, y\} \)
qs \in \text{lang} (\text{seq}[\text{Plus} (\text{Atom} x) \ \text{One}, \ \text{Atom} y, \ \text{Atom} x, \ \text{Star}(\text{Times} (\text{Atom} y)) \ (\text{Atom} x)), \ \text{Atom} x])

shows
\( T_{p\cdot on\cdot rand'} \ BIT \ s \ qs \leq 1.75 * T_p \ [x,y] \ qs \ (OPT2 qs \ [x,y]) \)
\wedge \ BIT_{\text{inv}} (\text{config'}_\text{rand} \ BIT \ s \ qs) (\text{last} qs) [x_0, y_0]

proof –
from assms have \( f: x0\neq y0 \) by auto
from assms(1,3) assms(2)[symmetric] have \( s = type0 \ [x0,y0] \ x \ y \)
apply(simp add: BIT_{\text{inv2}}[OF \ f] other_def) by fast

from assms(1,2) have \( \text{kas}: [x,y] = [x0,y0] \lor [x,y] = [y0,x0] \) by auto

from assms have \( \text{lqs}: \text{last} qs = x \) by fastforce
from assms(1,2) kas have \( BIT: T_{p\cdot on\cdot rand'} \ BIT \ s \ qs \leq 1.75 * T_p \ [x, y] \ qs \ (OPT2 qs \ [x, y]) \)

unfolding \( s \)
apply(safe)
apply(rule \( \text{bit'}_c' \))
apply(simp) apply(simp) using assms(5) apply(simp)
apply(rule \( \text{bit'}_c' \))
apply(simp) apply(simp) using assms(5) apply(simp)
done

from assms(1,2) kas have \( \text{config'}_\text{rand} \ BIT \ s \ qs = type0 \ [x0, y0] \ x \ y \)
unfolding \( s \)
apply(safe)
apply(rule \( \text{bit'}_c' \))
apply(simp) apply(simp) using assms(5) apply(simp)
apply(rule \( \text{bit'}_c' \))
apply(simp) apply(simp) using assms(5) apply(simp)
done

358
then have \(\text{BIT}_{\text{inv}}(\text{config}_\text{rand} \text{ BIT} s \text{ qs}) (\text{last qs}) [x0, y0]\)
apply(simp)
using assms(1) kas f lqs by(auto simp add: BIT_{\text{inv2}} other_def)

then show \(\text{thesis using BIT s by simp}\)
qed

lemma \(\text{BIT}_c2\); assumes A: \(x \neq y\)
init \(\in \{[x,y],[y,x]\}\)
v \(\in\) lang \([\text{seq \{\text{Atom} x, \text{Times} (\text{Atom} y) (\text{Atom} x), \text{Star} (\text{Times} (\text{Atom} y) (\text{Atom} x)), \text{Atom} x]\}]\)
shows \(T_{p \text{-on-rand}}' \text{ BIT} (\text{type0 init} x y) v = 0.75 * (\text{length} v - 1) - 0.5\) (is \(\text{?T}\))
and config\(\text{'}_{\text{rand}} \text{ BIT} (\text{type0 init} x y) v = (\text{type0 init} x y)\) (is \(\text{?C}\))
proof –
from assms(3) obtain w where vw: \(v = [x}@w\) and
\(w \in\) lang \([\text{seq \{\text{Times} (\text{Atom} y) (\text{Atom} x), \text{Star} (\text{Times} (\text{Atom} y) (\text{Atom} x)), \text{Atom} x]\}]\)
by (auto)
have c1: config\(\text{'}_{\text{rand}} \text{ BIT} (\text{type0 init} x y) [x] = \text{type0 init} x y\)
using assms by(auto simp add: oneBIT_step)
have t1: \(T_{p \text{-on-rand}}' \text{ BIT} (\text{type0 init} x y) [x] = 0\)
using assms by(auto simp add: costBIT)
show \(T_{p \text{-on-rand}}' \text{ BIT} (\text{type0 init} x y) v\)
\(= 0.75 * (\text{length} v - 1) - 0.5\)
unfolding vw apply(simp only: T_{on-rand}' append c1 BIT_c[OF assms(1,2) w] t1)
by (simp)
show config\(\text{'}_{\text{rand}} \text{ BIT} (\text{type0 init} x y) v = (\text{type0 init} x y)\)
unfolding vw by(simp only: config\(\text{'}_{\text{rand}}\) append c1 BIT_c[OF assms(1,2) w])
qed

lemma \(\text{bit}_c''2\); assumes
\(x \neq y \{x, y\} = \{x0, y0\}\)
set qs \(\subseteq \{x, y\}\)
qs \(\in\) lang \([\text{seq}\{\text{Atom} x, \text{Atom} y, \text{Atom} x, \text{Star} (\text{Times} (\text{Atom} y) (\text{Atom} x)), \text{Atom} x\}]\)
shows \(\text{BIT}_{\text{inv}}(\text{config}_\text{rand} \text{ BIT} s \text{ qs}) (\text{last qs}) [x0, y0] \land\)
\(T_{p \text{-on-rand}}' \text{ BIT} s \text{ qs} = 0.75 * (\text{length} \text{ qs} - 1) - 0.5\)
proof

- from assms have \( f : x \neq y \) by auto

- from assms(1,3) assms(2)[symmetric] have \( s : s = \text{type0} \ [x,y] x y \)
  apply(simp add: BIT_inv2[OF \( f \)] other_def) by fast

- from assms(1,2) have \( \text{kas} : [x,y] = [x0,y0] \lor [x,y] = [y0,x0] \) by auto

- from assms(5) have \( \text{lqs} : \text{last qs} = x \) by fastforce

- from assms(1,2) kas have \( \text{BIT} : T_{p\text{-on_rand'}} \text{BIT s qs} = 0.75 \ast (\text{length qs}-1) - 0.5 \)
  unfolding \( s \)
  apply(safe)
  apply(rule BIT_c2)
  apply(simp) apply(simp) using assms(5) apply(simp add: conc_assoc)
  apply(rule BIT_c2)
  apply(simp) apply(simp) using assms(5) apply(simp add: conc_assoc)
  done

- from assms(1,2) kas have \( \text{config'}\_\text{rand BIT s qs} = \text{type0} \ [x0, y0] x y \)
  unfolding \( s \)
  apply(safe)
  apply(rule BIT_c2)
  apply(simp) apply(simp) using assms(5) apply(simp add: conc_assoc)
  apply(rule BIT_c2)
  apply(simp) apply(simp) using assms(5) apply(simp add: conc_assoc)
  done

- then have \( \text{config} : \text{BIT}\_\text{inv} (\text{config'}\_\text{rand BIT s qs}) (\text{last qs}) \ [x0, y0] \)
  apply(simp)
  using assms(1) kas lqs by(auto simp add: BIT_inv2 other_def)

- show \( ?\text{thesis using BIT config by simp} \)

qed

17.2.7 Phase Form D

lemma bit.d; assumes \( x \neq y \ \{x, y\} = \{x0, y0\} \ \text{BIT}\_\text{inv} s x \ [x0, y0] \)
set \( qs \subseteq \{x, y\} \ \text{qs} \in \text{lang seq} [\text{Atom } x, \text{Atom } x] \)
shows \( T_{p\text{-on_rand'}} \text{BIT s qs} \leq 175 / 10^2 \ast \text{real} (T_p [x, y] \ \text{qs} (\text{OPT2 qs} [x, y])) \land \)
\( \text{BIT}\_\text{inv} (\text{config'}\_\text{rand BIT s qs}) (\text{last qs}) \ [x0, y0] \land \)
\( T_{p\text{-on_rand'}} \text{BIT s qs} = 0 \)
proof


from assms have qs: qs = [x,x] by auto
then have OPT: T_p [x, y] qs (OPT2 qs [x, y]) = 0 by (simp add: t_p_def step_def)

from assms have f: x0≠y0 by auto
from assms(1,3) assms(2)[symmetric] have s: s = type0 [x0,y0] x y
apply(simp add: BIT_inv2[OF f] other_def) by fast

from assms(1,2) have kas: [x,y] = [x0,y0] ∨ [x,y] = [y0,x0] by auto
have BIT: T_p_on_rand' BIT (type0 [x0,y0] x y) qs = 0
using kas assms(1,2) by (auto simp add: qs oneBIT_step costBIT)

have lqs: last qs = x last qs ∈ {x0,y0} using assms(2,4) qs by auton

have inv: config_rand BIT s qs = type0 [x0, y0] x y
using kas assms(1,2) by (auto simp add: qs s oneBIT_step0x)

then have BIT_inv (config_rand BIT s qs) (last qs) [x0, y0]
apply(simp)
using assms(1) kas f lqs by(auto simp add: BIT_inv2 other_def)

then show ?thesis using BIT s by(auto)
qed

lemma bit_d': assumes
  x ≠ y {x, y} = {x0, y0} BIT_inv s x [x0, y0]
set qs ⊆ {x, y} qs ∈ lang (seq [Atom x, Atom x])
shows BIT_inv (config_rand BIT s qs) (last qs) [x0, y0] ∧
T_p_on_rand' BIT s qs = 0
using bit_d[OF assms] by blast

17.3 Phase Partitioning

lemma BIT_inv_initial: assumes (x::nat) ≠ y
shows BIT_inv (map_pmf (Pair [x, y]) (fst BIT [x, y])) x [x, y]
using assms(1) apply(simp add: BIT_inv2 BIT_init_def type0_def)
apply(simp add: map_pmf_def other_def bind_return_pmf bind_assoc_pmf)
using bind_commute_pmf by fast

lemma D'': assumes qs ∈ Lxx a b
  a ≠ b {a, b} = {x, y} BIT_inv s a [x, y]
set qs ⊆ {a, b}
shows T_p_on_rand' BIT s qs ≤ 175 / 10^2 * real (T_p [a, b] qs (OPT2 qs

361
\[ [a, b]) \land 
\] 
\[ \text{BIT_inv} \ (\text{Partial\_Cost\_Model.config'} \text{\_rand} \text{ BIT s qs} \ (\text{last qs}) \ [x, y] 
\] 
apply(rule \text{LxxE[OF assms(1)]}) 
using \text{bit_d[OF assms(2–5)] apply(simp)} 
apply(rule \text{bit_b''[OF assms(2–5)] apply(simp)} 
apply(rule \text{bit_c''[OF assms(2–5)] apply(simp)} 
using \text{bit_a[OF assms(2–5)] apply(simp)} 
done

theorem \text{BIT.175comp_on_2:} 
assumes \( (x::\text{nat}) \neq y \ \text{set } \sigma \subseteq \{x,y\} \) 
shows \( T_{p\_on\_rand} \text{ BIT} \ [x,y] \ \sigma \leq 1.75 \ast \text{real} \ (T_{p\_opt} \ [x,y] \ \sigma) + 1.75 \)

proof (rule \text{Phase\_partitioning\_general[where P=BIT_inv], goal_cases}) 
case 4 
show \text{BIT_inv} \ (\text{map\_pmf} \ (\text{Pair} \ [x, y]) \ (\text{fst BIT} \ [x, y]))) \ x \ [x, y] 
by (rule \text{BIT_inv\_initial[OF assms(1)]}) 
next 
case (5 a b qs s) 
then show \( ?case \) by(rule \text{D''}) 
qed (simp\_all add: assms)

end

18 COMB

theory \text{Comb} 
imports \text{TS BIT.2comp_on2 BIT\_pairwise} 
begin

18.1 Definition of COMB

type\_synonym \text{CombState} = (\text{bool list} * \text{nat list}) + (\text{nat list})

definition \text{COMB\_init :: nat list => (nat state, CombState) alg\_on\_init} 
where 
\text{COMB\_init} \ h \ \text{init} = 
\text{Sum\_pmf} \ 0.8 \ (\text{fst BIT} \ \text{init}) \ (\text{fst (embed (rTS h)) init})

lemma \text{COMB\_init[simp]: COMB\_init} \ h \ \text{init} = 
do \{ 
(b::\text{bool}) \leftarrow \ (\text{bernoulli\_pmf} \ 0.8); 
(xs::\text{bool list}) \leftarrow \ \text{Prob\_Theory.bv (length init)}; 
\text{return\_pmf} \ (\text{if b then Inl \ (xs, init) else Inr \ h})
\begin{verbatim}
apply(simp add: bind_return_pmf COMB_init_def BIT_init_def rTS_def
  bind_assoc_pmf)
undoing map_pmf_def Sum_pmf_def
apply(simp add: if_distrib bind_return_pmf bind_assoc_pmf)
  apply(rule bind_pmf_cong)
    by(auto simp add: bind_return_pmf bind_assoc_pmf)

definition COMB_step :: (nat state, CombState, nat, answer) alg_on_step
where
  COMB_step s q = (case snd s of Inl b \Rightarrow
    map_pmf (\((a,b),c\) \Rightarrow ((a,b),Inl c)) (BIT_step (fst s, b) q)
    | Inr b \Rightarrow map_pmf (\((a,b),c\) \Rightarrow ((a,b),Inr c))
    (return_pmf (TS_step_d (fst s, b) q)))

definition COMB h = (COMB_init h, COMB_step)

18.2  Comb 1.6-competitive on 2 elements

abbreviation noc == (%x. case x of Inl (s,is) \Rightarrow (s,Inl is) | Inr (s,is) \Rightarrow
  (s,Inr is) )
abbreviation con == (%(s,is). case is of Inl is \Rightarrow Inl (s,is) | Inr is \Rightarrow
  Inr (s,is) )

definition inv_COMB s x i == (\exists Da Db. finite (set_pmf Da) \& finite
  (set_pmf Db) \&
    (map_pmf con s) = Sum_pmf 0.8 Da Db \& BIT_inv Da x i \& TS_inv
  Db x i)

lemma noccon: noc o con = id
  apply(rule ext)
  apply(case_tac x) by(auto simp add: sum_case_eq_if)

lemma connoc: con o noc = id
  apply(rule ext)
  apply(case_tac x) by(auto simp add: sum_case_eq_if)

lemma obligation1': assumes map_pmf con s = Sum_pmf (8 / 10) Da Db
  shows config'_rand (COMB h) s qs = map_pmf noc (Sum_pmf (8 / 10) (config'_rand BITS Da qs)
    (config'_rand (embed (rTS h)) Db qs))
  proof (induct qs rule: rev_induct)
    case Nil
    have s = map_pmf noc (map_pmf con s)
  
\end{verbatim}
by (simp add: pmf.map_comp noccon)
also
from assms have \ldots = map\_pmf noc (Sum\_pmf (8 / 10) Da Db)
  by simp
finally
show \case by simp
next
  case (snoc q qs)
  show \case apply (simp)
    apply (subst config', rand_append)
    apply (subst snoc)
    apply (simp)
    unfolding Sum\_pmf_def
    apply (simp add:
      bind_assoc\_pmf bind_return\_pmf COMB\_def COMB\_step\_def)
    apply (subst config', rand_append)
    apply (subst config', rand_append)
    apply (simp only: map\_pmf_def[where f=noc])
    apply (simp add: bind_return\_pmf bind_assoc\_pmf)
    apply (rule bind\_pmf\_cong)
    apply (simp)
    apply (simp only: set\_pmf\_bernoulli UNIV\_bool)
    apply (auto)
    apply (simp only: map\_pmf\_def[where f=Inl])
    apply (simp add: bind_return\_pmf bind_assoc\_pmf)
    apply (rule bind\_pmf\_cong)
    apply (simp add: bind_return\_pmf bind_assoc\_pmf)
    apply (simp add: split\_def)
    apply (simp add: bind_return\_pmf bind_assoc\_pmf map\_pmf\_def)
    apply (simp only: map\_pmf\_def[where f=Inr])
    apply (simp add: bind_return\_pmf bind_assoc\_pmf)
    apply (rule bind\_pmf\_cong)
    apply (simp add: bind_return\_pmf bind_assoc\_pmf)
    apply (simp add: split\_def)
    apply (simp add: bind_return\_pmf bind_assoc\_pmf map\_pmf\_def)
  rTS\_def)
  done
qed

lemma obligation1":
  shows config\_rand (COMB h) init qs =
  map\_pmf noc (Sum\_pmf (8 / 10) (config\_rand BIT init qs)
  (config\_rand (embed (rTS h)) init qs))
apply (rule obligation1')
apply(simp add: Sum_pmf_def COMB_def map_pmf_def bind_assoc_pmf bind_return_pmf split_def COMB_init_def del: COMB_init)
apply(rule bind_pmf_cong)
by(auto simp add: split_def map_pmf_def bind_return_pmf bind_assoc_pmf)

lemma obligation1: assumes map_pmf con s = Sum_pmf (8 / 10) Da Db
shows \( \text{map}_\text{pmf} \text{(config}'\text{rand} (\text{COMB} [])) s qs = \sum p_mf (8 / 10) (\text{config}'\text{rand} \text{BIT} Da qs)
\text{(config}'\text{rand} (\text{embed} (rTS [])) Db qs) \)

proof –
from obligation1[OF assms] have \( \text{map}_\text{pmf} \text{(config}'\text{rand} (\text{COMB} [])) s qs = \sum p_mf \text{(map}_\text{pmf} \text{nuc} (\sum p_mf (8 / 10) (\text{config}'\text{rand} \text{BIT} Da qs)
\text{(config}'\text{rand} (\text{embed} (rTS [])) Db qs))) \)
by simp
also have \( \ldots = \sum p_mf (8 / 10) (\text{config}'\text{rand} \text{BIT} Da qs)
\text{(config}'\text{rand} (\text{embed} (rTS [])) Db qs) \)
apply(simp only: pmf_map comp connoc) by simp
finally show \( \text{thesis} \).
qed

lemma BIT_config’_fin: finite (set_pmf s) \rightarrow finite (set_pmf (config’_rand BIT s qs))
apply(induct qs rule: rev_induct)
apply(simp)
by(simp add: config’_rand_append BIT_step_def)

lemma TS_config’_fin: finite (set_pmf s) \rightarrow finite (set_pmf (config’_rand (embed (rTS h)) s qs))
apply(induct qs rule: rev_induct)
apply(simp)
by(simp add: config’_rand_append rTS_def TS_step_d_def)

lemma obligation2: assumes map_pmf con s = Sum_pmf (8 / 10) Da Db
and finite (set_pmf Da)
and finite (set_pmf Db)
shows \( T_{p\_on\_rand}' (\text{COMB} []) s q s = \frac{2}{10} * T_{p\_on\_rand}' (\text{embed} (rTS [])) Da q s + \frac{8}{10} * T_{p\_on\_rand}' \text{BIT} Da q s \)
proof (induct qs rule: rev_induct)
case (snoc q qs)

365
have \( P: T_{p\cdot on\cdot rand'} (COMB []) (config'\cdot rand (COMB []) s qs) [q] \)
\[ = \frac{2}{10} \cdot T_{p\cdot on\cdot rand'} (embed (rTS [])) (config'\cdot rand (embed (rTS [])) Db qs) [q] \]
\[ + \frac{8}{10} \cdot T_{p\cdot on\cdot rand'} BIT (config'\cdot rand BIT Da qs) [q] \]
apply (subst obligation1 \((OF assms(1))\))
unfolding Sum_pmf_def
apply (simp)
apply (simp only: map_pmf_def [where \( f = noc\)])
apply (simp add: bind_assoc_pmf)
apply (subst E_bernoulli3)
apply (simp)
apply (simp add: set_pmf_bernoulli)
apply (simp add: BIT_step_def COMB_def COMB_step_def split_def)
apply (safe)
using BIT_config'_fin [OF assms(2)] apply (simp)
using TS_config'_fin [OF assms(3)] apply (simp)
apply (simp)
apply (simp only: map_pmf_def [where \( f = Inl\)])
apply (simp only: map_pmf_def [where \( f = Inr\)])
apply (simp add: bind_return_pmf bind_assoc_pmf COMB_def COMB_step_def)
apply (simp add: split_def)
apply (simp add: rTS_def map_pmf_def bind_return_pmf bind_assoc_pmf COMB_def COMB_step_def)
done

show \(?case\)
apply (simp only: T_on_rand' append)
apply (subst snoc)
apply (subst P) by algebra

qed simp

lemma Combination:
assumes \( qs \in \text{pattern} \) a \( \neq \) b \( \{ a, b \} = \{ x, y \} \) set qs \( \subseteq \) \( \{ a, b \} \)
and \( \text{inv\cdot COMB} s a [x,y] \)
and \( \text{TS}: \forall s h. a \neq b \implies \{ a, b \} = \{ x, y \} \implies TS\_inv s a [x, y] \implies \)
set qs \( \subseteq \) \( \{ a, b \} \)
\( \implies \) qs \( \in \) \text{pattern} \implies \)
\( TS\_inv (config'\cdot rand (embed (rTS h)) s qs) (\text{last} qs) [x, y] \)
\( \land T_{p\cdot on\cdot rand'} (embed (rTS h)) s qs = ts \)
and \( \text{BIT}: \forall s. a \neq b \implies \{ a, b \} = \{ x, y \} \implies BIT\_inv s a [x, y] \implies \)
set qs \( \subseteq \) \( \{ a, b \} \)
\( \implies \) qs \( \in \) \text{pattern} \implies \)
\begin{verbatim}

BIT_inv (config'_rand BIT s qs) (last qs) [x, y]
∧ T_p_on_rand' BIT s qs = bit
and OPT_cost: a ≠ b ⇒ qs ∈ pattern ⇒ real (T_p [a, b] qs (OPT2 qs [a, b])) = opt
and absch: qs ∈ pattern ⇒ 0.2 * ts + 0.8 * bit ≤ 1.6 * opt
shows T_p_on_rand' (COMB []) s qs ≤ 16 / 10 * real (T_p [a, b] qs (OPT2 qs [a, b])) ∧
inv.COMB (Partial_Cost_Model.config'_rand (COMB []) s qs) (last qs) [x, y]

proof –
let ?D = map_pmf con s
from assms(5) obtain Da Db where Daf: finite (set_pmf Da)
  and Dbf: finite (set_pmf Db)
  and D: ?D = Sum_pmf 0.8 Da Db
  and B: BIT_inv Da a [x, y] and T: TS_inv Db a [x, y]
unfolding inv.COMB_def by auto

let ?Da' = config'_rand BIT Da qs
from BIT[OF assms(2,3) B assms(4,1)]
  have B': BIT_inv ?Da' (last qs) [x, y]
  and B_cost: T_p_on_rand' BIT Da qs = bit by auto

let ?Db' = config'_rand (embed (rTS [])) Db qs
from TS[OF assms(2,3) T assms(4,1)]
  have T': TS_inv ?Db' (last qs) [x, y]
  and T_cost: T_p_on_rand' (embed (rTS [])) Db qs = ts by auto

have T_p_on_rand' (COMB []) s qs
  = 0.2 * T_p_on_rand' (embed (rTS [])) Db qs
  + 0.8 * T_p_on_rand' BIT Da qs
  using D apply(rule obligation2) apply(fact Daf) apply(fact Dbf)
done
also
  have ... ≤ 1.6 * opt
  by (simp only: B_cost T_cost absch[OF assms(1)])
also
  have ... = 1.6 * T_p [a, b] qs (OPT2 qs [a, b]) by (simp add: OPT_cost[OF assms(2,1)])
finally
  have Comb_cost: T_p_on_rand' (COMB []) s qs ≤ 1.6 * T_p [a, b] qs (OPT2 qs [a, b]) .

\end{verbatim}
have Comb_inv: inv_COMB (config’_rand (COMB [])) s qs (last qs) [x, y]
unfolding inv_COMB_def
apply(rule exI[where x=?Da’])
apply(rule exI[where x=?Db’])
apply(safe)
apply(rule BIT_config’_fin[OF Daf])
apply(rule TS_config’_fin[OF Dbf])
apply(rule obligation1)
apply(fact D)
apply(fact B’)
apply(fact T’) done

from Comb_cost Comb_inv show ?thesis by simp
qed

theorem COMB_OPT2’: (x::nat) ≠ y ⇒ set σ ⊆ {x, y}
⇒ T_p-on_rand (COMB []) [x,y] σ ≤ 1.6 * real (T_p-opt [x,y] σ) + 1.6

proof (rule Phase_partitioning_general[where P=inv_COMB], goal_cases)
case 4
let ?initBIT = (map_pmf (Pair [x, y]) (fst BIT [x, y]))
let ?initTS = (map_pmf (Pair [x, y]) (fst (embed (rTS [])) [x, y]))
show inv_COMB (map_pmf (Pair [x, y]) (fst (COMB [])) [x, y]) x [x, y]
unfolding inv_COMB_def
apply(rule exI[where x=?initBIT])
apply(rule exI[where x=?initTS])
apply(simp only: BIT_inv_initial[OF 4(1)])
apply(simp add: map_pmf_def bind_return_pmf bind_assoc_pmf COMB_def)
apply(simp add: Sum_pmf_def)
apply(safe)
apply(simp add: BIT_init_def)
apply(rule bind_pmf_cong)
apply(simp)
apply(simp add: bind_return_pmf bind_assoc_pmf rTS_def map_pmf_def)
BIT_init_def)
apply(simp add: TS_inv_def rTS_def)
done
next
case (5 a b qs s)
from 5(9)
show ?case
proof (rule LxxE, goal_cases)
case 4
then show \(?thesis\) apply (rule Combination)
apply (fact)+
using $TS_a''$ apply (simp)
apply (fact $bit_a''$)
apply (fact $OPT2_A'$)
apply (simp)
done
next
case 1
then show \(?case\)
apply (rule Combination)
apply (fact)+
apply (fact $TS_d''$)
apply (fact $bit_d'$)
by auto
next
case 2
then have $qs \in \text{lang} \{ (\text{seq} [\text{Atom} b, \text{Atom} a, \text{Star} (\text{Times} (\text{Atom} b)), \text{Atom} b, \text{Atom} b])$
\lor $qs \in \text{lang} \{ \text{seq} [\text{Atom} a, \text{Atom} b, \text{Atom} a, \text{Star} (\text{Times} (\text{Atom} b)), \text{Atom} b, \text{Atom} b]) \}$ by auto
then show \(?case\)
apply (rule disjE)
apply (erule Combination)
apply (fact)+
apply (fact $TS_b1''$)
apply (fact $bit_b''1$)
apply (fact $OPT2_B1$)
apply (simp add: ring_distrib)
apply (erule Combination)
apply (fact)+
apply (fact $TS_b2''$)
apply (fact $bit_b''2$)
apply (fact $OPT2_B2$)
apply (simp add: ring_distrib)
done
next
case 3
then have \(\text{len} : \text{length} \ qs \geq 2 \ \text{by} (\text{auto simp add: conc_def})\)
have \(\text{len2} : \ qs \in \text{lang} \ (\text{seq} [\text{Atom} a, \text{Atom} b, \text{Atom} a, \text{Star} (\text{Times} (\text{Atom} b)), \text{Atom} a]) \Rightarrow \text{length} \ qs \geq 3 \ \text{by} (\text{auto simp add: conc_def})\)
from 3 have \(qs \in \text{lang} \ (\text{seq} [\text{Atom} b, \text{Atom} a, \text{Star} (\text{Times} (\text{Atom} b))\])\)
\[(\text{Atom } a), (\text{Atom } a)]\)

\[\forall \text{qs} \in \text{lang} [\text{seq } [\text{Atom } a, \text{Atom } b, \text{Atom } a, \text{Star } (\text{Times } (\text{Atom } b)) (\text{Atom } a)], (\text{Atom } a)] \text{ by auto}

\text{then show} \ ?\text{case}

\text{apply} (\text{rule disjE})

\text{apply} (\text{erule Combination})

\text{apply} (\text{fact} +)

\text{apply} (\text{fact TS}_{c1}'')

\text{apply} (\text{fact} \text{ bit}_{c''1})

\text{apply} (\text{fact} \text{ OPT2}_{C1})

\text{using} \text{len} \text{apply} (\text{simp add: ring_distribs})

\text{apply} (\text{erule Combination})

\text{apply} (\text{fact} +)

\text{apply} (\text{fact TS}_{c2}'')

\text{apply} (\text{fact} \text{ bit}_{c''2})

\text{apply} (\text{fact} \text{ OPT2}_{C2})

\text{using} \text{len2 apply} (\text{simp add: ring_distribs conc_def})

\text{done}

\text{qed}

\text{qed} (\text{simp_all})

\textbf{18.3 \ COMB \ pairwise}\n
\textbf{lemma} \text{config\_rand\_COMB}: \text{config\_rand} (\text{COMB } h) \text{ init } \text{qs} = \text{do} \{\,
(b :: \text{bool}) \leftarrow (\text{bernoulli\_pmf} \ 0.8);\n(b1, b2) \leftarrow (\text{config\_rand} \ \text{BIT} \text{ init } \text{qs});\n(t1, t2) \leftarrow (\text{config\_rand} (\text{embed} (rTS \ h)) \text{ init } \text{qs});\n\text{return\_pmf} (\text{if } b \text{ then } (b1, \text{Inl } b2) \text{ else } (t1, \text{Inr } t2))\,
\} \text{ (is } ?\text{LHS} = ?\text{RHS})

\textbf{proof} \text{ (induct } \text{qs rule: } \text{rev\_induct})

\text{case} \ \text{Nil} \n
\text{show} \ ?\text{case}

\text{apply} (\text{simp add: BIT\_init\_def COMB\_def rTS\_def map\_pmf\_def bind\_return\_pmf bind\_assoc\_pmf})

\text{apply} (\text{rule bind\_pmf\_cong})

\text{apply} (\text{simp})

\text{apply} (\text{simp only: set\_pmf\_bernoulli})

\text{apply} (\text{case\_tac } x)\n
\text{by} (\text{simp\_all})

\text{next}

\text{case} (\text{snoc } q \text{ qs}) \n
\text{show} \ ?\text{case} \text{apply} (\text{simp add: suc\_conv\_app\_nth})

\text{apply} (\text{subst config\_rand\_append})

\text{apply} (\text{subst snoc})

370
apply(simp)
apply(simp add: bind_return_pmf bind_assoc_pmf split_def config'_rand_append)
apply(rule bind_pmf_cong)
apply(simp)
apply(simp only: set_pmf_beroulli)
apply(case_tac x)
by(simp_all add: COMB_def COMB_step_def rTS_def map_pmf_def
split_def bind_return_pmf bind_assoc_pmf)

qed

lemma COMB_no_paid: \( \forall ((\text{free}, \text{paid}), t) \in \text{set} \text{ pmf} (\text{snd} (\text{COMB []}) (s,\text{ is}) q). \text{paid} = [] \)
apply(simp add: COMB_def COMB_step_def split_def BIT_step_def TS_step_d_def)
apply(case_tac is)
by(simp_all add: BIT_step_def TS_step_d_def)

lemma COMB_pairwise: pairwise (COMB [])
proof(rule pairwise_property_lemma, goal_cases)
case (1 init qs x y)
then have qsininit: set qs \subseteq set init by simp

show Pbefore_in x y (COMB []) qs init
  = Pbefore_in x y (COMB []) (Lxy qs \{x, y\}) (Lxy init \{x, y\})
unfolding Pbefore_in_def
apply(subst config_rand_COMB)
apply(subst config_rand_COMB)
apply(simp only: map_pmf_def bind_assoc_pmf)
apply(rule bind_pmf_cong)
apply(simp)
apply(simp only: set_pmf_beroulli)
apply(case_tac xa)
apply(simp add: split_def)
using BIT_pairwise[OF qsininit 1(3,4,1), unfolded Pbefore_in_def map_pmf_def]
apply(simp add: bind_return_pmf bind_assoc_pmf)
apply(simp add: split_def)
using TS_pairwise[OF 1(2,3,4,1), unfolded Pbefore_in_def map_pmf_def]
by(simp add: bind_return_pmf bind_assoc_pmf)

next
case (2 xa r)
show ?case
  using COMB_no_paid
by (metis (mono_tags) case_prod_unfold surj_pair)

qed

18.4 COMB 1.6-competitive

lemma finite_config_TS: finite (set_pmf (config" (embed (rTS h)) qs init
and is finite ?D)
apply (subst config_embed)
by (simp)

lemma COMB_has_finite_config_set: assumes [simp]: distinct init
and set qs ⊆ set init
shows finite (set_pmf (config_rand (COMB h) init qs))
proof
from finite_config_TS [where n=length qs and qs=qs]
finit_config_BIT[OF assms (1)]
show ?thesis
apply (subst obligation1")
by (simp add: Sum_pmf_def)

qed

theorem COMB_competitive: ∀ s0∈{x::nat list. distinct x ∧ x≠[]}. 
∃ b≥0. ∀ qs∈{x. set x ⊆ set s0}.
    T_p_on_rand (COMB []) s0 qs ≤ ((8::nat)/(5::nat)) * T_p_opt s0 qs + b
proof (rule factoring_lemma_withconstant, goal_cases)
case 5
show ?case
proof (safe, goal_cases)
  case (1 init)
  note out=this
  show ?case
    apply (rule exI[where x=2])
    apply (simp)
    proof (safe, goal_cases)
      case (1 qs a b)
      then have a: a≠b by simp
      have twist: {a,b}={b,a} by auto
      have b1: set (Lxy qs {a,b}) ⊆ {a,b} unfolding Lxy_def by auto
      with this [unfolded twist] have b2: set (Lxy qs {b,a}) ⊆ {b,a}
      by (auto)
      have set (Lxy init {a,b}) = {a,b} ∩ (set init) apply (induct init)
unfolding \[Lxy\] def by (auto) with 1 have \(A: \text{set (Lxy init \(\{a, b\}\)) = \{a, b\}}\) by auto have finite \(\{a, b\}\) by auto from out have \(B: \text{distinct (Lxy init \(\{a, b\}\))}\) unfolding \(Lxy\) def by auto have \(C: \text{length (Lxy init \(\{a, b\}\)) = 2}\) using distinct_card[OF B, unfolded \(A\)] using \(a\) by auto

have \(\{xs. \text{set xs} = \{a, b\} \land \text{distinct xs } \land \text{length xs} = (2::nat)\}\) = \([a, b], [b, a]\) apply (auto simp: \(a\) \(a\) [symmetric]) proof (goal_cases)
  case \((1 \hspace{1cm} xs)\)
  from \(1\) \((4)\) obtain \(x \hspace{1cm} xs'\) where \(r:xs = x \# xs'\) by (metis Suc_length_conv add_2_eq_Suc' append_Nil length_append)
  with \(1\) \((4)\) have \(\text{length } xs' = 1\) by auto
  then obtain \(y\) where \(s: [y] = xs'\) by (metis One_nat_def length_0_conv length_Suc_conv)
  from \(r\) \(s\) have \(t: [x, y] = xs\) by auto
  moreover from \(t\) \(1\) \((1)\) have \(x = b\) using doubleton_eq_iff
  \(1\) \((2)\) by fastforce
  moreover from \(t\) \(1\) \((1)\) have \(y = a\) using doubleton_eq_iff
  \(1\) \((2)\) by fastforce
  ultimately show \(?case by auto\)
qed

with \(A B C\) have \(\text{pos: (Lxy init \(\{a, b\}\)) } = [a, b]\)
\(\lor (Lxy init \(\{a, b\}\)) = [b, a]\) by auto

show \(?case by auto\)
apply cases 
\((Lxy init \(\{a, b\}\)) = [a, b]\)
apply (simp) using COMB_OPT2[\(OF\ a\ b1\)] a apply (simp)
using pos apply (simp) unfolding twist
using COMB_OPT2[\(OF\ a[\text{symmetric}]\ b2\)] by simp
qed

next
case 4 then show \(?case by simp\)
next
case 7 then show \(?case apply\(\text{subst COMB_has_finite_config_set[}OF\ 7(1)]\)\)
  using set_take_subset apply fast by simp
qed (simp_all add: COMB_no_paid)
theorem COMB_competitive_nice: \text{compet_rand} (\text{COMB} []) ((8::nat)/(5::nat))
\{x::nat list. distinct x \land x\neq[]\}
unfolding \text{compet_rand_def} \text{static_def} using COMB_competitive by simp

end

References
