Analysis of List Update Algorithms

Maximilian P.L. Haslbeck and Tobias Nipkow

April 20, 2020

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 competitive- ness 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

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*
⟨*proof*⟩

lemma *before_in_setD2*: *x* < *y in xs* ⟹ *y* : *set xs*
⟨*proof*⟩

lemma *not_before_in*:
x : *set xs* ⟹ *y* : *set xs* ⟹ ¬ *x* < *y in xs* ⟷ *y* < *x in xs* ∨ *x*=*y*
⟨*proof*⟩

lemma *before_in_irefl*: *x* < *x in xs* = *False*
⟨*proof*⟩

lemma *no_before_inI*[*simp*]: *x* < *y in xs* ⟹ (¬ *y* < *x in xs*) = *True*
⟨*proof*⟩

lemma *finite_Invs*[*simp*]: *finite*(*Inv xs ys*)
⟨*proof*⟩

lemma *Inv_id*[*simp*]: *Inv xs xs* = {}
⟨*proof*⟩

lemma *card_Inv_sym*: *card*(*Inv xs ys*) = *card*(*Inv ys xs*)
⟨*proof*⟩

lemma *Inv_tri_ineq*:
dist_perm xs ys ⟹ *dist_perm ys zs* ⟹
Inv xs zs ⊆ *Inv xs ys Un Inv ys zs*

⟨*proof*⟩

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*)
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *swap_id*[*simp*]: *Suc n* ≥ *size xs* ⟹ *swap n xs* = *xs*
⟨*proof*⟩

lemma *distinct_swap*[*simp*]:
distinct(*swap i xs*) = *distinct xs*
⟨*proof*⟩

lemma *swap_Suc*[*simp*]: *swap* (*Suc n*) (*a* # *xs*) = *a* # *swap n xs*
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *set_swap*[*simp*]: *set*(*swap n xs*) = *set xs*
⟨*proof*⟩

lemma *nth_swap_id*[*simp*]: *Suc i* < *length xs* ⟹ *swap i xs* ! *i* = *xs*!(*i*+*1*)
⟨*proof*⟩

lemma *before_in_swap*:
dist_perm xs ys $\Longrightarrow$ *Suc n* $<$ *size xs* $\Longrightarrow$
x $<$ *y in* (*swap n xs*) $\longleftrightarrow$
x $<$ *y in xs* $\wedge$ $\neg$ (*x* $=$ *xs*!*n* $\wedge$ *y* $=$ *xs*!*Suc n*) $\vee$ *x* $=$ *xs*!*Suc n* $\wedge$ *y* $=$ *xs*!*n*
⟨*proof*⟩

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 $\cup$ {(*ys*!*n*, *ys*!*Suc n*)}
else Inv xs ys $-$ {(*ys*!*Suc n*, *ys*!*n*)}
else Inv xs ys)
⟨*proof*⟩

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* $\wedge$
size(*swaps sws xs*) $=$ *size xs* $\wedge$
distinct(*swaps sws xs*) $=$ *distinct xs*
⟨*proof*⟩

lemma *swaps_eq_Nil_iff*[*simp*]: *swaps acts xs* $=$ [] $\longleftrightarrow$ *xs* $=$ []
⟨*proof*⟩

lemma *swaps_map_Suc*[*simp*]:
swaps (*map Suc sws*) (*a* # *xs*) $=$ *a* # *swaps sws xs*
⟨*proof*⟩

lemma *card_Inv_swaps_le*:
distinct xs $\Longrightarrow$ *card* (*Inv xs* (*swaps sws xs*)) $\leq$ *length sws*
⟨*proof*⟩

lemma *nth_swaps*: $\forall$ *i*$\in$*set is*. *j* $<$ *i* $\Longrightarrow$ *swaps is xs* ! *j* $=$ *xs* ! *j*
⟨*proof*⟩

lemma *not_before0*[*simp*]: $^{\sim}$ *x* $<$ *xs* ! *0 in xs*
⟨*proof*⟩

lemma *before_id*[*simp*]: ⟦ *distinct xs*; *i* $<$ *size xs*; *j* $<$ *size xs* ⟧ $\Longrightarrow$
xs ! *i* $<$ *xs* ! *j in xs* $\longleftrightarrow$ *i* $<$ *j*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *card_Inv_swaps*:
⟦ *distinct is*; ∀ *i*∈*set is*. *Suc i* < *size xs*; *distinct xs* ⟧ ⟹
card(*Inv xs* (*swaps is xs*)) = *length is*
⟨*proof*⟩

lemma *swaps_eq_nth_take_drop*: *i* < *length xs* ⟹
swaps [*0*..<*i*] *xs* = *xs*!*i* # *take i xs* @ *drop* (*Suc i*) *xs*
⟨*proof*⟩

lemma *index_swaps_size*: *distinct s* ⟹
index s q ≤ *index* (*swaps sws s*) *q* + *length sws*
⟨*proof*⟩

lemma *index_swaps_last_size*: *distinct s* ⟹
size s ≤ *index* (*swaps sws s*) (*last s*) + *length sws* + *1*
⟨*proof*⟩

end

3 Deterministic Online and Offline Algorithms

theory *On_Off*
imports *Complex_Main*
begin

type_synonym ('*s*,'*r*,'*a*) *alg_off* = '*s* ⇒ '*r list* ⇒ '*a list*
type_synonym ('*s*,'*is*,'*r*,'*a*) *alg_on* = ('*s* ⇒ '*is*) ∗ ('*s* ∗ '*is* ⇒ '*r* ⇒ '*a* ∗ '*is*)

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

fun *T* :: '*state* ⇒ '*request list* ⇒ '*answer list* ⇒ *nat* **where**

$T\ s\ []\ []\ =\ 0\ |$
$T\ s\ (r\#rs)\ (a\#as)\ =\ t\ s\ r\ a\ +\ T\ (step\ s\ r\ a)\ rs\ as$

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

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

lemma *config'_snoc*: $config'\ A\ s\ (rs@[r])\ =\ Step\ A\ (config'\ A\ s\ rs)\ r$
⟨*proof*⟩

lemma *config'_append2*: $config'\ A\ s\ (xs@ys)\ =\ config'\ A\ (config'\ A\ s\ xs)\ ys$
⟨*proof*⟩

lemma *config'_induct*: $P\ (fst\ init)\ \Longrightarrow\ (\bigwedge s\ q\ a.\ P\ s\ \Longrightarrow\ P\ (step\ s\ q\ a))$
$\Longrightarrow\ P\ (fst\ (config'\ A\ init\ rs))$
⟨*proof*⟩

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

lemma *config_snoc*: $config\ A\ s\ (rs@[r])\ =\ Step\ A\ (config\ A\ s\ rs)\ r$
⟨*proof*⟩

lemma *config_append*: $config\ A\ s\ (xs@ys)\ =\ config'\ A\ (config\ A\ s\ xs)\ ys$
⟨*proof*⟩

lemma *config_induct*: $P\ s0\ \Longrightarrow\ (\bigwedge s\ q\ a.\ P\ s\ \Longrightarrow\ P\ (step\ s\ q\ a))\ \Longrightarrow\ P\ (fst\ (config\ A\ s0\ qs))$
⟨*proof*⟩

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*
⟨*proof*⟩

abbreviation *T_on''* :: (*'state*,*'is*,*'request*,*'answer*) *alg_on* ⇒ *'state* ⇒ *'request list* ⇒ *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*
⟨*proof*⟩

abbreviation *T_on_n A s0 xs n == T_on' A (config A s0 (take n xs)) [xs!n]*

lemma *T_on__as_sum*: *T_on'' A s0 rs = sum (T_on_n A s0 rs) {..<length rs}*
⟨*proof*⟩

fun *off2* :: (*'state*,*'is*,*'request*,*'answer*) *alg_on* ⇒ (*'state* ∗ *'is*,*'request*,*'answer*) *alg_off* **where**
off2 A s [] = [] |
off2 A s (r#rs) = fst (snd A s r) # off2 A (Step A s r) rs

abbreviation *off* :: (*'state*,*'is*,*'request*,*'answer*) *alg_on* ⇒ (*'state*,*'request*,*'answer*) *alg_off* **where**
off A s0 ≡ off2 A (s0, fst A s0)

abbreviation *T_off* :: (*'state*,*'request*,*'answer*) *alg_off* ⇒ *'state* ⇒ *'request list* ⇒ *nat* **where**
T_off A s0 rs == T s0 rs (A s0 rs)

abbreviation *T_on* :: (*'state*,*'is*,*'request*,*'answer*) *alg_on* ⇒ *'state* ⇒ *'request list* ⇒ *nat* **where**
T_on A == T_off (off A)

lemma *T_on_on'*: *T_off* (λ*s0*. (*off2 A* (*s0*, *x*))) *s0 qs* = *T_on' A* (*s0*,*x*) *qs*
⟨*proof*⟩

lemma *T_on_on''*: *T_on A s0 qs* = *T_on'' A s0 qs*
⟨*proof*⟩

lemma *T_on_as_sum*: *T_on A s0 rs* = *sum* (*T_on_n A s0 rs*) {..<*length rs*}

⟨*proof*⟩

definition *T_opt* :: '*state* ⇒ '*request list* ⇒ *nat* **where**
T_opt s rs = *Inf* {*T s rs as* | *as*. *size as* = *size rs*}

definition *compet* :: ('*state*,'*is*,'*request*,'*answer*) *alg_on* ⇒ *real* ⇒ '*state set*
⇒ *bool* **where**
compet A c S = (∀ *s*∈*S*. ∃ *b* ≥ *0*. ∀ *rs*. *wf s rs* ⟶ *real*(*T_on A s rs*) ≤ *c* ∗
T_opt s rs + *b*)

lemma *length_off*[*simp*]: *length*(*off2 A s rs*) = *length rs*
⟨*proof*⟩

lemma *compet_mono*: **assumes** *compet A c S0* **and** *c* ≤ *c'*
shows *compet A c' S0*
⟨*proof*⟩

lemma *competE*: **fixes** *c* :: *real*
assumes *compet A c S0 c* ≥ *0* ∀ *s0 rs*. *size*(*aoff s0 rs*) = *length rs s0*∈*S0*
shows ∃ *b*≥*0*. ∀ *rs*. *wf s0 rs* ⟶ *T_on A s0 rs* ≤ *c* ∗ *T_off aoff s0 rs* + *b*
⟨*proof*⟩

end

end

4 Probability Theory

theory *Prob_Theory*
imports *HOL−Probability.Probability*
begin

lemma *integral_map_pmf*[*simp*]:
fixes f::*real* $\Rightarrow$ *real*
shows $(\int x.\ f\ x\ \partial(map\_pmf\ g\ M)) = (\int x.\ f\ (g\ x)\ \partial M)$
⟨*proof*⟩

4.1 function E

definition E :: *real pmf* $\Rightarrow$ *real* **where**
$E\ M = (\int x.\ x\ \partial\ measure\_pmf\ M)$

translations
$\int\ x.\ f\ \partial M <= CONST\ lebesgue\_integral\ M\ (\lambda x.\ f)$

notation (*latex* **output**) E $(E[\_]\ [1]\ 100)$

lemma *E_const*[*simp*]: $E\ (return\_pmf\ a) = a$
⟨*proof*⟩

lemma *E_null*[*simp*]: $E\ (return\_pmf\ 0) = 0$
⟨*proof*⟩

lemma *E_finite_sum*: *finite* $(set\_pmf\ X) \Longrightarrow E\ X = (\sum x{\in}(set\_pmf\ X).\ pmf\ X\ x * x)$
⟨*proof*⟩

lemma *E_of_const*: $E(map\_pmf\ (\lambda x.\ y)\ (X{::}real\ pmf)) = y$ ⟨*proof*⟩

lemma *E_nonneg*:
shows $(\forall\, x{\in}set\_pmf\ X.\ 0{\leq}\ x) \Longrightarrow 0 \leq E\ X$
⟨*proof*⟩

lemma *E_nonneg_fun*: **fixes** f::$'a{\Rightarrow}real$
shows $(\forall\, x{\in}set\_pmf\ X.\ 0{\leq}f\ x) \Longrightarrow 0 \leq E\ (map\_pmf\ f\ X)$
⟨*proof*⟩

lemma *E_cong*:
fixes f::$'a \Rightarrow real$
shows *finite* $(set\_pmf\ X) \Longrightarrow (\forall\, x{\in}\ set\_pmf\ X.\ (f\ x) = (u\ x)) \Longrightarrow E\ (map\_pmf\ f\ X) = E\ (map\_pmf\ u\ X)$
⟨*proof*⟩

lemma *E_mono3*:
fixes f::$'a \Rightarrow real$

shows *integrable* (*measure_pmf X*) *f* ⟹ *integrable* (*measure_pmf X*) *u* ⟹ (∀ *x*∈ *set_pmf X*. (*f x*) ≤ (*u x*)) ⟹ *E* (*map_pmf f X*) ≤ *E* (*map_pmf u X*)
⟨*proof*⟩

lemma *E_mono2*:
fixes *f*::'*a* ⇒ *real*
shows *finite* (*set_pmf X*) ⟹ (∀ *x*∈ *set_pmf X*. (*f x*) ≤ (*u x*)) ⟹ *E* (*map_pmf f X*) ≤ *E* (*map_pmf u X*)
⟨*proof*⟩

lemma *E_linear_diff2*: *finite* (*set_pmf A*) ⟹ *E* (*map_pmf f A*) − *E* (*map_pmf g A*) = *E* (*map_pmf* (λ*x*. (*f x*) − (*g x*)) *A*)
⟨*proof*⟩

lemma *E_linear_plus2*: *finite* (*set_pmf A*) ⟹ *E* (*map_pmf f A*) + *E* (*map_pmf g A*) = *E* (*map_pmf* (λ*x*. (*f x*) + (*g x*)) *A*)
⟨*proof*⟩

lemma *E_linear_sum2*: *finite* (*set_pmf D*) ⟹ *E*(*map_pmf* (λ*x*. (∑ *i*<*up*. *f i x*)) *D*)
= (∑ *i*<(*up*::*nat*). *E*(*map_pmf* (*f i*) *D*))
⟨*proof*⟩

lemma *E_linear_sum_allg*: *finite* (*set_pmf D*) ⟹ *E*(*map_pmf* (λ*x*. (∑ *i*∈ *A*. *f i x*)) *D*)
= (∑ *i*∈ (*A*::'*a set*). *E*(*map_pmf* (*f i*) *D*))
⟨*proof*⟩

lemma *E_finite_sum_fun*: *finite* (*set_pmf X*) ⟹
E (*map_pmf f X*) = (∑ *x*∈*set_pmf X*. *pmf X x* ∗ *f x*)
⟨*proof*⟩

lemma *E_bernoulli*: 0≤*p* ⟹ *p*≤1 ⟹
E (*map_pmf f* (*bernoulli_pmf p*)) = *p*∗(*f True*) + (1−*p*)∗(*f False*)
⟨*proof*⟩

4.2 function *bv*

fun *bv*:: *nat* ⇒ *bool list pmf* **where**
bv 0 = *return_pmf* []
| *bv* (*Suc n*) = *do* {
(*xs*::*bool list*) ← *bv n*;
(*x*::*bool*) ← (*bernoulli_pmf 0.5*);

return_pmf $(x\#xs)$
}

lemma *bv_finite*: *finite* $(bv\ n)$
⟨*proof*⟩

lemma *len_bv_n*: $\forall\, xs \in set\_pmf\ (bv\ n).\ length\ xs = n$
⟨*proof*⟩

lemma *bv_set*: $set\_pmf\ (bv\ n) = \{x{::}bool\ list.\ length\ x = n\}$
⟨*proof*⟩

lemma *len_not_in_bv*: $length\ xs \neq n \Longrightarrow xs \notin set\_pmf\ (bv\ n)$
⟨*proof*⟩

lemma *not_n_bv_0*: $length\ xs \neq n \Longrightarrow pmf\ (bv\ n)\ xs = 0$
⟨*proof*⟩

lemma *bv_comp_bernoulli*: $n < l$
$\Longrightarrow map\_pmf\ (\lambda y.\ y!n)\ (bv\ l) = bernoulli\_pmf\ (5\ /\ 10)$
⟨*proof*⟩

lemma *pmf_2elemlist*: $pmf\ (bv\ (Suc\ 0))\ ([x]) = pmf\ (bv\ 0)\ [] * pmf\ (bernoulli\_pmf\ (5\ /\ 10))\ x$
⟨*proof*⟩

lemma *pmf_moreelemlist*: $pmf\ (bv\ (Suc\ n))\ (x\#xs) = pmf\ (bv\ n)\ xs * pmf\ (bernoulli\_pmf\ (5\ /\ 10))\ x$
⟨*proof*⟩

lemma *list_pmf*: $length\ xs = n \Longrightarrow pmf\ (bv\ n)\ xs = (1\ /\ 2)\hat{}n$
⟨*proof*⟩

lemma *bv_0_notlen*: $pmf\ (bv\ n)\ xs = 0 \Longrightarrow length\ xs \neq n$
⟨*proof*⟩

lemma $length\ xs > n \Longrightarrow pmf\ (bv\ n)\ xs = 0$
⟨*proof*⟩

lemma *map_hd_list_pmf*: $map\_pmf\ hd\ (bv\ (Suc\ n)) = bernoulli\_pmf\ (5\ /\ 10)$
⟨*proof*⟩

lemma *map_tl_list_pmf*: $map\_pmf\ tl\ (bv\ (Suc\ n)) = bv\ n$

⟨*proof*⟩

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*
⟨*proof*⟩

lemma *flip_out_of_bounds*: *y* ≥ *length X* ⟹ *flip y X* = *X*
⟨*proof*⟩

lemma *flip_other*: *y* < *length X* ⟹ *z* < *length X* ⟹ *z* ≠ *y* ⟹ *flip z X* ! *y* = *X* ! *y*
⟨*proof*⟩

lemma *flip_itself*: *y* < *length X* ⟹ *flip y X* ! *y* = (¬ *X* ! *y*)
⟨*proof*⟩

lemma *flip_twice*: *flip i* (*flip i b*) = *b*
⟨*proof*⟩

lemma *flipidiflip*: *y* < *length X* ⟹ *e* < *length X* ⟹ *flip e X* ! *y* = (*if e*=*y then* ~ (*X* ! *y*) *else X* ! *y*)
⟨*proof*⟩

lemma *bernoulli_Not*: *map_pmf Not* (*bernoulli_pmf* (*1* / *2*)) = (*bernoulli_pmf* (*1* / *2*))
⟨*proof*⟩

lemma *inv_flip_bv*: *map_pmf* (*flip i*) (*bv n*) = (*bv n*)
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩
lemma *b1*: *bernoulli_pmf 1 = return_pmf True*
⟨*proof*⟩

lemma *Sum_pmf_0*: *Sum_pmf 0 Da Db = map_pmf Inr Db*
⟨*proof*⟩

lemma *Sum_pmf_1*: *Sum_pmf 1 Da Db = map_pmf Inl Da*
⟨*proof*⟩

definition *Proj1_pmf D = map_pmf* (%*a*. *case a of Inl e* ⇒ *e*) (*cond_pmf D* {*f*. (∃ *e*. *Inl e = f*)})

lemma *A*: (*case_sum* (λ*e*. *e*) (λ*a*. *undefined*)) (*Inl e*) = *e*
⟨*proof*⟩

lemma *B*: *inj* (*case_sum* (λ*e*. *e*) (λ*a*. *undefined*))
⟨*proof*⟩

lemma *none*: *p* >*0* ⟹ *p* < *1* ⟹ (*set_pmf* (*bernoulli_pmf p* ⤜
(λ*b*. *if b then map_pmf Inl Da else map_pmf Inr Db*))
∩ {*f*. (∃ *e*. *Inl e = f*)}) ≠ {}
⟨*proof*⟩
lemma *none2*: *p* >*0* ⟹ *p* < *1* ⟹ (*set_pmf* (*bernoulli_pmf p* ⤜
(λ*b*. *if b then map_pmf Inl Da else map_pmf Inr Db*))
∩ {*f*. (∃ *e*. *Inr e = f*)}) ≠ {}
⟨*proof*⟩

lemma *C*: *set_pmf* (*Proj1_pmf* (*Sum_pmf 0.5 Da Db*)) = *set_pmf Da*
⟨*proof*⟩

thm *integral_measure_pmf*

thm *pmf_cond pmf_cond*[*OF none*]

lemma *proj1_pmf*: **assumes** *p>0 p<1* **shows** *Proj1_pmf* (*Sum_pmf p Da Db*) = *Da*
⟨*proof*⟩

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*⟩

definition *invSum invA invB D x i* == *invA* (*Proj1_pmf D*) *x i* ∧ *invB* (*Proj2_pmf D*) *x i*

lemma *invSum_split*: *p>0* ⟹ *p<1* ⟹ *invA Da x i* ⟹ *invB Db x i* ⟹ *invSum invA invB* (*Sum_pmf p Da Db*) *x i*
⟨*proof*⟩

term (%*a. case a of Inl e* ⇒ *Inl* (*fa e*) | *Inr e* ⇒ *Inr* (*fb e*))
definition *f_on2 fa fb* = (%*a. case a of Inl e* ⇒ *map_pmf Inl* (*fa e*) | *Inr e* ⇒ *map_pmf Inr* (*fb e*))

term *bind_pmf*

lemma *Sum_bind_pmf*: **assumes** *a*: *bind_pmf Da fa* = *Da'* **and** *b*: *bind_pmf Db fb* = *Db'*
shows *bind_pmf* (*Sum_pmf p Da Db*) (*f_on2 fa fb*)
= *Sum_pmf p Da' Db'*
⟨*proof*⟩

definition *sum_map_pmf fa fb* = (%*a. case a of Inl e* ⇒ *Inl* (*fa e*) | *Inr e* ⇒ *Inr* (*fb e*))

lemma *Sum_map_pmf*: **assumes** *a*: *map_pmf fa Da* = *Da'* **and** *b*: *map_pmf*

fb Db = Db'
shows *map_pmf (sum_map_pmf fa fb) (Sum_pmf p Da Db)*
= Sum_pmf p Da' Db'
⟨*proof*⟩

end

5 Randomized Online and Offline Algorithms

theory *Competitive_Analysis*
imports
Prob_Theory
On_Off
begin

5.1 Competitive Analysis Formalized

type_synonym $('s,'is,'r,'a)alg\_on\_step = ('s * 'is \Rightarrow 'r \Rightarrow ('a * 'is)\ pmf)$
type_synonym $('s,'is)alg\_on\_init = ('s \Rightarrow 'is\ pmf)$
type_synonym $('s,'is,'q,'a)alg\_on\_rand = ('s,'is)alg\_on\_init * ('s,'is,'q,'a)alg\_on\_step$

5.1.1 classes of algorithms

definition *deterministic_init* :: $('s,'is)alg\_on\_init \Rightarrow bool$ **where**
$deterministic\_init\ I \longleftrightarrow (\forall\, init.\ card(\ set\_pmf\ (I\ init)) = 1)$

definition *deterministic_step* :: $('s,'is,'q,'a)alg\_on\_step \Rightarrow bool$ **where**
$deterministic\_step\ S \longleftrightarrow (\forall\, i\ is\ q.\ card(\ set\_pmf\ (S\ (i,\ is)\ q)) = 1)$

definition *random_step* :: $('s,'is,'q,'a)alg\_on\_step \Rightarrow bool$ **where**
$random\_step\ S \longleftrightarrow {}^{\sim}\ deterministic\_step\ S$

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'*
⟨*proof*⟩

lemma *T_append*: *length qs* = *length as* ⟹ *T s* (*qs*@[*q*]) (*as*@[*a*]) = *T s qs as* + *t* (*steps s qs as*) *q a*
⟨*proof*⟩

lemma *T_append2*: *length qs* = *length as* ⟹ *T s* (*qs*@*qs'*) (*as*@*as'*) = *T s qs as* + *T* (*steps s qs as*) *qs' as'*
⟨*proof*⟩

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_snoc*: *config'_rand A s* (*rs*@[*r*]) = *config'_rand A s rs* ⤜ *Step_rand A r*
⟨*proof*⟩

lemma *config'_rand_append*: *config'_rand A s* (*xs*@*ys*) = *config'_rand A* (*config'_rand A s xs*) *ys*
⟨*proof*⟩

abbreviation *config_rand* **where**
config_rand A s0 rs == *config'_rand A* ((*fst A s0*) ⤜ (λ*is*. *return_pmf* (*s0*, *is*))) *rs*

lemma *config'_rand_induct*: (∀ *x* ∈ *set_pmf init*. *P* (*fst x*)) ⟹ (⋀*s q a*. *P s* ⟹ *P* (*step s q a*))
⟹ ∀ *x*∈*set_pmf* (*config'_rand A init qs*). *P* (*fst x*)
⟨*proof*⟩

lemma *config_rand_induct*: $P\ s0 \Longrightarrow (\bigwedge s\ q\ a.\ P\ s \Longrightarrow P\ (step\ s\ q\ a)) \Longrightarrow \forall\, x{\in}set\_pmf\ (config\_rand\ A\ s0\ qs).\ P\ (fst\ x)$
⟨*proof*⟩

fun *T_on_rand'* :: (*'state*,*'is*,*'request*,*'answer*) *alg_on_rand* ⇒ (*'state*∗*'is*) *pmf* ⇒ *'request list* ⇒ *real* **where**
T_on_rand' A s [] = *0* |
T_on_rand' A s (*r*#*rs*) = *E* (*s* ⤜ (λ*s*. *bind_pmf* (*snd A s r*) (λ(*a*,*is'*). *return_pmf* (*real* (*t* (*fst s*) *r a*)))))
+ *T_on_rand' A* (*s* ⤜ *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* (*config'_rand A s xs*) *ys*
⟨*proof*⟩

abbreviation *T_on_rand* :: (*'state*,*'is*,*'request*,*'answer*) *alg_on_rand* ⇒ *'state* ⇒ *'request list* ⇒ *real* **where**
T_on_rand A s rs == *T_on_rand' A* (*fst A s* ⤜ (λ*is*. *return_pmf* (*s*,*is*))) *rs*

lemma *T_on_rand_append*: *T_on_rand A s* (*xs*@*ys*) = *T_on_rand A s xs* + *T_on_rand' A* (*config_rand A s xs*) *ys*
⟨*proof*⟩

abbreviation *T_on_rand'_n A s0 xs n* == *T_on_rand' A* (*config'_rand A s0* (*take n xs*)) [*xs*!*n*]

lemma *T_on_rand'_as_sum*: *T_on_rand' A s0 rs* = *sum* (*T_on_rand'_n A s0 rs*) {..<*length rs*}
⟨*proof*⟩

abbreviation *T_on_rand_n A s0 xs n* == *T_on_rand' A* (*config_rand A s0* (*take n xs*)) [*xs*!*n*]

lemma *T_on_rand_as_sum*: *T_on_rand A s0 rs* = *sum* (*T_on_rand_n A s0 rs*) {..<*length rs*}
⟨*proof*⟩

lemma *T_on_rand'_nn*: *T_on_rand' A s qs* ≥ *0*

⟨*proof*⟩

lemma *T_on_rand_nn*: *T_on_rand* (*I*,*S*) *s0 qs* ≥ *0*
⟨*proof*⟩

definition *compet_rand* :: ('*state*,'*is*,'*request*,'*answer*) *alg_on_rand* ⇒ *real* ⇒ '*state set* ⇒ *bool* **where**
compet_rand A c S0 = (∀ *s*∈*S0*. ∃ *b* ≥ *0*. ∀ *rs*. *wf s rs* ⟶ *T_on_rand A s rs* ≤ *c* ∗ *T_opt s rs* + *b*)

5.2 embeding of deterministic into randomized algorithms

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

lemma *T_deter_rand*: *T_off* (λ*s0*. (*off2 A* (*s0*, *x*))) *s0 qs* = *T_on_rand'* (*embed A*) (*return_pmf* (*s0*,*x*)) *qs*
⟨*proof*⟩

lemma *config'_embed*: *config'_rand* (*embed A*) (*return_pmf s0*) *qs* = *return_pmf* (*config' A s0 qs*)
⟨*proof*⟩

lemma *config_embed*: *config_rand* (*embed A*) *s0 qs* = *return_pmf* (*config A s0 qs*)
⟨*proof*⟩

lemma *T_on_embed*: *T_on A s0 qs* = *T_on_rand* (*embed A*) *s0 qs*
⟨*proof*⟩

lemma *T_on'_embed*: *T_on' A* (*s0*,*x*) *qs* = *T_on_rand'* (*embed A*) (*return_pmf* (*s0*,*x*)) *qs*
⟨*proof*⟩

lemma *compet_embed*: *compet A c S0* = *compet_rand* (*embed A*) *c S0*
⟨*proof*⟩

end

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* :: $'a \Rightarrow 'a\ list \Rightarrow 'a\ list$ **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*
⟨*proof*⟩

lemma *mtf0*[*simp*]: *x* ∈ *set xs* ⟹ *mtf x xs* ! *0* = *x*
⟨*proof*⟩

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*⟩

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*}
⟨*proof*⟩

lemma *set_mtf*[*simp*]: *set*(*mtf x xs*) = *set xs*
⟨*proof*⟩

lemma *length_mtf*[*simp*]: *size* (*mtf x xs*) = *size xs*

⟨*proof*⟩

lemma *distinct_mtf*[*simp*]: *distinct* (*mtf x xs*) = *distinct xs*
⟨*proof*⟩

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*⟩

lemma *mtf20*[*simp*]: *mtf2 0 x xs* = *xs*
⟨*proof*⟩

lemma *length_mtf2*[*simp*]: *length* (*mtf2 n x xs*) = *length xs*
⟨*proof*⟩

lemma *set_mtf2*[*simp*]: *set*(*mtf2 n x xs*) = *set xs*
⟨*proof*⟩

lemma *distinct_mtf2*[*simp*]: *distinct* (*mtf2 n x xs*) = *distinct xs*
⟨*proof*⟩

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*⟩

6.3 Function Lxy

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

lemma *Lxy_length_cons*: *length* (*Lxy xs S*) ≤ *length* (*Lxy* (*x*#*xs*) *S*)
⟨*proof*⟩

lemma *Lxy_empty*[*simp*]: *Lxy* [] *S* = []
⟨*proof*⟩

lemma *Lxy_set_filter*: *set* (*Lxy xs S*) = *S* ∩ *set xs*

⟨*proof*⟩

lemma *Lxy_distinct*: *distinct xs* ⟹ *distinct* (*Lxy xs S*)
⟨*proof*⟩

lemma *Lxy_append*: *Lxy* (*xs*@*ys*) *S* = *Lxy xs S* @ *Lxy ys S*
⟨*proof*⟩

lemma *Lxy_snoc*: *Lxy* (*xs*@[*x*]) *S* = (*if x*∈*S then Lxy xs S* @ [*x*] *else Lxy xs S*)
⟨*proof*⟩

lemma *Lxy_not*: *S* ∩ *set xs* = {} ⟹ *Lxy xs S* = []
⟨*proof*⟩

lemma *Lxy_notin*: *set xs* ∩ *S* = {} ⟹ *Lxy xs S* = []
⟨*proof*⟩

lemma *Lxy_in*: *x*∈*S* ⟹ *Lxy* [*x*] *S* = [*x*]
⟨*proof*⟩

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*⟩

lemma *Lxy_mono*: {*x*,*y*} ⊆ *set xs* ⟹ *distinct xs* ⟹ *x* < *y in xs* = *x* < *y in Lxy xs* {*x*,*y*}
⟨*proof*⟩

6.4 List Update as Online/Offline Algorithm

type_synonym *'a state* = *'a list*
type_synonym *answer* = *nat* ∗ *nat list*

definition *step* :: *'a state* ⇒ *'a* ⇒ *answer* ⇒ *'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* ⟨*proof*⟩

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*: $length\ as = length\ rs \Longrightarrow T\ s\ rs\ as \geq length\ rs$
⟨*proof*⟩

lemma *T_off_neq0*: $(\bigwedge rs\ s0.\ size(alg\ s0\ rs) = length\ rs) \Longrightarrow$
$rs \neq [] \Longrightarrow T\_off\ alg\ s0\ rs \neq 0$
⟨*proof*⟩

lemma *length_step*[*simp*]: $length\ (step\ s\ r\ as) = length\ s$
⟨*proof*⟩

lemma *step_Nil_iff*[*simp*]: $step\ xs\ r\ act = [] \longleftrightarrow xs = []$
⟨*proof*⟩

lemma *set_step2*: $set(step\ s\ r\ (mf,sws)) = set\ s$
⟨*proof*⟩

lemma *set_step*: $set(step\ s\ r\ act) = set\ s$
⟨*proof*⟩

lemma *distinct_step*: $distinct(step\ s\ r\ as) = distinct\ s$
⟨*proof*⟩

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

definition $MTF :: ('a,unit)\ alg\_on$ **where**
$MTF = (\lambda\_.\ (),\ \lambda s\ r.\ ((size\ (fst\ s) - 1,[]),\ ()))$

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

lemma *potential*:
fixes $t :: nat \Rightarrow 'a$::*linordered_ab_group_add* **and** $p :: nat \Rightarrow 'a$
assumes *p0*: $p\ 0 = 0$ **and** *ppos*: $\bigwedge n.\ p\ n \geq 0$
and *ub*: $\bigwedge n.\ t\ n + p(n+1) - p\ n \leq u\ n$

shows $(\sum i{<}n.\ t\ i) \leq (\sum i{<}n.\ u\ i)$
⟨*proof*⟩

lemma *potential2*:
fixes $t :: nat \Rightarrow 'a{::}linordered\_ab\_group\_add$ **and** $p :: nat \Rightarrow 'a$
assumes *p0*: $p\ 0 = 0$ **and** *ppos*: $\bigwedge n.\ p\ n \geq 0$
and *ub*: $\bigwedge m.\ m{<}n \Longrightarrow t\ m + p(m{+}1) - p\ m \leq u\ m$
shows $(\sum i{<}n.\ t\ i) \leq (\sum i{<}n.\ u\ i)$
⟨*proof*⟩

abbreviation *before x xs* ≡ $\{y.\ y < x\ in\ xs\}$
abbreviation *after x xs* ≡ $\{y.\ x < y\ in\ xs\}$

lemma *finite_before*[*simp*]: *finite* (*before x xs*)
⟨*proof*⟩

lemma *finite_after*[*simp*]: *finite* (*after x xs*)
⟨*proof*⟩

lemma *before_conv_take*:
$x : set\ xs \Longrightarrow before\ x\ xs = set(take\ (index\ xs\ x)\ xs)$
⟨*proof*⟩

lemma *card_before*: $distinct\ xs \Longrightarrow x : set\ xs \Longrightarrow card\ (before\ x\ xs) = index\ xs\ x$
⟨*proof*⟩

lemma *before_Un*: $set\ xs = set\ ys \Longrightarrow x : set\ xs \Longrightarrow$
$before\ x\ ys = before\ x\ xs \cap before\ x\ ys\ Un\ after\ x\ xs \cap before\ x\ ys$
⟨*proof*⟩

lemma *phi_diff_aux*:
$card\ (Inv\ xs\ ys \cup$
$\{(y,\ x)\ |y.\ y < x\ in\ xs \wedge y < x\ in\ ys\} -$
$\{(x,\ y)\ |y.\ x < y\ in\ xs \wedge y < x\ in\ ys\}) =$
$card(Inv\ xs\ ys) + card(before\ x\ xs \cap before\ x\ ys)$
$- int(card(after\ x\ xs \cap before\ x\ ys))$
(**is** $card(?I \cup ?B - ?A) = card\ ?I + card\ ?b - int(card\ ?a)$)
⟨*proof*⟩

lemma *not_before_Cons*[*simp*]: $\neg\ x < y\ in\ y\ \#\ xs$
⟨*proof*⟩

lemma *before_Cons*[*simp*]:
$y \in set\ xs \Longrightarrow y \neq x \Longrightarrow before\ y\ (x\#xs) = insert\ x\ (before\ y\ xs)$
⟨*proof*⟩

lemma *card_before_le_index*: $card\ (before\ x\ xs) \leq index\ xs\ x$
⟨*proof*⟩

lemma *config_config_length*: $length\ (fst\ (config\ A\ init\ qs)) = length\ init$
⟨*proof*⟩

lemma *config_config_distinct*:
shows $distinct\ (fst\ (config\ A\ init\ qs)) = distinct\ init$
⟨*proof*⟩

lemma *config_config_set*:
shows $set\ (fst\ (config\ A\ init\ qs)) = set\ init$
⟨*proof*⟩

lemma *config_config*:
$set\ (fst\ (config\ A\ init\ qs)) = set\ init$
$\wedge\ distinct\ (fst\ (config\ A\ init\ qs)) = distinct\ init$
$\wedge\ length\ (fst\ (config\ A\ init\ qs)) = length\ init$
⟨*proof*⟩

lemma *config_dist_perm*:
$distinct\ init \Longrightarrow dist\_perm\ (fst\ (config\ A\ init\ qs))\ init$
⟨*proof*⟩

lemma *config_rand_length*: $\forall\, x{\in}set\_pmf\ (config\_rand\ \ A\ init\ qs).\ length\ (fst\ x) = length\ init$
⟨*proof*⟩

lemma *config_rand_distinct*:
shows $\forall\, x \in (config\_rand\ \ A\ init\ qs).\ distinct\ (fst\ x) = distinct\ init$
⟨*proof*⟩

lemma *config_rand_set*:
shows $\forall\, x \in (config\_rand\ \ \ A\ init\ qs).\ set\ (fst\ x) = set\ init$
⟨*proof*⟩

lemma *config_rand*:
$\forall\, x \in (config\_rand\ \ \ A\ \ init\ qs).\ set\ (fst\ x) = set\ init$

$\wedge$ *distinct* (*fst x*) = *distinct init* $\wedge$ *length* (*fst x*) = *length init*
⟨*proof*⟩

lemma *config_rand_dist_perm*:
distinct init $\Longrightarrow \forall x \in$ (*config_rand A init qs*). *dist_perm* (*fst x*) *init*
⟨*proof*⟩

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*⟩

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*
⟨*proof*⟩

lemma *dist_s_A*[*simp*]: *distinct*(*s_A n*)
⟨*proof*⟩

lemma *set_s_A*[*simp*]: *set*(*s_A n*) = *set s0*
⟨*proof*⟩

fun *s_mtf* :: *nat* ⇒ *'a list* **where**
s_mtf 0 = s0 |
s_mtf (*Suc n*) = *mtf* (*rs*!*n*) (*s_mtf n*)

definition *t_mtf* :: *nat* ⇒ *int* **where**
t_mtf n = *index* (*s_mtf n*) (*rs*!*n*) + *1*

definition *T_mtf* :: *nat* ⇒ *int* **where**
T_mtf n = (∑ *i*<*n*. *t_mtf i*)

definition *c_A* :: *nat* ⇒ *int* **where**
c_A n = *index* (*swaps* (*sw_A*!*n*) (*s_A n*)) (*rs*!*n*) + *1*

definition *f_A* :: *nat* ⇒ *int* **where**
f_A n = *min* (*mtf_A*!*n*) (*index* (*swaps* (*sw_A*!*n*) (*s_A n*)) (*rs*!*n*))

definition *p_A* :: *nat* ⇒ *int* **where**
p_A n = *size*(*sw_A*!*n*)

definition *t_A* :: *nat* ⇒ *int* **where**
t_A n = *c_A n* + *p_A n*

definition *T_A* :: *nat* ⇒ *int* **where**
T_A n = (∑ *i*<*n*. *t_A i*)

lemma *length_s_mtf*[*simp*]: *length*(*s_mtf n*) = *length s0*
⟨*proof*⟩

lemma *dist_s_mtf*[*simp*]: *distinct*(*s_mtf n*)
⟨*proof*⟩

lemma *set_s_mtf*[*simp*]: *set* (*s_mtf n*) = *set s0*
⟨*proof*⟩

lemma *dperm_inv*: *dist_perm* (*s_A n*) (*s_mtf n*)
⟨*proof*⟩

definition *Phi* :: *nat* ⇒ *int* (Φ) **where**
Phi n = *card*(*Inv* (*s_A n*) (*s_mtf n*))

lemma *phi0*: *Phi 0* = *0*
⟨*proof*⟩

lemma *phi_pos*: *Phi n* $\geq$ *0*
⟨*proof*⟩

lemma *mtf_ub*: *t_mtf n + Phi (n+1) − Phi n* $\leq$ *2 ∗ c_A n − 1 + p_A n − f_A n*
⟨*proof*⟩

theorem *Sleator_Tarjan*: *T_mtf n* $\leq$ $(\sum i{<}n.\ 2{*}c\_A\ i + p\_A\ i - f\_A\ i) - n$
⟨*proof*⟩

corollary *Sleator_Tarjan′*: *T_mtf n* $\leq$ *2∗T_A n − n*
⟨*proof*⟩

lemma *T_A_nneg*: *0* $\leq$ *T_A n*
⟨*proof*⟩

lemma *T_mtf_ub*: $\forall\, i{<}n.\ rs!i \in set\ s0 \Longrightarrow$ *T_mtf n* $\leq$ *n ∗ size s0*
⟨*proof*⟩

corollary *T_mtf_competitive*: **assumes** *s0* $\neq$ [] **and** $\forall\, i{<}n.\ rs!i \in set\ s0$
shows *T_mtf n* $\leq$ *(2 − 1/(size s0)) ∗ T_A n*
⟨*proof*⟩

lemma *t_A_t*: *n < length rs* $\Longrightarrow$ *t_A n = int (t (s_A n) (rs ! n) (as ! n))*
⟨*proof*⟩

lemma *T_A_eq_lem*: $(\sum i{=}0..{<}length\ rs.\ t\_A\ i) =$
T (s_A 0) (drop 0 rs) (drop 0 as)
⟨*proof*⟩

lemma *T_A_eq*: *T_A (length rs) = T s0 rs as*
⟨*proof*⟩

lemma *nth_off_MTF*: *n < length rs* $\Longrightarrow$ *off2 MTF s rs ! n = (size(fst s) − 1,[])*
⟨*proof*⟩

lemma *t_mtf_MTF*: *n < length rs* $\Longrightarrow$
t_mtf n = int (t (s_mtf n) (rs ! n) (off MTF s rs ! n))
⟨*proof*⟩

lemma *mtf_MTF*: *n < length rs* $\Longrightarrow$ *length s = length s0* $\Longrightarrow$ *mtf (rs ! n)*

$s =$
$\quad$ *step s (rs ! n) (off MTF s0 rs ! n)*
⟨*proof*⟩

lemma *T_mtf_eq_lem*: $(\sum i{=}0..{<}$*length rs. t_mtf i*$) =$
T (s_mtf 0) (drop 0 rs) (drop 0 (off MTF s0 rs))
⟨*proof*⟩

lemma *T_mtf_eq*: *T_mtf (length rs) = T_on MTF s0 rs*
⟨*proof*⟩

corollary *MTF_competitive2*: *s0* $\neq$ [] $\Longrightarrow \forall$ *i*$<$*length rs. rs*!*i* $\in$ *set s0* $\Longrightarrow$
T_on MTF s0 rs $\leq$ (*2* $-$ *1*/(*size s0*)) $*$ *T s0 rs as*
⟨*proof*⟩

corollary *MTF_competitive′*: *T_on MTF s0 rs* $\leq$ *2* $*$ *T s0 rs as*
⟨*proof*⟩

end

theorem *compet_MTF*: **assumes** *s0* $\neq$ [] *distinct s0 set rs* $\subseteq$ *set s0*
shows *T_on MTF s0 rs* $\leq$ (*2* $-$ *1*/(*size s0*)) $*$ *T_opt s0 rs*
⟨*proof*⟩

theorem *compet_MTF′*: **assumes** *distinct s0*
shows *T_on MTF s0 rs* $\leq$ (*2*::*real*) $*$ *T_opt s0 rs*
⟨*proof*⟩

theorem *MTF_is_2_competitive*: *compet MTF 2* {*s . distinct s*}
⟨*proof*⟩

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* :: *real*
assumes [*simp*]: *F* $\neq$ *bot*
assumes *0* $<$ *l*
assumes *ev*:
eventually (λ*n. l* $\leq$ *f n* / *g n*) *F*
eventually (λ*n.* (*f n* $+$ *c*) / (*g n* $+$ *d*) $\leq$ *u*) *F*
and
g: *LIM n F. g n* :> *at_top*

shows $l \leq u$
⟨*proof*⟩

lemma *compet_lb0*:
fixes *a Aon Aoff cruel*
defines *f s0 rs* == *real*(*T_on Aon s0 rs*)
defines *g s0 rs* == *real*(*T_off Aoff s0 rs*)
assumes $\bigwedge$*rs s0*. *size*(*Aoff s0 rs*) = *length rs* **and** $\bigwedge$*n*. *cruel n* ≠ []
assumes *compet Aon c S0* **and** *c*≥*0* **and** *s0* ∈ *S0*
and *l*: *eventually* (λ*n*. *f s0* (*cruel n*) / (*g s0* (*cruel n*) + *a*) ≥ *l*) *sequentially*
and *g*: *LIM n sequentially*. *g s0* (*cruel n*) :> *at_top*
and *l* > *0* **and** $\bigwedge$*n*. *static s0* (*cruel n*)
shows *l* ≤ *c*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *length_sort_sws_le*: *length*(*sort_sws k xs*) ≤ *length xs* ^ *2*
⟨*proof*⟩

lemma *swaps_ins_sws*:
swaps (*ins_sws k x xs*) (*x*#*xs*) = *insort_key k x xs*
⟨*proof*⟩

lemma *swaps_sort_sws*[*simp*]:
swaps (*sort_sws k xs*) *xs* = *sort_key k xs*
⟨*proof*⟩

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 (fst 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) n) ⊆ set s*
⟨*proof*⟩

lemma *static_cruel*: *s ≠ [] ⟹ static s (cruel A (s,is) n)*
⟨*proof*⟩

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)
⟨*proof*⟩

lemma *length_cruel*[*simp*]: *length (cruel A s n) = n*
⟨*proof*⟩

lemma *t_sort_sws*: *t s r (mf, sort_sws k s) ≤ size s ^ 2 + size s + 1*
⟨*proof*⟩

lemma *T_noop*:
n = length rs ⟹ T s rs (replicate n (0, [])) = (∑ r←rs. index s r + 1)
⟨*proof*⟩

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)
⟨*proof*⟩

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*⟩

lemma *T_adv*: **assumes** *l ≠ 0*

shows *T_off* (*adv A*) [*0*..<*l*] (*cruel A* ([*0*..<*l*],*fst A* [*0*..<*l*]) (*Suc n*))
$\leq l^2 + l + 1 + (l + 1) * n$ *div* 2 (**is** *?l* $\leq$ *?r*)
⟨*proof*⟩

The main theorem:

theorem *compet_lb2*:
assumes *compet A c* {*xs*::*nat list*. *size xs* = *l*} **and** $l \neq 0$ **and** $c \geq 0$
shows $c \geq 2*l/(l+1)$
⟨*proof*⟩

end

theory *Bit_Strings*
imports *Complex_Main*
begin

7 Lemmas about BitStrings and sets theirof

7.1 the set of bitstring of length m is finite

lemma *bitstrings_finite*: *finite* {*xs*::*bool list*. *length xs* = *m*}
⟨*proof*⟩

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

lemma *fbool*: *finite* {*xs*. (∀ *i*∈*X*. *xs* ! *i*) ∧ (∀ *i*∈*Y*. ¬ *xs* ! *i*) ∧ *length xs* = *m* ∧ *f* (*xs*!*e*)}
⟨*proof*⟩

fun *witness* :: *nat set* ⇒ *nat* ⇒ *bool list* **where**
witness X 0 = []
|*witness X* (*Suc n*) = (*witness X n*) @ [*n* ∈ *X*]

lemma *witness_length*: *length* (*witness X n*) = *n*
⟨*proof*⟩

lemma *iswitness*: *r*<*n* ⟹ ((*witness X n*)!*r*) = (*r*∈*X*)
⟨*proof*⟩

lemma *card1*: *finite S* ⟹ *finite X* ⟹ *finite Y* ⟹ *X* ∩ *Y* = {} ⟹ *S* ∩ (*X* ∪ *Y*) = {} ⟹ *S*∪*X*∪*Y*={*0*..<*m*} ⟹

card $\{xs.\ (\forall\, i{\in}X.\ xs\ !\ i) \wedge (\forall\, i{\in}Y.\ \neg\ xs\ !\ i)\ \wedge\ length\ xs = m\} = 2\hat{}(m - card\ X - card\ Y)$
⟨*proof*⟩

lemma *card2*: **assumes** *finite X* **and** *finite Y* **and** $X \cap Y = \{\}$ **and** x: $X \cup Y \subseteq \{0..<m\}$
shows *card* $\{xs.\ (\forall\, i{\in}X.\ xs\ !\ i) \wedge (\forall\, i{\in}Y.\ \neg\ xs\ !\ i) \wedge length\ xs = m\} = 2\ \hat{}\ (m - card\ X - card\ Y)$
⟨*proof*⟩

7.3 Average out the second sum for free-absch

lemma *Expactation2or1*: *finite S* $\Longrightarrow$ *finite Tr* $\Longrightarrow$ *finite Fa* $\Longrightarrow$ *card Tr* + *card Fa* + *card S* $\leq l \Longrightarrow$
$S \cap (Tr \cup Fa) = \{\} \Longrightarrow Tr \cap Fa = \{\} \Longrightarrow S \cup Tr \cup Fa \subseteq \{0..<l\} \Longrightarrow$
$(\sum x{\in}\{xs.\ (\forall\, i{\in}Tr.\ xs\ !\ i) \wedge (\forall\, i{\in}Fa.\ \neg\ xs\ !\ i) \wedge length\ xs = l\}.\ \sum j{\in}S.$ *if* $x\ !\ j$ *then 2 else 1*)
$= 3\ /\ 2 * real\ (card\ S) * 2\ \hat{}\ (l - card\ Tr - card\ Fa)$
⟨*proof*⟩

end

8 Effect of mtf2

theory *MTF2_Effects*
imports *Move_to_Front*
begin

lemma *difind_difelem*:
$i < length\ xs \Longrightarrow distinct\ xs \Longrightarrow xs\ !\ j = a \Longrightarrow j < length\ xs \Longrightarrow i \neq j$
$\Longrightarrow\ \sim\ a = xs\ !\ i$
⟨*proof*⟩

lemma *fullchar*: **assumes** $index\ xs\ q < length\ xs$
shows
$(i < length\ xs) =$
$(index\ xs\ q < i \wedge i < length\ xs$
$\vee\ index\ xs\ q = i$
$\vee\ index\ xs\ q - n \leq i \wedge i < index\ xs\ q$
$\vee\ i < index\ xs\ q - n)$

⟨*proof*⟩

lemma *mtf2_effect*:
$q \in set\ xs \Longrightarrow distinct\ xs \Longrightarrow (index\ xs\ q < i \wedge i < length\ xs \longrightarrow ($
$index\ (mtf2\ n\ q\ xs)\ (xs!i) = index\ xs\ (xs!i) \wedge index\ xs\ q < index\ (mtf2\ n$
$q\ xs)\ (xs!i) \wedge index\ (mtf2\ n\ q\ xs)\ (xs!i) < length\ xs))$
$\wedge\ (index\ xs\ q = i \longrightarrow (index\ (mtf2\ n\ q\ xs)\ (xs!i) = index\ xs\ q - n \wedge$
$index\ (mtf2\ n\ q\ xs)\ (xs!i) = index\ xs\ q - n))$
$\wedge\ (index\ xs\ q - n \leq i \wedge i < index\ xs\ q \longrightarrow (index\ (mtf2\ n\ q\ xs)\ (xs!i)$
$= Suc\ (index\ xs\ (xs!i)) \wedge index\ xs\ q - n < index\ (mtf2\ n\ q\ xs)\ (xs!i) \wedge$
$index\ (mtf2\ n\ q\ xs)\ (xs!i) \leq index\ xs\ q))$
$\wedge\ (i < index\ xs\ q - n \longrightarrow (index\ (mtf2\ n\ q\ xs)\ (xs!i) = index\ xs\ (xs!i)$
$\wedge\ index\ (mtf2\ n\ q\ xs)\ (xs!i) < index\ xs\ q - n))$
⟨*proof*⟩

lemma *mtf2_forward_effect1*:
$q \in set\ xs \Longrightarrow distinct\ xs \Longrightarrow index\ xs\ q < i \wedge i < length\ xs$
$\Longrightarrow index\ (mtf2\ n\ q\ xs)\ (xs\ !\ i) = index\ xs\ (xs\ !\ i) \wedge index\ xs\ q <$
$index\ (mtf2\ n\ q\ xs)\ (xs\ !\ i) \wedge index\ (mtf2\ n\ q\ xs)\ (xs\ !\ i) < length\ xs$ **and**

mtf2_forward_effect2: $q \in set\ xs \Longrightarrow distinct\ xs \Longrightarrow index\ xs\ q = i$
$\Longrightarrow index\ (mtf2\ n\ q\ xs)\ (xs!i) = index\ xs\ q - n \wedge index\ xs\ q - n =$
$index\ (mtf2\ n\ q\ xs)\ (xs!i)$ **and**
mtf2_forward_effect3: $q \in set\ xs \Longrightarrow distinct\ xs \Longrightarrow index\ xs\ q - n \leq i$
$\wedge\ i < index\ xs\ q$
$\Longrightarrow index\ (mtf2\ n\ q\ xs)\ (xs!i) = Suc\ (index\ xs\ (xs!i)) \wedge index\ xs\ q -$
$n < index\ (mtf2\ n\ q\ xs)\ (xs!i) \wedge index\ (mtf2\ n\ q\ xs)\ (xs!i) \leq index\ xs\ q$
and
mtf2_forward_effect4: $q \in set\ xs \Longrightarrow distinct\ xs \Longrightarrow i < index\ xs\ q - n$
$\Longrightarrow index\ (mtf2\ n\ q\ xs)\ (xs!i) = index\ xs\ (xs!i) \wedge index\ (mtf2\ n\ q\ xs)$
$(xs!i) < index\ xs\ q - n$
⟨*proof*⟩

lemma *yes*[*simp*]: $index\ xs\ x < length\ xs$
$\Longrightarrow (xs!index\ xs\ x\ ) = x$ ⟨*proof*⟩

lemma *mtf2_forward_effect1′*:
$q \in set\ xs \Longrightarrow distinct\ xs \Longrightarrow index\ xs\ q < index\ xs\ x \wedge index\ xs\ x <$
$length\ xs$
$\Longrightarrow index\ (mtf2\ n\ q\ xs)\ x = index\ xs\ x \wedge index\ xs\ q < index\ (mtf2\ n$
$q\ xs)\ x \wedge index\ (mtf2\ n\ q\ xs)\ x < length\ xs$
⟨*proof*⟩

lemma

mtf2_forward_effect2': $q \in set\ xs \Longrightarrow distinct\ xs \Longrightarrow index\ xs\ q = index\ xs\ x$
$\Longrightarrow index\ (mtf2\ n\ q\ xs)\ (xs!index\ xs\ x) = index\ xs\ q - n \wedge index\ xs\ q - n = index\ (mtf2\ n\ q\ xs)\ (xs!index\ xs\ x)$
⟨*proof*⟩

lemma
mtf2_forward_effect3': $q \in set\ xs \Longrightarrow distinct\ xs \Longrightarrow index\ xs\ q - n \leq index\ xs\ x \Longrightarrow index\ xs\ x < index\ xs\ q$
$\Longrightarrow index\ (mtf2\ n\ q\ xs)\ (xs!index\ xs\ x) = Suc\ (index\ xs\ (xs!index\ xs\ x)) \wedge index\ xs\ q - n < index\ (mtf2\ n\ q\ xs)\ (xs!index\ xs\ x) \wedge index\ (mtf2\ n\ q\ xs)\ (xs!index\ xs\ x) \leq index\ xs\ q$
⟨*proof*⟩

lemma
mtf2_forward_effect4': $q \in set\ xs \Longrightarrow distinct\ xs \Longrightarrow index\ xs\ x < index\ xs\ q - n$
$\Longrightarrow index\ (mtf2\ n\ q\ xs)\ (xs!index\ xs\ x) = index\ xs\ (xs!index\ xs\ x) \wedge index\ (mtf2\ n\ q\ xs)\ (xs!index\ xs\ x) < index\ xs\ q - n$
⟨*proof*⟩

lemma *splitit*: $(index\ xs\ q < i \wedge i < length\ xs \Longrightarrow P)$
$\Longrightarrow (index\ xs\ q = i \Longrightarrow P)$
$\Longrightarrow (index\ xs\ q - n \leq i \wedge i < index\ xs\ q \Longrightarrow P)$
$\Longrightarrow (i < index\ xs\ q - n \Longrightarrow P)$
$\Longrightarrow (i < length\ xs \Longrightarrow P)$
⟨*proof*⟩

lemma *mtf2_forward_beforeq*: $q \in set\ xs \Longrightarrow distinct\ xs \Longrightarrow i < index\ xs\ q$
$\Longrightarrow index\ (mtf2\ n\ q\ xs)\ (xs!i) \leq index\ xs\ q$
⟨*proof*⟩

lemma *x_stays_before_y_if_y_not_moved_to_front*:
assumes $q \in set\ xs\ distinct\ xs\ x \in set\ xs\ y \in set\ xs\ y \neq q$
and $x < y\ in\ xs$
shows $x < y\ in\ (mtf2\ n\ q\ xs)$
⟨*proof*⟩

corollary *swapped_by_mtf2*: $q \in set\ xs \Longrightarrow distinct\ xs \Longrightarrow x \in set\ xs \Longrightarrow$

$y \in set\ xs \Longrightarrow$
$x < y\ in\ xs \Longrightarrow y < x\ in\ (mtf2\ n\ q\ xs) \Longrightarrow y = q$
⟨*proof*⟩

lemma *x_stays_before_y_if_y_not_moved_to_front_2dir*: $q \in set\ xs \Longrightarrow distinct\ xs \Longrightarrow x \in set\ xs \Longrightarrow\ y \in set\ xs \Longrightarrow y \neq q \Longrightarrow$
$x < y\ in\ xs = x < y\ in\ (mtf2\ n\ q\ xs)$
⟨*proof*⟩

lemma *mtf2_backwards_effect1*:
assumes $index\ xs\ q < length\ xs\ q \in set\ xs\ distinct\ xs$
$index\ xs\ q < index\ (mtf2\ n\ q\ xs)\ (xs\ !\ i) \wedge index\ (mtf2\ n\ q\ xs)\ (xs\ !\ i) < length\ xs$
$i < length\ xs$
shows $index\ xs\ q <\ i \wedge i\ < length\ xs$
⟨*proof*⟩

lemma *mtf2_backwards_effect2*:
assumes $index\ xs\ q < length\ xs\ q \in set\ xs\ distinct\ xs\ index\ (mtf2\ n\ q\ xs)\ (xs\ !\ i) = index\ xs\ q - n$
$i < length\ xs$
shows $index\ xs\ q = i$
⟨*proof*⟩

lemma *mtf2_backwards_effect3*:
assumes $index\ xs\ q < length\ xs\ q \in set\ xs\ distinct\ xs$
$index\ xs\ q - n < index\ (mtf2\ n\ q\ xs)\ (xs\ !\ i) \wedge index\ (mtf2\ n\ q\ xs)\ (xs\ !\ i) \leq index\ xs\ q$
$i < length\ xs$
shows $index\ xs\ q - n \leq i \wedge i < index\ xs\ q$
⟨*proof*⟩

lemma *mtf2_backwards_effect4*:
assumes $index\ xs\ q < length\ xs\ q \in set\ xs\ distinct\ xs$
$index\ (mtf2\ n\ q\ xs)\ (xs\ !\ i) < index\ xs\ q - n$
$i < length\ xs$
shows $i < index\ xs\ q - n$
⟨*proof*⟩

lemma *mtf2_backwards_effect4'*:
assumes $index\ xs\ q < length\ xs\ q \in set\ xs\ distinct\ xs$
$index\ (mtf2\ n\ q\ xs)\ x < index\ xs\ q - n$
$x \in set\ xs$

shows $(index\ xs\ x) < index\ xs\ q - n$
⟨*proof*⟩

lemma
assumes *distA*: $distinct\ A$ **and**
asm: $q \in set\ A$
shows
mtf2_mono: $q< x\ in\ A \Longrightarrow q < x\ in\ (mtf2\ n\ q\ A)$ **and**
mtf2_q_after: $index\ (mtf2\ n\ q\ A)\ q = index\ A\ q - n$
⟨*proof*⟩

8.1 effect of mtf2 on index

lemma *swapsthrough*: $distinct\ xs \Longrightarrow q \in set\ xs \Longrightarrow index\ (\ swaps\ [index\ xs\ q - entf..<index\ xs\ q]\ xs\ )\ q = index\ xs\ q - entf$
⟨*proof*⟩

term *mtf2*
lemma *mtf2_moves_to_front*: $distinct\ xs \Longrightarrow q \in set\ xs \Longrightarrow index\ (mtf2\ (length\ xs)\ q\ xs)\ q = 0$
⟨*proof*⟩

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 $x < y\ in\ mtf2\ n\ q\ xs$
$= x < y\ in\ xs$
⟨*proof*⟩

lemma *mtf2_moves_to_frontm1*: $distinct\ xs \Longrightarrow q \in set\ xs \Longrightarrow index\ (mtf2\ (length\ xs\ -1)\ q\ xs)\ q = 0$
⟨*proof*⟩

lemma *mtf2_moves_to_front'*: $distinct\ xs \Longrightarrow y \in set\ xs \Longrightarrow x \in set\ xs \Longrightarrow x{\neq}y \Longrightarrow x < y\ in\ mtf2\ (length\ xs{-}1)\ x\ xs = True$
⟨*proof*⟩

lemma *mtf2_moves_to_front''*: $distinct\ xs \Longrightarrow y \in set\ xs \Longrightarrow x \in set\ xs \Longrightarrow x{\neq}y \Longrightarrow x < y\ in\ mtf2\ (length\ xs)\ x\ xs = True$

⟨*proof*⟩

end

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 (∑ *x*∈*A*. *f x*) − (∑ *x*∈*B*. *g x*)
= (∑ *x*∈(*A* ∩ *B*). *f x* − *g x*) + (∑ *x*∈*A*−*B*. *f x*) − (∑ *x*∈*B*−*A*. *g x*)
⟨*proof*⟩

lemma *sum_my2*: (∀ *x*∈*A*. *f x* = *g x*) ⟹ (∑ *x*∈*A*. *f x*) = (∑ *x*∈*A*. *g x*)
⟨*proof*⟩

9.1 Definition of BIT

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

lemma ~ *deterministic_init BIT_init*
⟨*proof*⟩

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*
⟨*proof*⟩

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*: ∀ ((*free*,*paid*),_) ∈ (*BIT_step s q*). *paid*=[]
⟨*proof*⟩

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* ∘ *snd*) (*config*′*_rand* (*BIT_init*, *BIT_step*) *init qs*)
= *map_pmf* (*snd* ∘ *snd*) *init*
⟨*proof*⟩

lemma *config_n_init*: *map_pmf* (*snd* ∘ *snd*) (*config_rand* (*BIT_init*, *BIT_step*) *s0 qs*) = *return_pmf s0*
⟨*proof*⟩

lemma *config_n_init2*: ∀ (_,(_,*x*)) ∈ *set_pmf* (*config_rand* (*BIT_init*, *BIT_step*) *init qs*). *x* = *init*
⟨*proof*⟩
lemma *config_n_init3*: ∀ *x* ∈ *set_pmf* (*config_rand* (*BIT_init*, *BIT_step*) *init qs*). *snd* (*snd x*) = *init*
⟨*proof*⟩

lemma *config*′*_n_bv*: **fixes** *qs init n*
shows *map_pmf* (*snd* ∘ *snd*) *init* = *return_pmf s0*
⟹ *map_pmf* (*fst* ∘ *snd*) *init* = *bv* (*length s0*)
⟹ *map_pmf* (*snd* ∘ *snd*) (*config*′*_rand* (*BIT_init*, *BIT_step*) *init qs*)
= *return_pmf s0*

$\wedge$ *map_pmf* (*fst* $\circ$ *snd*) (*config'_rand* (*BIT_init*, *BIT_step*) *init qs*) = *bv* (*length s0*)
⟨*proof*⟩

lemma *config_n_bv_2*: *map_pmf* (*snd* $\circ$ *snd*) (*config_rand* (*BIT_init*, *BIT_step*) *s0 qs*) = *return_pmf s0*
$\wedge$ *map_pmf* (*fst* $\circ$ *snd*) (*config_rand* (*BIT_init*, *BIT_step*) *s0 qs*) = *bv* (*length s0*)
⟨*proof*⟩

lemma *config_n_bv*: *map_pmf* (*fst* $\circ$ *snd*) (*config_rand* (*BIT_init*, *BIT_step*) *s0 qs*) = *bv* (*length s0*)
⟨*proof*⟩

lemma *config_n_fst_init_length*: $\forall$ (_,(*x*,_)) $\in$ *set_pmf* (*config_rand* (*BIT_init*, *BIT_step*) *s0 qs*). *length x* = *length s0*
⟨*proof*⟩

lemma *config_n_fst_init_length2*: $\forall$ *x* $\in$ *set_pmf* (*config_rand* (*BIT_init*, *BIT_step*) *s0 qs*). *length* (*fst* (*snd x*)) = *length s0*
⟨*proof*⟩

lemma *fperms*: *finite* {*x*::'*a list*. *length x* = *length init* $\wedge$ *distinct x* $\wedge$ *set x* = *set init*}
⟨*proof*⟩

lemma *finite_config_BIT*: **assumes** [*simp*]: *distinct init*
shows *finite* (*set_pmf* (*config_rand* (*BIT_init*, *BIT_step*) *init qs*)) (**is** *finite ?D*)
⟨*proof*⟩

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* :: *answer list*
fixes *qs* :: '*a list*

fixes *init* :: *'a list*
assumes *dist_init*[*simp*]: *distinct init*
assumes *len_acts*: *length acts* = *length qs*
begin

lemma *setinit*: (*index init*) ‘ *set init* = {*0*..<*length init*}
⟨*proof*⟩

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* (λ*x*. *Suc x* < *length init*)) *paid_A'*

lemma *len_paid_A*[*simp*]: *length paid_A* = *length qs*
⟨*proof*⟩
lemma *len_paid_A'*[*simp*]: *length paid_A'* = *length qs*
⟨*proof*⟩

lemma *paidAnm_inbound*: *n* < *length paid_A* ⟹ *m* < *length*(*paid_A*!*n*)
⟹ (*Suc* ((*paid_A*!*n*)!(*length* (*paid_A* ! *n*) − *Suc m*))) < *length init*
⟨*proof*⟩

fun *s_A'* :: *nat* ⇒ *'a list* **where**
s_A' 0 = *init* |
s_A'(*Suc n*) = *step* (*s_A' n*) (*qs*!*n*) (*free_A*!*n*, *paid_A'*!*n*)

lemma *length_s_A'*[*simp*]: *length*(*s_A' n*) = *length init*
⟨*proof*⟩

lemma *dist_s_A'*[*simp*]: *distinct*(*s_A' n*)
⟨*proof*⟩

lemma *set_s_A'*[*simp*]: *set*(*s_A' n*) = *set init*
⟨*proof*⟩

fun *s_A* :: *nat* ⇒ *'a list* **where**
s_A 0 = *init* |
s_A(*Suc n*) = *step* (*s_A n*) (*qs*!*n*) (*free_A*!*n*, *paid_A*!*n*)

lemma *length_s_A*[*simp*]: *length*(*s_A n*) = *length init*
⟨*proof*⟩

lemma *dist_s_A*[*simp*]: *distinct*(*s_A n*)
⟨*proof*⟩

lemma *set_s_A*[*simp*]: *set*(*s_A n*) = *set init*
⟨*proof*⟩

lemma *cost_paidAA'*: *n* < *length paid_A'* ⟹ *length* (*paid_A*!*n*) ≤ *length* (*paid_A'*!*n*)
⟨*proof*⟩

lemma *swaps_filtered*: *swaps* (*filter* (λ*x*. *Suc x* < *length xs*) *ys*) *xs* = *swaps* (*ys*) *xs*
⟨*proof*⟩

lemma *sAsA'*: *n* < *length paid_A'* ⟹ *s_A' n* = *s_A n*
⟨*proof*⟩

lemma *sAsA''*: *n* < *length qs* ⟹ *s_A n* = *s_A' n*
⟨*proof*⟩

definition *t_BIT* :: *nat* ⇒ *real* **where**
t_BIT n = *T_on_rand_n BIT init qs n*

definition *T_BIT* :: *nat* ⇒ *real* **where**
T_BIT n = (∑ *i*<*n*. *t_BIT i*)

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

definition *f_A* :: *nat* ⇒ *int* **where**
f_A n = *min* (*free_A*!*n*) (*index* (*swaps* (*paid_A*!*n*) (*s_A n*)) (*qs*!*n*))

definition *p_A* :: *nat* ⇒ *int* **where**
p_A n = *size*(*paid_A*!*n*)

definition *t_A* :: *nat* ⇒ *int* **where**
t_A n = *c_A n* + *p_A n*

definition $c\_A' :: nat \Rightarrow int$ **where**
$c\_A'\ n = index\ (swaps\ (paid\_A'!n)\ (s\_A'\ n))\ (qs!n) + 1$

definition $p\_A' :: nat \Rightarrow int$ **where**
$p\_A'\ n = size(paid\_A'!n)$
definition $t\_A' :: nat \Rightarrow int$ **where**
$t\_A'\ n = c\_A'\ n + p\_A'\ n$

lemma $t\_A\_A'\_leq$: $n < length\ paid\_A' \Longrightarrow t\_A\ n \leq t\_A'\ n$
⟨*proof*⟩

definition $T\_A' :: nat \Rightarrow int$ **where**
$T\_A'\ n = (\sum i{<}n.\ t\_A'\ i)$

definition $T\_A :: nat \Rightarrow int$ **where**
$T\_A\ n = (\sum i{<}n.\ t\_A\ i)$

lemma $T\_A\_A'\_leq$: $n \leq length\ paid\_A' \Longrightarrow T\_A\ n \leq T\_A'\ n$
⟨*proof*⟩

lemma $T\_A\_A'\_leq'$: $n \leq length\ qs \Longrightarrow T\_A\ n \leq T\_A'\ n$
⟨*proof*⟩

fun $s'\_A :: nat \Rightarrow nat \Rightarrow 'a\ list$ **where**
$s'\_A\ n\ 0 = s\_A\ n$
$|\ (s'\_A\ n\ (Suc\ m)) = swap\ ((paid\_A\ !\ n)!(length\ (paid\_A\ !\ n)\ -(Suc\ m))$
$)\ (s'\_A\ n\ m)$

lemma $set\_s'\_A[simp]$: $set\ (s'\_A\ n\ m) = set\ init$
⟨*proof*⟩

lemma $len\_s'\_A[simp]$: $length\ (s'\_A\ n\ m) = length\ init$
⟨*proof*⟩

lemma $distperm\_s'\_A[simp]$: $dist\_perm\ (s'\_A\ n\ m)\ init$
⟨*proof*⟩

lemma $s'A\_m\_le$: $m \leq (length\ (paid\_A\ !\ n)) \Longrightarrow swaps\ (drop\ (length\ (paid\_A$
$!\ n)\ -\ m)\ (paid\_A\ !\ n))\ (s\_A\ n) = s'\_A\ n\ m$
⟨*proof*⟩

lemma *s'A_m*: *swaps* (*paid_A* ! *n*) (*s_A n*) = *s'_A n* (*length* (*paid_A* ! *n*))
⟨*proof*⟩

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* < *length paid_A* **and** *2*: *m* < *length* (*paid_A* ! *n*)
shows *gebub n m* < *length init*
⟨*proof*⟩

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*⟩

lemma *Inv_empty2*: *length d* = *0* ⟹ *Inv c d* = {}
⟨*proof*⟩

corollary *Inv_empty3*: *length init* = *0* ⟹ *Inv c* (*s_A n*) = {}
⟨*proof*⟩

lemma *phi_empty2*: *length init* = *0* ⟹ *phi n* (*c*,(*b*,*i*)) = *0*
⟨*proof*⟩

lemma *phi_nonzero*: *phi n* (*c*,(*b*,*i*)) ≥ *0*
⟨*proof*⟩

definition *Phi* :: *nat* ⇒ *real* (Φ) **where**
Phi n = *E*(*map_pmf* (*φ n*) (*config'' BIT qs init n*))

definition *PhiPlus* :: *nat* ⇒ *real* (Φ^+) **where**
PhiPlus n = (*let*
nextconfig = *bind_pmf* (*config'' BIT qs init n*)
(λ(*s*,*is*). *bind_pmf* (*BIT_step* (*s*,*is*) (*qs*!*n*)) (λ(*a*,*nis*). *return_pmf*

$(step\ s\ (qs!n)\ a,nis))\ )$
in
$E(\ map\_pmf\ (phi\ (Suc\ n))\ nextconfig)\ )$

lemma *PhiPlus_is_Phi_Suc*: $n<length\ qs \Longrightarrow PhiPlus\ n = Phi\ (Suc\ n)$
⟨*proof*⟩

lemma *phi0*: $Phi\ 0 = 0$ ⟨*proof*⟩

lemma *phi_pos*: $Phi\ n \geq 0$
⟨*proof*⟩

9.3.3 Helper lemmas

lemma *swap_subs*: $dist\_perm\ X\ Y \Longrightarrow Inv\ X\ (swap\ z\ Y) \subseteq Inv\ X\ Y \cup \{(Y\ !\ z,\ Y\ !\ Suc\ z)\}$
⟨*proof*⟩

9.3.4 InvOf

term *Inv*
abbreviation $InvOf\ y\ bits\ as \equiv \{(x,y)|x.\ x < y\ in\ bits \wedge y < x\ in\ as\}$

lemma $InvOf\ y\ xs\ ys = \{(x,y)|x.\ (x,y)\in Inv\ xs\ ys\}$
⟨*proof*⟩

lemma $InvOf\ y\ xs\ ys \subseteq Inv\ xs\ ys$ ⟨*proof*⟩

lemma *numberofIsbeschr*: **assumes**
distxsys: *dist_perm xs ys* **and**
yinxs: $y \in set\ xs$
shows $index\ xs\ y \leq index\ ys\ y + card\ (InvOf\ y\ xs\ ys)$
(**is** $?iBit \leq ?iA + card\ ?I$)
⟨*proof*⟩

lemma $length\ init = 0 \Longrightarrow length\ xs = length\ init \Longrightarrow t\ xs\ q\ (mf, sws) = 1 + length\ sws$
⟨*proof*⟩

lemma *integr_index*: $integrable\ (measure\_pmf\ (config''(BIT\_init, BIT\_step)\ qs\ init\ n))$
$(\lambda(s,\ is).\ real\ (Suc\ (index\ s\ (qs\ !\ n))))$

⟨*proof*⟩

9.3.5 Upper Bound on the Cost of BIT

lemma *t_BIT_ub2*: $(qs!n) \notin set\ init \Longrightarrow t\_BIT\ n \leq Suc(size\ init)$
⟨*proof*⟩

lemma *t_BIT_ub*: $(qs!n) \in set\ init \Longrightarrow t\_BIT\ n \leq size\ init$
⟨*proof*⟩

lemma *T_BIT_ub*: $\forall\, i{<}n.\ qs!i \in set\ init \Longrightarrow T\_BIT\ n \leq n * size\ init$
⟨*proof*⟩

9.3.6 Main Lemma

lemma *myub*: $n < length\ qs \Longrightarrow t\_BIT\ n + Phi(n + 1) - Phi\ n \leq (7\ /\ 4) * t\_A\ n - 3/4$
⟨*proof*⟩

9.3.7 Lift the Result to the Whole Request List

lemma *T_BIT_absch_le*: **assumes** *nqs*: $n \leq length\ qs$
shows $T\_BIT\ n \leq (7\ /\ 4) * T\_A\ n - 3/4{*}n$
⟨*proof*⟩

lemma *T_BIT_absch*: **assumes** *nqs*: $n \leq length\ qs$
shows $T\_BIT\ n \leq (7\ /\ 4) * T\_A'\ n - 3/4{*}n$
⟨*proof*⟩

lemma *T_A_nneg*: $0 \leq T\_A\ n$
⟨*proof*⟩

lemma *T_BIT_eq*: $T\_BIT\ (length\ qs) = T\_on\_rand\ BIT\ init\ qs$
⟨*proof*⟩

corollary *T_BIT_competitive*: **assumes** $n \leq length\ qs$ **and** $init \neq []$ **and** $\forall\, i{<}n.\ qs!i \in set\ init$
shows $T\_BIT\ n \leq ((7\ /\ 4) - 3/(4 * size\ init)) * T\_A'\ n$
⟨*proof*⟩

lemma *t_A'_t*: $n < length\ qs \Longrightarrow t\_A'\ n = int\ (t\ (s\_A'\ n)\ (qs!n)\ (acts\ !\ n))$
⟨*proof*⟩

lemma *T_A'_eq_lem*: $(\sum i{=}0..{<}length\ qs.\ t\_A'\ i) =$
$T\ (s\_A'\ 0)\ (drop\ 0\ qs)\ (drop\ 0\ acts)$
⟨*proof*⟩

lemma *T_A'_eq*: $T\_A'\ (length\ qs) = T\ init\ qs\ acts$
⟨*proof*⟩

corollary *BIT_competitive3*: $init \neq [] \Longrightarrow \forall\, i{<}length\ qs.\ qs!i \in set\ init \Longrightarrow$
$T\_BIT\ (length\ qs) \leq (\ (7/4) - 3\ /\ (4 * length\ init)) * T\ init\ qs\ acts$
⟨*proof*⟩

corollary *BIT_competitive2*: $init \neq [] \Longrightarrow \forall\, i{<}length\ qs.\ qs!i \in set\ init \Longrightarrow$
$T\_on\_rand\ BIT\ init\ qs \leq (\ (7/4) - 3\ /\ (4 * length\ init)) * T\ init\ qs\ acts$
⟨*proof*⟩

corollary *BIT_absch_le*: $init \neq [] \Longrightarrow$
$T\_on\_rand\ BIT\ init\ qs \leq (7\ /\ 4) * (T\ init\ qs\ acts) - 3/4 * length\ qs$
⟨*proof*⟩

end

9.3.8 Generalize Competitivness of BIT

lemma *setdi*: $set\ xs = \{0..{<}length\ xs\} \Longrightarrow distinct\ xs$
⟨*proof*⟩

theorem *compet_BIT*: **assumes** $init \neq []$ $distinct\ init$ $set\ qs \subseteq set\ init$
shows $T\_on\_rand\ BIT\ init\ qs \leq (\ (7/4) - 3\ /\ (4 * length\ init)) * T\_opt\ init\ qs$
⟨*proof*⟩

theorem *compet_BIT4*: **assumes** $init \neq []$ $distinct\ init$
shows $T\_on\_rand\ BIT\ init\ qs \leq 7/4 * T\_opt\ init\ qs$
⟨*proof*⟩

theorem *compet_BIT_2*:
$compet\_rand\ BIT\ (7/4)\ \{init.\ init \neq [] \wedge distinct\ init\}$
⟨*proof*⟩

end

10 Partial cost model

theory *Partial_Cost_Model*
imports *Move_to_Front*
begin

definition $t_p :: 'a\ state \Rightarrow 'a \Rightarrow answer \Rightarrow nat$ **where**
$t_p\ s\ q\ a = (let\ (mf,sws) = a\ in\ index\ (swaps\ sws\ s)\ q + size\ sws)$

notation $(latex)\ t_p\ (t^*)$

lemma t_pt: $t_p\ s\ q\ a + 1 = t\ s\ q\ a\ \langle proof \rangle$

interpretation *On_Off step* t_p *static* $\langle proof \rangle$

abbreviation $T_p == T$
abbreviation $T_p\_opt == T\_opt$
abbreviation $T_p\_on == T\_on$
abbreviation $T_p\_on\_rand' == T\_on\_rand'$
abbreviation $T_p\_on\_n == T\_on\_n$
abbreviation $T_p\_on\_rand == T\_on\_rand$
abbreviation $T_p\_on\_rand\_n == T\_on\_rand\_n$
abbreviation $config_p == config$
abbreviation $compet_p == compet$

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*
⟨*proof*⟩

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* }
⟨*proof*⟩

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*⟩

fun *substW* :: *nat list* ⇒ (*nat* ⇒ *nat rexp option*) ⇒ *nat rexp* **where**

$substW\ as\ \sigma = subst\ (w2rexp\ as)\ \sigma$

fun $substL :: nat\ lang \Rightarrow (nat \Rightarrow nat\ rexp\ option) \Rightarrow nat\ rexp\ set$ **where**
$substL\ S\ \sigma = \{substW\ a\ \sigma | a.\ a \in S\}$

fun $L :: nat\ rexp\ set \Rightarrow nat\ lang$ **where**
$L\ S = (\bigcup r{\in}S.\ lang\ r)$

lemma *L_mono*: $S1 \subseteq S2 \Longrightarrow L\ S1 \subseteq L\ S2$
⟨*proof*⟩

definition $concS :: 'b\ rexp\ set \Rightarrow 'b\ rexp\ set \Rightarrow 'b\ rexp\ set$ **where**
$concS\ S1\ S2 = \{Times\ a\ b | a\ b.\ a{\in}S1 \wedge b{\in}S2\}$

lemma *substL_conc*: $L\ (substL\ (L1\ @@\ L2)\ \sigma) = L\ (concS\ (substL\ L1\ \sigma)$
$(substL\ L2\ \sigma))$
⟨*proof*⟩

lemma *L_conc*: $L(concS\ M1\ M2) = (L\ M1)\ @@\ (L\ M2)$
⟨*proof*⟩

lemma $L(M1 \cup M2) = (L\ M1) \cup (L\ M2)$
⟨*proof*⟩

fun $verund :: 'b\ rexp\ list \Rightarrow 'b\ rexp$ **where**
$verund\ [] = Zero$
$|\ verund\ [r] = r$
$|\ verund\ (r\#rs) = Plus\ r\ (verund\ rs)$

lemma *lang_verund*: $r \in L\ (set\ rs) = (r \in lang\ (verund\ rs))$
⟨*proof*⟩

lemma *obtainit*:
assumes $r \in lang\ (verund\ rs)$
shows $\exists x{\in}(set\ (rs{::}nat\ rexp\ list)).\ r \in lang\ x$
⟨*proof*⟩

lemma *lang_verund4*: $L\ (set\ rs) = lang\ (verund\ rs)$
⟨*proof*⟩

lemma *lang_verund1*: $r \in L\ (set\ rs) \Longrightarrow r \in lang\ (verund\ rs)$
⟨*proof*⟩

lemma *lang_verund2*: $r \in lang\ (verund\ rs) \Longrightarrow r \in L\ (set\ rs)$
⟨*proof*⟩

definition *starS* :: *'b rexp set* ⇒ *'b rexp set* **where**
starS S = {*Star* (*verund xs*)|*xs*. *set xs* ⊆ *S*}

lemma [] ∈ *L* (*starS S*)
⟨*proof*⟩

lemma *power_mono*: $L1 \subseteq L2 \Longrightarrow (L1::'a\ lang)\ \hat{}\hat{}\ n \subseteq L2\ \hat{}\hat{}\ n$
⟨*proof*⟩

lemma *star_mono*: $L1 \subseteq L2 \Longrightarrow star\ L1 \subseteq star\ L2$
⟨*proof*⟩

lemma *Lstar*: $L(starS\ M) = star\ (\ L(M)\ )$
⟨*proof*⟩

lemma *substL_star*: $L\ (substL\ (star\ L1)\ \sigma) = L\ (starS\ (substL\ L1\ \sigma))$
⟨*proof*⟩

lemma *substituitionslemma*:
fixes *E* :: *nat rexp*
shows $L\ (substL\ (\ lang(E)\ )\ \sigma) = lang\ (subst\ E\ \sigma)$
⟨*proof*⟩

corollary *lift*: $lang\ e1 = lang\ e2 \Longrightarrow lang\ (subst\ e1\ \sigma) = lang\ (subst\ e2\ \sigma)$
⟨*proof*⟩

11.1 Examples

lemma $lang\ (Plus\ (Atom\ (x::nat))\ (Atom\ x))\ = lang\ (Atom\ x)$
⟨*proof*⟩

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_4cases* (*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*),*Star*(*Times* (*Atom y*)(*Atom x*)),(*Atom x*)],
seq[(*Atom x*),(*Atom x*)]]

definition *L_A x y* = *seq*[*question* (*Atom x*),(*Atom y*), (*Atom y*)]
definition *L_B x y* = *seq*[*question* (*Atom x*),(*Atom y*),(*Atom x*),*Star*(*Times* (*Atom y*)(*Atom x*)),(*Atom y*),(*Atom y*)]
definition *L_C x y* = *seq*[*question* (*Atom x*),(*Atom y*),(*Atom x*),*Star*(*Times* (*Atom y*)(*Atom x*)),(*Atom x*)]
definition *L_D x y* = *seq*[(*Atom x*),(*Atom x*)]

lemma *L_4cases x y* = *verund* [*L_A x y*, *L_B x y*, *L_C x y*, *L_D x y*]
⟨*proof*⟩

definition *L_lasthasxx x y* = (*Plus* (*seq*[*question* (*Atom x*), *Star*(*Times* (*Atom y*)(*Atom x*)),(*Atom y*),(*Atom y*)])
(*seq*[*question* (*Atom y*), *Star*(*Times*(*Atom x*) (*Atom y*)),(*Atom x*),(*Atom x*)]))

lemma *lastxx_com*: *lang* (*L_lasthasxx* (*x*::*nat*) *y*) = *lang* (*L_lasthasxx y x*) (**is** *lang ?A* = *lang ?B*)
⟨*proof*⟩

lemma *lastxx_is_4cases*: *lang* (*L_4cases x y*) = *lang* (*L_lasthasxx x y*) (**is** *lang ?A* = *lang ?B*)
⟨*proof*⟩

definition *myUNIV x y* = *Star* (*Plus* (*Atom x*) (*Atom y*))

lemma *myUNIV_alle*: *lang* (*myUNIV x y*) = {*xs*. *set xs* ⊆ {*x*,*y*}}
⟨*proof*⟩

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*⟩

definition $mycasexxy\ x\ y = Plus\ (seq[Star\ (Plus\ (Atom\ x)\ (Atom\ y)),\ Atom\ x,\ Atom\ x,\ Atom\ y])$
$(seq[Star\ (Plus\ (Atom\ x)\ (Atom\ y)),\ Atom\ y,\ Atom\ y,\ Atom\ x])$
definition $mycasexyx\ x\ y = Plus\ (seq[Star\ (Plus\ (Atom\ x)\ (Atom\ y)),\ Atom\ x,\ Atom\ y,\ Atom\ x])$
$(seq[Star\ (Plus\ (Atom\ x)\ (Atom\ y)),\ Atom\ y,\ Atom\ x,\ Atom\ y])$
definition $mycasexx\ x\ y = Plus\ (seq[Star\ (Plus\ (Atom\ x)\ (Atom\ y)),\ Atom\ x,\ \ Atom\ x])$
$(seq[Star\ (Plus\ (Atom\ x)\ (Atom\ y)),\ Atom\ y,\ Atom\ y])$
definition $mycasexy\ x\ y = Plus\ (seq[Atom\ x,\ \ Atom\ y])\ (seq[Atom\ y,\ Atom\ x])$
definition $mycasex\ x\ y = Plus\ (Atom\ y)\ (Atom\ x)$

definition $mycases\ x\ y = Plus$
$(mycasexxy\ x\ y)$
$(Plus\ (mycasexyx\ x\ y)$
$(Plus\ (mycasexx\ x\ y)$
$(Plus\ (mycasexy\ x\ y)\ (Plus\ (mycasex\ x\ y)\ (One)))))$

lemma *mycases_char*: $lang\ (myUNIV\ (x::nat)\ y) = lang\ (mycases\ x\ y)$ (**is** $lang\ ?A = lang\ ?B$)
⟨*proof*⟩

end

12 OPT2

theory *OPT2*
imports

Partial_Cost_Model
RExp_Var
begin

lemma (*N*::*nat set*) ≠ {} ⟹ *Inf N* : *N*
⟨*proof*⟩

lemma *nn_contains_Inf*:
fixes *S* :: *nat set*
assumes *nn*: *S* ≠ {}
shows *Inf S* ∈ *S*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *OPT2x*: *OPT2* (*x*#*σ'*) [*x*,*y*] = (*0*,[])#(*OPT2* *σ'* [*x*,*y*])
⟨*proof*⟩

lemma *swapOpt*: T_{p}_*opt* [*x*,*y*] *σ* ≤ *1* + T_{p}_*opt* [*y*,*x*] *σ*
⟨*proof*⟩

lemma *tt*: *a* ∈ {*x*,*y*} ⟹ *OPT2* (*rest1*) (*step* [*x*,*y*] *a* (*hd* (*OPT2* (*a* # *rest1*) [*x*, *y*])))
= *tl* (*OPT2* (*a* # *rest1*) [*x*, *y*])
⟨*proof*⟩

lemma *splitqsallg*: *Strat* ≠ [] ⟹ *a* ∈ {*x*,*y*} ⟹
t_p [*x*, *y*] *a* (*hd* (*Strat*)) +
(*let L*=*step* [*x*,*y*] *a* (*hd* (*Strat*))
in T_p *L* (*rest1*) (*tl Strat*)) = T_p [*x*, *y*] (*a* # *rest1*) *Strat*
⟨*proof*⟩

lemma *splitqs*: $a \in \{x,y\} \Longrightarrow T_p\ [x,\ y]\ (a\ \#\ rest1)\ (OPT2\ (a\ \#\ rest1)\ [x,\ y])$
$= t_p\ [x,\ y]\ a\ (hd\ (OPT2\ (a\ \#\ rest1)\ [x,\ y]))\ +$
$(let\ L{=}step\ [x,y]\ a\ (hd\ (OPT2\ (a\ \#\ rest1)\ [x,\ y]))$
$in\ T_p\ L\ (rest1)\ (OPT2\ (rest1)\ L))$
⟨*proof*⟩

lemma *tpx*: $t_p\ [x,\ y]\ x\ (hd\ (OPT2\ (x\ \#\ rest1)\ [x,\ y])) = 0$
⟨*proof*⟩

lemma *yup*: $T_p\ [x,\ y]\ (x\ \#\ rest1)\ (OPT2\ (x\ \#\ rest1)\ [x,\ y])$
$= (let\ L{=}step\ [x,y]\ x\ (hd\ (OPT2\ (x\ \#\ rest1)\ [x,\ y]))$
$in\ T_p\ L\ (rest1)\ (OPT2\ (rest1)\ L))$
⟨*proof*⟩

lemma *swapsxy*: $A \in \{\ [x,y],\ [y,x]\} \Longrightarrow swaps\ sws\ A \in \{\ [x,y],\ [y,x]\}$
⟨*proof*⟩

lemma *mtf2xy*: $A \in \{\ [x,y],\ [y,x]\} \Longrightarrow r{\in}\{x,y\} \Longrightarrow mtf2\ a\ r\ A \in \{\ [x,y],\ [y,x]\}$
⟨*proof*⟩

lemma *stepxy*: **assumes** $q \in \{x,y\}\ A \in \{\ [x,y],\ [y,x]\}$
shows $step\ A\ q\ a \in \{\ [x,y],\ [y,x]\}$
⟨*proof*⟩

12.2 Proof of Optimality

lemma *OPT2_is_lb*: $set\ \sigma \subseteq \{x,y\} \Longrightarrow x{\neq}y \Longrightarrow T_p\ [x,y]\ \sigma\ (OPT2\ \sigma\ [x,y]) \leq T_{p\text{-}}opt\ [x,y]\ \sigma$
⟨*proof*⟩

lemma *OPT2_is_ub*: $set\ qs \subseteq \{x,y\} \Longrightarrow x{\neq}y \Longrightarrow T_p\ [x,y]\ qs\ (OPT2\ qs\ [x,y]) \geq T_{p\text{-}}opt\ [x,y]\ qs$
⟨*proof*⟩

lemma *OPT2_is_opt*: $set\ qs \subseteq \{x,y\} \Longrightarrow x{\neq}y \Longrightarrow T_p\ [x,y]\ qs\ (OPT2\ qs\ [x,y]) = T_{p\text{-}}opt\ [x,y]\ qs$
⟨*proof*⟩

12.3 Performance on the four phase forms

lemma *OPT2_A*: **assumes** $x \neq y$ $qs \in lang$ (*seq* [*Plus* (*Atom x*) *One*, *Atom y*, *Atom y*])
shows T_p [*x*,*y*] *qs* (*OPT2 qs* [*x*,*y*]) = *1*
⟨*proof*⟩

lemma *OPT2_A'*: **assumes** $x \neq y$ $qs \in lang$ (*seq* [*Plus* (*Atom x*) *One*, *Atom y*, *Atom y*])
shows *real* (T_p [*x*,*y*] *qs* (*OPT2 qs* [*x*,*y*])) = *1*
⟨*proof*⟩

lemma *OPT2_B*: **assumes** $x \neq y$ $qs = u@v$ $u = [] \vee u = [x]$ $v \in lang$ (*seq*[*Times* (*Atom y*) (*Atom x*), *Star*(*Times* (*Atom y*) (*Atom x*)), *Atom y*, *Atom y*])
shows T_p [*x*,*y*] *qs* (*OPT2 qs* [*x*,*y*]) = (*length v div 2*)
⟨*proof*⟩

lemma *OPT2_B1*: **assumes** $x \neq y$ $qs \in lang$ (*seq*[*Atom y*, *Atom x*, *Star*(*Times* (*Atom y*) (*Atom x*)), *Atom y*, *Atom y*])
shows *real* (T_p [*x*,*y*] *qs* (*OPT2 qs* [*x*,*y*])) = *length qs / 2*
⟨*proof*⟩

lemma *OPT2_B2*: **assumes** $x \neq y$ $qs \in lang$ (*seq*[*Atom x*, *Atom y*, *Atom x*, *Star*(*Times* (*Atom y*) (*Atom x*)), *Atom y*, *Atom y*])
shows T_p [*x*,*y*] *qs* (*OPT2 qs* [*x*,*y*]) = ((*length qs* − *1*) / *2*)
⟨*proof*⟩

lemma *OPT2_C*: **assumes** $x \neq y$ $qs = u@v$ $u = [] \vee u = [x]$
and $v \in lang$ (*seq*[*Atom y*, *Atom x*, *Star*(*Times* (*Atom y*) (*Atom x*)), *Atom x*])
shows T_p [*x*,*y*] *qs* (*OPT2 qs* [*x*,*y*]) = (*length v div 2*)
⟨*proof*⟩

lemma *OPT2_C1*: **assumes** $x \neq y$ $qs \in lang$ (*seq*[*Atom y*, *Atom x*, *Star*(*Times* (*Atom y*) (*Atom x*)), *Atom x*])
shows *real* (T_p [*x*,*y*] *qs* (*OPT2 qs* [*x*,*y*])) = (*length qs* − *1*) / *2*
⟨*proof*⟩

lemma *OPT2_C2*: **assumes** $x \neq y$ $qs \in lang$ (*seq*[*Atom x*, *Atom y*, *Atom x*, *Star*(*Times* (*Atom y*) (*Atom x*)), *Atom x*])
shows T_p [*x*,*y*] *qs* (*OPT2 qs* [*x*,*y*]) = ((*length qs* − *2*) / *2*)
⟨*proof*⟩

lemma *OPT2_ub*: *set qs* $\subseteq \{x,y\} \Longrightarrow T_p\ [x,y]\ qs\ (OPT2\ qs\ [x,y]) \leq length\ qs$
⟨*proof*⟩

lemma *OPT2_padded*: $R \in \{[x,y],[y,x]\} \Longrightarrow set\ qs \subseteq \{x,y\}$
$\Longrightarrow\ T_p\ R\ (qs@[x,x])\ (OPT2\ (qs@[x,x])\ R)$
$\leq T_p\ R\ (qs@[x])\ (OPT2\ (qs@[x])\ R) + 1$
⟨*proof*⟩

lemma *OPT2_split11*:
assumes *xy*: $x \neq y$
shows $R \in \{[x,y],[y,x]\} \Longrightarrow set\ xs \subseteq \{x,y\} \Longrightarrow set\ ys \subseteq \{x,y\} \Longrightarrow OPT2\ (xs@[x,x]@ys)\ R = OPT2\ (xs@[x,x])\ R\ @\ OPT2\ ys\ [x,y]$
⟨*proof*⟩

12.4 The function steps

lemma *steps_append*: *length qs* = *length as* $\Longrightarrow$ *steps s* $(qs@[q])\ (as@[a])$ = *step* $(steps\ s\ qs\ as)\ q\ a$
⟨*proof*⟩

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 *Lxx* **where**
Lxx (*x*::*nat*) *y* = *lang* (*L_lasthasxx x y*)

lemma *Lxx_not_nullable*: [] $\notin$ *Lxx x y*
⟨*proof*⟩

lemma *Lxx_ends_in_two_equal*: $xs \in Lxx\ x\ y \Longrightarrow \exists\, pref\ e.\ xs = pref\ @\ [e,e]$
⟨*proof*⟩

lemma $Lxx\ x\ y = Lxx\ y\ x$ ⟨*proof*⟩

definition $hideit\ x\ y = (Plus\ rexp.One\ (nodouble\ x\ y))$

lemma *Lxx_othercase*: $set\ qs \subseteq \{x,y\} \Longrightarrow \neg\ (\exists\, xs\ ys.\ qs = xs\ @\ ys \wedge xs \in Lxx\ x\ y) \Longrightarrow qs \in lang\ (hideit\ x\ y)$
⟨*proof*⟩

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*: $qs \neq [] \Longrightarrow set\ qs \subseteq \{x,y\} \Longrightarrow pad\ qs\ x\ y = qs\ @\ [last\ qs]$
⟨*proof*⟩

lemma *nodouble_padded*: $qs \neq [] \Longrightarrow qs \in lang\ (nodouble\ x\ y) \Longrightarrow pad\ qs\ x\ y \in Lxx\ x\ y$
⟨*proof*⟩

thm *UnE*
lemma $c \in A \cup B \Longrightarrow P$
⟨*proof*⟩

lemma *LxxE*: $qs \in Lxx\ x\ y$
$\Longrightarrow (qs \in lang\ (seq\ [Atom\ x,\ Atom\ x]) \Longrightarrow P\ x\ y\ qs)$
$\Longrightarrow (qs \in lang\ (seq\ [Plus\ (Atom\ x)\ rexp.One,\ Atom\ y,\ Atom\ x,\ Star\ (Times\ (Atom\ y)\ (Atom\ x)),\ Atom\ y,\ Atom\ y]) \Longrightarrow P\ x\ y\ qs)$
$\Longrightarrow (qs \in lang\ (seq\ [Plus\ (Atom\ x)\ rexp.One,\ Atom\ y,\ Atom\ x,\ Star\ (Times\ (Atom\ y)\ (Atom\ x)),\ Atom\ x]) \Longrightarrow P\ x\ y\ qs)$
$\Longrightarrow (qs \in lang\ (seq\ [Plus\ (Atom\ x)\ rexp.One,\ Atom\ y,\ Atom\ y]) \Longrightarrow P\ x\ y\ qs)$
$\Longrightarrow P\ x\ y\ qs$
⟨*proof*⟩

thm *UnE LxxE*

lemma $qs \in Lxx\ x\ y \Longrightarrow P$
⟨*proof*⟩

lemma *LxxI*: $(qs \in lang\ (seq\ [Atom\ x,\ Atom\ x]) \Longrightarrow P\ x\ y\ qs)$
$\Longrightarrow (qs \in lang\ (seq\ [Plus\ (Atom\ x)\ rexp.One,\ Atom\ y,\ Atom\ x,\ Star\ (Times\ (Atom\ y)\ (Atom\ x)),\ Atom\ y,\ Atom\ y]) \Longrightarrow P\ x\ y\ qs)$
$\Longrightarrow (qs \in lang\ (seq\ [Plus\ (Atom\ x)\ rexp.One,\ Atom\ y,\ Atom\ x,\ Star\ (Times\ (Atom\ y)\ (Atom\ x)),\ Atom\ x]) \Longrightarrow P\ x\ y\ qs)$
$\Longrightarrow (qs \in lang\ (seq\ [Plus\ (Atom\ x)\ rexp.One,\ Atom\ y,\ Atom\ y]) \Longrightarrow P\ x\ y\ qs)$
$\Longrightarrow (qs \in Lxx\ x\ y \Longrightarrow P\ x\ y\ qs)$
⟨*proof*⟩

lemma *Lxx1*: $xs \in Lxx\ x\ y \Longrightarrow length\ xs \geq 2$
⟨*proof*⟩

13.2 OPT2 Splitting

lemma *ayay*: $length\ qs = length\ as \Longrightarrow T_p\ s\ (qs@[q])\ (as@[a]) = T_p\ s\ qs\ as + t_p\ (steps\ s\ qs\ as)\ q\ a$
⟨*proof*⟩

lemma *tlofOPT2*: $Q \in \{x,y\} \Longrightarrow set\ QS \subseteq \{x,y\} \Longrightarrow R \in \{[x,\ y],\ [y,\ x]\} \Longrightarrow tl\ (OPT2\ ((Q\ \#\ QS)\ @\ [x,\ x])\ R) =$
$OPT2\ (QS\ @\ [x,\ x])\ (step\ R\ Q\ (hd\ (OPT2\ ((Q\ \#\ QS)\ @\ [x,\ x])\ R)))$
⟨*proof*⟩

lemma T_p*_split*: $length\ qs1 = length\ as1 \Longrightarrow T_p\ s\ (qs1@qs2)\ (as1@as2) = T_p\ s\ qs1\ as1 + T_p\ (steps\ s\ qs1\ as1)\ qs2\ as2$
⟨*proof*⟩

lemma T_p*_spliting*: $x \neq y \Longrightarrow set\ xs \subseteq \{x,y\} \Longrightarrow set\ ys \subseteq \{x,y\} \Longrightarrow$
$R \in \{[x,y],[y,x]\} \Longrightarrow$
$T_p\ R\ (xs@[x,x])\ (OPT2\ (xs@[x,x])\ R) + T_p\ [x,y]\ ys\ (OPT2\ ys\ [x,y]) = T_p\ R\ (xs@[x,x]@ys)\ (OPT2\ (xs@[x,x]@ys)\ R)$
⟨*proof*⟩

lemma *OPTauseinander*: $x \neq y \Longrightarrow set\ xs \subseteq \{x,y\} \Longrightarrow set\ ys \subseteq \{x,y\} \Longrightarrow$
$LTS \in \{[x,y],[y,x]\} \Longrightarrow hd\ LTS = last\ xs \Longrightarrow$
$xs = (pref\ @\ [hd\ LTS,\ hd\ LTS]) \Longrightarrow$

$T_p\ [x,y]\ xs\ (OPT2\ xs\ [x,y])\ +\ T_p\ LTS\ ys\ (OPT2\ ys\ LTS)$
$=\ T_p\ [x,y]\ (xs@ys)\ (OPT2\ (xs@ys)\ [x,y])$
⟨*proof*⟩

13.3 Phase Partitioning lemma

theorem *Phase_partitioning_general*:
fixes $P :: (nat\ state * 'is)\ pmf \Rightarrow nat \Rightarrow nat\ list \Rightarrow bool$
and $A :: (nat\ state, 'is, nat, answer)\ alg\_on\_rand$
assumes *xny*: $(x0::nat) \neq y0$
and *cpos*: $(c::real) \geq 0$
and *static*: $set\ \sigma \subseteq \{x0,y0\}$
and *initial*: $P\ (map\_pmf\ (\%is.\ ([x0,y0],is))\ (fst\ A\ [x0,y0]))\ x0\ [x0,y0]$
and *D*: $\bigwedge a\ b\ \sigma\ s.\ \sigma \in Lxx\ a\ b \Longrightarrow a \neq b \Longrightarrow \{a,b\}=\{x0,y0\} \Longrightarrow P\ s\ a\ [x0,y0] \Longrightarrow set\ \sigma \subseteq \{a,b\} \Longrightarrow T\_on\_rand'\ A\ s\ \sigma \leq c * T_p\ [a,b]\ \sigma\ (OPT2\ \sigma\ [a,b]) \wedge P\ (config'\_rand\ A\ s\ \sigma)\ (last\ \sigma)\ [x0,y0]$
shows $T_{p}\_on\_rand\ A\ [x0,y0]\ \sigma \leq c * T_{p}\_opt\ [x0,y0]\ \sigma + c$
⟨*proof*⟩

term $A::(nat,'is)\ alg\_on$

theorem *Phase_partitioning_general_det*:
fixes $P :: (nat\ state * 'is) \Rightarrow nat \Rightarrow nat\ list \Rightarrow bool$
and $A :: (nat,'is)\ alg\_on$
assumes *xny*: $(x0::nat) \neq y0$
and *cpos*: $(c::real) \geq 0$
and *static*: $set\ \sigma \subseteq \{x0,y0\}$
and *initial*: $P\ ([x0,y0],(fst\ A\ [x0,y0]))\ x0\ [x0,y0]$
and *D*: $\bigwedge a\ b\ \sigma\ s.\ \sigma \in Lxx\ a\ b \Longrightarrow a \neq b \Longrightarrow \{a,b\}=\{x0,y0\} \Longrightarrow P\ s\ a\ [x0,y0] \Longrightarrow set\ \sigma \subseteq \{a,b\} \Longrightarrow T\_on'\ A\ s\ \sigma \leq c * T_p\ [a,b]\ \sigma\ (OPT2\ \sigma\ [a,b]) \wedge P\ (config'\ A\ s\ \sigma)\ (last\ \sigma)\ [x0,y0]$
shows $T_{p}\_on\ A\ [x0,y0]\ \sigma \leq c * T_{p}\_opt\ [x0,y0]\ \sigma + c$
⟨*proof*⟩

end

14 List factoring technique

theory *List_Factoring*
imports

Partial_Cost_Model
MTF2_Effects
begin

hide_const *config compet*

14.1 Helper functions

14.1.1 Helper lemmas

lemma *befaf*: **assumes** *q∈set s distinct s*
shows *before q s ∪ {q} ∪ after q s = set s*
⟨*proof*⟩

lemma *index_sum*: **assumes** *distinct s q∈set s*
shows *index s q = (∑ e∈set s. if e < q in s then 1 else 0)*
⟨*proof*⟩

14.1.2 ALG

fun *ALG* :: *'a ⇒ 'a list ⇒ nat ⇒ ('a list * 'is) ⇒ nat* **where**
ALG x qs i s = (if x < (qs!i) in fst s then 1::nat else 0)

lemma *t_p_sumofALG*: *distinct (fst s) ⟹ snd a = [] ⟹ (qs!i)∈set (fst s)*

⟹ t_p (fst s) (qs!i) a = (∑ e∈set (fst s). ALG e qs i s)
⟨*proof*⟩

lemma *t_p_sumofALGreal*: **assumes** *distinct (fst s) snd a = [] qs!i ∈ set(fst s)*
shows *real(t_p (fst s) (qs!i) a) = (∑ e∈set (fst s). real(ALG e qs i s))*
⟨*proof*⟩

14.1.3 The function steps'

fun *steps'* **where**
steps' s _ _ 0 = s
| *steps' s [] [] (Suc n) = s*
| *steps' s (q#qs) (a#as) (Suc n) = steps' (step s q a) qs as n*

lemma *steps'_steps*: *length as = length qs ⟹ steps' s as qs (length as) = steps s as qs*
⟨*proof*⟩

lemma *steps'_length*: *length qs = length as* $\Longrightarrow$ *n* $\leq$ *length as*
$\Longrightarrow$ *length* (*steps' s qs as n*) = *length s*
⟨*proof*⟩

lemma *steps'_set*: *length qs = length as* $\Longrightarrow$ *n* $\leq$ *length as*
$\Longrightarrow$ *set* (*steps' s qs as n*) = *set s*
⟨*proof*⟩

lemma *steps'_distinct2*: *length qs = length as* $\Longrightarrow$ *n* $\leq$ *length as*
$\Longrightarrow$ *distinct s* $\Longrightarrow$ *distinct* (*steps' s qs as n*)
⟨*proof*⟩

lemma *steps'_distinct*: *length qs = length as* $\Longrightarrow$ *length as = n*
$\Longrightarrow$ *distinct* (*steps' s qs as n*) = *distinct s*
⟨*proof*⟩

lemma *steps'_dist_perm*: *length qs = length as* $\Longrightarrow$ *length as = n*
$\Longrightarrow$ *dist_perm s s* $\Longrightarrow$ *dist_perm* (*steps' s qs as n*) (*steps' s qs as n*)
⟨*proof*⟩

lemma *steps'_rests*: *length qs = length as* $\Longrightarrow$ *n* $\leq$ *length as* $\Longrightarrow$ *steps' s qs as n = steps' s* (*qs@r1*) (*as@r2*) *n*
⟨*proof*⟩

lemma *steps'_append*: *length qs = length as* $\Longrightarrow$ *length qs = n* $\Longrightarrow$ *steps' s* (*qs@*[*q*]) (*as@*[*a*]) (*Suc n*) = *step* (*steps' s qs as n*) *q a*
⟨*proof*⟩

14.1.4 *ALG'_det*

definition *ALG'_det Strat qs init i x = ALG x qs i* (*swaps* (*snd* (*Strat*!*i*)) (*steps' init qs Strat i*),())

lemma *ALG'_det_append*: *n < length Strat* $\Longrightarrow$ *n < length qs* $\Longrightarrow$ *ALG'_det Strat* (*qs@a*) *init n x*
= *ALG'_det Strat qs init n x*
⟨*proof*⟩

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*
⟨*proof*⟩

14.1.6 *ALGxy_det*

definition *ALGxy_det* **where**
ALGxy_det A qs init x y = (∑ *i*∈{..<*length qs*}. (*if* (*qs!i* ∈ {*y*,*x*}) *then ALG′_det A qs init i y + ALG′_det A qs init i x*
else 0::*nat*))

lemma *ALGxy_det_alternativ*: *ALGxy_det A qs init x y*
= (∑ *i*∈{*i*. *i*<*length qs* ∧ (*qs!i* ∈ {*y*,*x*})}. *ALG′_det A qs init i y + ALG′_det A qs init i x*)
⟨*proof*⟩

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*⟩
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*⟩

lemma *ALGxy_wholerange*: *ALGxy A qs init x y*
= (∑ *i*<(*length qs*). (*if qs* ! *i* ∈ {*y*, *x*}
then ALG′ A qs init i y + ALG′ A qs init i x
else 0))
⟨*proof*⟩

14.2 Transformation to Blocking Cost

lemma *umformung*:
fixes *A* :: ((′*a*::*linorder*) *list*,′*is*,′*a*,(*nat* ∗ *nat list*)) *alg_on_rand*
assumes *no_paid*: ⋀*is s q*. ∀((*free*,*paid*),_) ∈ (*snd A* (*s*,*is*) *q*). *paid*=[]
assumes *inlist*: *set qs* ⊆ *set init*

assumes *dist*: *distinct init*
assumes $\bigwedge x.\ x < length\ qs \Longrightarrow finite\ (set\_pmf\ (config''\ A\ qs\ init\ x))$
shows $T_p\_on\_rand\ A\ init\ qs =$
$(\sum (x,y) \in \{(x,y).\ x \in set\ init \wedge y \in set\ init \wedge x < y\}.\ ALGxy\ A\ qs\ init\ x\ y)$
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *before_in_index*:
fixes *l*
assumes *set l* = {*x*,*y*} **and** *length l* = *2* **and** $x \neq y$
shows (*x* < *y in l*) = (*index l x* = *0*)
⟨*proof*⟩

14.3 The pairwise property

definition *pairwise* **where**
pairwise A = $(\forall\, init.\ distinct\ init \longrightarrow (\forall\, qs \in \{xs.\ set\ xs \subseteq set\ init\}.\ \forall\, (x::('a::linorder),y) \in \{(x,y).\ x \in set\ init \wedge y \in set\ init \wedge x < y\}.\ T_p\_on\_rand\ A\ (Lxy\ init\ \{x,y\})\ (Lxy\ qs\ \{x,y\}) = ALGxy\ A\ qs\ init\ x\ y))$

definition *Pbefore_in x y A qs init* = *map_pmf* ($\lambda p.\ x < y\ in\ fst\ p$) (*config_rand A init qs*)

lemma *T_on_n_no_paid*:
assumes
nopaid: $\bigwedge s\ n.\ map\_pmf\ (\lambda x.\ snd\ (fst\ x))\ (snd\ A\ s\ n) = return\_pmf\ []$
shows $T\_on\_rand\_n\ A\ init\ qs\ i = E\ (config''\ A\ qs\ init\ i \gg\!= (\lambda p.\ return\_pmf\ (real(index\ (fst\ p)\ (qs\ !\ i)))))$

⟨*proof*⟩

lemma *pairwise_property_lemma*:
assumes
relativeorder: (⋀*init qs*. *distinct init* ⟹ *qs* ∈ {*xs*. *set xs* ⊆ *set init*}
⟹ (⋀*x y*. (*x*,*y*)∈ {(*x*,*y*). *x* ∈ *set init* ∧ *y*∈*set init* ∧ *x*≠*y*}
⟹ *x* ≠ *y*
⟹ *Pbefore_in x y A qs init* = *Pbefore_in x y A* (*Lxy qs* {*x*,*y*})
(*Lxy init* {*x*,*y*})
))
and *nopaid*: ⋀*xa r*. ∀ *z*∈ *set_pmf*(*snd A xa r*). *snd*(*fst z*) = []
shows *pairwise A*
⟨*proof*⟩

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*⟩

14.4 List Factoring for OPT

fun *ALG_P* :: *nat list* ⇒ *'a* ⇒ *'a* ⇒ *'a list* ⇒ *nat* **where**
ALG_P [] *x y xs* = (*0*::*nat*)
| *ALG_P* (*s*#*ss*) *x y xs* = (*if Suc s* < *length* (*swaps ss xs*)
then (*if* ((*swaps ss xs*)!*s*=*x* ∧ (*swaps ss xs*)!(*Suc s*)=*y*)
∨ ((*swaps ss xs*)!*s*=*y* ∧ (*swaps ss xs*)!(*Suc s*)=*x*)
then 1
else 0)
else 0) + *ALG_P ss x y xs*

lemma *ALG_P_erwischt_alle*:
assumes *dinit*: *distinct init*
shows
∀ *l*< *length sws*. *Suc* (*sws*!*l*) < *length init* ⟹ *length sws*
= (∑ (*x*,*y*)∈{(*x*,*y*). *x* ∈ *set* (*init*::(*'a*::*linorder*) *list*) ∧ *y*∈*set init* ∧
x<*y*}. *ALG_P sws x y init*)

⟨*proof*⟩

lemma t_p*_sumofALGALGP*:
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*⟩

definition *ALG_P' Strat qs init i x y* = *ALG_P* (*snd* (*Strat*!*i*)) *x y* (*steps'* *init qs Strat i*)

lemma *ALG_P'_rest*: *n* < *length qs* ⟹ *n* < *length Strat* ⟹
ALG_P' Strat (*take n qs* @ [*qs* ! *n*]) *init n x y* =
ALG_P' (*take n Strat* @ [*Strat* ! *n*]) (*take n qs* @ [*qs* ! *n*]) *init n x y*
⟨*proof*⟩

lemma *ALG_P'_rest2*: *n* < *length qs* ⟹ *n* < *length Strat* ⟹
ALG_P' Strat qs init n x y =
ALG_P' (*Strat*@*r1*) (*qs*@*r2*) *init n x y*
⟨*proof*⟩

definition *ALG_Pxy* **where**
ALG_Pxy Strat qs init x y = (∑ *i*<*length qs*. *ALG_P' Strat qs init i x y*)

lemma *wegdamit*: *length A* < *length Strat* ⟹ *b* ∉ {*x*,*y*} ⟹ *ALGxy_det Strat* (*A* @ [*b*]) *init x y*
= *ALGxy_det Strat A init x y*
⟨*proof*⟩

lemma *ALG_P_split*: *length qs* < *length Strat* ⟹ *ALG_Pxy Strat* (*qs*@[*q*]) *init x y* = *ALG_Pxy Strat qs init x y*

$+ \;\; ALG\_P'\ Strat\ (qs@[q])\ init\ (length\ qs)\ x\ y$
⟨*proof*⟩

lemma *swap0in2*: **assumes** *set l* = {*x*,*y*} *x*≠*y length l* = 2 *dist_perm l l*
shows
x < *y in* (*swap* 0) *l* = (~ *x* < *y in l*)
⟨*proof*⟩

lemma *before_in_swap2*:
dist_perm xs ys ⟹ *Suc n* < *size xs* ⟹ *x*≠*y* ⟹
x < *y in* (*swap n xs*) ⟷
(~ *x* < *y in xs* ∧ (*y* = *xs*!*n* ∧ *x* = *xs*!*Suc n*)
∨ *x* < *y in xs* ∧ ~(*y* = *xs*!*Suc n* ∧ *x* = *xs*!*n*))
⟨*proof*⟩

lemma *projected_paid_same_effect*:
assumes
d: *dist_perm s1 s1*
and *ee*: *x*≠*y*
and *f*: *set s2* = {*x*, *y*}
and *g*: *length s2* = 2
and *h*: *dist_perm s2 s2*
shows *x* < *y in s1* = *x* < *y in s2* ⟹
x < *y in swaps acs s1* = *x* < *y in* (*swap* 0 ^^ *ALG_P acs x y s1*) *s2*
⟨*proof*⟩

lemma *steps_steps'*:
length qs = *length as* ⟹ *steps s qs as* = *steps' s qs as* (*length as*)
⟨*proof*⟩

lemma *T1_7'*: T_p *init qs Strat* = T_{p}*_opt init qs* ⟹ *length Strat* = *length qs*
⟹ *n*≤*length qs* ⟹
x≠(*y*::('*a*::*linorder*)) ⟹
x∈ *set init* ⟹ *y* ∈ *set init* ⟹ *distinct init* ⟹
set qs ⊆ *set init* ⟹
(∃ *Strat2 sws*.
T_{p}*_opt* (*Lxy init* (*x*,*y*)) (*Lxy* (*take n qs*) (*x*,*y*)) ≤ T_p (*Lxy init* (*x*,*y*)) (*Lxy* (*take n qs*) (*x*,*y*)) *Strat2* ∧ *length Strat2* = *length* (*Lxy* (*take*

n qs) $\{x,y\}$)
$\wedge$ $(x < y$ *in* $(steps'$ *init* $(take$ n $qs)$ $(take$ n *Strat*$)$ $n))$
$= (x < y$ *in* $(swaps$ *sws* $(steps'$ $(Lxy$ *init* $\{x,y\})$ $(Lxy$ $(take$ n $qs)$ $\{x,y\})$ *Strat2* $(length$ *Strat2*$))))$
$\wedge$ T_p $(Lxy$ *init* $\{x,y\})$ $(Lxy$ $(take$ n $qs)$ $\{x,y\})$ *Strat2* $+$ *length sws* $=$
ALGxy_det Strat $(take$ n $qs)$ *init* x y $+$ *ALG_Pxy Strat* $(take$ n $qs)$ *init* x $y)$
⟨*proof*⟩

lemma *T1_7*:
assumes T_p *init qs Strat* $= T_p$*_opt init qs length Strat* $=$ *length qs*
$x \neq (y::('a::linorder))$ $x \in$ *set init* $y \in$ *set init distinct init*
set qs $\subseteq$ *set init*
shows T_p*_opt* $(Lxy$ *init* $\{x,y\})$ $(Lxy$ *qs* $\{x,y\})$
$\leq$ *ALGxy_det Strat qs init x y* $+$ *ALG_Pxy Strat qs init x y*
⟨*proof*⟩

lemma *T_snoc*: *length rs* $=$ *length as*
$\Longrightarrow$ *T init* $(rs@[r])$ $(as@[a])$
$=$ *T init rs as* $+$ t_p $(steps'$ *init rs as* $(length$ *rs*$))$ *r a*
⟨*proof*⟩

lemma *steps'_snoc*: *length rs* $=$ *length as* $\Longrightarrow$ $n = (length$ *as*$)$
$\Longrightarrow$ *steps'* *init* $(rs@[r])$ $(as@[a])$ $(Suc$ $n) =$ *step* $(steps'$ *init rs as* $n)$ *r a*
⟨*proof*⟩

lemma *steps'_take*:
assumes $n<$*length qs length qs* $=$ *length Strat*
shows *steps'* *init* $(take$ n *qs*$)$ $(take$ n *Strat*$)$ n
$=$ *steps'* *init qs Strat n*
⟨*proof*⟩

lemma *Tp_darstellung*: *length qs* $=$ *length Strat*
$\Longrightarrow$ T_p *init qs Strat* $=$
$(\sum i \in \{..<length\ qs\}.\ t_p$ $(steps'$ *init qs Strat i*$)$ $(qs!i)$ $(Strat!i))$
⟨*proof*⟩

lemma *umformung_OPT'*:
assumes *inlist*: *set qs* $\subseteq$ *set init*

assumes *dist*: *distinct init*
assumes *qsStrat*: *length qs* = *length Strat*
assumes *noStupid*: $\bigwedge x\ l.\ x<$*length Strat* $\Longrightarrow l<$ *length* (*snd* (*Strat* ! *x*)) $\Longrightarrow$ *Suc* ((*snd* (*Strat* ! *x*))!*l*) $<$ *length init*
shows T_p *init qs Strat* =
$(\sum (x,y)\in\{(x,y::('a::linorder)).\ x \in set\ init \wedge y\in set\ init \wedge x<y\}.$
ALGxy_det Strat qs init x y + *ALG_Pxy Strat qs init x y*)
⟨*proof*⟩

lemma *nn_contains_Inf*:
fixes *S* :: *nat set*
assumes *nn*: $S \neq \{\}$
shows *Inf S* $\in S$
⟨*proof*⟩

lemma *steps_length*: *length qs* = *length as* $\Longrightarrow$ *length* (*steps s qs as*) = *length s*
⟨*proof*⟩

lemma *OPT_noStupid*:
fixes *Strat*
assumes [*simp*]: *length Strat* = *length qs*
assumes *opt*: T_p *init qs Strat* = T_p*_opt init qs*
assumes *init_nempty*: *init*$\neq$[]
shows $\bigwedge x\ l.\ x <$ *length Strat* $\Longrightarrow$
l $<$ *length* (*snd* (*Strat* ! *x*)) $\Longrightarrow$
Suc ((*snd* (*Strat* ! *x*))!*l*) $<$ *length init*
⟨*proof*⟩

lemma *umformung_OPT*:
assumes *inlist*: *set qs* $\subseteq$ *set init*
assumes *dist*: *distinct init*
assumes *a*: T_p*_opt init qs* = T_p *init qs Strat*
assumes *b*: *length qs* = *length Strat*
assumes *c*: *init*$\neq$[]
shows T_p*_opt init qs* =
$(\sum (x,y)\in\{(x,y::('a::linorder)).\ x \in set\ init \wedge y\in set\ init \wedge x<y\}.$

ALGxy_det Strat qs init x y + ALG_Pxy Strat qs init x y)
⟨*proof*⟩

corollary *OPT_zerlegen*:
assumes
dist: *distinct init*
and *c*: *init*≠[]
and *setqsinit*: *set qs* ⊆ *set init*
shows (∑ (*x*,*y*)∈{(*x*,*y*::('*a*::*linorder*)). *x* ∈ *set init* ∧ *y*∈*set init* ∧ *x*<*y*}. (*T*$_p$*_opt* (*Lxy init* {*x*,*y*}) (*Lxy qs* {*x*,*y*})))
≤ *T*$_p$*_opt init qs*
⟨*proof*⟩

14.5 Factoring Lemma

lemma *cardofpairs*: *S* ≠ [] ⟹ *sorted S* ⟹ *distinct S* ⟹ *card* {(*x*,*y*). *x* ∈ *set S* ∧ *y*∈*set S* ∧ *x*<*y*} = ((*length S*)∗(*length S*−*1*)) / *2*
⟨*proof*⟩

lemma *factoringlemma_withconstant*:
fixes *A*
and *b*::*real*
and *c*::*real*
assumes *c*: *c* ≥ *1*
assumes *dist*: ∀ *e*∈*S0*. *distinct e*
assumes *notempty*: ∀ *e*∈*S0*. *length e* > *0*

assumes *pw*: *pairwise A*

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*$_p$*_on_rand A* (*Lxy s0* {*x*,*y*}) (*Lxy qs* {*x*,*y*}) ≤ *c* ∗ (*T*$_p$*_opt* (*Lxy s0* {*x*,*y*}) (*Lxy qs* {*x*,*y*})) + *b*
assumes *nopaid*: ⋀*is s q*. ∀ ((*free*,*paid*),_) ∈ (*snd A* (*s*, *is*) *q*). *paid*=[]
assumes *4*: ⋀*init qs*. *distinct init* ⟹ *set qs* ⊆ *set init* ⟹ (⋀*x*. *x*<*length qs* ⟹ *finite* (*set_pmf* (*config*'' *A qs init x*)))

shows ∀ *s0*∈*S0*. ∃ *b*≥*0*. ∀ *qs*∈{*x*. *set x* ⊆ *set s0*}.
T$_p$*_on_rand A s0 qs* ≤ *c* ∗ *real* (*T*$_p$*_opt s0 qs*) + *b*
⟨*proof*⟩

lemma *factoringlemma_withconstant'*:

fixes A
and b::*real*
and c::*real*
assumes c: $c \geq 1$
assumes *dist*: $\forall\, e{\in}S0.\ distinct\ e$
assumes *notempty*: $\forall\, e{\in}S0.\ length\ e > 0$

assumes *pw*: *pairwise A*

assumes *on2*: $\forall\, s0{\in}S0.\ \exists\, b{\geq}0.\ \forall\, qs{\in}\{x.\ set\ x \subseteq set\ s0\}.\ \forall\,(x,y){\in}\{(x,y).\ x \in set\ s0 \wedge y{\in}set\ s0 \wedge x{<}y\}.\ T_p\_on\_rand\ A\ (Lxy\ s0\ \{x,y\})\ (Lxy\ qs\ \{x,y\}) \leq c * (T_p\_opt\ (Lxy\ s0\ \{x,y\})\ (Lxy\ qs\ \{x,y\}))) + b$
assumes *nopaid*: $\bigwedge is\ s\ q.\ \forall\,((free,paid),\_) \in (snd\ A\ (s,\ is)\ q).\ paid{=}[]$
assumes *4*: $\bigwedge init\ qs.\ distinct\ init \Longrightarrow set\ qs \subseteq set\ init \Longrightarrow (\bigwedge x.\ x{<}length\ qs \Longrightarrow finite\ (set\_pmf\ (config''\ A\ qs\ init\ x)))$

shows *compet_rand A c S0*
⟨*proof*⟩

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))
```

definition *rTS* :: *nat list* ⇒ (*nat*,*nat list*) *alg_on* **where** *rTS h* = ((λ*s*. *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*))) = []
⟨*proof*⟩

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*)
⟨*proof*⟩

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*
⟨*proof*⟩

15.2 Behaviour of TS on lists of length 2

lemma
fixes *hs x y*
assumes $x \neq y$
shows *oneTS_step* : *TS_step_d* ([*x*, *y*], *x*#*y*#*hs*) *y* = ((*1*, []), *y* # *x* # *y* # *hs*)
and *oneTS_stepyyy*: *TS_step_d* ([*x*, *y*], *y*#*x*#*hs*) *y* = ((*Suc 0*, []), *y*#*y*#*x*#*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*], []) *y* = ((*0*, []), [*y*])
and *oneTS_stepxy*: *TS_step_d* ([*x*, *y*], [*x*]) *y* = ((*0*, []), [*y*, *x*])
and *oneTS_stepyy*: *TS_step_d* ([*x*, *y*], [*y*]) *y* = ((*Suc 0*, []), [*y*, *y*])
and *oneTS_stepyx*: *TS_step_d* ([*x*, *y*], *hs*) *x* = ((*0*, []), *x* # *hs*)
⟨*proof*⟩

lemmas *oneTS_steps* = *oneTS_stepx oneTS_stepxy oneTS_stepyx oneTS_stepy oneTS_stepyy oneTS_stepyyy oneTS_step*

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*: $a \neq b \Longrightarrow \{a,b\}=\{x,y\} \Longrightarrow z \in \{x,y\} \Longrightarrow$ *TS_inv c z* [*a*,*b*] = *TS_inv c z* [*x*,*y*]
⟨*proof*⟩

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*),[]))
⟨*proof*⟩

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*),[])
⟨*proof*⟩

15.3.1 (yx)*?

lemma *TS_yx'*: **assumes** $x \neq y$ $qs \in lang\ (Star(Times\ (Atom\ y)\ (Atom\ x)))$
$\exists hs.\ h=[x,y]@hs$
shows $T\_on'\ (rTS\ h0)\ ([x,y],h)\ (qs@r) = length\ qs + T\_on'\ (rTS\ h0)\ ([x,y],((rev\ qs)\ @h))\ \ r$
$\wedge\ (\exists hs.\ ((rev\ qs)\ @h) = [x,\ y]\ @\ hs)$
$\wedge\ config'\ (rTS\ h0)\ ([x,\ y],h)\ qs = ([x,y],rev\ qs\ @\ h)$
⟨*proof*⟩

15.3.2 ?x

lemma *TS_x'*: $T\_on'\ (rTS\ h0)\ ([x,y],h)\ [x] = 0 \wedge config'\ (rTS\ h0)\ ([x,y],h)\ [x] = ([x,y],\ rev\ [x]\ @\ h)$
⟨*proof*⟩

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*⟩

15.3.4 yx(yx)*?

lemma *TS_yxyx'*: **assumes** [*simp*]: $x \neq y$ **and** $qs \in lang\ (seq[Times\ (Atom\ y)\ (Atom\ x),\ Star(Times\ (Atom\ y)\ (Atom\ x))])$
$(\exists hs.\ h=[x,x]@hs) \vee index\ h\ y = length\ h$
shows $T\_on'\ (rTS\ h0)\ ([x,y],h)\ (qs@r) = length\ qs - 1 + T\_on'\ (rTS\ h0)\ ([x,y],rev\ qs\ @\ h)\ r$
$\wedge\ (\exists hs.\ (rev\ qs\ @\ h) = [x,\ y]\ @\ hs)$
$\wedge\ config'\ (rTS\ h0)\ ([x,\ y],h)\ qs = ([x,y],\ rev\ qs\ @\ h)$
⟨*proof*⟩

lemma *TS_xr'*: **assumes** $x \neq y$ $qs \in lang\ (Plus\ (Atom\ x)\ One)$
$h = [] \vee (\exists hs.\ h = [x,\ x]\ @\ hs)$
shows $T\_on'\ (rTS\ h0)\ ([x,y],h)\ (qs@r) = T\_on'\ (rTS\ h0)\ ([x,y],rev\ qs@h)\ r$
$((\exists hs.\ (rev\ qs\ @\ h) = [x,\ x]\ @\ hs) \vee (rev\ qs\ @\ h) = [x] \vee (rev\ qs\ @\ h)=[])$

config′ (rTS h0) ([x,y],h) (qs@r) = config′ (rTS h0) ([x,y],rev qs @ h) r
⟨*proof*⟩

15.3.5 (x+1)yx(yx)*yy

lemma *ts_b′*: **assumes** $x \neq y$
$v \in lang\ (seq[Times\ (Atom\ y)\ (Atom\ x),\ Star\ (Times\ (Atom\ y)\ (Atom\ x)),\ Atom\ y,\ Atom\ y])$
$(\exists hs.\ h = [x,\ x]\ @\ hs) \vee h = [x] \vee h = []$
shows $T\_on′\ (rTS\ h0)\ ([x,\ y],\ h)\ v = (length\ v - 2)$
$\wedge\ (\exists hs.\ (rev\ v\ @\ h) = [y,y]@hs) \wedge config′\ (rTS\ h0)\ ([x,y],\ h)\ v = ([y,x],\ rev\ v\ @\ h)$
⟨*proof*⟩

lemma *TS_b′1*: **assumes** $x \neq y\ h = [] \vee (\exists hs.\ h = [x,\ x]\ @\ hs)$
$qs \in lang\ (seq\ [Atom\ y,\ Atom\ x,\ Star\ (Times\ (Atom\ y)\ (Atom\ x)),\ Atom\ y,\ Atom\ y])$
shows $T\_on′\ (rTS\ h0)\ ([x,\ y],\ h)\ qs = (length\ qs - 2)$
$\wedge\ TS\_inv′\ (config′\ (rTS\ h0)\ ([x,\ y],\ h)\ qs)\ (last\ qs)\ [x,y]$
⟨*proof*⟩

lemma *TS_b1′′*: **assumes**
$x \neq y\ \{x,\ y\} = \{x0,\ y0\}\ TS\_inv\ s\ x\ [x0,\ y0]$
$set\ qs \subseteq \{x,\ y\}$
$qs \in lang\ (seq\ [Atom\ y,\ Atom\ x,\ Star\ (Times\ (Atom\ y)\ (Atom\ x)),\ Atom\ y,\ Atom\ y])$
shows $TS\_inv\ (config′\_rand\ (embed\ (rTS\ h0))\ s\ qs)\ (last\ qs)\ [x0,\ y0]$
$\wedge\ T\_on\_rand′\ (embed\ (rTS\ h0))\ s\ qs = (length\ qs - 2)$
⟨*proof*⟩

lemma *ts_b2′*: **assumes** $x \neq y$
$qs \in lang\ (seq[Atom\ x,\ Times\ (Atom\ y)\ (Atom\ x),\ Star\ (Times\ (Atom\ y)\ (Atom\ x)),\ Atom\ y,\ Atom\ y])$
$(\exists hs.\ h = [x,\ x]\ @\ hs) \vee h = []$
shows $T\_on′\ (rTS\ h0)\ ([x,\ y],\ h)\ qs = (length\ qs - 3)$
$\wedge\ config′\ (rTS\ h0)\ ([x,y],\ h)\ qs = ([y,x],rev\ qs@h) \wedge (\exists hs.\ (rev\ qs\ @\ h) = [y,y]@hs)$
⟨*proof*⟩

lemma *TS_b2''*: **assumes**
$x \neq y$ $\{x, y\} = \{x0, y0\}$ *TS_inv s x* $[x0, y0]$
set qs $\subseteq \{x, y\}$
$qs \in lang$ (*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*) $[x0, y0]$
$\wedge$ *T_on_rand'* (*embed* (*rTS h0*)) *s qs* = (*length qs* − *3*)
⟨*proof*⟩

lemma *TS_b'*: **assumes** $x \neq y$ $h = [] \vee (\exists hs.\ h = [x, x] @ hs)$
$qs \in lang$ (*seq* [*Plus* (*Atom x*) *rexp.One*, *Atom y*, *Atom x*, *Star* (*Times* (*Atom y*) (*Atom x*)), *Atom y*, *Atom y*])
shows *T_on'* (*rTS h0*) ([*x*, *y*], *h*) *qs*
$\leq$ *2* $*$ T_p [*x*, *y*] *qs* (*OPT2 qs* [*x*, *y*]) $\wedge$ *TS_inv'* (*config'* (*rTS h0*) ([*x*, *y*], *h*) *qs*) (*last qs*) [*x*,*y*]
⟨*proof*⟩

15.3.6 (x+1)yy

lemma *ts_a'*: **assumes** $x \neq y$ $qs \in lang$ (*seq* [*Plus* (*Atom x*) *One*, *Atom y*, *Atom y*])
$h = [] \vee (\exists hs.\ h = [x, x] @ hs)$
shows *TS_inv'* (*config'* (*rTS h0*) ([*x*, *y*], *h*) *qs*) (*last qs*) [*x*,*y*]
$\wedge$ *T_on'* (*rTS h0*) ([*x*, *y*], *h*) *qs* = *2*
⟨*proof*⟩

lemma *TS_a'*: **assumes** $x \neq y$
$h = [] \vee (\exists hs.\ h = [x, x] @ hs)$
and $qs \in lang$ (*seq* [*Plus* (*Atom x*) *rexp.One*, *Atom y*, *Atom y*])
shows *T_on'* (*rTS h0*) ([*x*, *y*], *h*) *qs* $\leq$ *2* $*$ T_p [*x*, *y*] *qs* (*OPT2 qs* [*x*, *y*])
$\wedge$ *TS_inv'* (*config'* (*rTS h0*) ([*x*, *y*], *h*) *qs*) (*last qs*) [*x*, *y*]
$\wedge$ *T_on'* (*rTS h0*) ([*x*, *y*], *h*) *qs* = *2*
⟨*proof*⟩

lemma *TS_a''*: **assumes**
$x \neq y$ $\{x, y\} = \{x0, y0\}$ *TS_inv s x* $[x0, y0]$
set qs $\subseteq \{x, y\}$ $qs \in lang$ (*seq* [*Plus* (*Atom x*) *One*, *Atom y*, *Atom y*])
shows
TS_inv (*config'_rand* (*embed* (*rTS h0*)) *s qs*) (*last qs*) $[x0, y0]$
$\wedge$ T_p*_on_rand'* (*embed* (*rTS h0*)) *s qs* = *2*
⟨*proof*⟩

15.3.7 x+yx(yx)*x

lemma *ts_c'*: **assumes** $x \neq y$
$v \in lang\ (seq[Times\ (Atom\ y)\ (Atom\ x),\ Star\ (Times\ (Atom\ y)\ (Atom\ x)),\ Atom\ x])$
$(\exists\, hs.\ h = [x,\ x]\ @\ hs) \vee h = [x] \vee h = []$
shows $T\_on'\ (rTS\ h0)\ ([x,\ y],\ h)\ v = (length\ v - 2)$
$\wedge\ \ config'\ (rTS\ h0)\ ([x,y],\ h)\ v = ([x,y],rev\ v@h) \wedge (\exists\, hs.\ (rev\ v\ @\ h) = [x,x]@hs)$
⟨*proof*⟩

lemma *TS_c1''*: **assumes**
$x \neq y\ \{x,\ y\} = \{x0,\ y0\}\ TS\_inv\ s\ x\ [x0,\ y0]$
$set\ qs \subseteq \{x,\ y\}$
$qs \in lang\ (seq\ [Atom\ y,\ Atom\ x,\ Star\ (Times\ (Atom\ y)\ (Atom\ x)),\ Atom\ x])$
shows $TS\_inv\ (config'\_rand\ (embed\ (rTS\ h0))\ s\ qs)\ (last\ qs)\ [x0,\ y0]$
$\wedge\ T\_on\_rand'\ (embed\ (rTS\ h0))\ s\ qs = (length\ qs - 2)$
⟨*proof*⟩

lemma *ts_c2'*: **assumes** $x \neq y$
$qs \in lang\ (seq[Atom\ x,\ Times\ (Atom\ y)\ (Atom\ x),\ Star\ (Times\ (Atom\ y)\ (Atom\ x)),\ Atom\ x])$
$(\exists\, hs.\ h = [x,\ x]\ @\ hs) \vee h = []$
shows $T\_on'\ (rTS\ h0)\ ([x,\ y],\ h)\ qs = (length\ qs - 3)$
$\wedge\ \ config'\ (rTS\ h0)\ ([x,y],\ h)\ qs = ([x,y],rev\ qs@h) \wedge (\exists\, hs.\ (rev\ qs\ @\ h) = [x,x]@hs)$
⟨*proof*⟩

lemma *TS_c2''*: **assumes**
$x \neq y\ \{x,\ y\} = \{x0,\ y0\}\ TS\_inv\ s\ x\ [x0,\ y0]$
$set\ qs \subseteq \{x,\ y\}$
$qs \in lang\ (seq\ [Atom\ x,\ Atom\ y,\ Atom\ x,\ Star\ (Times\ (Atom\ y)\ (Atom\ x)),\ Atom\ x])$
shows $TS\_inv\ (config'\_rand\ (embed\ (rTS\ h0))\ s\ qs)\ (last\ qs)\ [x0,\ y0]$
$\wedge\ T\_on\_rand'\ (embed\ (rTS\ h0))\ s\ qs = (length\ qs - 3)$
⟨*proof*⟩

lemma *TS_c'*: **assumes** $x \neq y\ h = [] \vee (\exists\, hs.\ h = [x,\ x]\ @\ hs)$
$qs \in lang\ (seq\ [Plus\ (Atom\ x)\ rexp.One,\ Atom\ y,\ Atom\ x,\ Star\ (Times\ (Atom\ y)\ (Atom\ x)),\ Atom\ x])$

shows $T\_on'\ (rTS\ h0)\ ([x,\ y],\ h)\ qs$
$\leq 2 * T_p\ [x,\ y]\ qs\ (OPT2\ qs\ [x,\ y]) \wedge\ TS\_inv'\ (config'\ (rTS\ h0)\ ([x,\ y],\ h)\ qs)\ (last\ qs)\ [x,y]$
⟨*proof*⟩

15.3.8 xx

lemma *request_first*: $x \neq y \Longrightarrow Step\ (rTS\ h)\ ([x,\ y],\ is)\ x = ([x,y], x\#is)$
⟨*proof*⟩

lemma *ts_d'*: $qs \in Lxx\ x\ y \Longrightarrow$
$x \neq y \Longrightarrow$
$h = [] \vee (\exists\, hs.\ h = [x,\ x]\ @\ hs) \Longrightarrow$
$qs \in lang\ (seq\ [Atom\ x,\ Atom\ x]) \Longrightarrow$
$T\_on'\ (rTS\ h0)\ ([x,\ y],\ h)\ qs = 0 \wedge$
$TS\_inv'\ (config'\ (rTS\ h0)\ ([x,\ y],\ h)\ qs)\ x\ [x,y]$
⟨*proof*⟩

lemma *TS_d'*: **assumes** *xny*: $x \neq y$ **and** $h = [] \vee (\exists\, hs.\ h = [x,\ x]\ @\ hs)$
and *qsis*: $qs \in lang\ (seq\ [Atom\ x,\ Atom\ x])$
shows $T\_on'\ (rTS\ h0)\ ([x,y],h)\ qs \leq 2 * T_p\ [x,\ y]\ qs\ (OPT2\ qs\ [x,\ y])$

and $TS\_inv'\ (config'\ (rTS\ h0)\ ([x,y],h)\ qs)\ \ (last\ qs)\ [x,\ y]$
and $T\_on'\ (rTS\ h0)\ ([x,y],h)\ qs = 0$
⟨*proof*⟩

lemma *TS_d''*: **assumes**
$x \neq y\ \{x,\ y\} = \{x0,\ y0\}\ TS\_inv\ s\ x\ [x0,\ y0]$
$set\ qs \subseteq \{x,\ y\}$
$qs \in lang\ (seq\ [Atom\ x,\ Atom\ x])$
shows $TS\_inv\ (config'\_rand\ (embed\ (rTS\ h0))\ s\ qs)\ (last\ qs)\ [x0,\ y0]$
$\wedge\ T\_on\_rand'\ (embed\ (rTS\ h0))\ s\ qs = 0$
⟨*proof*⟩

15.4 Phase Partitioning

lemma *D'*: **assumes** $\sigma' \in Lxx\ x\ y$ **and** $x \neq y$ **and** $TS\_inv'\ ([x,\ y],\ h)\ x\ [x,\ y]$
shows $T\_on'\ (rTS\ h0)\ ([x,\ y],\ h)\ \sigma' \leq 2 * T_p\ [x,\ y]\ \sigma'\ (OPT2\ \sigma'\ [x,\ y])$

$\wedge\ TS\_inv\ (config'\_rand\ (embed\ (rTS\ h0))\ (return\_pmf\ ([x,\ y],\ h))\ \sigma')$

(*last* σ') [x, y]
⟨*proof*⟩

theorem *TS_OPT2'*: (*x*::*nat*) ≠ *y* ⟹ *set* $\sigma \subseteq \{x,y\}$
⟹ T_{p}_*on* (*rTS* []) [*x*,*y*] σ ≤ 2 ∗ *real* (T_{p}_*opt* [*x*,*y*] σ) + 2
⟨*proof*⟩

15.5 TS is pairwise

lemma *config'_distinct*[*simp*]:
shows *distinct* (*fst* (*config'* *A* *S* *qs*)) = *distinct* (*fst* *S*)
⟨*proof*⟩

lemma *config'_set*[*simp*]:
shows *set* (*fst* (*config'* *A* *S* *qs*)) = *set* (*fst* *S*)
⟨*proof*⟩

lemma *s_TS_append*: *i*≤*length as* ⟹*s_TS init h* (*as*@*bs*) *i* = *s_TS init h as i*
⟨*proof*⟩

lemma *s_TS_distinct*: *distinct init* ⟹ *i*<*length qs* ⟹ *distinct* (*fst* (*TSdet init h qs i*))
⟨*proof*⟩

lemma *othersdontinterfere*: *distinct init* ⟹ *i* < *length qs* ⟹ *a*∈*set init* ⟹ *b*∈*set init*
⟹ *set qs* ⊆ *set init* ⟹ *qs*!*i*∉{*a*,*b*} ⟹ *a* < *b in s_TS init h qs i* ⟹ *a* < *b in s_TS init h qs* (*Suc i*)
⟨*proof*⟩

lemma *TS_mono*:
fixes *l*::*nat*
assumes *1*: *x* < *y in s_TS init h xs* (*length xs*)
and *l_in_cs*: *l* ≤ *length cs*
and *firstocc*: ∀ *j*<*l*. *cs* ! *j* ≠ *y*
and *x* ∉ *set cs*
and *di*: *distinct init*
and *inin*: *set* (*xs* @ *cs*) ⊆ *set init*
shows *x* < *y in s_TS init h* (*xs*@*cs*) (*length* (*xs*)+*l*)
⟨*proof*⟩

lemma *step_no_action*: *step s q* (*0*,[]) = *s*
⟨*proof*⟩

lemma *s_TS_set*: $i \leq length\ qs \Longrightarrow set\ (s\_TS\ init\ h\ qs\ i) = set\ init$
⟨*proof*⟩

lemma *count_notin2*: $count\_list\ xs\ x = 0 \Longrightarrow x \notin set\ xs$
⟨*proof*⟩

lemma *count_append*: $count\_list\ (xs@ys)\ x = count\_list\ xs\ x + count\_list\ ys\ x$
⟨*proof*⟩

lemma *count_rev*: $count\_list\ (rev\ xs)\ x = count\_list\ xs\ x$
⟨*proof*⟩

lemma *mtf2_q_passes*: **assumes** $q \in set\ xs\ distinct\ xs$
and $index\ xs\ q - n \leq index\ xs\ x\ index\ xs\ x < index\ xs\ q$
shows $q < x\ in\ (mtf2\ n\ q\ xs)$
⟨*proof*⟩

lemma *twotox*:
assumes $count\_list\ bs\ y \leq 1$
and $distinct\ init$
and $x \in set\ init$
and $y : set\ init$
and $x \notin set\ bs$
and $x \neq y$
shows $x < y\ in\ s\_TS\ init\ h\ (as@[x]@bs@[x])\ (length\ (as@[x]@bs@[x]))$
⟨*proof*⟩

lemma *count_drop*: $count\_list\ (drop\ n\ cs)\ x \leq count\_list\ cs\ x$
⟨*proof*⟩

lemma *count_take_less*: **assumes** $n \leq m$
shows $count\_list\ (take\ n\ cs)\ x \leq count\_list\ (take\ m\ cs)\ x$
⟨*proof*⟩

lemma *count_take*: $count\_list\ (take\ n\ cs)\ x \leq count\_list\ cs\ x$
⟨*proof*⟩

lemma *casexxy*: **assumes** $\sigma = as@[x]@bs@[x]@cs$
and $x \notin set\ cs$
and $set\ cs \subseteq set\ init$
and $x \in set\ init$
and $distinct\ init$

and $x \notin set\ bs$
and $set\ as \subseteq set\ init$
and $set\ bs \subseteq set\ init$
shows $(\%i.\ i{<}length\ cs \longrightarrow (\forall j{<}i.\ cs!j{\neq}cs!i) \longrightarrow cs!i{\neq}x \longrightarrow (cs!i) \notin set\ bs \longrightarrow x < (cs!i)\ in\ (s\_TS\ init\ h\ \sigma\ (length\ (as@[x]@bs@[x]) + i{+}1)))\ i$
⟨*proof*⟩

lemma *nopaid*: $snd\ (fst\ (TS\_step\_d\ s\ q)) = []$ ⟨*proof*⟩

lemma *staysuntouched*:
assumes $d[simp]$: $distinct\ (fst\ S)$
and x: $x \in set\ (fst\ S)$
and y: $y \in set\ (fst\ S)$
shows $set\ qs \subseteq set\ (fst\ S) \Longrightarrow x \notin set\ qs \Longrightarrow y \notin set\ qs \Longrightarrow x < y\ in\ fst\ (config'\ (rTS\ [])\ S\ qs) = x < y\ in\ fst\ S$
⟨*proof*⟩

lemma *staysuntouched'*:
assumes $d[simp]$: $distinct\ init$
and x: $x \in set\ init$
and y: $y \in set\ init$
and $set\ qs \subseteq set\ init$
and $x \notin set\ qs$ **and** $y \notin set\ qs$
shows $x < y\ in\ fst\ (config\ (rTS\ [])\ init\ qs) = x < y\ in\ init$
⟨*proof*⟩

lemma *projEmpty*: $Lxy\ qs\ S = [] \Longrightarrow x \in S \Longrightarrow x \notin set\ qs$
⟨*proof*⟩

lemma *Lxy_index_mono*:
assumes $x{\in}S\ y{\in}S$
and $index\ xs\ x < index\ xs\ y$
and $index\ xs\ y < length\ xs$
and $x{\neq}y$
shows $index\ (Lxy\ xs\ S)\ x < index\ (Lxy\ xs\ S)\ y$
⟨*proof*⟩

lemma *proj_Cons*:
assumes *filterd_cons*: $Lxy\ qs\ S = a\#as$
and *a_filter*: $a{\in}S$
obtains *pre suf* **where** $qs = pre\ @\ [a]\ @\ suf$ **and** $\bigwedge x.\ x \in S \Longrightarrow x \notin set\ pre$
and $Lxy\ suf\ S = as$

⟨*proof*⟩

lemma *Lxy_rev*: *rev* (*Lxy qs S*) = *Lxy* (*rev qs*) *S*
⟨*proof*⟩

lemma *proj_Snoc*:
assumes *filterd_cons*: *Lxy qs S* = *as*@[*a*]
and *a_filter*: *a*∈*S*
obtains *pre suf* **where** *qs* = *pre* @ [*a*] @ *suf* **and** ⋀*x*. *x* ∈ *S* ⟹ *x* ∉ *set suf*
and *Lxy pre S* = *as*
⟨*proof*⟩

lemma *sndTSconfig'*: *snd* (*config'* (*rTS initH*) (*init*,[]) *qs*) = *rev qs* @ []
⟨*proof*⟩

lemma *projxx*:
fixes *e a bs*
assumes *axy*: *a*∈{*x*,*y*}
assumes *ane*: *a*≠*e*
assumes *exy*: *e*∈{*x*,*y*}
assumes *add*: *f*∈{[],[*e*]}
assumes *bsaxy*: *set* (*bs* @ [*a*] @ *f*) ⊆ {*x*,*y*}
assumes *Lxyinitxy*: *Lxy init* {*x*, *y*} ∈ {[*x*,*y*],[*y*,*x*]}
shows *a* < *e in fst* (*config*$_p$ (*rTS* []) (*Lxy init* {*x*, *y*}) ((*bs* @ [*a*] @ *f*) @ [*a*]))
⟨*proof*⟩

lemma *oneposs*:
assumes *set xs* = {*x*,*y*}
assumes *x*≠*y*
assumes *distinct xs*
assumes *True*: *x*<*y in xs*
shows *xs* = [*x*,*y*]
⟨*proof*⟩

lemma *twoposs*:
assumes *set xs* = {*x*,*y*}
assumes *x*≠*y*
assumes *distinct xs*
shows *xs* ∈ {[*x*,*y*], [*y*,*x*]}
⟨*proof*⟩

lemma *TS_pairwise'*: **assumes** *qs* ∈ {*xs*. *set xs* ⊆ *set init*}

$(x, y) \in \{(x, y).\ x \in set\ init \wedge y \in set\ init \wedge x \neq y\}$
$x \neq y$ *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*⟩

theorem *TS_pairwise*: *pairwise* (*embed* (*rTS* []))
⟨*proof*⟩

15.6 TS is 2-compet

lemma *TS_compet'*: *pairwise* (*embed* (*rTS* [])) ⟹
$\forall\, s0 \in \{init::(nat\ list).\ distinct\ init \wedge init \neq []\}.\ \exists\, b \geq 0.\ \forall\, qs \in \{x.\ set\ x \subseteq set\ s0\}.\ T_p\_on\_rand\ (embed\ (rTS\ [])\ s0\ qs \leq (2::real) *\ T_p\_opt\ s0\ qs + b$
⟨*proof*⟩

lemma *TS_compet*: *compet_rand* (*embed* (*rTS* [])) *2* {*init*. *distinct init* ∧ *init* ≠ []}
⟨*proof*⟩

end

16 BIT is pairwise

theory *BIT_pairwise*
imports *List_Factoring BIT*
begin

lemma *L_nths*: $S \subseteq \{..<length\ init\}$
⟹ *map_pmf* (λ*l*. *nths l S*) (*Prob_Theory.bv* (*length init*))
= (*Prob_Theory.bv* (*length* (*nths init S*)))
⟨*proof*⟩

lemma *L_nths_Lxy*:
assumes $x \in set\ init\ y \in set\ init\ x \neq y\ distinct\ init$
shows *map_pmf* (λ*l*. *nths l* {*index init x*,*index init y*}) (*Prob_Theory.bv* (*length init*))
= (*Prob_Theory.bv* (*length* (*Lxy init* {*x*,*y*})))
⟨*proof*⟩

lemma *nths_map*: *map f* (*nths xs S*) = *nths* (*map f xs*) *S*
⟨*proof*⟩

lemma *nths_empty*: $(\forall\, i{\in}S.\ i{\geq}length\ xs) \Longrightarrow nths\ xs\ S = []$
⟨*proof*⟩

lemma *nths_project'*: $i < length\ xs \Longrightarrow j < length\ xs \Longrightarrow i{<}j$
$\Longrightarrow nths\ xs\ \{i,j\} = [xs!i,\ xs!j]$
⟨*proof*⟩

lemma *nths_project*:
assumes $i < length\ xs\ j < length\ xs\ i{<}j$
shows $nths\ xs\ \{i,j\}\ !\ 0 = xs\ !\ i \wedge nths\ xs\ \{i,j\}\ !\ 1 = xs\ !\ j$
⟨*proof*⟩

lemma *BIT_pairwise'*:
assumes $set\ qs \subseteq set\ init$
$(x,y){\in}\ \{(x,y).\ x \in set\ init \wedge y{\in}set\ init \wedge x{\neq}y\}$
and $xny{:}x \neq y$ **and** *dinit*: *distinct init*
shows $Pbefore\_in\ x\ y\ BIT\ qs\ init = Pbefore\_in\ x\ y\ BIT\ (Lxy\ qs\ \{x,y\})$
$(Lxy\ init\ \{x,y\})$
⟨*proof*⟩

theorem *BIT_pairwise*: *pairwise BIT*
⟨*proof*⟩

end

17 BIT is 1.75 competitive on lists of length 2

theory *BIT_2comp_on2*
imports *BIT Phase_Partitioning*
begin

17.1 auxliary 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*⟩

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))*
⟨*proof*⟩

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 ≫=

$(\lambda(a,\ is').\ return\_pmf\ (real\ (t_p\ (fst\ s)\ x\ a))))) = 0$
⟨*proof*⟩

lemma *costBIT_0y*:
assumes $x{\neq}y$ $x : \{x0,y0\}$ $y{\in}\{x0,y0\}$
shows
$E\ (type0\ [x0,\ y0]\ x\ y \gg\!=$
$(\lambda s.\ BIT\_step\ s\ y \gg\!=$
$(\lambda(a,\ is').\ return\_pmf\ (real\ (t_p\ (fst\ s)\ y\ a))))) = 1$
⟨*proof*⟩

lemma *costBIT_1x*:
assumes $x{\neq}y$ $x : \{x0,y0\}$ $y{\in}\{x0,y0\}$
shows
$E\ (type1\ [x0,\ y0]\ x\ y \gg\!=$
$(\lambda s.\ BIT\_step\ s\ x \gg\!=$
$(\lambda(a,\ is').\ return\_pmf\ (real\ (t_p\ (fst\ s)\ x\ a))))) = 1/4$
⟨*proof*⟩

lemma *costBIT_1y*:
assumes $x{\neq}y$ $x : \{x0,y0\}$ $y{\in}\{x0,y0\}$
shows
$E\ (type1\ [x0,\ y0]\ x\ y \gg\!=$
$(\lambda s.\ BIT\_step\ s\ y \gg\!=$
$(\lambda(a,\ is').\ return\_pmf\ (real\ (t_p\ (fst\ s)\ y\ a))))) = 3/4$
⟨*proof*⟩

lemma *costBIT_3x*:
assumes $x{\neq}y$ $x : \{x0,y0\}$ $y{\in}\{x0,y0\}$
shows
$E\ (type3\ [x0,\ y0]\ x\ y \gg\!=$
$(\lambda s.\ BIT\_step\ s\ x \gg\!=$
$(\lambda(a,\ is').\ return\_pmf\ (real\ (t_p\ (fst\ s)\ x\ a))))) = 3/4$
⟨*proof*⟩

lemma *costBIT_3y*:
assumes $x{\neq}y$ $x : \{x0,y0\}$ $y{\in}\{x0,y0\}$
shows
$E\ (type3\ [x0,\ y0]\ x\ y \gg\!=$
$(\lambda s.\ BIT\_step\ s\ y \gg\!=$
$(\lambda(a,\ is').\ return\_pmf\ (real\ (t_p\ (fst\ s)\ y\ a))))) = 1/4$
⟨*proof*⟩

lemma *costBIT_4x*:

assumes $x{\neq}y$ $x : \{x0,y0\}$ $y{\in}\{x0,y0\}$
shows
$E\ (type4\ [x0,\ y0]\ x\ y \gg\!=$
$(\lambda s.\ BIT\_step\ s\ x \gg\!=$
$(\lambda(a,\ is').\ return\_pmf\ (real\ (t_p\ (fst\ s)\ x\ a))))) = 0.5$
⟨*proof*⟩

lemma *costBIT_4y*:
assumes $x{\neq}y$ $x : \{x0,y0\}$ $y{\in}\{x0,y0\}$
shows
$E\ (type4\ [x0,\ y0]\ x\ y \gg\!=$
$(\lambda s.\ BIT\_step\ s\ y \gg\!=$
$(\lambda(a,\ is').\ return\_pmf\ (real\ (t_p\ (fst\ s)\ y\ a))))) = 0.5$
⟨*proof*⟩

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 $BIT\_Step\ s\ x == (s \gg\!= (\lambda s.\ BIT\_step\ s\ x \gg\!= (\lambda(a,\ is').\ return\_pmf\ (step\ (fst\ s)\ x\ a,\ is'))))$

lemma *oneBIT_step0x*:
assumes $x{\neq}y$ $x : \{x0,y0\}$ $y{\in}\{x0,y0\}$
shows $BIT\_Step\ (type0\ [x0,\ y0]\ x\ y)\ x = type0\ [x0,\ y0]\ x\ y$
⟨*proof*⟩

lemma *oneBIT_step0y*:
assumes $x{\neq}y$ $x : \{x0,y0\}$ $y{\in}\{x0,y0\}$
shows $BIT\_Step\ (type0\ [x0,\ y0]\ x\ y)\ y = type4\ [x0,\ y0]\ x\ y$
⟨*proof*⟩

lemma *oneBIT_step1x*:
assumes $x{\neq}y$ $x : \{x0,y0\}$ $y{\in}\{x0,y0\}$
shows $BIT\_Step\ (type1\ [x0,\ y0]\ x\ y)\ x = type0\ [x0,\ y0]\ x\ y$
⟨*proof*⟩

lemma *oneBIT_step1y*:
assumes $x{\neq}y$ $x : \{x0,y0\}$ $y{\in}\{x0,y0\}$
shows $BIT\_Step\ (type1\ [x0,\ y0]\ x\ y)\ y = type3\ [x0,\ y0]\ x\ y$
⟨*proof*⟩

lemma *oneBIT_step3x*:

assumes $x{\neq}y$ $x{:}\{x0,y0\}$ $y{:}\{x0,y0\}$
shows *BIT_Step* (*type3* [*x0*, *y0*] *x y*) *x* = *type1* [*x0*, *y0*] *x y*
⟨*proof*⟩

lemma *oneBIT_step3y*:
assumes $x{\neq}y$ $x : \{x0,y0\}$ $y{\in}\{x0,y0\}$
shows *BIT_Step* (*type3* [*x0*, *y0*] *x y*) *y* = *type0* [*x0*, *y0*] *y x*
⟨*proof*⟩

lemma *oneBIT_step4x*:
assumes $x{\neq}y$ $x : \{x0,y0\}$ $y{\in}\{x0,y0\}$
shows *BIT_Step* (*type4* [*x0*, *y0*] *x y*) *x* = *type1* [*x0*, *y0*] *x y*
⟨*proof*⟩

lemma *oneBIT_step4y*:
assumes $x{\neq}y$ $x : \{x0,y0\}$ $y{\in}\{x0,y0\}$
shows *BIT_Step* (*type4* [*x0*, *y0*] *x y*) *y* = *type0* [*x0*, *y0*] *y x*
⟨*proof*⟩

lemmas *oneBIT_step* = *oneBIT_step0x oneBIT_step0y oneBIT_step1x oneBIT_step1y oneBIT_step3x oneBIT_step3y oneBIT_step4x oneBIT_step4y*

17.2 Analysis of the four phase forms

17.2.1 yx

lemma *bit_yx*: **assumes** $x \neq y$
and *kas*: *init* ∈ {[*x*,*y*],[*y*,*x*]}
and *qs* ∈ *lang* (*Star*(*Times* (*Atom y*) (*Atom x*)))
shows T_p*_on_rand′ BIT* (*type1 init x y*) (*qs*@*r*) = *0.75* ∗ *length qs* + T_p*_on_rand′ BIT* (*type1 init x y*) *r*
∧ *config′_rand BIT* (*type1 init x y*) *qs* = (*type1 init x y*)
⟨*proof*⟩

17.2.2 (yx)*yx

lemma *bit_yxyx*: **assumes** $x \neq y$ **and** *kas*: *init* ∈ {[*x*,*y*],[*y*,*x*]} **and**
qs ∈ *lang* (*seq*[*Times* (*Atom y*) (*Atom x*), *Star*(*Times* (*Atom y*) (*Atom x*))])
shows T_p*_on_rand′ BIT* (*type0 init x y*) (*qs*@*r*) = *0.75* ∗ *length qs* + T_p*_on_rand′ BIT* (*type1 init x y*) *r*
∧ *config′_rand BIT* (*type0 init x y*) *qs* = (*type1 init x y*)
⟨*proof*⟩

17.2.3 $x\hat{}+..$

lemma *BIT_x*: **assumes** $x \neq y$
$init \in \{[x,y],[y,x]\}$ $qs \in lang\ (Plus\ (Atom\ x)\ One)$
shows $T_p\_on\_rand'\ BIT\ (type0\ init\ x\ y)\ (qs@r) = T_p\_on\_rand'\ BIT\ (type0\ init\ x\ y)\ r$
$\wedge\ config'\_rand\ BIT\ \ (type0\ init\ x\ y)\ qs = (type0\ init\ x\ y)$
⟨*proof*⟩

17.2.4 Phase Form A

lemma *BIT_a*: **assumes** $x \neq y$
$init \in \{[x,y],[y,x]\}$
$qs \in lang\ (seq\ [Plus\ (Atom\ x)\ One,\ Atom\ y,\ Atom\ y])$
shows $config'\_rand\ BIT\ (type0\ init\ x\ y)\ qs = (type0\ init\ y\ x)$ (**is** *?C*)
and *b*: $T_p\_on\_rand'\ BIT\ (type0\ init\ x\ y)\ qs = 1.5$ (**is** *?T*)
⟨*proof*⟩

lemma *bit_a*: **assumes**
$x \neq y\ \{x,\ y\} = \{x0,\ y0\}\ BIT\_inv\ s\ x\ [x0,\ y0]$
$set\ qs \subseteq \{x,\ y\}\ qs \in lang\ (seq\ [Plus\ (Atom\ x)\ One,\ Atom\ y,\ Atom\ y])$
shows
$T_p\_on\_rand'\ BIT\ s\ qs\ \leq 1.75 * T_p\ [x,y]\ qs\ (OPT2\ qs\ [x,y])$
$\wedge\ \ BIT\_inv\ (config'\_rand\ BIT\ s\ qs)\ (last\ qs)\ [x0,\ y0]$
$\wedge\ T_p\_on\_rand'\ BIT\ s\ qs = 1.5$
⟨*proof*⟩

lemma *bit_a''*: $a \neq b \Longrightarrow$
$\{a,\ b\} = \{x,\ y\} \Longrightarrow$
$BIT\_inv\ s\ a\ [x,\ y] \Longrightarrow$
$set\ qs \subseteq \{a,\ b\} \Longrightarrow$
$qs \in lang\ (seq\ [question\ (Atom\ a),\ Atom\ b,\ Atom\ b]) \Longrightarrow$
$BIT\_inv\ (Partial\_Cost\_Model.config'\_rand\ BIT\ s\ qs)\ (last\ qs)\ [x,\ y]$
$\wedge\ T_p\_on\_rand'\ BIT\ s\ qs = 1.5$
⟨*proof*⟩

17.2.5 Phase Form B

lemma *BIT_b*: **assumes** *A*: $x \neq y$
$init \in \{[x,y],[y,x]\}$
$v \in lang\ (seq\ [Times\ (Atom\ y)\ (Atom\ x),\ Star\ (Times\ (Atom\ y)\ (Atom\ x)),\ Atom\ y,\ Atom\ y])$
shows $T_p\_on\_rand'\ BIT\ (type0\ init\ x\ y)\ v = 0.75 * length\ v - 0.5$ (**is** *?T*)
and $config'\_rand\ BIT\ \ (type0\ init\ x\ y)\ v = (type0\ init\ y\ x)$ (**is** *?C*)

⟨*proof*⟩

lemma *bit_b''1*: **assumes**
$x \neq y$ $\{x, y\} = \{x0, y0\}$ *BIT_inv s x* $[x0, y0]$
set qs $\subseteq \{x, y\}$
qs $\in$ *lang* (*seq*[*Atom y*, *Atom x*, *Star*(*Times* (*Atom y*) (*Atom x*)), *Atom y*, *Atom y*])
shows *BIT_inv* (*config'_rand BIT s qs*) (*last qs*) $[x0, y0] \wedge$
T_p*_on_rand' BIT s qs* $= 0.75 *$ *length qs* $- 0.5$
⟨*proof*⟩

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*) $v = 0.75 * (\text{length } v - 1) - 0.5$ (**is** *?T*)
and *config'_rand BIT* (*type0 init x y*) $v =$ (*type0 init y x*) (**is** *?C*)
⟨*proof*⟩

lemma *bit_b''2*: **assumes**
$x \neq y$ $\{x, y\} = \{x0, y0\}$ *BIT_inv s x* $[x0, y0]$
set qs $\subseteq \{x, y\}$
qs $\in$ *lang* (*seq*[*Atom x*, *Atom y*, *Atom x*, *Star*(*Times* (*Atom y*) (*Atom x*)), *Atom y*, *Atom y*])
shows *BIT_inv* (*config'_rand BIT s qs*) (*last qs*) $[x0, y0] \wedge$
T_p*_on_rand' BIT s qs* $= 0.75 * (\text{length } qs - 1) - 0.5$
⟨*proof*⟩

lemma *bit_b*: **assumes** $x \neq y$
init $\in \{[x,y],[y,x]\}$
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*_on_rand' BIT* (*type0 init x y*) *qs* $\leq 1.75 * T_p$ $[x,y]$ *qs* (*OPT2 qs* $[x,y]$)
and *config'_rand BIT* (*type0 init x y*) *qs* = *type0 init y x*
⟨*proof*⟩

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*_on_rand' BIT s qs* $\leq$ *1.75* $*$ T_p [*x*,*y*] *qs* (*OPT2 qs* [*x*,*y*])
$\wedge$ *BIT_inv* (*config'_rand BIT s qs*) (*last qs*) [*x0*, *y0*]
⟨*proof*⟩

lemma *bit_b'''*: *a* $\neq$ *b* $\Longrightarrow$
{a, b} = *{x, y}* $\Longrightarrow$
BIT_inv s a [*x*, *y*] $\Longrightarrow$
set qs $\subseteq$ *{a, b}* $\Longrightarrow$
qs $\in$ *lang* (*seq*[*Plus* (*Atom x*) *One*, *Atom y*, *Atom x*, *Star*(*Times* (*Atom y*) (*Atom x*)), *Atom y*, *Atom y*]) $\Longrightarrow$
BIT_inv (*Partial_Cost_Model.config'_rand BIT s qs*) (*last qs*) [*x*, *y*] $\wedge$ T_p*_on_rand' BIT s qs* = *1.5*
⟨*proof*⟩

17.2.6 Phase Form C

lemma *BIT_c*: **assumes** *x* $\neq$ *y*
init $\in$ {[*x*,*y*],[*y*,*x*]}
v $\in$ *lang* (*seq* [*Times* (*Atom y*) (*Atom x*), *Star* (*Times* (*Atom y*) (*Atom x*)), *Atom x*])
shows T_p*_on_rand' BIT* (*type0 init x y*) *v* = *0.75* $*$ *length v* $-$ *0.5*
and *config'_rand BIT* (*type0 init x y*) *v* = (*type0 init x y*) (**is** *?C*)
⟨*proof*⟩

lemma *bit_c''1*: **assumes**
x $\neq$ *y* *{x, y}* = *{x0, y0}* *BIT_inv s x* [*x0*, *y0*]
set qs $\subseteq$ *{x, y}*
qs $\in$ *lang* (*seq*[*Atom y*, *Atom x*, *Star*(*Times* (*Atom y*) (*Atom x*)), *Atom x*])
shows *BIT_inv* (*config'_rand BIT s qs*) (*last qs*) [*x0*, *y0*] $\wedge$
T_p*_on_rand' BIT s qs* = *0.75* $*$ *length qs* $-$ *0.5*
⟨*proof*⟩

lemma *bit_c*: **assumes** *x* $\neq$ *y*
init $\in$ {[*x*,*y*],[*y*,*x*]}
qs $\in$ *lang* (*seq*[*Plus* (*Atom x*) *One*, *Atom y*, *Atom x*, *Star*(*Times* (*Atom y*) (*Atom x*)), *Atom x*])
shows T_p*_on_rand' BIT* (*type0 init x y*) *qs* $\leq$ *1.75* $*$ T_p [*x*,*y*] *qs* (*OPT2 qs* [*x*,*y*])
and *config'_rand BIT* (*type0 init x y*) *qs* = *type0 init x y*

⟨*proof*⟩

lemma *bit_c''*: **assumes**
x ≠ *y* {*x*, *y*} = {*x0*, *y0*} *BIT_inv s x* [*x0*, *y0*]
set qs ⊆ {*x*, *y*}
qs ∈ *lang* (*seq*[*Plus* (*Atom x*) *One*, *Atom y*, *Atom x*, *Star*(*Times* (*Atom y*) (*Atom x*)), *Atom x*])
shows
T_p*_on_rand' BIT s qs* ≤ *1.75* ∗ T_p [*x*,*y*] *qs* (*OPT2 qs* [*x*,*y*])
∧ *BIT_inv* (*config'_rand BIT s qs*) (*last qs*) [*x0*, *y0*]
⟨*proof*⟩

lemma *BIT_c2*: **assumes** *A*: *x* ≠ *y*
init ∈ {[*x*,*y*],[*y*,*x*]}
v ∈ *lang* (*seq* [*Atom x*, *Times* (*Atom y*) (*Atom x*), *Star* (*Times* (*Atom y*) (*Atom x*)), *Atom x*])
shows T_p*_on_rand' BIT* (*type0 init x y*) *v* = *0.75* ∗ (*length v* − *1*) − *0.5* (**is** *?T*)
and *config'_rand BIT* (*type0 init x y*) *v* = (*type0 init x y*) (**is** *?C*)
⟨*proof*⟩

lemma *bit_c''2*: **assumes**
x ≠ *y* {*x*, *y*} = {*x0*, *y0*} *BIT_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 *BIT_inv* (*config'_rand BIT s qs*) (*last qs*) [*x0*, *y0*] ∧
T_p*_on_rand' BIT s qs* = *0.75* ∗ (*length qs* − *1*) − *0.5*
⟨*proof*⟩

17.2.7 Phase Form D

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 T_p*_on_rand' BIT s qs* ≤ *175* / 10^2 ∗ *real* (T_p [*x*, *y*] *qs* (*OPT2 qs* [*x*, *y*])) ∧
BIT_inv (*config'_rand BIT s qs*) (*last qs*) [*x0*, *y0*] ∧
T_p*_on_rand' BIT s qs* = *0*
⟨*proof*⟩

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*
⟨*proof*⟩

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*]
⟨*proof*⟩

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* [*a*, *b*])) ∧
BIT_inv (*Partial_Cost_Model.config'_rand BIT s qs*) (*last qs*) [*x*, *y*]
⟨*proof*⟩

theorem *BIT_175comp_on_2*:
assumes (*x*::*nat*) ≠ *y set* σ ⊆ {*x*,*y*}
shows T_p*_on_rand BIT* [*x*,*y*] σ ≤ *1.75* ∗ *real* (T_p*_opt* [*x*,*y*] σ) + *1.75*

⟨*proof*⟩

end

18 COMB

theory *Comb*
imports *TS BIT_2comp_on2 BIT_pairwise*
begin

18.1 Definition of COMB

type_synonym *CombState* = (*bool list* ∗ *nat list*) + (*nat list*)

definition *COMB_init* :: *nat list* ⇒ (*nat state*, *CombState*) *alg_on_init*
where
COMB_init h init =
Sum_pmf 0.8 (*fst BIT init*) (*fst* (*embed* (*rTS h*)) *init*)

lemma *COMB_init*[*simp*]: *COMB_init h init* =
do {
(*b*::*bool*) ← (*bernoulli_pmf 0.8*);
(*xs*::*bool list*) ← *Prob_Theory.bv* (*length init*);
return_pmf (*if b then Inl* (*xs*, *init*) *else Inr h*)
}
⟨*proof*⟩

definition *COMB_step* :: (*nat state*, *CombState*, *nat*, *answer*) *alg_on_step*
where
COMB_step s q = (*case snd s of Inl b* ⇒ *map_pmf* (λ((*a*,*b*),*c*). ((*a*,*b*),*Inl c*)) (*BIT_step* (*fst s*, *b*) *q*)
| *Inr b* ⇒ *map_pmf* (λ((*a*,*b*),*c*). ((*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*) ⇒ (*s*,*Inl is*) | *Inr* (*s*,*is*) ⇒ (*s*,*Inr is*))
abbreviation *con* == (%(*s*,*is*). *case is of Inl is* ⇒ *Inl* (*s*,*is*) | *Inr is* ⇒ *Inr* (*s*,*is*))

definition *inv_COMB s x i* == (∃ *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*
⟨*proof*⟩

lemma *connoc*: *con o noc* = *id*
⟨*proof*⟩

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 BIT Da qs*)
(*config′_rand* (*embed* (*rTS h*)) *Db qs*))
⟨*proof*⟩

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*))
⟨*proof*⟩

lemma *obligation1*: **assumes** *map_pmf con s* = *Sum_pmf* (*8 / 10*) *Da Db*
shows *map_pmf con* (*config′_rand* (*COMB* []) *s qs*) =
Sum_pmf (*8 / 10*) (*config′_rand BIT Da qs*)
(*config′_rand* (*embed* (*rTS* [])) *Db qs*)
⟨*proof*⟩

lemma *BIT_config′_fin*: *finite* (*set_pmf s*) ⟹ *finite* (*set_pmf* (*config′_rand BIT s qs*))
⟨*proof*⟩

lemma *TS_config′_fin*: *finite* (*set_pmf s*) ⟹ *finite* (*set_pmf* (*config′_rand* (*embed* (*rTS h*)) *s qs*))
⟨*proof*⟩

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′* (*COMB* []) *s qs* =
2 / 10 ∗ T_p*_on_rand′* (*embed* (*rTS* [])) *Db qs* +
8 / 10 ∗ T_p*_on_rand′ BIT Da qs*
⟨*proof*⟩

lemma *Combination*:
fixes *bit*
assumes *qs* ∈ *pattern a* ≠ *b* {*a*, *b*} = {*x*, *y*} *set qs* ⊆ {*a*, *b*}
and *inv_COMB s a* [*x*,*y*]
and *TS*: ⋀*s h*. *a* ≠ *b* ⟹ {*a*, *b*} = {*x*, *y*} ⟹ *TS_inv s a* [*x*, *y*] ⟹ *set qs* ⊆ {*a*, *b*}
⟹ *qs* ∈ *pattern* ⟹
TS_inv (*config′_rand* (*embed* (*rTS h*)) *s qs*) (*last qs*) [*x*, *y*]
∧ T_p*_on_rand′* (*embed* (*rTS h*)) *s qs* = *ts*
and *BIT*: ⋀*s*. *a* ≠ *b* ⟹ {*a*, *b*} = {*x*, *y*} ⟹ *BIT_inv s a* [*x*, *y*] ⟹ *set qs* ⊆ {*a*, *b*}
⟹ *qs* ∈ *pattern* ⟹
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*⟩

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*⟩

18.3 COMB pairwise

lemma *config_rand_COMB*: *config_rand* (*COMB h*) *init qs* = *do* {
(*b*::*bool*) ← (*bernoulli_pmf 0.8*);
(*b1*,*b2*) ← (*config_rand BIT init qs*);
(*t1*,*t2*) ← (*config_rand* (*embed* (*rTS h*)) *init qs*);
return_pmf (*if b then* (*b1*, *Inl b2*) *else* (*t1*, *Inr t2*))
} (**is** *?LHS* = *?RHS*)
⟨*proof*⟩

lemma *COMB_no_paid*: ∀((*free*, *paid*), *t*)∈*set_pmf* (*snd* (*COMB* []) (*s*, *is*) *q*). *paid* = []
⟨*proof*⟩

lemma *COMB_pairwise*: *pairwise* (*COMB* [])
⟨*proof*⟩

18.4 COMB 1.6-competitive

lemma *finite_config_TS*: *finite* (*set_pmf* (*config''* (*embed* (*rTS h*)) *qs init n*)) (**is** *finite ?D*)
⟨*proof*⟩

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*⟩

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*⟩

theorem *COMB_competitive_nice*: *compet_rand* (*COMB* []) ((*8*::*nat*)/(*5*::*nat*))
{*x*::*nat list*. *distinct x* ∧ *x*≠[]}
⟨*proof*⟩

end

References

[BEY98] Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[HN16] Maximilian P.L. Haslbeck and Tobias Nipkow. Verified analysis of list update algorithms. http://www.in.tum.de/~nipkow/pubs/list_update.pdf, 2016.