PiCore: A Rely-guarantee Framework for Event-based Systems

Yongwang Zhao
School of Computer Science and Engineering, Beihang University, China
zhaoyongwang@gmail.com, zhaoyw@buaa.edu.cn

March 18, 2019

Contents

theory *Heap*
imports *Main*
begin

0.1 References

definition *ref* = (*UNIV*::*nat set*)

typedef *ref* = *ref* **by** (*simp add*: *ref-def*)

code-datatype *Abs-ref*

lemma *finite-nat-ex-max*:
  assumes *fin*: *finite* (*N*::*nat set*)
  shows $\exists\, m.\ \forall\, n{\in}N.\ n < m$
using *fin*
proof (*induct*)
  case *empty*
  show *?case* **by** *auto*
next
  case (*insert k N*)
  have $\exists\, m.\ \forall\, n{\in}N.\ n < m$ **by** *fact*

```
  then obtain m where m-max: ∀ n∈N. n < m..
  show ∃ m. ∀ n∈insert k N. n < m
  proof (rule exI [where x=Suc (max k m)])
  qed (insert m-max, auto simp add: max-def)
qed

lemma infinite-nat: ¬finite (UNIV::nat set)
proof
  assume fin: finite (UNIV::nat set)
  then obtain m::nat where ∀ n∈UNIV. n < m
    by (rule finite-nat-ex-max [elim-format] ) auto
  moreover have m∈UNIV..
  ultimately show False by blast
qed

lemma infinite-ref [simp,intro]: ¬finite (UNIV::ref set)
proof
  assume finite (UNIV::ref set)
  hence finite (range Rep-ref)
    by simp
  moreover
  have range Rep-ref = ref
  proof
    show range Rep-ref ⊆ ref
      by (simp add: ref-def)
  next
    show ref ⊆ range Rep-ref
    proof
      fix x
      assume x: x ∈ ref
      show x ∈ range Rep-ref
        by (rule Rep-ref-induct) (auto simp add: ref-def)
    qed
  qed
  ultimately have finite ref
    by simp
  thus False
    by (simp add: ref-def infinite-nat)
qed

consts Null :: ref

definition new :: ref set ⇒ ref where
  new A = (SOME a. a ∉ {Null} ∪ A)
```

Constant *Null* can be defined later on. Conceptually *Null* and *new* are *fixes* of a locale with $finite\ A \Longrightarrow new\ A \notin A \cup \{Null\}$. But since definitions relative to a locale do not yet work in Isabelle2005 we use this workaround to avoid lots of parameters in definitions.

lemma *new-notin* [*simp*,*intro*]:
finite A $\Longrightarrow$ *new* (*A*) $\notin$ *A*
apply (*unfold new-def*)
apply (*rule someI2-ex*)
apply (*fastforce intro*: *ex-new-if-finite*)
apply *simp*
done

lemma *new-not-Null* [*simp*,*intro*]:
finite A $\Longrightarrow$ *new* (*A*) $\neq$ *Null*
apply (*unfold new-def*)
apply (*rule someI2-ex*)
apply (*fastforce intro*: *ex-new-if-finite*)
apply *simp*
done

end
theory *aux-lemma*
imports *Main*
begin

lemma *mod-div-self*: (*a*::*nat*) *mod b* = *0* $\Longrightarrow$ (*a div b*) $*$ *b* = *a*
by *auto*

lemma *mod-div-mult*: (*a*::*nat*) *mod b* = *0* $\Longrightarrow$ *a div b* $\leq$ (*c* $-$ *1*) $\Longrightarrow$ *a* $\leq$ *c* $*$ *b* $-$ *b*
apply(*subgoal-tac a* $\leq$ (*c* $-$ *1*) $*$ *b*)
apply (*simp add*: *left-diff-distrib'*)
by *fastforce*

lemma *mod0-div-self*: (*a*::*nat*) *mod b* = *0* $\Longrightarrow$ *b* $*$ (*a div b*) = *a* **by** *auto*

lemma *m-mod-div*: *n mod x* = *0* $\Longrightarrow$ (*m*::*nat*) $*$ *n div x* = *m* $*$ (*n div x*)
by *auto*

lemma *pow-mod-0*: *x* $\geq$ *y* $\Longrightarrow$ (*m*::*nat*) ^ *x mod m* ^ *y* = *0*
by (*simp add*: *le-imp-power-dvd*)

lemma *ge-pow-mod-0*: (*x*::*nat*) > *y* $\Longrightarrow$ *4* $*$ *n* $*$ (*4*::*nat*) ^ *x mod 4* ^ *y* = *0*
by (*metis less-imp-le-nat mod-mod-trivial mod-mult-right-eq mult-0-right pow-mod-0*)

lemma *div2-eq-minus*: *x* $\neq$ *0* $\wedge$ *m* $\geq$ *n* $\Longrightarrow$ (*x*::*nat*) ^ *m div x* ^ *n* = *x* ^ (*m* $-$ *n*)
by (*metis add-diff-cancel-left' div-mult-self1-is-m gr0I le-Suc-ex power-add power-not-zero*)

lemma *pow-lt-mod0*: (*n*::*nat*) > *0* $\wedge$ (*x*::*nat*) > *y* $\Longrightarrow$ (*n* ^ *x div n* ^ *y*) *mod n* = *0*
by (*simp add*: *div2-eq-minus*)

lemma *mod-div-gt*:
$(m::nat) < n \Longrightarrow n \ mod \ x = 0 \Longrightarrow m \ div \ x < n \ div \ x$
by (*simp add*: *less-mult-imp-div-less mod-div-self*)

lemma *div2-eq-divmul*: $(a::nat) \ div \ b \ div \ c = a \ div \ (b * c)$
by (*simp add*: *Divides.div-mult2-eq*)

lemma *addr-in-div*:
$(addr::nat) \in \{j2 * M \ ..< (Suc \ j2) * M\} \Longrightarrow addr \ div \ M = j2$
by (*simp add*: *div-nat-eqI mult.commute*)

lemma *divn-mult-n*: $x > 0 \Longrightarrow (n::nat) = m \ div \ x * x \Longrightarrow (if \ m \ mod \ x = 0 \ then$
$m = n \ else \ n < m \wedge m < n + x \wedge n \ mod \ x = 0)$
apply *auto*
apply (*metis div-mult-mod-eq less-add-same-cancel1*)
by (*metis add-le-cancel-left div-mult-mod-eq mod-less-divisor not-less*)

lemma *mod-minus-0*:
$(m::nat) \leq n \wedge 0 < m \Longrightarrow a * (4::nat) \ \hat{} \ n \ mod \ 4 \ \hat{} \ (n - m) = 0$
by (*metis diff-le-self mod-mult-right-eq mod-mult-self2-is-0 mult-0 mult-0-right pow-mod-0*)

lemma *mod-minus-div-4*:
$(m::nat) \leq n \wedge 0 < m \Longrightarrow a * (4::nat) \ \hat{} \ n \ div \ 4 \ \hat{} \ (n - m) \ mod \ 4 = 0$
by (*metis add.left-neutral add-lessD1 diff-less m-mod-div mod-0 mod-mult-right-eq*

mult-0-right nat-less-le pow-lt-mod0 pow-mod-0 zero-less-numeral)

lemma *modn0-xy-n*: $(n::nat) > 0 \Longrightarrow x \ mod \ n = 0 \Longrightarrow y \ mod \ n = 0 \Longrightarrow x < y$
$\Longrightarrow x + n \leq y$
by (*metis Nat.le-diff-conv2 add.commute add.left-neutral add-diff-cancel-left'*
le-less less-imp-add-positive mod-add-left-eq mod-less not-less)

lemma *divn-multn-addn-le*: $(n::nat) > 0 \Longrightarrow y \ mod \ n = 0 \Longrightarrow x < y \Longrightarrow x \ div$
$n * n + n \leq y$
using *divn-mult-n*[*of n x div n * n x*] *modn0-xy-n*
apply(*case-tac x mod n = 0*)
apply(*rule subst*[**where** *s=x* **and** *t=x div n * n*]) **apply** *metis*
by *auto*

lemma *div-in-suc*: $y > 0 \Longrightarrow n = (x::nat) \ div \ y \Longrightarrow x \in \{n * y \ ..< Suc \ n * y\}$
by (*simp add*: *dividend-less-div-times*)

lemma *int1-eq*:$P \cap \{V\} \neq \{\} \Longrightarrow P \cap \{V\} = \{V\}$ **by** *auto*

lemma *int1-belong*: $P \cap \{V\} = \{V\} \Longrightarrow V \in P$ **by** *auto*

lemma *two-int-one*: $P \cap \{V\} \cap \{Va\} \neq \{\} \Longrightarrow V = Va \wedge \{V\} = P \cap \{V\} \cap$

$\{Va\}$ **by** *auto*

end
theory *List-aux*
imports *aux-lemma*
begin

primrec *list-updates* :: *'a list* ⇒ *nat* ⇒ *nat* ⇒ *'a* ⇒ *'a list* **where**
  list-updates [] *i1 i2 v* = [] |
  list-updates (*x#xs*) *i1 i2 v* =
    (*case i1 of 0* ⇒ (*if i2* > *0 then v* # *list-updates xs 0* (*i2* − *1*) *v else* (*v#xs*))
|
        Suc j ⇒ (*if i2* > *j then x* # *list-updates xs j* (*i2* − *1*) *v else* (*x#xs*)))

value *list-updates* [*1*::*nat*,*2*,*3*,*4*,*5*] *9 0 6*

lemma *length-list-update2* [*simp*]: *length* (*list-updates l i1 i2 v*) = *length l*
  apply(*induct l arbitrary*: *i1 i2 v*)
    apply *simp*
    apply(*case-tac i1*)
      apply(*case-tac i2*) **apply** *simp+*
  done

lemma *list-updates-eq* [*simp*]: ⟦*i1* ≤ *i*; *i* ≤ *i2*; *i2* < *length l*⟧ ⟹ (*list-updates l i1 i2 v*)!*i* = *v*
  apply(*induct l arbitrary*: *i i1 i2 v*)
    apply *simp*
    apply(*case-tac i1*) **apply** *auto*
      apply(*case-tac i2*) **apply** *simp*
    by (*metis* (*no-types*, *lifting*) *One-nat-def Suc-less-SucD diff-Suc-1*
        le-SucE le-zero-eq not-less-eq-eq nth-Cons' zero-induct)

lemma *list-updates-neq* [*simp*]: *i* < *i1* ∨ *i* > *i2* ⟹ (*list-updates l i1 i2 v*)!*i* = *l*!*i*
  apply(*induct l arbitrary*: *i i1 i2 v*)
    apply *simp*
    apply(*case-tac i1*) **apply** *simp*
    apply(*case-tac i2*) **apply** *simp* **apply**(*case-tac i*) **apply** *simp+*
  done

lemma *list-updates-beyond*[*simp*]: *i1* ≥ *length l* ⟹ (*list-updates l i1 i2 v*) = *l*
  apply(*induct l arbitrary*: *i1 i2 v*)
    apply *simp* **apply**(*case-tac i1*) **by** *auto*

lemma *list-updates-beyond2*[*simp*]: *i2* < *i1* ⟹ (*list-updates l i1 i2 v*) = *l*
  apply(*induct l arbitrary*: *i1 i2 v*)
    apply *simp* **apply**(*case-tac i1*) **by** *auto*

lemma *list-updates-nonempty*[*simp*]: (*list-updates l i1 i2 v*) = [] ⟷ *l* = []

by (*metis length-greater-0-conv length-list-update2*)

lemma *list-updates-same-conv*:
$i1 < length\ l \wedge i2 < length\ l \Longrightarrow ((list\text{-}updates\ l\ i1\ i2\ v) = l) = (\forall\, i.\ i \geq i1 \wedge i \leq i2 \longrightarrow l\ !\ i = v)$
apply(*induct l arbitrary*: *i1 i2 v*)
apply *simp*
apply(*case-tac i1* $\leq$ *i2*) **apply**(*rule iffI*)
apply (*metis list-updates-eq*)
apply (*smt length-list-update2 list-updates-eq list-updates-neq not-le-imp-less nth-equalityI*)
by (*metis* (*mono-tags*, *lifting*) *list-updates-beyond2 list-updates-eq not-le-imp-less*)

lemma *list-updates-append1*:
$i2 < length\ l \Longrightarrow list\text{-}updates\ (l\ @\ t)\ i1\ i2\ v = list\text{-}updates\ l\ i1\ i2\ v\ @\ t$
apply(*induct l arbitrary*: *i1 i2 v*)
apply *simp*
apply(*case-tac i1* $\leq$ *i2*)
apply(*case-tac i1*) **apply** *simp*
apply(*case-tac i2*) **apply** *simp* **apply** *auto*[*1*]
by (*metis list-updates-beyond2 not-less*)

primrec *list-updates-fstn* :: *'a list* ⇒ *nat* ⇒ *'a* ⇒ *'a list* **where**
list-updates-fstn [] *n v* = [] |
list-updates-fstn (*x#xs*) *n v* =
(*case n of 0* ⇒ *x#xs* | *Suc m* ⇒ *v#list-updates-fstn xs m v*)

primrec *list-updates-n* :: *'a list* ⇒ *nat* ⇒ *nat* ⇒ *'a* ⇒ *'a list* **where**
list-updates-n [] *i n v* = [] |
list-updates-n (*x#xs*) *i n v* =
(*case i of 0* ⇒ *list-updates-fstn* (*x#xs*) *n v* | *Suc j* ⇒ *x#list-updates-n xs j n v*)

value *list-updates-n* [*1*::*nat*,*2*,*3*,*4*,*5*] *0 9 6*

lemma *length-list-update-fstn* [*simp*]: *length* (*list-updates-fstn l n v*) = *length l*
apply(*induct l arbitrary*: *n v*)
apply *simp* **apply**(*case-tac n*) **apply** *simp+*
done

lemma *length-list-update-n* [*simp*]: *length* (*list-updates-n l i n v*) = *length l*
apply(*induct l arbitrary*: *i n v*)
apply *simp*
apply(*case-tac i*)
apply(*case-tac n*) **apply** *simp+*
done

lemma *list-updates-fstn-eq* [*simp*]: $[\![i < length\ l;\ i < n]\!] \Longrightarrow (list\text{-}updates\text{-}fstn\ l\ n\ v)!i = v$
  apply(*induct l arbitrary*: *i n v*) **apply** *simp*
    apply(*case-tac i*)
    apply(*case-tac n*) **apply** *simp+*
    apply(*case-tac n*) **apply** *simp+*
done

lemma *list-updates-n-eq* [*simp*]: $[\![i \leq j;\ j < length\ l;\ j < i + n]\!] \Longrightarrow (list\text{-}updates\text{-}n\ l\ i\ n\ v)!j = v$
  apply(*induct l arbitrary*: *i j n v*) **apply** *simp*
    apply(*case-tac i*) **apply** *auto*
    apply(*case-tac n*) **apply** *auto*
  using *less-Suc-eq-0-disj* **by** *auto*

lemma *list-updates-fst0* [*simp*]: *list-updates-fstn l 0 v* = *l*
  apply(*induct l arbitrary*: *v*) **by** *simp+*

lemma *list-updates-0* [*simp*]: *list-updates-n l i 0 v* = *l*
  apply(*induct l arbitrary*: *i v*) **apply** *simp* **apply**(*case-tac i*) **apply** *simp+*
done

lemma *list-updates-fstn-neq* [*simp*]: $j \geq n \Longrightarrow (list\text{-}updates\text{-}fstn\ l\ n\ v)!j = l!j$
  apply(*induct l arbitrary*: *j n v*) **apply** *simp*
  apply(*case-tac n*) **apply** *simp+*
done

lemma *list-updates-n-neq* [*simp*]: $j < i \vee j \geq i + n \Longrightarrow (list\text{-}updates\text{-}n\ l\ i\ n\ v)!j = l!j$
  apply(*induct l arbitrary*: *i j n v*) **apply** *simp*
    apply(*case-tac i*) **apply**(*case-tac n*) **apply** *simp+*
    apply(*case-tac n*) **apply** *simp* **apply**(*case-tac j*) **apply** *simp* **apply** *auto*
done

lemma *list-updates-n-beyond*[*simp*]: $i \geq length\ l \Longrightarrow (list\text{-}updates\text{-}n\ l\ i\ n\ v) = l$
  apply(*induct l arbitrary*: *i n v*)
    apply *simp* **apply**(*case-tac i*) **by** *auto*

lemma *lst-udptn-set-eq*: $n > 0 \Longrightarrow list\text{-}updates\text{-}n\ (lst[jj := TAG])\ (jj\ div\ n * n)\ n\ TAG1 =$
    $list\text{-}updates\text{-}n\ lst\ (jj\ div\ n * n)\ n\ TAG1$
apply(*rule nth-equalityI*) **apply** *simp*
apply *clarify*
apply(*case-tac i = jj*)
  apply(*subgoal-tac* $i \geq jj\ div\ n * n$) **prefer** *2* **apply** (*metis divn-mult-n less-or-eq-imp-le*)
  apply(*subgoal-tac* $i < jj\ div\ n * n + n$) **prefer** *2*
  apply (*metis* (*no-types*) *add.commute dividend-less-div-times*)
  apply *simp*

by (*metis length-list-update length-list-update-n list-updates-n-eq list-updates-n-neq not-less nth-list-update-neq*)

thm *list-updates-n.simps*
lemma *list-updates-n-simps2*: *list-updates-n* $(a\#lst)$ $(Suc\ ii)$ $m\ v = a\ \#$ *list-updates-n lst ii m v*
by *fastforce*

lemma *list-updates-n-simps2'*: $ii > 0 \Longrightarrow$ *list-updates-n* $(a\#lst)$ $ii\ m\ v = a\ \#$ *list-updates-n lst* $(ii - 1)$ $m\ v$
using *list-updates-n-simps2*[*of a lst ii* $-$ *1 m v*] **by** *force*

lemma *lst-updt1-eq-upd*: *list-updates-n lst ii 1 v* $=$ *lst*[*ii* := *v*]
apply(*induct lst arbitrary*: *ii*) **apply** *simp*
apply(*case-tac ii* $= 0$) **apply** *simp*
using *list-updates-n-simps2'*
by (*metis One-nat-def Suc-pred list-update-code*(*3*) *neq0-conv*)

lemma *list-neq-udpt-neq*:
$\forall\, i<length\ l.\ l\ !\ i \neq P \Longrightarrow$
$l' =$ *list-updates-n l s n Q* $\Longrightarrow$
$P \neq Q \Longrightarrow$
$\forall\, i<length\ l'.\ l'\ !\ i \neq P$
apply(*induct* l' *arbitrary*:*l*) **apply** *simp*
by (*metis le-neq-implies-less length-list-update-n list-updates-n-eq list-updates-n-neq nat-le-linear*)

lemma *lst-updts-eq-updts-updt*:
$1 \leq ii \Longrightarrow$
list-updates-n lst st $(ii - 1)$ *TAG* [$st + ii - 1$:= *TAG*] $=$
list-updates-n lst st ii TAG
apply(*rule nth-equalityI*)
apply *simp*
apply *clarsimp* **apply**(*rename-tac ia*)
apply(*case-tac ia* $<$ *st*) **using** *list-updates-n-neq* **apply** *simp*
apply(*case-tac ia* $\geq st + ii$) **using** *list-updates-n-neq* **apply** *simp*
apply(*case-tac ia* $< st + ii - 1$) **using** *list-updates-n-eq* **apply** *simp*
apply(*subgoal-tac ia* $= st + ii - 1$) **prefer** *2*
apply *force*
apply(*subgoal-tac length lst* $=$ *length* (*list-updates-n lst st ii TAG*))
prefer *2* **apply** *simp*
apply(*subgoal-tac length lst* $=$ *length* (*list-updates-n lst st* $(ii - 1)$ *TAG*))
prefer *2* **using** *length-list-update-n* **apply** *metis*
apply(*case-tac ia* $\geq$ *length lst*) **apply** *linarith*
apply(*subgoal-tac list-updates-n lst st* $(ii - 1)$ *TAG* [$st + ii - 1$:= *TAG*] ! *ia* $=$ *TAG*) **prefer** *2*

apply (*metis nth-list-update-eq*)
apply(*subgoal-tac list-updates-n lst st ii TAG* ! *ia* = *TAG*) **prefer** *2*
apply (*meson list-updates-n-eq not-less*)
using *One-nat-def* **by** *presburger*

primrec *removes* :: *'a list* ⇒ *'a list* ⇒ *'a list*
where *removes* [] *l* = *l* |
removes (*x*#*xs*) *l* = *removes xs* (*remove1 x l*)

lemma *removes-distinct* [*simp*]: *distinct l* ⟹ *distinct* (*removes rs l*)
apply(*induct rs arbitrary*:*l*) **by** *auto*

lemma *removes-length* [*simp*]: ⟦*set rs* ⊆ *set l*; *distinct l*; *distinct rs* ⟧
⟹ *length rs* + *length* (*removes rs l*) = *length l*
apply(*induct rs arbitrary*:*l*)
apply *simp* **apply** *auto*
by (*metis* (*no-types*, *lifting*) *One-nat-def Suc-pred distinct-remove1*
in-set-remove1 length-pos-if-in-set length-remove1 subset-eq)

lemma *removes-empty* [*simp*]: *removes rs* [] = []
apply(*induct rs*) **by** *simp*+

lemma *removes-subs1* [*simp*]: *set* (*removes rs l*) ⊆ *set l*
apply(*induct rs arbitrary*: *l*) **apply** *simp* **apply** *simp*
apply(*subgoal-tac set* (*remove1 a l*) ⊆ *set l*) **apply** *auto*[*1*]
by (*simp add*: *set-remove1-subset*)

lemma *removes-subs2* [*simp*]: *distinct l* ⟹ *set* (*removes* (*a*#*rs*) *l*) ⊆ *set* (*removes rs l*)
apply *simp*
apply(*induct rs arbitrary*: *l a*)
apply *auto* **by** (*metis* (*full-types*) *distinct-remove1 remove1-commute set-mp*)

lemma *removes-nin* [*simp*]: ⟦*x* ∈ *set rs*; *distinct l*⟧ ⟹ *x* ∉ *set* (*removes rs l*)
apply(*induct rs arbitrary*:*l x*)
apply *simp*
apply *simp* **apply** *auto*
by (*metis DiffE contra-subsetD removes-subs1 set-remove1-eq singletonI*)

lemma *rmvs-empty*: *a* ∈ *set es* ⟹ *removes es* [*a*] = []
apply(*induct es*) **apply** *simp* **apply** *auto*
done

lemma *rmvs-unchg*: *a* ∉ *set es* ⟹ *removes es* [*a*] = [*a*]
apply(*induct es*) **apply** *simp* **apply** *auto*
done

lemma *rmvs-onemore-same*:
distinct lst $\Longrightarrow$ *e* $\notin$ *set lst* $\Longrightarrow$ *removes* (*es*@[*e*]) *lst* = *removes es lst*
apply(*induct es arbitrary*:*lst*)
apply (*simp add*: *remove1-idem*)
apply *auto*
done

lemma *rmvs-rev*: *removes* (*es*@[*e*]) *lst* = *remove1 e* (*removes es lst*)
apply(*induct es arbitrary*:*lst*) **apply** *simp* **apply** *auto*
done

definition *inserts xs l* $\equiv$ *l* @ *xs*

lemma *inserts-set-un*: *set* (*inserts xs l*) = *set xs* $\cup$ *set l*
by (*simp add*: *inserts-def sup-commute*)

lemma *inserts-emp1*: *set* (*inserts xs* []) = *set xs*
using *inserts-set-un*[*of xs* []] **by** *auto*

lemma *inserts-emp2*: *set* (*inserts* [] *l*) = *set l*
using *inserts-set-un*[*of* [] *l*] **by** *auto*

lemma *list-updt-samelen*: *length l* = *length* (*l*[*jj* := *a*]) **by** *simp*

lemma *list-nhd-in-tl-set*: *el* $\in$ *set l* $\Longrightarrow$ *el* $\neq$ *hd l* $\Longrightarrow$ *el* $\in$ *set* (*tl l*)
by (*metis empty-iff empty-set list.exhaust-sel set-ConsD*)

lemma *dist-hd-nin-tl*: *distinct l* $\Longrightarrow$ *a*$\in$*set* (*tl l*) $\Longrightarrow$ *a* $\neq$ *hd l*
by (*metis distinct.simps*(*2*) *equals0D list.collapse set-empty tl-Nil*)

end

theory *mem-spec*
imports *Main Heap PiCore*−*SIMP.picore-SIMP-lemma List-aux*
begin

1 data types and state

typedecl *Thread*

typedef *mempool-ref* = *ref* **by** (*simp add*: *ref-def*)

we define memory address as nat

type-synonym *mem-ref* = *nat*

abbreviation *NULL* $\equiv$ *0* :: *nat* we have a thread scheduler, thread has 3 types. BLOCKED means a thread is waiting for memory and is in wait queue

datatype *Thread-State-Type* = *READY* | *RUNNING* | *BLOCKED*

a memory block: a ref to a memor pool, a level index and a block index in this level, a start address "data". max number of levels is n_level of a memory pool. So @level should be ¡ n_levels. The number of blocks at level 0 is n_max. the max number of blocks at level n is $n\_max * 4^n$. the block index should less then this number.

record *Mem-block* = *pool* :: *mempool-ref*
level :: *nat*
block :: *nat*
data :: *mem-ref*

BlockState defines the bit info in bitmap. We uses different types, while not 0 or 1 in this design. Then the blockstate could be implemented as 0 or 1, with additional information.

basic states of memory block are ALLOCATED, FREE, DIVIDED and NOEXIST. The levels of bitmap is actually a quad-tree of BlockState. ALLOCATED: the block is allocated to a thread FREE: the block is free DIVIDED: the block is divided, which means is was splited to 4 subblocks NOEXIST: the block is not exist

ALLOCACTED and FREE blocks are the leaf blocks of the quad-tree. DIVIDED blocks are inner nodes of the quad-tree. Otherwise is NOEXIST.

we also introduce FREEING and ALLOCATING state to avoid a case that a FREEING block may be allocated by other threads and a ALLOCATING block may be freed by other threads. In OS implementation, the allocating/freeing block is an inner block of alloc/free services, and other threads will not manipulate them. they are used to indicate state of the block which are going to be merged during freeing a block, and the block which is going to be split during allocating a block.

we may remove FREEING/ALLOCATING state later by revising alloc and free syscalls to avoid allocate or free blocks in freeing_node and allocating_node.

datatype *BlockState* = *ALLOCATED* | *FREE* | *DIVIDED* | *NOEXIST* | *FREEING* | *ALLOCATING*

data stucture at each level, a bitmap and a free block list

record *Mem-pool-lvl* =
bits :: *BlockState list*
free-list :: *mem-ref list*

a memory pool is actually a forest of @n_max numbers of blocks with size of @max_sz. A block may be split to 4 sub-blocks and so on, at most for

@n_levels times. Thus, each block may be split as a quad-tree. a memory pool maintains a big memory block, where @buf is the start address of the memory block. The size of a memory pool is @n_max * @max_sz. @max_sz has a constraint. a small block at last level (level index is @n_levels - 1) should be aligned by 4 bits, i.e. the size of block at last level should be 4*n (n ¿ 0). Here, we dont demand 4^n, which is a special case of 4*n. Thus, @max_sz should be $4 * n * 4^n\_levels$.

@levels maintain the information at each level including a bitmap and a free block list. @wait_q is a list of threads, which is blocked on this memory pool.

record *Mem-pool* = *buf* :: *mem-ref*
max-sz :: *nat*
n-max :: *nat*
n-levels :: *nat*

levels :: *Mem-pool-lvl list*
wait-q :: *Thread list*

The state of memory management consists of thread state, memory pools, and local variables of each thread. In monocore OSs, there is only one currently executing thread @cur, where None means the scheduler has not choose a thread. @tick save a time for the system. @mem_pools maintains the refs of all memory pools. @mem_pool_info shows the detailed information of each memory pool by its ref. we assume that all memory pools are shared by all threads. This is the most relaxed case. The case that some memory pool is only shared by a set of thread is just a special case. Other fields are local vars of each thread used in alloc/free syscalls.

for each thread, we use freeing_node to maintain the freeing node in free syscall. when free a block, we set it to FREEING, and check if its other 3 partner blocks are also free. If so, we set the 4 blocks to NOEXIST and set their parent block to FREEING, and so on. until that other 3 partner blocks are not all free, then set the FREEING block to FREE. This design avoids the FREEING node is allocated by other threads.

we use allocating_node to maintain the allocating node in alloc syscall. when alloc a block, we find a free block at the nearest upper level, and set it to ALLOCATING. if size of the block is too big, we split it into 4 child blocks. We set the first child block to ALLOCATING and other 3 blocks to FREE, and so on. until that the size of block is suitable, then set the ALLOCATING block to ALLOCATED. This design avoids the ALLOCATING node is freed by other threads.

record *State* =

cur :: *Thread option*
tick :: *nat*

$thd\text{-}state :: Thread \Rightarrow Thread\text{-}State\text{-}Type$

$mem\text{-}pools :: mempool\text{-}ref\ set$

$mem\text{-}pool\text{-}info :: mempool\text{-}ref \Rightarrow Mem\text{-}pool$

$i :: Thread \Rightarrow nat$
$j :: Thread \Rightarrow nat$
$ret :: Thread \Rightarrow int$
$endt :: Thread \Rightarrow nat$
$rf :: Thread \Rightarrow bool$
$tmout :: Thread \Rightarrow int$
$lsizes :: Thread \Rightarrow nat\ list$
$alloc\text{-}l :: Thread \Rightarrow int$
$free\text{-}l :: Thread \Rightarrow int$
$from\text{-}l :: Thread \Rightarrow int$
$blk :: Thread \Rightarrow mem\text{-}ref$
$nodev :: Thread \Rightarrow mem\text{-}ref$
$bn :: Thread \Rightarrow nat$
$lbn :: Thread \Rightarrow nat$
$lsz :: Thread \Rightarrow nat$
$block2 :: Thread \Rightarrow mem\text{-}ref$
$free\text{-}block\text{-}r :: Thread \Rightarrow bool$
$alloc\text{-}lsize\text{-}r :: Thread \Rightarrow bool$
$lvl :: Thread \Rightarrow nat$
$bb :: Thread \Rightarrow nat$
$block\text{-}pt :: Thread \Rightarrow mem\text{-}ref$
$th :: Thread \Rightarrow Thread$
$need\text{-}resched :: Thread \Rightarrow bool$
$mempoolalloc\text{-}ret :: Thread \Rightarrow Mem\text{-}block\ option$

$freeing\text{-}node :: Thread \Rightarrow Mem\text{-}block\ option$
$allocating\text{-}node :: Thread \Rightarrow Mem\text{-}block\ option$

2 specification of events

2.1 data types

Since Zephyr uses fine-grained locks for shared memory pools, interleaving among scheduling, syscalls (alloc, free), and clock tick are allowed. Thus, we use 3 event systems to model scheduling, syscalls from threads, and clock tick. Then the whole system is the parallel composition of the three event systems. Actually, we have 1 scheduler, 1 timer, and n threads.

datatype $Core = \mathcal{S} \mid \mathcal{T}\ Thread \mid Timer$

labels for different events

datatype $EL = ScheduleE \mid TickE \mid Mem\text{-}pool\text{-}allocE \mid Mem\text{-}pool\text{-}freeE \mid Mem\text{-}pool\text{-}defineE$

data types for event parameters

datatype $Parameter = Thread\ Thread \mid MPRef\ mempool\text{-}ref \mid MRef\ mem\text{-}ref \mid Block\ Mem\text{-}block \mid Natural\ nat \mid Integer\ int$

type-synonym $EventLabel = EL \times (Parameter\ list \times Core)$

definition $get\text{-}evt\text{-}label :: EL \Rightarrow Parameter\ list \Rightarrow Core \Rightarrow EventLabel$ ($- - \Rrightarrow -$ $[30,30,30]\ 20$)
  where $get\text{-}evt\text{-}label\ el\ ps\ k \equiv (el,(ps,k))$

define the waiting mode for alloc. FOREVER means that if allocating fails, the thread will wait forever until allocating succeed. NOWAIT means that if allocating fails, alloc syscall return error immediately. otherwise n ¿ 0, means the thread will wait for a timeout n.

abbreviation $FOREVER \equiv (-1)::int$
abbreviation $NOWAIT \equiv 0::int$

return CODE for alloc and free syscalls. free syscall always succeed, so it returns OK. alloc syscall may succeed (OK), timeout (ETIMEOUT), fails(ENOMEM), fails due to request too large size (ESIZEERR).

EAGAIN is an inner flag of alloc syscall. After it finds an available block for request, the block may be allocated immediately by other threads. In such a case, alloc will provide EAGAIN and try to allocate again.

We introduce ESIZEERR for Zephyr to avoid a dead loop. We introduce ETIMEOUT for Zephyr for robustness.

abbreviation $EAGAIN \equiv (-2)::int$
abbreviation $ENOMEM \equiv (-3)::int$
abbreviation $ESIZEERR \equiv (-4)::int$
abbreviation $OK \equiv 0 :: int$
abbreviation $ETIMEOUT \equiv (-1) :: int$

due to fine-grained lock used by Zephyr, we use a command for each atomic statement in free/alloc syscalls. the statements of syscalls from a thread t can only be executed when t is the currently executing thread by the scheduler. We use the AWAIT statement to represent this semantics.

definition $stm :: Thread \Rightarrow State\ com \Rightarrow State\ com$ ($- \blacktriangleright -$ $[0,0]\ 21$)
where $stm\ t\ p = AWAIT\ {}^\prime cur = Some\ t\ THEN\ p\ END$

2.2 aux definitions for events

definition $ALIGN4 :: nat \Rightarrow nat$
where $ALIGN4\ n \equiv ((n + 3)\ div\ 4) * 4$

lemma *align40*: *n mod 4* = *0* ⟹ *ALIGN4 n* = *n*
  unfolding *ALIGN4-def* **by** *auto*

lemma *align41*: *n mod 4* = *1* ⟹ *ALIGN4 n* = *n* + *3*
  unfolding *ALIGN4-def*
proof −
  assume *n mod 4* = *1*
  then have (*n* + *3*) *mod 4* = *0*
    by *presburger*
  then show (*n* + *3*) *div 4* ∗ *4* = *n* + *3*
    by *fastforce*
qed

lemma *align42*: *n mod 4* = *2* ⟹ *ALIGN4 n* = *n* + *2*
  unfolding *ALIGN4-def*
proof −
  assume *n mod 4* = *2*
  then have (*n* + *2*) *mod 4* = *0*
    using *mod-add-left-eq* **by** *presburger*
  then show (*n* + *3*) *div 4* ∗ *4* = *n* + *2*
    by *fastforce*
qed

lemma *align43*: *n mod 4* = *3* ⟹ *ALIGN4 n* = *n* + *1*
  unfolding *ALIGN4-def*
proof −
  assume *n mod 4* = *3*
  then have (*n* + *1*) *mod 4* = *0*
    using *mod-add-left-eq* **by** *presburger*
  then show (*n* + *3*) *div 4* ∗ *4* = *n* + *1*
    by *fastforce*
qed

lemma *align-mod0*: *ALIGN4 n mod 4* = *0*
  unfolding *ALIGN4-def* **by** *simp*

lemma *align4-gt*: *ALIGN4 n* ≥ *n* ∧ *ALIGN4 n* ≤ *n* + *3*
  apply(*case-tac n mod 4* = *0*)
    using *align40* **apply** *simp*
  apply(*case-tac n mod 4* = *1*)
    using *align41* **apply** *simp*
  apply(*case-tac n mod 4* = *2*)
    using *align42* **apply** *simp*
  apply(*case-tac n mod 4* = *3*)
    using *align43* **apply** *simp*
  by *auto*

lemma *align2-eq-align*: *ALIGN4* (*ALIGN4 n*) = *ALIGN4 n*
  unfolding *ALIGN4-def* **by** *auto*

Zephyr uses two events: reschedule for free and swap for alloc for context switch

definition *reschedule* :: *State com*
where *reschedule* ≡
  ´thd-state := *´thd-state*(*the ´cur* := *READY*);;
  ´cur := *Some* (*SOME t. ´thd-state t* = *READY*);;
  ´thd-state := *´thd-state*(*the ´cur* := *RUNNING*)

definition *swap* :: *State com*
where *swap* ≡
  IF (∃ *t. ´thd-state t* = *READY*) *THEN*
    ´cur := *Some* (*SOME t. ´thd-state t* = *READY*);;
    ´thd-state := *´thd-state*(*the ´cur* := *RUNNING*)
  ELSE
    ´cur := *None*
  FI

definition *block-num* :: *Mem-pool* ⇒ *mem-ref* ⇒ *nat* ⇒ *nat*
where *block-num p bl sz* ≡ (*bl* − (*buf p*)) *div sz*

definition *clear-free-bit* :: (*mempool-ref* ⇒ *Mem-pool*) ⇒ *mempool-ref* ⇒ *nat* ⇒ *nat* ⇒ (*mempool-ref* ⇒ *Mem-pool*)
where *clear-free-bit mp-info p l b* ≡
    mp-info (*p* := (*mp-info p*) (|*levels* := (*levels* (*mp-info p*))
      [*l* := ((*levels* (*mp-info p*)) ! *l*) (|*bits* := (*bits* ((*levels* (*mp-info p*)) ! *l*))
[*b* := *ALLOCATED*]|)] |))

definition *set-bit* :: (*mempool-ref* ⇒ *Mem-pool*) ⇒ *mempool-ref* ⇒ *nat* ⇒ *nat* ⇒ *BlockState* ⇒ (*mempool-ref* ⇒ *Mem-pool*)
where *set-bit mp-info p l b st* ≡
    mp-info (*p* := (*mp-info p*) (|*levels* := (*levels* (*mp-info p*))
      [*l* := ((*levels* (*mp-info p*)) ! *l*) (|*bits* := (*bits* ((*levels* (*mp-info p*)) ! *l*))
[*b* := *st*]|)] |))

abbreviation *set-bit-free mp-info p l b* ≡ *set-bit mp-info p l b FREE*
abbreviation *set-bit-alloc mp-info p l b* ≡ *set-bit mp-info p l b ALLOCATED*
abbreviation *set-bit-divide mp-info p l b* ≡ *set-bit mp-info p l b DIVIDED*
abbreviation *set-bit-noexist mp-info p l b* ≡ *set-bit mp-info p l b NOEXIST*
abbreviation *set-bit-freeing mp-info p l b* ≡ *set-bit mp-info p l b FREEING*
abbreviation *set-bit-allocating mp-info p l b* ≡ *set-bit mp-info p l b ALLOCATING*

definition *set-bit-s* :: *State* ⇒ *mempool-ref* ⇒ *nat* ⇒ *nat* ⇒ *BlockState* ⇒ *State*
where *set-bit-s s p l b st* ≡
    s(|*mem-pool-info* := *set-bit* (*mem-pool-info s*) *p l b st* |)

lemma *set-bit-prev-len*:
length (*bits* (*levels* (*mp-info p*) ! *l*)) = *length* (*bits* (*levels* ((*set-bit mp-info p l b flg*) *p*) ! *l*))
  apply(*simp add*:*set-bit-def*)

using *list-updt-samelen*
by (*metis* (*no-types*, *lifting*) *Mem-pool-lvl.select-convs*(*1*) *Mem-pool-lvl.surjective*

Mem-pool-lvl.update-convs(*1*) *list-update-beyond not-less nth-list-update-eq*)

lemma *set-bit-prev-len2*:
$l \neq t \Longrightarrow$ *length* (*bits* (*levels* (*mp-info p*) ! *l*)) = *length* (*bits* (*levels* ((*set-bit mp-info p t b flg*) *p*) ! *l*))
by(*simp add*:*set-bit-def*)

abbreviation *get-bit* :: (*mempool-ref* ⇒ *Mem-pool*) ⇒ *mempool-ref* ⇒ *nat* ⇒ *nat* ⇒ *BlockState*
where *get-bit mp-info p l b* ≡ (*bits* ((*levels* (*mp-info p*)) ! *l*)) ! *b*

abbreviation *get-bit-s* :: *State* ⇒ *mempool-ref* ⇒ *nat* ⇒ *nat* ⇒ *BlockState*
where *get-bit-s s p l b* ≡ *get-bit* (*mem-pool-info s*) *p l b*

lemma *set-bit-get-bit-eq*:
$l <$ *length* (*levels* (*mp-info p*)) $\Longrightarrow$
$b <$ *length* (*bits* (*levels* (*mp-info p*) ! *l*)) $\Longrightarrow$
mp-info2 = *set-bit mp-info p l b st* $\Longrightarrow$
get-bit mp-info2 p l b = *st*
by (*simp add*:*set-bit-def*)

lemma *set-bit-get-bit-eq2*:
$l <$ *length* (*levels* ((*mem-pool-info Va*) *p*)) $\Longrightarrow$
$b <$ *length* (*bits* (*levels* ((*mem-pool-info Va*) *p*) ! *l*)) $\Longrightarrow$
get-bit-s (*Va*⦇ *mem-pool-info* := *set-bit* (*mem-pool-info Va*) *p l b st*⦈) *p l b* = *st*
using *set-bit-get-bit-eq*
[*of l* (*mem-pool-info Va*) *p b set-bit* (*mem-pool-info Va*) *p l b st st*]
by *simp*

lemma *set-bit-get-bit-neq*:
$p \neq p1 \lor l \neq l1 \lor b \neq b1 \Longrightarrow$
mp-info2 = *set-bit mp-info p l b st* $\Longrightarrow$
get-bit mp-info2 p1 l1 b1 = *get-bit mp-info p1 l1 b1*
apply(*simp add*:*set-bit-def*) **apply** *auto*
by (*metis* (*no-types*, *lifting*) *Mem-pool-lvl.select-convs*(*1*) *Mem-pool-lvl.surjective*

Mem-pool-lvl.update-convs(*1*) *list-update-beyond not-less nth-list-update-eq nth-list-update-neq*)

lemma *set-bit-get-bit-neq2*:
$p \neq p1 \lor l \neq l1 \lor b \neq b1 \Longrightarrow$
get-bit-s (*Va*⦇ *mem-pool-info* := *set-bit* (*mem-pool-info Va*) *p l b st*⦈) *p1 l1 b1*
= *get-bit-s Va p1 l1 b1*
using *set-bit-get-bit-neq*
[*of p p1 l l1 b b1 set-bit* (*mem-pool-info Va*) *p l b st mem-pool-info Va*]

by *simp*

definition *buf-size* :: *Mem-pool* ⇒ *nat*
where *buf-size m* ≡ *n-max m* ∗ *max-sz m*

definition *block-fits* :: *Mem-pool* ⇒ *mem-ref* ⇒ *nat* ⇒ *bool*
where *block-fits p b bsz* ≡ *b* + *bsz* < *buf-size p* + *buf p* + *1*

definition *block-ptr* :: *Mem-pool* ⇒ *nat* ⇒ *nat* ⇒ *mem-ref*
where *block-ptr p lsize b* ≡ *buf p* + *lsize* ∗ *b*

definition *partner-bits* :: *Mem-pool* ⇒ *nat* ⇒ *nat* ⇒ *bool*
where *partner-bits p l b* ≡ *let bits* = *bits* (*levels p* ! *l*);
a = (*b div 4*) ∗ *4 in*
bits!*a* = *FREE* ∧ *bits*!(*a*+*1*) = *FREE* ∧ *bits*!(*a*+*2*) = *FREE* ∧ *bits*!(*a*+*3*) = *FREE*

lemma *partbits-div4*: *a div 4* = *b div 4* ⟹ *partner-bits p l a* = *partner-bits p l b*
by(*simp add*:*partner-bits-def*)

abbreviation *noexist-bits* :: *Mem-pool* ⇒ *nat* ⇒ *nat* ⇒ *bool*
where *noexist-bits mp ii jj* ≡ (*bits* (*levels mp* ! *ii*)) ! *jj* = *NOEXIST*
∧ (*bits* (*levels mp* ! *ii*)) ! (*jj* + *1*) = *NOEXIST*
∧ (*bits* (*levels mp* ! *ii*)) ! (*jj* + *2*) = *NOEXIST*
∧ (*bits* (*levels mp* ! *ii*)) ! (*jj* + *3*) = *NOEXIST*

definition *level-empty* :: *Mem-pool* ⇒ *nat* ⇒ *bool*
where *level-empty p n* ≡ *free-list* (*levels p*!*n*) = []

definition *head-free-list* :: *Mem-pool* ⇒ *nat* ⇒ *mem-ref*
where *head-free-list p l* ≡ *hd* (*free-list* ((*levels p*) ! *l*))

definition *rmhead-free-list* :: *Mem-pool* ⇒ *nat* ⇒ *Mem-pool*
where *rmhead-free-list p l* ≡
p⦇*levels* := (*levels p*)
[*l* := ((*levels p*) ! *l*) ⦇*free-list* := *tl* (*free-list* ((*levels p*) ! *l*))⦈] ⦈

definition *remove-free-list* :: *Mem-pool* ⇒ *nat* ⇒ *mem-ref* ⇒ *Mem-pool*
where *remove-free-list p l b* ≡
p⦇*levels* := (*levels p*)
[*l* := ((*levels p*) ! *l*) ⦇*free-list* := *remove1 b* (*free-list* ((*levels p*) ! *l*))⦈] ⦈

definition *append-free-list* :: *Mem-pool* ⇒ *nat* ⇒ *mem-ref* ⇒ *Mem-pool*
where *append-free-list p l b* ≡
p⦇*levels* := (*levels p*)
[*l* := ((*levels p*) ! *l*) ⦇*free-list* := (*free-list* ((*levels p*) ! *l*)) @ [*b*]⦈] ⦈

definition *in-free-list* :: *mem-ref* ⇒ *mem-ref list* ⇒ *bool*

where *in-free-list v fl* ≡ (∃ *i*<*length fl*. *fl*!*i* = *v*)

2.3 specification of events

lemma *timeout-lm*: (*timeout* = *FOREVER* ∨ *timeout* = *NOWAIT* ∨ *timeout* > *0*) = (*timeout* ≥ −*1*)
by *auto*

definition *Mem-pool-alloc* :: *Thread* ⇒ *mempool-ref* ⇒ *nat* ⇒ *int* ⇒ (*EventLabel*, *Core*, *State*, *State com option*) *event*
where *Mem-pool-alloc t p sz timeout* =
EVENT Mem-pool-allocE [*MPRef p*, *Natural sz*, *Integer timeout*] ⇛ (𝒯 *t*)
WHEN
p ∈ ´*mem-pools*
(∗ ∧ ´*cur* = *Some t*∗) (∗ *t is the current thread* ∗) (∗∗ *this condition is not stable on rely condition* ∗∗)
∧ *timeout* ≥ −*1* (∗ *equv to* (*timeout* = *FOREVER* ∨ *timeout* = *NOWAIT* ∨ *timeout* > *0*) ∗)
(∗ ∧ *p* ∈ ´*pools-of-thread t* ∗) (∗ *the mem pool p is shared in the thread t* ∗)
THEN
(*t* ▸ ´*tmout* := ´*tmout*(*t* := *timeout*));;

(*t* ▸ ´*endt* := ´*endt*(*t* := *0*));;
(*t* ▸ *IF timeout* > *0 THEN*
´*endt* := ´*endt*(*t* := ´*tick* + *nat timeout*)
FI);;
(*t* ▸ ´*mempoolalloc-ret* := ´*mempoolalloc-ret* (*t* := *None*));;
(*t* ▸ ´*ret* := ´*ret*(*t* := *ESIZEERR*));;
(*t* ▸ ´*rf* := ´*rf*(*t* := *False*));;
WHILE ¬ (´*rf t*) *DO*

(∗ ==== *start*: *ret* = *pool-alloc*(*p*, *block*, *size*); ========================== ∗)
(∗(*t* ▸ ´*lsizes* := ´*lsizes*(*t* := []));;∗)
(*t* ▸ ´*blk* := ´*blk*(*t* := *NULL*));;
(*t* ▸ ´*alloc-lsize-r* := ´*alloc-lsize-r*(*t* := *False*));;
(*t* ▸ ´*alloc-l* := ´*alloc-l*(*t* := −*1*));;
(*t* ▸ ´*free-l* := ´*free-l*(*t* := −*1*));;
(*t* ▸ ´*lsizes* := ´*lsizes*(*t* := [*ALIGN4* (*max-sz* (´*mem-pool-info p*))]));;
(*t* ▸ ´*i* := ´*i*(*t* := *0*));;
WHILE ´*i t* < *n-levels* (´*mem-pool-info p*) ∧ ¬ ´*alloc-lsize-r t DO*
IF ´*i t* > *0 THEN*
(*t* ▸ ´*lsizes* := ´*lsizes*(*t* := ´*lsizes t* @ [*ALIGN4* (´*lsizes t* ! (´*i t* − *1*) *div 4*)]))
FI;;
IF ´*lsizes t* ! ´*i t* < *sz THEN*
(*t* ▸ ´*alloc-lsize-r* := ´*alloc-lsize-r*(*t* := *True*))
ELSE
(*t* ▸ ´*alloc-l* := ´*alloc-l*(*t* := *int* (´*i t*)));;

```
      IF ¬ level-empty (´mem-pool-info p) (´i t) THEN
        (t ▶ ´free-l := ´free-l(t := int (´i t)))
      FI;;
      (t ▶ ´i := ´i(t := ´i t + 1))
    FI
  OD;;

  IF ´alloc-l t < 0 THEN
    (t ▶ ´ret := ´ret(t := ESIZEERR))
  ELSE
    IF ´free-l t < 0 THEN
      (∗ block−>data = NULL; ∗)
      (t ▶ ´ret := ´ret(t := ENOMEM))
    ELSE
      (∗ ==== start: blk = alloc-block(p, free-l, lsizes[free-l]); ∗)
      (t ▶ ATOMIC
        (∗ ==== start: block = sys-dlist-get(&p−>levels[l].free-list); ∗)
        IF level-empty (´mem-pool-info p) (nat (´free-l t)) THEN
          ´blk := ´blk(t := NULL)
        ELSE
          ´blk := ´blk(t := head-free-list (´mem-pool-info p) (nat (´free-l t)));;

          (∗ sys-dlist-remove(node); ∗)
      ´mem-pool-info := ´mem-pool-info (p := rmhead-free-list (´mem-pool-info
p) (nat (´free-l t)))

        FI;;
        (∗ ==== end: block = sys-dlist-get(&p−>levels[l].free-list); ∗)

        IF ´blk t ≠ NULL THEN
          (∗ clear-free-bit(p, l, block-num(p, block, lsz)); ∗)
          ´mem-pool-info := set-bit-allocating ´mem-pool-info p (nat (´free-l t))
                      (block-num (´mem-pool-info p) (´blk t) ((´lsizes t)!(nat
(´free-l t))));;
          (∗ set the allocating node info of the thread ∗)
          ´allocating-node := ´allocating-node (t := Some (|pool = p, level = nat
(´free-l t),
                block = (block-num (´mem-pool-info p) (´blk t) ((´lsizes t)!(nat
(´free-l t)))), data = ´blk t |))
        FI
      END);;
      (∗ ==== end: blk = alloc-block(p, free-l, lsizes[free-l]); ∗)

      IF ´blk t = NULL THEN
        (t ▶ ´ret := ´ret (t := EAGAIN))
      ELSE

        FOR (t ▶ ´from-l := ´from-l(t := ´free-l t));
          (∗ level-empty (´mem-pool-info p) (nat (´alloc-l t)) ∧ ∗) ´from-l t <
```

´*alloc-l t*;

(∗∗∗∗∗∗∗∗∗ *we remove the FOR termination condition "level-empty" to remove a concurrency BUG here* ∗∗∗∗∗∗∗∗∗∗∗)

(*t* ► ´*from-l* := ´*from-l*(*t* := ´*from-l t* + *1*)) *DO*

(∗ ==== *start*: *blk* = *break-block*(*p*, *blk*, *from-l*, *lsizes*); ∗)

(*t* ► *ATOMIC*

´*bn* := ´*bn* (*t* := *block-num* (´*mem-pool-info p*) (´*blk t*) ((´*lsizes t*)!(*nat* (´*from-l t*))));;

´*mem-pool-info* := *set-bit-divide* ´*mem-pool-info p* (*nat* (´*from-l t*)) (´*bn t*);;

´*mem-pool-info* := *set-bit-allocating* ´*mem-pool-info p* (*nat* (´*from-l t* + *1*)) (*4* ∗ ´*bn t*);;

(∗ *set the allocating node info of the thread* ∗)

´*allocating-node* := ´*allocating-node* (*t* := *Some* (|*pool* = *p*, *level* = *nat* (´*from-l t* + *1*),

block = *4* ∗ ´*bn t*, *data* = ´*blk t* |));;

FOR ´*i* := ´*i* (*t* := *1*);

´*i t* < *4*;

´*i* := ´*i* (*t* := ´*i t* + *1*) *DO*

´*lbn* := ´*lbn* (*t* := *4* ∗ ´*bn t* + ´*i t*);;

´*lsz* := ´*lsz* (*t* := (´*lsizes t*) ! (*nat* (´*from-l t* + *1*)));;

´*block2* := ´*block2*(*t* := ´*lsz t* ∗ ´*i t* + ´*blk t*);;

(∗ *set-free-bit*(*p*, *l* + *1*, *lbn*); ∗)

´*mem-pool-info* := *set-bit-free* ´*mem-pool-info p* (*nat* (´*from-l t* + *1*)) (´*lbn t*);;

IF block-fits (´*mem-pool-info p*) (´*block2 t*) (´*lsz t*) *THEN*

(∗ *sys-dlist-append*(&*p*−>*levels*[*l* + *1*].*free-list*, *block2*); ∗)

´*mem-pool-info* := ´*mem-pool-info* (*p* :=

append-free-list (´*mem-pool-info p*) (*nat* (´*from-l t* + *1*)) (´*block2 t*))

FI

ROF

END)

(∗ ==== *end*: *blk* = *break-block*(*p*, *blk*, *from-l*, *lsizes*); ∗)

ROF;;

(∗ *finally set the node from allocating to allocated and remove the allocating node info of the thread* ∗)

(*t* ► ´*mem-pool-info* := *set-bit-alloc* ´*mem-pool-info p* (*nat* (´*alloc-l t*))

(*block-num* (´*mem-pool-info* *p*) (´*blk* *t*) ((´*lsizes* *t*)!(*nat* (´*alloc-l* *t*))));;
´*allocating-node* := ´*allocating-node* (*t* := *None*)
);;

(*t* ▶ ´*mempoolalloc-ret* := ´*mempoolalloc-ret* (*t* :=
Some (|*pool* = *p*, *level* = *nat* (´*alloc-l* *t*),
block = *block-num* (´*mem-pool-info* *p*) (´*blk* *t*) ((´*lsizes* *t*)!(*nat* (´*alloc-l* *t*))),
data = ´*blk* *t* |)));;

(*t* ▶ ´*ret* := ´*ret* (*t* := *OK*))
FI
FI
FI;;
(∗ ==== *end*: *ret* = *pool-alloc*(*p*, *block*, *size*); ==================== ∗)

(∗ *IF* ´*ret* *t* = *0* ∨ *timeout* = *NOWAIT* ∨ ´*ret* *t* = *EAGAIN* ∨ ´*ret* *t* ≠ *ENOMEM* *THEN* ∗)
(∗∗∗∗∗ *we change the IF condition to remove a functional BUG here* ∗∗∗∗∗)
IF ´*ret* *t* = *OK* ∨ *timeout* = *NOWAIT* ∨ ´*ret* *t* = *ESIZEERR* *THEN*
(*t* ▶ ´*rf* := ´*rf*(*t* := *True*));;
IF ´*ret* *t* = *EAGAIN* *THEN* (∗*EAGAIN should not export to users*∗)
(*t* ▶ ´*ret* := ´*ret*(*t* := *ENOMEM*))
FI
ELSE
IF ´*ret* *t* = *EAGAIN* *THEN* *SKIP*
ELSE
(*t* ▶ *ATOMIC*

(∗ *-pend-current-thread*(&*p*−>*wait-q*, *timeout*); ∗)
´*thd-state* := ´*thd-state*(*the* ´*cur* := *BLOCKED*);;
(∗´*cur* := *None*;;∗)
´*mem-pool-info* := ´*mem-pool-info*(*p* := ´*mem-pool-info* *p*(|*wait-q* := *wait-q* (´*mem-pool-info* *p*) @ [*the* ´*cur*] |));;

(∗ *-Swap*(*key*); ∗)
swap

END);;

IF ´*tmout* *t* ≠ *FOREVER* *THEN*
(*t* ▶ ´*tmout* := ´*tmout* (*t* := *int* (´*endt* *t*) − *int* ´*tick*));;
IF ´*tmout* *t* < *0* *THEN*
(*t* ▶ ´*rf* := ´*rf*(*t* := *True*));;
(*t* ▶ ´*ret* := ´*ret* (*t* := *ETIMEOUT*))
FI
FI

FI
FI
OD

END

definition *Mem-pool-free* :: *Thread* ⇒ *Mem-block* ⇒ (*EventLabel*, *Core*, *State*, *State com option*) *event*
where *Mem-pool-free t b* =
EVENT Mem-pool-freeE [*Block b*] ⇛ ($\mathcal{T}$ *t*)
WHEN
pool b ∈ ´*mem-pools*
∧ *level b* < *length* (*levels* (´*mem-pool-info* (*pool b*)))
∧ *block b* < *length* (*bits* (*levels* (´*mem-pool-info* (*pool b*))!(*level b*)))
∧ *data b* = *block-ptr* (´*mem-pool-info* (*pool b*)) ((*ALIGN4* (*max-sz* (´*mem-pool-info* (*pool b*)))) *div* (*4* ^ (*level b*))) (*block b*)
(∗∧ (*bits* ((*levels* (´*mem-pool-info* (*pool b*))) ! (*level b*))) ! (*block b*) = *ALLOCATED*
∧ ´*cur* = *Some t*∗) (∗ *t is the current thread* ∗)
(∗ ∧ *pool b* ∈ ´*pools-of-thread t* ∗) (∗ *the mem pool is shared in the thread t* ∗)
THEN
(∗ *here we set the bit to FREEING, so that other thread cannot mem-pool-free the same block*
it also requires that it can only free ALLOCATED block ∗)
(*t* ▶ *AWAIT* (*bits* ((*levels* (´*mem-pool-info* (*pool b*))) ! (*level b*))) ! (*block b*) = *ALLOCATED THEN*
´*mem-pool-info* := *set-bit-freeing* ´*mem-pool-info* (*pool b*) (*level b*) (*block b*);;
´*freeing-node* := ´*freeing-node* (*t* := *Some b*) (∗ *set the freeing node of current thread* ∗)
END);;

(*t* ▶ ´*need-resched* := ´*need-resched*(*t* := *False*));;
(∗ (*t* ▶ ´*lsizes* := ´*lsizes*(*t* := []));; ∗)
(*t* ▶ ´*lsizes* := ´*lsizes*(*t* := [*ALIGN4* (*max-sz* (´*mem-pool-info* (*pool b*)))]));;
FOR (*t* ▶ ´*i* := ´*i*(*t* := *1*));
´*i t* ≤ *level b*;
(*t* ▶ ´*i* := ´*i*(*t* := ´*i t* + *1*)) *DO*
(*t* ▶ ´*lsizes* := ´*lsizes*(*t* := ´*lsizes t* @ [*ALIGN4* (´*lsizes t* ! (´*i t* − *1*) *div 4*)]))
ROF;;

(∗ === *start*: *free-block*(*get-pool*(*block*−>*id.pool*), *block*−>*id.level*, *lsizes*, *block*−>*id.block*); ∗)
(*t* ▶ ´*free-block-r* := ´*free-block-r* (*t* := *True*));;
(*t* ▶ ´*bn* := ´*bn* (*t* := *block b*));;
(*t* ▶ ´*lvl* := ´*lvl* (*t* := *level b*));;

WHILE ´*free-block-r t DO*
(*t* ▶ ´*lsz* := ´*lsz* (*t* := ´*lsizes t* ! (´*lvl t*)));;

$(t \blacktriangleright$ ´*blk* := ´*blk* (*t* := *block-ptr* (´*mem-pool-info* (*pool b*)) (´*lsz t*) (´*bn t*)));;

$(t \blacktriangleright$ *ATOMIC*

´*mem-pool-info* := *set-bit-free* ´*mem-pool-info* (*pool b*) (´*lvl t*) (´*bn t*);;
´*freeing-node* := ´*freeing-node* (*t* := *None*);; (∗ *remove the freeing node info of the thread* ∗)

IF ´*lvl t* > *0* ∧ *partner-bits* (´*mem-pool-info* (*pool b*)) (´*lvl t*) (´*bn t*) *THEN*
FOR ´*i* := ´*i*(*t* := *0*);
´*i t* < *4*;
´*i* := ´*i*(*t* := ´*i t* + *1*) *DO*
´*bb* := ´*bb* (*t* := (´*bn t div 4*) ∗ *4* + ´*i t*);;
(∗($t \blacktriangleright$ ´*mem-pool-info* := *clear-free-bit* ´*mem-pool-info* (*pool b*) (´*lvl t*) (´*bb t*));;∗)
´*mem-pool-info* := *set-bit-noexist* ´*mem-pool-info* (*pool b*) (´*lvl t*) (´*bb t*);;
´*block-pt* := ´*block-pt* (*t* := *block-ptr* (´*mem-pool-info* (*pool b*)) (´*lsz t*) (´*bb t*));;
IF ´*bn t* ≠ ´*bb t* ∧ *block-fits* (´*mem-pool-info* (*pool b*))
(´*block-pt t*)
(´*lsz t*) *THEN*

(∗ *sys-dlist-remove*(*block-ptr*(*p*, *lsz*, *b*)); ∗)
´*mem-pool-info* := ´*mem-pool-info* ((*pool b*) :=
remove-free-list (´*mem-pool-info* (*pool b*)) (´*lvl t*) (´*block-pt t*))
FI
ROF;;

(
(∗´*j* := ´*j* (*t* := ´*lvl t*);; (∗ *use lbn and j to store the previous lvl and bn, or can not give the post condition* ∗)
´*lbn* := ´*lbn* (*t* := ´*bn t*);; (∗ *since the lbn and j are not used in M-pool-free* ∗)
´*lvl* := ´*lvl* (*t* := ´*j t* − *1*);;
´*bn* := ´*bn* (*t* := ´*lbn t div 4*);;∗)
´*lvl* := ´*lvl* (*t* := ´*lvl t* − *1*);;
´*bn* := ´*bn* (*t* := ´*bn t div 4*);;
(∗ *we add this statement. set the parent node from divided to freeing* ∗)
´*mem-pool-info* := *set-bit-freeing* ´*mem-pool-info* (*pool b*) (´*lvl t*) (´*bn t*);;
(∗´*freeing-node* := ´*freeing-node* (*t* := *Some* ⦇*pool* = (*pool b*), *level* = (´*lvl t*),
block = (´*bn t*), *data* = *block-ptr* (´*mem-pool-info* (*pool b*)) (´*lsz t*) (´*bn t*) ⦈)∗)
´*freeing-node* := ´*freeing-node* (*t* := *Some* ⦇*pool* = (*pool b*), *level* = (´*lvl t*),
block = (´*bn t*),
data = *block-ptr* (´*mem-pool-info* (*pool b*))
(((*ALIGN4* (*max-sz* (´*mem-pool-info* (*pool b*)))) *div* (*4* ^

```
(´lvl t))))
                          (´bn t) |))
         )

       ELSE
         IF block-fits (´mem-pool-info (pool b)) (´blk t) (´lsz t) THEN

           (* sys-dlist-append(&p->levels[level].free-list, block); *)
           ´mem-pool-info := ´mem-pool-info ((pool b) :=
                 append-free-list (´mem-pool-info (pool b)) (´lvl t) (´blk t) )
         FI;;

         ´free-block-r := ´free-block-r (t := False)
       FI

     END)

   OD;;
  (* === end: free-block(get-pool(block->id.pool), block->id.level, lsizes, block->id.block);
*)

   (t ► ATOMIC

     WHILE wait-q (´mem-pool-info (pool b)) ≠ [] DO
       ´th := ´th (t := hd (wait-q (´mem-pool-info (pool b))));;
       (* -unpend-thread(th); *)
       ´mem-pool-info := ´mem-pool-info (pool b := ´mem-pool-info (pool b)
             (|wait-q := tl (wait-q (´mem-pool-info (pool b)))|));;
       (* -ready-thread(th); *)
       ´thd-state := ´thd-state (´th t := READY);;
       ´need-resched := ´need-resched(t := True)
     OD;;

     IF ´need-resched t THEN
       reschedule
     FI
   END)
 END
```

definition *Schedule* :: *Thread* ⇒ (*EventLabel*, *Core*, *State*, *State com option*) *event*

```
where Schedule t ≡
  EVENT ScheduleE [Thread t] ⇛ 𝒮
  THEN
    AWAIT ´thd-state t = READY THEN  (* only schedule the READY threads
*)
      IF (´cur ≠ None) THEN
        ´thd-state := ´thd-state(the (´cur) := READY);;
        ´cur := None
```

FI;;
´cur := *Some t*;;
´thd-state := *´thd-state*(*t* := *RUNNING*)
END
END

definition *Tick* :: (*EventLabel*, *Core*, *State*, *State com option*) *event*
where *Tick* ≡
EVENT TickE [] ⇛ *Timer*
THEN
´tick := *´tick* + *1*
END

end

theory *invariant*
imports *mem-spec HOL−Eisbach.Eisbach-Tools*
begin

this theory defines the invariant and its lemmas.

3 invariants

3.1 defs of invariants

we consider multi-threaded execution on mono-core. A thread is the currently executing thread iff it is in RUNNING state.

definition *inv-cur* :: *State* ⇒ *bool*
where *inv-cur s* ≡ ∀ *t*. *cur s* = *Some t* ⟷ *thd-state s t* = *RUNNING*

abbreviation *dist-list* :: *'a list* ⇒ *bool*
where *dist-list l* ≡ ∀ *i j*. *i* < *length l* ∧ *j* < *length l* ∧ *i* ≠ *j* ⟶ *l*!*i* ≠ *l*!*j*

the relation of thread state and wait queue. here we dont consider other modules of zephyr, so blocked thread is in wait que of mem pools.

definition *inv-thd-waitq* :: *State* ⇒ *bool*
where *inv-thd-waitq s* ≡
(∀ *p*∈*mem-pools s*. ∀ *t*∈ *set* (*wait-q* (*mem-pool-info s p*)). *thd-state s t* = *BLOCKED*)

(∗ *thread in waitq is BLOCKED* ∗)
∧ (∀ *t*. *thd-state s t* = *BLOCKED* ⟶ (∃ *p*∈*mem-pools s*. *t* ∈ *set* (*wait-q* (*mem-pool-info s p*))))
(∗ *BLOCKED thread is in a waitq* ∗)
∧ (∀ *p*∈*mem-pools s*. *dist-list* (*wait-q* (*mem-pool-info s p*)))
(∗ *threads in a waitq are different with each other*, *which means a thread could not waiting for the same pool two times* ∗)

$\wedge$ ($\forall$ *p q. p*$\in$*mem-pools s* $\wedge$ *q*$\in$*mem-pools s* $\wedge$ *p* $\neq$ *q* $\longrightarrow$ ($\nexists$ *t. t* $\in$ *set* (*wait-q* (*mem-pool-info s p*))
$\wedge$ *t*$\in$ *set* (*wait-q* (*mem-pool-info s q*))))

invariant of configuration of memory pools. its actually a well-formed property for memory configuration. (1) the max size (the size of top-level (level 0) block) is 4^{n_levels} times of block size of the lowest level. 4 * n means that the block size of the lowest level is alignd with 4. (2) the block number at level 0 (n_max) ¿ 0, and the max number of levels is n_levels ¿ 0 (3) n_level is equal to the length of levels list. (4) the length of bitmap list at each level is equal to the block number at the same level. Thus, bitmap saves a complete quad-tree with height of n_levels. A real memory pool is a top subtree of the complete tree. bits of subnodes of a leaf node (ALLOCATED, FREE, ALLOCATING, FREEING) is NOEXIST.

abbreviation *inv-mempool-info-mp* :: *State* $\Rightarrow$ *mempool-ref* $\Rightarrow$ *bool*
where *inv-mempool-info-mp s p* $\equiv$
let mp = *mem-pool-info s p in*
buf mp $\neq$ *NULL* $\wedge$ ($\exists$ *n*>*0. max-sz mp* = (*4* $*$ *n*) $*$ (*4* ^ *n-levels mp*))
$\wedge$ *n-max mp* > *0* $\wedge$ *n-levels mp* > *0*
$\wedge$ *n-levels mp* = *length* (*levels mp*)
$\wedge$ ($\forall$ *i*<*length* (*levels mp*). *length* (*bits* (*levels mp* ! *i*)) = (*n-max mp*) $*$ *4* ^ *i*)

definition *inv-mempool-info* :: *State* $\Rightarrow$ *bool*
where *inv-mempool-info s* $\equiv$ $\forall$ *p*$\in$*mem-pools s. inv-mempool-info-mp s p*

lemma *inv-max-sz-gt0*: *inv-mempool-info s* $\Longrightarrow$ $\forall$ *p*$\in$*mem-pools s. let mp* = *mem-pool-info s p in max-sz mp* > *0*
unfolding *inv-mempool-info-def* **using** *neq0-conv* **by** *fastforce*

invariant between bitmap and free block list at each level. (1) bit of a block is FREE, iff its start address is in free list. the start address is buf mp + j * (max_sz mp div (4^i)), the start address of the mempool + block size at this level * block index (2) start address of blocks in free list is valid, i.e. it is the start address of some block (index n), where n is in the range of block index at the level (3) start address of blocks in free list are different with each other.

abbreviation *inv-bitmap-freelist-mp* :: *State* $\Rightarrow$ *mempool-ref* $\Rightarrow$ *bool*
where *inv-bitmap-freelist-mp s p* $\equiv$
let mp = *mem-pool-info s p in*
$\forall$ *i* < *length* (*levels mp*).
let bts = *bits* (*levels mp* ! *i*);
fl = *free-list* (*levels mp* ! *i*) *in*
($\forall$ *j* < *length bts. bts* ! *j* = *FREE* $\longleftrightarrow$ *buf mp* + *j* $*$ (*max-sz mp div* (*4* ^ *i*)) $\in$ *set fl*)
($*$ *the block corresponding to a free bit iff it is in freelist* $*$)
$\wedge$ ($\forall$ *j* < *length fl.* ($\exists$ *n. n* < *n-max mp* $*$ (*4* ^ *i*) $\wedge$ *fl* ! *j* = *buf mp* + *n*

$*$ *(max-sz mp div (4 ^ i))))*
(* *pointers in freelist are head address of blocks* *)
$\wedge$ *distinct fl* (*($\forall$ *k j. k < length fl* $\wedge$ *j < length fl* $\longrightarrow$ *fl!k = fl!j* $\longrightarrow$ *k = j*) *)
(* *pointers in freelist are different with each other* *)

definition *inv-bitmap-freelist* :: *State* $\Rightarrow$ *bool*
where *inv-bitmap-freelist s* $\equiv$
$\forall$ *p*$\in$*mem-pools s. inv-bitmap-freelist-mp s p*

this invariant represents that a memory pools is forest of valid quad-trees of blocks. parent node of a leaf node (ALLOCATED, FREE, ALLOCATING, FREEING) is an inner node (DIVIDED). parent node of an inner node (DIVIDED) is also a DIVIDED node. child nodes of a NOEXIST node is also NOEXIST nodes. parent node of a NOEXIST node should not be DIVIDE nodes (may be NOEXIST, ALLOCATED, FREE, ALLOCATING, FREEING)

abbreviation *inv-bitmap-mp* :: *State* $\Rightarrow$ *mempool-ref* $\Rightarrow$ *bool*
where *inv-bitmap-mp s p* $\equiv$
let mp = mem-pool-info s p in
$\forall$ *i < length (levels mp).*
let bts = bits (levels mp ! i) in
($\forall$ *j < length bts.*
(*bts ! j = FREE* $\vee$ *bts ! j = FREEING* $\vee$ *bts ! j = ALLOCATED* $\vee$ *bts ! j = ALLOCATING* $\longrightarrow$
(*i > 0* $\longrightarrow$ (*bits (levels mp ! (i − 1))) ! (j div 4) = DIVIDED*)
$\wedge$ (*i < length (levels mp) − 1* $\longrightarrow$ *noexist-bits mp (i+1) (j*4)*
))
$\wedge$ (*bts ! j = DIVIDED* $\longrightarrow$ *i > 0* $\longrightarrow$ (*bits (levels mp ! (i − 1))) ! (j div 4) = DIVIDED*)
$\wedge$ (*bts ! j = NOEXIST* $\longrightarrow$ *i < length (levels mp) − 1*
$\longrightarrow$ *noexist-bits mp (i+1) (j*4)*)
$\wedge$ (*bts ! j = NOEXIST* $\wedge$ *i > 0* $\longrightarrow$ (*bits (levels mp ! (i − 1))) ! (j div 4)* $\neq$ *DIVIDED*))

definition *inv-bitmap* :: *State* $\Rightarrow$ *bool*
where *inv-bitmap s* $\equiv$
$\forall$ *p*$\in$*mem-pools s. inv-bitmap-mp s p*

due to the rule of merge as possible, there should not exist a node with 4 FREE child blocks. In free syscall, 4 free child blocks should be merged to a bigger block.

abbreviation *inv-bitmap-not4free-mp* :: *State* $\Rightarrow$ *mempool-ref* $\Rightarrow$ *bool*
where *inv-bitmap-not4free-mp s p* $\equiv$
let mp = mem-pool-info s p in
$\forall$ *i < length (levels mp).*
let bts = bits (levels mp ! i) in
($\forall$ *j < length bts. i > 0* $\longrightarrow$ $\neg$ *partner-bits mp i j*)

definition *inv-bitmap-not4free* :: *State* ⇒ *bool*
where *inv-bitmap-not4free s* ≡
$\forall p \in$ *mem-pools s*. *inv-bitmap-not4free-mp s p*

blocks at level 0 should not be NOEXIST. If so, the memory pool does not exist. We only allow real memory pools.

definition *inv-bitmap0* :: *State* ⇒ *bool*
where *inv-bitmap0 s* ≡
$\forall p \in$ *mem-pools s*. *let bits0* = *bits* (*levels* (*mem-pool-info s p*) ! *0*) *in* $\forall i <$ *length bits0*. *bits0* ! *i* $\neq$ *NOEXIST*

blocks at last level (n_level - 1) should not be split again, thus should not be DIVIDED

definition *inv-bitmapn* :: *State* ⇒ *bool*
where *inv-bitmapn s* ≡
$\forall p \in$ *mem-pools s*. *let bitsn* = *bits* ((*levels* (*mem-pool-info s p*) ! (*length* (*levels* (*mem-pool-info s p*)) − *1*)))
in $\forall i <$ *length bitsn*. *bitsn* ! *i* $\neq$ *DIVIDED*

definition *mem-block-addr-valid* :: *State* ⇒ *Mem-block* ⇒ *bool*
where *mem-block-addr-valid s b* ≡
data b = *buf* (*mem-pool-info s* (*pool b*)) + (*block b*) ∗ ((*max-sz* (*mem-pool-info s* (*pool b*))) *div* (*4* ˆ (*level b*)))

invariants between FREEING/ALLOCATING blocks and freeing/allocating_node variables.

definition *inv-aux-vars* :: *State* ⇒ *bool*
where *inv-aux-vars s* ≡
($\forall$ *t n*. *freeing-node s t* = *Some n* ⟶ *get-bit* (*mem-pool-info s*) (*pool n*) (*level n*) (*block n*) = *FREEING*)
(∗ *freeing node is state of FREEING* ∗)
∧ ($\forall$ *n*. *get-bit* (*mem-pool-info s*) (*pool n*) (*level n*) (*block n*) = *FREEING* ∧ *mem-block-addr-valid s n*
⟶ ($\exists$ *t*. *freeing-node s t* = *Some n*))
(∗ *node of state of FREEING is freeing* ∗)
∧ ($\forall$ *t n*. *allocating-node s t* = *Some n* ⟶ *get-bit* (*mem-pool-info s*) (*pool n*) (*level n*) (*block n*) = *ALLOCATING*)
(∗ *freeing node is state of FREEING* ∗)
∧ ($\forall$ *n*. *get-bit* (*mem-pool-info s*) (*pool n*) (*level n*) (*block n*) = *ALLOCATING* ∧ *mem-block-addr-valid s n*
⟶ ($\exists$ *t*. *allocating-node s t* = *Some n*))
(∗ *node of state of FREEING is freeing* ∗)
∧ ($\forall$ *t1 t2 n1 n2*. *t1* $\neq$ *t2* ∧ *freeing-node s t1* = *Some n1* ∧ *freeing-node s t2* = *Some n2*
⟶ ¬(*pool n1* = *pool n2* ∧ *level n1* = *level n2* ∧ *block n1* = *block n2*))
(∗*here we only consider the pool*, *level*, *and block*, *not the first addr of the block* ∗)

(* *freeing nodes are different each other* *)
$\wedge$ ($\forall$ *t1 t2 n1 n2. t1* $\neq$ *t2* $\wedge$ *allocating-node s t1* = *Some n1* $\wedge$ *allocating-node s t2* = *Some n2*
$\longrightarrow \neg$(*pool n1* = *pool n2* $\wedge$ *level n1* = *level n2* $\wedge$ *block n1* = *block n2*))
(* *allocating node are different each other* *)
$\wedge$ ($\forall$ *t1 t2 n1 n2. allocating-node s t1* = *Some n1* $\wedge$ *freeing-node s t2* = *Some n2*
$\longrightarrow \neg$(*pool n1* = *pool n2* $\wedge$ *level n1* = *level n2* $\wedge$ *block n1* = *block n2*))

definition *inv* :: *State* $\Rightarrow$ *bool*
where *inv s* $\equiv$ *inv-cur s* $\wedge$ *inv-thd-waitq s* $\wedge$ *inv-mempool-info s*
$\wedge$ *inv-bitmap-freelist s* $\wedge$ *inv-bitmap s* $\wedge$ *inv-aux-vars s*
$\wedge$ *inv-bitmap0 s* $\wedge$ *inv-bitmapn s* $\wedge$ *inv-bitmap-not4free s*

method *simp-inv* = (*simp add*:*inv-def inv-bitmap-def inv-bitmap-freelist-def*
inv-mempool-info-def inv-thd-waitq-def inv-cur-def inv-aux-vars-def
inv-bitmap0-def inv-bitmapn-def
inv-bitmap-not4free-def mem-block-addr-valid-def)

method *unfold-inv* = (*unfold inv-def inv-bitmap-def inv-bitmap-freelist-def*
inv-mempool-info-def inv-thd-waitq-def inv-cur-def inv-aux-vars-def
inv-bitmap0-def inv-bitmapn-def
inv-bitmap-not4free-def mem-block-addr-valid-def)[*1*]

lemma *inv-imp-fl-lt0*:
inv Va $\Longrightarrow$
$\forall$ *p*$\in$*mem-pools Va*.
let mp = *mem-pool-info Va p in*
$\forall$ *i* < *length* (*levels mp*).
$\forall$ *j* < *length* (*free-list* (*levels mp* ! *i*)). *free-list* (*levels mp* ! *i*) ! *j* > *0*
apply(*simp add*:*inv-def inv-mempool-info-def inv-bitmap-freelist-def*)
apply(*simp add*:*Let-def*) **apply** *clarsimp*
by *fastforce*

3.2 initial state s_0

we dont consider mem_pool_init, only define $s0$ to show the state after memory pool initialization.

axiomatization *s0*::*State* **where**
s0a1: *cur s0* = *None* **and**
s0a2: *tick s0* = *0* **and**
s0a3: *thd-state s0* = (λ*t*. *READY*) **and**
s0a5: *mem-pools s0* $\neq$ {} **and**
s0a7: $\forall$ *p*$\in$*mem-pools s0*. *wait-q* (*mem-pool-info s0 p*) = [] **and**
s0a6: $\forall$ *p*$\in$*mem-pools s0*. *let mp* = *mem-pool-info s0 p in*

buf mp > *0* ∧ (∃ *n*>*0*. *max-sz mp* = (*4* ∗ *n*) ∗ (*4* ^ *n-levels mp*))
∧ *n-max mp* > *0* ∧ *n-levels mp* > *1*
∧ *n-levels mp* = *length* (*levels mp*) **and**
s0a8: ∀ *p*∈*mem-pools s0*. (∗ *defines level 1 to n* ∗)
let mp = *mem-pool-info s0 p in*
∀ *i*. *i* > *0* ∧ *i* < *length* (*levels mp*) ⟶
length (*bits* (*levels mp* ! *i*)) = *n-max mp* ∗ *4* ^ *i*
∧ (∀ *j*<*length* (*bits* (*levels mp* ! *i*)). *bits* (*levels mp* ! *i*) ! *j* = *NOEXIST*)
∧ *free-list* (*levels mp* ! *i*) = [] **and**
s0a9: ∀ *p*∈*mem-pools s0*. (∗ *defines the level0* ∗)
let mp = *mem-pool-info s0 p*;
lv0 = (*levels mp*)!*0 in*
length (*bits lv0*) = *n-max mp*
∧ *length* (*free-list lv0*) = *n-max mp*
∧ (∀ *i*<*length* (*bits lv0*). (*bits lv0*)!*i* = *FREE*)
∧ (∀ *i*<*length* (*free-list lv0*). (*free-list lv0*) ! *i* = (*buf mp*) + *i* ∗ *max-sz mp*)
∧ *distinct* (*free-list lv0*) **and**
s0a4: *freeing-node s0* = *Map.empty* **and**
s0a10: *allocating-node s0* = *Map.empty* **and**
s0a11: ∄ *n*. *get-bit-s s0* (*pool n*) (*level n*) (*block n*) = *FREEING* **and**
s0a12: ∄ *n*. *get-bit-s s0* (*pool n*) (*level n*) (*block n*) = *ALLOCATING*

lemma *s0-max-sz-gt0*: ∀ *p*∈*mem-pools s0*. *let mp* = *mem-pool-info s0 p in max-sz mp* > *0*
using *s0a6 zero-less-power* **by** *fastforce*

lemma *s0-inv-cur*: *inv-cur s0*
by (*simp add*:*inv-cur-def s0a1 s0a3*)

lemma *s0-inv-thdwaitq*: *inv-thd-waitq s0*
by (*simp add*: *inv-thd-waitq-def s0a7 s0a3*)

lemma *s0-inv-mempool-info*: *inv-mempool-info s0*
apply (*simp add*: *inv-mempool-info-def Let-def*) **apply** *clarsimp*
apply(*rule conjI*) **apply** (*metis neq0-conv s0a6*)
apply(*rule conjI*) **apply** (*meson s0a6*)
apply(*rule conjI*) **apply** (*meson s0a6*)
apply(*rule conjI*) **using** *neq0-conv s0a6* **apply** *fastforce*
apply(*rule conjI*) **apply** (*meson s0a6*)
by (*metis One-nat-def mult-numeral-1-right neq0-conv numeral-1-eq-Suc-0 power.simps*(*1*) *s0a8 s0a9*)

lemma *s0-inv-bitmap-freelist*: *inv-bitmap-freelist s0*
apply(*simp add*:*inv-bitmap-freelist-def*)
apply(*simp add*: *Let-def*) **apply** *clarsimp*

apply(*case-tac i = 0*)
apply(*rule conjI*) **apply** *clarsimp* **apply** (*metis nth-mem s0a9*)
apply(*rule conjI*) **apply** *clarsimp* **apply** (*metis s0a9*)
apply (*meson s0a9*)

apply(*rule conjI*) **apply** *clarsimp*
apply(*subgoal-tac n-levels* (*mem-pool-info s0 p*) = *length* (*levels* (*mem-pool-info s0 p*)))
prefer *2* **apply** (*meson s0a6*)
apply(*subgoal-tac get-bit-s s0 p i j* $\neq$ *FREE*)
prefer *2* **apply** (*metis BlockState.distinct*(*13*) *s0a8*)
apply(*subgoal-tac set* (*free-list* (*levels* (*mem-pool-info s0 p*) ! *i*)) = {})
prefer *2* **apply** (*metis all-not-in-conv in-set-conv-nth length-greater-0-conv neq0-conv not-less-zero s0a8*)
apply *simp*

apply(*rule conjI*) **apply** *clarsimp*
apply (*metis length-greater-0-conv neq0-conv not-less-zero s0a8*)
apply (*metis distinct-conv-nth length-0-conv neq0-conv not-less-zero s0a8*)
done

lemma *s0-inv-bitmap*: *inv-bitmap s0*
apply(*simp add*: *inv-bitmap-def*)
apply(*simp add*: *Let-def*) **apply** *clarsimp*
apply(*case-tac i = 0*)
apply *clarsimp* **using** *s0a6 s0a8 s0a9* **apply**(*simp add*:*Let-def partner-bits-def*)

apply(*rule conjI*) **apply** *clarsimp* **using** *s0a6 s0a8 s0a9* **apply**(*simp add*:*Let-def*)

apply(*rule conjI*) **apply** *clarsimp* **using** *s0a6 s0a8 s0a9* **apply**(*simp add*:*Let-def*)

apply(*rule conjI*) **apply** *clarsimp* **using** *s0a6 s0a8 s0a9* **apply**(*simp add*:*Let-def*)

apply(*rule conjI*) **apply** *clarsimp* **using** *s0a6 s0a8 s0a9* **apply**(*simp add*:*Let-def*)

apply(*rule conjI*) **apply** *clarsimp* **using** *s0a6 s0a8 s0a9* **apply**(*simp add*:*Let-def*)

apply(*rule conjI*) **apply** *clarsimp* **using** *s0a6 s0a8 s0a9* **apply**(*simp add*:*Let-def*)

apply(*case-tac i = 1*) **apply** *clarsimp* **using** *s0a6 s0a8 s0a9* **apply**(*simp add*:*Let-def*)
apply(*subgoal-tac i* > *1*) **prefer** *2* **apply** *simp*
apply(*subgoal-tac get-bit-s s0 p* (*i* − *Suc NULL*) (*j div 4*) = *NOEXIST*)
prefer *2* **using** *s0a8* **apply**(*simp add*: *Let-def*)
apply(*subgoal-tac j div 4* < *length* (*bits* (*levels* (*mem-pool-info s0 p*) ! (*i* − *1*))))
prefer *2* **using** *s0a6* **apply**(*simp add*:*Let-def*)
apply(*subgoal-tac n-max* (*mem-pool-info s0 p*) > *0*)

prefer *2* **using** *s0a6* **apply**(*simp add*:*Let-def*)
apply(*simp add*: *power-eq-if*)
apply *auto*[*1*]
apply *simp*

done

lemma *s0-inv-bitmap-not4free*: *inv-bitmap-not4free s0*
apply(*simp add*: *inv-bitmap-not4free-def*)
apply(*simp add*: *Let-def*) **apply** *clarsimp*
using *s0a6 s0a8 s0a9* **apply**(*simp add*:*Let-def partner-bits-def*)
done

lemma *s0-inv-aux-vars*: *inv-aux-vars s0*
apply(*simp add*: *inv-aux-vars-def Let-def*)
apply(*rule conjI*) **apply** (*simp add*: *s0a4*)
apply(*rule conjI*) **apply** *clarify* **using** *s0a11* **apply** *auto*[*1*]
apply(*rule conjI*) **apply** (*simp add*: *s0a10*)
apply(*rule conjI*) **apply** *clarify* **using** *s0a12* **apply** *auto*[*1*]
apply(*rule conjI*) **apply** (*simp add*: *s0a4*)
apply(*rule conjI*) **apply** (*simp add*: *s0a10*)
apply (*simp add*: *s0a4 s0a10*)
done

lemma *s0-inv-bitmap-freelist0*: *inv-bitmap0 s0*
apply(*simp add*: *inv-bitmap0-def Let-def*)
using *s0a9* **apply**(*simp add*:*Let-def*)
done

lemma *s0-inv-bitmap-freelistn*: *inv-bitmapn s0*
apply(*simp add*: *inv-bitmapn-def Let-def*)
using *s0a8* **apply**(*simp add*:*Let-def*) **apply** *clarify*
apply(*subgoal-tac get-bit-s s0 p* (*length* (*levels* (*mem-pool-info s0 p*)) − *Suc 0*) *i* = *NOEXIST*)
prefer *2* **apply**(*subgoal-tac length* (*levels* (*mem-pool-info s0 p*)) > *0*)
prefer *2* **using** *s0a6* **apply**(*simp add*:*Let-def*) **apply** *auto*[*1*]
using *s0a6* **apply**(*simp add*:*Let-def*) **apply** *auto*[*1*]
apply *simp*
done

lemma *s0-inv*: *inv s0*
apply(*unfold inv-def*)
apply(*rule conjI*) **using** *s0-inv-cur* **apply** *fast*
apply(*rule conjI*) **using** *s0-inv-thdwaitq* **apply** *fast*
apply(*rule conjI*) **using** *s0-inv-mempool-info* **apply** *fast*
apply(*rule conjI*) **using** *s0-inv-bitmap-freelist* **apply** *fast*
apply(*rule conjI*) **using** *s0-inv-bitmap* **apply** *fast*
apply(*rule conjI*) **using** *s0-inv-aux-vars* **apply** *fast*
apply(*rule conjI*) **using** *s0-inv-bitmap-freelist0* **apply** *fast*

apply(*rule conjI*) **using** *s0-inv-bitmap-freelistn* **apply** *fast*
using *s0-inv-bitmap-not4free* **apply** *fast*
done

3.3 lemmas of invariants

lemma *inv-bitmap-presv-setbit-0*:
$\neg\ (x = l \wedge y = b) \Longrightarrow$
$Vb = Va(\!| mem\text{-}pool\text{-}info := (mem\text{-}pool\text{-}info\ Va)$
$(p := mem\text{-}pool\text{-}info\ Va\ p$
$(\!| levels := levels\ (mem\text{-}pool\text{-}info\ Va\ p)$
$[l := (levels\ (mem\text{-}pool\text{-}info\ Va\ p)\ !\ l)(\!| bits := bits\ (levels\ (mem\text{-}pool\text{-}info\ Va\ p)\ !\ l)[b := st] |\!)] |\!))|\!) \Longrightarrow$
$get\text{-}bit\text{-}s\ Va\ p\ x\ y = get\text{-}bit\text{-}s\ Vb\ p\ x\ y$
apply *simp* **by** (*metis* (*no-types*, *lifting*) *Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*)
Mem-pool-lvl.surjective list-update-beyond not-less nth-list-update-eq nth-list-update-neq)

lemma *inv-bitmap-presv-setbit*:
$inv\text{-}bitmap\ Va \Longrightarrow$
$get\text{-}bit\text{-}s\ Va\ p\ l\ b = FREE \vee get\text{-}bit\text{-}s\ Va\ p\ l\ b = FREEING \vee get\text{-}bit\text{-}s\ Va\ p\ l\ b = ALLOCATED$
$\vee\ get\text{-}bit\text{-}s\ Va\ p\ l\ b = ALLOCATING \Longrightarrow$
$st = FREE \vee st = FREEING \vee st = ALLOCATED \vee st = ALLOCATING \Longrightarrow$
$Vb = set\text{-}bit\text{-}s\ Va\ p\ l\ b\ st \Longrightarrow$
$inv\text{-}bitmap\ Vb$
apply(*simp add*:*inv-bitmap-def*) **apply**(*simp add*:*set-bit-s-def set-bit-def*)

apply(*simp add*:*Let-def*) **apply** *clarify* **apply**(*rename-tac ii jj*)
apply(*subgoal-tac* $p \in mem\text{-}pools\ Va$) **prefer** *2* **apply**(*simp add*:*set-bit-s-def set-bit-def*)
apply(*subgoal-tac* $jj < length\ (bits\ (levels\ (mem\text{-}pool\text{-}info\ Va\ p)\ !\ ii))$)
prefer *2* **apply**(*simp add*:*set-bit-s-def set-bit-def*)
apply (*metis* (*no-types*, *lifting*) *Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*)
Mem-pool-lvl.surjective list-updt-samelen nth-list-update-eq nth-list-update-neq)

apply(*rule conjI*) **apply** *clarify* **apply**(*rule conjI*) **apply** *clarify*

apply(*subgoal-tac* $(bits\ (levels\ (mem\text{-}pool\text{-}info\ Va\ p)\ !\ (ii - 1)))\ !\ (jj\ div\ 4) = DIVIDED$)
prefer *2* **apply** (*smt Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*) *Mem-pool-lvl.surjective*
One-nat-def nth-list-update-eq nth-list-update-neq)
apply(*subgoal-tac* $(bits\ (levels\ (mem\text{-}pool\text{-}info\ Va\ p)\ !\ (ii - 1)))\ !\ (jj\ div\ 4)$
$= (bits\ (levels\ (mem\text{-}pool\text{-}info\ Vb\ p)\ !\ (ii - 1)))\ !\ (jj\ div\ 4)$)
prefer *2* **apply**(*case-tac* $ii - 1 = l \wedge jj\ div\ 4 = b$) **apply** *simp* **using** *inv-bitmap-presv-setbit-0*

apply *simp*
apply *simp*

apply *clarify* **apply**(*rule conjI*)

apply(*subgoal-tac* (*bits* (*levels* (*mem-pool-info Va p*) ! *Suc ii*)) ! (*jj* ∗ *4*) = *NOEXIST*)
prefer *2* **apply** (*smt Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*) *Mem-pool-lvl.surjective*

nth-list-update-eq nth-list-update-neq)
apply(*subgoal-tac* (*bits* (*levels* (*mem-pool-info Va p*) ! *Suc ii*)) ! (*jj* ∗ *4*)
= (*bits* (*levels* (*mem-pool-info Vb p*) ! *Suc ii*)) ! (*jj* ∗ *4*))
prefer *2* **apply**(*case-tac Suc ii* = *l* ∧ *jj* ∗ *4* = *b*) **apply** *simp* **using** *inv-bitmap-presv-setbit-0*
apply *simp*
apply *simp*

apply(*rule conjI*)
apply(*subgoal-tac* (*bits* (*levels* (*mem-pool-info Va p*) ! *Suc ii*)) ! (*jj* ∗ *4* + *1*) = *NOEXIST*)
prefer *2* **apply** (*smt Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*) *Mem-pool-lvl.surjective*

Nat.add-0-right One-nat-def add-Suc-right nth-list-update-eq nth-list-update-neq)

apply(*subgoal-tac* (*bits* (*levels* (*mem-pool-info Va p*) ! *Suc ii*)) ! (*jj* ∗ *4* + *1*)
= (*bits* (*levels* (*mem-pool-info Vb p*) ! *Suc ii*)) ! (*jj* ∗ *4* + *1*))
prefer *2* **apply**(*case-tac Suc ii* = *l* ∧ *jj* ∗ *4* + *1* = *b*) **apply** *simp* **using** *inv-bitmap-presv-setbit-0* **apply** *metis*
apply *simp*

apply(*rule conjI*)
apply(*subgoal-tac* (*bits* (*levels* (*mem-pool-info Va p*) ! *Suc ii*)) ! (*jj* ∗ *4* + *2*) = *NOEXIST*)
prefer *2*
apply (*smt Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*) *Mem-pool-lvl.surjective*

add-2-eq-Suc′ nth-list-update-eq nth-list-update-neq)
apply(*subgoal-tac* (*bits* (*levels* (*mem-pool-info Va p*) ! *Suc ii*)) ! (*jj* ∗ *4* + *2*)
= (*bits* (*levels* (*mem-pool-info Vb p*) ! *Suc ii*)) ! (*jj* ∗ *4* + *2*))
prefer *2* **apply**(*case-tac Suc ii* = *l* ∧ *jj* ∗ *4* + *2* = *b*) **apply** *simp* **using** *inv-bitmap-presv-setbit-0* **apply** *metis*
apply *simp*

apply(*subgoal-tac* (*bits* (*levels* (*mem-pool-info Va p*) ! *Suc ii*)) ! (*jj* ∗ *4* + *3*) = *NOEXIST*)
prefer *2*
apply (*smt Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*) *Mem-pool-lvl.surjective*

add-2-eq-Suc′ nth-list-update-eq nth-list-update-neq)
apply(*subgoal-tac* (*bits* (*levels* (*mem-pool-info Va p*) ! *Suc ii*)) ! (*jj* ∗ *4* + *3*)

$= (bits\ (levels\ (mem\text{-}pool\text{-}info\ Vb\ p)\ !\ Suc\ ii))\ !\ (jj * 4 + 3))$
prefer *2* **apply**(*case-tac Suc ii* $=$ *l* $\wedge$ *jj* $*$ *4* $+$ *3* $=$ *b*) **apply** *simp* **using** *inv-bitmap-presv-setbit-0* **apply** *metis*
apply *simp*

apply(*rule conjI*) **apply** *clarify* **apply**(*rule conjI*) **apply** *clarify*

apply(*subgoal-tac* (*bits* (*levels* (*mem-pool-info Va p*) ! (*ii* $-$ *1*))) ! (*jj div 4*) $=$ *DIVIDED*)
prefer *2* **apply** (*smt Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*) *Mem-pool-lvl.surjective*

One-nat-def nth-list-update-eq nth-list-update-neq)
apply(*subgoal-tac* (*bits* (*levels* (*mem-pool-info Va p*) ! (*ii* $-$ *1*))) ! (*jj div 4*)
$=$ (*bits* (*levels* (*mem-pool-info Vb p*) ! (*ii* $-$ *1*))) ! (*jj div 4*)))
prefer *2* **apply**(*case-tac ii* $-$ *1* $=$ *l* $\wedge$ *jj div 4* $=$ *b*) **apply** *simp* **using** *inv-bitmap-presv-setbit-0*
apply *simp*
apply *simp*

apply *clarify* **apply**(*rule conjI*)

apply(*subgoal-tac* (*bits* (*levels* (*mem-pool-info Va p*) ! *Suc ii*)) ! (*jj* $*$ *4*) $=$ *NOEXIST*)
prefer *2* **apply** (*smt Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*) *Mem-pool-lvl.surjective*

nth-list-update-eq nth-list-update-neq)
apply(*subgoal-tac* (*bits* (*levels* (*mem-pool-info Va p*) ! *Suc ii*)) ! (*jj* $*$ *4*)
$=$ (*bits* (*levels* (*mem-pool-info Vb p*) ! *Suc ii*)) ! (*jj* $*$ *4*)))
prefer *2* **apply**(*case-tac Suc ii* $=$ *l* $\wedge$ *jj* $*$ *4* $=$ *b*) **apply** *simp* **using** *inv-bitmap-presv-setbit-0*
apply *simp*
apply *simp*

apply(*rule conjI*)
apply(*subgoal-tac* (*bits* (*levels* (*mem-pool-info Va p*) ! *Suc ii*)) ! (*jj* $*$ *4* $+$ *1*) $=$ *NOEXIST*)
prefer *2* **apply** (*smt Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*) *Mem-pool-lvl.surjective*

Nat.add-0-right One-nat-def add-Suc-right nth-list-update-eq nth-list-update-neq)

apply(*subgoal-tac* (*bits* (*levels* (*mem-pool-info Va p*) ! *Suc ii*)) ! (*jj* $*$ *4* $+$ *1*)
$=$ (*bits* (*levels* (*mem-pool-info Vb p*) ! *Suc ii*)) ! (*jj* $*$ *4* $+$ *1*)))
prefer *2* **apply**(*case-tac Suc ii* $=$ *l* $\wedge$ *jj* $*$ *4* $+$ *1* $=$ *b*) **apply** *simp* **using** *inv-bitmap-presv-setbit-0* **apply** *metis*
apply *simp*

apply(*rule conjI*)
apply(*subgoal-tac* (*bits* (*levels* (*mem-pool-info Va p*) ! *Suc ii*)) ! (*jj* $*$ *4* $+$ *2*) $=$ *NOEXIST*)
prefer *2*

apply (*smt Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*) *Mem-pool-lvl.surjective*

add-2-eq-Suc′ nth-list-update-eq nth-list-update-neq)
apply(*subgoal-tac* (*bits* (*levels* (*mem-pool-info Va p*) ! *Suc ii*)) ! (*jj* ∗ *4* + *2*)
= (*bits* (*levels* (*mem-pool-info Vb p*) ! *Suc ii*)) ! (*jj* ∗ *4* + *2*))
prefer *2* **apply**(*case-tac Suc ii* = *l* ∧ *jj* ∗ *4* + *2* = *b*) **apply** *simp* **using** *inv-bitmap-presv-setbit-0* **apply** *metis*
apply *simp*

apply(*subgoal-tac* (*bits* (*levels* (*mem-pool-info Va p*) ! *Suc ii*)) ! (*jj* ∗ *4* + *3*) = *NOEXIST*)
prefer *2*
apply (*smt Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*) *Mem-pool-lvl.surjective*

add-2-eq-Suc′ nth-list-update-eq nth-list-update-neq)
apply(*subgoal-tac* (*bits* (*levels* (*mem-pool-info Va p*) ! *Suc ii*)) ! (*jj* ∗ *4* + *3*)
= (*bits* (*levels* (*mem-pool-info Vb p*) ! *Suc ii*)) ! (*jj* ∗ *4* + *3*))
prefer *2* **apply**(*case-tac Suc ii* = *l* ∧ *jj* ∗ *4* + *3* = *b*) **apply** *simp* **using** *inv-bitmap-presv-setbit-0* **apply** *metis*
apply *simp*

apply(*rule conjI*) **apply** *clarify* **apply**(*rule conjI*) **apply** *clarify*

apply(*subgoal-tac* (*bits* (*levels* (*mem-pool-info Va p*) ! (*ii* − *1*))) ! (*jj div 4*) = *DIVIDED*)
prefer *2* **apply** (*smt Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*) *Mem-pool-lvl.surjective*

One-nat-def nth-list-update-eq nth-list-update-neq)
apply(*subgoal-tac* (*bits* (*levels* (*mem-pool-info Va p*) ! (*ii* − *1*))) ! (*jj div 4*)
= (*bits* (*levels* (*mem-pool-info Vb p*) ! (*ii* − *1*))) ! (*jj div 4*))
prefer *2* **apply**(*case-tac ii* − *1* = *l* ∧ *jj div 4* = *b*) **apply** *simp* **using** *inv-bitmap-presv-setbit-0*
apply *simp*
apply *simp*

apply *clarify* **apply**(*rule conjI*)

apply(*subgoal-tac* (*bits* (*levels* (*mem-pool-info Va p*) ! *Suc ii*)) ! (*jj* ∗ *4*) = *NOEXIST*)
prefer *2* **apply** (*smt Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*) *Mem-pool-lvl.surjective*

nth-list-update-eq nth-list-update-neq)
apply(*subgoal-tac* (*bits* (*levels* (*mem-pool-info Va p*) ! *Suc ii*)) ! (*jj* ∗ *4*)
= (*bits* (*levels* (*mem-pool-info Vb p*) ! *Suc ii*)) ! (*jj* ∗ *4*))
prefer *2* **apply**(*case-tac Suc ii* = *l* ∧ *jj* ∗ *4* = *b*) **apply** *simp* **using** *inv-bitmap-presv-setbit-0*
apply *simp*
apply *simp*

apply(*rule conjI*)

apply(*subgoal-tac* (*bits* (*levels* (*mem-pool-info Va p*) ! *Suc ii*)) ! (*jj* ∗ *4* + *1*) = *NOEXIST*)
prefer *2* **apply** (*smt Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*) *Mem-pool-lvl.surjective*

Nat.add-0-right One-nat-def add-Suc-right nth-list-update-eq nth-list-update-neq)

apply(*subgoal-tac* (*bits* (*levels* (*mem-pool-info Va p*) ! *Suc ii*)) ! (*jj* ∗ *4* + *1*)
= (*bits* (*levels* (*mem-pool-info Vb p*) ! *Suc ii*)) ! (*jj* ∗ *4* + *1*))
prefer *2* **apply**(*case-tac Suc ii* = *l* ∧ *jj* ∗ *4* + *1* = *b*) **apply** *simp* **using** *inv-bitmap-presv-setbit-0* **apply** *metis*
apply *simp*

apply(*rule conjI*)
apply(*subgoal-tac* (*bits* (*levels* (*mem-pool-info Va p*) ! *Suc ii*)) ! (*jj* ∗ *4* + *2*) = *NOEXIST*)
prefer *2*
apply (*smt Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*) *Mem-pool-lvl.surjective*

add-2-eq-Suc′ nth-list-update-eq nth-list-update-neq)
apply(*subgoal-tac* (*bits* (*levels* (*mem-pool-info Va p*) ! *Suc ii*)) ! (*jj* ∗ *4* + *2*)
= (*bits* (*levels* (*mem-pool-info Vb p*) ! *Suc ii*)) ! (*jj* ∗ *4* + *2*))
prefer *2* **apply**(*case-tac Suc ii* = *l* ∧ *jj* ∗ *4* + *2* = *b*) **apply** *simp* **using** *inv-bitmap-presv-setbit-0* **apply** *metis*
apply *simp*

apply(*subgoal-tac* (*bits* (*levels* (*mem-pool-info Va p*) ! *Suc ii*)) ! (*jj* ∗ *4* + *3*) = *NOEXIST*)
prefer *2*
apply (*smt Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*) *Mem-pool-lvl.surjective*

add-2-eq-Suc′ nth-list-update-eq nth-list-update-neq)
apply(*subgoal-tac* (*bits* (*levels* (*mem-pool-info Va p*) ! *Suc ii*)) ! (*jj* ∗ *4* + *3*)
= (*bits* (*levels* (*mem-pool-info Vb p*) ! *Suc ii*)) ! (*jj* ∗ *4* + *3*))
prefer *2* **apply**(*case-tac Suc ii* = *l* ∧ *jj* ∗ *4* + *3* = *b*) **apply** *simp* **using** *inv-bitmap-presv-setbit-0* **apply** *metis*
apply *simp*

apply(*rule conjI*) **apply** *clarify* **apply**(*rule conjI*) **apply** *clarify*

apply(*subgoal-tac* (*bits* (*levels* (*mem-pool-info Va p*) ! (*ii* − *1*))) ! (*jj div 4*) = *DIVIDED*)
prefer *2* **apply** (*smt Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*) *Mem-pool-lvl.surjective*

One-nat-def nth-list-update-eq nth-list-update-neq)
apply(*subgoal-tac* (*bits* (*levels* (*mem-pool-info Va p*) ! (*ii* − *1*))) ! (*jj div 4*)
= (*bits* (*levels* (*mem-pool-info Vb p*) ! (*ii* − *1*))) ! (*jj div 4*))
prefer *2* **apply**(*case-tac ii* − *1* = *l* ∧ *jj div 4* = *b*) **apply** *simp* **using** *inv-bitmap-presv-setbit-0*
apply *simp*

apply *simp*

apply *clarify* **apply**(*rule conjI*)

apply(*subgoal-tac* (*bits* (*levels* (*mem-pool-info Va p*) ! *Suc ii*)) ! (*jj* $*$ *4*) = *NOEX-IST*)
 prefer *2* **apply** (*smt Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*) *Mem-pool-lvl.surjective nth-list-update-eq nth-list-update-neq*)
apply(*subgoal-tac* (*bits* (*levels* (*mem-pool-info Va p*) ! *Suc ii*)) ! (*jj* $*$ *4*)
 = (*bits* (*levels* (*mem-pool-info Vb p*) ! *Suc ii*)) ! (*jj* $*$ *4*))
 prefer *2* **apply**(*case-tac Suc ii* = *l* $\wedge$ *jj* $*$ *4* = *b*) **apply** *simp* **using** *inv-bitmap-presv-setbit-0*
apply *simp*
apply *simp*

apply(*rule conjI*)
apply(*subgoal-tac* (*bits* (*levels* (*mem-pool-info Va p*) ! *Suc ii*)) ! (*jj* $*$ *4* + *1*) = *NOEXIST*)
 prefer *2* **apply** (*smt Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*) *Mem-pool-lvl.surjective*

 Nat.add-0-right One-nat-def add-Suc-right nth-list-update-eq nth-list-update-neq)

apply(*subgoal-tac* (*bits* (*levels* (*mem-pool-info Va p*) ! *Suc ii*)) ! (*jj* $*$ *4* + *1*)
 = (*bits* (*levels* (*mem-pool-info Vb p*) ! *Suc ii*)) ! (*jj* $*$ *4* + *1*))
 prefer *2* **apply**(*case-tac Suc ii* = *l* $\wedge$ *jj* $*$ *4* + *1* = *b*) **apply** *simp* **using** *inv-bitmap-presv-setbit-0* **apply** *metis*
apply *simp*

apply(*rule conjI*)
apply(*subgoal-tac* (*bits* (*levels* (*mem-pool-info Va p*) ! *Suc ii*)) ! (*jj* $*$ *4* + *2*) = *NOEXIST*)
 prefer *2*
 apply (*smt Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*) *Mem-pool-lvl.surjective*

 add-2-eq-Suc′ nth-list-update-eq nth-list-update-neq)
apply(*subgoal-tac* (*bits* (*levels* (*mem-pool-info Va p*) ! *Suc ii*)) ! (*jj* $*$ *4* + *2*)
 = (*bits* (*levels* (*mem-pool-info Vb p*) ! *Suc ii*)) ! (*jj* $*$ *4* + *2*))
 prefer *2* **apply**(*case-tac Suc ii* = *l* $\wedge$ *jj* $*$ *4* + *2* = *b*) **apply** *simp* **using** *inv-bitmap-presv-setbit-0* **apply** *metis*
apply *simp*

apply(*subgoal-tac* (*bits* (*levels* (*mem-pool-info Va p*) ! *Suc ii*)) ! (*jj* $*$ *4* + *3*) = *NOEXIST*)
 prefer *2*
 apply (*smt Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*) *Mem-pool-lvl.surjective*

 add-2-eq-Suc′ nth-list-update-eq nth-list-update-neq)
apply(*subgoal-tac* (*bits* (*levels* (*mem-pool-info Va p*) ! *Suc ii*)) ! (*jj* $*$ *4* + *3*)
 = (*bits* (*levels* (*mem-pool-info Vb p*) ! *Suc ii*)) ! (*jj* $*$ *4* + *3*))
 prefer *2* **apply**(*case-tac Suc ii* = *l* $\wedge$ *jj* $*$ *4* + *3* = *b*) **apply** *simp* **using**

inv-bitmap-presv-setbit-0 **apply** *metis*
apply *simp*

apply(*rule conjI*)

apply *clarify*
apply(*subgoal-tac* (*bits* (*levels* (*mem-pool-info Va p*) ! (*ii* − *1*))) ! (*jj div 4*) = *DIVIDED*)
prefer *2* **apply** (*smt Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*) *Mem-pool-lvl.surjective*

One-nat-def nth-list-update-eq nth-list-update-neq)
apply(*subgoal-tac* (*bits* (*levels* (*mem-pool-info Va p*) ! (*ii* − *1*))) ! (*jj div 4*)
= (*bits* (*levels* (*mem-pool-info Vb p*) ! (*ii* − *1*))) ! (*jj div 4*))
prefer *2* **apply**(*case-tac ii* − *1* = *l* ∧ *jj div 4* = *b*) **apply** *simp* **using** *inv-bitmap-presv-setbit-0*
apply *simp*
apply *simp*

apply(*rule conjI*)

apply *clarify*
apply(*rule conjI*)
apply(*subgoal-tac* (*bits* (*levels* (*mem-pool-info Va p*) ! (*ii* + *1*))) ! (*jj* ∗ *4*) = *NOEXIST*)
prefer *2* **apply** (*smt Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*) *Mem-pool-lvl.surjective*

Nat.add-0-right One-nat-def add-Suc-right nth-list-update-eq nth-list-update-neq)
apply(*subgoal-tac* (*bits* (*levels* (*mem-pool-info Va p*) ! (*ii* + *1*))) ! (*jj* ∗ *4*)
= (*bits* (*levels* (*mem-pool-info Vb p*) ! (*ii* + *1*))) ! (*jj* ∗ *4*))
prefer *2* **apply**(*case-tac ii* + *1* = *l* ∧ *jj* ∗ *4* = *b*) **apply** *simp* **using** *inv-bitmap-presv-setbit-0*
apply *simp*
apply *simp*

apply(*rule conjI*)
apply(*subgoal-tac* (*bits* (*levels* (*mem-pool-info Va p*) ! (*ii* + *1*))) ! (*jj* ∗ *4* + *1*) = *NOEXIST*)
prefer *2* **apply** (*smt Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*) *Mem-pool-lvl.surjective*

Nat.add-0-right One-nat-def add-Suc-right nth-list-update-eq nth-list-update-neq)
apply(*subgoal-tac* (*bits* (*levels* (*mem-pool-info Va p*) ! (*ii* + *1*))) ! (*jj* ∗ *4* + *1*)
= (*bits* (*levels* (*mem-pool-info Vb p*) ! (*ii* + *1*))) ! (*jj* ∗ *4* + *1*))
prefer *2* **apply**(*case-tac ii* + *1* = *l* ∧ *jj* ∗ *4* + *1* = *b*) **apply** *auto*[*1*] **using** *inv-bitmap-presv-setbit-0* **apply** *simp*
apply *simp*

apply(*rule conjI*)
apply(*subgoal-tac* (*bits* (*levels* (*mem-pool-info Va p*) ! (*ii* + *1*))) ! (*jj* ∗ *4* + *2*) =

NOEXIST)
prefer *2* **apply** (*smt Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*) *Mem-pool-lvl.surjective*

Nat.add-0-right One-nat-def add-2-eq-Suc′ add-Suc-right nth-list-update-eq nth-list-update-neq)
apply(*subgoal-tac* (*bits* (*levels* (*mem-pool-info Va p*) ! (*ii* + *1*))) ! (*jj* ∗ *4* + *2*)
= (*bits* (*levels* (*mem-pool-info Vb p*) ! (*ii* + *1*))) ! (*jj* ∗ *4* + *2*))
prefer *2* **apply**(*case-tac ii* + *1* = *l* ∧ *jj* ∗ *4* + *2* = *b*) **apply** *auto*[*1*] **using** *inv-bitmap-presv-setbit-0* **apply** *simp*
apply *simp*

apply(*subgoal-tac* (*bits* (*levels* (*mem-pool-info Va p*) ! (*ii* + *1*))) ! (*jj* ∗ *4* + *3*) = *NOEXIST*)
prefer *2* **apply** (*smt Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*) *Mem-pool-lvl.surjective*

Nat.add-0-right One-nat-def add-2-eq-Suc′ add-Suc-right nth-list-update-eq nth-list-update-neq)
apply(*subgoal-tac* (*bits* (*levels* (*mem-pool-info Va p*) ! (*ii* + *1*))) ! (*jj* ∗ *4* + *3*)
= (*bits* (*levels* (*mem-pool-info Vb p*) ! (*ii* + *1*))) ! (*jj* ∗ *4* + *3*))
prefer *2* **apply**(*case-tac ii* + *1* = *l* ∧ *jj* ∗ *4* + *3* = *b*) **apply** *auto*[*1*] **using** *inv-bitmap-presv-setbit-0* **apply** *simp*
apply *simp*

apply *clarify*

apply(*subgoal-tac bits* (*levels* (*mem-pool-info Va p*) ! *ii*) ! *jj* = *NOEXIST*)
prefer *2* **apply**(*case-tac ii* = *l* ∧ *jj* = *b*) **apply** *auto*[*1*] **using** *inv-bitmap-presv-setbit-0*
apply *simp*
apply(*subgoal-tac bits* (*levels* (*mem-pool-info Va p*) ! (*ii* − *1*)) ! (*jj div 4*) ≠ *DIVIDED*)
prefer *2* **apply** (*smt Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*) *Mem-pool-lvl.surjective*

One-nat-def nth-list-update-eq nth-list-update-neq)
apply(*subgoal-tac* (*bits* (*levels* (*mem-pool-info Va p*) ! (*ii* − *1*))) ! (*jj div 4*)
= *bits* (*levels* (*mem-pool-info Va p*)
[*l* := (*levels* (*mem-pool-info Va p*) ! *l*)(|*bits* := *bits* (*levels* (*mem-pool-info Va p*) ! *l*)[*b* := *st*]|)] !
(*ii* − *1*)) ! (*jj div 4*)) **prefer** *2*
apply(*case-tac ii* − *1* = *l* ∧ *jj div 4* = *b*) **apply** *clarsimp*
apply(*subgoal-tac bits* (*levels* (*mem-pool-info Va p*) ! (*ii* − *Suc NULL*))[*jj div 4* := *st*] ! (*jj div 4*) = *st*)
prefer *2* **apply** (*metis list-update-beyond not-less nth-list-update-eq*)
apply *simp*
using *inv-bitmap-presv-setbit-0* **apply** *simp*
apply *clarsimp*

done

lemma *inv-bitmap-freelist-fl-bnum-in*:
inv-bitmap-freelist Va $\Longrightarrow$
inv-mempool-info Va $\Longrightarrow$
p$\in$*mem-pools Va* $\Longrightarrow$
ii < *length* (*levels* (*mem-pool-info Va p*)) $\Longrightarrow$
jj < *length* (*free-list* ((*levels* (*mem-pool-info Va p*)) ! *ii*)) $\Longrightarrow$
block-num (*mem-pool-info Va p*)
((*free-list* ((*levels* (*mem-pool-info Va p*)) ! *ii*)) ! *jj*)
(*max-sz* (*mem-pool-info Va p*) *div* *4* ^ *ii*) < *length* (*bits* (*levels* (*mem-pool-info Va p*) ! *ii*))
apply(*simp add*:*inv-bitmap-freelist-def inv-mempool-info-def block-num-def Let-def*)
apply(*subgoal-tac* $\exists$ *n*. *n* < *n-max* (*mem-pool-info Va p*) $*$ (*4* ^ *ii*) $\wedge$ *free-list* (*levels* (*mem-pool-info Va p*) ! *ii*) ! *jj*
= *buf* (*mem-pool-info Va p*) + *n* $*$ (*max-sz* (*mem-pool-info Va p*) *div* *4* ^ *ii*))
prefer *2* **apply** *blast*
apply(*subgoal-tac free-list* (*levels* (*mem-pool-info Va p*) ! *ii*) ! *jj* $\geq$ *buf* (*mem-pool-info Va p*))
prefer *2* **apply** *linarith*
using *nonzero-mult-div-cancel-right* **by** *force*

lemma *inv-bitmap-freelist-fl-FREE*:
inv-bitmap-freelist Va $\Longrightarrow$
inv-mempool-info Va $\Longrightarrow$
p$\in$*mem-pools Va* $\Longrightarrow$
ii < *length* (*levels* (*mem-pool-info Va p*)) $\Longrightarrow$
jj < *length* (*free-list* ((*levels* (*mem-pool-info Va p*)) ! *ii*)) $\Longrightarrow$
get-bit-s Va p ii (*block-num* (*mem-pool-info Va p*)
((*free-list* ((*levels* (*mem-pool-info Va p*)) ! *ii*)) ! *jj*)
(*max-sz* (*mem-pool-info Va p*) *div* *4* ^ *ii*)) = *FREE*
apply(*simp add*:*inv-bitmap-freelist-def inv-mempool-info-def block-num-def Let-def*)
apply(*subgoal-tac* $\exists$ *n*. *n* < *n-max* (*mem-pool-info Va p*) $*$ (*4* ^ *ii*) $\wedge$ *free-list* (*levels* (*mem-pool-info Va p*) ! *ii*) ! *jj*
= *buf* (*mem-pool-info Va p*) + *n* $*$ (*max-sz* (*mem-pool-info Va p*) *div* *4* ^ *ii*))
prefer *2* **apply** *blast*
by (*metis add-diff-cancel-left' div-0 mult-0-right neq0-conv nonzero-mult-div-cancel-right not-less-zero nth-mem*)

lemma *inv-buf-le-fl*:
inv-bitmap-freelist Va $\Longrightarrow$
inv-mempool-info Va $\Longrightarrow$
p$\in$*mem-pools Va* $\Longrightarrow$
ii < *length* (*levels* (*mem-pool-info Va p*)) $\Longrightarrow$
jj < *length* (*free-list* ((*levels* (*mem-pool-info Va p*)) ! *ii*)) $\Longrightarrow$
buf (*mem-pool-info Va p*) $\leq$ (*free-list* ((*levels* (*mem-pool-info Va p*)) ! *ii*)) ! *jj*
apply(*simp add*:*inv-bitmap-freelist-def inv-mempool-info-def Let-def*)

apply(*subgoal-tac* $\exists n$. *free-list* (*levels* (*mem-pool-info Va p*) ! *ii*) ! *jj*
= *buf* (*mem-pool-info Va p*) + *n* $*$ (*max-sz* (*mem-pool-info Va p*) *div* *4* ^ *ii*))
prefer *2* **apply** *blast*
by *linarith*

lemma *inv-fl-mod-sz0*:
inv-bitmap-freelist Va $\Longrightarrow$
inv-mempool-info Va $\Longrightarrow$
p$\in$*mem-pools Va* $\Longrightarrow$
ii < *length* (*levels* (*mem-pool-info Va p*)) $\Longrightarrow$
jj < *length* (*free-list* ((*levels* (*mem-pool-info Va p*)) ! *ii*)) $\Longrightarrow$
((*free-list* ((*levels* (*mem-pool-info Va p*)) ! *ii*)) ! *jj* − *buf* (*mem-pool-info Va p*)) *mod*
(*max-sz* (*mem-pool-info Va p*) *div* *4* ^ *ii*) = *0*
apply(*simp add*:*inv-bitmap-freelist-def inv-mempool-info-def Let-def*)
apply(*subgoal-tac* $\exists n$. *free-list* (*levels* (*mem-pool-info Va p*) ! *ii*) ! *jj*
= *buf* (*mem-pool-info Va p*) + *n* $*$ (*max-sz* (*mem-pool-info Va p*) *div* *4* ^ *ii*))
prefer *2* **apply** *blast*
by *force*

lemma *sameinfo-inv-bitmap-mp*:
mem-pool-info Va p = *mem-pool-info Vb p* $\Longrightarrow$ *inv-bitmap-mp Va p* = *inv-bitmap-mp Vb p*
apply(*simp add*: *Let-def*)
done

lemma *sameinfo-inv-bitmap-freelist-mp*:
mem-pool-info Va p = *mem-pool-info Vb p* $\Longrightarrow$ *inv-bitmap-freelist-mp Va p* = *inv-bitmap-freelist-mp Vb p*
apply(*simp add*: *Let-def*)
done

lemma *inv-bitmap-presv-mpls-mpi*:
mem-pools Va = *mem-pools Vb* $\Longrightarrow$
mem-pool-info Va = *mem-pool-info Vb* $\Longrightarrow$
inv-bitmap Va $\Longrightarrow$
inv-bitmap Vb
by(*simp add*:*inv-bitmap-def Let-def*)

lemma *inv-bitmap-presv-mpls-mpi2*:
mem-pools Va = *mem-pools Vb* $\Longrightarrow$
($\forall$ *p*. *length* (*levels* (*mem-pool-info Va p*)) = *length* (*levels* (*mem-pool-info Vb p*))) $\Longrightarrow$
($\forall$ *p ii*. *ii* < *length* (*levels* (*mem-pool-info Va p*))
$\longrightarrow$ *bits* (*levels* (*mem-pool-info Va p*) ! *ii*) = *bits* (*levels* (*mem-pool-info Vb*

p) ! *ii*)) $\Longrightarrow$
  inv-bitmap Va $\Longrightarrow$
  inv-bitmap Vb
by (*simp add*: *inv-bitmap-def Let-def*)

lemma *inv-bitmap-freeing2free*:
inv-bitmap-mp V p $\Longrightarrow$
 $\exists$ *lv bl*. *bits* (*levels* (*mem-pool-info V p*) ! *lv*) ! *bl* = *FREEING*
  $\wedge$ *bits* (*levels* (*mem-pool-info V2 p*) ! *lv*) = *bits* (*levels* (*mem-pool-info V p*) ! *lv*) [*bl* := *FREE*]
  $\wedge$ ($\forall$ *lv'*. *lv* $\neq$ *lv'* $\longrightarrow$ *bits* (*levels* (*mem-pool-info V2 p*) ! *lv'*) = *bits* (*levels* (*mem-pool-info V p*) ! *lv'*)) $\Longrightarrow$
 length (*levels* (*mem-pool-info V p*)) = *length* (*levels* (*mem-pool-info V2 p*))
 $\Longrightarrow$ *inv-bitmap-mp V2 p*
apply(*simp add*:*Let-def*) **apply** *clarify*
apply(*subgoal-tac length* (*bits* (*levels* (*mem-pool-info V p*) ! *i*)) = *length* (*bits* (*levels* (*mem-pool-info V2 p*) ! *i*)))
 prefer *2* **apply**(*case-tac i* = *lv*) **apply** *auto*[*1*] **apply** *auto*[*1*]
apply(*rule conjI*) **apply** *clarify*
 apply(*rule conjI*) **apply** *clarify*
  apply(*case-tac i* = *lv* $\wedge$ *j* = *bl*) **apply** *clarsimp*
   apply(*subgoal-tac get-bit-s V2 p i j* = *get-bit-s V p i j*)
    prefer *2* **apply**(*case-tac i* = *lv*) **apply** *clarsimp* **apply** *presburger*
  apply (*metis BlockState.distinct*(*21*) *nth-list-update-neq*)

 apply *clarify*
  apply(*case-tac i* = *lv* $\wedge$ *j* = *bl*) **apply** *clarsimp*
   apply(*subgoal-tac get-bit-s V2 p i j* = *get-bit-s V p i j*)
    prefer *2* **apply**(*case-tac i* = *lv*) **apply** *clarsimp* **apply** *presburger*
 apply (*smt BlockState.distinct*(*25*) *nth-list-update-neq*)

apply(*rule conjI*) **apply** *clarify*
 apply(*case-tac i* = *lv* $\wedge$ *j* = *bl*) **apply** *clarsimp*
  apply(*subgoal-tac get-bit-s V2 p i j* = *get-bit-s V p i j*)
   prefer *2* **apply**(*case-tac i* = *lv*) **apply** *clarsimp* **apply** *presburger*
 apply (*smt BlockState.distinct*(*21*) *BlockState.distinct*(*25*) *nth-list-update-neq*)

apply(*rule conjI*) **apply** *clarify*
 apply(*case-tac i* = *lv* $\wedge$ *j* = *bl*) **apply** *clarsimp*
  apply(*subgoal-tac get-bit-s V2 p i j* = *get-bit-s V p i j*)
   prefer *2* **apply**(*case-tac i* = *lv*) **apply** *clarsimp* **apply** *presburger*
 apply (*smt BlockState.distinct*(*21*) *BlockState.distinct*(*25*) *nth-list-update-neq*)

apply(*rule conjI*) **apply** *clarify*
 apply(*case-tac i* = *lv* $\wedge$ *j* = *bl*) **apply** *clarsimp*
  apply(*subgoal-tac get-bit-s V2 p i j* = *get-bit-s V p i j*)
   prefer *2* **apply**(*case-tac i* = *lv*) **apply** *clarsimp* **apply** *presburger*
 apply (*smt BlockState.distinct*(*21*) *BlockState.distinct*(*25*) *nth-list-update-neq*)

apply(*rule conjI*) **apply** *clarify*
  apply(*case-tac* $i = lv \wedge j = bl$) **apply** *clarsimp*
    apply(*subgoal-tac get-bit-s V2 p i j = get-bit-s V p i j*)
      prefer *2* **apply**(*case-tac* $i = lv$) **apply** *clarsimp* **apply** *presburger*
  apply (*smt BlockState.distinct*(*21*) *BlockState.distinct*(*25*) *nth-list-update-neq*)

apply(*rule conjI*) **apply** *clarify*
  apply(*case-tac* $i = lv \wedge j = bl$) **apply** *clarsimp*
    apply(*subgoal-tac get-bit-s V2 p i j = get-bit-s V p i j*)
      prefer *2* **apply**(*case-tac* $i = lv$) **apply** *clarsimp* **apply** *presburger*
  apply (*smt BlockState.distinct*(*21*) *BlockState.distinct*(*25*) *nth-list-update-neq*)

apply *clarify*
  apply(*case-tac* $i = lv \wedge j = bl$) **apply** *clarsimp*
    apply(*subgoal-tac get-bit-s V2 p i j = get-bit-s V p i j*)
      prefer *2* **apply**(*case-tac* $i = lv$) **apply** *clarsimp* **apply** *presburger*
  apply (*smt BlockState.distinct*(*11*) *list-update-beyond not-less nth-list-update-eq nth-list-update-neq*)

done

lemma *inv-bitmap-allocating2allocate*:
inv-bitmap-mp V p $\Longrightarrow$
  $\exists$ *lv bl*. *bits* (*levels* (*mem-pool-info V p*) ! *lv*) ! *bl* = *ALLOCATING*
          $\wedge$ *bits* (*levels* (*mem-pool-info V2 p*) ! *lv*) = *bits* (*levels* (*mem-pool-info V p*) ! *lv*) [*bl* := *ALLOCATED*]
        $\wedge$ ($\forall$ *lv'*. *lv* $\neq$ *lv'* $\longrightarrow$ *bits* (*levels* (*mem-pool-info V2 p*) ! *lv'*) = *bits* (*levels* (*mem-pool-info V p*) ! *lv'*)) $\Longrightarrow$
  length (*levels* (*mem-pool-info V p*)) = *length* (*levels* (*mem-pool-info V2 p*))
  $\Longrightarrow$ *inv-bitmap-mp V2 p*
apply(*simp add*:*Let-def*) **apply** *clarify*
apply(*subgoal-tac length* (*bits* (*levels* (*mem-pool-info V p*) ! *i*)) = *length* (*bits* (*levels* (*mem-pool-info V2 p*) ! *i*)))
  prefer *2* **apply**(*case-tac* $i = lv$) **apply** *auto*[*1*] **apply** *auto*[*1*]
apply(*rule conjI*) **apply** *clarify*
  apply(*rule conjI*) **apply** *clarify*
    apply(*case-tac* $i = lv \wedge j = bl$) **apply** *clarsimp*
      apply(*subgoal-tac get-bit-s V2 p i j = get-bit-s V p i j*)
        prefer *2* **apply**(*case-tac* $i = lv$) **apply** *clarsimp* **apply** *presburger*
    apply (*metis BlockState.distinct*(*23*) *nth-list-update-neq*)

  apply *clarify*
    apply(*case-tac* $i = lv \wedge j = bl$) **apply** *clarsimp*
      apply(*subgoal-tac get-bit-s V2 p i j = get-bit-s V p i j*)
        prefer *2* **apply**(*case-tac* $i = lv$) **apply** *clarsimp* **apply** *presburger*
  apply (*smt BlockState.distinct*(*27*) *nth-list-update-neq*)

apply(*rule conjI*) **apply** *clarify*

apply(*case-tac* $i = lv \wedge j = bl$) **apply** *clarsimp*
apply(*subgoal-tac get-bit-s V2 p i j* = *get-bit-s V p i j*)
prefer *2* **apply**(*case-tac* $i = lv$) **apply** *clarsimp* **apply** *presburger*
apply (*smt BlockState.distinct*(*23*) *BlockState.distinct*(*27*) *nth-list-update-neq*)

apply(*rule conjI*) **apply** *clarify*
apply(*case-tac* $i = lv \wedge j = bl$) **apply** *clarsimp*
apply(*subgoal-tac get-bit-s V2 p i j* = *get-bit-s V p i j*)
prefer *2* **apply**(*case-tac* $i = lv$) **apply** *clarsimp* **apply** *presburger*
apply (*smt BlockState.distinct*(*23*) *BlockState.distinct*(*27*) *nth-list-update-neq*)

apply(*rule conjI*) **apply** *clarify*
apply(*case-tac* $i = lv \wedge j = bl$) **apply** *clarsimp*
apply(*subgoal-tac get-bit-s V2 p i j* = *get-bit-s V p i j*)
prefer *2* **apply**(*case-tac* $i = lv$) **apply** *clarsimp* **apply** *presburger*
apply (*smt BlockState.distinct*(*23*) *BlockState.distinct*(*27*) *nth-list-update-neq*)

apply(*rule conjI*) **apply** *clarify*
apply(*case-tac* $i = lv \wedge j = bl$) **apply** *clarsimp*
apply(*subgoal-tac get-bit-s V2 p i j* = *get-bit-s V p i j*)
prefer *2* **apply**(*case-tac* $i = lv$) **apply** *clarsimp* **apply** *presburger*
apply (*smt BlockState.distinct*(*23*) *BlockState.distinct*(*27*) *nth-list-update-neq*)

apply(*rule conjI*) **apply** *clarify*
apply(*case-tac* $i = lv \wedge j = bl$) **apply** *clarsimp*
apply(*subgoal-tac get-bit-s V2 p i j* = *get-bit-s V p i j*)
prefer *2* **apply**(*case-tac* $i = lv$) **apply** *clarsimp* **apply** *presburger*
apply (*smt BlockState.distinct*(*23*) *BlockState.distinct*(*27*) *nth-list-update-neq*)

apply *clarify*
apply(*case-tac* $i = lv \wedge j = bl$) **apply** *clarsimp*
apply(*subgoal-tac get-bit-s V2 p i j* = *get-bit-s V p i j*)
prefer *2* **apply**(*case-tac* $i = lv$) **apply** *clarsimp* **apply** *presburger*
apply (*smt BlockState.distinct*(*3*) *list-update-beyond not-less nth-list-update-eq nth-list-update-neq*)
done

lemma *inv-bitmap-freelist-presv-setbit-notfree-h*:
$\neg$ ($x = lv \wedge y = bkn$) $\Longrightarrow$
Vb = *set-bit-s V p lv bkn st* $\Longrightarrow$
get-bit-s V p x y= *get-bit-s Vb p x y*
apply(*simp add*:*set-bit-s-def set-bit-def*)
by (*metis* (*no-types*, *lifting*) *Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*)
Mem-pool-lvl.surjective list-update-beyond not-less nth-list-update-eq nth-list-update-neq)

lemma *inv-bitmap-freelist-presv-setbit-notfree*:

$p \in mem\text{-}pools\ V \Longrightarrow$
  $inv\text{-}mempool\text{-}info\ V \wedge inv\text{-}aux\text{-}vars\ V \wedge inv\text{-}bitmap\text{-}freelist\ V \Longrightarrow$
  $st \neq FREE \Longrightarrow$
  $get\text{-}bit\text{-}s\ V\ p\ lv\ bkn \neq FREE \Longrightarrow$
  $inv\text{-}bitmap\text{-}freelist\ (set\text{-}bit\text{-}s\ V\ p\ lv\ bkn\ st)$
apply(*simp add:inv-bitmap-freelist-def*) **apply**(*simp add:set-bit-s-def set-bit-def*)
apply(*simp add:Let-def*)
apply *clarsimp* **apply**(*rename-tac ii*)
apply(*rule conjI*)
  apply *clarsimp* **apply**(*rename-tac jj*)
  apply(*subgoal-tac length (bits (levels (mem-pool-info V p)*
                              [lv := (levels (mem-pool-info V p) ! lv)(|bits := bits (levels
(mem-pool-info V p) ! lv)[bkn := st]|)] ! ii))
                      = length (bits (levels (mem-pool-info V p) ! ii))) **prefer** *2*
    apply(*case-tac ii = lv*) **apply** *fastforce* **apply** *fastforce*
  apply(*case-tac ii = lv ∧ jj = bkn*)
    using *inv-bitmap-freelist-presv-setbit-notfree-h* **apply** *force*
   apply(*subgoal-tac (bits (levels (mem-pool-info V p)[lv := (levels (mem-pool-info*
V p) ! lv)
                          (|bits := bits (levels (mem-pool-info V p) ! lv)[bkn := st]|)] !
ii)) ! jj =
                        (bits (levels (mem-pool-info V p) ! ii)) ! jj) **prefer** *2*
      apply(*case-tac ii ≠ lv*) **apply** *fastforce*
      apply(*case-tac jj ≠ bkn*) **apply** *fastforce*
      apply *fastforce*
   apply(*subgoal-tac free-list (levels (mem-pool-info V p)[lv := (levels (mem-pool-info*
V p) ! lv)
                        (|bits := bits (levels (mem-pool-info V p) ! lv)[bkn := st]|)] ! ii)
                    = free-list (levels (mem-pool-info V p) ! ii)) **prefer** *2*
      apply(*case-tac ii ≠ lv*) **apply** *fastforce* **apply** *fastforce*
    apply *auto[1]*

apply(*rule conjI*)
apply *clarsimp* **apply**(*rename-tac jj*)
  apply(*subgoal-tac length (bits (levels (mem-pool-info V p)*
                              [lv := (levels (mem-pool-info V p) ! lv)(|bits := bits (levels
(mem-pool-info V p) ! lv)[bkn := st]|)] ! ii))
                      = length (bits (levels (mem-pool-info V p) ! ii))) **prefer** *2*
    apply(*case-tac ii = lv*) **apply** *fastforce* **apply** *fastforce*
 apply(*subgoal-tac free-list (levels (mem-pool-info V p)[lv := (levels (mem-pool-info*
V p) ! lv)
                        (|bits := bits (levels (mem-pool-info V p) ! lv)[bkn := st]|)] ! ii)
                    = free-list (levels (mem-pool-info V p) ! ii)) **prefer** *2*
      apply(*case-tac ii ≠ lv*) **apply** *fastforce* **apply** *fastforce*
  apply *auto[1]*

apply(*subgoal-tac free-list (levels (mem-pool-info V p)[lv := (levels (mem-pool-info*
V p) ! lv)

$(\!|bits := bits\ (levels\ (mem\text{-}pool\text{-}info\ V\ p)\ !\ lv)[bkn := st]|\!)]\ !\ ii)$
$= free\text{-}list\ (levels\ (mem\text{-}pool\text{-}info\ V\ p)\ !\ ii))$ **prefer** *2*
apply(*case-tac ii* $\neq$ *lv*) **apply** *fastforce* **apply** *fastforce*
apply *auto*[*1*]
done

end

theory *memory-cover*
imports *invariant*
begin

4 partition of memory addresses of a pool

declare [[*smt-timeout* = *300*]]

this theory shows that all memory blocks are a COVER of address space of a memory pool. A COVER means blocks are disjoint and continuous. It means that for any memory address of a memory pool, the address is in the address range of only one block.

Due to algorithm, address range of each block is implicitly derived. address range of a block at level ii with block index jj at this level is jj * (max_sz mp div (4^ii)) ..¡ Suc jj * (max_sz mp div (4^ii)).

abbreviation *addr-in-block mp addr ii jj* ≡
ii < *length* (*levels mp*) ∧ *jj* < *length* (*bits* (*levels mp* ! *ii*))
∧ (*bits* (*levels mp* ! *ii*) ! *jj* = *FREE* ∨ *bits* (*levels mp* ! *ii*) ! *jj* = *FREEING* ∨ *bits* (*levels mp* ! *ii*) ! *jj* = *ALLOCATED* ∨ *bits* (*levels mp* ! *ii*) ! *jj* = *ALLOCATING*)
∧ *addr* ∈ {*jj* ∗ (*max-sz mp div* (*4* ˆ *ii*)) ..< *Suc jj* ∗ (*max-sz mp div* (*4* ˆ *ii*))}

abbreviation *mem-cover-mp* :: *State* ⇒ *mempool-ref* ⇒ *bool*
where *mem-cover-mp s p* ≡
let mp = *mem-pool-info s p in* (∀ *addr* < *n-max mp* ∗ *max-sz mp*. (∃!(*i*,*j*). *addr-in-block mp addr i j*))

definition *mem-cover* :: *State* ⇒ *bool*
where *mem-cover s* ≡ ∀ *p*∈*mem-pools s*. *mem-cover-mp s p*

lemma *split-div-lemma*:
assumes *0* < *n*
shows *n* ∗ *q* ≤ *m* ∧ *m* < *n* ∗ *Suc q* ⟷ *q* = ((*m*::*nat*) *div n*) (**is** *?lhs* ⟷ *?rhs*)
proof
assume *?rhs*
with *minus-mod-eq-mult-div* [*symmetric*] **have** *nq*: *n* ∗ *q* = *m* − (*m mod n*) **by** *simp*
then have *A*: *n* ∗ *q* ≤ *m* **by** *simp*

have $n - (m \ mod \ n) > 0$ **using** *mod-less-divisor assms* **by** *auto*
then have $m < m + (n - (m \ mod \ n))$ **by** *simp*
then have $m < n + (m - (m \ mod \ n))$ **by** *simp*
with *nq* **have** $m < n + n * q$ **by** *simp*
then have B: $m < n * Suc \ q$ **by** *simp*
from *A B* **show** *?lhs* **..**
next
assume *P*: *?lhs*
then show *?rhs*
using *div-nat-eqI* **by** *blast*
qed

lemma *align-up-ge-low*:
$sz1 > 0 \Longrightarrow sz2 > sz1 \Longrightarrow sz2 \ mod \ sz1 = 0 \Longrightarrow (addr::nat) \ div \ sz2 * sz2 + sz2 \geq addr \ div \ sz1 * sz1 + sz1$
apply(*subgoal-tac* $\exists n>0. \ sz2 = n * sz1$) **prefer** *2* **apply** *auto*[*1*]
apply(*rule subst*[**where** $s=addr - addr \ mod \ sz1$ **and** $t=addr \ div \ sz1 * sz1$])
using *minus-mod-eq-div-mult* **apply** *auto*[*1*]
apply(*rule subst*[**where** $s=addr - addr \ mod \ sz2$ **and** $t=addr \ div \ sz2 * sz2$])
using *minus-mod-eq-div-mult* **apply** *auto*[*1*]
apply(*subgoal-tac* $sz1 - addr \ mod \ sz1 \leq sz2 - addr \ mod \ sz2$)
apply(*subgoal-tac* $addr \ mod \ sz1 < sz1$) **prefer** *2* **apply** *simp*
apply(*subgoal-tac* $addr \ mod \ sz2 < sz2$) **prefer** *2* **apply** *simp*
apply *simp*

apply *clarsimp*
apply(*case-tac* $addr \ mod \ (sz1 * n) \geq sz1 * (n - 1)$)
apply(*subgoal-tac* $addr \ mod \ (sz1 * n) = sz1 * (n - 1) + addr \ mod \ sz1$)
prefer *2* **using** *Suc-lessD Suc-pred' mod-less-divisor*
mult-div-mod-eq nat-0-less-mult-iff mod-mult-self4 split-div-lemma
apply (*metis mod-mult2-eq*)
apply *clarsimp*
apply (*metis* (*no-types*, *lifting*) *Nat.diff-diff-right One-nat-def Suc-lessD add-diff-cancel-left'*

le-add1 less-numeral-extra(*3*) *less-or-eq-imp-le mult.commute mult-eq-if*)
by (*metis* (*no-types*, *lifting*) *Nat.le-diff-conv2 Suc-lessD add-mono-thms-linordered-semiring*(*1*)

diff-le-self less-numeral-extra(*3*) *mod-le-divisor mult.commute mult-eq-if*
mult-pos-pos nat-le-linear)

lemma *addr-exist-block-h1-1*:
$li < ii \Longrightarrow ii < nl \Longrightarrow (4::nat) \ \hat{} \ nl \ div \ 4 \ \hat{} \ ii < 4 \ \hat{} \ nl \ div \ 4 \ \hat{} \ li$
apply(*rule subst*[**where** $s=4 \ \hat{} \ (nl - ii)$ **and** $t=4 \ \hat{} \ nl \ div \ 4 \ \hat{} \ ii$])
apply (*simp add*: *div2-eq-minus*)
apply(*rule subst*[**where** $s=4 \ \hat{} \ (nl - li)$ **and** $t=4 \ \hat{} \ nl \ div \ 4 \ \hat{} \ li$])
apply (*simp add*: *div2-eq-minus*)
apply *auto*
done

lemma *mod-time*: $(x::nat)$ *mod* $m = 0 \Longrightarrow n * x$ *mod* $(n * m) = 0$
by *simp*

lemma *addr-exist-block-h1*:
$li < ii \Longrightarrow$
$\exists n>0.\ msz = (4 * n) * (4\ \hat{}\ nl) \Longrightarrow$
$ii < nl \Longrightarrow$
Suc (*addr div* (*msz div* 4 ˆ *ii*)) ∗ (*msz div* 4 ˆ *ii*)
≤ *Suc* (*addr div* (*msz div* 4 ˆ *ii*) *div* 4 ˆ (*ii* − *li*)) ∗ (*msz div* 4 ˆ *li*)
apply(*rule subst*[**where** *s*=(*addr div* (*msz div* 4 ˆ *ii*)) ∗ (*msz div* 4 ˆ *ii*) + (*msz div* 4 ˆ *ii*)
and *t*=*Suc* (*addr div* (*msz div* 4 ˆ *ii*)) ∗ (*msz div* 4 ˆ *ii*)]) **apply** *auto*[*1*]
apply(*rule subst*[**where** *s*=(*addr div* (*msz div* 4 ˆ *ii*) *div* 4 ˆ (*ii* − *li*)) ∗ (*msz div* 4 ˆ *li*) + (*msz div* 4 ˆ *li*)
and *t*=*Suc* (*addr div* (*msz div* 4 ˆ *ii*) *div* 4 ˆ (*ii* − *li*)) ∗ (*msz div* 4 ˆ *li*)]) **apply** *auto*[*1*]
apply(*rule subst*[**where** *s*=*addr div* (*msz div* 4 ˆ *li*) **and** *t*=*addr div* (*msz div* 4 ˆ *ii*) *div* 4 ˆ (*ii* − *li*)])
apply(*rule subst*[**where** *s*=*addr div* (*msz div* 4 ˆ *ii* ∗ 4 ˆ (*ii* − *li*)) **and** *t*=*addr div* (*msz div* 4 ˆ *ii*) *div* 4 ˆ (*ii* − *li*)])
using *div2-eq-divmul*[*of addr msz div* 4 ˆ *ii* 4 ˆ (*ii* − *li*)] **apply** *simp*
apply(*rule subst*[**where** *s*=*msz div* 4 ˆ *li* **and** *t*=*msz div* 4 ˆ *ii* ∗ 4 ˆ (*ii* − *li*)])
apply(*subgoal-tac msz mod* 4 ˆ *ii* = *0*) **prefer** *2*
using *ge-pow-mod-0* **apply** *auto*[*1*]
apply (*smt add-diff-inverse-nat less-imp-le-nat mod-div-self mult.left-commute nonzero-mult-div-cancel-left not-less power-add power-not-zero rel-simps*(*76*))
apply *fast*

apply(*rule align-up-ge-low*[*of msz div* 4 ˆ *ii msz div* 4 ˆ *li addr*])
apply (*metis ge-pow-mod-0 mod-div-self nat-0-less-mult-iff zero-less-numeral zero-less-power*)
apply *clarsimp* **apply**(*subgoal-tac* 4 ˆ *nl div* 4 ˆ *ii* < 4 ˆ *nl div* 4 ˆ *li*)
prefer *2* **using** *addr-exist-block-h1-1*[*of li ii nl*] **apply** *simp*
using *m-mod-div pow-mod-0* **apply** *auto*[*1*]
apply *clarsimp* **using** *mod-time*[*of* 4 ˆ *nl div* 4 ˆ *li* 4 ˆ *nl div* 4 ˆ *ii*]
by (*smt less-imp-add-positive mod-div-self mod-mult-self1-is-0 mult.left-commute*

nonzero-mult-div-cancel-left power-add power-not-zero zero-neq-numeral)

lemma *divornoe-imp-div-noe-neigh*:
∀ *li*≤*ii*. *get-bit-s s p li* (*jj div* 4 ˆ (*ii* − *li*)) = *DIVIDED* ∨ *get-bit-s s p li* (*jj div* 4 ˆ (*ii* − *li*)) = *NOEXIST* ⟹
get-bit-s s p NULL (*jj div* 4 ˆ *ii*) = *DIVIDED* ⟹
get-bit-s s p ii jj = *NOEXIST* ⟹
ii > *0* ⟹
∃ *n*. *n* > *0* ∧ *n*≤*ii* ∧ *get-bit-s s p* (*n*−*1*) (*jj div* 4 ˆ (*ii* − (*n*−*1*))) = *DIVIDED* ∧

get-bit-s s p n (jj div 4 ^ (ii − n)) = NOEXIST
apply(*induction ii arbitrary*: *jj*)
apply *simp*

apply(*case-tac get-bit-s s p ii (jj div 4) = DIVIDED*)
apply *auto*[*1*]

apply(*subgoal-tac get-bit-s s p ii (jj div 4) = NOEXIST*)
prefer *2*
apply (*metis One-nat-def Suc-diff-Suc diff-self-eq-0 lessI less-imp-le-nat power-one-right*)

apply(*case-tac ii = 0*) **apply** *auto*[*1*]

apply(*subgoal-tac* ∀ *li*≤*ii*. *get-bit-s s p li ((jj div 4) div 4 ^ (ii − li)) = DIVIDED*

∨ *get-bit-s s p li ((jj div 4) div 4 ^ (ii − li)) = NOEXIST*)
prefer *2* **apply** *clarsimp*
apply (*metis Suc-diff-le div-mult2-eq le-SucI power-Suc*)

apply(*subgoal-tac* ∃ *n*>*NULL*. *n* ≤ *ii* ∧
get-bit-s s p (n − 1) ((jj div 4) div 4 ^ (ii − (n − 1))) = DIVIDED ∧
get-bit-s s p n ((jj div 4) div 4 ^ (ii − n)) = NOEXIST)
prefer *2* **apply** (*simp add*: *Divides.div-mult2-eq*)
proof −
fix *iia* :: *nat* **and** *jja* :: *nat*
assume ∃ *n*>*NULL*. *n* ≤ *iia* ∧ *get-bit-s s p (n − 1) (jja div 4 div 4 ^ (iia − (n − 1))) = DIVIDED*
∧ *get-bit-s s p n (jja div 4 div 4 ^ (iia − n)) = NOEXIST*
then obtain *nn* :: *nat* **where**
f1: *NULL* < *nn* ∧ *nn* ≤ *iia* ∧ *get-bit-s s p nn (jja div 4 div 4 ^ (iia − nn)) = NOEXIST*
∧ *get-bit-s s p (nn − 1) (jja div 4 div 4 ^ (iia − (nn − 1))) = DIVIDED*
by *meson*
then have *f2*: *get-bit-s s p nn (jja div 4 ^ Suc (iia − nn)) = NOEXIST*
by (*metis* (*no-types*) *div-mult2-eq semiring-normalization-rules*(*27*))
have *f3*: *get-bit-s s p (nn − 1) (jja div 4 ^ Suc (Suc (iia − nn))) = DIVIDED*
using *f1* **by** (*metis* (*no-types*) *Suc-diff-eq-diff-pred Suc-diff-le div-mult2-eq semiring-normalization-rules*(*27*))
have *nn* ≤ *iia* ∧ *NULL* < *nn*
using *f1* **by** *meson*
then show ∃ *n*>*NULL*. *n* ≤ *Suc iia* ∧ *get-bit-s s p (n − 1) (jja div 4 ^ (Suc iia − (n − 1))) = DIVIDED*
∧ *get-bit-s s p n (jja div 4 ^ (Suc iia − n)) = NOEXIST*
using *f3 f2 Suc-diff-le le-Suc-eq* **by** *auto*
qed

lemma *addr-exist-block*:
assumes
p2: *inv-bitmap0 s* **and**
p3: *inv-bitmap s* **and**
p6: *inv-mempool-info s* **and**
p4: *p* ∈ *mem-pools s* **and**
p7: *inv-bitmapn s* **and**
p5: *addr* < *n-max* (*mem-pool-info s p*) ∗ *max-sz* (*mem-pool-info s p*)
shows ∃ *i j*. *addr-in-block* (*mem-pool-info s p*) *addr i j*
proof −
obtain *ii* **where** *ii*: *ii* = *length* (*levels* (*mem-pool-info s p*)) − *1* **by** *auto*
obtain *jj* **where** *jj*: *jj* = *addr div* (*max-sz* (*mem-pool-info s p*) *div* (*4* ˆ *ii*)) **by** *auto*
have *bits-len-nmax*: ∀ *i*<*length* (*levels* (*mem-pool-info s p*)). *length* (*bits* (*levels* (*mem-pool-info s p*) ! *i*)) = (*n-max* (*mem-pool-info s p*)) ∗ *4* ˆ *i*
using *p6 p4* **by**(*simp add*:*inv-mempool-info-def Let-def*)
have *maxsz*: ∃ *n*>*0*. *max-sz* (*mem-pool-info s p*) = (*4* ∗ *n*) ∗ (*4* ˆ *n-levels* (*mem-pool-info s p*))
using *p4 p6* **apply**(*simp add*:*inv-mempool-info-def Let-def*) **by** *auto*
have *nl-eq-len*: *n-levels* (*mem-pool-info s p*) = *length* (*levels* (*mem-pool-info s p*))
using *p4 p6* **by**(*simp add*:*inv-mempool-info-def Let-def*)
from *ii* **have** *ii-len*: *ii* < *length* (*levels* (*mem-pool-info s p*))
by (*metis diff-less inv-mempool-info-def length-greater-0-conv p4 p6 rel-simps*(*68*))
from *ii p6* **have** *blk-ii*: *max-sz* (*mem-pool-info s p*) *div 4* ˆ *ii* > *0*
by (*smt Euclidean-Division.div-eq-0-iff divisors-zero gr0I ii-len less-imp-le-nat m-mod-div maxsz mod-if nl-eq-len pow-mod-0 power-not-zero zero-neq-numeral*)
hence *addr-ran*: *addr* ∈ {*jj* ∗ (*max-sz* (*mem-pool-info s p*) *div* (*4* ˆ *ii*)) ..< *Suc jj* ∗ (*max-sz* (*mem-pool-info s p*) *div* (*4* ˆ *ii*))}
using *jj div-in-suc*[*of max-sz* (*mem-pool-info s p*) *div 4* ˆ *ii jj addr*] **by** *blast*
have *jj-lt-maxdiv4ii*: *jj* < *n-max* (*mem-pool-info s p*) ∗ *4* ˆ *ii*
apply(*rule subst*[**where** *s*=*addr div* (*max-sz* (*mem-pool-info s p*) *div 4* ˆ *ii*) **and** *t*=*jj*]) **using** *jj* **apply** *fast*
apply(*rule subst*[**where** *s*=*n-max* (*mem-pool-info s p*) ∗ *max-sz* (*mem-pool-info s p*) *div* (*max-sz* (*mem-pool-info s p*) *div 4* ˆ *ii*)
and *t*=*n-max* (*mem-pool-info s p*) ∗ *4* ˆ *ii*]) **using** *ii-len maxsz*
apply (*metis* (*no-types*, *lifting*) *blk-ii ge-pow-mod-0 inv-mempool-info-def m-mod-div*
mod-div-self mod-mult-self1-is-0 neq0-conv nonzero-mult-div-cancel-left p4 p6)
apply(*rule mod-div-gt*[*of addr n-max* (*mem-pool-info s p*) ∗ *max-sz* (*mem-pool-info s p*)
max-sz (*mem-pool-info s p*) *div 4* ˆ *ii*]) **using** *p5* **apply** *fast*
using *maxsz nl-eq-len*
apply (*metis ge-pow-mod-0 ii-len mod-div-self mod-mult-right-eq mod-mult-self1-is-0 mult-0-right*)

```
   done
  have lvlii-eq-last: levels (mem-pool-info s p) ! ii = last (levels (mem-pool-info s
p))
     apply(subgoal-tac length (levels (mem-pool-info s p)) > 0)
     prefer 2 using p4 p6 ii jj-lt-maxdiv4ii p4 p6 ii-len apply(simp add:inv-mempool-info-def
Let-def)
     using ii apply clarsimp
     by (simp add: last-conv-nth)
  have jj-lt-len-lstbits: jj < length (bits (last (levels (mem-pool-info s p))))
      using ii jj-lt-maxdiv4ii p4 p6 ii-len apply(simp add:inv-mempool-info-def
Let-def)
      apply(subgoal-tac length (bits (levels (mem-pool-info s p) ! ii)) = n-max
(mem-pool-info s p) * 4 ^ ii)
      prefer 2 apply auto[1]
    apply(subgoal-tac levels (mem-pool-info s p) ! ii = last (levels (mem-pool-info
s p)))
      prefer 2 apply(subgoal-tac length (levels (mem-pool-info s p)) > 0)
      prefer 2 using p4 p6 apply(simp add:inv-mempool-info-def Let-def) apply
clarsimp
      apply (simp add: last-conv-nth)
    by fastforce

  have ∃ li≤ii. addr-in-block (mem-pool-info s p) addr li (jj div 4 ^ (ii − li))
    proof −
    {
     assume asm: ¬ (∃ li≤ii. addr-in-block (mem-pool-info s p) addr li (jj div 4 ^
(ii − li)))

      from asm have ∀ li≤ii. ¬ addr-in-block (mem-pool-info s p) addr li (jj div 4
^ (ii − li)) by fast
      moreover
      from ii have ii-len: ii < length (levels (mem-pool-info s p))
      by (metis diff-less inv-mempool-info-def length-greater-0-conv p4 p6 rel-simps(68))
      moreover
      have ∀ li≤ii. addr ∈ {jj div 4 ^ (ii − li) * (max-sz (mem-pool-info s p) div
4 ^ li)..<
                          Suc (jj div 4 ^ (ii − li)) * (max-sz (mem-pool-info s p) div
4 ^ li)}
          apply(subgoal-tac ∃ n>0. max-sz (mem-pool-info s p) = (4 * n) * (4 ^
n-levels (mem-pool-info s p)))
            prefer 2 using p4 p6 apply(simp add:inv-mempool-info-def Let-def)
apply auto[1]
      apply(subgoal-tac n-levels (mem-pool-info s p) = length (levels (mem-pool-info
s p)) ∧
                          length (levels (mem-pool-info s p)) > 0)
            prefer 2 using p4 p6 apply(simp add:inv-mempool-info-def Let-def)
apply auto[1]
        apply clarify apply auto
        apply(subgoal-tac jj * (max-sz (mem-pool-info s p) div 4 ^ ii)
```

$\geq$ *jj div 4 ^ (ii − li) ∗ (max-sz (mem-pool-info s p) div 4 ^ li))*
prefer *2* **apply**(*case-tac li = ii*) **apply** *auto*[*1*]
using *Divides.div-mult2-eq Groups.mult-ac*(*2*) *blk-ii add-diff-inverse-nat calculation*(*2*)
div-mult-self-is-m divisors-zero ge-pow-mod-0 mod-div-self neq0-conv not-less power-add
semiring-normalization-rules(*17*) *split-div-lemma zero-less-numeral zero-less-power*
apply (*smt div-mult-self1-is-m nat-mult-le-cancel-disj*)
using *addr-ran* **apply** *auto*[*1*]

apply(*subgoal-tac Suc jj ∗ (max-sz (mem-pool-info s p) div (4 ^ ii))*
$\leq$ *Suc (jj div 4 ^ (ii − li)) ∗ (max-sz (mem-pool-info s p) div 4 ^ li))*
prefer *2* **apply**(*case-tac li = ii*) **apply** *simp*
apply(*rule subst*[**where** *s=addr div (max-sz (mem-pool-info s p) div (4 ^ ii))* **and** *t=jj*]) **using** *jj* **apply** *fast*
using *addr-exist-block-h1*[*of - ii max-sz (mem-pool-info s p) n-levels (mem-pool-info s p) addr*]
ii-len nl-eq-len maxsz **apply** *fastforce*
using *addr-ran ii-len* **apply** *auto*[*1*]
done
moreover
have *li-len*: ∀ *li*≤*ii. jj div 4 ^ (ii − li) < length (bits (levels (mem-pool-info s p) ! li))*
apply *clarsimp*
apply(*subgoal-tac length (bits (levels (mem-pool-info s p) ! li)) = (n-max (mem-pool-info s p)) ∗ 4 ^ li*)
prefer *2* **using** *p4 p6 ii-len* **apply**(*simp add*:*inv-mempool-info-def Let-def*)
using *jj maxsz nl-eq-len jj-lt-maxdiv4ii Divides.div-mult2-eq add-diff-cancel-left′ blk-ii div-eq-0-iff gr-implies-not0*
le-Suc-ex less-not-refl2 mult.commute mult.left-commute mult-0 mult-is-0 p5 power-add
by (*smt not-less*)
ultimately have ∀ *li*≤*ii.* ¬ (*get-bit-s s p li (jj div 4 ^ (ii − li)) = FREE* ∨
get-bit-s s p li (jj div 4 ^ (ii − li)) = FREEING ∨
get-bit-s s p li (jj div 4 ^ (ii − li)) = ALLOCATED ∨ *get-bit-s s p li (jj div 4 ^ (ii − li)) = ALLOCATING*)
by *auto*
hence *all-dv-ne*: ∀ *li*≤*ii. get-bit-s s p li (jj div 4 ^ (ii − li)) = DIVIDED* ∨ *get-bit-s s p li (jj div 4 ^ (ii − li)) = NOEXIST*
using *BlockState.exhaust* **by** *blast*
moreover
have *bit-lvl0*: *get-bit-s s p 0 (jj div 4 ^ ii) = DIVIDED* **using** *all-dv-ne p2 p4* **apply**(*simp add*:*inv-bitmap0-def Let-def*)
using *li-len* **by** *fastforce*
moreover
have *bit-lvln*: *get-bit-s s p ii jj = NOEXIST*
using *all-dv-ne p4 p7* **apply**(*simp add*:*inv-bitmapn-def inv-bitmap-not4free-def*

Let-def)
using *jj-lt-len-lstbits ii lvlii-eq-last*
by (*metis One-nat-def diff-self-eq-0 div-by-Suc-0 eq-imp-le power-0*)
ultimately have $\exists n.\ n > 0 \wedge n \leq ii \wedge$ *get-bit-s s p* $(n-1)$ $(jj\ div\ 4\ \hat{}\ (ii - (n-1))) = DIVIDED \wedge$
get-bit-s s p n $(jj\ div\ 4\ \hat{}\ (ii - n)) = NOEXIST$
using *divornoe-imp-div-noe-neigh*[*of ii s p jj*] **by** *fastforce*

then obtain *n* **where** $n > 0 \wedge n \leq ii \wedge$ *get-bit-s s p* $(n-1)$ $(jj\ div\ 4\ \hat{}\ (ii - (n-1))) = DIVIDED \wedge$
get-bit-s s p n $(jj\ div\ 4\ \hat{}\ (ii - n)) = NOEXIST$ **by** *auto*
moreover
with *p3* **have** *get-bit-s s p* $(n - Suc\ NULL)$ $(jj\ div\ 4\ \hat{}\ (ii - (n - Suc\ NULL))) \neq DIVIDED$
apply(*simp add*:*inv-bitmap-def Let-def*)
using *Divides.div-mult2-eq One-nat-def Suc-diff-eq-diff-pred Suc-pred diff-Suc-eq-diff-pred diff-commute ii less-Suc-eq-le li-len p4 power-minus-mult zero-less-diff*
by (*smt le-imp-less-Suc zero-le-numeral*)
ultimately have *False* **by** *simp*

} **thus** *?thesis* **by** *auto*
qed

thus *?thesis* **by** *auto*
qed

lemma *div-imp-up-alldiv*:
$\forall$ *i1 j1 j2*. *inv-bitmap s* $\wedge$ *inv-bitmap0 s* $\wedge$
inv-mempool-info s $\wedge$
$p \in$ *mem-pools s* $\wedge$
$i1 <$ *length* (*levels* (*mem-pool-info s p*)) $\wedge$
$j1 <$ *length* (*bits* (*levels* (*mem-pool-info s p*) ! *i1*)) $\wedge$
$i2 <$ *length* (*levels* (*mem-pool-info s p*)) $\wedge$
$j2 <$ *length* (*bits* (*levels* (*mem-pool-info s p*) ! *i2*)) $\wedge$
get-bit-s s p i2 j2 $= DIVIDED \wedge$
$i1 < i2 \wedge$
$j1 = j2\ div\ 4\ \hat{}\ (i2 - i1) \longrightarrow$
get-bit-s s p i1 j1 $= DIVIDED$
apply(*induct i2*)
apply *simp*

apply *clarsimp*
apply(*case-tac* $i1 = i2$)
apply *clarsimp* **apply**(*simp add*:*inv-bitmap-def Let-def*)
apply *fastforce*

apply(*subgoal-tac* $i1 < i2$) **prefer** *2* **apply** *simp*

apply(*subgoal-tac get-bit-s s p i2* (*j2 div 4*) = *DIVIDED*) **prefer** *2*
apply(*simp add*:*inv-bitmap-def Let-def*) **apply** *fastforce*
apply(*subgoal-tac get-bit-s s p i1* ((*j2 div 4*) *div 4* ˆ (*i2* − *i1*)) = *DIVIDED*)
prefer *2*
apply(*subgoal-tac* (*j2 div 4*) *div 4* ˆ (*i2* − *i1*) < *length* (*bits* (*levels* (*mem-pool-info s p*) ! *i1*)))
prefer *2* **apply** (*simp add*: *Divides.div-mult2-eq Suc-diff-le*)
apply(*subgoal-tac j2 div 4* < *length* (*bits* (*levels* (*mem-pool-info s p*) ! *i2*)))
prefer *2* **apply**(*simp add*:*inv-mempool-info-def Let-def*)
apply *fastforce*
apply(*subgoal-tac j2 div 4 div 4* ˆ (*i2* − *i1*) = *j2 div 4* ˆ (*Suc i2* − *i1*))
prefer *2*
apply (*metis Suc-diff-le div-mult2-eq less-or-eq-imp-le power-Suc*)
apply *fastforce*
done

lemma *block-imp-up-alldiv*:
inv-bitmap s ⟹ *inv-bitmap0 s* ⟹
inv-mempool-info s ⟹
p ∈ *mem-pools s* ⟹
i1 < *length* (*levels* (*mem-pool-info s p*)) ⟹
j1 < *length* (*bits* (*levels* (*mem-pool-info s p*) ! *i1*)) ⟹
i2 < *length* (*levels* (*mem-pool-info s p*)) ⟹
j2 < *length* (*bits* (*levels* (*mem-pool-info s p*) ! *i2*)) ⟹
(*get-bit-s s p i2 j2* = *FREE* ∨
get-bit-s s p i2 j2 = *FREEING* ∨ *get-bit-s s p i2 j2* = *ALLOCATED* ∨ *get-bit-s s p i2 j2* = *ALLOCATING*) ⟹
i1 < *i2* ⟹
j1 = *j2 div 4* ˆ (*i2* − *i1*) ⟹
get-bit-s s p i1 j1 = *DIVIDED*
apply(*subgoal-tac get-bit-s s p* (*i2* − *1*) (*j2 div 4*) = *DIVIDED*)
prefer *2* **apply**(*simp add*:*inv-bitmap-def Let-def*)
apply (*metis neq0-conv not-less-zero*)
apply(*case-tac i1* = *i2* − *1*)
apply *simp*

apply *clarsimp*
apply(*rule div-imp-up-alldiv*[*rule-format*,*of s p i1 j2 div 4* ˆ (*i2* − *i1*) *i2* − *1 j2 div 4*])
apply *clarsimp*
apply(*rule conjI*) **apply** *simp*
apply(*rule conjI*) **apply**(*simp add*:*inv-mempool-info-def Let-def*)
using *One-nat-def div-eq-0-iff gr-implies-not0 nat-0-less-mult-iff*
apply (*metis* (*no-types*, *lifting*) *less-mult-imp-div-less nat-neq-iff power-minus-mult semiring-normalization-rules*(*17*))
using *Divides.div-mult2-eq Suc-diff-Suc Suc-pred linorder-neqE-nat not-less-eq not-less-zero power-Suc*
by (*metis not-less*)

lemma *addr-in-same-block*:
inv-bitmap0 s $\Longrightarrow$ *inv-bitmap s* $\Longrightarrow$ *inv-mempool-info s* $\Longrightarrow$
p $\in$ *mem-pools s* $\Longrightarrow$ *addr* $<$ *n-max* (*mem-pool-info s p*) $*$ *max-sz* (*mem-pool-info s p*) $\Longrightarrow$
addr-in-block (*mem-pool-info s p*) *addr i1 j1* $\Longrightarrow$
addr-in-block (*mem-pool-info s p*) *addr i2 j2* $\Longrightarrow$
i1 = *i2* $\wedge$ *j1* = *j2*
apply(*case-tac i1* = *i2*)
  apply(*rule conjI*) **apply** *fast*
  apply *clarsimp*
  apply(*case-tac j1* $<$ *j2*)
    apply (*smt Groups.mult-ac*(*2*) *mult-Suc-right nat-0-less-mult-iff neq0-conv not-le split-div-lemma*)
  apply(*case-tac j1* $>$ *j2*)
    apply (*smt Groups.mult-ac*(*2*) *mult-Suc-right nat-0-less-mult-iff neq0-conv not-le split-div-lemma*)
  apply *simp*

apply(*subgoal-tac* $\exists$ *n*$>$*0*. *max-sz* (*mem-pool-info s p*) = (*4* $*$ *n*) $*$ (*4* ˆ *n-levels* (*mem-pool-info s p*)))
  prefer *2* **apply**(*simp add*:*inv-mempool-info-def Let-def*) **apply** *metis*
apply(*subgoal-tac length* (*levels* (*mem-pool-info s p*)) = *n-levels* (*mem-pool-info s p*))
  prefer *2* **apply**(*simp add*:*inv-mempool-info-def Let-def*)

apply(*case-tac i1* $<$ *i2*)
apply(*subgoal-tac addr div* (*max-sz* (*mem-pool-info s p*) *div 4* ˆ *i1*) = *j1*)
  prefer *2* **using** *addr-in-div*[*of addr j1 max-sz* (*mem-pool-info s p*) *div 4* ˆ *i1*] **apply** *simp*
apply(*subgoal-tac addr div* (*max-sz* (*mem-pool-info s p*) *div 4* ˆ *i2*) = *j2*)
  prefer *2* **using** *addr-in-div*[*of addr j2 max-sz* (*mem-pool-info s p*) *div 4* ˆ *i2*] **apply** *simp*

apply(*subgoal-tac j1* = *j2 div* (*4* ˆ (*i2* − *i1*))) **prefer** *2*
  apply(*rule subst*[**where** *s*=*addr div* (*max-sz* (*mem-pool-info s p*) *div 4* ˆ *i2*) *div 4* ˆ (*i2* − *i1*) **and** *t*=*j2 div 4* ˆ (*i2* − *i1*)])
    apply *fast*
  apply(*rule subst*[**where** *s*=*addr div* ((*max-sz* (*mem-pool-info s p*) *div 4* ˆ *i2*) $*$ *4* ˆ (*i2* − *i1*))
                    and *t*=*addr div* (*max-sz* (*mem-pool-info s p*) *div 4* ˆ *i2*) *div 4* ˆ (*i2* − *i1*)])
    using *div2-eq-divmul*[*of addr max-sz* (*mem-pool-info s p*) *div 4* ˆ *i2 4* ˆ (*i2* − *i1*)] **apply** *simp*
  apply(*rule subst*[**where** *s*=*max-sz* (*mem-pool-info s p*) *div 4* ˆ *i1* **and**
                        t=*max-sz* (*mem-pool-info s p*) *div 4* ˆ *i2* $*$ *4* ˆ (*i2* − *i1*)])
    apply(*subgoal-tac max-sz* (*mem-pool-info s p*) *mod* (*4* ˆ *i1*) = *0*)
      prefer *2* **apply** (*metis ge-pow-mod-0*)
    apply(*subgoal-tac max-sz* (*mem-pool-info s p*) *mod* (*4* ˆ *i2*) = *0*)

prefer *2* **apply** (*metis ge-pow-mod-0*)
apply(*smt add-diff-inverse-nat div2-eq-minus less-imp-le-nat m-mod-div minus-div-mult-eq-mod*

minus-mult-div-eq-mod mod-div-self mod-mult-self2-is-0 not-less power-add zero-neq-numeral)
apply *fast*

apply(*subgoal-tac get-bit-s s p i1 j1 = DIVIDED*)
prefer *2* **using** *block-imp-up-alldiv*[*of s p i1 j1 i2 j2*] **apply** *fast*
apply *auto*[*1*]

apply(*case-tac i1 > i2*)
apply(*subgoal-tac addr div* (*max-sz* (*mem-pool-info s p*) *div 4* ˆ *i1*) = *j1*)
prefer *2* **using** *addr-in-div*[*of addr j1 max-sz* (*mem-pool-info s p*) *div 4* ˆ *i1*]
apply *simp*
apply(*subgoal-tac addr div* (*max-sz* (*mem-pool-info s p*) *div 4* ˆ *i2*) = *j2*)
prefer *2* **using** *addr-in-div*[*of addr j2 max-sz* (*mem-pool-info s p*) *div 4* ˆ *i2*]
apply *simp*

apply(*subgoal-tac j2 = j1 div* (*4* ˆ (*i1* − *i2*))) **prefer** *2*
apply(*rule subst*[**where** *s=addr div* (*max-sz* (*mem-pool-info s p*) *div 4* ˆ *i1*) *div 4* ˆ (*i1* − *i2*) **and** *t=j1 div 4* ˆ (*i1* − *i2*)])
apply *fast*
apply(*rule subst*[**where** *s=addr div* ((*max-sz* (*mem-pool-info s p*) *div 4* ˆ *i1*) ∗ *4* ˆ (*i1* − *i2*))
and *t=addr div* (*max-sz* (*mem-pool-info s p*) *div 4* ˆ *i1*) *div 4* ˆ (*i1* − *i2*)])
using *div2-eq-divmul*[*of addr max-sz* (*mem-pool-info s p*) *div 4* ˆ *i1 4* ˆ (*i1* − *i2*)] **apply** *simp*
apply(*rule subst*[**where** *s=max-sz* (*mem-pool-info s p*) *div 4* ˆ *i2* **and**
t=max-sz (*mem-pool-info s p*) *div 4* ˆ *i1* ∗ *4* ˆ (*i1* − *i2*)])
apply(*subgoal-tac max-sz* (*mem-pool-info s p*) *mod* (*4* ˆ *i1*) = *0*)
prefer *2* **apply** (*metis ge-pow-mod-0*)
apply(*subgoal-tac max-sz* (*mem-pool-info s p*) *mod* (*4* ˆ *i2*) = *0*)
prefer *2* **apply** (*metis ge-pow-mod-0*)
apply(*smt add-diff-inverse-nat div2-eq-minus less-imp-le-nat m-mod-div minus-div-mult-eq-mod*

minus-mult-div-eq-mod mod-div-self mod-mult-self2-is-0 not-less power-add zero-neq-numeral)
apply *fast*

apply(*subgoal-tac get-bit-s s p i2 j2 = DIVIDED*)
prefer *2* **using** *block-imp-up-alldiv*[*of s p i2 j2 i1 j1*] **apply** *fast*
apply *auto*[*1*]

apply *auto*
done

lemma *inv-impl-mem-cover′*:

inv-mempool-info s $\Longrightarrow$
inv-bitmap0 s $\Longrightarrow$ *inv-bitmap s* $\Longrightarrow$ *inv-bitmapn s* $\Longrightarrow$ *mem-cover s*
apply(*simp add*: *mem-cover-def Let-def*)
apply *clarify*
apply(*rule ex-ex1I*)
  apply *clarsimp* **using** *addr-exist-block*[*of s*] **apply** *fastforce*
  apply *clarsimp* **using** *addr-in-same-block*[*of s*] **apply** *force*
done

lemma *inv-impl-mem-cover*: *inv s* $\Longrightarrow$ *mem-cover s*
  apply(*simp add*:*inv-def*)
  using *inv-impl-mem-cover'* **apply** *fast*
done

abbreviation *divide-noexist-cont'* :: *State* $\Rightarrow$ *mempool-ref* $\Rightarrow$ *bool*
where *divide-noexist-cont' s p* $\equiv$
          let mp = *mem-pool-info s p in*
            $\forall$ *i* < *length* (*levels mp*).
              let bts = *bits* (*levels mp* ! *i*) *in*
              ($\forall$ *j* < *length bts*. (*bts* ! *j* = *DIVIDED* $\longrightarrow$ *i* > *0* $\longrightarrow$ (*bits* (*levels mp* ! (*i* − *1*))) ! (*j div 4*) = *DIVIDED*)
              $\wedge$ (*bts* ! *j* = *NOEXIST* $\longrightarrow$ *i* < *length* (*levels mp*) − *1* $\longrightarrow$ *noexist-bits mp* (*i*+*1*) (*j*∗*4*)))

definition *divide-noexist-cont* :: *State* $\Rightarrow$ *bool*
where *divide-noexist-cont s* $\equiv$
        $\forall$ *p*∈*mem-pools s*. *divide-noexist-cont' s p*

end

theory *rg-cond*
imports *mem-spec invariant*
begin

5 Rely-guarantee condition of events

5.1 defs of rely-guarantee conditions

definition *lvars-nochange* :: *Thread* $\Rightarrow$ *State* $\Rightarrow$ *State* $\Rightarrow$ *bool*
where *lvars-nochange t r s* $\equiv$
    i r t = *i s t* $\wedge$ *j r t* = *j s t* $\wedge$ *ret r t* = *ret s t*
    $\wedge$ *endt r t* = *endt s t* $\wedge$ *rf r t* = *rf s t* $\wedge$ *tmout r t* = *tmout s t*
    $\wedge$ *lsizes r t* = *lsizes s t* $\wedge$ *alloc-l r t* = *alloc-l s t* $\wedge$ *free-l r t* = *free-l s t*
    $\wedge$ *from-l r t* = *from-l s t* $\wedge$ *blk r t* = *blk s t* $\wedge$ *nodev r t* = *nodev s t*
    $\wedge$ *bn r t* = *bn s t* $\wedge$ *lbn r t* = *lbn s t* $\wedge$ *lsz r t* = *lsz s t* $\wedge$ *block2 r t* = *block2 s t*
    $\wedge$ *free-block-r r t* = *free-block-r s t* $\wedge$ *alloc-lsize-r r t* = *alloc-lsize-r s t* $\wedge$ *lvl r t* = *lvl s t* $\wedge$ *bb r t* = *bb s t*
    $\wedge$ *block-pt r t* = *block-pt s t* $\wedge$ *th r t* = *th s t* $\wedge$ *need-resched r t* = *need-resched*

$s\ t$
  $\wedge$ *mempoolalloc-ret r t* = *mempoolalloc-ret s t*
  $\wedge$ *freeing-node r t* = *freeing-node s t* $\wedge$ *allocating-node r t* = *allocating-node s t*

definition *lvars-nochange-rel* :: *Thread* $\Rightarrow$ (*State* $\times$ *State*) *set*
where *lvars-nochange-rel t* $\equiv$ {(*s*,*r*). *lvars-nochange t s r*}

definition *lvars-nochange-4all* :: (*State* $\times$ *State*) *set*
where *lvars-nochange-4all* $\equiv$ {(*s*,*r*). $\forall$ *t*. *lvars-nochange t s r*}

definition *lvars-nochange1* :: *Thread* $\Rightarrow$ *State* $\Rightarrow$ *State* $\Rightarrow$ *bool*
where *lvars-nochange1 t r s* $\equiv$ *freeing-node r t* = *freeing-node s t* $\wedge$ *allocating-node r t* = *allocating-node s t*

definition *lvars-nochange1-rel* :: *Thread* $\Rightarrow$ (*State* $\times$ *State*) *set*
where *lvars-nochange1-rel t* $\equiv$ {(*s*,*r*). *lvars-nochange1 t s r*}

definition *lvars-nochange1-4all* :: (*State* $\times$ *State*) *set*
where *lvars-nochange1-4all* $\equiv$ {(*s*,*r*). $\forall$ *t*. *lvars-nochange1 t s r*}

lemma *lvars-nochange-trans*:
lvars-nochange t x y $\Longrightarrow$ *lvars-nochange t y z* $\Longrightarrow$ *lvars-nochange t x z*
apply(*simp add*:*lvars-nochange-def*)
done

lemma *lvars-nochange-sym*:
lvars-nochange t x y $\Longrightarrow$ *lvars-nochange t y x*
apply(*simp add*:*lvars-nochange-def*)
done

lemma *lvars-nochange-refl*:
lvars-nochange t x x
apply(*simp add*:*lvars-nochange-def*)
done

lemma *lvars-nc-nc1*: *lvars-nochange t r s* $\Longrightarrow$ *lvars-nochange1 t r s*
  unfolding *lvars-nochange-def lvars-nochange1-def* **by** *simp*

lemma *lv-noch-all1*: (*s*,*r*)$\in$*lvars-nochange-4all*
  $\Longrightarrow$ (*s*,*r*)$\in$*lvars-nochange-rel t* $\wedge$ ($\forall$ *t'*. *t'* $\neq$ *t* $\longrightarrow$ (*s*,*r*)$\in$*lvars-nochange-rel t'*)
  unfolding *lvars-nochange-4all-def lvars-nochange-rel-def* **by** *auto*

lemma *lv-noch-all2*: (*s*,*r*)$\in$*lvars-nochange-rel t* $\wedge$ ($\forall$ *t'*. *t'* $\neq$ *t* $\longrightarrow$ *lvars-nochange t' s r*)
  $\Longrightarrow$ (*s*,*r*)$\in$*lvars-nochange-4all*
  unfolding *lvars-nochange-4all-def lvars-nochange-rel-def* **by** *auto*

definition *gvars-nochange* :: *State* $\Rightarrow$ *State* $\Rightarrow$ *bool*
where *gvars-nochange s r* $\equiv$ *cur r* = *cur s* $\wedge$ *tick r* = *tick s* $\wedge$ *thd-state r* =

thd-state s
$\wedge$ *mem-pools r* = *mem-pools s* $\wedge$ *mem-pool-info r* = *mem-pool-info s*

definition *gvars-nochange-rel* :: (*State* $\times$ *State*) *set*
where *gvars-nochange-rel* $\equiv$ {(*s*,*r*). *gvars-nochange s r*}

definition *gvars-conf* :: *State* $\Rightarrow$ *State* $\Rightarrow$ *bool*
where *gvars-conf s r* $\equiv$
mem-pools r = *mem-pools s*
$\wedge$ ($\forall$ *p*. *buf* (*mem-pool-info s p*) = *buf* (*mem-pool-info r p*)
$\wedge$ *max-sz* (*mem-pool-info s p*) = *max-sz* (*mem-pool-info r p*)
$\wedge$ *n-max* (*mem-pool-info s p*) = *n-max* (*mem-pool-info r p*)
$\wedge$ *n-levels* (*mem-pool-info s p*) = *n-levels* (*mem-pool-info r p*)
$\wedge$ *length* (*levels* (*mem-pool-info s p*)) = *length* (*levels* (*mem-pool-info r p*))
$\wedge$ ($\forall$ *i*. *length* (*bits* (*levels* (*mem-pool-info s p*) ! *i*))
= *length* (*bits* (*levels* (*mem-pool-info r p*) ! *i*))))

definition *gvars-conf-stable* :: (*State* $\times$ *State*) *set*
where *gvars-conf-stable* $\equiv$ {(*s*,*r*). *gvars-conf s r*}

definition *inv-sta-rely* :: (*State* $\times$ *State*) *set*
where *inv-sta-rely* $\equiv$ {(*s*,*r*). *inv s* $\longrightarrow$ *inv r*}

definition *inv-sta-guar* :: (*State* $\times$ *State*) *set*
where *inv-sta-guar* $\equiv$ {(*s*,*r*). *inv s* $\longrightarrow$ *inv r*}

lemma *glnochange-inv0*:
(*a*, *b*) $\in$ *lvars-nochange1-4all* $\Longrightarrow$ *cur a* = *cur b* $\Longrightarrow$
thd-state a = *thd-state b* $\Longrightarrow$ *mem-pools a* = *mem-pools b* $\Longrightarrow$
mem-pool-info a = *mem-pool-info b* $\Longrightarrow$ *inv a* $\Longrightarrow$ *inv b*
apply(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def inv-def*)
apply(*rule conjI*) **apply**(*simp add*:*inv-cur-def*)
apply(*rule conjI*) **apply**(*simp add*:*inv-thd-waitq-def*) **apply** *auto*[*1*]
apply(*rule conjI*) **apply**(*simp add*:*inv-mempool-info-def*)
apply(*rule conjI*) **apply**(*simp add*:*inv-bitmap-freelist-def*)
apply(*rule conjI*) **apply**(*simp add*:*inv-bitmap-def*)
apply(*rule conjI*) **apply**(*simp add*: *inv-aux-vars-def mem-block-addr-valid-def*)

apply(*rule conjI*) **apply**(*simp add*:*inv-bitmap0-def*)
apply(*rule conjI*) **apply**(*simp add*:*inv-bitmapn-def*)
apply(*simp add*:*inv-bitmap-not4free-def*)
done

lemma *glnochange-inv1*:
(*a*, *b*) $\in$ *lvars-nochange-4all* $\Longrightarrow$ *cur a* = *cur b* $\Longrightarrow$
thd-state a = *thd-state b* $\Longrightarrow$ *mem-pools a* = *mem-pools b* $\Longrightarrow$
mem-pool-info a = *mem-pool-info b* $\Longrightarrow$ *inv a* $\Longrightarrow$ *inv b*

apply(*simp add:lvars-nochange-4all-def lvars-nochange-def*)
using *glnochange-inv0*
apply(*simp add:lvars-nochange1-4all-def lvars-nochange1-def*)
by *metis*

lemma *glnochange-inv*:
$inv\ a \Longrightarrow \forall t'.\ t' \neq t1 \longrightarrow lvars\text{-}nochange\ t'\ a\ b$
$\Longrightarrow gvars\text{-}nochange\ a\ b \Longrightarrow lvars\text{-}nochange\ t1\ a\ b \Longrightarrow inv\ b$
apply(*subgoal-tac* $(a,\ b) \in$ *lvars-nochange-4all*)
apply(*simp add*: *gvars-nochange-def*)
using *glnochange-inv1* **apply** *auto*
using *lv-noch-all2*[*of a b t1*] **apply** *auto*[*1*]
by(*simp add*: *lvars-nochange-rel-def*)

definition *Schedule-rely* :: (*State* × *State*) *set*
where *Schedule-rely* ≡ $\{(s,r).\ inv\ s \longrightarrow inv\ r\} \cup Id$

definition *Schedule-guar* :: (*State* × *State*) *set*
where *Schedule-guar* ≡
$((*\{\!|(^{o}cur \neq Some\ t \longrightarrow$
$(^{o}cur \neq None \longrightarrow {}^{a}thd\text{-}state = (^{o}thd\text{-}state\ (the\ (^{o}cur) := READY))(t :=$
$RUNNING) \wedge {}^{a}cur = Some\ t)$
$\wedge\ (^{o}cur = None \longrightarrow {}^{a}thd\text{-}state = {}^{o}thd\text{-}state\ (t := RUNNING)) \wedge {}^{a}cur =$
$Some\ t)$
$\wedge\ (^{o}cur = Some\ t \longrightarrow {}^{a}thd\text{-}state = {}^{o}thd\text{-}state \wedge {}^{o}cur = {}^{a}cur)\ |\!\}*)$
$\{(s,r).\ inv\ s \longrightarrow inv\ r\}$
$\cap\ \{\!|{}^{o}tick = {}^{a}tick \wedge {}^{o}mem\text{-}pools = {}^{a}mem\text{-}pools \wedge {}^{o}mem\text{-}pool\text{-}info = {}^{a}mem\text{-}pool\text{-}info|\!\}$

$\cap\ (\bigcap t.\ lvars\text{-}nochange\text{-}rel\ t)) \cup Id$

definition *Schedule-RGCond* :: *Thread* ⇒ (*State*) *PiCore-Hoare.rgformula*
where *Schedule-RGCond t* ≡
RG[$\{s.\ inv\ s\}$,
Schedule-rely, *Schedule-guar*,
$\{s.\ inv\ s\}$]

definition *Tick-rely* :: (*State* × *State*) *set*
where *Tick-rely* ≡ $\{\!|{}^{o}tick = {}^{a}tick|\!\} \cup Id$

definition *Tick-guar* :: (*State* × *State*) *set*
where *Tick-guar* ≡ $(\{\!|{}^{a}tick = {}^{o}tick + 1 \wedge {}^{o}cur = {}^{a}cur \wedge {}^{o}thd\text{-}state = {}^{a}thd\text{-}state$
$\wedge\ {}^{o}mem\text{-}pools = {}^{a}mem\text{-}pools \wedge {}^{o}mem\text{-}pool\text{-}info = {}^{a}mem\text{-}pool\text{-}info|\!\}$
$\cap\ (\bigcap t.\ lvars\text{-}nochange\text{-}rel\ t)) \cup Id$

definition *Tick-RGCond* :: (*State*) *PiCore-Hoare.rgformula*
where *Tick-RGCond* ≡
RG[$\{\!|True|\!\}$, *Tick-rely*, *Tick-guar*, $\{\!|True|\!\}$]

abbreviation *alloc-blk-valid* :: *State* ⇒ *mempool-ref* ⇒ *nat* ⇒ *nat* ⇒ *mem-ref*

$\Rightarrow$ *bool*
where *alloc-blk-valid s p lv bnum blkaddr*
$\equiv$ (*blkaddr* = *buf* (*mem-pool-info s p*) + *bnum* $*$ ((*max-sz* (*mem-pool-info s p*)) *div* (*4* ^ *lv*))
$\wedge$ *bnum* < *n-max* (*mem-pool-info s p*) $*$ (*4* ^ *lv*))

abbreviation *alloc-memblk-data-valid* :: *State* $\Rightarrow$ *mempool-ref* $\Rightarrow$ *Mem-block* $\Rightarrow$ *bool*
where *alloc-memblk-data-valid s p mb* $\equiv$ *alloc-blk-valid s p* (*level mb*) (*block mb*) (*data mb*)

definition *alloc-memblk-valid* :: *State* $\Rightarrow$ *mempool-ref* $\Rightarrow$ *nat* $\Rightarrow$ *Mem-block* $\Rightarrow$ *bool*
where *alloc-memblk-valid s p sz mb* $\equiv$
p = *pool mb* $\wedge$ *p* $\in$ *mem-pools s*
$\wedge$ *sz* $\leq$ (*max-sz* (*mem-pool-info s p*)) *div* (*4* ^ (*level mb*)) ($*$ *block size of level mb* + *1* < *sz* $\leq$ *block size of level mb* $*$)
$\wedge$ (*level mb* < *n-levels* (*mem-pool-info s p*) − *1* $\longrightarrow$ *sz* > (*max-sz* (*mem-pool-info s p*)) *div* (*4* ^ (*level mb* + *1*)))
$\wedge$ *alloc-memblk-data-valid s p mb*

abbreviation *Mem-pool-alloc-pre* :: *Thread* $\Rightarrow$ *State set*
where *Mem-pool-alloc-pre t* $\equiv$ {*s*. *inv s* $\wedge$ *allocating-node s t* = *None* $\wedge$ *freeing-node s t* = *None*}

definition *Mem-pool-alloc-rely* :: *Thread* $\Rightarrow$ (*State* $\times$ *State*) *set*
where *Mem-pool-alloc-rely t* $\equiv$
((*lvars-nochange-rel t* $\cap$ *gvars-conf-stable*
$\cap$ {(*s*,*r*). *inv s* $\longrightarrow$ *inv r*}
$\cap$ {(*s*,*r*).(*cur s* = *Some t* $\longrightarrow$ *mem-pool-info s* = *mem-pool-info r*
$\wedge$ ($\forall$ *t'*. *t'* $\neq$ *t* $\longrightarrow$ *lvars-nochange t' s r*))}) $\cup$ *Id*)

definition *Mem-pool-alloc-guar* :: *Thread* $\Rightarrow$ (*State* $\times$ *State*) *set*
where *Mem-pool-alloc-guar t* $\equiv$
((*gvars-conf-stable* $\cap$
{(*s*,*r*). (*cur s* $\neq$ *Some t* $\longrightarrow$ *gvars-nochange s r* $\wedge$ *lvars-nochange t s r*)
$\wedge$ (*cur s* = *Some t* $\longrightarrow$ *inv s* $\longrightarrow$ *inv r*)
$\wedge$ ($\forall$ *t'*. *t'* $\neq$ *t* $\longrightarrow$ *lvars-nochange t' s r*) }
$\cap$ $\{\!|{}^{\text{o}}tick = {}^{\text{a}}tick|\!\}$) $\cup$ *Id*)

definition *Mem-pool-alloc-post* :: *Thread* $\Rightarrow$ *mempool-ref* $\Rightarrow$ *nat* $\Rightarrow$ *int* $\Rightarrow$ *State set*
where *Mem-pool-alloc-post t p sz timeout* $\equiv$
{*s*. *inv s* $\wedge$ *allocating-node s t* = *None* $\wedge$ *freeing-node s t* = *None*
$\wedge$ (*timeout* = *FOREVER* $\longrightarrow$ (*ret s t* = *ESIZEERR* $\wedge$ *mempoolalloc-ret s t* = *None*

$\vee$ *ret s t* = *OK* $\wedge$ ($\exists$ *mblk*. *mempoolalloc-ret s t* = *Some mblk* $\wedge$ *alloc-memblk-valid s p sz mblk*)))
$\wedge$ (*timeout* = *NOWAIT* $\longrightarrow$ ((*ret s t* = *ENOMEM* $\vee$ *ret s t* = *ESIZEERR*) $\wedge$ *mempoolalloc-ret s t* = *None*)
$\vee$ (*ret s t* = *OK* $\wedge$ ($\exists$ *mblk*. *mempoolalloc-ret s t* = *Some mblk* $\wedge$ *alloc-memblk-valid s p sz mblk*)))
$\wedge$ (*timeout* > *0* $\longrightarrow$ ((*ret s t* = *ETIMEOUT* $\vee$ *ret s t* = *ESIZEERR*) $\wedge$ *mempoolalloc-ret s t* = *None*)
$\vee$ (*ret s t* = *OK* $\wedge$ ($\exists$ *mblk*. *mempoolalloc-ret s t* = *Some mblk* $\wedge$ *alloc-memblk-valid s p sz mblk*)))}

definition *Mem-pool-alloc-RGCond* :: *Thread* $\Rightarrow$ *mempool-ref* $\Rightarrow$ *nat* $\Rightarrow$ *int* $\Rightarrow$ (*State*) *PiCore-Hoare.rgformula*
where *Mem-pool-alloc-RGCond t p sz timeout* $\equiv$
RG[*Mem-pool-alloc-pre t*,
Mem-pool-alloc-rely t,
Mem-pool-alloc-guar t,
Mem-pool-alloc-post t p sz timeout]

abbreviation *Mem-pool-free-pre* :: *Thread* $\Rightarrow$ *State set*
where *Mem-pool-free-pre t* $\equiv$ {*s*. *inv s* $\wedge$ *allocating-node s t* = *None* $\wedge$ *freeing-node s t* = *None*}

definition *Mem-pool-free-rely* :: *Thread* $\Rightarrow$ (*State* $\times$ *State*) *set*
where *Mem-pool-free-rely t* $\equiv$
((*lvars-nochange-rel t* $\cap$ *gvars-conf-stable*
$\cap$ {(*s*,*r*). *inv s* $\longrightarrow$ *inv r*}
$\cap$ {(*s*,*r*).(*cur s* = *Some t* $\longrightarrow$ *mem-pool-info s* = *mem-pool-info r*
$\wedge$ ($\forall$ *t'*. *t'* $\neq$ *t* $\longrightarrow$ *lvars-nochange t' s r*))}) $\cup$ *Id*)

definition *Mem-pool-free-guar* :: *Thread* $\Rightarrow$ (*State* $\times$ *State*) *set*
where *Mem-pool-free-guar t* $\equiv$
((*gvars-conf-stable* $\cap$
{(*s*,*r*). (*cur s* $\neq$ *Some t* $\longrightarrow$ *gvars-nochange s r* $\wedge$ *lvars-nochange t s r*)
$\wedge$ (*cur s* = *Some t* $\longrightarrow$ *inv s* $\longrightarrow$ *inv r*)
$\wedge$ ($\forall$ *t'*. *t'* $\neq$ *t* $\longrightarrow$ *lvars-nochange t' s r*) }
$\cap$ $\{\!|{}^{\mathrm{o}}tick = {}^{\mathrm{a}}tick|\!\}$) $\cup$ *Id*)

definition *Mem-pool-free-post* :: *Thread* $\Rightarrow$ *State set*
where *Mem-pool-free-post t* $\equiv$ {*s*. *inv s* $\wedge$ *allocating-node s t* = *None* $\wedge$ *freeing-node s t* = *None*}

definition *Mem-pool-free-RGCond* :: *Thread* $\Rightarrow$ *Mem-block* $\Rightarrow$ (*State*) *PiCore-Hoare.rgformula*
where *Mem-pool-free-RGCond t b* $\equiv$
RG[*Mem-pool-free-pre t*,
Mem-pool-free-rely t,
Mem-pool-free-guar t,
Mem-pool-free-post t]

5.2 stablility, subset relations of rely-guarantee conditions

lemma *stable-inv-free-rely*:
  $(s,r) \in$ *Mem-pool-free-rely t* $\Longrightarrow$ *inv s* $\Longrightarrow$ *inv r*
  apply (*simp add*:*Mem-pool-free-rely-def*)
  apply(*case-tac cur s* = *Some t*) **apply** *simp*
    apply(*subgoal-tac* $(s, r) \in$ *lvars-nochange-4all*)
      apply(*simp add*:*lvars-nochange-4all-def lvars-nochange-def*)
      apply(*simp add*:*inv-def*) **unfolding** *gvars-conf-stable-def gvars-conf-def*
      apply(*rule conjI*) **apply**(*simp add*:*inv-cur-def*) **apply** *auto*[*1*] **apply** *metis*
        apply(*simp add*:*lvars-nochange-4all-def lvars-nochange-rel-def*)
        apply *auto*[*1*] **apply**(*simp add*:*lvars-nochange-def*)
        apply *auto*
done

lemma *stable-inv-free-rely1*: *stable* $\{\!\mid$ ´*inv* $\mid\!\}$ (*Mem-pool-free-rely t*)
  using *stable-inv-free-rely* **unfolding** *stable-def* **by** *auto*

lemma *stable-inv-alloc-rely*:
  $(s,r) \in$ *Mem-pool-alloc-rely t* $\Longrightarrow$ *inv s* $\Longrightarrow$ *inv r*
  apply(*subgoal-tac Mem-pool-alloc-rely t* = *Mem-pool-free-rely t*)
  using *stable-inv-free-rely* **apply** *simp*
  by (*simp add*:*Mem-pool-alloc-rely-def Mem-pool-free-rely-def*)

lemma *stable-inv-alloc-rely1*: *stable* $\{\!\mid$ ´*inv* $\mid\!\}$ (*Mem-pool-alloc-rely t*)
  using *stable-inv-alloc-rely* **unfolding** *stable-def* **by** *auto*

lemma *stable-inv-sched-rely*:
  $(s,r) \in$ *Schedule-rely* $\Longrightarrow$ *inv s* $\Longrightarrow$ *inv r*
  apply (*simp add*:*Schedule-rely-def*) **by** *auto*

lemma *stable-inv-sched-rely1*: *stable* $\{\!\mid$ ´*inv* $\mid\!\}$ *Schedule-rely*
  using *stable-inv-sched-rely* **unfolding** *stable-def* **by** *auto*

lemma *free-guar-stb-inv*: *stable* $\{\!\mid$ ´*inv* $\mid\!\}$ (*Mem-pool-free-guar t*)
proof −
{
  fix *x*
  assume *a0*: *inv x*
  {
    fix *y*
    assume *b0*: $(x,y) \in$ *Mem-pool-free-guar t*
   hence $(x,y) \in \{(s,r).$ (*cur s* $\neq$ *Some t* $\longrightarrow$ *gvars-nochange s r* $\wedge$ *lvars-nochange t s r*)
        $\wedge$ (*cur s* = *Some t* $\longrightarrow$ *inv s* $\longrightarrow$ *inv r*)
        $\wedge$ ($\forall t'.\ t' \neq t \longrightarrow$ *lvars-nochange t' s r*)$\}$
      unfolding *Mem-pool-free-guar-def gvars-nochange-def lvars-nochange-def* **by** *auto*
    hence (*cur x* $\neq$ *Some t* $\longrightarrow$ *gvars-nochange x y* $\wedge$ *lvars-nochange t x y*)
        $\wedge$ (*cur x* = *Some t* $\longrightarrow$ *inv x* $\longrightarrow$ *inv y*)

$\wedge$ ($\forall$ *t'*. *t'* $\neq$ *t* $\longrightarrow$ *lvars-nochange t' x y*) **by** *simp*
hence *inv y*
apply(*case-tac cur x* $\neq$ *Some t*)
apply (*simp add*: *gvars-nochange-def lvars-nochange-def*) **using** *a0* **apply** *clarify*
apply(*simp add*:*inv-def*)
apply(*rule conjI*) **apply**(*simp add*:*inv-cur-def*)
apply(*rule conjI*) **apply**(*simp add*:*inv-thd-waitq-def*) **apply** *metis*
apply(*rule conjI*) **apply**(*simp add*:*inv-mempool-info-def*)
apply(*rule conjI*) **using** *inv-bitmap-freelist-def* **apply** *metis*
apply(*rule conjI*) **apply**(*simp add*:*inv-bitmap-def*)
apply(*rule conjI*) **apply**(*simp add*:*inv-aux-vars-def*)
apply(*rule conjI*) **apply** *metis*
apply(*rule conjI*) **apply**(*simp add*:*mem-block-addr-valid-def*) **apply** *metis*

apply(*rule conjI*) **apply** *metis*
apply(*rule conjI*) **apply**(*simp add*:*mem-block-addr-valid-def*) **apply** *metis*

apply(*rule conjI*) **apply** *metis*
apply(*rule conjI*) **apply** *metis*
apply *metis*
apply(*rule conjI*) **apply**(*simp add*:*inv-bitmap0-def*)
apply(*rule conjI*) **apply**(*simp add*:*inv-bitmapn-def*)
apply(*simp add*:*inv-bitmap-not4free-def*)
using *a0* **by** *auto*
}
}
then show *?thesis* **by** (*simp add*:*stable-def*)
qed

lemma *alloc-guar-stb-inv*: *stable* {|´*inv*|} (*Mem-pool-alloc-guar t*)
apply(*subgoal-tac Mem-pool-alloc-guar t* = *Mem-pool-free-guar t*)
using *free-guar-stb-inv* **apply** *simp*
by (*simp add*:*Mem-pool-alloc-guar-def Mem-pool-free-guar-def*)

lemma *sched-guar-stb-inv*:
(*s*,*r*)$\in$*Schedule-guar* $\Longrightarrow$ *inv s* $\Longrightarrow$ *inv r*
apply(*simp add*:*Schedule-guar-def*)
apply(*erule disjE*) **by** *auto*

lemma *tick-guar-stb-inv*:
(*s*,*r*)$\in$*Tick-guar* $\Longrightarrow$ *inv s* $\Longrightarrow$ *inv r*
apply(*simp add*:*Tick-guar-def*) **apply**(*erule disjE*)
using *glnochange-inv0 lvars-nc-nc1*
unfolding *lvars-nochange1-4all-def lvars-nochange-rel-def* **apply** *auto*[*1*] **apply** *blast*
by *auto*

lemma *mem-pool-alloc-pre-stb*: *stable* (*Mem-pool-alloc-pre t*) (*Mem-pool-alloc-rely*

t)
  apply(*rule subst*[**where** *t*=⦃´*inv* ∧ ´*allocating-node t* = *None* ∧ ´*freeing-node t* = *None*⦄
        and *s*=⦃´*inv*⦄ ∩ ⦃´*allocating-node t* = *None* ∧ ´*freeing-node t* = *None*⦄])
    apply *auto*[*1*]
  apply(*rule stable-int2*) **apply** (*simp add*: *stable-inv-alloc-rely1*)
 apply(*simp add*:*stable-def Mem-pool-alloc-rely-def gvars-conf-stable-def lvars-nochange-rel-def lvars-nochange-def*)
done

lemma *mp-alloc-post-stb*: *stable* (*Mem-pool-alloc-post t p sz timeout*) (*Mem-pool-alloc-rely t*)
  apply(*simp add*:*stable-def*) **apply**(*rule allI*) **apply**(*rule impI*) **apply**(*rule allI*) **apply**(*rule impI*)
  apply(*simp add*:*Mem-pool-alloc-rely-def Mem-pool-alloc-post-def*)
  apply(*rule conjI*)
    apply(*simp add*:*gvars-conf-stable-def*) **unfolding** *gvars-conf-def* **apply** *metis*
    apply(*simp add*:*lvars-nochange-rel-def lvars-nochange-def*)
    apply(*case-tac x* = *y*)
      apply *simp* **apply** *clarify*
      apply(*simp add*:*alloc-memblk-valid-def gvars-conf-def gvars-conf-stable-def*)
done

lemma *mem-pool-free-pre-stb*: *stable* (*Mem-pool-free-pre t*) (*Mem-pool-free-rely t*)
  apply(*rule subst*[**where** *t*=⦃´*inv* ∧ ´*allocating-node t* = *None* ∧ ´*freeing-node t* = *None*⦄
        and *s*=⦃´*inv*⦄ ∩ ⦃´*allocating-node t* = *None* ∧ ´*freeing-node t* = *None*⦄])
    apply *auto*[*1*]
  apply(*rule stable-int2*) **apply** (*simp add*: *stable-inv-free-rely1*)
 apply(*simp add*:*stable-def Mem-pool-free-rely-def gvars-conf-stable-def lvars-nochange-rel-def lvars-nochange-def*)

done

lemma *mem-pool-free-post-stb*: *stable* (*Mem-pool-free-post t*) (*Mem-pool-free-rely t*)
  using *mem-pool-free-pre-stb* **apply**(*simp add*:*Mem-pool-free-post-def*)
done

lemma *allocguar-in-allocrely*: *t1* ≠ *t2* ⟹ *Mem-pool-alloc-guar t1* ⊆ *Mem-pool-alloc-rely t2*
  apply *clarify*
  proof −
    fix *a b*
    assume *p0*: *t1* ≠ *t2*
      and *p1*: (*a*, *b*) ∈ *Mem-pool-alloc-guar t1*
    hence *p2*: (*a*, *b*) ∈ *gvars-conf-stable*
                ∧ (*cur a* ≠ *Some t1* ⟶ *gvars-nochange a b* ∧ *lvars-nochange t1 a b*)

$\wedge$ (*cur a* = *Some t1* $\longrightarrow$ *inv a* $\longrightarrow$ *inv b*)
$\wedge$ ($\forall$ *t'*. *t'* $\neq$ *t1* $\longrightarrow$ *lvars-nochange t' a b*)
$\wedge$ *tick a* = *tick b* $\vee$ *a* = *b*
unfolding *Mem-pool-alloc-guar-def* **by** *auto*

from *p0 p2* **have**
(*a*, *b*) $\in$ *lvars-nochange-rel t2* $\wedge$ (*a*, *b*) $\in$ *gvars-conf-stable*
$\wedge$ (*inv a* $\longrightarrow$ *inv b*)
$\wedge$ (*cur a* = *Some t2* $\longrightarrow$ *mem-pool-info a* = *mem-pool-info b*
$\wedge$ ($\forall$ *t'*. *t'* $\neq$ *t2* $\longrightarrow$ *lvars-nochange t' a b*))
$\vee$ *a* = *b*
apply *clarify*
apply(*rule conjI*) **apply**(*simp add*:*lvars-nochange-rel-def*)
apply(*rule conjI*) **apply** *simp*
apply(*rule conjI*) **apply** *clarify* **using** *glnochange-inv* **apply** *auto*[*1*]
apply *clarify*
apply(*rule conjI*) **apply**(*simp add*:*gvars-nochange-def*)
by *auto*

thus (*a*, *b*) $\in$ *Mem-pool-alloc-rely t2* **unfolding** *Mem-pool-alloc-rely-def* **by** *simp*
qed

lemma *schedguar-in-allocrely*: *Schedule-guar* $\subseteq$ *Mem-pool-alloc-rely t2*
apply *clarify*
proof –
fix *a b*
assume *p0*: (*a*, *b*) $\in$ *Schedule-guar*
hence *p1*: (*inv a* $\longrightarrow$ *inv b*) $\wedge$ *tick a* = *tick b* $\wedge$ *mem-pools a* = *mem-pools b* $\wedge$ *mem-pool-info a* = *mem-pool-info b*
$\wedge$ (*a*,*b*)$\in$($\bigcap$ *t*. *lvars-nochange-rel t*) $\vee$ *a* = *b*
by(*simp add*:*Schedule-guar-def*)

hence (*a*, *b*) $\in$ *lvars-nochange-rel t2* $\wedge$ (*a*, *b*) $\in$ *gvars-conf-stable*
$\wedge$ (*inv a* $\longrightarrow$ *inv b*)
$\wedge$ (*cur a* = *Some t2* $\longrightarrow$ *mem-pool-info a* = *mem-pool-info b*
$\wedge$ ($\forall$ *t'*. *t'* $\neq$ *t2* $\longrightarrow$ *lvars-nochange t' a b*))
$\vee$ *a* = *b*
apply *clarify*
apply(*rule conjI*) **apply**(*simp add*:*lvars-nochange-rel-def*)
apply(*rule conjI*) **apply**(*simp add*:*gvars-conf-stable-def gvars-conf-def*)
apply(*rule conjI*) **apply** *clarify* **apply** *clarify*
by(*simp add*:*lvars-nochange-rel-def*)

thus (*a*, *b*) $\in$ *Mem-pool-alloc-rely t2* **by**(*simp add*:*Mem-pool-alloc-rely-def*)
qed

lemma *schedguar-in-tickrely*: *Schedule-guar* $\subseteq$ *Tick-rely*
apply(*simp add*:*Schedule-guar-def Tick-rely-def*)

by *auto*

lemma *allocguar-in-tickrely*: *Mem-pool-alloc-guar t* ⊆ *Tick-rely*
apply(*simp add*:*Mem-pool-alloc-guar-def Tick-rely-def*)
by *auto*

lemma *tickguar-in-allocrely*: *Tick-guar* ⊆ *Mem-pool-alloc-rely t*
apply *clarify*
proof −
fix *a b*
assume *p0*: (*a*, *b*) ∈ *Tick-guar*
hence *p1*: *tick b* = *tick a* + *1* ∧ *cur a* = *cur b* ∧ *thd-state a* = *thd-state b*
∧ *mem-pools a* = *mem-pools b* ∧ *mem-pool-info a* = *mem-pool-info b*
∧ (*a*,*b*)∈(⋂ *t*. *lvars-nochange-rel t*) ∨ *a* = *b*
by(*simp add*:*Tick-guar-def*)

hence (*a*, *b*) ∈ *lvars-nochange-rel t* ∧ (*a*, *b*) ∈ *gvars-conf-stable*
∧ (*inv a* ⟶ *inv b*)
∧ (*cur a* = *Some t* ⟶ *mem-pool-info a* = *mem-pool-info b*
∧ (∀ *t*′. *t*′ ≠ *t* ⟶ *lvars-nochange t*′ *a b*))
∨ *a* = *b*
apply *clarify*
apply(*rule conjI*) **apply**(*simp add*:*lvars-nochange-rel-def*)
apply(*rule conjI*) **apply**(*simp add*:*gvars-conf-stable-def gvars-conf-def*)
apply(*rule conjI*) **using** *glnochange-inv0 lvars-nc-nc1* **unfolding** *lvars-nochange-rel-def lvars-nochange1-4all-def*
apply *auto*[*1*] **apply** *blast*
by *auto*

thus (*a*, *b*) ∈ *Mem-pool-alloc-rely t* **by**(*simp add*:*Mem-pool-alloc-rely-def*)
qed

lemma *allocguar-in-schedrely*: *Mem-pool-alloc-guar t* ⊆ *Schedule-rely*
apply(*simp add*:*Mem-pool-alloc-guar-def Schedule-rely-def*)
apply *clarify*
apply(*case-tac cur a* = *Some t*)
apply *simp*
apply *clarify*
using *glnochange-inv* **by** *auto*

lemma *tickguar-in-schedrely*: *Tick-guar* ⊆ *Schedule-rely*
apply *clarify*
proof −
fix *a b*
assume *p0*: (*a*, *b*) ∈ *Tick-guar*
thus (*a*, *b*) ∈ *Schedule-rely*
apply(*simp add*:*Tick-guar-def Schedule-rely-def*) **apply** *auto*
using *glnochange-inv1* **by**(*simp add*:*lvars-nochange-4all-def lvars-nochange-rel-def*)

qed

end

theory *func-cor-lemma*
imports *rg-cond*
begin

declare [[*smt-timeout* = *300*]]

6 some lemmas for functional correctness by rely guarantee proof

lemma *inv-mempool-info-maxsz-mod4*:
inv-mempool-info s $\Longrightarrow$ $\forall$ *p*$\in$*mem-pools s*. *max-sz* (*mem-pool-info s p*) *mod 4* = *0*
unfolding *inv-mempool-info-def*
by (*metis mod-mult-left-eq mod-mult-self1-is-0 mod-mult-self2-is-0 mult-0*)

lemma *inv-mempool-info-maxsz-align4*:
inv-mempool-info s $\Longrightarrow$ $\forall$ *p*$\in$*mem-pools s*. *ALIGN4* (*max-sz* (*mem-pool-info s p*)) = *max-sz* (*mem-pool-info s p*)
using *inv-mempool-info-maxsz-mod4 align40* **by** *simp*

lemma *inv-maxsz-align4*:
inv s $\Longrightarrow$ $\forall$ *p*$\in$*mem-pools s*. *ALIGN4* (*max-sz* (*mem-pool-info s p*)) = *max-sz* (*mem-pool-info s p*)
unfolding *inv-def* **using** *inv-mempool-info-maxsz-align4* **by** *simp*

lemma *lsizes-mod4*:
assumes *p0*: *inv V*
and *p1*: $\forall$ *ii*<*length ls*. *ls* ! *ii* = *ALIGN4* (*max-sz* (*mem-pool-info V p*)) *div 4* ^ *ii*
and *p2*: *length ls* $\leq$ *length* (*levels* (*mem-pool-info V p*))
and *p3*: *p* $\in$ *mem-pools V*
shows $\forall$ *ii*<*length ls*. (*ls* ! *ii*) *mod 4* = *0*
proof −
{
fix *ii*
assume *a0*: *ii* < *length ls*
from *p0 p3* **have** $\exists$ *n*>*0*. *max-sz* (*mem-pool-info V p*) = (*4* ∗ *n*) ∗ (*4* ^ (*length* (*levels* (*mem-pool-info V p*))))
apply(*simp add*:*inv-def inv-mempool-info-def Let-def*) **by** *auto*
then obtain *n* **where** *n* > *0* $\wedge$ *max-sz* (*mem-pool-info V p*) = (*4* ∗ *n*) ∗ (*4* ^

(*length* (*levels* (*mem-pool-info V p*)))) **by** *auto*
hence *a1*: $n > 0 \wedge$ *max-sz* (*mem-pool-info V p*) $=$ $n * (4$ ^ (*length* (*levels* (*mem-pool-info V p*)) $+ 1$)) **by** *auto*

hence *ALIGN4* (*max-sz* (*mem-pool-info V p*)) $=$ *max-sz* (*mem-pool-info V p*)
using *align40* **by** *auto*
with *a0 p1* **have** *a2*: *ls* ! *ii* $=$ *max-sz* (*mem-pool-info V p*) *div* 4 ^ *ii* **by** *auto*
with *a1* **have** *ls* ! *ii* $=$ $n * (4$ ^ (*length* (*levels* (*mem-pool-info V p*)) $+ 1$)) *div* 4 ^ *ii* **by** *simp*
moreover
from *a0 p2* **have** (4::*nat*) ^ (*length* (*levels* (*mem-pool-info V p*)) $+ 1$) *mod* 4 ^ *ii* $= 0$
using *pow-mod-0*[*of ii length* (*levels* (*mem-pool-info V p*)) $+ 1$ 4] **by** *auto*
ultimately have *a3*: *ls* ! *ii* $=$ $n * ((4$ ^ (*length* (*levels* (*mem-pool-info V p*)) $+ 1$)) *div* 4 ^ *ii*)
using *m-mod-div* **by** *auto*

from *a0 p2* **have** $4 \neq$ *NULL* $\wedge$ *ii* $\leq$ *length* (*levels* (*mem-pool-info V p*)) $+ 1$
by *linarith*
hence ((4::*nat*) ^ (*length* (*levels* (*mem-pool-info V p*)) $+ 1$)) *div* 4 ^ *ii*
$=$ 4 ^ (*length* (*levels* (*mem-pool-info V p*)) $+ 1 -$ *ii*)
using *div2-eq-minus*[*of* 4 *ii* (*length* (*levels* (*mem-pool-info V p*)) $+ 1$)] **by** *simp*
hence $n *$ (((4::*nat*) ^ (*length* (*levels* (*mem-pool-info V p*)) $+ 1$)) *div* 4 ^ *ii*)
$= n * (4$ ^ (*length* (*levels* (*mem-pool-info V p*)) $+ 1 -$ *ii*)) **by** *auto*
with *a3* **have** *ls* ! *ii* $= n * (4$ ^ (*length* (*levels* (*mem-pool-info V p*)) $+ 1 -$ *ii*)) **by** *auto*
with *a0 p2* **have** *ls* ! *ii mod* 4 $= 0$ **by** *auto*
}
then show *?thesis* **by** *auto*
qed

lemma *gvars-conf-stb-inv-mpinf*: $(x,y) \in$ *gvars-conf-stable* $\Longrightarrow$ *inv-mempool-info y* $\Longrightarrow$ *inv-mempool-info x*
apply(*simp add*:*gvars-conf-stable-def gvars-conf-def inv-mempool-info-def*)
apply *clarify*
apply(*rule conjI*) **apply** *metis* **apply**(*rule conjI*) **apply** *metis*
apply(*rule conjI*) **apply** *metis* **apply**(*rule conjI*) **apply** *metis*
apply(*rule conjI*) **apply** *metis* **apply** *metis*
done

lemma *ref-byblkn-self*:
$R \geq$ *buf* (*mem-pool-info Va p*) $\Longrightarrow$
($R -$ *buf* (*mem-pool-info Va p*)) *mod sz* $= 0 \Longrightarrow$
buf (*mem-pool-info Va p*) $+$ *block-num* (*mem-pool-info Va p*) *R sz* $*$ *sz* $= R$
apply(*simp add*:*block-num-def*)
apply(*rule subst*[**where** *t*=($R -$ *buf* (*mem-pool-info Va p*)) *div sz* $*$ *sz* **and** *s*=$R -$ *buf* (*mem-pool-info Va p*)])

by *auto*

lemma *partnerbits-udptn-notbit-partbits*:
$\forall jj{<}length\ lst.\ \neg\ (let\ a = (jj\ div\ 4) * 4\ in$
$lst!a = TAG \wedge lst!(a+1) = TAG \wedge lst!(a+2) = TAG \wedge lst!(a+3) = TAG) \Longrightarrow$
$TAG \neq TAG2 \Longrightarrow lst' = \ list\text{-}updates\text{-}n\ lst\ ii\ m\ TAG2 \Longrightarrow$
$\forall jj{<}length\ lst'.\ \neg\ (let\ a = (jj\ div\ 4) * 4\ in$
$lst'!a = TAG \wedge lst'!(a+1) = TAG \wedge lst'!(a+2) = TAG \wedge lst'!(a+3) = TAG)$
apply(*unfold Let-def*) **apply**(*rule allI*, *rule impI*)
apply($case\text{-}tac\ lst'\ !\ (jj\ div\ 4 * 4) = TAG \wedge lst'\ !\ (jj\ div\ 4 * 4 + 1) = TAG$
$\wedge\ lst'\ !\ (jj\ div\ 4 * 4 + 2) = TAG \wedge lst'\ !\ (jj\ div\ 4 * 4 + 3) = TAG$)
apply(*subgoal-tac length lst = length lst′*) **prefer** *2* **apply** *simp*
apply($subgoal\text{-}tac\ \neg\ (lst\ !\ (jj\ div\ 4 * 4) = TAG \wedge lst\ !\ (jj\ div\ 4 * 4 + 1) = TAG$
$\wedge\ lst\ !\ (jj\ div\ 4 * 4 + 2) = TAG \wedge lst\ !\ (jj\ div\ 4 * 4 + 3) = TAG)$)
prefer *2* **apply** *presburger*
apply($case\text{-}tac\ jj\ div\ 4 * 4 + 3 < ii$) **using** *list-updates-n-neq*
apply (*smt One-nat-def add.right-neutral add-Suc-right add-lessD1 numeral-Bit1 numeral-One one-add-one plus-nat.simps*(*2*))
apply($case\text{-}tac\ jj\ div\ 4 * 4 \geq ii + m$) **using** *list-updates-n-neq* **apply** (*smt le-add1 le-trans*)
using *list-updates-eq* **apply** (*smt One-nat-def Suc-leI add.right-neutral add-Suc-right add-lessD1*
div-mult-mod-eq le-less-trans list-updates-n-beyond list-updates-n-eq list-updates-n-neq not-le numeral-Bit1 numeral-One one-add-one)
by *assumption*

end

theory *func-cor-other*
imports *func-cor-lemma*
begin

7 Functional correctness of Schedule

lemma *Schedule-satRG-h1*:
$\Gamma \vdash_I Some\ (IF\ \exists y.\ ´cur = Some\ y\ THEN\ ´thd\text{-}state := ´thd\text{-}state(the\ ´cur := READY);;\ Basic\ (cur\text{-}update\ Map.empty)\ FI;;$
$Basic\ (cur\text{-}update\ (\lambda\text{-}.\ Some\ t));;$
$´thd\text{-}state := ´thd\text{-}state$
$(t := RUNNING))\ sat_p\ [\{\!|´inv|\!\} \cap \{\!|´thd\text{-}state\ t = READY|\!\} \cap$
$\{V\}, \{(s, t).\ s = t\}, UNIV, \{\!|´(Pair\ V) \in Schedule\text{-}guar|\!\} \cap \{\!|´inv|\!\}]$

apply(*case-tac* ⦃´*inv*⦄ ∩ ⦃´*thd-state t* = *READY*⦄ ∩ {*V*} = {})
using *Emptyprecond* **apply** *auto*[*1*]
apply *simp*
apply(*case-tac* ∃ *y*. *cur V* = *Some y*)

apply(*rule Seq*[**where** *mid* = {*V*(|*thd-state* := (*thd-state V*)(*the* (*cur V*) := *READY*)|)(|*cur* := *None*|)(|*cur* := *Some t*|)}])
apply(*rule Seq*[**where** *mid* = {*V*(|*thd-state* := (*thd-state V*)(*the* (*cur V*) := *READY*)|)(|*cur* := *None*|)}])
apply(*rule Cond*)
apply(*simp add*:*stable-def*)
apply(*rule Seq*[**where** *mid* = {*V*(|*thd-state* := (*thd-state V*)(*the* (*cur V*) := *READY*)|)}])
apply(*rule Basic*)
apply *auto*[*1*]
apply(*simp add*:*stable-def*)+
apply(*rule Basic*)
apply *auto*[*1*]
apply(*simp add*:*stable-def*)+
apply(*simp add*:*Skip-def*) **apply**(*rule Basic*) **apply**(*simp add*:*stable-def*)+

apply(*rule Basic*)
apply *auto*[*1*]
apply(*simp add*:*stable-def*)+

apply(*rule Basic*)
apply(*simp add*:*Schedule-guar-def*)
apply(*subgoal-tac inv* (*V*(|*cur* := *Some t*, *thd-state* := (*thd-state V*)(*the* (*cur V*) := *READY*, *t* := *RUNNING*)|)) ∧
(∀ *x*. (*V*, *V*(|*cur* := *Some t*, *thd-state* := (*thd-state V*)(*the* (*cur V*) := *READY*, *t* := *RUNNING*)|)) ∈ *lvars-nochange-rel x*))
apply *simp*
apply(*rule conjI*) **apply**(*simp add*:*inv-def*) **apply** *clarify*
apply(*rule conjI*) **apply**(*simp add*:*inv-cur-def*) **apply** *force*
apply(*rule conjI*) **apply**(*simp add*:*inv-thd-waitq-def inv-cur-def*)
apply (*metis Thread-State-Type*.*distinct*(*3*) *Thread-State-Type*.*distinct*(*6*))
apply(*rule conjI*) **apply**(*simp add*:*inv-mempool-info-def*)
apply(*rule conjI*) **apply**(*simp add*:*inv-bitmap-freelist-def*)
apply(*rule conjI*) **apply**(*simp add*:*inv-bitmap-def*)
apply(*rule conjI*) **apply**(*simp add*:*inv-aux-vars-def mem-block-addr-valid-def*)
apply(*rule conjI*) **apply**(*simp add*:*inv-bitmap0-def*)
apply(*rule conjI*) **apply**(*simp add*:*inv-bitmapn-def*)
apply(*simp add*:*inv-bitmap-not4free-def*)
apply *auto*[*1*] **using** *lvars-nochange-rel-def lvars-nochange-def* **apply** *simp*
apply(*simp add*: *stable-def*)+

apply(*rule Seq*[**where** *mid* = {*V*(|*cur* := *Some t*|)}])
apply(*rule Seq*[**where** *mid* = {*V*}])
apply(*rule Cond*)

apply(*simp add*:*stable-def*)
apply(*rule Seq*[**where** *mid* = {}])
apply(*rule Basic*)
apply *auto*[*1*]
apply(*simp add*:*stable-def*)+
apply(*rule Basic*)
apply *auto*[*1*]
apply(*simp add*:*stable-def*)+
apply(*simp add*:*Skip-def*) **apply**(*rule Basic*) **apply**(*simp add*:*stable-def*)+
apply(*rule Basic*)
apply *auto*[*1*]
apply(*simp add*:*stable-def*)+
apply(*rule Basic*)
apply(*simp add*:*Schedule-guar-def*)
apply(*subgoal-tac inv* (V(|*cur* := *Some t*, *thd-state* := (*thd-state* V)(t := *RUNNING*)|)) $\wedge$
($\forall x$. (V, V(|*cur* := *Some t*, *thd-state* := (*thd-state* V)(t := *RUNNING*)|)) $\in$ *lvars-nochange-rel* x))
apply *simp*
apply(*rule conjI*) **apply**(*simp add*:*inv-def*) **apply** *clarify*
apply(*rule conjI*) **apply**(*simp add*:*inv-cur-def*)
apply(*rule conjI*) **apply**(*simp add*:*inv-thd-waitq-def*) **apply** *auto*[*1*]
apply(*rule conjI*) **apply**(*simp add*:*inv-mempool-info-def*)
apply(*rule conjI*) **apply**(*simp add*:*inv-bitmap-freelist-def*)
apply(*rule conjI*) **apply**(*simp add*:*inv-bitmap-def*)
apply(*rule conjI*) **apply**(*simp add*:*inv-aux-vars-def mem-block-addr-valid-def*)
apply(*rule conjI*) **apply**(*simp add*:*inv-bitmap0-def*)
apply(*rule conjI*) **apply**(*simp add*:*inv-bitmapn-def*)
apply(*simp add*:*inv-bitmap-not4free-def*)

apply *auto*[*1*] **using** *lvars-nochange-rel-def lvars-nochange-def* **apply** *simp*
apply(*simp add*:*stable-def*)+
done

lemma *Schedule-satRG*: Γ (*Schedule t*) $\vdash$ *Schedule-RGCond t*
apply(*simp add*:*Evt-sat-RG-def*)
apply (*simp add*: *Schedule-def Schedule-RGCond-def*)
apply(*rule BasicEvt*)
apply(*simp add*:*body-def* Pre_f*-def* $Post_f$*-def guard-def*
$Rely_f$*-def* $Guar_f$*-def getrgformula-def*)
apply(*rule Await*)
using *stable-inv-sched-rely1* **apply** *simp* **using** *stable-inv-sched-rely1* **apply** *simp*
using *Schedule-satRG-h1* **apply** *simp*

apply(*simp add*:Pre_f*-def* $Rely_f$*-def getrgformula-def*)
using *stable-inv-sched-rely1* **apply** *simp*

by(*simp add*:$Guar_f$*-def getrgformula-def Schedule-guar-def*)

8 Functional correctness of Tick

lemma *Tick-satRG*: Γ *Tick* ⊢ *Tick-RGCond*
  apply(*simp add*:*Evt-sat-RG-def*)
  apply (*simp add*: *Tick-def Tick-RGCond-def Tick-rely-def Tick-guar-def*)
  apply(*rule BasicEvt*)
    apply(*simp add*:*body-def* Pre_f*-def* $Post_f$*-def guard-def*
        $Rely_f$*-def* $Guar_f$*-def getrgformula-def*)
    apply(*rule Basic*)
      apply *simp*
      using *lvars-nochange-rel-def lvars-nochange-def* **apply** *simp* **apply** *auto*[*1*]
      apply(*simp add*:*stable-def*)+
    apply(*simp add*: *stable-def* Pre_f*-def getrgformula-def* $Rely_f$*-def*) **apply** *auto*[*1*]
    by (*simp add*: $Guar_f$*-def getrgformula-def*)

end

theory *func-cor-mempoolfree*
imports *func-cor-lemma*
begin

9 Functional correctness of $k\_mem\_pool\_free$

9.1 intermediate conditions and their stable to rely cond

abbreviation *mp-free-precond1-ext t b* ≡
  {|*pool b* ∈ ´*mem-pools* ∧ *level b* < *length* (*levels* (´*mem-pool-info* (*pool b*)))
    ∧ *block b* < *length* (*bits* (*levels* (´*mem-pool-info* (*pool b*))!(*level b*)))
  ∧ *data b* = *block-ptr* (´*mem-pool-info* (*pool b*)) ((*ALIGN4* (*max-sz* (´*mem-pool-info* (*pool b*)))) *div* (*4* ˆ (*level b*))) (*block b*)|}

abbreviation *mp-free-precond1 t b* ≡
  Mem-pool-free-pre t ∩ *mp-free-precond1-ext t b*

lemma *mp-free-precond1-ext-stb*: *stable* (*mp-free-precond1-ext t b*) (*Mem-pool-free-rely t*)
  apply(*simp add*:*stable-def*) **apply** *clarify*
  apply(*rule conjI*) **apply**(*simp add*:*Mem-pool-free-rely-def gvars-conf-stable-def gvars-conf-def*) **apply** *metis*
  apply(*rule conjI*) **apply**(*simp add*:*Mem-pool-free-rely-def gvars-conf-stable-def*) **unfolding** *gvars-conf-def* **apply** *metis*
  apply(*rule conjI*)
  apply(*simp add*:*Mem-pool-free-rely-def gvars-conf-stable-def*) **unfolding** *gvars-conf-def* **apply** *metis*
    apply(*simp add*:*block-ptr-def*)
  apply(*simp add*:*Mem-pool-free-rely-def gvars-conf-stable-def gvars-conf-def*) **apply** *metis*

done

lemma *mp-free-precond1-stb* : *stable* (*mp-free-precond1 t b*) (*Mem-pool-free-rely t*)
  apply(*rule stable-int2*)
  apply(*simp add*:*mem-pool-free-pre-stb*)
  apply(*simp add*:*mp-free-precond1-ext-stb*)
done

abbreviation *mp-free-precond1-0 t b* ≡
  {*s. inv s* ∧ *allocating-node s t* = *None*} ∩ *mp-free-precond1-ext t b*

lemma *mp-free-precond1-0-stb*: *stable* (*mp-free-precond1-0 t b*) (*Mem-pool-free-rely t*)
  apply(*rule stable-int2*)
  apply(*rule subst*[**where** *t*=⦃´*inv* ∧ ´*allocating-node t* = *None*⦄
        and *s*=⦃´*inv*⦄ ∩ ⦃´*allocating-node t* = *None*⦄])
    apply *force*
  apply(*rule stable-int2*)
  apply(*simp add*:*stable-inv-free-rely1*)
  apply(*simp add*:*stable-def Mem-pool-free-rely-def*)
    apply(*simp add*:*lvars-nochange-rel-def lvars-nochange-def*)
  apply(*simp add*:*mp-free-precond1-ext-stb*)
done

abbreviation *mp-free-precond2-ext t b* ≡ ⦃´*freeing-node t* = *Some b*⦄

abbreviation *mp-free-precond2 t b* ≡
  mp-free-precond1-0 t b ∩ *mp-free-precond2-ext t b*

lemma *mp-free-precond2-ext-stb*: *stable* (*mp-free-precond2-ext t b*) (*Mem-pool-free-rely t*)

  apply(*simp add*:*stable-def*) **apply**(*rule allI*) **apply**(*rule impI*) **apply**(*rule allI*) **apply**(*rule impI*)
  apply(*simp add*:*Mem-pool-free-rely-def*)
  apply(*simp add*:*lvars-nochange-rel-def lvars-nochange-def*) **apply** *smt*
done

lemma *mp-free-precond2-stb*: *stable* (*mp-free-precond2 t b*) (*Mem-pool-free-rely t*)
  apply(*rule stable-int2*)

  apply(*simp add*:*mp-free-precond1-0-stb*)
  apply(*simp add*:*mp-free-precond2-ext-stb*)
done

abbreviation *mp-free-precond3-ext t b* ≡ ⦃´*need-resched t* = *False*⦄

abbreviation *mp-free-precond3 t b* ≡ (*mp-free-precond2 t b*) ∩ *mp-free-precond3-ext t b*

lemma *mp-free-precond3-ext-stb* : *stable* (*mp-free-precond3-ext t b*) (*Mem-pool-free-rely t*)
apply(*simp add*:*stable-def*) **apply**(*rule allI*) **apply**(*rule impI*) **apply**(*rule allI*) **apply**(*rule impI*)
apply(*simp add*:*Mem-pool-free-rely-def lvars-nochange-rel-def lvars-nochange-def*)
by *auto*

lemma *mp-free-precond3-stb* : *stable* (*mp-free-precond3 t b*) (*Mem-pool-free-rely t*)
apply(*rule stable-int2*)
using *mp-free-precond2-stb* **apply** *simp*
using *mp-free-precond3-ext-stb* **apply** *simp*
done

abbreviation *mp-free-precond4-ext t b* ≡ ⦃´*lsizes t* = [*ALIGN4* (*max-sz* (´*mem-pool-info* (*pool b*)))]⦄

abbreviation *mp-free-precond4 t b* ≡
mp-free-precond3 t b ∩ *mp-free-precond4-ext t b*

lemma *mp-free-precond4-ext-stb*:
stable (*mp-free-precond4-ext t b*) (*Mem-pool-free-rely t*)
apply(*simp add*:*stable-def*) **apply**(*rule allI*) **apply**(*rule impI*) **apply**(*rule allI*) **apply**(*rule impI*)
apply(*simp add*:*Mem-pool-free-rely-def ALIGN4-def*)
apply(*simp add*:*gvars-conf-stable-def gvars-conf-def*)
apply(*case-tac x = y*) **apply** *simp*
apply *clarify* **apply**(*simp add*: *lvars-nochange-rel-def lvars-nochange-def*)
done

lemma *mp-free-precond4-stb* : *stable* (*mp-free-precond4 t b*) (*Mem-pool-free-rely t*)
apply(*rule stable-int2*)
using *mp-free-precond3-stb* **apply** *simp*
using *mp-free-precond4-ext-stb* **apply** *blast*
done

abbreviation *mp-free-precond4-0-ext t b* ≡
⦃(∀ *ii*<*length* (´*lsizes t*). ´*lsizes t* ! *ii* = (*ALIGN4* (*max-sz* (´*mem-pool-info* (*pool b*)))) *div* (*4* ^ *ii*))
∧ *length* (´*lsizes t*) > *0*⦄

abbreviation *mp-free-precond4-0 t b* ≡ *mp-free-precond3 t b* ∩ *mp-free-precond4-0-ext t b*

lemma *mp-free-precond4-0-ext-stb*:
stable (*mp-free-precond4-0-ext t b*) (*Mem-pool-free-rely t*)
apply(*simp add*:*stable-def*) **apply**(*rule allI*) **apply**(*rule impI*) **apply**(*rule allI*)

apply(*rule impI*)
  apply(*simp add*:*Mem-pool-free-rely-def ALIGN4-def*)
    apply(*simp add*:*gvars-conf-stable-def gvars-conf-def*)
    apply(*case-tac* $x = y$) **apply** *simp*
    apply *clarify* **apply**(*simp add*: *lvars-nochange-rel-def lvars-nochange-def*)
done

lemma *mp-free-precond4-0-stb* : *stable* (*mp-free-precond4-0 t b*) (*Mem-pool-free-rely t*)
apply(*rule stable-int2*)
  using *mp-free-precond3-stb* **apply** *simp*
  using *mp-free-precond4-0-ext-stb* **apply** *blast*
done

abbreviation *mp-free-precond4-1 t b* ≡
  mp-free-precond4-0 t b ∩ ⦃*length* (´*lsizes t*) = ´*i t*⦄

lemma *mp-free-precond4-1-stb* : *stable* (*mp-free-precond4-1 t b*) (*Mem-pool-free-rely t*)
  apply(*rule stable-int2*)
  using *mp-free-precond4-0-stb* **apply** *simp*
 apply(*simp add*:*stable-def*) **apply**(*rule allI*) **apply**(*rule impI*) **apply**(*rule allI*)+
apply(*rule impI*)
 apply(*simp add*:*Mem-pool-free-rely-def lvars-nochange-rel-def lvars-nochange-def*)
apply *smt*
done

abbreviation *mp-free-precond4-2 t b* ≡
  mp-free-precond4-1 t b ∩ ⦃´*i t* ≤ *level b* ⦄

lemma *mp-free-precond4-2-stb* : *stable* (*mp-free-precond4-2 t b*) (*Mem-pool-free-rely t*)
  apply(*rule stable-int2*)
  using *mp-free-precond4-1-stb* **apply** *simp*
 apply(*simp add*:*stable-def*) **apply**(*rule allI*) **apply**(*rule impI*) **apply**(*rule allI*)+
apply(*rule impI*)
 apply(*simp add*:*Mem-pool-free-rely-def lvars-nochange-rel-def lvars-nochange-def*)
by *smt*

abbreviation *mp-free-precond4-3 t b* ≡
  mp-free-precond4-0 t b ∩ (⦃´*i t* ≤ *level b* ⦄ ∩ ⦃*length* (´*lsizes t*) = ´*i t* + *1*⦄)

lemma *mp-free-precond4-3-stb* : *stable* (*mp-free-precond4-3 t b*) (*Mem-pool-free-rely t*)
  apply(*rule stable-int2*)
  using *mp-free-precond4-0-stb* **apply** *simp*
 apply(*simp add*:*stable-def*) **apply**(*rule allI*) **apply**(*rule impI*) **apply**(*rule allI*)+
apply(*rule impI*)
 apply(*simp add*:*Mem-pool-free-rely-def lvars-nochange-rel-def lvars-nochange-def*)

by *smt*

abbreviation *mp-free-precond5-ext t b* ≡
{|(∀ *ii*<*length* (´*lsizes t*). ´*lsizes t* ! *ii* = (*ALIGN4* (*max-sz* (´*mem-pool-info* (*pool b*)))) *div* (*4* ^ *ii*))
∧ *length* (´*lsizes t*) > *level b*|}

abbreviation *mp-free-precond5 t b* ≡ *mp-free-precond3 t b* ∩ *mp-free-precond5-ext t b*
term *mp-free-precond5 t b*

lemma *mp-free-precond5-ext-stb*:
stable (*mp-free-precond5-ext t b*) (*Mem-pool-free-rely t*)
apply(*simp add*:*stable-def*) **apply**(*rule allI*) **apply**(*rule impI*) **apply**(*rule allI*) **apply**(*rule impI*)
apply(*simp add*:*Mem-pool-free-rely-def ALIGN4-def*)
apply(*simp add*:*gvars-conf-stable-def gvars-conf-def*)
apply(*case-tac x* = *y*) **apply** *simp*
apply *clarify* **apply**(*simp add*: *lvars-nochange-rel-def lvars-nochange-def*)
done

lemma *mp-free-precond5-stb* : *stable* (*mp-free-precond5 t b*) (*Mem-pool-free-rely t*)
apply(*rule stable-int2*)
using *mp-free-precond3-stb* **apply** *simp*
using *mp-free-precond5-ext-stb* **apply** *blast*
done

abbreviation *mp-free-precond6 t b* ≡
mp-free-precond5 t b ∩ {|´*free-block-r t* = *True*|}

lemma *mp-free-precond6-stb* : *stable* (*mp-free-precond6 t b*) (*Mem-pool-free-rely t*)

apply(*rule stable-int2*)
using *mp-free-precond5-stb* **apply** *simp*
apply(*simp add*:*stable-def*) **apply**(*rule allI*) **apply**(*rule impI*) **apply**(*rule allI*)+ **apply**(*rule impI*)
apply(*simp add*:*Mem-pool-free-rely-def lvars-nochange-rel-def lvars-nochange-def*)
by *auto*

abbreviation *mp-free-precond7 t b* ≡
mp-free-precond6 t b ∩ {|´*bn t* = *block b*|}

lemma *mp-free-precond7-stb* : *stable* (*mp-free-precond7 t b*) (*Mem-pool-free-rely t*)

apply(*rule stable-int2*)
using *mp-free-precond6-stb* **apply** *simp*
apply(*simp add*:*stable-def*) **apply**(*rule allI*) **apply**(*rule impI*) **apply**(*rule allI*)+

apply(*rule impI*)
apply(*simp add:Mem-pool-free-rely-def lvars-nochange-rel-def lvars-nochange-def*)
by *smt*

abbreviation *mp-free-precond8 t b* $\equiv$
mp-free-precond1-0 t b $\cap$ $\{\!|$*level b* $<$ *length* (´*lsizes t*)
$\wedge$ ($\forall$ *ii*<*length* (´*lsizes t*). ´*lsizes t* ! *ii* = (*ALIGN4* (*max-sz* (´*mem-pool-info* (*pool b*)))) *div* (*4* ^ *ii*))
$\wedge$ ´*bn t* $<$ *length* (*bits* (*levels* (´*mem-pool-info* (*pool b*))!(´*lvl t*)))
$\wedge$ ´*bn t* = (*block b*) *div* (*4* ^ (*level b* $-$ ´*lvl t*))
$\wedge$ ´*lvl t* $\leq$ *level b*
$\wedge$ (´*free-block-r t* $\longrightarrow$
($\exists$ *blk*. ´*freeing-node t* = *Some blk* $\wedge$ *pool blk* = *pool b* $\wedge$ *level blk* = ´*lvl t* $\wedge$ *block blk* = ´*bn t*)
$\wedge$ ´*alloc-memblk-data-valid* (*pool b*) (*the* (´*freeing-node t*)))
$\wedge$ ($\neg$ ´*free-block-r t* $\longrightarrow$ ´*freeing-node t* = *None*)
(*$\wedge$ ((*if* ´*freeing-node t* $\neq$ *None then* ´*lvl t* + *1 else 0*) $>$ *0*
$\longrightarrow$ ´*free-block-r t*)*) (* *this cond is implied by upper conds* *) $|\!\}$

abbreviation *mp-free-precond8-inv t b* α $\equiv$
mp-free-precond8 t b $\cap$ $\{\!|$ α = (*if* ´*freeing-node t* $\neq$ *None then* ´*lvl t* + *1 else 0*) $|\!\}$

lemma *inv-*α*gt0-imp-looppre*:
mp-free-precond8-inv t b α $\cap$ $\{\!|\alpha > 0|\!\}$ $\subseteq$ *mp-free-precond8 t b* $\cap$ $\{\!|$´*free-block-r t*$|\!\}$
by *auto*

lemma *looppre-imp-exist-*α*gt0*:
x $\in$ *mp-free-precond8 t b* $\cap$ $\{\!|$´*free-block-r t*$|\!\}$ $\Longrightarrow$ $\exists$ α. *x* $\in$ *mp-free-precond8-inv t b* α $\cap$ $\{\!|\alpha > 0|\!\}$
by *clarsimp*

lemma *x* $\in$ *mp-free-precond8-inv t b* α $\cap$ $\{\!|\alpha > 0|\!\}$ $\Longrightarrow$ *x* $\in$ *mp-free-precond8 t b* $\cap$ $\{\!|$´*free-block-r t*$|\!\}$
using *inv-*α*gt0-imp-looppre*[*of t b* α]
subsetI[*of mp-free-precond8-inv t b* α $\cap$ $\{\!|\alpha > 0|\!\}$
mp-free-precond8 t b $\cap$ $\{\!|$´*free-block-r t*$|\!\}$]
by *blast*

lemma *loopbody-sat-invterm-imp-inv-post*:
$\Gamma \vdash_I P\ sat_p$ [*mp-free-precond8-inv t b* α $\cap$ $\{\!|\alpha > 0|\!\}$, *rely*, *guar*, *mp-free-precond8-inv t b* (α $-$ *1*)]
$\Longrightarrow$ $\Gamma \vdash_I P\ sat_p$ [*mp-free-precond8-inv t b* α $\cap$ $\{\!|\alpha > 0|\!\}$, *rely*, *guar*,*mp-free-precond8 t b*]
using *Conseq* [*of mp-free-precond8-inv t b* α $\cap$ $\{\!|\alpha > 0|\!\}$ *mp-free-precond8-inv t b*

$\alpha \cap \{\!|\alpha > 0|\!\}$
rely rely guar guar mp-free-precond8-inv t b $(\alpha - 1)$
mp-free-precond8 t b P] **by** *blast*

lemma *stm8-inv-imp-prepost*:
$(\forall\,\alpha.\ \Gamma \vdash_I P\ sat_p$ [*mp-free-precond8-inv t b* $\alpha \cap \{\!|\alpha > 0|\!\}$, *rely*, *guar*, *mp-free-precond8-inv t b* $(\alpha - 1)$])
$\Longrightarrow \Gamma \vdash_I P\ sat_p$ [*mp-free-precond8 t b* $\cap$ $\{\!|$´*free-block-r t*$|\!\}$, *rely*, *guar*,*mp-free-precond8 t b*]

apply(*rule subst*[**where** *s*=$\forall\,v.\ v\in$*mp-free-precond8 t b* $\cap$ $\{\!|$´*free-block-r t*$|\!\}$ $\longrightarrow$
$\Gamma \vdash_I P\ sat_p$ [$\{v\}$, *rely*, *guar*,*mp-free-precond8 t b*] **and**
t=$\Gamma \vdash_I P\ sat_p$ [*mp-free-precond8 t b* $\cap$ $\{\!|$´*free-block-r t*$|\!\}$, *rely*, *guar*,*mp-free-precond8 t b*]])
using *allpre-eq-pre*[*of mp-free-precond8 t b* $\cap$ $\{\!|$´*free-block-r t*$|\!\}$
P rely guar mp-free-precond8 t b] **apply** *blast*

apply(*rule allI*) **apply**(*rule impI*)
apply(*subgoal-tac* $\exists\,\alpha.\ v \in$ *mp-free-precond8-inv t b* $\alpha \cap \{\!|\alpha > 0|\!\}$)
prefer *2* **using** *looppre-imp-exist-*α*gt0* **apply** *blast*

apply(*erule exE*)

using *sat-pre-imp-allinpre*[*of P - rely guar mp-free-precond8 t b*]
loopbody-sat-invterm-imp-inv-post **apply** *blast*
done

lemma *loopbody-sat-invterm-imp-inv-post2*:
$\exists\,\beta{<}\alpha.\ \Gamma \vdash_I P\ sat_p$ [*mp-free-precond8-inv t b* $\alpha \cap \{\!|\alpha > 0|\!\}$, *rely*, *guar*, *mp-free-precond8-inv t b* β]
$\Longrightarrow\ \Gamma \vdash_I P\ sat_p$ [*mp-free-precond8-inv t b* $\alpha \cap \{\!|\alpha > 0|\!\}$, *rely*, *guar*,*mp-free-precond8 t b*]
using *Conseq* [*of mp-free-precond8-inv t b* $\alpha \cap \{\!|\alpha > 0|\!\}$ *mp-free-precond8-inv t b* $\alpha \cap \{\!|\alpha > 0|\!\}$
rely rely guar guar mp-free-precond8-inv t b -
mp-free-precond8 t b P] **by** *blast*

lemma *stm8-inv-imp-prepost2*:
$(\forall\,\alpha.\ \exists\,\beta{<}\alpha.\ \Gamma \vdash_I P\ sat_p$ [*mp-free-precond8-inv t b* $\alpha \cap \{\!|\alpha > 0|\!\}$, *rely*, *guar*, *mp-free-precond8-inv t b* β])
$\Longrightarrow \Gamma \vdash_I P\ sat_p$ [*mp-free-precond8 t b* $\cap$ $\{\!|$´*free-block-r t*$|\!\}$, *rely*, *guar*,*mp-free-precond8 t b*]

apply(*rule subst*[**where** *s*=$\forall\,v.\ v\in$*mp-free-precond8 t b* $\cap$ $\{\!|$´*free-block-r t*$|\!\}$ $\longrightarrow$
$\Gamma \vdash_I P\ sat_p$ [$\{v\}$, *rely*, *guar*,*mp-free-precond8 t b*] **and**
t=$\Gamma \vdash_I P\ sat_p$ [*mp-free-precond8 t b* $\cap$ $\{\!|$´*free-block-r t*$|\!\}$, *rely*, *guar*,*mp-free-precond8 t b*]])

using *allpre-eq-pre*[*of mp-free-precond8 t b* ∩ {| ´*free-block-r t*|}
P rely guar mp-free-precond8 t b] **apply** *blast*

apply(*rule allI*) **apply**(*rule impI*)
apply(*subgoal-tac* ∃ *α*. *v* ∈ *mp-free-precond8-inv t b α* ∩ {|*α* > *0*|})
prefer *2* **using** *looppre-imp-exist-αgt0* **apply** *blast*

apply(*erule exE*)

using *sat-pre-imp-allinpre*[*of P - rely guar mp-free-precond8 t b*]
loopbody-sat-invterm-imp-inv-post **apply** *blast*
done

lemma *stm8-loopinv0*: *mp-free-precond8-inv t b 0* ⊆ {|¬ ´*free-block-r t*|}
by *auto*

lemma *stm8-loopinv-α*: *α* > *0* ⟹ *mp-free-precond8-inv t b α* ⊆ {|´*free-block-r t*|}
by *auto*

lemma *inv-αeq0-eq-looppre*:
mp-free-precond8-inv t b 0 = *mp-free-precond8 t b* ∩ {|¬ ´*free-block-r t*|}
by *auto*

term *mp-free-precond8 t b*

lemma *alloc-memblk-data-valid-stb-free*:
alloc-memblk-data-valid x (*pool b*) (*the* (*freeing-node x t*)) ⟹
(*x*, *y*) ∈ *lvars-nochange-rel t* ⟹
(*x*, *y*) ∈ *gvars-conf-stable* ⟹
alloc-memblk-data-valid y (*pool b*) (*the* (*freeing-node y t*))
apply(*subgoal-tac blk x t* = *blk y t*)
prefer *2* **apply**(*simp add*: *lvars-nochange-rel-def lvars-nochange-def*)
apply(*subgoal-tac buf* (*mem-pool-info x* (*pool b*)) = *buf* (*mem-pool-info y* (*pool b*)))
prefer *2* **apply**(*simp add*: *gvars-conf-stable-def gvars-conf-def*)
apply(*subgoal-tac lsizes x t* = *lsizes y t*)
prefer *2* **apply**(*simp add*: *lvars-nochange-rel-def lvars-nochange-def*)
apply(*subgoal-tac free-l x t* = *free-l y t*)
prefer *2* **apply**(*simp add*: *lvars-nochange-rel-def lvars-nochange-def*)
apply(*subgoal-tac max-sz* (*mem-pool-info x* (*pool b*)) = *max-sz* (*mem-pool-info y* (*pool b*)))
prefer *2* **apply**(*simp add*: *gvars-conf-stable-def gvars-conf-def*)
apply(*subgoal-tac freeing-node x t* = *freeing-node y t*)
prefer *2* **apply**(*simp add*: *lvars-nochange-rel-def lvars-nochange-def*)
apply (*simp add*: *gvars-conf-def gvars-conf-stable-def*)
done

lemma *mp-free-precond8-stb* : *stable* (*mp-free-precond8 t b*) (*Mem-pool-free-rely t*)

apply(*rule stable-int2*) **apply**(*rule stable-int2*)

apply(*simp add*:*stable-def*)
apply *clarify*
apply(*rule conjI*)
using *stable-inv-free-rely* **apply** *blast*
apply(*simp add*: *Mem-pool-free-rely-def lvars-nochange-rel-def lvars-nochange-def*)
apply *smt*

apply(*simp add*:*stable-def*)
apply *clarify*
apply(*rule conjI*)
apply(*simp add*:*Mem-pool-free-rely-def gvars-conf-stable-def gvars-conf-def*) **apply** *metis*
apply(*rule conjI*)
apply(*simp add*:*Mem-pool-free-rely-def gvars-conf-stable-def gvars-conf-def*) **apply** *metis*
apply(*rule conjI*)
apply(*simp add*:*Mem-pool-free-rely-def gvars-conf-stable-def gvars-conf-def*) **apply** *metis*
apply(*simp add*: *block-ptr-def ALIGN4-def lvars-nochange-rel-def lvars-nochange-def*

gvars-conf-stable-def gvars-conf-def)
apply(*simp add*:*Mem-pool-free-rely-def gvars-conf-stable-def gvars-conf-def*) **apply** *metis*

apply(*simp add*: *Mem-pool-free-rely-def stable-def*)
apply *clarify*
apply(*rule conjI*) **apply** *clarify*
apply(*rule conjI*)
apply(*simp add*: *gvars-conf-stable-def gvars-conf-def lvars-nochange-rel-def lvars-nochange-def*)
apply(*rule conjI*) **apply**(*simp add*: *ALIGN4-def lvars-nochange-rel-def lvars-nochange-def gvars-conf-stable-def gvars-conf-def*)
apply(*rule conjI*) **apply**(*simp add*: *ALIGN4-def lvars-nochange-rel-def lvars-nochange-def gvars-conf-stable-def gvars-conf-def*) **apply** *metis*
apply(*rule conjI*) **apply**(*simp add*: *lvars-nochange-rel-def lvars-nochange-def*) **apply** *metis*
apply(*rule conjI*) **apply**(*simp add*: *lvars-nochange-rel-def lvars-nochange-def*)
apply(*rule conjI*) **apply** *clarify*
apply(*rule conjI*) **apply**(*simp add*: *lvars-nochange-rel-def lvars-nochange-def*) **apply** *metis*
apply(*simp add*: *ALIGN4-def lvars-nochange-rel-def lvars-nochange-def gvars-conf-stable-def gvars-conf-def*)

apply *clarify* **apply**(*simp add*: *lvars-nochange-rel-def lvars-nochange-def*)
apply *clarify*

done

lemma *mp-free-precond8-inv-stb* : *stable* (*mp-free-precond8-inv t b* α) (*Mem-pool-free-rely t*)
  apply(*rule stable-int2*)
  using *mp-free-precond8-stb* **apply** *fast*
  apply(*unfold stable-def*) **apply** *clarify*

  apply(*subgoal-tac lvl x t = lvl y t*) **prefer** *2*
   apply(*simp add*:*Mem-pool-free-rely-def lvars-nochange-rel-def lvars-nochange-def*)
apply *smt*
  apply(*subgoal-tac freeing-node x t = freeing-node y t*) **prefer** *2*
   apply(*simp add*:*Mem-pool-free-rely-def lvars-nochange-rel-def lvars-nochange-def*)
apply *smt*
by *simp*

lemma *mp-free-precond8-inv-presv-rely*:
s∈*mp-free-precond8-inv t b* $\alpha \Longrightarrow$ (*s*,*r*)∈*Mem-pool-free-rely t* $\Longrightarrow \exists\beta\leq\alpha$. *r*∈*mp-free-precond8-inv t b* β
apply(*rule exI*[**where** *x*=α])
apply(*rule conjI*) **apply** *fast*
using *mp-free-precond8-inv-stb*[*of t b* α] **apply**(*unfold stable-def*) **apply** *blast*
done

abbreviation *mp-free-precond8-1 t b* $\alpha \equiv$
  mp-free-precond8-inv t b α ∩ {|α > *0*|}

lemma *mp-free-precond8-1-imp-free-block-r*:
mp-free-precond8-1 t b α ⊆ {|´*free-block-r t*|}
  using *stm8-loopinv-*α **by** *blast*

lemma *mp-free-precond8-1-stb* : *stable* (*mp-free-precond8-1 t b* α) (*Mem-pool-free-rely t*)
  apply(*rule stable-int2*)
  using *mp-free-precond8-inv-stb* **apply** *blast*
  apply(*simp add*:*stable-def*)
done

abbreviation *mp-free-precond8-1′ t b* ≡
  mp-free-precond8 t b ∩ {|´*free-block-r t*|}

lemma *mp-free-precond8-1′-stb* : *stable* (*mp-free-precond8-1′ t b*) (*Mem-pool-free-rely t*)
  apply(*rule stable-int2*)
  using *mp-free-precond8-stb* **apply** *blast*
  apply(*simp add*:*stable-def*) **apply** *clarify*
 apply(*simp add*:*Mem-pool-free-rely-def lvars-nochange-rel-def lvars-nochange-def*)
by *smt*

abbreviation *mp-free-precond8-2 t b* $\alpha \equiv$
mp-free-precond8-1 t b $\alpha \cap$ {|´*lsz t* = ´*lsizes t* ! (´*lvl t*)|}

lemma *mp-free-precond8-2-stb* : *stable* (*mp-free-precond8-2 t b* α) (*Mem-pool-free-rely t*)
apply(*rule stable-int2*)
using *mp-free-precond8-1-stb* **apply** *blast*
apply(*simp add*:*stable-def*) **apply** *clarify*
apply(*simp add*:*Mem-pool-free-rely-def lvars-nochange-rel-def lvars-nochange-def*)
by *smt*

abbreviation *mp-free-precond8-3 t b* $\alpha \equiv$
mp-free-precond8-2 t b $\alpha \cap$ {|´*blk t* = *block-ptr* (´*mem-pool-info* (*pool b*)) (´*lsz t*) (´*bn t*)|}

lemma *mp-free-precond8-3-stb* : *stable* (*mp-free-precond8-3 t b* α) (*Mem-pool-free-rely t*)
apply(*rule stable-int2*)
using *mp-free-precond8-2-stb* **apply** *blast*
apply(*simp add*:*stable-def block-ptr-def Mem-pool-free-rely-def*) **apply** *clarify*
apply(*case-tac x* = *y*) **apply** *simp* **apply** *clarsimp*
apply(*subgoal-tac blk x t* = *blk y t*)
apply(*subgoal-tac lsz x t* = *lsz y t*)
apply(*subgoal-tac bn x t* = *bn y t*)
apply(*subgoal-tac buf* (*mem-pool-info x* (*pool b*)) = *buf* (*mem-pool-info y* (*pool b*)))
apply *simp*
apply(*simp add*: *gvars-conf-stable-def gvars-conf-def*)
apply(*simp add*: *lvars-nochange-rel-def lvars-nochange-def*)
apply(*simp add*: *lvars-nochange-rel-def lvars-nochange-def*)
apply(*simp add*: *lvars-nochange-rel-def lvars-nochange-def*) **apply** *metis*
done

abbreviation *mp-free-precond9 t b* $\equiv$ *mp-free-precond1 t b*
term *mp-free-precond1 t b*

lemma *mp-free-precond9-stb* : *stable* (*mp-free-precond9 t b*) (*Mem-pool-free-rely t*)

using *mp-free-precond1-stb* **apply** *auto*[*1*]
done

9.2 proof of each statement

lemma *mempool-free-stm1-inv-mempool-info*:
inv-mempool-info Va $\wedge$ *inv-bitmap-freelist Va* $\Longrightarrow$
block b < *length* (*bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)) $\Longrightarrow$
level b < *length* (*levels* (*mem-pool-info Va* (*pool b*))) $\Longrightarrow$
pool b $\in$ *mem-pools Va* $\Longrightarrow$
get-bit (*mem-pool-info Va*) (*pool b*) (*level b*) (*block b*) = *ALLOCATED* $\Longrightarrow$

inv-mempool-info
$(Va(\!|mem\text{-}pool\text{-}info := (mem\text{-}pool\text{-}info\ Va)$
$(pool\ b := mem\text{-}pool\text{-}info\ Va\ (pool\ b)$
$(\!|levels := levels\ (mem\text{-}pool\text{-}info\ Va\ (pool\ b))$
$[level\ b := (levels\ (mem\text{-}pool\text{-}info\ Va\ (pool\ b))\ !\ level\ b)$
$(\!|bits := bits\ (levels\ (mem\text{-}pool\text{-}info\ Va\ (pool\ b))\ !\ level\ b)[block\ b$
$:= FREEING]|\!)]|\!)),$
$freeing\text{-}node := freeing\text{-}node\ Va(t \mapsto b)|\!))$
apply(*simp add*:*inv-mempool-info-def*)
apply(*rule conjI*) **apply** *metis*
apply(*rule conjI*) **apply** *metis*
apply(*rule conjI*) **apply** *metis*
apply(*rule conjI*) **apply**(*simp add*:*inv-bitmap-freelist-def*) **apply** (*simp add*:*Let-def*)
apply *auto*[*1*]
apply(*rule conjI*) **apply**(*simp add*:*inv-bitmap-freelist-def*) **apply** (*simp add*:*Let-def*)
apply(*rule allI*) **apply**(*rule impI*)
apply(*subgoal-tac* $(\forall\, i{<}length\ (levels\ (mem\text{-}pool\text{-}info\ Va\ (pool\ b))).$
$length\ (bits\ (levels\ (mem\text{-}pool\text{-}info\ Va\ (pool\ b))\ !\ i)) = n\text{-}max\ (mem\text{-}pool\text{-}info$
$Va\ (pool\ b)) * 4\ \hat{}\ i))$
apply(*case-tac* $i = level\ b$)
apply *auto*[*1*] **apply** *auto*[*1*]
apply(*simp add*:*Let-def*)
done

lemma *mempool-free-stm1-inv-bitmap-freelist*:
$inv\text{-}cur\ Va \wedge inv\text{-}thd\text{-}waitq\ Va \wedge inv\text{-}mempool\text{-}info\ Va \wedge inv\text{-}bitmap\text{-}freelist\ Va \wedge$
$inv\text{-}bitmap\ Va \wedge inv\text{-}aux\text{-}vars\ Va \Longrightarrow$
$block\ b < length\ (bits\ (levels\ (mem\text{-}pool\text{-}info\ Va\ (pool\ b))\ !\ level\ b)) \Longrightarrow$
$level\ b < length\ (levels\ (mem\text{-}pool\text{-}info\ Va\ (pool\ b))) \Longrightarrow$
$pool\ b \in mem\text{-}pools\ Va \Longrightarrow$
$get\text{-}bit\ (mem\text{-}pool\text{-}info\ Va)\ (pool\ b)\ (level\ b)\ (block\ b) = ALLOCATED \Longrightarrow$
inv-bitmap-freelist
$(Va(\!|mem\text{-}pool\text{-}info := (mem\text{-}pool\text{-}info\ Va)$
$(pool\ b := mem\text{-}pool\text{-}info\ Va\ (pool\ b)$
$(\!|levels := levels\ (mem\text{-}pool\text{-}info\ Va\ (pool\ b))$
$[level\ b := (levels\ (mem\text{-}pool\text{-}info\ Va\ (pool\ b))\ !\ level\ b)$
$(\!|bits := bits\ (levels\ (mem\text{-}pool\text{-}info\ Va\ (pool\ b))\ !\ level\ b)[block\ b$
$:= FREEING]|\!)]|\!)),$
$freeing\text{-}node := freeing\text{-}node\ Va(t \mapsto b)|\!))$
apply(*simp add*:*inv-bitmap-freelist-def*)
apply(*rule allI*) **apply**(*rule impI*)+ **apply**(*simp add*:*Let-def*)
apply(*rule conjI*) **apply**(*rule allI*) **apply**(*rule impI*)
apply(*case-tac* $i = level\ b \wedge j = block\ b$) **apply** *auto*[*1*] **apply** *fastforce*
apply(*case-tac* $i \neq level\ b$) **apply** *auto*[*1*]
apply(*case-tac* $j \neq block\ b$) **apply** *auto*[*1*]
apply *auto*[*1*]

apply(*rule conjI*) **apply**(*rule allI*) **apply**(*rule impI*)
apply(*case-tac* $i = level\ b \wedge j = block\ b$) **apply** *auto*[*1*]

apply(*case-tac i* ≠ *level b*) **apply** *auto*[*1*]
apply(*case-tac j* ≠ *block b*) **apply** *auto*[*1*]
apply *auto*[*1*]
apply(*simp add*:*distinct-def*)
apply(*case-tac i* = *level b*) **apply** *auto*[*1*]
apply *auto*[*1*]
done

lemma *mempool-free-stm1-inv-bitmap*:
inv-cur Va ∧ *inv-thd-waitq Va* ∧ *inv-mempool-info Va* ∧ *inv-bitmap-freelist Va* ∧ *inv-bitmap Va* ∧ *inv-aux-vars Va* ⟹
block b < *length* (*bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)) ⟹
level b < *length* (*levels* (*mem-pool-info Va* (*pool b*))) ⟹
pool b ∈ *mem-pools Va* ⟹
get-bit (*mem-pool-info Va*) (*pool b*) (*level b*) (*block b*) = *ALLOCATED* ⟹
inv-bitmap
(*Va*⦇*mem-pool-info* := (*mem-pool-info Va*)
(*pool b* := *mem-pool-info Va* (*pool b*)
⦇*levels* := *levels* (*mem-pool-info Va* (*pool b*))
[*level b* := (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)
⦇*bits* := *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)[*block b* := *FREEING*]⦈]⦈),
freeing-node := *freeing-node Va*(*t* ↦ *b*)⦈)
apply(*simp add*:*inv-bitmap-def*)
apply(*rule allI*) **apply**(*simp add*:*Let-def*) **apply**(*rule impI*) **apply**(*rule allI*) **apply**(*rule impI*)
apply(*rule conjI*) **apply**(*rule impI*)
apply(*rule conjI*)
apply(*case-tac i* = *level b* ∧ *j* = *block b*) **apply** *auto*[*1*]
apply(*case-tac i* − *1* = *level b* ∧ *j div 4* = *block b*)
apply (*metis* (*no-types*, *lifting*) *BlockState*.*distinct*(*3*) *One-nat-def Suc-pred lessI nat-neq-iff nth-list-update-neq*)
apply(*rule impI*)
apply(*subgoal-tac bits* (*levels* (*mem-pool-info Va* (*pool b*))
[*level b* := (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)
⦇*bits* := *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)[*block b* := *FREEING*]⦈] ! *i*) ! *j*
= *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *i*) ! *j*)
apply(*subgoal-tac bits* (*levels* (*mem-pool-info Va* (*pool b*))
[*level b* := (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)
⦇*bits* := *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)[*block b* := *FREEING*]⦈] ! (*i* − *Suc NULL*)) ! (*j div 4*)
= *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! (*i* − *Suc NULL*)) ! (*j div 4*))
apply (*metis Mem-pool-lvl*.*cases Mem-pool-lvl*.*simps*(*1*) *Mem-pool-lvl*.*simps*(*4*)

length-list-update nth-list-update-eq nth-list-update-neq)
apply (*metis Mem-pool-lvl*.*cases Mem-pool-lvl*.*simps*(*1*) *Mem-pool-lvl*.*simps*(*4*)

One-nat-def nth-list-update-eq nth-list-update-neq)
apply (*metis* (*no-types*, *lifting*) *Mem-pool-lvl.cases Mem-pool-lvl.simps*(*1*)
Mem-pool-lvl.simps(*4*) *nth-list-update-eq nth-list-update-neq*)

apply(*rule impI*)
apply(*rule conjI*)
apply(*case-tac* $i = level\ b \wedge j = block\ b$) **apply** *auto*[*1*]
apply(*case-tac* $Suc\ i = level\ b \wedge j * 4 = block\ b$)
apply (*metis BlockState.distinct*(*5*) *less-Suc-eq nth-list-update-neq order-less-irrefl*)
apply(*subgoal-tac bits* (*levels* (*mem-pool-info Va* (*pool b*))
[*level b* := (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)
(|*bits* := *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)[*block b*
:= *FREEING*]|)] ! *i*) ! *j*
= *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *i*) ! *j*)
apply(*subgoal-tac bits* (*levels* (*mem-pool-info Va* (*pool b*))
[*level b* := (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)
(|*bits* := *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)[*block b*
:= *FREEING*]|)] ! (*Suc i*)) ! (*j* ∗ *4*)
= *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! (*Suc i*)) ! (*j* ∗ *4*))
apply (*metis Mem-pool-lvl.cases Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*)

length-list-update nth-list-update-eq nth-list-update-neq)
apply (*metis Mem-pool-lvl.cases Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*)

nth-list-update-eq nth-list-update-neq)
apply (*metis* (*no-types*, *lifting*) *Mem-pool-lvl.cases Mem-pool-lvl.simps*(*1*)
Mem-pool-lvl.simps(*4*) *nth-list-update-eq nth-list-update-neq*)
apply(*rule conjI*)
apply(*case-tac* $i = level\ b \wedge j = block\ b$) **apply** *auto*[*1*]
apply(*case-tac* $Suc\ i = level\ b \wedge Suc\ (j * 4) = block\ b$)
apply (*metis BlockState.distinct*(*5*) *less-Suc-eq nth-list-update-neq order-less-irrefl*)
apply(*subgoal-tac bits* (*levels* (*mem-pool-info Va* (*pool b*))
[*level b* := (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)
(|*bits* := *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)[*block b*
:= *FREEING*]|)] ! *i*) ! *j*
= *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *i*) ! *j*)
apply(*subgoal-tac bits* (*levels* (*mem-pool-info Va* (*pool b*))
[*level b* := (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)
(|*bits* := *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)[*block b*
:= *FREEING*]|)] ! (*Suc i*)) ! *Suc* (*j* ∗ *4*)
= *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! (*Suc i*)) ! *Suc* (*j* ∗ *4*))
apply (*metis Mem-pool-lvl.cases Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*)

length-list-update nth-list-update-eq nth-list-update-neq)
apply (*metis Mem-pool-lvl.cases Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*)

nth-list-update-eq nth-list-update-neq)
apply (*metis* (*no-types*, *lifting*) *Mem-pool-lvl.cases Mem-pool-lvl.simps*(*1*)
Mem-pool-lvl.simps(*4*) *nth-list-update-eq nth-list-update-neq*)

apply(*rule conjI*)
  apply(*case-tac* $i = level\ b \wedge j = block\ b$) **apply** *auto*[*1*]
  apply(*case-tac* $Suc\ i = level\ b \wedge Suc\ (Suc\ (j * 4)) = block\ b$)
 apply (*metis BlockState.distinct*(*5*) *less-Suc-eq nth-list-update-neq order-less-irrefl*)
  apply(*subgoal-tac bits* (*levels* (*mem-pool-info Va* (*pool b*))
     [*level b* := (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)
      (|*bits* := *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)[*block b*
:= *FREEING*]|)] ! *i*) ! *j*
     = *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *i*) ! *j*)
  apply(*subgoal-tac bits* (*levels* (*mem-pool-info Va* (*pool b*))
     [*level b* := (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)
      (|*bits* := *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)[*block b*
:= *FREEING*]|)] ! (*Suc i*)) ! *Suc* (*Suc* (*j* ∗ *4*))
     = *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! (*Suc i*)) ! *Suc* (*Suc* (*j* ∗
4)))
 apply (*metis Mem-pool-lvl.cases Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*)

   length-list-update nth-list-update-eq nth-list-update-neq)
 apply (*metis Mem-pool-lvl.cases Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*)

   nth-list-update-eq nth-list-update-neq)
  apply (*metis* (*no-types*, *lifting*) *Mem-pool-lvl.cases Mem-pool-lvl.simps*(*1*)
   Mem-pool-lvl.simps(*4*) *nth-list-update-eq nth-list-update-neq*)

  apply(*case-tac* $i = level\ b \wedge j = block\ b$) **apply** *auto*[*1*]
  apply(*case-tac* $Suc\ i = level\ b \wedge (j * 4 + 3) = block\ b$)
 apply (*metis BlockState.distinct*(*5*) *less-Suc-eq nth-list-update-neq order-less-irrefl*)
  apply(*subgoal-tac bits* (*levels* (*mem-pool-info Va* (*pool b*))
     [*level b* := (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)
      (|*bits* := *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)[*block b*
:= *FREEING*]|)] ! *i*) ! *j*
     = *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *i*) ! *j*)
  apply(*subgoal-tac bits* (*levels* (*mem-pool-info Va* (*pool b*))
     [*level b* := (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)
      (|*bits* := *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)[*block b*
:= *FREEING*]|)] ! (*Suc i*)) ! (*j* ∗ *4* + *3*)
     = *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! (*Suc i*)) ! (*j* ∗ *4* + *3*))
 apply (*metis Mem-pool-lvl.cases Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*)

   length-list-update nth-list-update-eq nth-list-update-neq)
 apply (*metis Mem-pool-lvl.cases Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*)

   nth-list-update-eq nth-list-update-neq)
  apply (*metis* (*no-types*, *lifting*) *Mem-pool-lvl.cases Mem-pool-lvl.simps*(*1*)
   Mem-pool-lvl.simps(*4*) *nth-list-update-eq nth-list-update-neq*)

apply(*rule conjI*) **apply**(*rule impI*)
 apply(*rule conjI*) **apply**(*rule impI*)
  apply(*case-tac* $i = level\ b \wedge j = block\ b$) **apply** *auto*[*1*]

apply(*case-tac* $i - 1 = level\ b \wedge j\ div\ 4 = block\ b$)
apply (*metis* (*no-types*, *lifting*) *BlockState.distinct*(*3*) *One-nat-def Suc-pred lessI nat-neq-iff nth-list-update-neq*)
apply(*subgoal-tac* $bits\ (levels\ (mem\text{-}pool\text{-}info\ Va\ (pool\ b))$
$[level\ b := (levels\ (mem\text{-}pool\text{-}info\ Va\ (pool\ b))\ !\ level\ b)$
$(\|bits := bits\ (levels\ (mem\text{-}pool\text{-}info\ Va\ (pool\ b))\ !\ level\ b)[block\ b := FREEING]\|)]\ !\ i)\ !\ j$
$= bits\ (levels\ (mem\text{-}pool\text{-}info\ Va\ (pool\ b))\ !\ i)\ !\ j$)
apply(*subgoal-tac* $bits\ (levels\ (mem\text{-}pool\text{-}info\ Va\ (pool\ b))$
$[level\ b := (levels\ (mem\text{-}pool\text{-}info\ Va\ (pool\ b))\ !\ level\ b)$
$(\|bits := bits\ (levels\ (mem\text{-}pool\text{-}info\ Va\ (pool\ b))\ !\ level\ b)[block\ b := FREEING]\|)]\ !\ (i - Suc\ NULL))\ !\ (j\ div\ 4)$
$= bits\ (levels\ (mem\text{-}pool\text{-}info\ Va\ (pool\ b))\ !\ (i - Suc\ NULL))\ !\ (j\ div\ 4)$)
apply (*metis Mem-pool-lvl.cases Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*)

length-list-update nth-list-update-eq nth-list-update-neq)
apply (*metis Mem-pool-lvl.cases Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*)

One-nat-def nth-list-update-eq nth-list-update-neq)
apply (*metis* (*no-types*, *lifting*) *Mem-pool-lvl.cases Mem-pool-lvl.simps*(*1*)
Mem-pool-lvl.simps(*4*) *nth-list-update-eq nth-list-update-neq*)
apply(*rule impI*)
apply(*rule conjI*)
apply(*case-tac* $i = level\ b \wedge j = block\ b$) **apply** *auto*[*1*]
apply(*case-tac* $Suc\ i = level\ b \wedge j * 4 = block\ b$)
apply (*metis BlockState.distinct*(*5*) *less-Suc-eq nth-list-update-neq order-less-irrefl*)
apply(*subgoal-tac* $bits\ (levels\ (mem\text{-}pool\text{-}info\ Va\ (pool\ b))$
$[level\ b := (levels\ (mem\text{-}pool\text{-}info\ Va\ (pool\ b))\ !\ level\ b)$
$(\|bits := bits\ (levels\ (mem\text{-}pool\text{-}info\ Va\ (pool\ b))\ !\ level\ b)[block\ b := FREEING]\|)]\ !\ i)\ !\ j$
$= bits\ (levels\ (mem\text{-}pool\text{-}info\ Va\ (pool\ b))\ !\ i)\ !\ j$)
apply(*subgoal-tac* $bits\ (levels\ (mem\text{-}pool\text{-}info\ Va\ (pool\ b))$
$[level\ b := (levels\ (mem\text{-}pool\text{-}info\ Va\ (pool\ b))\ !\ level\ b)$
$(\|bits := bits\ (levels\ (mem\text{-}pool\text{-}info\ Va\ (pool\ b))\ !\ level\ b)[block\ b := FREEING]\|)]\ !\ (Suc\ i))\ !\ (j * 4)$
$= bits\ (levels\ (mem\text{-}pool\text{-}info\ Va\ (pool\ b))\ !\ (Suc\ i))\ !\ (j * 4)$)
apply (*metis Mem-pool-lvl.cases Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*)

length-list-update nth-list-update-eq nth-list-update-neq)
apply (*metis Mem-pool-lvl.cases Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*)

nth-list-update-eq nth-list-update-neq)
apply (*metis* (*no-types*, *lifting*) *Mem-pool-lvl.cases Mem-pool-lvl.simps*(*1*)
Mem-pool-lvl.simps(*4*) *nth-list-update-eq nth-list-update-neq*)
apply(*rule conjI*)
apply(*case-tac* $i = level\ b \wedge j = block\ b$) **apply** *auto*[*1*]
apply(*case-tac* $Suc\ i = level\ b \wedge Suc\ (j * 4) = block\ b$)
apply (*metis BlockState.distinct*(*5*) *less-Suc-eq nth-list-update-neq order-less-irrefl*)

```
    apply(subgoal-tac bits (levels (mem-pool-info Va (pool b))
                [level b := (levels (mem-pool-info Va (pool b)) ! level b)
                  (|bits := bits (levels (mem-pool-info Va (pool b)) ! level b)[block b
:= FREEING]|)] ! i) ! j
                = bits (levels (mem-pool-info Va (pool b)) ! i) ! j)
    apply(subgoal-tac bits (levels (mem-pool-info Va (pool b))
                [level b := (levels (mem-pool-info Va (pool b)) ! level b)
                  (|bits := bits (levels (mem-pool-info Va (pool b)) ! level b)[block b
:= FREEING]|)] ! (Suc i)) ! Suc (j * 4)
                = bits (levels (mem-pool-info Va (pool b)) ! (Suc i)) ! Suc (j * 4))
   apply (metis Mem-pool-lvl.cases Mem-pool-lvl.simps(1) Mem-pool-lvl.simps(4)

            length-list-update nth-list-update-eq nth-list-update-neq)
   apply (metis Mem-pool-lvl.cases Mem-pool-lvl.simps(1) Mem-pool-lvl.simps(4)

             nth-list-update-eq nth-list-update-neq)
    apply (metis (no-types, lifting) Mem-pool-lvl.cases Mem-pool-lvl.simps(1)
            Mem-pool-lvl.simps(4) nth-list-update-eq nth-list-update-neq)
   apply(rule conjI)
    apply(case-tac i = level b ∧ j = block b) apply auto[1]
    apply(case-tac Suc i = level b ∧ Suc (Suc (j * 4)) = block b)
   apply (metis BlockState.distinct(5) less-Suc-eq nth-list-update-neq order-less-irrefl)
    apply(subgoal-tac bits (levels (mem-pool-info Va (pool b))
                [level b := (levels (mem-pool-info Va (pool b)) ! level b)
                  (|bits := bits (levels (mem-pool-info Va (pool b)) ! level b)[block b
:= FREEING]|)] ! i) ! j
                = bits (levels (mem-pool-info Va (pool b)) ! i) ! j)
    apply(subgoal-tac bits (levels (mem-pool-info Va (pool b))
                [level b := (levels (mem-pool-info Va (pool b)) ! level b)
                  (|bits := bits (levels (mem-pool-info Va (pool b)) ! level b)[block b
:= FREEING]|)] ! (Suc i)) ! Suc (Suc (j * 4))
                = bits (levels (mem-pool-info Va (pool b)) ! (Suc i)) ! Suc (Suc (j *
4)))
   apply (metis Mem-pool-lvl.cases Mem-pool-lvl.simps(1) Mem-pool-lvl.simps(4)

            length-list-update nth-list-update-eq nth-list-update-neq)
   apply (metis Mem-pool-lvl.cases Mem-pool-lvl.simps(1) Mem-pool-lvl.simps(4)

             nth-list-update-eq nth-list-update-neq)
    apply (metis (no-types, lifting) Mem-pool-lvl.cases Mem-pool-lvl.simps(1)
            Mem-pool-lvl.simps(4) nth-list-update-eq nth-list-update-neq)

    apply(case-tac i = level b ∧ j = block b) apply auto[1]
    apply(case-tac Suc i = level b ∧ (j * 4 + 3) = block b)
   apply (metis BlockState.distinct(5) less-Suc-eq nth-list-update-neq order-less-irrefl)
    apply(subgoal-tac bits (levels (mem-pool-info Va (pool b))
                [level b := (levels (mem-pool-info Va (pool b)) ! level b)
                  (|bits := bits (levels (mem-pool-info Va (pool b)) ! level b)[block b
:= FREEING]|)] ! i) ! j
```

$= bits\ (levels\ (mem\text{-}pool\text{-}info\ Va\ (pool\ b))\ !\ i)\ !\ j)$

apply(*subgoal-tac bits (levels (mem-pool-info Va (pool b))*
[level b := (levels (mem-pool-info Va (pool b)) ! level b)
*(|bits := bits (levels (mem-pool-info Va (pool b)) ! level b)[block b := FREEING]|)] ! (Suc i)) ! (j * 4 + 3)*
*= bits (levels (mem-pool-info Va (pool b)) ! (Suc i)) ! (j * 4 + 3))*

apply (*metis Mem-pool-lvl.cases Mem-pool-lvl.simps(1) Mem-pool-lvl.simps(4)*

length-list-update nth-list-update-eq nth-list-update-neq)

apply (*metis Mem-pool-lvl.cases Mem-pool-lvl.simps(1) Mem-pool-lvl.simps(4)*

nth-list-update-eq nth-list-update-neq)

apply (*metis (no-types, lifting) Mem-pool-lvl.cases Mem-pool-lvl.simps(1) Mem-pool-lvl.simps(4) nth-list-update-eq nth-list-update-neq*)

apply(*rule conjI*) **apply**(*rule impI*)
apply(*rule conjI*) **apply**(*rule impI*)
apply(*case-tac i = level b ∧ j = block b*) **apply** *auto[1]*
apply(*case-tac i − 1 = level b ∧ j div 4 = block b*)
apply (*metis (no-types, lifting) BlockState.distinct(3) One-nat-def Suc-pred lessI nat-neq-iff nth-list-update-neq*)
apply(*subgoal-tac bits (levels (mem-pool-info Va (pool b))*
[level b := (levels (mem-pool-info Va (pool b)) ! level b)
(|bits := bits (levels (mem-pool-info Va (pool b)) ! level b)[block b := FREEING]|)] ! i) ! j
= bits (levels (mem-pool-info Va (pool b)) ! i) ! j)
apply(*subgoal-tac bits (levels (mem-pool-info Va (pool b))*
[level b := (levels (mem-pool-info Va (pool b)) ! level b)
(|bits := bits (levels (mem-pool-info Va (pool b)) ! level b)[block b := FREEING]|)] ! (i − Suc NULL)) ! (j div 4)
= bits (levels (mem-pool-info Va (pool b)) ! (i − Suc NULL)) ! (j div 4))

apply (*metis Mem-pool-lvl.cases Mem-pool-lvl.simps(1) Mem-pool-lvl.simps(4)*

length-list-update nth-list-update-eq nth-list-update-neq)

apply (*metis Mem-pool-lvl.cases Mem-pool-lvl.simps(1) Mem-pool-lvl.simps(4)*

One-nat-def nth-list-update-eq nth-list-update-neq)

apply (*metis (no-types, lifting) Mem-pool-lvl.cases Mem-pool-lvl.simps(1) Mem-pool-lvl.simps(4) nth-list-update-eq nth-list-update-neq*)
apply(*rule impI*)
apply(*rule conjI*)
apply(*case-tac i = level b ∧ j = block b*) **apply** *auto[1]*
apply(*case-tac Suc i = level b ∧ j * 4 = block b*)
apply (*metis BlockState.distinct(5) less-Suc-eq nth-list-update-neq order-less-irrefl*)
apply(*subgoal-tac bits (levels (mem-pool-info Va (pool b))*
[level b := (levels (mem-pool-info Va (pool b)) ! level b)
(|bits := bits (levels (mem-pool-info Va (pool b)) ! level b)[block b

:= *FREEING*]⦈] ! *i*) ! *j*
= *bits* (*levels* (*mem-pool-info* *Va* (*pool* *b*)) ! *i*) ! *j*)
apply(*subgoal-tac* *bits* (*levels* (*mem-pool-info* *Va* (*pool* *b*))
[*level* *b* := (*levels* (*mem-pool-info* *Va* (*pool* *b*)) ! *level* *b*)
⦇*bits* := *bits* (*levels* (*mem-pool-info* *Va* (*pool* *b*)) ! *level* *b*)[*block* *b*
:= *FREEING*]⦈] ! (*Suc* *i*)) ! (*j* ∗ *4*)
= *bits* (*levels* (*mem-pool-info* *Va* (*pool* *b*)) ! (*Suc* *i*)) ! (*j* ∗ *4*))
apply (*metis* *Mem-pool-lvl.cases* *Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*)

length-list-update *nth-list-update-eq* *nth-list-update-neq*)
apply (*metis* *Mem-pool-lvl.cases* *Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*)

nth-list-update-eq *nth-list-update-neq*)
apply (*metis* (*no-types*, *lifting*) *Mem-pool-lvl.cases* *Mem-pool-lvl.simps*(*1*)
Mem-pool-lvl.simps(*4*) *nth-list-update-eq* *nth-list-update-neq*)
apply(*rule* *conjI*)
apply(*case-tac* *i* = *level* *b* ∧ *j* = *block* *b*) **apply** *auto*[*1*]
apply(*case-tac* *Suc* *i* = *level* *b* ∧ *Suc* (*j* ∗ *4*) = *block* *b*)
apply (*metis* *BlockState.distinct*(*5*) *less-Suc-eq* *nth-list-update-neq* *order-less-irrefl*)
apply(*subgoal-tac* *bits* (*levels* (*mem-pool-info* *Va* (*pool* *b*))
[*level* *b* := (*levels* (*mem-pool-info* *Va* (*pool* *b*)) ! *level* *b*)
⦇*bits* := *bits* (*levels* (*mem-pool-info* *Va* (*pool* *b*)) ! *level* *b*)[*block* *b*
:= *FREEING*]⦈] ! *i*) ! *j*
= *bits* (*levels* (*mem-pool-info* *Va* (*pool* *b*)) ! *i*) ! *j*)
apply(*subgoal-tac* *bits* (*levels* (*mem-pool-info* *Va* (*pool* *b*))
[*level* *b* := (*levels* (*mem-pool-info* *Va* (*pool* *b*)) ! *level* *b*)
⦇*bits* := *bits* (*levels* (*mem-pool-info* *Va* (*pool* *b*)) ! *level* *b*)[*block* *b*
:= *FREEING*]⦈] ! (*Suc* *i*)) ! *Suc* (*j* ∗ *4*)
= *bits* (*levels* (*mem-pool-info* *Va* (*pool* *b*)) ! (*Suc* *i*)) ! *Suc* (*j* ∗ *4*))
apply (*metis* *Mem-pool-lvl.cases* *Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*)

length-list-update *nth-list-update-eq* *nth-list-update-neq*)
apply (*metis* *Mem-pool-lvl.cases* *Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*)

nth-list-update-eq *nth-list-update-neq*)
apply (*metis* (*no-types*, *lifting*) *Mem-pool-lvl.cases* *Mem-pool-lvl.simps*(*1*)
Mem-pool-lvl.simps(*4*) *nth-list-update-eq* *nth-list-update-neq*)
apply(*rule* *conjI*)
apply(*case-tac* *i* = *level* *b* ∧ *j* = *block* *b*) **apply** *auto*[*1*]
apply(*case-tac* *Suc* *i* = *level* *b* ∧ *Suc* (*Suc* (*j* ∗ *4*)) = *block* *b*)
apply (*metis* *BlockState.distinct*(*5*) *less-Suc-eq* *nth-list-update-neq* *order-less-irrefl*)
apply(*subgoal-tac* *bits* (*levels* (*mem-pool-info* *Va* (*pool* *b*))
[*level* *b* := (*levels* (*mem-pool-info* *Va* (*pool* *b*)) ! *level* *b*)
⦇*bits* := *bits* (*levels* (*mem-pool-info* *Va* (*pool* *b*)) ! *level* *b*)[*block* *b*
:= *FREEING*]⦈] ! *i*) ! *j*
= *bits* (*levels* (*mem-pool-info* *Va* (*pool* *b*)) ! *i*) ! *j*)
apply(*subgoal-tac* *bits* (*levels* (*mem-pool-info* *Va* (*pool* *b*))
[*level* *b* := (*levels* (*mem-pool-info* *Va* (*pool* *b*)) ! *level* *b*)
⦇*bits* := *bits* (*levels* (*mem-pool-info* *Va* (*pool* *b*)) ! *level* *b*)[*block* *b*

:= *FREEING*]⦈] ! (*Suc i*)) ! *Suc* (*Suc* (*j* ∗ *4*))
= *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! (*Suc i*)) ! *Suc* (*Suc* (*j* ∗ *4*)))
apply (*metis Mem-pool-lvl.cases Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*)

length-list-update nth-list-update-eq nth-list-update-neq)
apply (*metis Mem-pool-lvl.cases Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*)

nth-list-update-eq nth-list-update-neq)
apply (*metis* (*no-types*, *lifting*) *Mem-pool-lvl.cases Mem-pool-lvl.simps*(*1*)
Mem-pool-lvl.simps(*4*) *nth-list-update-eq nth-list-update-neq*)

apply(*case-tac i* = *level b* ∧ *j* = *block b*) **apply** *auto*[*1*]
apply(*case-tac Suc i* = *level b* ∧ (*j* ∗ *4* + *3*) = *block b*)
apply (*metis BlockState.distinct*(*5*) *less-Suc-eq nth-list-update-neq order-less-irrefl*)
apply(*subgoal-tac bits* (*levels* (*mem-pool-info Va* (*pool b*))
[*level b* := (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)
⦇*bits* := *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)[*block b* := *FREEING*]⦈] ! *i*) ! *j*
= *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *i*) ! *j*)
apply(*subgoal-tac bits* (*levels* (*mem-pool-info Va* (*pool b*))
[*level b* := (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)
⦇*bits* := *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)[*block b* := *FREEING*]⦈] ! (*Suc i*)) ! (*j* ∗ *4* + *3*)
= *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! (*Suc i*)) ! (*j* ∗ *4* + *3*))
apply (*metis Mem-pool-lvl.cases Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*)

length-list-update nth-list-update-eq nth-list-update-neq)
apply (*metis Mem-pool-lvl.cases Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*)

nth-list-update-eq nth-list-update-neq)
apply (*metis* (*no-types*, *lifting*) *Mem-pool-lvl.cases Mem-pool-lvl.simps*(*1*)
Mem-pool-lvl.simps(*4*) *nth-list-update-eq nth-list-update-neq*)

apply(*rule conjI*) **apply**(*rule impI*)
apply(*rule conjI*) **apply**(*rule impI*)
apply(*case-tac i* = *level b* ∧ *j* = *block b*) **apply** *auto*[*1*]
apply(*case-tac i* − *1* = *level b* ∧ *j div 4* = *block b*)
apply (*metis* (*no-types*, *lifting*) *BlockState.distinct*(*3*) *One-nat-def Suc-pred lessI nat-neq-iff nth-list-update-neq*)
apply(*subgoal-tac bits* (*levels* (*mem-pool-info Va* (*pool b*))
[*level b* := (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)
⦇*bits* := *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)[*block b* := *FREEING*]⦈] ! *i*) ! *j*
= *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *i*) ! *j*)
apply(*subgoal-tac bits* (*levels* (*mem-pool-info Va* (*pool b*))
[*level b* := (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)
⦇*bits* := *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)[*block b*

:= *FREEING*]⦈] ! (*i* − *Suc NULL*)) ! (*j div 4*)
= *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! (*i* − *Suc NULL*)) ! (*j div 4*))
apply (*metis Mem-pool-lvl.cases Mem-pool-lvl.simps(1) Mem-pool-lvl.simps(4)*

length-list-update nth-list-update-eq nth-list-update-neq)
apply (*metis Mem-pool-lvl.cases Mem-pool-lvl.simps(1) Mem-pool-lvl.simps(4)*

One-nat-def nth-list-update-eq nth-list-update-neq)
apply (*metis* (*no-types*, *lifting*) *Mem-pool-lvl.cases Mem-pool-lvl.simps(1)*
Mem-pool-lvl.simps(4) nth-list-update-eq nth-list-update-neq)
apply(*rule impI*)
apply(*rule conjI*)
apply(*case-tac i* = *level b* ∧ *j* = *block b*) **apply** *auto*[1]
apply(*case-tac Suc i* = *level b* ∧ *j* ∗ *4* = *block b*)
apply (*metis BlockState.distinct(5) less-Suc-eq nth-list-update-neq order-less-irrefl*)
apply(*subgoal-tac bits* (*levels* (*mem-pool-info Va* (*pool b*))
[*level b* := (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)
⦇*bits* := *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)[*block b*
:= *FREEING*]⦈] ! *i*) ! *j*
= *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *i*) ! *j*)
apply(*subgoal-tac bits* (*levels* (*mem-pool-info Va* (*pool b*))
[*level b* := (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)
⦇*bits* := *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)[*block b*
:= *FREEING*]⦈] ! (*Suc i*)) ! (*j* ∗ *4*)
= *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! (*Suc i*)) ! (*j* ∗ *4*))
apply (*metis Mem-pool-lvl.cases Mem-pool-lvl.simps(1) Mem-pool-lvl.simps(4)*

length-list-update nth-list-update-eq nth-list-update-neq)
apply (*metis Mem-pool-lvl.cases Mem-pool-lvl.simps(1) Mem-pool-lvl.simps(4)*

nth-list-update-eq nth-list-update-neq)
apply (*metis* (*no-types*, *lifting*) *Mem-pool-lvl.cases Mem-pool-lvl.simps(1)*
Mem-pool-lvl.simps(4) nth-list-update-eq nth-list-update-neq)
apply(*rule conjI*)
apply(*case-tac i* = *level b* ∧ *j* = *block b*) **apply** *auto*[1]
apply(*case-tac Suc i* = *level b* ∧ *Suc* (*j* ∗ *4*) = *block b*)
apply (*metis BlockState.distinct(5) less-Suc-eq nth-list-update-neq order-less-irrefl*)
apply(*subgoal-tac bits* (*levels* (*mem-pool-info Va* (*pool b*))
[*level b* := (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)
⦇*bits* := *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)[*block b*
:= *FREEING*]⦈] ! *i*) ! *j*
= *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *i*) ! *j*)
apply(*subgoal-tac bits* (*levels* (*mem-pool-info Va* (*pool b*))
[*level b* := (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)
⦇*bits* := *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)[*block b*
:= *FREEING*]⦈] ! (*Suc i*)) ! *Suc* (*j* ∗ *4*)
= *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! (*Suc i*)) ! *Suc* (*j* ∗ *4*))
apply (*metis Mem-pool-lvl.cases Mem-pool-lvl.simps(1) Mem-pool-lvl.simps(4)*

length-list-update nth-list-update-eq nth-list-update-neq)
apply (*metis Mem-pool-lvl.cases Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*)

nth-list-update-eq nth-list-update-neq)
apply (*metis* (*no-types*, *lifting*) *Mem-pool-lvl.cases Mem-pool-lvl.simps*(*1*)
Mem-pool-lvl.simps(*4*) *nth-list-update-eq nth-list-update-neq*)
apply(*rule conjI*)
apply(*case-tac i* = *level b* ∧ *j* = *block b*) **apply** *auto*[*1*]
apply(*case-tac Suc i* = *level b* ∧ *Suc* (*Suc* (*j* ∗ *4*)) = *block b*)
apply (*metis BlockState.distinct*(*5*) *less-Suc-eq nth-list-update-neq order-less-irrefl*)
apply(*subgoal-tac bits* (*levels* (*mem-pool-info Va* (*pool b*))
[*level b* := (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)
(|*bits* := *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)[*block b*
:= *FREEING*]|)] ! *i*) ! *j*
= *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *i*) ! *j*)
apply(*subgoal-tac bits* (*levels* (*mem-pool-info Va* (*pool b*))
[*level b* := (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)
(|*bits* := *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)[*block b*
:= *FREEING*]|)] ! (*Suc i*)) ! *Suc* (*Suc* (*j* ∗ *4*))
= *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! (*Suc i*)) ! *Suc* (*Suc* (*j* ∗
4)))
apply (*metis Mem-pool-lvl.cases Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*)

length-list-update nth-list-update-eq nth-list-update-neq)
apply (*metis Mem-pool-lvl.cases Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*)

nth-list-update-eq nth-list-update-neq)
apply (*metis* (*no-types*, *lifting*) *Mem-pool-lvl.cases Mem-pool-lvl.simps*(*1*)
Mem-pool-lvl.simps(*4*) *nth-list-update-eq nth-list-update-neq*)

apply(*case-tac i* = *level b* ∧ *j* = *block b*) **apply** *auto*[*1*]
apply(*case-tac Suc i* = *level b* ∧ (*j* ∗ *4* + *3*) = *block b*)
apply (*metis BlockState.distinct*(*5*) *less-Suc-eq nth-list-update-neq order-less-irrefl*)
apply(*subgoal-tac bits* (*levels* (*mem-pool-info Va* (*pool b*))
[*level b* := (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)
(|*bits* := *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)[*block b*
:= *FREEING*]|)] ! *i*) ! *j*
= *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *i*) ! *j*)
apply(*subgoal-tac bits* (*levels* (*mem-pool-info Va* (*pool b*))
[*level b* := (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)
(|*bits* := *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)[*block b*
:= *FREEING*]|)] ! (*Suc i*)) ! (*j* ∗ *4* + *3*)
= *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! (*Suc i*)) ! (*j* ∗ *4* + *3*))
apply (*metis Mem-pool-lvl.cases Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*)

length-list-update nth-list-update-eq nth-list-update-neq)
apply (*metis Mem-pool-lvl.cases Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*)

nth-list-update-eq nth-list-update-neq)
apply (*metis* (*no-types*, *lifting*) *Mem-pool-lvl.cases Mem-pool-lvl.simps*(*1*)
Mem-pool-lvl.simps(*4*) *nth-list-update-eq nth-list-update-neq*)

apply(*rule conjI*)
apply(*rule impI*)+
apply(*case-tac i = level b ∧ j = block b*) **apply** *auto*[*1*]
apply(*case-tac i − 1 = level b ∧ j div 4 = block b*)
apply (*metis* (*no-types*, *lifting*) *BlockState.distinct*(*3*) *One-nat-def Suc-pred lessI nat-neq-iff nth-list-update-neq*)
apply(*subgoal-tac bits* (*levels* (*mem-pool-info Va* (*pool b*))
[*level b* := (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)
(|*bits* := *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)[*block b* := *FREEING*]|)] ! *i*) ! *j*
= *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *i*) ! *j*)
apply(*subgoal-tac bits* (*levels* (*mem-pool-info Va* (*pool b*))
[*level b* := (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)
(|*bits* := *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)[*block b* := *FREEING*]|)] ! (*i* − *Suc NULL*)) ! (*j div 4*)
= *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! (*i* − *Suc NULL*)) ! (*j div 4*))
apply (*metis Mem-pool-lvl.cases Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*)

length-list-update nth-list-update-eq nth-list-update-neq)
apply (*metis Mem-pool-lvl.cases Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*)

One-nat-def nth-list-update-eq nth-list-update-neq)
apply (*metis* (*no-types*, *lifting*) *Mem-pool-lvl.cases Mem-pool-lvl.simps*(*1*)
Mem-pool-lvl.simps(*4*) *nth-list-update-eq nth-list-update-neq*)

apply(*rule conjI*)
apply(*rule impI*)+
apply(*rule conjI*)
apply(*case-tac i = level b ∧ j = block b*) **apply** *auto*[*1*]
apply(*case-tac Suc i = level b ∧ j ∗ 4 = block b*)
apply (*metis BlockState.distinct*(*5*) *less-Suc-eq nth-list-update-neq order-less-irrefl*)
apply(*subgoal-tac bits* (*levels* (*mem-pool-info Va* (*pool b*))
[*level b* := (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)
(|*bits* := *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)[*block b* := *FREEING*]|)] ! *i*) ! *j*
= *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *i*) ! *j*)
apply(*subgoal-tac bits* (*levels* (*mem-pool-info Va* (*pool b*))
[*level b* := (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)
(|*bits* := *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)[*block b* := *FREEING*]|)] ! (*Suc i*)) ! (*j ∗ 4*)
= *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! (*Suc i*)) ! (*j ∗ 4*))
apply (*metis Mem-pool-lvl.cases Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*)

length-list-update nth-list-update-eq nth-list-update-neq)

apply (*metis Mem-pool-lvl.cases Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*)

nth-list-update-eq nth-list-update-neq)
apply (*metis* (*no-types*, *lifting*) *Mem-pool-lvl.cases Mem-pool-lvl.simps*(*1*)
Mem-pool-lvl.simps(*4*) *nth-list-update-eq nth-list-update-neq*)
apply(*rule conjI*)
apply(*case-tac* $i = level\ b \wedge j = block\ b$) **apply** *auto*[*1*]
apply(*case-tac* $Suc\ i = level\ b \wedge Suc\ (j * 4) = block\ b$)
apply (*metis BlockState.distinct*(*5*) *less-Suc-eq nth-list-update-neq order-less-irrefl*)
apply(*subgoal-tac bits* (*levels* (*mem-pool-info Va* (*pool b*))
[*level b* := (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)
(|*bits* := *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)[*block b* :=
FREEING]|)] ! *i*) ! *j*
= *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *i*) ! *j*)
apply(*subgoal-tac bits* (*levels* (*mem-pool-info Va* (*pool b*))
[*level b* := (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)
(|*bits* := *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)[*block b* :=
FREEING]|)] ! (*Suc i*)) ! *Suc* (*j* ∗ *4*)
= *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! (*Suc i*)) ! *Suc* (*j* ∗ *4*))
apply (*metis Mem-pool-lvl.cases Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*)

length-list-update nth-list-update-eq nth-list-update-neq)
apply (*metis Mem-pool-lvl.cases Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*)

nth-list-update-eq nth-list-update-neq)
apply (*metis* (*no-types*, *lifting*) *Mem-pool-lvl.cases Mem-pool-lvl.simps*(*1*)
Mem-pool-lvl.simps(*4*) *nth-list-update-eq nth-list-update-neq*)
apply(*rule conjI*)
apply(*case-tac* $i = level\ b \wedge j = block\ b$) **apply** *auto*[*1*]
apply(*case-tac* $Suc\ i = level\ b \wedge Suc\ (Suc\ (j * 4)) = block\ b$)
apply (*metis BlockState.distinct*(*5*) *less-Suc-eq nth-list-update-neq order-less-irrefl*)
apply(*subgoal-tac bits* (*levels* (*mem-pool-info Va* (*pool b*))
[*level b* := (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)
(|*bits* := *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)[*block b* :=
FREEING]|)] ! *i*) ! *j*
= *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *i*) ! *j*)
apply(*subgoal-tac bits* (*levels* (*mem-pool-info Va* (*pool b*))
[*level b* := (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)
(|*bits* := *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)[*block b* :=
FREEING]|)] ! (*Suc i*)) ! *Suc* (*Suc* (*j* ∗ *4*))
= *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! (*Suc i*)) ! *Suc* (*Suc* (*j* ∗
4)))
apply (*metis Mem-pool-lvl.cases Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*)

length-list-update nth-list-update-eq nth-list-update-neq)
apply (*metis Mem-pool-lvl.cases Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*)

nth-list-update-eq nth-list-update-neq)
apply (*metis* (*no-types*, *lifting*) *Mem-pool-lvl.cases Mem-pool-lvl.simps*(*1*)

Mem-pool-lvl.simps(*4*) *nth-list-update-eq nth-list-update-neq*)

apply(*case-tac i* = *level b* ∧ *j* = *block b*) **apply** *auto*[*1*]
apply(*case-tac Suc i* = *level b* ∧ (*j* ∗ *4* + *3*) = *block b*)
apply (*metis BlockState.distinct*(*5*) *less-Suc-eq nth-list-update-neq order-less-irrefl*)
apply(*subgoal-tac bits* (*levels* (*mem-pool-info Va* (*pool b*))
[*level b* := (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)
(|*bits* := *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)[*block b* :=
FREEING]|)] ! *i*) ! *j*
= *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *i*) ! *j*)
apply(*subgoal-tac bits* (*levels* (*mem-pool-info Va* (*pool b*))
[*level b* := (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)
(|*bits* := *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)[*block b* :=
FREEING]|)] ! (*Suc i*)) ! (*j* ∗ *4* + *3*)
= *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! (*Suc i*)) ! (*j* ∗ *4* + *3*))
apply (*metis Mem-pool-lvl.cases Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*)

length-list-update nth-list-update-eq nth-list-update-neq)
apply (*metis Mem-pool-lvl.cases Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*)

nth-list-update-eq nth-list-update-neq)
apply (*metis* (*no-types*, *lifting*) *Mem-pool-lvl.cases Mem-pool-lvl.simps*(*1*)
Mem-pool-lvl.simps(*4*) *nth-list-update-eq nth-list-update-neq*)

apply(*rule impI*)+
apply(*case-tac i* = *level b* ∧ *j* = *block b*) **apply** *auto*[*1*]
apply(*case-tac i* − *1* = *level b* ∧ *j div 4* = *block b*) **apply** *auto*[*1*]
apply(*subgoal-tac bits* (*levels* (*mem-pool-info Va* (*pool b*))
[*level b* := (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)
(|*bits* := *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)[*block b* :=
FREEING]|)] ! *i*) ! *j*
= *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *i*) ! *j*)
prefer *2* **apply** (*metis* (*no-types*, *lifting*) *Mem-pool-lvl.cases Mem-pool-lvl.simps*(*1*)

Mem-pool-lvl.simps(*4*) *nth-list-update-eq nth-list-update-neq*)
apply(*subgoal-tac bits* (*levels* (*mem-pool-info Va* (*pool b*))
[*level b* := (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)
(|*bits* := *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)[*block b* :=
FREEING]|)] ! (*i* − *1*)) ! (*j div 4*)
= *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! (*i* − *1*)) ! (*j div 4*))
prefer *2* **apply** (*metis Mem-pool-lvl.cases Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*)

nth-list-update-eq nth-list-update-neq)
apply(*subgoal-tac bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! (*i* − *Suc NULL*))
! (*j div 4*) ≠ *DIVIDED*)
prefer *2* **apply**(*subgoal-tac length* (*bits* (*levels* (*mem-pool-info Va* (*pool b*))
[*level b* := (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)
(|*bits* := *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *level*
b)[*block b* := *FREEING*]|)] !

i)) = $length$ ($bits$ ($levels$ (mem-$pool$-$info$ Va ($pool$ b)) ! i)))
prefer 2 **apply**($case$-tac i = $level$ b)
apply $auto[1]$ **apply** $auto[1]$
apply $simp$

apply $simp$
done

lemma $mempool$-$free$-$stm1$-inv-$auxvars$:
inv-cur Va $\wedge$ inv-thd-$waitq$ Va $\wedge$ inv-$mempool$-$info$ Va $\wedge$ inv-$bitmap$-$freelist$ Va $\wedge$ inv-$bitmap$ Va $\wedge$ inv-aux-$vars$ Va $\Longrightarrow$
$block$ b < $length$ ($bits$ ($levels$ (mem-$pool$-$info$ Va ($pool$ b)) ! $level$ b)) $\Longrightarrow$
$level$ b < $length$ ($levels$ (mem-$pool$-$info$ Va ($pool$ b))) $\Longrightarrow$
$pool$ b $\in$ mem-$pools$ Va $\Longrightarrow$
$data$ b = $block$-ptr (mem-$pool$-$info$ Va ($pool$ b)) ($ALIGN4$ (max-sz (mem-$pool$-$info$ Va ($pool$ b))) div 4 ^ $level$ b) ($block$ b) $\Longrightarrow$
get-bit (mem-$pool$-$info$ Va) ($pool$ b) ($level$ b) ($block$ b) = $ALLOCATED$ $\Longrightarrow$
$allocating$-$node$ Va t = $None$ $\Longrightarrow$
$freeing$-$node$ Va t = $None$ $\Longrightarrow$
inv-aux-$vars$
(Va(|mem-$pool$-$info$:= (mem-$pool$-$info$ Va)
($pool$ b := mem-$pool$-$info$ Va ($pool$ b)
(|$levels$:= $levels$ (mem-$pool$-$info$ Va ($pool$ b))
[$level$ b := ($levels$ (mem-$pool$-$info$ Va ($pool$ b)) ! $level$ b)
(|$bits$:= $bits$ ($levels$ (mem-$pool$-$info$ Va ($pool$ b)) ! $level$ b)[$block$ b := $FREEING$]|)]|)),
$freeing$-$node$:= $freeing$-$node$ Va(t $\mapsto$ b)|))
apply($unfold$ inv-aux-$vars$-def)
apply($rule$ $conjI$)
apply $clarify$
apply($case$-tac ta = t) **apply** $auto[1]$
apply($subgoal$-tac $\neg$($pool$ n = $pool$ b $\wedge$ $level$ n = $level$ b $\wedge$ $block$ n = $block$ b))
apply($subgoal$-tac $freeing$-$node$
(Va(|mem-$pool$-$info$:= (mem-$pool$-$info$ Va)
($pool$ b := mem-$pool$-$info$ Va ($pool$ b)
(|$levels$:= $levels$ (mem-$pool$-$info$ Va ($pool$ b))
[$level$ b := ($levels$ (mem-$pool$-$info$ Va ($pool$ b)) ! $level$ b)
(|$bits$:= $bits$ ($levels$ (mem-$pool$-$info$ Va ($pool$ b)) ! $level$ b)[$block$ b := $FREEING$]|)]|)),
$freeing$-$node$:= $freeing$-$node$ Va(t $\mapsto$ b)|)) ta = $freeing$-$node$ Va ta)
apply($subgoal$-tac get-bit (mem-$pool$-$info$ Va) ($pool$ n) ($level$ n) ($block$ n) = $FREEING$)
apply($subgoal$-tac get-bit (mem-$pool$-$info$ Va) ($pool$ n) ($level$ n) ($block$ n) =
get-bit
(mem-$pool$-$info$
(Va(|mem-$pool$-$info$:= (mem-$pool$-$info$ Va)
($pool$ b := mem-$pool$-$info$ Va ($pool$ b)
(|$levels$:= $levels$ (mem-$pool$-$info$ Va ($pool$ b))
[$level$ b := ($levels$ (mem-$pool$-$info$ Va ($pool$ b)) ! $level$ b)

$\llparenthese$*bits* := *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)[*block b* := *FREEING*]$\rrparenthese$]$\rrparenthese$),
freeing-node := *freeing-node Va*(*t* $\mapsto$ *b*)$\rrparenthese$))
(*pool n*) (*level n*) (*block n*))
apply *auto*[*1*]
apply(*case-tac* ¬ *pool n* = *pool b*) **apply** *simp*
apply(*case-tac* ¬ *level n* = *level b*) **apply** *simp*
apply(*case-tac* ¬ *block n* = *block b*) **apply** *simp* **apply** *simp*
apply *auto*[*1*] **apply** *auto*[*1*]

apply(*subgoal-tac freeing-node Va ta* = *Some n*) **prefer** *2* **apply** *auto*[*1*]
apply *fastforce*

apply(*rule conjI*)
apply *clarify*
apply(*case-tac* ¬(*pool n* = *pool b* ∧ *level n* = *level b* ∧ *block n* = *block b*))
apply(*subgoal-tac get-bit* (*mem-pool-info Va*) (*pool n*) (*level n*) (*block n*) = *FREEING*)
prefer *2* **apply** *auto*[*1*]
using *set-bit-def set-bit-get-bit-neq* **apply** *auto*[*1*]
using *set-bit-def set-bit-get-bit-neq* **apply** *auto*[*1*]
apply(*subgoal-tac mem-block-addr-valid Va n*)
prefer *2* **using** *mem-block-addr-valid-def* **apply** *auto*[*1*]
apply(*subgoal-tac* ∃ *t'*. *t'* ≠ *t* ∧ *freeing-node Va t'* = *Some n*)
prefer *2* **apply** (*metis option.discI*)
apply *auto*[*1*]

apply(*subgoal-tac data b* = *data n*)
prefer *2* **apply**(*simp add:block-ptr-def mem-block-addr-valid-def inv-mempool-info-maxsz-align4*)
apply *auto*[*1*]

apply(*rule conjI*)
apply *clarify*
apply(*case-tac ta* = *t*) **apply** *auto*[*1*]
apply(*subgoal-tac allocating-node Va ta* = *Some n*)
prefer *2* **apply** *auto*[*1*]
apply(*subgoal-tac get-bit-s Va* (*pool n*) (*level n*) (*block n*) = *ALLOCATING*)
prefer *2* **apply** *auto*[*1*]
apply(*case-tac* ¬(*pool n* = *pool b* ∧ *level n* = *level b* ∧ *block n* = *block b*))
apply(*case-tac* ¬ *pool n* = *pool b*) **apply** *simp*
apply(*case-tac* ¬ *level n* = *level b*) **apply** *force*
apply(*case-tac* ¬ *block n* = *block b*) **apply** *force* **apply** *simp*
apply *fastforce*

apply(*rule conjI*)
apply *clarify*
apply(*case-tac* ¬(*pool n* = *pool b* ∧ *level n* = *level b* ∧ *block n* = *block b*))
apply(*subgoal-tac get-bit* (*mem-pool-info Va*) (*pool n*) (*level n*) (*block n*) = *ALLOCATING*)

```
      prefer 2 apply auto[1]
        using set-bit-def set-bit-get-bit-neq apply auto[1]
        using set-bit-def set-bit-get-bit-neq apply auto[1]
    apply(subgoal-tac mem-block-addr-valid Va n)
      prefer 2 using mem-block-addr-valid-def apply auto[1]
    apply(subgoal-tac ∃ t'. t' ≠ t ∧ allocating-node Va t' = Some n)
      prefer 2 apply (metis option.discI)
    apply auto[1]

    apply(subgoal-tac data b = data n)
    prefer 2 apply(simp add:block-ptr-def mem-block-addr-valid-def inv-mempool-info-maxsz-align4)
      apply auto[1]

apply(rule conjI)
  apply clarify
  apply(case-tac t1 ≠ t ∧ t2 ≠ t)
    apply auto[1]
    apply(case-tac t1 = t)
      apply clarify
      apply(subgoal-tac freeing-node Va t2 = Some n2)
        prefer 2 apply auto[1]
      apply(subgoal-tac b = n1)
        prefer 2 apply auto[1]
      apply simp

    apply(case-tac t2 = t)
      apply clarify
      apply(subgoal-tac freeing-node Va t1 = Some n1)
        prefer 2 apply auto[1]
      apply(subgoal-tac b = n2)
        prefer 2 apply auto[1]
      apply fastforce

    apply simp

apply(rule conjI)
  apply clarify
  apply(case-tac t1 ≠ t ∧ t2 ≠ t)
    apply auto[1]
    apply(case-tac t1 = t)
      apply clarify
      apply(subgoal-tac freeing-node Va t2 = Some n2)
        prefer 2 apply auto[1]
      apply(subgoal-tac b = n1)
        prefer 2 apply auto[1]
      apply simp

    apply(case-tac t2 = t)
      apply clarify
```

apply(*subgoal-tac freeing-node Va t1 = Some n1*)
prefer *2* **apply** *auto*[*1*]
apply(*subgoal-tac b = n2*)
prefer *2* **apply** *auto*[*1*]
apply *fastforce*

apply *simp*

apply *clarify*
apply(*case-tac t1* ≠ *t* ∧ *t2* ≠ *t*)
apply *auto*[*1*]
apply(*case-tac t1* = *t*)
apply *clarify*
apply(*subgoal-tac freeing-node Va t2 = Some n2*)
prefer *2* **apply** *auto*[*1*]
apply(*subgoal-tac b = n1*)
prefer *2* **apply** *auto*[*1*]
apply *simp*

apply(*case-tac t2* = *t*)
apply *clarify*
apply(*subgoal-tac allocating-node Va t1 = Some n1*)
prefer *2* **apply** *auto*[*1*]
apply(*subgoal-tac b = n2*)
prefer *2* **apply** *auto*[*1*]
apply *fastforce*

apply *simp*
done

lemma *mempool-free-stm1-inv-lvl0*:
inv-cur Va ∧ *inv-thd-waitq Va* ∧ *inv-mempool-info Va* ∧ *inv-bitmap-freelist Va*
∧ *inv-bitmap Va* ∧ *inv-aux-vars Va* ∧ *inv-bitmap0 Va* ⟹
block b < *length* (*bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)) ⟹
level b < *length* (*levels* (*mem-pool-info Va* (*pool b*))) ⟹
pool b ∈ *mem-pools Va* ⟹
data b = *block-ptr* (*mem-pool-info Va* (*pool b*)) (*ALIGN4* (*max-sz* (*mem-pool-info Va* (*pool b*))) *div 4* ˆ *level b*) (*block b*) ⟹
get-bit (*mem-pool-info Va*) (*pool b*) (*level b*) (*block b*) = *ALLOCATED* ⟹
allocating-node Va t = *None* ⟹
freeing-node Va t = *None* ⟹
inv-bitmap0
(*Va*(|*mem-pool-info* := (*mem-pool-info Va*)
(*pool b* := *mem-pool-info Va* (*pool b*)
(|*levels* := *levels* (*mem-pool-info Va* (*pool b*))
[*level b* := (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)
(|*bits* := *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)[*block b* := *FREEING*]|)]|)|)),

freeing-node := *freeing-node Va*($t \mapsto b$)⦈)
apply(*simp add*: *inv-bitmap0-def Let-def*)
apply *clarsimp*
apply(*case-tac level b* = *0*)
apply(*case-tac block b* = *i*) **apply** *auto*[*1*] **apply** *simp*
apply *simp*
done

lemma *mempool-free-stm1-inv-lvln*:
inv-cur Va ∧ *inv-thd-waitq Va* ∧ *inv-mempool-info Va* ∧ *inv-bitmap-freelist Va*
∧ *inv-bitmap Va* ∧ *inv-aux-vars Va* ∧ *inv-bitmapn Va* ⟹
block b < *length* (*bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)) ⟹
level b < *length* (*levels* (*mem-pool-info Va* (*pool b*))) ⟹
pool b ∈ *mem-pools Va* ⟹
data b = *block-ptr* (*mem-pool-info Va* (*pool b*)) (*ALIGN4* (*max-sz* (*mem-pool-info Va* (*pool b*))) *div 4* ^ *level b*) (*block b*) ⟹
get-bit (*mem-pool-info Va*) (*pool b*) (*level b*) (*block b*) = *ALLOCATED* ⟹
allocating-node Va t = *None* ⟹
freeing-node Va t = *None* ⟹
inv-bitmapn
(*Va*⦇*mem-pool-info* := (*mem-pool-info Va*)
(*pool b* := *mem-pool-info Va* (*pool b*)
⦇*levels* := *levels* (*mem-pool-info Va* (*pool b*))
[*level b* := (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)
⦇*bits* := *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)[*block b* := *FREEING*]⦈]⦈),
freeing-node := *freeing-node Va*($t \mapsto b$)⦈)
apply(*simp add*: *inv-bitmapn-def Let-def*)
apply *clarsimp*
apply(*case-tac level b* = *length* (*levels* (*mem-pool-info Va* (*pool b*))) − *1*)
apply(*case-tac block b* = *i*) **apply** *auto*[*1*] **apply** *simp*
apply *simp*
done

lemma *mempool-free-stm1-inv-lvls-not4free*:
inv-cur Va ∧ *inv-thd-waitq Va* ∧ *inv-mempool-info Va* ∧ *inv-bitmap-freelist Va*
∧ *inv-bitmap Va* ∧ *inv-aux-vars Va* ∧ *inv-bitmap-not4free Va* ⟹
block b < *length* (*bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)) ⟹
level b < *length* (*levels* (*mem-pool-info Va* (*pool b*))) ⟹
pool b ∈ *mem-pools Va* ⟹
data b = *block-ptr* (*mem-pool-info Va* (*pool b*)) (*ALIGN4* (*max-sz* (*mem-pool-info Va* (*pool b*))) *div 4* ^ *level b*) (*block b*) ⟹
get-bit (*mem-pool-info Va*) (*pool b*) (*level b*) (*block b*) = *ALLOCATED* ⟹
allocating-node Va t = *None* ⟹
freeing-node Va t = *None* ⟹
inv-bitmap-not4free
(*Va*⦇*mem-pool-info* := (*mem-pool-info Va*)
(*pool b* := *mem-pool-info Va* (*pool b*)
⦇*levels* := *levels* (*mem-pool-info Va* (*pool b*))

[*level b* := (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)
(|*bits* := *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)[*block b*
:= *FREEING*]|)]|)),
freeing-node := *freeing-node Va*($t \mapsto b$)|))
apply(*simp add*: *inv-bitmap-not4free-def Let-def partner-bits-def*)
apply *clarsimp*
apply(*case-tac level b* = *i*) **prefer** *2* **apply** *auto*[*1*]
apply(*case-tac block b* = *j div 4* ∗ *4*) **apply** *auto*[*1*]
apply(*case-tac block b* = *j div 4* ∗ *4* + *1*) **apply** *auto*[*1*]
apply(*case-tac block b* = *j div 4* ∗ *4* + *2*) **apply** *auto*[*1*]
apply(*case-tac block b* = *j div 4* ∗ *4* + *3*) **apply** *auto*[*1*]

apply *simp*
done

lemma *mempool-free-smt1-inv*:
inv Va ⟹
block b < *length* (*bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)) ⟹
level b < *length* (*levels* (*mem-pool-info Va* (*pool b*))) ⟹
pool b ∈ *mem-pools Va* ⟹
data b = *block-ptr* (*mem-pool-info Va* (*pool b*)) (*ALIGN4* (*max-sz* (*mem-pool-info Va* (*pool b*))) *div 4* ^ *level b*) (*block b*) ⟹
get-bit (*mem-pool-info Va*) (*pool b*) (*level b*) (*block b*) = *ALLOCATED* ⟹
allocating-node Va t = *None* ⟹
freeing-node Va t = *None* ⟹
inv (*Va*(|*mem-pool-info* := (*mem-pool-info Va*)
(*pool b* := *mem-pool-info Va* (*pool b*)
(|*levels* := *levels* (*mem-pool-info Va* (*pool b*))
[*level b* := (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)
(|*bits* := *bits* (*levels* (*mem-pool-info Va* (*pool b*)) ! *level b*)[*block b*
:= *FREEING*]|)]|)),
freeing-node := *freeing-node Va*($t \mapsto b$)|))
apply(*simp add*:*inv-def*)
apply(*rule conjI*) **apply**(*simp add*:*inv-cur-def Mem-pool-free-guar-def*)
apply(*rule conjI*) **apply**(*simp add*:*inv-thd-waitq-def*)
apply(*rule conjI*) **apply** *clarify* **apply** *metis*
apply *clarify* **apply** *metis*
apply(*rule conjI*) **using** *mempool-free-stm1-inv-mempool-info* **apply** *auto*[*1*]
apply(*rule conjI*) **using** *mempool-free-stm1-inv-bitmap-freelist* **apply** *auto*[*1*]
apply(*rule conjI*) **using** *mempool-free-stm1-inv-bitmap* **apply** *auto*[*1*]
apply(*rule conjI*) **using** *mempool-free-stm1-inv-auxvars* **apply** *auto*[*1*]
apply(*rule conjI*) **using** *mempool-free-stm1-inv-lvl0* **apply** *auto*[*1*]
apply(*rule conjI*) **using** *mempool-free-stm1-inv-lvln* **apply** *auto*[*1*]
using *mempool-free-stm1-inv-lvls-not4free* **apply** *auto*[*1*]
done

lemma *mempool-free-stm1-h1*:
Mem-pool-free-pre t ∩

$\{\!|pool\ b \in\ ´mem\text{-}pools\ \wedge$
$level\ b < length\ (levels\ (´mem\text{-}pool\text{-}info\ (pool\ b)))\ \wedge$
$block\ b < length\ (bits\ (levels\ (´mem\text{-}pool\text{-}info\ (pool\ b))\ !\ level\ b))\ \wedge$
$data\ b =$
$block\text{-}ptr\ (´mem\text{-}pool\text{-}info\ (pool\ b))\ (ALIGN4\ (max\text{-}sz\ (´mem\text{-}pool\text{-}info\ (pool\ b)))\ div\ 4\ \hat{}\ level\ b)\ (block\ b)|\!\}\ \cap$
$\{\!|´cur = Some\ t|\!\}\ \cap$
$\{Va\}\ \cap$
$\{\!|´get\text{-}bit\text{-}s\ (pool\ b)\ (level\ b)\ (block\ b) = ALLOCATED|\!\}\ \cap$
$\{Va\} \neq$
$\{\} \Longrightarrow$
$\Gamma \vdash_I Some\ (´mem\text{-}pool\text{-}info := set\text{-}bit\text{-}freeing\ ´mem\text{-}pool\text{-}info\ (pool\ b)\ (level\ b)\ (block\ b);;$
$´freeing\text{-}node := ´freeing\text{-}node(t \mapsto$
$b))\ sat_p\ [Mem\text{-}pool\text{-}free\text{-}pre\ t\ \cap$
$\{\!|pool\ b \in\ ´mem\text{-}pools\ \wedge$
$level\ b < length\ (levels\ (´mem\text{-}pool\text{-}info\ (pool\ b)))\ \wedge$
$block\ b < length\ (bits\ (levels\ (´mem\text{-}pool\text{-}info\ (pool\ b))\ !\ level\ b))\ \wedge$
$data\ b =$
$block\text{-}ptr\ (´mem\text{-}pool\text{-}info\ (pool\ b))\ (ALIGN4\ (max\text{-}sz\ (´mem\text{-}pool\text{-}info\ (pool\ b)))\ div\ 4\ \hat{}\ level\ b)$
$(block\ b)|\!\}\ \cap$
$\{\!|´cur = Some\ t|\!\}\ \cap$
$\{Va\}\ \cap$
$\{\!|´get\text{-}bit\text{-}s\ (pool\ b)\ (level\ b)\ (block\ b) = ALLOCATED|\!\}\ \cap$
$\{Va\},\ \{(x,\ y).$
$x = y\},\ UNIV,\ \{\!|´(Pair\ Va) \in Mem\text{-}pool\text{-}free\text{-}guar\ t|\!\}\ \cap$
$(\{\!|´invariant.inv\ \wedge\ ´allocating\text{-}node\ t = None|\!\}\ \cap$
$\{\!|pool\ b \in\ ´mem\text{-}pools\ \wedge$
$level\ b < length\ (levels\ (´mem\text{-}pool\text{-}info\ (pool\ b)))\ \wedge$
$block\ b < length\ (bits\ (levels\ (´mem\text{-}pool\text{-}info\ (pool\ b))\ !\ level\ b))\ \wedge$
$data\ b =$
$block\text{-}ptr\ (´mem\text{-}pool\text{-}info\ (pool\ b))$
$(ALIGN4\ (max\text{-}sz\ (´mem\text{-}pool\text{-}info\ (pool\ b)))\ div\ 4\ \hat{}\ level\ b)\ (block\ b)|\!\}\ \cap$
$mp\text{-}free\text{-}precond2\text{-}ext\ t\ b)]$

apply *clarsimp*
apply(*rule Seq*[**where** *mid*=$\{Va(|mem\text{-}pool\text{-}info := set\text{-}bit\text{-}freeing\ (mem\text{-}pool\text{-}info\ Va)\ (pool\ b)\ (level\ b)\ (block\ b)|)\}$])
apply(*rule Basic*)
apply *auto*[*1*] **apply**(*simp add*:*stable-def*)+
apply(*rule Basic*)
apply(*simp add*: *set-bit-def*)
apply(*rule conjI*)
apply(*simp add*:*Mem-pool-free-guar-def*)
apply(*rule disjI1*)

apply(*rule conjI*)
apply(*simp add*:*gvars-conf-stable-def gvars-conf-def*) **apply** *auto*[*1*]
apply(*case-tac i* = *level b*) **apply** *auto*[*1*] **apply** *auto*[*1*]

apply(*rule conjI*)
using *mempool-free-smt1-inv* **apply** *auto*[*1*]
apply(*simp add*:*lvars-nochange-def*)

apply(*rule conjI*)
using *mempool-free-smt1-inv* **apply** *auto*[*1*]
apply(*simp add*:*block-ptr-def*)

apply(*simp add*:*stable-def*)+
done

lemma *mempool-free-stm1*:
$\Gamma \vdash_I$ *Some* (*t* ▶ *AWAIT bits* (*levels* (´*mem-pool-info* (*pool b*)) ! *level b*) ! *block b* = *ALLOCATED THEN*
´*mem-pool-info* := *set-bit-freeing* ´*mem-pool-info* (*pool b*) (*level b*) (*block b*);;
´*freeing-node* := ´*freeing-node* (*t* := *Some b*)
END) sat_p
[*mp-free-precond1 t b*, *Mem-pool-free-rely t*, *Mem-pool-free-guar t*, *mp-free-precond2 t b*]
apply(*simp add*:*stm-def*)
apply(*rule Await*)

using *mp-free-precond1-stb* **apply** *auto*[*1*]
using *mp-free-precond2-stb* **apply** *auto*[*1*]

apply(*rule allI*)
apply(*rule Await*)
apply(*simp add*:*stable-def*) **apply**(*auto simp add*:*stable-def*)
apply(*case-tac V* ≠ *Va*) **apply** *auto*[*1*] **using** *Emptyprecond* **apply** *blast*
apply *clarsimp*
apply(*case-tac mp-free-precond1 t b* ∩ ⦃´*cur* = *Some t*⦄ ∩ {*Va*} ∩
⦃*get-bit* ´*mem-pool-info* (*pool b*) (*level b*) (*block b*) = *ALLOCATED*⦄ ∩
{*Va*} = {})
apply *simp* **using** *Emptyprecond* **apply** *auto*[*1*]
using *mempool-free-stm1-h1* **apply** *force*
done

lemma *mempool-free-stm2*:
$\Gamma \vdash_I$ *Some* (*t* ▶ ´*need-resched* := ´*need-resched*(*t* := *False*)) sat_p
[*mp-free-precond2 t b*, *Mem-pool-free-rely t*, *Mem-pool-free-guar t*, *mp-free-precond3 t b*]
apply(*simp add*:*stm-def*)

apply(*rule Await*)
using *mp-free-precond2-stb* **apply** *simp*
using *mp-free-precond3-stb* **apply** *simp*

apply *clarify*
apply(*rule Basic*)
apply(*case-tac mp-free-precond2 t b* $\cap$ $\{\!|$´*cur* = *Some t*$|\!\}$ $\cap$ $\{V\}$ = $\{\}$)
apply *auto*[*1*]
apply *clarsimp*
apply(*rule conjI*)
apply(*simp add*:*Guar*$_f$-*def gvars-conf-stable-def gvars-conf-def Mem-pool-free-guar-def*)
apply(*rule disjI1*)
apply(*rule conjI*)
apply(*subgoal-tac* (*V*, *V*$(\!|$*need-resched* := (*need-resched V*)(*t* := *False*)$|\!)$)$\in$*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def*
lvars-nochange1-def)

apply *clarify* **apply**(*simp add*: *lvars-nochange-def*)

apply(*subgoal-tac* (*V*, *V*$(\!|$*need-resched* := (*need-resched V*)(*t* := *False*)$|\!)$)$\in$*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def*
lvars-nochange1-def)

apply(*simp add*:*stable-def*)+

done

lemma *mempool-free-stm3*:
$\Gamma \vdash_I$ *Some* (*t* ▶ ´*lsizes* := ´*lsizes*(*t* := [*ALIGN4* (*max-sz* (´*mem-pool-info* (*pool*
b)))])) *sat*$_p$
[*mp-free-precond3 t b*, *Mem-pool-free-rely t*, *Mem-pool-free-guar t*, *mp-free-precond4*
t b]
apply(*simp add*:*stm-def*)
apply(*rule Await*)
using *mp-free-precond3-stb* **apply** *simp*
using *mp-free-precond4-stb* **apply** *simp*

apply *clarify*
apply(*rule Basic*)
apply(*case-tac mp-free-precond3 t b* $\cap$ $\{\!|$´*cur* = *Some t*$|\!\}$ $\cap$ $\{V\}$ = $\{\}$)
apply *auto*[*1*]
apply *clarsimp*
apply(*rule conjI*)
apply(*simp add*:*Guar*$_f$-*def gvars-conf-stable-def gvars-conf-def Mem-pool-free-guar-def*)
apply(*rule disjI1*)
apply(*rule conjI*)
apply(*subgoal-tac* (*V*, *V*$(\!|$*lsizes* := (*lsizes V*)(*t* := [*ALIGN4* (*max-sz* (*mem-pool-info*
V (*pool b*)))])$|\!)$)
$\in$*lvars-nochange1-4all*)

using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def*)

apply *clarify* **apply**(*simp add*: *lvars-nochange-def*)
apply(*subgoal-tac* (*V*,*V*⦇*lsizes* := (*lsizes V*)(*t* := [*ALIGN4* (*max-sz* (*mem-pool-info V* (*pool b*)))])⦈)
∈*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def*)
apply(*simp add*:*stable-def*)+

done

lemma *mempool-free-stm41-h1-1*: (*n*::*nat*) > *0* ∧ (*x*::*nat*) *mod y* = *0* ⟹ *n* ∗ *x mod y* = *0*
by *auto*

lemma *mempool-free-stm41-h1*:
assumes *p1*: *i V t* ≤ *level b*
and *p2*: *length* (*lsizes V t*) = *i V t*
and *p3*: *inv V*
and *p4*: ∀ *ii*<*i V t*. *lsizes V t* ! *ii* = *ALIGN4* (*max-sz* (*mem-pool-info V* (*pool b*))) *div 4* ˆ *ii*
and *p5*: *lsizes V t* ≠ []
and *p6*: *pool b* ∈ *mem-pools V*
and *p7*: *level b* < *length* (*levels* (*mem-pool-info V* (*pool b*)))
and *p8*: *block b* < *length* (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *level b*))
and *p9*: *ii* = *i V t*
shows (*lsizes V t* @ [*ALIGN4* (*lsizes V t* ! (*i V t* − *Suc NULL*) *div 4*)]) ! *ii*
= *ALIGN4* (*max-sz* (*mem-pool-info V* (*pool b*))) *div 4* ˆ *ii*
proof −
from *p2 p9* **have** *a0*: (*lsizes V t* @ [*ALIGN4* (*lsizes V t* ! (*i V t* − *1*) *div 4*)]) ! *ii*
= *ALIGN4* (*lsizes V t* ! (*i V t* − *1*) *div 4*)
by (*metis nth-append-length*)

from *p2 p4 p5* **have** *lsizes V t* ! (*i V t* − *1*) *div 4* = *ALIGN4* (*max-sz* (*mem-pool-info V* (*pool b*))) *div 4* ˆ (*i V t* − *1*) *div 4*
by (*metis One-nat-def diff-less-mono2 diff-zero length-greater-0-conv zero-less-Suc*)
hence *a1*: *lsizes V t* ! (*i V t* − *1*) *div 4* = *ALIGN4* (*max-sz* (*mem-pool-info V* (*pool b*))) *div 4* ˆ (*i V t*)
by (*metis One-nat-def Suc-pred div-mult2-eq length-greater-0-conv p2 p5 plus-1-eq-Suc power-add power-commutes power-one-right*)

from *p6 p3* **have** ∃ *n*>*0*. *max-sz* (*mem-pool-info V* (*pool b*))
= (*4* ∗ *n*) ∗ (*4* ˆ (*length* (*levels* (*mem-pool-info V* (*pool b*)))))
apply(*simp add*:*inv-def inv-mempool-info-def Let-def*) **by** *auto*
then obtain *n* **where** *n* > *0* ∧ *max-sz* (*mem-pool-info V* (*pool b*))
= (*4* ∗ *n*) ∗ (*4* ˆ (*length* (*levels* (*mem-pool-info V* (*pool b*)))))

by *auto*
hence *a2*: $n > 0 \wedge$ *max-sz* (*mem-pool-info V* (*pool b*))
$= n * (4$ ^ (*length* (*levels* (*mem-pool-info V* (*pool b*))) + *1*)) **by** *auto*
hence *max-sz* (*mem-pool-info V* (*pool b*)) *mod 4* = *0* **by** *simp*
hence *a3*: *ALIGN4* (*max-sz* (*mem-pool-info V* (*pool b*))) = *max-sz* (*mem-pool-info V* (*pool b*))
using *align40* **by** *auto*
with *a1* **have** *a4*: *lsizes V t* ! (*i V t* − *1*) *div 4* = *max-sz* (*mem-pool-info V* (*pool b*)) *div 4* ^ (*i V t*) **by** *simp*

from *p1 p2 p7 a2* **have** (*max-sz* (*mem-pool-info V* (*pool b*)) *div 4* ^ *i V t*) *mod 4* = *0*
apply(*subgoal-tac 4* ^ (*length* (*levels* (*mem-pool-info V* (*pool b*))) + *1*) *div 4* ^ *i V t mod 4* = *NULL*)
prefer *2* **using** *pow-lt-mod0*[*of 4 i V t length* (*levels* (*mem-pool-info V* (*pool b*)))+*1*] **apply** *auto*[*1*]
apply *simp* **using** *mempool-free-stm41-h1-1*
[*of n 4* ∗ *4* ^ *length* (*levels* (*mem-pool-info V* (*pool b*))) *div 4* ^ *i V t 4*]
using *m-mod-div mempool-free-stm41-h1-1 pow-mod-0* **by** *force*
with *a0 a1 a3 a4 p9* **show** *?thesis* **using** *align40* **by** *simp*
qed

lemma *mempool-free-stm41*:
$\Gamma \vdash_I$ *Some* (´*lsizes* := ´*lsizes*
(*t* := ´*lsizes t* @ [*ALIGN4* (´*lsizes t* ! (´*i t* − *1*) *div 4*)]))
sat_p [*mp-free-precond4-2 t b* ∩ ⦃´*cur* = *Some t*⦄ ∩ {*V*}, {(*s*, *t*). *s* = *t*}, *UNIV*,
⦃´(*Pair V*) ∈ *Mem-pool-free-guar t*⦄ ∩ (*mp-free-precond2 t b* ∩ ⦃¬ ´*need-resched t*⦄ ∩
⦃(∀ *ii*<*length* (´*lsizes t*).
´*lsizes t* ! *ii* = *ALIGN4* (*max-sz* (´*mem-pool-info* (*pool b*))) *div 4* ^ *ii*) ∧ ´*lsizes t* ≠ []⦄ ∩
(⦃´*i t* ≤ *level b*⦄ ∩ ⦃*length* (´*lsizes t*) = *Suc* (´*i t*)⦄))]
apply(*rule Basic*)
apply(*case-tac mp-free-precond4-2 t b* ∩ ⦃´*cur* = *Some t*⦄ ∩ {*V*} = {})
apply *simp* **apply** *clarify* **apply** *auto*[*1*]
apply(*simp add*:$Guar_f$-*def gvars-conf-stable-def gvars-conf-def Mem-pool-free-guar-def*)

apply(*rule disjI1*)
apply(*rule conjI*)
apply(*subgoal-tac* (*V*,*V*(|*lsizes* := (*lsizes V*)(*t* := *lsizes V t* @ [*ALIGN4* (*lsizes V t* ! (*i V t* − *Suc NULL*) *div 4*)])|))∈*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def*)
apply(*simp add*:*lvars-nochange-def*)
apply(*subgoal-tac* (*V*,*V*(|*lsizes* := (*lsizes V*)(*t* := *lsizes V t* @ [*ALIGN4* (*lsizes V t* ! (*i V t* − *Suc NULL*) *div 4*)])|))∈*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def*)

apply(*case-tac ii* < *i V t*) **apply** (*simp add*: *nth-append*)
apply(*case-tac ii* = *i V t*)
using *mempool-free-stm41-h1* **apply** *metis*
apply *simp*
by (*simp add*:*stable-def*)+

lemma *mempool-free-stm4*:
$\Gamma \vdash_I$ *Some* (*FOR* (t ▶ ´i := ´i(t := *1*));
´i t ≤ *level* b;
(t ▶ ´i := ´i(t := ´i t + *1*)) *DO*
(t ▶ ´*lsizes* := ´*lsizes*(t := ´*lsizes* t @ [*ALIGN4* (´*lsizes* t ! (´i t − *1*) *div* *4*)]))
ROF) *sat*$_p$ [*mp-free-precond4* t b, *Mem-pool-free-rely* t, *Mem-pool-free-guar* t, *mp-free-precond5* t b]
apply(*rule Seq*[**where** *mid*=*mp-free-precond4-1* t b])

apply(*simp add*:*stm-def*)
apply(*rule Await*)
using *mp-free-precond4-stb* **apply** *simp*
using *mp-free-precond4-1-stb* **apply** *simp*
apply(*rule allI*)
apply(*rule Basic*)
apply(*case-tac mp-free-precond4* t b ∩ {|´*cur* = *Some* t|} ∩ {*V*} = {})
apply *auto*[*1*]
apply(*simp add*:*Guar$_f$-def gvars-conf-stable-def gvars-conf-def Mem-pool-free-guar-def*)
apply *auto*[*1*]
apply(*subgoal-tac* (*V*,*V*(|*i* := (*i V*)(t := *Suc NULL*)|))∈*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def*)
apply(*simp add*:*lvars-nochange-def*)
apply(*subgoal-tac* (*V*,*V*(|*i* := (*i V*)(t := *Suc NULL*)|))∈*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def*)
apply(*simp add*:*stable-def*)+

apply(*rule While*)
using *mp-free-precond4-1-stb* **apply** *simp*
apply *auto*[*1*]
using *mp-free-precond5-stb* **apply** *simp*

apply(*rule Seq*[**where** *mid*=*mp-free-precond4-3* t b])

apply(*simp add*:*stm-def*)
apply(*rule Await*)

using *mp-free-precond4-2-stb* **apply** *simp*
using *mp-free-precond4-3-stb* **apply** *simp*
apply(*rule allI*)
using *mempool-free-stm41* **apply** *simp*

apply(*simp add:stm-def*)
apply(*rule Await*)
using *mp-free-precond4-3-stb* **apply** *simp*
using *mp-free-precond4-1-stb* **apply** *simp*
apply(*rule allI*)
apply(*rule Basic*)
apply(*case-tac mp-free-precond4-3 t b* $\cap$ $\{\!|\,\acute{}cur = Some\ t|\!\}$ $\cap$ $\{V\} = \{\}$)
apply *auto*[*1*] **apply** *clarify* **apply**(*simp add:gvars-conf-stable-def gvars-conf-def Mem-pool-free-guar-def*) **apply** *auto*[*1*]
apply(*subgoal-tac* $(V, V(\!|i := (i\ V)(t := Suc\ (i\ V\ t))|\!))$$\in$*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add:lvars-nochange1-4all-def lvars-nochange1-def*)
apply(*simp add:lvars-nochange-def*)
apply(*subgoal-tac* $(V, V(\!|i := (i\ V)(t := Suc\ (i\ V\ t))|\!))$$\in$*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add:lvars-nochange1-4all-def lvars-nochange1-def*)
apply(*simp add:stable-def*)+ **apply**(*simp add:*$Guar_f$*-def Mem-pool-free-guar-def Id-def*)

done

lemma *mempool-free-stm5*:
$\Gamma \vdash_I$ *Some* (t ▶ ´*free-block-r* := ´*free-block-r* (t := *True*))
sat_p [*mp-free-precond5 t b*, *Mem-pool-free-rely t*, *Mem-pool-free-guar t*, *mp-free-precond6 t b*]
apply(*simp add:stm-def*)
apply(*rule Await*)
using *mp-free-precond5-stb* **apply** *simp*
using *mp-free-precond6-stb* **apply** *simp*

apply *clarify*
apply(*rule Basic*)
apply(*case-tac mp-free-precond5 t b* $\cap$ $\{\!|\,\acute{}cur = Some\ t|\!\}$ $\cap$ $\{V\} = \{\}$)
apply *auto*[*1*]
apply *clarsimp*
apply(*rule conjI*)
apply(*simp add:*$Guar_f$*-def gvars-conf-stable-def gvars-conf-def Mem-pool-free-guar-def*)
apply(*rule disjI1*)
apply(*rule conjI*)
apply(*subgoal-tac* $(V, V(\!|$*free-block-r* := (*free-block-r* V)(t := *True*)$|\!))$ $\in$*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add:lvars-nochange1-4all-def*

lvars-nochange1-def)

apply *clarify* **apply**(*simp add*: *lvars-nochange-def*)
apply(*subgoal-tac* (V,V(|*free-block-r* := (*free-block-r* V)(t := *True*)|)) ∈*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def*
lvars-nochange1-def)
apply(*simp add*:*stable-def*)+
done

lemma *mempool-free-stm6*:
$\Gamma \vdash_I$ *Some* (t ▶ ´*bn* := ´*bn* (t := *block b*))
sat$_p$ [*mp-free-precond6 t b*, *Mem-pool-free-rely t*, *Mem-pool-free-guar t*, *mp-free-precond7*
t b]
apply(*simp add*:*stm-def*)
apply(*rule Await*)
using *mp-free-precond6-stb* **apply** *simp*
using *mp-free-precond7-stb* **apply** *simp*

apply *clarify*
apply(*rule Basic*)
apply(*case-tac mp-free-precond6 t b* ∩ {|´*cur* = *Some t*|} ∩ {V} = {})
apply *auto*[*1*]
apply *clarsimp*
apply(*rule conjI*)
apply(*simp add*:*Guar*$_f$-*def gvars-conf-stable-def gvars-conf-def Mem-pool-free-guar-def*)
apply(*rule disjI1*)
apply(*rule conjI*)
apply(*subgoal-tac* (V,V(|*bn* := (*bn* V)(t := *block b*)|)) ∈*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def*
lvars-nochange1-def)

apply *clarify* **apply**(*simp add*: *lvars-nochange-def*)
apply(*subgoal-tac* (V,V(|*bn* := (*bn* V)(t := *block b*)|)) ∈*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def*
lvars-nochange1-def)
apply(*simp add*:*stable-def*)+
done

lemma *mempool-free-stm7*:
$\Gamma \vdash_I$ *Some* (t ▶ ´*lvl* := ´*lvl* (t := *level b*))
sat$_p$ [*mp-free-precond7 t b*, *Mem-pool-free-rely t*, *Mem-pool-free-guar t*, *mp-free-precond8*
t b]
apply(*unfold stm-def*)
apply(*rule Await*)
using *mp-free-precond7-stb* **apply** *simp*
using *mp-free-precond8-stb*[*of t b*] **apply** *fast*

apply *clarify*
apply(*rule Basic*)

apply(*case-tac mp-free-precond7 t b* $\cap$ $\{\!|\,{}^\prime cur = Some\ t|\!\}$ $\cap$ $\{V\} = \{\}$)
apply *auto*[*1*]
apply *clarsimp*
apply(*rule conjI*)
apply(*simp add*:$Guar_f$*-def gvars-conf-stable-def gvars-conf-def Mem-pool-free-guar-def*)
apply(*rule disjI1*)
apply(*rule conjI*)
apply(*subgoal-tac* $(V, V(\!|lvl := (lvl\ V)(t := level\ b)|\!))$ $\in$*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def*)

apply *clarify* **apply**(*simp add*: *lvars-nochange-def*)
apply(*rule conjI*)
apply(*subgoal-tac* $(V, V(\!|lvl := (lvl\ V)(t := level\ b)|\!))$ $\in$*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def*)

apply *auto*[*1*]
apply (*simp add*: *block-ptr-def inv-maxsz-align4*)
apply (*metis inv-mempool-info-def inv-def*)
apply *simp*
apply(*simp add*:*stable-def*)
using *stable-id2* **apply** *metis*
done

9.3 statement 8

abbreviation *free-stm8-precond1 Va t b* $\equiv$ *Va*$(\!|$*mem-pool-info* := *set-bit-free* (*mem-pool-info Va*) (*pool b*) (*lvl Va t*) (*bn Va t*)$|\!)$
abbreviation *free-stm8-precond2 Va t b* $\equiv$ (*free-stm8-precond1 Va t b*)$(\!|$*freeing-node* := (*freeing-node Va*)(t := *None*)$|\!)$
abbreviation *free-stm8-loopinv1 Va t b* $\equiv$
$\{V.$ *let minf0* = (*mem-pool-info Va*)(*pool b*);
lvl0 = (*levels minf0*) ! (*lvl Va t*);
minf1 = (*mem-pool-info V*)(*pool b*);
lvl1 = (*levels minf1*) ! (*lvl Va t*) *in*
cur V = *cur Va* $\wedge$ *tick V* = *tick Va* $\wedge$ *thd-state V* = *thd-state Va* $\wedge$ (*V*,*Va*)$\in$*gvars-conf-stable*
$\wedge$ ($\forall p.\ p \neq$ *pool b* $\longrightarrow$ *mem-pool-info V p* = *mem-pool-info Va p*)
$\wedge$ ($\forall j.\ j \neq$ *lvl Va t* $\longrightarrow$ (*levels minf0*)!*j* = (*levels minf1*)!*j*)
$\wedge$ (*bits lvl1* = *list-updates-n* (*bits lvl0*) ((*bn Va t div 4*) $*$ *4*) (*i V t*) *NOEXIST*)
$\wedge$ (*free-list lvl1* = *removes* (*map* (λ*ii*. *block-ptr minf0* (*lsz Va t*) ((*bn Va t div 4*) $*$ *4* + *ii*)) [*0*..<(*i V t*)]) (*free-list lvl0*))
$\wedge$ (*wait-q minf0* = *wait-q minf1*)
$\wedge$ ($\forall t'.\ t' \neq t \longrightarrow$ *lvars-nochange* t' *V Va*)
$\wedge$ *freeing-node Va t* = *freeing-node V t* $\wedge$ *allocating-node Va t* = *allocating-node V t* $\wedge$ *free-block-r Va t* = *free-block-r V t*
$\wedge$ *bn Va t* = *bn V t* $\wedge$ *lvl Va t* = *lvl V t* $\wedge$ *lsz Va t* = *lsz V t* $\wedge$ *lsizes Va t* = *lsizes V t*

$\wedge$ *i V t* $\leq$ *4*}

lemma *V-free-stm8-loopinv1*: *i V t = 0* $\Longrightarrow$ *V* $\in$ *free-stm8-loopinv1 V t b*
by(*simp add*:*Let-def gvars-conf-stable-def gvars-conf-def lvars-nochange-def*)

abbreviation *free-stm8-precond3 Va t b* $\equiv$ *free-stm8-loopinv1* (*free-stm8-precond2 Va t b*) *t b*
abbreviation *free-stm8-precond4 Va t b* $\equiv$ *free-stm8-precond3 Va t b* $\cap$ $\{\!|$´*i t = 4*$|\!\}$

abbreviation *free-stm8-precond30 Va t b* $\equiv$ *free-stm8-precond3 Va t b* $\cap$ $\{\!|$´*i t < 4*$|\!\}$
abbreviation *free-stm8-precond31 V t b* $\equiv$ *V*(|*bb* := (*bb V*) (*t*:=(*bn V t div 4*) $*$ *4* + *i V t*)|)
abbreviation *free-stm8-precond32 V t b* $\equiv$
let minf = mem-pool-info V (*pool b*) *in*
V(|*mem-pool-info*:= (*mem-pool-info V*) (*pool b* := *minf* (|*levels* := (*levels minf*)

[*lvl V t* := ((*levels minf*) ! (*lvl V t*)) (|*bits* := (*bits* ((*levels minf*) ! (*lvl V t*))) [*bb V t* := *NOEXIST*]|)] |))|)
abbreviation *free-stm8-precond33 V t b* $\equiv$
V(|*block-pt* := (*block-pt V*) (*t*:=*block-ptr* (*mem-pool-info V* (*pool b*)) (*lsz V t*) (*bb V t*))|)
abbreviation *free-stm8-precond34 V t b* $\equiv$
let minf = mem-pool-info V (*pool b*) *in*
V(|*mem-pool-info*:= (*mem-pool-info V*) (*pool b* := *minf* (|*levels* := (*levels minf*)

[*lvl V t* := ((*levels minf*) ! (*lvl V t*)) (|*free-list* := *remove1* (*block-pt V t*) (*free-list* ((*levels minf*) ! (*lvl V t*)))|)] |))|)

lemma *mempool-free-stm8-atombody-h1*:
{*free-stm8-precond1 V t b*} $\subseteq$ $\{\!|$´(*freeing-node-update* (λ-. ´*freeing-node*(*t* := *None*))) $\in$ {*free-stm8-precond2 V t b*}$|\!\}$
by *fastforce*

lemma *block-fits0-h1*: *maxsz mod mm = 0* $\Longrightarrow$ *aa < nmax* $*$ *mm* $\Longrightarrow$
maxsz div mm $*$ *aa* + *maxsz div mm < Suc* (*nmax* $*$ *maxsz*)
apply(*subgoal-tac maxsz div mm* $*$ *aa* $\leq$ *maxsz div mm* $*$ (*nmax* $*$ *mm* − *1*))
prefer *2* **apply** *auto*[*1*]

by (*smt Groups.add-ac*(*2*) *Groups.mult-ac*(*2*) *Groups.mult-ac*(*3*) *One-nat-def Suc-leI distrib-left*
le-imp-less-Suc mod-div-self mult.right-neutral mult-0-right mult-Suc-right mult-less-cancel2 mult-zero-right not-le plus-nat.simps(*2*))

lemma *block-fits0-h2*: (*lvlt*::*nat*) > *0* $\Longrightarrow$ *lvlt* $\leq$ *lvlb* $\Longrightarrow$ *ivt* < (*4*::*nat*) $\Longrightarrow$ *blockb* < *nmax* $*$ *4* ˆ *lvlb* $\Longrightarrow$

blockb div 4 ^ (lvlb − lvlt) div 4 ∗ 4 + ivt < nmax ∗ 4 ^ lvlt
apply(*subgoal-tac nmax > 0*) **prefer** *2* **using** *mult-not-zero* **apply** *fastforce*
apply(*subgoal-tac blockb div 4 ^ (lvlb − lvlt) < (nmax ∗ 4 ^ lvlt)*) **prefer** *2*
apply(*subgoal-tac blockb < nmax ∗ 4 ^ (lvlt + (lvlb − lvlt)) ∧ nmax ∗ 4 ^ lvlt ≠ 0*) **prefer** *2* **apply** *simp*
apply(*subgoal-tac ⋀n na nb. ¬ n < na ∗ nb ∨ n div na < nb ∨ nb = NULL*)
prefer *2*
apply (*simp add*: *less-mult-imp-div-less mult.commute*)
apply (*metis mult.commute mult.left-commute power-add*)
apply(*subgoal-tac blockb div 4 ^ (lvlb − lvlt) div 4 ∗ 4 + 4 ≤ nmax ∗ 4 ^ lvlt*)
prefer *2* **apply**(*subgoal-tac ⋀x. x < nmax ∗ 4 ^ lvlt ⟶ x div 4 ∗ 4 + 4 ≤ nmax ∗ 4 ^ lvlt*)
prefer *2* **apply**(*case-tac x mod 4 = 0*) **apply** *auto*[*1*] **apply**(*rule modn0-xy-n*) **apply** *auto*[*1*] **apply** *auto*[*1*] **apply** *auto*[*1*] **apply** *auto*[*1*]
apply *auto*[*1*] **apply**(*rule divn-multn-addn-le*) **apply** *auto*[*1*] **apply** *auto*[*1*] **apply** *auto*[*1*]
apply *auto*
done

lemma *block-fits0*:
V ∈ mp-free-precond8-3 t b α ∩ ⦃´cur = Some t⦄ ⟹
vt ∈ free-stm8-precond3 V t b ∩ ⦃´i t < 4⦄ ⟹
{free-stm8-precond2 V t b} ∩ ⦃0 < ´lvl t ∧ partner-bits (´mem-pool-info (pool b)) (´lvl t) (´bn t)⦄ ≠ {} ⟹
free-stm8-precond33 (free-stm8-precond32 (free-stm8-precond31 vt t b) t b) t b
∈ ⦃block-fits (´mem-pool-info (pool b)) (´block-pt t) (´lsz t)⦄
apply(*unfold block-fits-def block-ptr-def buf-size-def*) **apply** *clarsimp*
apply(*rule subst*[**where** *s=lsz vt* **and** *t=lsz (let minf = mem-pool-info vt (pool b)*
in vt⦇bb := (bb vt)(t := bn vt t div 4 ∗ 4 + i vt t), mem-pool-info := (mem-pool-info vt)
(pool b := minf⦇levels := levels minf[lvl vt t := (levels minf ! lvl vt t)
⦇bits := bits (levels minf ! lvl vt t)[bn vt t div 4 ∗ 4 + i vt t := NOEXIST]⦈]⦈)⦈)])
apply(*simp add*:*Let-def*)
apply(*rule subst*[**where** *s=bn vt t div 4 ∗ 4 + i vt t* **and** *t=bb (let minf = mem-pool-info vt (pool b)*
in vt⦇bb := (bb vt)(t := bn vt t div 4 ∗ 4 + i vt t), mem-pool-info := (mem-pool-info vt)
(pool b := minf⦇levels := levels minf[lvl vt t := (levels minf ! lvl vt t)
⦇bits := bits (levels minf ! lvl vt t)[bn vt t div 4 ∗ 4 + i vt t := NOEXIST]⦈]⦈)⦈) t])
apply(*simp add*:*Let-def*)
apply(*rule subst*[**where** *s=n-max (mem-pool-info vt (pool b))* **and** *t=n-max (mem-pool-info*
(let minf = mem-pool-info vt (pool b)

in vt(|*bb* := (*bb vt*)(*t* := *bn vt t div 4* ∗ *4* + *i vt t*),
mem-pool-info := (*mem-pool-info vt*)
(*pool b* := *minf* (|*levels* := *levels minf*[*lvl vt t* := (*levels minf* ! *lvl vt t*)
(|*bits* := *bits* (*levels minf* ! *lvl vt t*)[*bn vt t div 4* ∗ *4* + *i vt t* := *NOEXIST*]|)]|))|))
(*pool b*))])
apply(*simp add*:*Let-def*)
apply(*rule subst*[**where** *s*=*max-sz* (*mem-pool-info vt* (*pool b*)) **and** *t*=*max-sz* (*mem-pool-info*
(*let minf* = *mem-pool-info vt* (*pool b*)
in vt(|*bb* := (*bb vt*)(*t* := *bn vt t div 4* ∗ *4* + *i vt t*),
mem-pool-info := (*mem-pool-info vt*)
(*pool b* := *minf* (|*levels* := *levels minf*[*lvl vt t* := (*levels minf* ! *lvl vt t*)
(|*bits* := *bits* (*levels minf* ! *lvl vt t*)[*bn vt t div 4* ∗ *4* + *i vt t* := *NOEXIST*]|)]|))|))
(*pool b*))])
apply(*simp add*:*Let-def*)

apply(*rule subst*[**where** *s*=*n-max* (*mem-pool-info V* (*pool b*)) **and** *t*=*n-max* (*mem-pool-info vt* (*pool b*))])
apply(*simp add*:*Let-def set-bit-def gvars-conf-stable-def gvars-conf-def*)

apply(*rule subst*[**where** *s*=*max-sz* (*mem-pool-info V* (*pool b*)) **and** *t*=*max-sz* (*mem-pool-info vt* (*pool b*))])
apply(*simp add*:*Let-def set-bit-def gvars-conf-stable-def gvars-conf-def*)
apply(*rule subst*[**where** *s*=*ALIGN4* (*max-sz* (*mem-pool-info V* (*pool b*))) *div 4* ˆ *lvl V t* **and** *t*=*lsz vt t*])
apply(*simp add*:*Let-def*) **apply** *metis*
apply(*rule subst*[**where** *s*=*block b div 4* ˆ (*level b* − *lvl V t*) **and** *t*=*bn vt t*])
apply(*simp add*:*Let-def*) **apply** *metis*
apply(*rule subst*[**where** *s*=*max-sz* (*mem-pool-info V* (*pool b*)) **and** *t*=*ALIGN4* (*max-sz* (*mem-pool-info V* (*pool b*)))])
apply(*simp add*: *inv-def*) **using** *inv-mempool-info-maxsz-align4*[*rule-format*,*of V pool b*] **apply** *metis*

apply(*subgoal-tac length* (*bits* ((*levels* (*mem-pool-info V* (*pool b*))) ! *level b*)) = (*n-max* (*mem-pool-info V* (*pool b*))) ∗ *4* ˆ (*level b*))
prefer *2* **apply**(*simp add*: *inv-def inv-mempool-info-def Let-def*)

apply(*subgoal-tac max-sz* (*mem-pool-info V* (*pool b*)) *mod 4* ˆ *lvl V t* = *0*)
prefer *2* **apply**(*subgoal-tac* ∃ *n*. *max-sz* (*mem-pool-info V* (*pool b*)) = (*4* ∗ *n*) ∗ (*4* ˆ *n-levels* (*mem-pool-info V* (*pool b*))))
prefer *2* **apply**(*simp add*:*inv-def*) **using** *inv-mempool-info-def*[*rule-format*, *of V*] **apply** *meson*
apply(*subgoal-tac length* (*levels* (*mem-pool-info V* (*pool b*))) = *n-levels* (*mem-pool-info V* (*pool b*)))

prefer *2* **apply**(*simp add*:*inv-def inv-mempool-info-def*) **apply** *metis*
apply(*simp add*: *inv-def inv-mempool-info-def*)
using *ge-pow-mod-0*[*of lvl V t n-levels* (*mem-pool-info V* (*pool b*))]
apply (*metis add-diff-inverse-nat add-lessD1 ge-pow-mod-0 le-antisym nat-less-le*)

apply(*subgoal-tac block b div 4 ^ (level b − lvl V t) div 4 ∗ 4 + i vt t < n-max* (*mem-pool-info V* (*pool b*)) *∗ 4 ^ lvl V t*)
prefer *2* **apply**(*rule block-fits0-h2*[*of lvl V t level b i vt t block b n-max* (*mem-pool-info V* (*pool b*))])
apply *blast* **apply** *blast* **apply** *blast* **apply** *linarith*

apply(*rule block-fits0-h1*[*of max-sz* (*mem-pool-info V* (*pool b*)) *4 ^ lvl V t*
block b div 4 ^ (level b − lvl V t) div 4 ∗ 4 + i vt t n-max (*mem-pool-info V* (*pool b*))])
apply *blast* **apply** *blast*
done

lemma *block-fits1*:
$V \in$ *mp-free-precond8-3 t b* $\alpha \cap$ {|´*cur = Some t*|} $\Longrightarrow$
vt $\in$ *free-stm8-precond3 V t b* $\cap$ {|´*i t* < *4*|} $\Longrightarrow$
{*free-stm8-precond2 V t b*} $\cap$ {|*NULL* < ´*lvl t* $\wedge$ *partner-bits* (´*mem-pool-info* (*pool b*)) (´*lvl t*) (´*bn t*)|} $\neq$ {} $\Longrightarrow$
{*free-stm8-precond33* (*free-stm8-precond32* (*free-stm8-precond31 vt t b*) *t b*) *t b*}
$\cap$
− {|*block-fits* (´*mem-pool-info* (*pool b*)) (´*block-pt t*) (´*lsz t*)|} = {}
using *block-fits0*[*of V t b* α *vt*] **apply** *fast*
done

lemma *mempool-free-stm8-set4partbits-while-one-h1*:
$\neg$ *bn* (*free-stm8-precond33* (*free-stm8-precond32* (*vt*(|*bb* := (*bb vt*)(*t* := *bn vt t div 4 ∗ 4 + i vt t*)|)) *t b*) *t b*) *t* $\neq$
bb (*free-stm8-precond33* (*free-stm8-precond32* (*vt*(|*bb* := (*bb vt*)(*t* := *bn vt t div 4 ∗ 4 + i vt t*)|)) *t b*) *t b*) *t* $\Longrightarrow$
{*free-stm8-precond33* (*free-stm8-precond32* (*vt*(|*bb* := (*bb vt*)(*t* := *bn vt t div 4 ∗ 4 + i vt t*)|)) *t b*) *t b*}
$\subseteq$ {|´*id* $\in$ {*let vv = free-stm8-precond33* (*free-stm8-precond32* (*vt*(|*bb* := (*bb vt*)(*t* := *bn vt t div 4 ∗ 4 + i vt t*)|)) *t b*) *t b*
in if bn vv t = bb vv t then vv else free-stm8-precond34 vv t b}|}
by(*simp add*:*Let-def*)

lemma *mempool-free-stm8-set4partbits-while-one-isuc-h1-1*:
$\forall$ *p*. ($\forall$ *i*. *length* (*bits* (*levels* (*mem-pool-info vt p*) ! *i*)) =
length (*bits* (*levels* (*if p = pool b*
then mem-pool-info V (*pool b*)
(|*levels* := *levels* (*mem-pool-info V* (*pool b*))
[*lvl vt t* := (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl vt t*)

(|*bits* := *bits* (*levels* (*mem-pool-info* *V* (*pool* *b*)) ! *lvl* *vt* *t*)[*bn* *vt* *t* := *FREE*]|)]|)
else *mem-pool-info* *V* *p*) !
i))) $\Longrightarrow$
$\forall$ *j*. *j* $\neq$ *lvl* *vt* *t* $\longrightarrow$ *levels* (*mem-pool-info* *V* (*pool* *b*)) ! *j* = *levels* (*mem-pool-info* *vt* (*pool* *b*)) ! *j* $\Longrightarrow$
length (*levels* (*mem-pool-info* *V* (*pool* *b*)))=*length* (*levels* (*mem-pool-info* *vt* (*pool* *b*))) $\Longrightarrow$
length (*bits* (*levels* (*mem-pool-info* *vt* (*pool* *b*))
[*lvl* *vt* *t* := (*levels* (*mem-pool-info* *vt* (*pool* *b*)) ! *lvl* *vt* *t*)
(|*bits* := *list-updates-n* (*bits* (*levels* (*mem-pool-info* *V* (*pool* *b*)) ! *lvl* *vt* *t*)[*bn* *vt* *t* := *FREE*]) (*bn* *vt* *t* *div* *4* $*$ *4*) (*i* *vt* *t*)
NOEXIST
[*bn* *vt* *t* := *NOEXIST*]|)] !
ia)) =
length (*bits* (*levels* (*mem-pool-info* *V* (*pool* *b*))
[*lvl* *vt* *t* := (*levels* (*mem-pool-info* *V* (*pool* *b*)) ! *lvl* *vt* *t*)
(|*bits* := *bits* (*levels* (*mem-pool-info* *V* (*pool* *b*)) ! *lvl* *vt* *t*)[*bn* *vt* *t* := *FREE*]|)] !
ia))

apply(*case-tac* *ia* < *length* (*levels* (*mem-pool-info* *V* (*pool* *b*))))
apply(*case-tac* *ia* = *lvl* *vt* *t*) **apply** *auto*[*1*]
apply (*metis* (*no-types*, *lifting*) *nth-list-update-neq*)
by (*smt* *list-eq-iff-nth-eq* *list-update-beyond* *not-less* *nth-list-update-neq*)

lemma *mempool-free-stm8-set4partbits-while-one-isuc-h1-2*:
$\neg$ *free-block-r* *vt* *t* $\longrightarrow$ *freeing-node* *V* *t* = *None* $\Longrightarrow$
free-block-r *V* *t* = *free-block-r* *vt* *t* $\Longrightarrow$
α = (*if* $\exists$ *y*. *freeing-node* *V* *t* = *Some* *y* *then* *lvl* *V* *t* + *1* *else* *NULL*) $\Longrightarrow$
block *b* *div* *4* ^ (*level* *b* − *lvl* *vt* *t*) = *bn* *vt* *t* $\Longrightarrow$
V $\in$ (*if* *NULL* < (*if* $\exists$ *y*. *freeing-node* *V* *t* = *Some* *y* *then* *lvl* *V* *t* + *1* *else* *NULL*) *then* *UNIV* *else* {}) $\Longrightarrow$
free-block-r *V* *t*
by *force*

lemma *mempool-free-stm8-set4partbits-while-one-isuc-h2*:
inv *V* $\Longrightarrow$
pool *b* $\in$ *mem-pools* *V* $\Longrightarrow$
lvl *vt* *t* < *length* (*levels* (*mem-pool-info* *V* (*pool* *b*))) $\Longrightarrow$
bn *vt* *t* < *length* (*bits* (*levels* (*mem-pool-info* *V* (*pool* *b*)) ! *lvl* *vt* *t*)) $\Longrightarrow$
get-bit-s *V* (*pool* *b*) (*lvl* *vt* *t*) (*bn* *vt* *t*) $\neq$ *FREE* $\Longrightarrow$
buf (*mem-pool-info* *V* (*pool* *b*)) + *max-sz* (*mem-pool-info* *V* (*pool* *b*)) *div* *4* ^ *lvl* *vt* *t* $*$ *bn* *vt* *t*
$\notin$ *set* (*free-list* (*levels* (*mem-pool-info* *V* (*pool* *b*)) ! *lvl* *vt* *t*))
apply(*simp* *add*:*inv-def* *inv-bitmap-freelist-def* *Let-def*)
apply(*rule* *subst*[**where** *t*=*max-sz* (*mem-pool-info* *V* (*pool* *b*)) *div* *4* ^ *lvl* *vt* *t* $*$ *bn*

vt t **and**
s=bn vt t $*$ (*max-sz* (*mem-pool-info V* (*pool b*)) *div 4* ^ *lvl vt t*)])
apply *simp* **apply** *simp*
done

lemma *mempool-free-stm8-set4partbits-while-one-isuc-h1-3*:
$\forall$ *p*. ($\forall$ *i*. *length* (*bits* (*levels* (*mem-pool-info vt p*) ! *i*)) =
length (*bits* (*levels* (*if p* = *pool b*
then mem-pool-info V (*pool b*)
(|*levels* := *levels* (*mem-pool-info V* (*pool b*))
[*lvl vt t* := (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl*
vt t)
(|*bits* := *bits* (*levels* (*mem-pool-info V* (*pool b*))
! *lvl vt t*)[*bn vt t* := *FREE*]|)]|)
else mem-pool-info V p) !
i))) $\Longrightarrow$
$\forall$ *j*. *j* $\neq$ *lvl vt t* $\longrightarrow$ *levels* (*mem-pool-info V* (*pool b*)) ! *j* = *levels* (*mem-pool-info vt* (*pool b*)) ! *j* $\Longrightarrow$
length (*levels* (*mem-pool-info V* (*pool b*)))=*length* (*levels* (*mem-pool-info vt* (*pool b*))) $\Longrightarrow$
length (*bits* (*levels* (*mem-pool-info vt* (*pool b*))
[*lvl vt t* := (*levels* (*mem-pool-info vt* (*pool b*)) ! *lvl vt t*)
(|*bits* := *list-updates-n* (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl vt t*)[*bn vt t* := *FREE*]) (*bn vt t div 4* $*$ *4*) (*i vt t*)
NOEXIST
[*bn vt t div 4* $*$ *4* + *i vt t* := *NOEXIST*],
free-list :=
remove1 (*block-ptr*
(*mem-pool-info vt* (*pool b*)
(|*levels* := *levels* (*mem-pool-info vt* (*pool b*))
[*lvl vt t* := (*levels* (*mem-pool-info vt* (*pool b*)) ! *lvl vt t*)
(|*bits* := *list-updates-n* (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl vt t*)[*bn vt t* := *FREE*])
(*bn vt t div 4* $*$ *4*) (*i vt t*) *NOEXIST*
[*bn vt t div 4* $*$ *4* + *i vt t* := *NOEXIST*]|)]|))
(*lsz vt t*) (*bn vt t div 4* $*$ *4* + *i vt t*))
(*removes* (*map* (λ*ii*. *block-ptr*
(*mem-pool-info V* (*pool b*)
(|*levels* := *levels* (*mem-pool-info V* (*pool b*))
[*lvl vt t* := (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl vt t*)
(|*bits* := *bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl vt t*)[*bn vt t* := *FREE*]|)]|))
(*lsz vt t*) (*bn vt t div 4* $*$ *4* + *ii*))
[*NULL*..<*i vt t*])
(*free-list* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl vt t*)))|)] !
ia)) =

length (bits (levels (mem-pool-info V (pool b))
[lvl vt t := (levels (mem-pool-info V (pool b)) ! lvl vt t)
(|bits := bits (levels (mem-pool-info V (pool b)) ! lvl vt t)[bn vt t := FREE]|)] !
ia))

apply(case-tac ia < length (levels (mem-pool-info V (pool b))))
apply(case-tac ia = lvl vt t) **apply** auto[1]
apply (metis (no-types, lifting) nth-list-update-neq)
by (smt list-eq-iff-nth-eq list-update-beyond not-less nth-list-update-neq)

lemma mempool-free-stm8-set4partbits-while-one-isuc-h1:
$V \in$ mp-free-precond8-3 t b $\alpha \cap$ {|´cur = Some t|} $\Longrightarrow$
vt $\in$ free-stm8-precond3 V t b $\cap$ {|´i t < 4|} $\Longrightarrow$
{free-stm8-precond2 V t b} $\cap$ {|NULL < ´lvl t $\wedge$ partner-bits (´mem-pool-info (pool b)) (´lvl t) (´bn t)|} $\neq$ {} $\Longrightarrow$
{let vv = free-stm8-precond33 (free-stm8-precond32 (free-stm8-precond31 vt t b) t b) t b in
if bn vv t = bb vv t then vv else free-stm8-precond34 vv t b}
$\subseteq$ {s. s(|i := (i s) (t := Suc (i s t))|) $\in$ free-stm8-precond3 V t b}
apply(simp add:Let-def set-bit-def)

apply(rule conjI)
apply clarsimp
apply(rule conjI)
apply(simp add: gvars-conf-stable-def gvars-conf-def)
apply clarsimp
apply(subgoal-tac length (levels (mem-pool-info V (pool b)))=length (levels (mem-pool-info vt (pool b))))
prefer 2 **apply** simp
using mempool-free-stm8-set4partbits-while-one-isuc-h1-1 **apply** blast
apply(rule conjI)
apply(subgoal-tac length (levels (mem-pool-info V (pool b)))=length (levels (mem-pool-info vt (pool b))))
prefer 2 **apply**(simp add: gvars-conf-stable-def gvars-conf-def)

apply(rule subst[**where** s=list-updates-n (bits (levels (mem-pool-info V (pool b)) ! lvl vt t)[bn vt t := FREE]) (bn vt t div 4 * 4) (i vt t) NOEXIST
[bn vt t := NOEXIST] **and** t=bits (levels (mem-pool-info vt (pool b))
[lvl vt t := (levels (mem-pool-info vt (pool b)) ! lvl vt t)
(|bits := list-updates-n (bits (levels (mem-pool-info V (pool b)) ! lvl vt t)[bn vt t := FREE]) (bn vt t div 4 * 4) (i vt t) NOEXIST
[bn vt t := NOEXIST]|)] !
lvl vt t)]) **apply** auto[1]
using lst-updts-eq-updts-updt[of Suc (i vt t) bits (levels (mem-pool-info V (pool b)) ! lvl vt t)[bn vt t := FREE]
bn vt t div 4 * 4 NOEXIST] **apply** auto[1]
apply(rule conjI)
apply(simp add:block-ptr-def)

apply(*rule subst*[**where** *s=free-list* (*levels* (*mem-pool-info vt* (*pool b*)) ! *lvl vt t*) **and**
t=free-list (*levels* (*mem-pool-info vt* (*pool b*))
[*lvl vt t* := (*levels* (*mem-pool-info vt* (*pool b*)) ! *lvl vt t*)
(|*bits* := *list-updates-n* (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl vt t*)[*bn vt t* := *FREE*]) (*bn vt t div 4* ∗ *4*) (*i vt t*) *NOEXIST*
[*bn vt t* := *NOEXIST*]|)] ! *lvl vt t*)])
apply(*case-tac lvl vt t* < *length* (*levels* (*mem-pool-info vt* (*pool b*)))) **apply** *auto*[*1*] **apply** *auto*[*1*]
apply(*subgoal-tac removes* (*map* (λ*ii*. *buf* (*mem-pool-info V* (*pool b*)) + *lsz vt t* ∗ (*bn vt t div 4* ∗ *4* + *ii*)) [*NULL*..<*i vt t*] @
[*buf* (*mem-pool-info V* (*pool b*)) + *lsz vt t* ∗ *bn vt t*])
(*free-list* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl vt t*)) =
removes (*map* (λ*ii*. *buf* (*mem-pool-info V* (*pool b*)) + *lsz vt t* ∗ (*bn vt t div 4* ∗ *4* + *ii*)) [*NULL*..<*i vt t*])
(*free-list* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl vt t*))) **apply** *metis*
apply(*rule rmvs-onemore-same*)
apply(*simp add*:*inv-def inv-bitmap-freelist-def Let-def*)
apply(*subgoal-tac get-bit* (*mem-pool-info V*) (*pool b*) (*lvl vt t*) (*bn vt t*) = *FREEING*) **prefer** *2*
apply(*subgoal-tac free-block-r V t*) **prefer** *2*
using *mempool-free-stm8-set4partbits-while-one-isuc-h1-2* **apply** *blast*
apply(*subgoal-tac* ∃ *blk*. *freeing-node V t* = *Some blk* ∧ *pool blk* = *pool b* ∧ *level blk* = *lvl vt t* ∧ *block blk* = *bn vt t*)
prefer *2* **apply** *fast*
apply(*simp add*:*inv-def inv-aux-vars-def*) **apply** *metis*
apply(*rule subst*[**where** *s=max-sz* (*mem-pool-info V* (*pool b*)) *div 4* ˆ *lvl vt t* **and** *t=lsz vt t*])
using *inv-maxsz-align4*[*rule-format*, *of V pool b*] **apply** *force*
apply(*subgoal-tac get-bit* (*mem-pool-info V*) (*pool b*) (*lvl vt t*) (*bn vt t*) ≠ *FREE*) **prefer** *2* **apply** *auto*[*1*]
apply(*subgoal-tac lvl vt t* < *length* (*levels* (*mem-pool-info V* (*pool b*)))) **prefer** *2* **apply** *force*
using *mempool-free-stm8-set4partbits-while-one-isuc-h2* **apply** *blast*

apply *clarsimp* **apply**(*simp add*:*block-ptr-def lvars-nochange-def*)

apply *clarsimp*
apply(*rule conjI*)
apply(*simp add*: *gvars-conf-stable-def gvars-conf-def*)
apply *clarsimp*
apply(*subgoal-tac length* (*levels* (*mem-pool-info V* (*pool b*)))=*length* (*levels* (*mem-pool-info vt* (*pool b*))))
prefer *2* **apply** *simp*
using *mempool-free-stm8-set4partbits-while-one-isuc-h1-3* **apply** *blast*
apply(*rule conjI*)
apply(*subgoal-tac length* (*levels* (*mem-pool-info V* (*pool b*)))=*length* (*levels* (*mem-pool-info vt* (*pool b*))))

prefer *2* **apply**(*simp add*: *gvars-conf-stable-def gvars-conf-def*)
apply(*rule subst*[**where** *s*=*list-updates-n* (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl vt t*)[*bn vt t* := *FREE*]) (*bn vt t div 4* ∗ *4*) (*i vt t*) *NOEXIST*
[*bn vt t div 4* ∗ *4* + *i vt t* := *NOEXIST*] **and** *t*=*bits* (*levels* (*mem-pool-info vt* (*pool b*))
[*lvl vt t* :=
(*levels* (*mem-pool-info vt* (*pool b*))
[*lvl vt t* := (*levels* (*mem-pool-info vt* (*pool b*)) ! *lvl vt t*)
(|*bits* := *list-updates-n* (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl vt t*)[*bn vt t* := *FREE*]) (*bn vt t div 4* ∗ *4*) (*i vt t*) *NOEXIST*
[*bn vt t div 4* ∗ *4* + *i vt t* := *NOEXIST*]|)] !
lvl vt t)
(|*free-list* :=
remove1 (*block-ptr*
(*mem-pool-info vt* (*pool b*)
(|*levels* := *levels* (*mem-pool-info vt* (*pool b*))
[*lvl vt t* := (*levels* (*mem-pool-info vt* (*pool b*)) ! *lvl vt t*)
(|*bits* := *list-updates-n* (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl vt t*)[*bn vt t* := *FREE*]) (*bn vt t div 4* ∗ *4*) (*i vt t*)
NOEXIST
[*bn vt t div 4* ∗ *4* + *i vt t* := *NOEXIST*]|)]|))
(*lsz vt t*) (*bn vt t div 4* ∗ *4* + *i vt t*))
(*free-list*
(*levels* (*mem-pool-info vt* (*pool b*))
[*lvl vt t* := (*levels* (*mem-pool-info vt* (*pool b*)) ! *lvl vt t*)
(|*bits* := *list-updates-n* (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl vt t*)[*bn vt t* := *FREE*]) (*bn vt t div 4* ∗ *4*) (*i vt t*) *NOEXIST*
[*bn vt t div 4* ∗ *4* + *i vt t* := *NOEXIST*]|)] !
lvl vt t))|)] !
lvl vt t)]) **apply** *auto*[*1*]
using *lst-updts-eq-updts-updt*[*of Suc* (*i vt t*) *bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl vt t*)[*bn vt t* := *FREE*]
bn vt t div 4 ∗ *4 NOEXIST*] **apply** *auto*[*1*]

apply(*rule conjI*)
apply(*simp add*:*block-ptr-def*)
apply(*subgoal-tac lvl vt t* < *length* (*levels* (*mem-pool-info vt* (*pool b*)))) **prefer** *2*
apply(*subgoal-tac length* (*levels* (*mem-pool-info V* (*pool b*)))=*length* (*levels* (*mem-pool-info vt* (*pool b*))))
prefer *2* **apply**(*simp add*: *gvars-conf-stable-def gvars-conf-def*)
apply *force*
apply(*rule subst*[**where** *s*=*free-list* (*levels* (*mem-pool-info vt* (*pool b*)) ! *lvl vt t*) **and**
t=*free-list* (*levels* (*mem-pool-info vt* (*pool b*))
[*lvl vt t* := (*levels* (*mem-pool-info vt* (*pool b*)) ! *lvl vt t*)
(|*bits* := *list-updates-n* (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl vt t*)[*bn vt t* := *FREE*]) (*bn vt t div 4* ∗ *4*) (*i vt t*) *NOEXIST*
[*bn vt t div 4* ∗ *4* + *i vt t* := *NOEXIST*]|)] ! *lvl vt t*)])

apply *auto*[*1*]
apply(*rule subst*[**where** *s=remove1* (*buf* (*mem-pool-info vt* (*pool b*)) + *lsz vt t* ∗ (*bn vt t div 4* ∗ *4* + *i vt t*))
(*free-list* (*levels* (*mem-pool-info vt* (*pool b*)) ! *lvl vt t*)) **and**
t=free-list (*levels* (*mem-pool-info vt* (*pool b*))
[*lvl vt t* :=
(*levels* (*mem-pool-info vt* (*pool b*))
[*lvl vt t* := (*levels* (*mem-pool-info vt* (*pool b*)) ! *lvl vt t*)
(|*bits* := *list-updates-n* (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl vt t*)[*bn vt t* := *FREE*]) (*bn vt t div 4* ∗ *4*) (*i vt t*) *NOEXIST*
[*bn vt t div 4* ∗ *4* + *i vt t* := *NOEXIST*]|)] !
lvl vt t)
(|*free-list* :=
remove1 (*buf* (*mem-pool-info vt* (*pool b*)) + *lsz vt t* ∗ (*bn vt t div 4* ∗ *4* + *i vt t*))
(*free-list* (*levels* (*mem-pool-info vt* (*pool b*)) ! *lvl vt t*))|)] !
lvl vt t)])
apply *auto*[*1*]
apply(*subgoal-tac buf* (*mem-pool-info vt* (*pool b*)) = *buf* (*mem-pool-info V* (*pool b*))) **prefer** *2*
apply(*simp add*: *gvars-conf-stable-def gvars-conf-def*)
using *rmvs-rev*[*of* (*map* (*λii. buf* (*mem-pool-info V* (*pool b*)) + *lsz vt t* ∗ (*bn vt t div 4* ∗ *4* + *ii*)) [*NULL..<i vt t*])
buf (*mem-pool-info V* (*pool b*)) + *lsz vt t* ∗ (*bn vt t div 4* ∗ *4* + *i vt t*)
free-list (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl vt t*)] **apply** *simp*

apply *clarsimp* **apply**(*simp add*:*block-ptr-def lvars-nochange-def*)
done

lemma *mempool-free-stm8-set4partbits-while-one-isuc*:
$V = Va \Longrightarrow$
$V \in$ *mp-free-precond8-3 t b* $\alpha \cap$ {|´*cur* = *Some t*|} $\Longrightarrow$
vt ∈ *free-stm8-precond3 Va t b* ∩ {|´*i t* < *4*|} ⟹
{*free-stm8-precond2 V t b*} ∩ {|*NULL* < ´*lvl t* ∧ *partner-bits* (´*mem-pool-info* (*pool b*)) (´*lvl t*) (´*bn t*)|} ≠ {} ⟹
$\Gamma \vdash_I$ *Some* (´*i* := ´*i*(*t* := *Suc* (´*i t*)))
sat_p [{*let vv* = *free-stm8-precond33* (*free-stm8-precond32* (*free-stm8-precond31 vt t b*) *t b*) *t b in*
if bn vv t = *bb vv t then vv else free-stm8-precond34 vv t b*}, {(*s*, *t*). *s* = *t*}, *UNIV*, *free-stm8-precond3 Va t b*]
apply(*rule Basic*)
defer *1*
apply *fast* **using** *stable-id2* **apply** *fast* **using** *stable-id2* **apply** *fast*

using *mempool-free-stm8-set4partbits-while-one-isuc-h1*[*of Va t b α vt*] **apply** *fast*

done

lemma *mempool-free-stm8-set4partbits-while-one*:
$V = Va \Longrightarrow$
$V \in$ *mp-free-precond8-3 t b* $\alpha \cap \{\!|\,\acute{}cur = Some\ t|\!\} \Longrightarrow$
$vt \in$ *free-stm8-precond3 Va t b* $\cap \{\!|\,\acute{}i\ t < 4|\!\} \Longrightarrow$
$\{$*free-stm8-precond2 V t b*$\} \cap \{\!|NULL < \acute{}lvl\ t \wedge$ *partner-bits* $(\acute{}$*mem-pool-info* $(pool\ b))\ (\acute{}lvl\ t)\ (\acute{}bn\ t)|\!\} \neq \{\} \Longrightarrow$
$\Gamma \vdash_I Some\ (\acute{}bb := \acute{}bb(t := \acute{}bn\ t\ div\ 4 * 4 + \acute{}i\ t);;$
$\acute{}$*mem-pool-info* := *set-bit-noexist* $\acute{}$*mem-pool-info* $(pool\ b)\ (\acute{}lvl\ t)\ (\acute{}bb\ t);;$
$\acute{}$*block-pt* := $\acute{}$*block-pt* $(t :=$ *block-ptr* $(\acute{}$*mem-pool-info* $(pool\ b))\ (\acute{}lsz\ t)\ (\acute{}bb\ t));;$
IF $\acute{}bn\ t \neq \acute{}bb\ t \wedge$ *block-fits* $(\acute{}$*mem-pool-info* $(pool\ b))\ (\acute{}$*block-pt* $t)\ (\acute{}lsz\ t)$ *THEN*
$\acute{}$*mem-pool-info* := $\acute{}$*mem-pool-info*$(pool\ b :=$ *remove-free-list* $(\acute{}$*mem-pool-info* $(pool\ b))\ (\acute{}lvl\ t)\ (\acute{}$*block-pt* $t))$
FI;;
$\acute{}i := \acute{}i(t := Suc\ (\acute{}i\ t)))$
$sat_p\ [\{vt\},\ \{(s,\ t).\ s = t\},\ UNIV,$ *free-stm8-precond3 Va t b*$]$
apply(*rule Seq*[**where** *mid*=$\{$*let vv* = *free-stm8-precond33* (*free-stm8-precond32* (*free-stm8-precond31 vt t b*) *t b*) *t b in*
if bn vv t = bb vv t then vv else free-stm8-precond34 vv t b$\}$])
apply(*rule Seq*[**where** *mid*=$\{$*free-stm8-precond33* (*free-stm8-precond32* (*free-stm8-precond31 vt t b*) *t b*) *t b*$\}$])
apply(*rule Seq*[**where** *mid*=$\{$*free-stm8-precond32* (*free-stm8-precond31 vt t b*) *t b*$\}$])
apply(*rule Seq*[**where** *mid*=$\{$*free-stm8-precond31 vt t b*$\}$])

apply(*rule Basic*)
apply *fast* **apply** *fast* **using** *stable-id2* **apply** *fast* **using** *stable-id2* **apply** *fast*

apply(*rule Basic*)
apply(*simp add*:*Let-def set-bit-def*)
apply *fast* **using** *stable-id2* **apply** *fast* **using** *stable-id2* **apply** *fast*

apply(*rule Basic*)
apply *fast* **apply** *fast* **using** *stable-id2* **apply** *fast* **using** *stable-id2* **apply** *fast*

apply(*rule Cond*)
using *stable-id2* **apply** *fast*

apply(*rule Basic*)
apply(*simp add*:*Let-def remove-free-list-def*) **apply** *auto*[*1*]

apply *fast* **using** *stable-id2* **apply** *fast* **using** *stable-id2* **apply** *fast*

apply(*case-tac bn* (*free-stm8-precond33* (*free-stm8-precond32* (*free-stm8-precond31* *vt t b*) *t b*) *t b*) *t*
$\neq$ *bb* (*free-stm8-precond33* (*free-stm8-precond32* (*free-stm8-precond31* *vt t b*) *t b*) *t b*) *t*)
apply(*rule subst*[**where** *s*={*free-stm8-precond33* (*free-stm8-precond32* (*free-stm8-precond31* *vt t b*) *t b*) *t b*} ∩
− {|*block-fits* (´*mem-pool-info* (*pool b*)) (´*block-pt t*) (´*lsz t*)|}
and *t*={*free-stm8-precond33* (*free-stm8-precond32* (*free-stm8-precond31* *vt t b*) *t b*) *t b*} ∩
− {|´*bn t* ≠ ´*bb t* ∧ *block-fits* (´*mem-pool-info* (*pool b*)) (´*block-pt* *t*) (´*lsz t*)|}]) **apply** *fast*
apply(*rule subst*[**where** *s*={} **and** *t*={*free-stm8-precond33* (*free-stm8-precond32* (*free-stm8-precond31 vt t b*) *t b*) *t b*} ∩
− {|*block-fits* (´*mem-pool-info* (*pool b*)) (´*block-pt t*) (´*lsz t*)|}])
using *block-fits1*[*of V t b* α *vt*] **apply** *fast*
using *Emptyprecond* **apply** *fast*

apply(*rule subst*[**where** *s*={*free-stm8-precond33* (*free-stm8-precond32* (*free-stm8-precond31* *vt t b*) *t b*) *t b*}
and *t*={*free-stm8-precond33* (*free-stm8-precond32* (*free-stm8-precond31* *vt t b*) *t b*) *t b*} ∩
− {|´*bn t* ≠ ´*bb t* ∧ *block-fits* (´*mem-pool-info* (*pool b*)) (´*block-pt* *t*) (´*lsz t*)|}])
apply *fast*

apply(*unfold Skip-def*)
apply(*rule Basic*)
using *mempool-free-stm8-set4partbits-while-one-h1* **apply** *fast*
apply *fast*
using *stable-id2* **apply** *fast*
using *stable-id2* **apply** *fast*

apply *fast*

using *mempool-free-stm8-set4partbits-while-one-isuc*[*of V Va t b* α *vt*] **apply** *fast*
done

lemma *mempool-free-stm8-set4partbits-while*:
V = *Va* ⟹
V∈*mp-free-precond8-3 t b* α ∩ {|´*cur* = *Some t*|} ⟹
{*free-stm8-precond2 V t b*} ∩ {|*NULL* < ´*lvl t* ∧ *partner-bits* (´*mem-pool-info* (*pool b*)) (´*lvl t*) (´*bn t*)|} ≠ {} ⟹
Γ $\vdash_I$ *Some*(´*bb* := ´*bb*(*t* := ´*bn t div 4* ∗ *4* + ´*i t*);;
´*mem-pool-info* := *set-bit-noexist* ´*mem-pool-info* (*pool b*) (´*lvl t*) (´*bb t*);;

´block-pt := *´block-pt* (*t* := *block-ptr* (*´mem-pool-info* (*pool b*)) (*´lsz t*) (*´bb t*));;
IF ´bn t ≠ *´bb t* ∧ *block-fits* (*´mem-pool-info* (*pool b*)) (*´block-pt t*) (*´lsz t*) *THEN*
´mem-pool-info := *´mem-pool-info*(*pool b* := *remove-free-list* (*´mem-pool-info* (*pool b*)) (*´lvl t*) (*´block-pt t*))
FI;;
´i := *´i*(*t* := *Suc* (*´i t*)))
sat_p [*free-stm8-precond3 Va t b* ∩ ⦃*´i t* < *4*⦄, {(*s*, *t*). *s* = *t*}, *UNIV*, *free-stm8-precond3 Va t b*]
using *mempool-free-stm8-set4partbits-while-one*[*of V Va t b α*]
Allprecond[**where** *U*=*free-stm8-precond3 Va t b* ∩ ⦃*´i t* < *4*⦄ **and**
P=*Some* (*´bb* := *´bb*(*t* := *´bn t div 4* ∗ *4* + *´i t*);;
´mem-pool-info := *set-bit-noexist ´mem-pool-info* (*pool b*) (*´lvl t*) (*´bb t*);;
´block-pt := *´block-pt* (*t* := *block-ptr* (*´mem-pool-info* (*pool b*)) (*´lsz t*) (*´bb t*));;
IF ´bn t ≠ *´bb t* ∧ *block-fits* (*´mem-pool-info* (*pool b*)) (*´block-pt t*) (*´lsz t*) *THEN*
´mem-pool-info := *´mem-pool-info*(*pool b* := *remove-free-list* (*´mem-pool-info* (*pool b*)) (*´lvl t*) (*´block-pt t*))
FI;;
´i := *´i*(*t* := *Suc* (*´i t*))) **and**
rely={(*x*, *y*). *x* = *y*} **and**
guar= *UNIV* **and** *post*= *free-stm8-precond3 Va t b*]
apply *meson*
done

term *free-stm8-precond3 Va t b*

abbreviation *free-stm8-atombody-rest-cond1 V t b* ≡ *V*⦇*lvl* := (*lvl V*)(*t* := *lvl V t* − *1*)⦈
abbreviation *free-stm8-atombody-rest-cond2 V t b* ≡ *V*⦇*bn* := (*bn V*)(*t* := *bn V t div 4*)⦈
abbreviation *free-stm8-atombody-rest-cond3 V t b* ≡
let minf = *mem-pool-info V* (*pool b*) *in*
V⦇*mem-pool-info*:= (*mem-pool-info V*) (*pool b* := *minf* ⦇*levels* := (*levels minf*)

[*lvl V t* := ((*levels minf*) ! (*lvl V t*)) ⦇*bits* := (*bits* ((*levels minf*) ! (*lvl V t*))) [*bn V t* := *FREEING*]⦈] ⦈)⦈

lemma *mempool-free-stm8-atombody-rest-one-finalstm-inv-cur*:
V∈*mp-free-precond8-3 t b α* ∩ ⦃*´cur* = *Some t*⦄ ⟹
(∗ {*free-stm8-precond2 V t b*} ∩ ⦃*NULL* < *´lvl t* ∧ *partner-bits* (*´mem-pool-info* (*pool b*)) (*´lvl t*) (*´bn t*)⦄ ≠ {} ⟹ ∗)
V2 ∈ *free-stm8-precond3 V t b* ∩ ⦃*´i t* = *4*⦄ ⟹
x = *free-stm8-atombody-rest-cond3* (*V2*⦇*lvl* := (*lvl V2*)(*t* := *lvl V2 t* − *1*), *bn*

:= (*bn V2*)(*t* := *bn V2 t div 4*)|)) *t b* ⟹
  y = *x*(|*freeing-node* := (*freeing-node x*) (*t* := *Some* (|*pool* = *pool b*, *level* = *lvl x t*, *block* = *bn x t*,
            data = *block-ptr* (*mem-pool-info x* (*pool b*)) (*ALIGN4* (*max-sz* (*mem-pool-info x* (*pool b*))) *div 4* ^ *lvl x t*) (*bn x t*)|))|) ⟹
  inv-cur y
apply(*rule subst*[**where** *s*=*inv-cur x* **and** *t*=*inv-cur y*])
apply(*simp add*:*block-ptr-def inv-cur-def*)

apply(*simp add*:*Let-def inv-def inv-cur-def*)
apply(*subgoal-tac thd-state V t* = *RUNNING*) **prefer** *2* **apply** *fast*
apply *clarsimp*
done

lemma *mempool-free-stm8-atombody-rest-one-finalstm-inv-thd-waitq*:
  V∈*mp-free-precond8-3 t b α* ∩ {|´*cur* = *Some t*|} ⟹
  (∗ {*free-stm8-precond2 V t b*} ∩ {|*NULL* < ´*lvl t* ∧ *partner-bits* (´*mem-pool-info* (*pool b*)) (´*lvl t*) (´*bn t*)|} ≠ {} ⟹ ∗)
  V2 ∈ *free-stm8-precond3 V t b* ∩ {|´*i t* = *4*|} ⟹
  x = *free-stm8-atombody-rest-cond3* (*V2*(|*lvl* := (*lvl V2*)(*t* := *lvl V2 t* − *1*), *bn* := (*bn V2*)(*t* := *bn V2 t div 4*)|)) *t b* ⟹
  y = *x*(|*freeing-node* := (*freeing-node x*) (*t* := *Some* (|*pool* = *pool b*, *level* = *lvl x t*, *block* = *bn x t*,
            data = *block-ptr* (*mem-pool-info x* (*pool b*)) (*ALIGN4* (*max-sz* (*mem-pool-info x* (*pool b*))) *div 4* ^ *lvl x t*) (*bn x t*)|))|) ⟹
  inv-thd-waitq y
apply(*rule subst*[**where** *s*=*inv-thd-waitq x* **and** *t*=*inv-thd-waitq y*])
apply(*simp add*:*block-ptr-def inv-thd-waitq-def*)

apply(*simp add*:*Let-def inv-def inv-thd-waitq-def*)
apply(*simp add*:*set-bit-def*)
apply(*subgoal-tac mem-pools V* = *mem-pools V2*)
  prefer *2* **apply**(*simp add*:*gvars-conf-stable-def gvars-conf-def*)

apply(*rule conjI*)
  apply *clarify* **apply** *metis*

apply(*rule conjI*)
  apply *clarify* **apply** *metis*

apply(*rule conjI*)
  apply *clarify* **apply** *metis*
apply *metis*
done

lemma *mempool-free-stm8-atombody-rest-one-finalstm-inv-mempool-info-h1*:
∀ *p*. *buf* (*mem-pool-info V2 p*) =
      buf (*if p* = *pool b*

then mem-pool-info V (pool b)
(|levels := levels (mem-pool-info V (pool b))
[lvl V t := (levels (mem-pool-info V (pool b)) ! lvl V t)(|bits := bits (levels (mem-pool-info V (pool b)) ! lvl V t)[block b div 4 ^ (level b − lvl V t) := FREE]|)]|)
else mem-pool-info V p) ∧
max-sz (mem-pool-info V2 p) =
max-sz (if p = pool b
then mem-pool-info V (pool b)
(|levels := levels (mem-pool-info V (pool b))
[lvl V t := (levels (mem-pool-info V (pool b)) ! lvl V t)
(|bits := bits (levels (mem-pool-info V (pool b)) ! lvl V t)[block b div 4 ^ (level b − lvl V t) := FREE]|)]|)
else mem-pool-info V p) ∧
n-max (mem-pool-info V2 p) =
n-max (if p = pool b
then mem-pool-info V (pool b)
(|levels := levels (mem-pool-info V (pool b))
[lvl V t := (levels (mem-pool-info V (pool b)) ! lvl V t)
(|bits := bits (levels (mem-pool-info V (pool b)) ! lvl V t)[block b div 4 ^ (level b − lvl V t) := FREE]|)]|)
else mem-pool-info V p) ∧
n-levels (mem-pool-info V2 p) =
n-levels (if p = pool b
then mem-pool-info V (pool b)
(|levels := levels (mem-pool-info V (pool b))
[lvl V t := (levels (mem-pool-info V (pool b)) ! lvl V t)
(|bits := bits (levels (mem-pool-info V (pool b)) ! lvl V t)[block b div 4 ^ (level b − lvl V t) := FREE]|)]|)
else mem-pool-info V p) ∧
length (levels (mem-pool-info V2 p)) =
length (levels (if p = pool b
then mem-pool-info V (pool b)
(|levels := levels (mem-pool-info V (pool b))
[lvl V t := (levels (mem-pool-info V (pool b)) ! lvl V t)
(|bits := bits (levels (mem-pool-info V (pool b)) ! lvl V t)[block b div 4 ^ (level b − lvl V t) := FREE]|)]|)
else mem-pool-info V p)) ∧
(∀ i. length (bits (levels (mem-pool-info V2 p) ! i)) =
length (bits (levels (if p = pool b
then mem-pool-info V (pool b)
(|levels := levels (mem-pool-info V (pool b))
[lvl V t := (levels (mem-pool-info V (pool b)) ! lvl V t)
(|bits := bits (levels (mem-pool-info V (pool b)) ! lvl V t)[block b div 4 ^ (level b − lvl V t) := FREE]|)]|)
else mem-pool-info V p) !
i))) ⟹
ia < length (levels (mem-pool-info V (pool b))) ⟹

length (*bits* (*levels* (*mem-pool-info* *V* (*pool* *b*)) ! *ia*)) = *length* (*bits* (*levels* (*mem-pool-info* *V2* (*pool* *b*)) ! *ia*))
apply *auto*
apply(*case-tac lvl V t = ia*) **apply** *auto*[*1*] **apply** *auto*[*1*]
done

lemma *mempool-free-stm8-atombody-rest-one-finalstm-inv-mempool-info-h2*:
ia < *length* (*levels* (*mem-pool-info* *V* (*pool* *b*))) $\Longrightarrow$
length (*bits* (*levels* (*mem-pool-info* *V2* (*pool* *b*))
[*lvl V2 t* − *Suc NULL* := (*levels* (*mem-pool-info* *V2* (*pool* *b*)) ! (*lvl V2 t* − *Suc NULL*))
(|*bits* := *bits* (*levels* (*mem-pool-info* *V2* (*pool* *b*)) ! (*lvl V2 t* − *Suc NULL*))[*bn V2 t div 4* := *FREEING*]|)] !
ia)) = *length* (*bits* (*levels* (*mem-pool-info* *V2* (*pool* *b*)) ! *ia*))
apply(*case-tac lvl V2 t* − *Suc NULL* = *ia*)
apply(*case-tac ia* < *length* (*levels* (*mem-pool-info* *V2* (*pool* *b*)))) **apply** *auto*
done

lemma *mempool-free-stm8-atombody-rest-one-finalstm-inv-mempool-info-h3*:
mem-pools V = *mem-pools V2* $\Longrightarrow$
p ∈ *mem-pools V2* $\Longrightarrow$
∀ *p*∈*mem-pools V2*.
NULL < *buf* (*mem-pool-info V p*) ∧
(∃ *n*>*NULL*. *max-sz* (*mem-pool-info V p*) = *4* ∗ *n* ∗ *4* ^ *n-levels* (*mem-pool-info V p*)) ∧
NULL < *n-max* (*mem-pool-info V p*) ∧
NULL < *n-levels* (*mem-pool-info V p*) ∧
n-levels (*mem-pool-info V p*) = *length* (*levels* (*mem-pool-info V p*)) ∧
(∀ *i*<*length* (*levels* (*mem-pool-info V p*)). *length* (*bits* (*levels* (*mem-pool-info V p*) ! *i*)) = *n-max* (*mem-pool-info V p*) ∗ *4* ^ *i*) $\Longrightarrow$
mem-pools V = *mem-pools V2* $\Longrightarrow$
pool b ∈ *mem-pools V2* $\Longrightarrow$ *levels* (*mem-pool-info V p*) ≠ []
apply *auto*
done

lemma *mempool-free-stm8-atombody-rest-one-finalstm-h1-1′*:
∀ *j*. *j* ≠ *lvl V t* ⟶
levels (*mem-pool-info V* (*pool b*))
[*lvl V t* := (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)
(|*bits* := *bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)[*block b div 4* ^ (*level b* − *lvl V t*) := *FREE*]|)] !
j =
levels (*mem-pool-info V2* (*pool b*)) ! *j* $\Longrightarrow$
bits (*levels* (*mem-pool-info V2* (*pool b*)) ! *lvl V t*) =
list-updates-n
(*bits* (*levels* (*mem-pool-info V* (*pool b*))
[*lvl V t* := (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)
(|*bits* := *bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)[*block b div 4* ^

(*level b* − *lvl V t*) := *FREE*]⦈] !
lvl V t))
(*block b div 4* ˆ (*level b* − *lvl V t*) *div 4* ∗ *4*) *4 NOEXIST* ⟹
length (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *ia*)) =
length (*bits* (*levels* (*mem-pool-info V2* (*pool b*))
[*lvl V2 t* − *Suc NULL* := (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))
⦇*bits* := *bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))[*bn V2 t div 4* := *FREEING*]⦈] !
ia))
apply(*rule subst*[**where** *s*=*length* (*bits* (*levels* (*mem-pool-info V2* (*pool b*))!*ia*))
and *t*=*length* (*bits* (*levels* (*mem-pool-info V2* (*pool b*))
[*lvl V2 t* − *Suc 0* := (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc 0*))
⦇*bits* := *bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc 0*))[*bn V2 t div 4* := *FREEING*]⦈] !
ia))])
apply(*case-tac ia* = *lvl V2 t* − *Suc 0*)
apply(*case-tac ia* < *length* (*levels* (*mem-pool-info V2* (*pool b*))))
apply *auto*[*1*] **apply** *auto*[*1*] **apply** *auto*[*1*]

apply(*case-tac ia* = *lvl V t*)
apply(*subgoal-tac length* (*list-updates-n*
(*bits* (*levels* (*mem-pool-info V* (*pool b*))
[*lvl V t* := (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)
⦇*bits* := *bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)[*block b div 4* ˆ (*level b* − *lvl V t*) := *FREE*]⦈] !
ia))
(*block b div 4* ˆ (*level b* − *lvl V t*) *div 4* ∗ *4*) *4 NOEXIST*) = *length* (*bits* (*levels* (*mem-pool-info V* (*pool b*))
[*lvl V t* := (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)
⦇*bits* := *bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)[*block b div 4* ˆ (*level b* − *lvl V t*) := *FREE*]⦈] !
ia)))
prefer *2* **using** *length-list-update-n* **apply** *fast*
apply(*subgoal-tac length* (*bits* (*levels* (*mem-pool-info V* (*pool b*))
[*lvl V t* := (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)
⦇*bits* := *bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)[*block b div 4* ˆ (*level b* − *lvl V t*) := *FREE*]⦈] !
ia)) = *length* (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *ia*)))
prefer *2* **apply**(*case-tac ia* = *lvl V t*)
apply(*case-tac ia* < *length* (*levels* (*mem-pool-info V* (*pool b*))))
apply *auto*[*1*] **apply** *auto*[*1*] **apply** *auto*[*1*]
apply *auto*[*1*]

apply(*subgoal-tac length* (*bits* (*levels* (*mem-pool-info V* (*pool b*))
[*lvl V t* := (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)
⦇*bits* := *bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)[*block b div 4* ˆ (*level*

$b - lvl\ V\ t) := FREE]\rparr] !$
$ia)) = length\ (bits\ (levels\ (mem\text{-}pool\text{-}info\ V\ (pool\ b))\ !\ ia)))$
prefer *2* **apply**(*case-tac ia = lvl V t*) **apply**(*case-tac ia < length (levels (mem-pool-info V (pool b))))*
apply *auto*[*1*] **apply** *auto*[*1*] **apply** *auto*[*1*]
apply(*subgoal-tac levels (mem-pool-info V (pool b))*
[lvl V t := (levels (mem-pool-info V (pool b)) ! lvl V t)
(|bits := bits (levels (mem-pool-info V (pool b)) ! lvl V t)[block b div 4 ^ (level b − lvl V t) := FREE]|)] !
ia = levels (mem-pool-info V2 (pool b)) ! ia)
prefer *2* **apply** *fast*
apply *auto*
done

lemma *mempool-free-stm8-atombody-rest-one-finalstm-h1-1*:
$\forall j.\ j \neq lvl\ V\ t \longrightarrow$
levels (set-bit-free (mem-pool-info V) (pool b) (lvl V t) (block b div 4 ^ (level b − lvl V t)) (pool b)) ! j
$=$ *levels (mem-pool-info V2 (pool b)) ! j* $\Longrightarrow$
bits (levels (mem-pool-info V2 (pool b)) ! lvl V t) =
list-updates-n (bits (levels (set-bit-free (mem-pool-info V) (pool b) (lvl V t)
(block b div 4 ^ (level b − lvl V t)) (pool b)) ! lvl V t))
(block b div 4 ^ (level b − lvl V t) div 4 ∗ 4) 4 NOEXIST $\Longrightarrow$
length (bits (levels (mem-pool-info V (pool b)) ! ia)) =
length (bits (levels (mem-pool-info V2 (pool b))
[lvl V2 t − Suc NULL := (levels (mem-pool-info V2 (pool b)) ! (lvl V2 t − Suc NULL))
(|bits := bits (levels (mem-pool-info V2 (pool b)) ! (lvl V2 t − Suc NULL))[bn V2 t div 4 := FREEING]|)] !
ia))
apply(*rule subst*[**where** *s=length (bits (levels (mem-pool-info V2 (pool b))!ia))* **and** *t=length (bits (levels (mem-pool-info V2 (pool b))*
[lvl V2 t − Suc 0 := (levels (mem-pool-info V2 (pool b)) ! (lvl V2 t − Suc 0))
(|bits := bits (levels (mem-pool-info V2 (pool b)) ! (lvl V2 t − Suc 0))[bn V2 t div 4 := FREEING]|)] !
ia))])
apply(*case-tac ia = lvl V2 t − Suc 0*)
apply(*case-tac ia < length (levels (mem-pool-info V2 (pool b))))*
apply *auto*[*1*] **apply** *auto*[*1*] **apply** *auto*[*1*]

apply(*simp add:set-bit-def*)
apply(*case-tac ia = lvl V t*)
apply(*subgoal-tac length (list-updates-n*
(bits (levels (mem-pool-info V (pool b))
[lvl V t := (levels (mem-pool-info V (pool b)) ! lvl V t)
(|bits := bits (levels (mem-pool-info V (pool b)) ! lvl V t)[block b div 4 ^ (level b − lvl V t) := FREE]|)] !
ia))

(block b div 4 ^ (level b − lvl V t) div 4 ∗ 4) 4 NOEXIST) = length (bits (levels (mem-pool-info V (pool b))
[lvl V t := (levels (mem-pool-info V (pool b)) ! lvl V t)
(|bits := bits (levels (mem-pool-info V (pool b)) ! lvl V t)[block b div 4 ^ (level b − lvl V t) := FREE]|)] !
ia)))
prefer 2 **using** length-list-update-n **apply** fast
apply(subgoal-tac length (bits (levels (mem-pool-info V (pool b))
[lvl V t := (levels (mem-pool-info V (pool b)) ! lvl V t)
(|bits := bits (levels (mem-pool-info V (pool b)) ! lvl V t)[block b div 4 ^ (level b − lvl V t) := FREE]|)] !
ia)) = length (bits (levels (mem-pool-info V (pool b)) ! ia)))
prefer 2 **apply**(case-tac ia = lvl V t)
apply(case-tac ia < length (levels (mem-pool-info V (pool b))))
apply auto[1] **apply** auto[1] **apply** auto[1]
apply auto[1]

apply(subgoal-tac length (bits (levels (mem-pool-info V (pool b))
[lvl V t := (levels (mem-pool-info V (pool b)) ! lvl V t)
(|bits := bits (levels (mem-pool-info V (pool b)) ! lvl V t)[block b div 4 ^ (level b − lvl V t) := FREE]|)] !
ia)) = length (bits (levels (mem-pool-info V (pool b)) ! ia)))
prefer 2 **apply**(case-tac ia = lvl V t) **apply**(case-tac ia < length (levels (mem-pool-info V (pool b))))
apply auto[1] **apply** auto[1] **apply** auto[1]
apply(subgoal-tac levels (mem-pool-info V (pool b))
[lvl V t := (levels (mem-pool-info V (pool b)) ! lvl V t)
(|bits := bits (levels (mem-pool-info V (pool b)) ! lvl V t)[block b div 4 ^ (level b − lvl V t) := FREE]|)] !
ia = levels (mem-pool-info V2 (pool b)) ! ia)
prefer 2 **apply** auto[1]
apply auto
done

lemma mempool-free-stm8-atombody-rest-one-finalstm-inv-mempool-info:
V∈mp-free-precond8-3 t b α ∩ {|´cur = Some t|} ⟹
(∗ {free-stm8-precond2 V t b} ∩ {|NULL < ´lvl t ∧ partner-bits (´mem-pool-info (pool b)) (´lvl t) (´bn t)|} ≠ {} ⟹ ∗)
V2 ∈ free-stm8-precond3 V t b ∩ {|´i t = 4|} ⟹
x = free-stm8-atombody-rest-cond3 (V2(|lvl := (lvl V2)(t := lvl V2 t − 1), bn := (bn V2)(t := bn V2 t div 4)|)) t b ⟹
y = x(|freeing-node := (freeing-node x) (t := Some (|pool = pool b, level = lvl x t, block = bn x t,
data = block-ptr (mem-pool-info x (pool b)) (ALIGN4 (max-sz (mem-pool-info x (pool b))) div 4 ^ lvl x t) (bn x t)|))|) ⟹
inv-mempool-info y
apply(rule subst[**where** s=inv-mempool-info x **and** t=inv-mempool-info y])
apply(simp add:block-ptr-def inv-mempool-info-def)

apply(*simp add*:*Let-def inv-def inv-mempool-info-def*)
apply(*simp add*:*set-bit-def*)
apply(*simp add*:*gvars-conf-stable-def gvars-conf-def*)

apply(*subgoal-tac mem-pools V* = *mem-pools V2*)
prefer *2* **apply** *simp*

apply *clarify*
apply(*rule conjI*) **apply** *clarify*
apply(*rule conjI*) **apply** *metis*
apply(*rule conjI*)
apply(*subgoal-tac length* (*levels* (*mem-pool-info V* (*pool b*))) > *0*) **prefer** *2*
apply *metis* **apply** *fast*
apply *clarify*
apply(*subgoal-tac length* (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *ia*))
= *length* (*bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! *ia*)))
prefer *2* **using** *mempool-free-stm8-atombody-rest-one-finalstm-inv-mempool-info-h1*
apply *blast*
apply(*subgoal-tac length* (*bits* (*levels* (*mem-pool-info V2* (*pool b*))
[*lvl V2 t* − *Suc NULL* := (*levels* (*mem-pool-info V2* (*pool b*)) !
(*lvl V2 t* − *Suc NULL*))
(|*bits* := *bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* −
Suc NULL))
[*bn V2 t div 4* := *FREEING*]|)] !
ia)) = *length* (*bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! *ia*)))
prefer *2* **using** *mempool-free-stm8-atombody-rest-one-finalstm-inv-mempool-info-h2*
apply *blast*
apply *metis*

apply *clarify*
apply(*rule conjI*)
apply *metis*
using *mempool-free-stm8-atombody-rest-one-finalstm-inv-mempool-info-h3* **apply**
blast
done

lemma *free-stm8-atombody-rest-one-finalstm-VV2-len*:
∀ *p*. *length* (*levels* (*mem-pool-info V2 p*)) =
length (*levels* (*if p* = *pool b*
then mem-pool-info V (*pool b*)
(|*levels* := *levels* (*mem-pool-info V* (*pool b*))
[*lvl V t* := (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)
(|*bits* := *bits* (*levels* (*mem-pool-info V* (*pool b*)) !
lvl V t)[*block b div 4* ˆ (*level b* − *lvl V t*) := *FREE*]|)]|)
else mem-pool-info V p)) =⇒
length (*levels* (*mem-pool-info V p*)) = *length* (*levels* (*mem-pool-info V2 p*))
by *auto*

lemma *free-stm8-atombody-rest-one-finalstm-bits-len*:
lvl V t = lvl V2 t $\Longrightarrow$
p = pool b $\Longrightarrow$
length (*bits* (*levels* (*if p = pool b*
then mem-pool-info V (*pool b*)
(|*levels* := *levels* (*mem-pool-info V* (*pool b*))
[*lvl V t* := (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)
(|*bits* := *bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)
[*block b div 4* ^ (*level b* − *lvl V t*) := *FREE*]|)]|)
else mem-pool-info V p) !
(*lvl V2 t*))) = *length* (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V2 t*))
apply(*case-tac lvl V2 t* < *length* (*levels* (*mem-pool-info V* (*pool b*))))
apply *auto*
done

lemma *free-stm8-atombody-rest-one-finalstm-ltlen*:
lvl V2 t > *0* $\Longrightarrow$
lvl V2 t = lvl V t $\Longrightarrow$
length (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V2 t*))
= (*n-max* (*mem-pool-info V* (*pool b*))) ∗ *4* ^ *lvl V2 t* $\Longrightarrow$
block b div 4 ^ (*level b* − *lvl V2 t*) < *length* (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V2 t*)) $\Longrightarrow$
block b div 4 ^ (*level b* − *lvl V2 t*) *div 4* ∗ *4* + *4*
≤ *length* (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V2 t*))
apply(*rule divn-multn-addn-le*[*of 4 length* (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V2 t*))
block b div 4 ^ (*level b* − *lvl V2 t*)])
apply *simp* **apply** *simp* **apply** *simp*
done

lemma *free-stm8-atombody-rest-one-finalstm-jj*:
lvl V2 t > *0* $\Longrightarrow$
lvl V2 t = lvl V t $\Longrightarrow$
length (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V2 t*))
= (*n-max* (*mem-pool-info V* (*pool b*))) ∗ *4* ^ *lvl V2 t* $\Longrightarrow$
block b div 4 ^ (*level b* − *lvl V2 t*) < *length* (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V2 t*)) $\Longrightarrow$
jj ∈ {*block b div 4* ^ (*level b* − *lvl V t*) *div 4* ∗ *4* ..<
block b div 4 ^ (*level b* − *lvl V t*) *div 4* ∗ *4* + *4*} $\Longrightarrow$
jj < *length* (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V2 t*))
apply *clarsimp*
apply(*subgoal-tac block b div 4* ^ (*level b* − *lvl V2 t*) *div 4* ∗ *4* + *4*
≤ *length* (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V2 t*)))
prefer *2*
apply(*rule free-stm8-atombody-rest-one-finalstm-ltlen*)

apply *simp+*
done

lemma *mempool-free-stm8-atombody-rest-one-finalstm-inv-bitmap'-h1*:
bits (*levels* (*mem-pool-info V2* (*pool b*)) ! *lvl V t*) =
  list-updates-n
   (*bits* (*levels* (*mem-pool-info V* (*pool b*))
        [*lvl V t* := (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)
          (|*bits* := *bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)[*block b div 4* ˆ
(*level b* − *lvl V t*) := *FREE*]|)] !
        lvl V t))
   (*block b div 4* ˆ (*level b* − *lvl V t*) *div 4* ∗ *4*) (*i V2 t*) *NOEXIST* =⇒
i V2 t = *4* =⇒
level b < *length* (*levels* (*mem-pool-info V* (*pool b*))) =⇒
lvl V t = *lvl V2 t* =⇒
lvl V2 t ≤ *level b* =⇒
ia = *lvl V t* =⇒
ia > *0* =⇒
p = *pool b* =⇒
block b div 4 ˆ (*level b* − *lvl V2 t*) < *length* (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V2 t*)) =⇒
length (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V2 t*))
   = (*n-max* (*mem-pool-info V* (*pool b*))) ∗ *4* ˆ *lvl V2 t* =⇒
length (*levels* (*mem-pool-info V* (*pool b*)))=*length* (*levels* (*mem-pool-info V2* (*pool b*))) =⇒
jj ∈ {*block b div 4* ˆ (*level b* − *lvl V t*) *div 4* ∗ *4* ..<
     block b div 4 ˆ (*level b* − *lvl V t*) *div 4* ∗ *4* + *4*} =⇒
get-bit-s (*V2*(|*mem-pool-info* := (*mem-pool-info V2*)
      (*pool b* := *mem-pool-info V2* (*pool b*)
        (|*levels* := *levels* (*mem-pool-info V2* (*pool b*))
          [*lvl V2 t* − *Suc NULL* := (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))
            (|*bits* := *bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))
             [*bn V2 t div 4* := *FREEING*]|)]|))|)) *p ia jj* = *NOEXIST*
apply(*rule subst*[**where** *s*=*get-bit-s V2 p ia jj*])
apply(*subgoal-tac list-updates-n*
   (*bits* (*levels* (*mem-pool-info V* (*pool b*))
        [*lvl V t* := (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)
          (|*bits* := *bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)[*block b div 4* ˆ
(*level b* − *lvl V t*) := *FREE*]|)] !
        lvl V t))
   (*block b div 4* ˆ (*level b* − *lvl V t*) *div 4* ∗ *4*) (*i V2 t*) *NOEXIST* !
  jj =
  NOEXIST) **prefer** *2*
  apply(*rule list-updates-n-eq*[*of block b div 4* ˆ (*level b* − *lvl V t*) *div 4* ∗ *4 jj*
     bits (*levels* (*mem-pool-info V* (*pool b*))
        [*lvl V t* := (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)
          (|*bits* := *bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)[*block b div*

4 ^ (*level b* − *lvl V t*) := *FREE*]⦈] !
lvl V t) *i V2 t NOEXIST*])
apply *fastforce*
apply(*rule subst*[**where** *s*=*length* (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*))
and *t*=*length* (*bits* (*levels* (*mem-pool-info V* (*pool b*))
[*lvl V t* := (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)
(⦇*bits* := *bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)
[*block b div 4* ^ (*level b* − *lvl V t*) := *FREE*]⦈] !
lvl V t))]) **apply** *force*
apply(*rule free-stm8-atombody-rest-one-finalstm-jj*)
apply *fast* **apply** *fast* **apply** *presburger* **apply** *argo*
apply *fast* **apply** *force*

apply *argo*
by *fastforce*

lemma *mempool-free-stm8-atombody-rest-one-finalstm-inv-bitmap′-h2*:
get-bit-s
(*V2*⦇*mem-pool-info* := (*mem-pool-info V2*)
(*pool b* := *mem-pool-info V2* (*pool b*)
⦇*levels* := *levels* (*mem-pool-info V2* (*pool b*))
[*lvl V2 t* − *Suc NULL* := (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))
⦇*bits* := *bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))
[*bn V2 t div 4* := *FREEING*]⦈]⦈)⦈)
p ia jj =
FREE ∨
get-bit-s
(*V2*⦇*mem-pool-info* := (*mem-pool-info V2*)
(*pool b* := *mem-pool-info V2* (*pool b*)
⦇*levels* := *levels* (*mem-pool-info V2* (*pool b*))
[*lvl V2 t* − *Suc NULL* := (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))
⦇*bits* := *bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))
[*bn V2 t div 4* := *FREEING*]⦈]⦈)⦈)
p ia jj =
FREEING ∨
get-bit-s
(*V2*⦇*mem-pool-info* := (*mem-pool-info V2*)
(*pool b* := *mem-pool-info V2* (*pool b*)
⦇*levels* := *levels* (*mem-pool-info V2* (*pool b*))
[*lvl V2 t* − *Suc NULL* := (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))
⦇*bits* := *bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))

$[bn\ V2\ t\ div\ 4 := FREEING]\|)]\|))\|))$
$p\ ia\ jj =$
$ALLOCATED \vee$
$get\text{-}bit\text{-}s$
$(V2(\|mem\text{-}pool\text{-}info := (mem\text{-}pool\text{-}info\ V2)$
$(pool\ b := mem\text{-}pool\text{-}info\ V2\ (pool\ b)$
$(\|levels := levels\ (mem\text{-}pool\text{-}info\ V2\ (pool\ b))$
$[lvl\ V2\ t - Suc\ NULL := (levels\ (mem\text{-}pool\text{-}info\ V2\ (pool\ b))\ !\ (lvl\ V2\ t - Suc\ NULL))$
$(\|bits := bits\ (levels\ (mem\text{-}pool\text{-}info\ V2\ (pool\ b))\ !\ (lvl\ V2\ t - Suc\ NULL))$
$[bn\ V2\ t\ div\ 4 := FREEING]\|)]\|))\|))$
$p\ ia\ jj =$
$ALLOCATING \Longrightarrow$
$get\text{-}bit\text{-}s$
$(V2(\|mem\text{-}pool\text{-}info := (mem\text{-}pool\text{-}info\ V2)$
$(pool\ b := mem\text{-}pool\text{-}info\ V2\ (pool\ b)$
$(\|levels := levels\ (mem\text{-}pool\text{-}info\ V2\ (pool\ b))$
$[lvl\ V2\ t - Suc\ NULL := (levels\ (mem\text{-}pool\text{-}info\ V2\ (pool\ b))\ !\ (lvl\ V2\ t - Suc\ NULL))$
$(\|bits := bits\ (levels\ (mem\text{-}pool\text{-}info\ V2\ (pool\ b))\ !\ (lvl\ V2\ t - Suc\ NULL))$
$[bn\ V2\ t\ div\ 4 := FREEING]\|)]\|))\|))$
$p\ ia\ jj =$
$NOEXIST \Longrightarrow$
$get\text{-}bit\text{-}s$
$(V2(\|mem\text{-}pool\text{-}info := (mem\text{-}pool\text{-}info\ V2)$
$(pool\ b := mem\text{-}pool\text{-}info\ V2\ (pool\ b)$
$(\|levels := levels\ (mem\text{-}pool\text{-}info\ V2\ (pool\ b))$
$[lvl\ V2\ t - Suc\ NULL := (levels\ (mem\text{-}pool\text{-}info\ V2\ (pool\ b))\ !\ (lvl\ V2\ t - Suc\ NULL))$
$(\|bits := bits\ (levels\ (mem\text{-}pool\text{-}info\ V2\ (pool\ b))\ !\ (lvl\ V2\ t - Suc\ NULL))$
$[bn\ V2\ t\ div\ 4 := FREEING]\|)]\|))\|))$
$p\ (ia - 1)\ (jj\ div\ 4) =$
$DIVIDED$
by *force*

axiomatization where *mempool-free-stm8-atombody-rest-one-finalstm-inv-bitmap*:
$V \in mp\text{-}free\text{-}precond8\text{-}3\ t\ b\ \alpha \cap \{\!|\ ´cur = Some\ t|\!\} \Longrightarrow$
$(*\ \{free\text{-}stm8\text{-}precond2\ V\ t\ b\} \cap \{\!|NULL < ´lvl\ t \wedge partner\text{-}bits\ (´mem\text{-}pool\text{-}info\ (pool\ b))\ (´lvl\ t)\ (´bn\ t)|\!\} \neq \{\} \Longrightarrow *)$
$V2 \in free\text{-}stm8\text{-}precond3\ V\ t\ b \cap \{\!|´i\ t = 4|\!\} \Longrightarrow$
$x = free\text{-}stm8\text{-}atombody\text{-}rest\text{-}cond3\ (V2(\|lvl := (lvl\ V2)(t := lvl\ V2\ t - 1),\ bn := (bn\ V2)(t := bn\ V2\ t\ div\ 4)\|))\ t\ b \Longrightarrow$
$y = x(\|freeing\text{-}node := (freeing\text{-}node\ x)\ (t := Some\ (\|pool = pool\ b,\ level = lvl\ x\ t,\ block = bn\ x\ t,$
$data = block\text{-}ptr\ (mem\text{-}pool\text{-}info\ x\ (pool\ b))\ (ALIGN4\ (max\text{-}sz$

$(mem\text{-}pool\text{-}info\ x\ (pool\ b)))\ div\ 4\ \hat{}\ lvl\ x\ t)\ (bn\ x\ t)|)|)|) \Longrightarrow$
  inv-bitmap y

axiomatization where *mempool-free-stm8-atombody-rest-one-finalstm-inv-bitmap-freelist*:
  $V \in mp\text{-}free\text{-}precond8\text{-}3\ t\ b\ \alpha \cap \{|\acute{}cur = Some\ t|\} \Longrightarrow$
  $(*\ \{free\text{-}stm8\text{-}precond2\ V\ t\ b\} \cap \{|NULL < \acute{}lvl\ t \wedge partner\text{-}bits\ (\acute{}mem\text{-}pool\text{-}info\ (pool\ b))\ (\acute{}lvl\ t)\ (\acute{}bn\ t)|\} \neq \{\} \Longrightarrow *)$
  $V2 \in free\text{-}stm8\text{-}precond3\ V\ t\ b \cap \{|\acute{}i\ t = 4|\} \Longrightarrow$
  $x = free\text{-}stm8\text{-}atombody\text{-}rest\text{-}cond3\ (V2(|lvl := (lvl\ V2)(t := lvl\ V2\ t - 1),\ bn := (bn\ V2)(t := bn\ V2\ t\ div\ 4)|))\ t\ b \Longrightarrow$
  $y = x(|freeing\text{-}node := (freeing\text{-}node\ x)\ (t := Some\ (|pool = pool\ b,\ level = lvl\ x\ t,\ block = bn\ x\ t,$
      $data = block\text{-}ptr\ (mem\text{-}pool\text{-}info\ x\ (pool\ b))\ (ALIGN4\ (max\text{-}sz\ (mem\text{-}pool\text{-}info\ x\ (pool\ b)))\ div\ 4\ \hat{}\ lvl\ x\ t)\ (bn\ x\ t)|)|)|) \Longrightarrow$
  inv-bitmap-freelist y

lemma *mempool-free-stm8-atombody-rest-one-finalstm-len-bits1*:
$\forall j.\ j \neq lvl\ V\ t \longrightarrow levels\ (mem\text{-}pool\text{-}info\ V\ (pool\ b))\ !\ j = levels\ (mem\text{-}pool\text{-}info\ V2\ (pool\ b))\ !\ j \Longrightarrow$
 $bits\ (levels\ (mem\text{-}pool\text{-}info\ V2\ (pool\ b))\ !\ lvl\ V\ t) =$
 list-updates-n
  $(bits\ (levels\ (mem\text{-}pool\text{-}info\ V\ (pool\ b))$
    $[lvl\ V\ t := (levels\ (mem\text{-}pool\text{-}info\ V\ (pool\ b))\ !\ lvl\ V\ t)$
      $(|bits := bits\ (levels\ (mem\text{-}pool\text{-}info\ V\ (pool\ b))\ !\ lvl\ V\ t)[block\ b\ div\ 4\ \hat{}\ (level\ b - lvl\ V\ t) := FREE]|)]\ !$
    $lvl\ V\ t))$
  $(block\ b\ div\ 4\ \hat{}\ (level\ b - lvl\ V\ t)\ div\ 4 * 4)\ (i\ V2\ t)\ NOEXIST \Longrightarrow$
 $(i\ V2\ t) = 4 \Longrightarrow$
 $length\ (bits\ (levels\ (mem\text{-}pool\text{-}info\ V\ (pool\ b))\ !\ (lvl\ V2\ t - Suc\ NULL)))$
   $= length\ (bits\ (levels\ (mem\text{-}pool\text{-}info\ V2\ (pool\ b))\ !\ (lvl\ V2\ t - Suc\ NULL)))$
apply(*rule subst*[**where** $s{=}length\ (bits\ (levels\ (mem\text{-}pool\text{-}info\ V2\ (pool\ b))$
    $[lvl\ V2\ t - Suc\ NULL := (levels\ (mem\text{-}pool\text{-}info\ V2\ (pool\ b))\ !\ (lvl\ V2\ t - Suc\ NULL))$
      $(|bits := bits\ (levels\ (mem\text{-}pool\text{-}info\ V2\ (pool\ b))\ !\ (lvl\ V2\ t - Suc\ NULL))[bn\ V2\ t\ div\ 4 := FREEING]|)]\ !$
    $(lvl\ V2\ t - Suc\ NULL)))$ **and** $t{=}length\ (bits\ (levels\ (mem\text{-}pool\text{-}info\ V\ (pool\ b))\ !\ (lvl\ V2\ t - Suc\ NULL)))$])
 using *mempool-free-stm8-atombody-rest-one-finalstm-h1-1'*[*of V t b V2 lvl V2 t − Suc NULL*] **apply** *auto*[*1*]
apply(*case-tac* $lvl\ V2\ t - Suc\ NULL < length\ (levels\ (mem\text{-}pool\text{-}info\ V2\ (pool\ b)))$)
  apply *auto*
done

lemma *lm11*:
$lvl\ V2\ t \leq level\ b \wedge level\ b > 0 \wedge level\ b < length\ (levels\ (mem\text{-}pool\text{-}info\ V\ (pool$

b))) $\wedge$
  0 $<$ *lvl V2 t* $\Longrightarrow$
  block b $<$ *n-max* (*mem-pool-info V* (*pool b*)) $*$ *4* ^ *level b* $\Longrightarrow$
  block b div 4 ^ (*level b* $-$ *lvl V2 t*) $=$ *bn V2 t* $\Longrightarrow$
  bn V2 t div 4 $<$ *n-max* (*mem-pool-info V* (*pool b*)) $*$ *4* ^ (*lvl V2 t* $-$ *Suc NULL*)
apply(*rule subst*[**where** *s*=*block b div 4* ^ (*level b* $-$ *lvl V2 t*) *div 4* **and** *t*=*bn V2 t div 4*])
  apply *simp*

apply(*rule subst*[**where** *s*=*n-max* (*mem-pool-info V* (*pool b*)) $*$ *4* ^ *level b div 4* ^ (*level b* $-$ *lvl V2 t*) *div 4*
  and *t*=*n-max* (*mem-pool-info V* (*pool b*)) $*$ *4* ^ (*lvl V2 t* $-$ *Suc NULL*)])
 apply (*smt Groups.mult-ac*(*2*) *Groups.mult-ac*(*3*) *One-nat-def add-diff-cancel-left'* *div-mult-self1-is-m*
  le-Suc-ex power-add power-minus-mult zero-less-numeral zero-less-power)

apply(*subgoal-tac n-max* (*mem-pool-info V* (*pool b*)) $*$ *4* ^ *level b mod 4* ^ (*level b* $-$ *lvl V2 t*) $=$ *0*)
  prefer *2* **using** *mod-minus-0*[*of lvl V2 t level b n-max* (*mem-pool-info V* (*pool b*))] **apply** *fast*

apply(*subgoal-tac block b div 4* ^ (*level b* $-$ *lvl V2 t*) $<$ *n-max* (*mem-pool-info V* (*pool b*)) $*$ *4* ^ *level b div 4* ^ (*level b* $-$ *lvl V2 t*))
  prefer *2* **using** *mod-div-gt*[*of block b n-max* (*mem-pool-info V* (*pool b*)) $*$ *4* ^ *level b 4* ^ (*level b* $-$ *lvl V2 t*)]
  apply *fast*
apply(*subgoal-tac n-max* (*mem-pool-info V* (*pool b*)) $*$ *4* ^ *level b div 4* ^ (*level b* $-$ *lvl V2 t*) *mod 4* $=$ *0*)
  prefer *2* **using** *mod-minus-div-4*[*of lvl V2 t level b n-max* (*mem-pool-info V* (*pool b*))] **apply** *fast*

using *mod-div-gt*[*of block b div 4* ^ (*level b* $-$ *lvl V2 t*)
  n-max (*mem-pool-info V* (*pool b*)) $*$ *4* ^ *level b div 4* ^ (*level b* $-$ *lvl V2 t*) *4*]
apply *fast*
done

lemma *mempool-free-stm8-atombody-rest-one-finalstm-inv-aux-vars-h2*:
pool b $\in$ *mem-pools V* $\Longrightarrow$
 inv-mempool-info V $\Longrightarrow$
 level b $<$ *length* (*levels* (*mem-pool-info V* (*pool b*))) $\Longrightarrow$
 block b div 4 ^ (*level b* $-$ *lvl V t*) $=$ *bn V2 t* $\Longrightarrow$
 block b $<$ *length* (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *level b*)) $\Longrightarrow$
 lvl V t $=$ *lvl V2 t* $\Longrightarrow$
 lvl V2 t $\leq$ *level b* $\Longrightarrow$
 0 $<$ *lvl V2 t* $\Longrightarrow$
  length (*levels* (*mem-pool-info V* (*pool b*))) $=$ *length* (*levels* (*mem-pool-info V2* (*pool b*))) $\Longrightarrow$

length (*bits* (*levels* (*mem-pool-info* *V* (*pool* *b*)) ! (*lvl* *V2* *t* − *Suc* *NULL*)))
= *length* (*bits* (*levels* (*mem-pool-info* *V2* (*pool* *b*)) ! (*lvl* *V2* *t* − *Suc* *NULL*)))
$\Longrightarrow$
bits (*levels* (*mem-pool-info* *V2* (*pool* *b*))
[*lvl* *V2* *t* − *Suc* *NULL* := (*levels* (*mem-pool-info* *V2* (*pool* *b*)) ! (*lvl* *V2* *t* − *Suc* *NULL*))
(|*bits* := *bits* (*levels* (*mem-pool-info* *V2* (*pool* *b*)) ! (*lvl* *V2* *t* − *Suc* *NULL*))[*bn* *V2* *t* *div* *4* := *FREEING*]|)] !
(*lvl* *V2* *t* − *Suc* *NULL*)) !
(*bn* *V2* *t* *div* *4*) =
FREEING
apply(*subgoal-tac lvl V2 t* − *Suc 0* < *length* (*levels* (*mem-pool-info* *V2* (*pool* *b*))))
prefer *2* **apply** *auto*[*1*]
apply(*subgoal-tac bn V2 t div 4* < *length* (*bits* (*levels* (*mem-pool-info* *V2* (*pool* *b*)) ! (*lvl* *V2* *t* − *Suc* *NULL*))))
prefer *2*

apply(*subgoal-tac level b* > *0*) **prefer** *2* **apply** *auto*[*1*]
apply(*subgoal-tac n-max* (*mem-pool-info* *V* (*pool* *b*)) ∗ *4* ˆ (*lvl* *V2* *t* − *Suc* *NULL*)
= *length* (*bits* (*levels* (*mem-pool-info* *V* (*pool* *b*)) ! (*lvl* *V2* *t* − *Suc* *NULL*))))
prefer *2* **apply**(*simp add*:*inv-mempool-info-def Let-def*)
apply (*metis inv-mempool-info-def lm11*)
apply *auto*
done

lemma *mempool-free-stm8-atombody-rest-one-finalstm-len-lvls*:
(*V2*, *V*(|*mem-pool-info* := (*mem-pool-info* *V*)
(*pool* *b* := *mem-pool-info* *V* (*pool* *b*)
(|*levels* := *levels* (*mem-pool-info* *V* (*pool* *b*))
[*lvl* *V2* *t* := (*levels* (*mem-pool-info* *V* (*pool* *b*)) ! *lvl* *V2* *t*)
(|*bits* := *bits* (*levels* (*mem-pool-info* *V* (*pool* *b*)) ! *lvl* *V2* *t*)[*bn* *V2* *t* := *FREE*]|)]|)),
freeing-node := (*freeing-node* *V*)(*t* := *None*)|))
∈ *gvars-conf-stable* $\Longrightarrow$
length (*levels* (*mem-pool-info* *V2* (*pool* *b*))) = *length* (*levels* (*mem-pool-info* *V* (*pool* *b*)))
apply(*simp add*:*gvars-conf-stable-def gvars-conf-def*)
done

axiomatization where *mempool-free-stm8-atombody-rest-one-finalstm-inv-aux-vars*:
V∈*mp-free-precond8-3* *t* *b* α ∩ {|´*cur* = *Some* *t*|} $\Longrightarrow$
{*free-stm8-precond2* *V* *t* *b*} ∩ {|*NULL* < ´*lvl* *t* ∧ *partner-bits* (´*mem-pool-info* (*pool* *b*)) (´*lvl* *t*) (´*bn* *t*)|} ≠ {} $\Longrightarrow$
V2 ∈ *free-stm8-precond3* *V* *t* *b* ∩ {|´*i* *t* = *4*|} $\Longrightarrow$
x = *free-stm8-atombody-rest-cond3* (*V2*(|*lvl* := (*lvl* *V2*)(*t* := *lvl* *V2* *t* − *1*), *bn*

:= (*bn V2*)(*t* := *bn V2 t div 4*)⦈) *t b* ⟹
  y = *x*⦇*freeing-node* := (*freeing-node x*) (*t* := *Some* ⦇*pool* = *pool b*, *level* = *lvl x t*, *block* = *bn x t*,
                    data = *block-ptr* (*mem-pool-info x* (*pool b*)) (*ALIGN4* (*max-sz* (*mem-pool-info x* (*pool b*))) *div 4* ^ *lvl x t*) (*bn x t*)⦈)⦈ ⟹
  inv-aux-vars y

lemma *mempool-free-stm8-atombody-rest-one-finalstm-inv-lvl0-case1-h1*:
NULL < *length* (*levels* (*mem-pool-info V2* (*pool b*))) ⟹
  bits (*levels* (*mem-pool-info V2* (*pool b*)) ! *NULL*)[*ia* := *FREEING*] =
  bits (*levels* (*mem-pool-info V2* (*pool b*))
      [*NULL* := (*levels* (*mem-pool-info V2* (*pool b*)) ! *NULL*)
      ⦇*bits* := *bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! *NULL*)[*ia* := *FREEING*]⦈]
!
      NULL)
by *auto*

lemma *mempool-free-stm8-atombody-rest-one-finalstm-inv-lvl0-case1*:
pool b ∈ *mem-pools V2* ⟹
  inv V ⟹
  NULL < *lvl V2 t* ⟹
  pool b ∈ *mem-pools V* ⟹
  (∗(*V2*, *V*⦇*mem-pool-info* := *set-bit-free* (*mem-pool-info V*) (*pool b*) (*lvl V2 t*) (*bn V2 t*),
      freeing-node := (*freeing-node V*)(*t* := *None*)⦈)
  ∈ *gvars-conf-stable* ⟹
  ∀ *p*. *p* ≠ *pool b* ⟶ *mem-pool-info V2 p* = *set-bit-free* (*mem-pool-info V*) (*pool b*) (*lvl V2 t*) (*bn V2 t*) *p* ⟹∗)
  ∀ *j*. *j* ≠ *lvl V2 t* ⟶
     levels (*set-bit-free* (*mem-pool-info V*) (*pool b*) (*lvl V2 t*) (*bn V2 t*) (*pool b*)) ! *j* =
     levels (*mem-pool-info V2* (*pool b*)) ! *j* ⟹
  lvl V2 t ≤ *level b* ⟹
  lvl V t = *lvl V2 t* ⟹
  ∀ *i*<*length* (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *NULL*)). *get-bit-s V* (*pool b*) *NULL i* ≠ *NOEXIST* ⟹
  ia < *length* (*bits* (*levels* (*mem-pool-info V2* (*pool b*))
              [*lvl V2 t* − *Suc NULL* := (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))
               ⦇*bits* := *bits* (*levels* (*mem-pool-info V2* (*pool b*)) !
                (*lvl V2 t* − *Suc NULL*))[*bn V2 t div 4* := *FREEING*]⦈] !
              NULL)) ⟹
  bits (*levels* (*mem-pool-info V2* (*pool b*))
      [*lvl V2 t* − *Suc NULL* := (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))
          ⦇*bits* := *bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))[*bn V2 t div 4* := *FREEING*]⦈] !
      NULL) !

ia =
NOEXIST ⟹
False
apply(*simp add:set-bit-def*)
apply(*subgoal-tac levels* (*mem-pool-info V* (*pool b*)) ! (*lvl V2 t* − *1*)
= *levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *1*)) **prefer** *2*
apply(*subgoal-tac levels* (*mem-pool-info V* (*pool b*))
[*lvl V2 t* := (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V2 t*)
(|*bits* := *bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V2 t*)[*bn V2 t* := *FREE*]|)] ! (*lvl V2 t* − *1*)
= *levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *1*)) **prefer** *2* **apply** *auto*[*1*]
apply *auto*[*1*]
apply *auto*[*1*]

apply(*case-tac lvl V2 t* − *Suc 0* = *0*)
apply(*subgoal-tac length* (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *0*))
= *length* (*bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! *0*))) **prefer** *2*
apply *auto*[*1*]
apply(*subgoal-tac length* (*levels* (*mem-pool-info V2* (*pool b*)))>*0*) **prefer** *2* **apply** *auto*[*1*]
apply(*subgoal-tac ia* < *length* (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *0*)))
prefer *2*
apply(*rule subst*[**where** *s*=*length* (*bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! *NULL*)) **and**
t=*length* (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *NULL*))])
apply *auto*[*1*]
apply(*rule subst*[**where** *t*=*length* (*bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! *NULL*)) **and**
s=*length* (*bits* (*levels* (*mem-pool-info V2* (*pool b*))
[*lvl V2 t* − *Suc NULL* := (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))
(|*bits* := *bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))[*bn V2 t div 4* := *FREEING*]|)] !
NULL))])
apply *auto*[*1*]
apply *auto*[*1*]

apply(*case-tac ia* = *bn V2 t div 4*)
apply(*subgoal-tac bits* (*levels* (*mem-pool-info V2* (*pool b*))
[*lvl V2 t* − *Suc NULL* := (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))
(|*bits* := *bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))[*bn V2 t div 4* := *FREEING*]|)] !
NULL) ! *ia* = *FREEING*) **prefer** *2*
apply(*rule subst*[**where** *s*=*0* **and** *t*=*lvl V2 t* − *Suc 0*]) **apply** *metis*
apply(*rule subst*[**where** *s*=*ia* **and** *t*=*bn V2 t div 4*]) **apply** *metis*
apply(*rule subst*[**where** *s*=*bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! *NULL*)[*ia*

:= *FREEING*] **and**
t=bits (*levels* (*mem-pool-info V2* (*pool b*))
[*NULL* := (*levels* (*mem-pool-info V2* (*pool b*)) ! *NULL*)(|*bits* := *bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! *NULL*)
[*ia* := *FREEING*]|)] ! *NULL*)]) **apply** *auto*[*1*] **apply** *auto*[*1*]
apply (*metis BlockState.distinct*(*25*))

apply(*subgoal-tac bits* (*levels* (*mem-pool-info V2* (*pool b*))
[*lvl V2 t* − *Suc NULL* := (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))
(|*bits* := *bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))[*bn V2 t div 4* := *FREEING*]|)] !
NULL) ! *ia* ≠ *NOEXIST*) **prefer** *2*
apply(*rule subst*[**where** *s=0* **and** *t=lvl V2 t* − *Suc 0*]) **apply** *metis*
apply(*rule subst*[**where** *s=bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! *NULL*)[*bn V2 t div 4* := *FREEING*] **and**
t=bits (*levels* (*mem-pool-info V2* (*pool b*))
[*NULL* := (*levels* (*mem-pool-info V2* (*pool b*)) ! *NULL*)(|*bits* := *bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! *NULL*)
[*bn V2 t div 4* := *FREEING*]|)] ! *NULL*)]) **apply** *auto*[*1*] **apply** *auto*[*1*]
apply *fast*

apply(*subgoal-tac bits* (*levels* (*mem-pool-info V2* (*pool b*))
[*lvl V2 t* − *Suc NULL* := (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))
(|*bits* := *bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))[*bn V2 t div 4* := *FREEING*]|)] !
NULL) ! *ia* ≠ *NOEXIST*) **prefer** *2*
apply(*rule subst*[**where** *s=levels* (*mem-pool-info V2* (*pool b*)) ! *NULL* **and** *t=levels* (*mem-pool-info V2* (*pool b*))
[*lvl V2 t* − *Suc NULL* := (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))
(|*bits* := *bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))[*bn V2 t div 4* := *FREEING*]|)] !
NULL]) **apply** *simp*
apply(*subgoal-tac length* (*bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! *NULL*)) = *length* (*bits* (*levels* (*mem-pool-info V2* (*pool b*))
[*lvl V2 t* − *Suc NULL* := (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))
(|*bits* := *bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))[*bn V2 t div 4* := *FREEING*]|)] !
NULL))) **prefer** *2* **apply** *simp*
apply *presburger*
apply *fast*
done

lemma *mempool-free-stm8-atombody-rest-one-finalstm-inv-lvl0-case2*:
p ∈ *mem-pools V2* ⟹
  inv V ⟹
  NULL < *lvl V2 t* ⟹
  pool b ∈ *mem-pools V* ⟹
  (*V2*, *V*⦇*mem-pool-info* := *set-bit-free* (*mem-pool-info V*) (*pool b*) (*lvl V2 t*) (*bn V2 t*),
    freeing-node := (*freeing-node V*)(*t* := *None*)⦈)
    ∈ *gvars-conf-stable* ⟹
  ∀ *p*. *p* ≠ *pool b* ⟶ *mem-pool-info V2 p* = *set-bit-free* (*mem-pool-info V*) (*pool b*) (*lvl V2 t*) (*bn V2 t*) *p* ⟹
  level b < *length* (*lsizes V2 t*) ⟹
  p ≠ *pool b* ⟹
  ia < *length* (*bits* (*levels* (*set-bit-free* (*mem-pool-info V*) (*pool b*) (*lvl V2 t*) (*bn V2 t*) *p*) ! *NULL*)) ⟹
  get-bit (*set-bit-free* (*mem-pool-info V*) (*pool b*) (*lvl V2 t*) (*bn V2 t*)) *p NULL ia* = *NOEXIST* ⟹ *False*
apply(*simp add*:*set-bit-def*)
apply(*subgoal-tac* ∀ *i*<*length* (*bits* (*levels* (*mem-pool-info V p*) ! *0*)). (*bits* (*levels* (*mem-pool-info V p*) ! *0*)) ! *i* ≠ *NOEXIST*)
  prefer *2* **apply**(*subgoal-tac mem-pools V2* = *mem-pools V*) **prefer** *2* **apply**(*simp add*:*gvars-conf-stable-def gvars-conf-def*)
  apply(*simp add*:*inv-def inv-bitmap0-def Let-def*)
apply *auto*
done

lemma *mempool-free-stm8-atombody-rest-one-finalstm-inv-lvl0*:
  V∈*mp-free-precond8-3 t b α* ∩ ⦃´*cur* = *Some t*⦄ ⟹
   {*free-stm8-precond2 V t b*} ∩ ⦃*NULL* < ´*lvl t* ∧ *partner-bits* (´*mem-pool-info* (*pool b*)) (´*lvl t*) (´*bn t*)⦄ ≠ {} ⟹
  V2 ∈ *free-stm8-precond3 V t b* ∩ ⦃´*i t* = *4*⦄ ⟹
  x = *free-stm8-atombody-rest-cond3* (*V2*⦇*lvl* := (*lvl V2*)(*t* := *lvl V2 t* − *1*), *bn* := (*bn V2*)(*t* := *bn V2 t div 4*)⦈) *t b* ⟹
  y = *x*⦇*freeing-node* := (*freeing-node x*) (*t* := *Some* ⦇*pool* = *pool b*, *level* = *lvl x t*, *block* = *bn x t*,
    data = *block-ptr* (*mem-pool-info x* (*pool b*)) (*ALIGN4* (*max-sz* (*mem-pool-info x* (*pool b*))) *div 4* ˆ *lvl x t*) (*bn x t*)⦈)⦈ ⟹
  inv-bitmap0 y
apply(*simp add*:*inv-bitmap0-def Let-def*)
apply *clarify*

apply(*rule conjI*)
  apply *clarsimp*
  apply(*subgoal-tac* ∀ *i*<*length* (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *0*)).
    (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *0*)) ! *i* ≠ *NOEXIST*)
  prefer *2* **apply**(*simp add*:*inv-def inv-bitmap0-def Let-def*)
  apply(*subgoal-tac levels* (*mem-pool-info V* (*pool b*)) ! (*lvl V2 t* − *1*)
    = *levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *1*)) **prefer** *2*
   apply(*subgoal-tac levels* (*set-bit-free* (*mem-pool-info V*) (*pool b*) (*lvl V2 t*) (*bn*

V2 t) (*pool b*)) ! (*lvl V2 t* − *1*)
= *levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *1*)) **prefer** *2* **apply** *auto*[*1*]
apply(*simp add*:*set-bit-def*)

using *mempool-free-stm8-atombody-rest-one-finalstm-inv-lvl0-case1* **apply** *blast*

apply *clarsimp*
using *mempool-free-stm8-atombody-rest-one-finalstm-inv-lvl0-case2* **apply** *blast*
done

term *mp-free-precond8-3 t b* α
term *free-stm8-precond2 V t b*
term *free-stm8-precond3 V t b*

lemma *mempool-free-stm8-atombody-rest-one-finalstm-inv-lvln-case1*:
pool b ∈ *mem-pools V2* ⟹
inv V ⟹
NULL < *lvl V2 t* ⟹
pool b ∈ *mem-pools V* ⟹
level b < *length* (*levels* (*mem-pool-info V* (*pool b*))) ⟹
(*V2*, *V*⦇*mem-pool-info* := *set-bit-free* (*mem-pool-info V*) (*pool b*) (*lvl V2 t*) (*bn V2 t*),
freeing-node := (*freeing-node V*)(*t* := *None*)⦈)
∈ *gvars-conf-stable* ⟹
∀ *j*. *j* ≠ *lvl V2 t* ⟶
levels (*set-bit-free* (*mem-pool-info V*) (*pool b*) (*lvl V2 t*) (*bn V2 t*) (*pool b*)) ! *j* =
levels (*mem-pool-info V2* (*pool b*)) ! *j* ⟹
bits (*levels* (*mem-pool-info V2* (*pool b*)) ! *lvl V2 t*) =
list-updates-n (*bits* (*levels* (*set-bit-free* (*mem-pool-info V*) (*pool b*) (*lvl V2 t*) (*bn V2 t*) (*pool b*)) ! *lvl V2 t*))
(*bn V2 t div 4* ∗ *4*) *4 NOEXIST* ⟹
lvl V2 t ≤ *level b* ⟹
ia < *length* (*bits* (*levels* (*mem-pool-info V2* (*pool b*))
[*lvl V2 t* − *Suc NULL* := (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))
⦇*bits* := *bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))
[*bn V2 t div 4* := *FREEING*]⦈] !
(*length* (*levels* (*mem-pool-info V2* (*pool b*))) − *Suc NULL*))) ⟹
bits (*levels* (*mem-pool-info V2* (*pool b*))
[*lvl V2 t* − *Suc NULL* := (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))
⦇*bits* := *bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))[*bn V2 t div 4* := *FREEING*]⦈] !
(*length* (*levels* (*mem-pool-info V2* (*pool b*))) − *Suc NULL*)) ! *ia* = *DIVIDED* ⟹

False
apply(*simp add*:*set-bit-def*)
apply(*subgoal-tac length* (*levels* (*mem-pool-info V2* (*pool b*)))
= *length* (*levels* (*mem-pool-info V* (*pool b*)))) **prefer** *2*
using *mempool-free-stm8-atombody-rest-one-finalstm-len-lvls* **apply** *blast*

apply(*subgoal-tac let bitsn* = *bits* ((*levels* (*mem-pool-info V* (*pool b*)) ! (*length* (*levels* (*mem-pool-info V* (*pool b*))) − *1*)))
in ∀ *i*<*length bitsn*. *bitsn* ! *i* ≠ *DIVIDED*) **prefer** *2*
apply(*simp add*:*inv-def inv-bitmapn-def*)

apply(*case-tac lvl V2 t* = *length* (*levels* (*mem-pool-info V2* (*pool b*))) − *Suc 0*)
apply(*subgoal-tac bits* (*levels* (*mem-pool-info V2* (*pool b*))
[*lvl V2 t* − *Suc NULL* := (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))
(|*bits* := *bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))[*bn V2 t div 4* := *FREEING*]|)] !
(*length* (*levels* (*mem-pool-info V2* (*pool b*))) − *Suc NULL*)) ! *ia* ≠ *DIVIDED*)

apply *auto*[*1*]
apply(*rule subst*[**where** *s*=*bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! (*length* (*levels* (*mem-pool-info V2* (*pool b*))) − *Suc NULL*))
and *t*=*bits* (*levels* (*mem-pool-info V2* (*pool b*))
[*lvl V2 t* − *Suc NULL* := (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))
(|*bits* := *bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))[*bn V2 t div 4* := *FREEING*]|)] !
(*length* (*levels* (*mem-pool-info V2* (*pool b*))) − *Suc NULL*))])
apply *auto*[*1*]
apply(*unfold Let-def*)[*1*]
apply(*subgoal-tac* ∀ *i*<*length* (*bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! *lvl V2 t*)). *get-bit-s V2* (*pool b*) (*lvl V2 t*) *i* ≠ *DIVIDED*)
apply *auto*[*1*]

apply(*rule list-neq-udpt-neq*[*of bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V2 t*) *DIVIDED*
bits (*levels* (*mem-pool-info V2* (*pool b*)) ! *lvl V2 t*) (*bn V2 t div 4* ∗ *4*) *4 NOEXIST*])
apply *auto*[*1*]
using *lst-udptn-set-eq*[*of 4 bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V2 t*) *bn V2 t FREE NOEXIST*]
apply *auto*[*1*]
apply *blast*

apply(*subgoal-tac bits* (*levels* (*mem-pool-info V2* (*pool b*))
[*lvl V2 t* − *Suc NULL* := (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))
(|*bits* := *bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc*

NULL))[*bn V2 t div 4* := *FREEING*]⦈] !
  (*length* (*levels* (*mem-pool-info V2* (*pool b*))) − *Suc NULL*)) ! *ia* ≠ *DIVIDED*)

apply *fast*
apply(*rule subst*[**where** *s*=*levels* (*mem-pool-info V2* (*pool b*)) !
  (*length* (*levels* (*mem-pool-info V2* (*pool b*))) − *Suc NULL*) **and** *t*=*levels* (*mem-pool-info V2* (*pool b*))
  [*lvl V2 t* − *Suc NULL* := (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))
  ⦇*bits* := *bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))[*bn V2 t div 4* := *FREEING*]⦈] !
  (*length* (*levels* (*mem-pool-info V2* (*pool b*))) − *Suc NULL*)])
apply *auto*[*1*]
apply(*unfold Let-def*)[*1*]
apply(*subgoal-tac levels* (*mem-pool-info V* (*pool b*)) ! (*length* (*levels* (*mem-pool-info V* (*pool b*))) − *1*) =
  levels (*mem-pool-info V2* (*pool b*)) ! (*length* (*levels* (*mem-pool-info V* (*pool b*))) − *1*))
prefer *2* **apply** (*metis One-nat-def*)
by (*metis One-nat-def Suc-diff-1 inv-mempool-info-def invariant.inv-def not-less nth-list-update-neq*)

lemma *mempool-free-stm8-atombody-rest-one-finalstm-inv-lvln-case2*:
p ∈ *mem-pools V2* ⟹
inv V ⟹
NULL < *lvl V2 t* ⟹
level b < *length* (*levels* (*mem-pool-info V* (*pool b*))) ⟹
(*V2*, *V*⦇*mem-pool-info* := *set-bit-free* (*mem-pool-info V*) (*pool b*) (*lvl V2 t*) (*bn V2 t*),
  freeing-node := (*freeing-node V*)(*t* := *None*)⦈)
∈ *gvars-conf-stable* ⟹
∀ *p*. *p* ≠ *pool b* ⟶ *mem-pool-info V2 p* = *set-bit-free* (*mem-pool-info V*) (*pool b*) (*lvl V2 t*) (*bn V2 t*) *p* ⟹
lvl V2 t ≤ *level b* ⟹
p ≠ *pool b* ⟹
ia < *length* (*bits* (*levels* (*set-bit-free* (*mem-pool-info V*) (*pool b*) (*lvl V2 t*) (*bn V2 t*) *p*) !
  (*length* (*levels* (*set-bit-free* (*mem-pool-info V*) (*pool b*) (*lvl V2 t*) (*bn V2 t*) *p*)) − *Suc NULL*))) ⟹
get-bit (*set-bit-free* (*mem-pool-info V*) (*pool b*) (*lvl V2 t*) (*bn V2 t*)) *p*
 (*length* (*levels* (*set-bit-free* (*mem-pool-info V*) (*pool b*) (*lvl V2 t*) (*bn V2 t*) *p*)) − *Suc NULL*) *ia* =
DIVIDED ⟹
False
apply(*simp add*:*set-bit-def*)
apply(*subgoal-tac* ∀ *i*<*length* (*bits* ((*levels* (*mem-pool-info V p*) ! (*length* (*levels* (*mem-pool-info V p*)) − *1*)))).
  bits ((*levels* (*mem-pool-info V p*) ! (*length* (*levels* (*mem-pool-info V p*)) − *1*))) ! *i* ≠ *DIVIDED*)

prefer *2* **apply**(*subgoal-tac mem-pools V2 = mem-pools V*) **prefer** *2* **apply**(*simp add:gvars-conf-stable-def gvars-conf-def*)
apply(*simp add:inv-def inv-bitmapn-def Let-def*)
apply *auto*
done

lemma *mempool-free-stm8-atombody-rest-one-finalstm-inv-lvln*:
$V \in$*mp-free-precond8-3 t b* $\alpha \cap$ ⦃´*cur = Some t*⦄ $\Longrightarrow$
{*free-stm8-precond2 V t b*} $\cap$ ⦃*NULL* < ´*lvl t* $\wedge$ *partner-bits* (´*mem-pool-info* (*pool b*)) (´*lvl t*) (´*bn t*)⦄ $\neq$ {} $\Longrightarrow$
V2 $\in$ *free-stm8-precond3 V t b* $\cap$ ⦃´*i t* = *4*⦄ $\Longrightarrow$
x = *free-stm8-atombody-rest-cond3* (*V2*⦇*lvl* := (*lvl V2*)(*t* := *lvl V2 t* − *1*), *bn* := (*bn V2*)(*t* := *bn V2 t div 4*)⦈) *t b* $\Longrightarrow$
y = *x*⦇*freeing-node* := (*freeing-node x*) (*t* := *Some* (⦇*pool* = *pool b*, *level* = *lvl x t*, *block* = *bn x t*,
data = *block-ptr* (*mem-pool-info x* (*pool b*)) (*ALIGN4* (*max-sz* (*mem-pool-info x* (*pool b*))) *div 4* ^ *lvl x t*) (*bn x t*)⦈)⦈ $\Longrightarrow$
inv-bitmapn y
apply(*simp add:inv-bitmapn-def Let-def*)
apply *clarify*

apply(*rule conjI*)
apply *clarsimp*
using *mempool-free-stm8-atombody-rest-one-finalstm-inv-lvln-case1* **apply** *blast*

apply *clarsimp*
using *mempool-free-stm8-atombody-rest-one-finalstm-inv-lvln-case2* **apply** *blast*

done

lemma *mempool-free-stm8-atombody-rest-one-finalstm-inv-lvls-not4free-case1-h1*:
lvl V2 t − *Suc NULL* = *ia* $\Longrightarrow$
lvl V2 t − *Suc NULL* < *length* (*levels* (*mem-pool-info V2* (*pool b*))) $\Longrightarrow$
bits (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))[*bn V2 t div 4* := *FREEING*] =
bits (*levels* (*mem-pool-info V2* (*pool b*)
(⦇*levels* := *levels* (*mem-pool-info V2* (*pool b*))
[*lvl V2 t* − *Suc NULL* := (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))
(⦇*bits* := *bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))[*bn V2 t div 4* := *FREEING*]⦈]⦈) !
ia)
by *simp*

lemma *mempool-free-stm8-atombody-rest-one-finalstm-inv-lvls-not4free-case1-h2*:
ia < *length* (*levels* (*mem-pool-info V2* (*pool b*))) $\Longrightarrow$
jj < *length* (*bits* (*levels* (*mem-pool-info V2* (*pool b*))

[*lvl V2 t* − *Suc NULL* := (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))
(|*bits* := *bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))[*bn V2 t div 4* := *FREEING*]|)] !
ia)) ⟹
NULL < *ia* ⟹
length (*levels* (*mem-pool-info V2* (*pool b*))) = *length* (*levels* (*mem-pool-info V* (*pool b*))) ⟹
length (*bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! *ia*)) = *length* (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *ia*)) ⟹
∀ *jj*<*length* (*bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! *ia*)).
¬ (*let bits* = *bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *ia*); *a* = *jj div 4* ∗ *4*
in bits ! *a* = *FREE* ∧ *bits* ! (*a* + *1*) = *FREE* ∧ *bits* ! (*a* + *2*) = *FREE* ∧ *bits* ! (*a* + *3*) = *FREE*) ⟹
lvl V2 t − *Suc NULL* = *ia* ⟹
levels (*mem-pool-info V* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*) = *levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*) ⟹
¬ (*let bits* = *bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))[*bn V2 t div 4* := *FREEING*]; *a* = *jj div 4* ∗ *4*
in bits ! *a* = *FREE* ∧ *bits* ! (*a* + *1*) = *FREE* ∧ *bits* ! (*a* + *2*) = *FREE* ∧ *bits* ! (*a* + *3*) = *FREE*)
apply(*unfold Let-def*)
apply(*rule subst*[**where** *s*=*list-updates-n* (*bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))) (*bn V2 t div 4*) *1 FREEING* **and**
t=*bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))[*bn V2 t div 4* := *FREEING*]])
using *lst-updt1-eq-upd* **apply** *fast*
apply(*subgoal-tac length* (*list-updates-n* (*bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))) (*bn V2 t div 4*) *1 FREEING*)
= *length* (*bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! *ia*)))
prefer *2* **using** *length-list-update-n* **apply** *fast*
apply(*subgoal-tac* ∀ *jj*<*length* (*list-updates-n* (*bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))) (*bn V2 t div 4*) *1 FREEING*).
¬ (*let a* = *jj div 4* ∗ *4*
in list-updates-n (*bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))) (*bn V2 t div 4*) *1 FREEING* ! *a* = *FREE* ∧
list-updates-n (*bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))) (*bn V2 t div 4*) *1 FREEING* ! (*a* + *1*) = *FREE* ∧
list-updates-n (*bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))) (*bn V2 t div 4*) *1 FREEING* ! (*a* + *2*) = *FREE* ∧
list-updates-n (*bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))) (*bn V2 t div 4*) *1 FREEING* ! (*a* + *3*) = *FREE*))
prefer *2*
apply(*rule partnerbits-udptn-notbit-partbits*[*of bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! *ia*) *FREE FREEING*
list-updates-n (*bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))) (*bn V2 t div 4*) *1 FREEING* (*bn V2 t div 4*) *1*])
apply(*unfold Let-def*)[*1*] **apply** *metis*
apply *blast*

apply *fast*
apply(*unfold Let-def*)
apply(*subgoal-tac length* (*bits* (*levels* (*mem-pool-info V2* (*pool b*))
[*lvl V2 t* − *Suc 0* := (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc 0*))
(|*bits* := *bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc 0*))[*bn V2 t div 4* := *FREEING*]|)] !
ia)) = *length* (*bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! *ia*)))
prefer *2*
using *mempool-free-stm8-atombody-rest-one-finalstm-inv-mempool-info-h2* **apply** *fast*
by *metis*

lemma *mempool-free-stm8-atombody-rest-one-finalstm-inv-lvls-not4free-case1-h3*:
lvl V2 t − *Suc NULL* ≠ *ia* ⟹
bits (*levels* (*mem-pool-info V2* (*pool b*)) ! *ia*) =
bits (*levels* (*mem-pool-info V2* (*pool b*)
(|*levels* := *levels* (*mem-pool-info V2* (*pool b*))
[*lvl V2 t* − *Suc NULL* := (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))
(|*bits* := *bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))[*bn V2 t div 4* := *FREEING*]|)]|)) !
ia)
by *auto*

lemma *mempool-free-stm8-atombody-rest-one-finalstm-inv-lvls-not4free-case1*:
pool b ∈ *mem-pools V2* ⟹
inv V ⟹
NULL < *lvl V2 t* ⟹
partner-bits (*set-bit-free* (*mem-pool-info V*) (*pool b*) (*lvl V2 t*) (*bn V2 t*) (*pool b*)) (*lvl V2 t*) (*bn V2 t*) ⟹
pool b ∈ *mem-pools V* ⟹
level b < *length* (*levels* (*mem-pool-info V* (*pool b*))) ⟹
(*V2*, *V*(|*mem-pool-info* := *set-bit-free* (*mem-pool-info V*) (*pool b*) (*lvl V2 t*) (*bn V2 t*),
freeing-node := (*freeing-node V*)(*t* := *None*)|))
∈ *gvars-conf-stable* ⟹
block b < *length* (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *level b*)) ⟹
∀ *j*. *j* ≠ *lvl V2 t* ⟶
levels (*set-bit-free* (*mem-pool-info V*) (*pool b*) (*lvl V2 t*) (*bn V2 t*) (*pool b*)) ! *j* =
levels (*mem-pool-info V2* (*pool b*)) ! *j* ⟹
level b < *length* (*lsizes V2 t*) ⟹
bits (*levels* (*mem-pool-info V2* (*pool b*)) ! *lvl V2 t*) =
list-updates-n (*bits* (*levels* (*set-bit-free* (*mem-pool-info V*) (*pool b*) (*lvl V2 t*) (*bn V2 t*) (*pool b*)) ! *lvl V2 t*))
(*bn V2 t div 4* ∗ *4*) *4 NOEXIST* ⟹
lvl V2 t ≤ *level b* ⟹

ia < *length* (*levels* (*mem-pool-info V2* (*pool b*))) ⟹
jj < *length* (*bits* (*levels* (*mem-pool-info V2* (*pool b*))
[*lvl V2 t* − *Suc NULL* := (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))
(|*bits* := *bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))
[*bn V2 t div 4* := *FREEING*]|)] !
ia)) ⟹
NULL < *ia* ⟹
partner-bits
(*mem-pool-info V2* (*pool b*)
(|*levels* := *levels* (*mem-pool-info V2* (*pool b*))
[*lvl V2 t* − *Suc NULL* := (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))
(|*bits* := *bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))[*bn V2 t div 4* := *FREEING*]|)]|))
ia jj ⟹
False
apply(*simp add:set-bit-def*)
apply(*subgoal-tac length* (*levels* (*mem-pool-info V2* (*pool b*)))
= *length* (*levels* (*mem-pool-info V* (*pool b*)))) **prefer** *2*
using *mempool-free-stm8-atombody-rest-one-finalstm-len-lvls* **apply** *blast*

apply(*subgoal-tac* ¬ *partner-bits* (*mem-pool-info V2* (*pool b*) (|*levels* := *levels* (*mem-pool-info V2* (*pool b*))
[*lvl V2 t* − *Suc NULL* := (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))
(|*bits* := *bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))[*bn V2 t div 4* := *FREEING*]|)]|)
ia jj) **apply** *fast*

apply(*subgoal-tac length* (*bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! *ia*)) = *length* (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *ia*)))
prefer *2* **apply**(*case-tac lvl V2 t* = *ia*) **apply**(*simp add:set-bit-def*) **apply** *presburger*
apply(*subgoal-tac* ∀ *jj*<*length* (*bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! *ia*)). ¬ *partner-bits* (*mem-pool-info V* (*pool b*)) *ia jj*)
prefer *2* **apply**(*simp add:inv-def inv-bitmap-not4free-def Let-def*)

apply(*case-tac lvl V2 t* = *ia*)
apply(*rule subst*[**where** *s*=*partner-bits* (*mem-pool-info V2* (*pool b*)) *ia jj* **and**
t=*partner-bits* (*mem-pool-info V2* (*pool b*) (|*levels* := *levels* (*mem-pool-info V2* (*pool b*))
[*lvl V2 t* − *Suc NULL* := (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))
(|*bits* := *bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))[*bn V2 t div 4* := *FREEING*]|)]|)
ia jj]) **apply**(*simp add:partner-bits-def Let-def*)

apply(*subgoal-tac bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! *lvl V2 t*) =
list-updates-n (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V2 t*)) (*bn V2 t div 4* ∗ *4*) *4 NOEXIST*) **prefer** *2*
using *lst-udptn-set-eq*[*of 4 bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V2 t*) *bn V2 t FREE NOEXIST*] **apply** *simp*

apply(*unfold partner-bits-def*)[*1*]
apply(*subgoal-tac* ¬ (*let a* = *jj div 4* ∗ *4*
in list-updates-n (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *ia*)) (*bn V2 t div 4* ∗ *4*) *4 NOEXIST* ! *a* = *FREE* ∧
list-updates-n (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *ia*)) (*bn V2 t div 4* ∗ *4*) *4 NOEXIST* ! (*a* + *1*) = *FREE* ∧
list-updates-n (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *ia*)) (*bn V2 t div 4* ∗ *4*) *4 NOEXIST* ! (*a* + *2*) = *FREE* ∧
list-updates-n (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *ia*)) (*bn V2 t div 4* ∗ *4*) *4 NOEXIST* ! (*a* + *3*) = *FREE*))
prefer *2*
apply(*rule partnerbits-udptn-notbit-partbits*[*rule-format*, *of bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *ia*) *FREE NOEXIST*
list-updates-n (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *ia*)) (*bn V2 t div 4* ∗ *4*) *4 NOEXIST*
bn V2 t div 4 ∗ *4 4 jj*])
apply(*unfold Let-def*)[*1*] **apply** *presburger*
apply *blast* **apply** *fast* **apply** *force*
apply(*unfold Let-def*)[*1*] **apply** *presburger*

apply(*case-tac lvl V2 t* − *Suc 0* = *ia*)
apply(*unfold partner-bits-def*)
apply(*rule subst*[**where** *s*=*bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc 0*))[*bn V2 t div 4* := *FREEING*]
and *t*=*bits* (*levels* (*mem-pool-info V2* (*pool b*)
(|*levels* := *levels* (*mem-pool-info V2* (*pool b*))
[*lvl V2 t* − *Suc 0* := (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc 0*))
(|*bits* := *bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc 0*))[*bn V2 t div 4* := *FREEING*]|)]|)) !
ia)])
apply(*subgoal-tac lvl V2 t* − *Suc NULL* < *length* (*levels* (*mem-pool-info V2* (*pool b*)))) **prefer** *2* **apply** *blast*
using *mempool-free-stm8-atombody-rest-one-finalstm-inv-lvls-not4free-case1-h1* **apply** *blast*

apply(*subgoal-tac levels* (*mem-pool-info V* (*pool b*)) ! (*lvl V2 t* − *Suc 0*) = *levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc 0*))
prefer *2* **apply** *presburger*
using *mempool-free-stm8-atombody-rest-one-finalstm-inv-lvls-not4free-case1-h2* **apply** *blast*

apply(*rule subst*[**where** *s*=*bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! *ia*) **and**
t=*bits* (*levels* (*mem-pool-info V2* (*pool b*)
(|*levels* := *levels* (*mem-pool-info V2* (*pool b*))
[*lvl V2 t* − *Suc NULL* := (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))
(|*bits* := *bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))[*bn V2 t div 4* := *FREEING*]|)]|)) !
ia)])
using *mempool-free-stm8-atombody-rest-one-finalstm-inv-lvls-not4free-case1-h3* **apply** *blast*

apply(*rule subst*[**where** *s*=*levels* (*mem-pool-info V* (*pool b*)) ! *ia* **and** *t*=*levels* (*mem-pool-info V2* (*pool b*)) ! *ia*])
apply *metis*
apply(*unfold Let-def*)

apply(*subgoal-tac length* (*bits* (*levels* (*mem-pool-info V2* (*pool b*))
[*lvl V2 t* − *Suc NULL* := (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))
(|*bits* := *bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))[*bn V2 t div 4* := *FREEING*]|)] !
ia)) =
length (*bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! *ia*))) **prefer** *2*
using *mempool-free-stm8-atombody-rest-one-finalstm-inv-mempool-info-h2* **apply** *blast*
by *metis*

lemma *mempool-free-stm8-atombody-rest-one-finalstm-inv-lvls-not4free-case2*:
p ∈ *mem-pools V2* ⟹
inv V ⟹
NULL < *lvl V2 t* ⟹
partner-bits (*set-bit-free* (*mem-pool-info V*) (*pool b*) (*lvl V2 t*) (*bn V2 t*) (*pool b*)) (*lvl V2 t*) (*bn V2 t*) ⟹
pool b ∈ *mem-pools V* ⟹
level b < *length* (*levels* (*mem-pool-info V* (*pool b*))) ⟹
(*V2*, *V*(|*mem-pool-info* := *set-bit-free* (*mem-pool-info V*) (*pool b*) (*lvl V2 t*) (*bn V2 t*),
freeing-node := (*freeing-node V*)(*t* := *None*)|))
∈ *gvars-conf-stable* ⟹
lvl V2 t ≤ *level b* ⟹
∀ *p*. *p* ≠ *pool b* ⟶ *mem-pool-info V2 p* = *set-bit-free* (*mem-pool-info V*) (*pool b*) (*lvl V2 t*) (*bn V2 t*) *p* ⟹
p ≠ *pool b* ⟹
ia < *length* (*levels* (*set-bit-free* (*mem-pool-info V*) (*pool b*) (*lvl V2 t*) (*bn V2 t*) *p*)) ⟹
jj < *length* (*bits* (*levels* (*set-bit-free* (*mem-pool-info V*) (*pool b*) (*lvl V2 t*) (*bn V2

t) *p*) ! *ia*)) ⟹
NULL < *ia* ⟹ *partner-bits* (*set-bit-free* (*mem-pool-info V*) (*pool b*) (*lvl V2 t*) (*bn V2 t*) *p*) *ia jj* ⟹ *False*
apply(*simp add*:*set-bit-def*)
apply(*subgoal-tac length* (*levels* (*mem-pool-info V2* (*pool b*)))
= *length* (*levels* (*mem-pool-info V* (*pool b*)))) **prefer** *2*
using *mempool-free-stm8-atombody-rest-one-finalstm-len-lvls* **apply** *blast*

apply(*subgoal-tac* ¬ *partner-bits* (*mem-pool-info V p*) *ia jj*) **apply** *fast*

apply(*subgoal-tac p* ∈ *mem-pools V*) **prefer** *2* **apply**(*simp add*:*gvars-conf-stable-def gvars-conf-def*)
apply(*subgoal-tac* ∀ *jj*<*length* (*bits* (*levels* (*mem-pool-info V p*) ! *ia*)). ¬ *partner-bits* (*mem-pool-info V p*) *ia jj*)
prefer *2* **apply**(*simp add*:*inv-def inv-bitmap-not4free-def Let-def*)

by *blast*

lemma *mempool-free-stm8-atombody-rest-one-finalstm-inv-lvls-not4free*:
V∈*mp-free-precond8-3 t b α* ∩ ⦃´*cur* = *Some t*⦄ ⟹
{*free-stm8-precond2 V t b*} ∩ ⦃*NULL* < ´*lvl t* ∧ *partner-bits* (´*mem-pool-info* (*pool b*)) (´*lvl t*) (´*bn t*)⦄ ≠ {} ⟹
V2 ∈ *free-stm8-precond3 V t b* ∩ ⦃´*i t* = *4*⦄ ⟹
x = *free-stm8-atombody-rest-cond3* (*V2*⦇*lvl* := (*lvl V2*)(*t* := *lvl V2 t* − *1*), *bn* := (*bn V2*)(*t* := *bn V2 t div 4*)⦈) *t b* ⟹
y = *x*⦇*freeing-node* := (*freeing-node x*) (*t* := *Some* ⦇*pool* = *pool b*, *level* = *lvl x t*, *block* = *bn x t*,
data = *block-ptr* (*mem-pool-info x* (*pool b*)) (*ALIGN4* (*max-sz* (*mem-pool-info x* (*pool b*))) *div 4* ^ *lvl x t*) (*bn x t*)⦈)⦈ ⟹
inv-bitmap-not4free y
apply(*simp add*:*inv-bitmap-not4free-def Let-def*)
apply *clarify*
apply(*rule conjI*)
apply *clarsimp*
using *mempool-free-stm8-atombody-rest-one-finalstm-inv-lvls-not4free-case1* **apply** *blast*

apply *clarsimp*
using *mempool-free-stm8-atombody-rest-one-finalstm-inv-lvls-not4free-case2* **apply** *blast*
done

lemma *mempool-free-stm8-atombody-rest-one-finalstm-inv′*:
V∈*mp-free-precond8-3 t b α* ∩ ⦃´*cur* = *Some t*⦄ ⟹
{*free-stm8-precond2 V t b*} ∩ ⦃*NULL* < ´*lvl t* ∧ *partner-bits* (´*mem-pool-info* (*pool b*)) (´*lvl t*) (´*bn t*)⦄ ≠ {} ⟹
V2 ∈ *free-stm8-precond3 V t b* ∩ ⦃´*i t* = *4*⦄ ⟹
x = *free-stm8-atombody-rest-cond3* (*V2*⦇*lvl* := (*lvl V2*)(*t* := *lvl V2 t* − *1*), *bn*

:= (*bn V2*)(*t* := *bn V2 t div 4*)⦈) *t b* ⟹
  y = *x*⦇*freeing-node* := (*freeing-node x*) (*t* := *Some* ⦇*pool* = *pool b*, *level* = *lvl x t*, *block* = *bn x t*,
                    data = *block-ptr* (*mem-pool-info x* (*pool b*)) (*ALIGN4* (*max-sz* (*mem-pool-info x* (*pool b*))) *div 4* ˆ *lvl x t*) (*bn x t*)⦈)⦈ ⟹
  inv y
apply(*rule subst*[**where** *s*=*inv-cur y* ∧ *inv-thd-waitq y* ∧ *inv-mempool-info y*
  ∧ *inv-bitmap-freelist y* ∧ *inv-bitmap y* ∧ *inv-aux-vars y*
  ∧ *inv-bitmap0 y* ∧ *inv-bitmapn y* ∧ *inv-bitmap-not4free y* **and** *t*=*inv y*])
using *inv-def*[*of y*] **apply** *fast*

apply(*rule conjI*) **using** *mempool-free-stm8-atombody-rest-one-finalstm-inv-cur*[*of V t b α V2 x y*] **apply** *fast*
apply(*rule conjI*) **using** *mempool-free-stm8-atombody-rest-one-finalstm-inv-thd-waitq*[*of V t b α V2 x y*] **apply** *fast*
apply(*rule conjI*) **using** *mempool-free-stm8-atombody-rest-one-finalstm-inv-mempool-info*[*of V t b α V2 x y*] **apply** *fast*
apply(*rule conjI*) **using** *mempool-free-stm8-atombody-rest-one-finalstm-inv-bitmap-freelist*[*of V t b α V2 x y*] **apply** *fast*
apply(*rule conjI*) **using** *mempool-free-stm8-atombody-rest-one-finalstm-inv-bitmap*[*of V t b α V2 x y*] **apply** *fast*
apply(*rule conjI*) **using** *mempool-free-stm8-atombody-rest-one-finalstm-inv-aux-vars*[*of V t b α V2 x y*] **apply** *fast*
apply(*rule conjI*) **using** *mempool-free-stm8-atombody-rest-one-finalstm-inv-lvl0*[*of V t b α V2 x y*] **apply** *fast*
apply(*rule conjI*) **using** *mempool-free-stm8-atombody-rest-one-finalstm-inv-lvln*[*of V t b α V2 x y*] **apply** *fast*
  using *mempool-free-stm8-atombody-rest-one-finalstm-inv-lvls-not4free*[*of V t b α V2 x y*] **apply** *fast*

done

lemma *mempool-free-stm8-atombody-rest-one-finalstm-inv*:
  V∈*mp-free-precond8-3 t b α* ∩ ⦃´*cur* = *Some t*⦄ ⟹
  {*free-stm8-precond2 V t b*} ∩ ⦃*NULL* < ´*lvl t* ∧ *partner-bits* (´*mem-pool-info* (*pool b*)) (´*lvl t*) (´*bn t*)⦄ ≠ {} ⟹
  V2 ∈ *free-stm8-precond3 V t b* ∩ ⦃´*i t* = *4*⦄ ⟹
  x = *free-stm8-atombody-rest-cond3* (*V2*⦇*lvl* := (*lvl V2*)(*t* := *lvl V2 t* − *1*), *bn* := (*bn V2*)(*t* := *bn V2 t div 4*)⦈) *t b* ⟹
  x⦇*freeing-node* := (*freeing-node x*) (*t* := *Some* ⦇*pool* = *pool b*, *level* = *lvl x t*, *block* = *bn x t*,
                    data = *block-ptr* (*mem-pool-info x* (*pool b*)) (*ALIGN4* (*max-sz* (*mem-pool-info x* (*pool b*))) *div 4* ˆ *lvl x t*) (*bn x t*)⦈)⦈
          ∈ ⦃´*inv*⦄
using *mempool-free-stm8-atombody-rest-one-finalstm-inv′*[*of V t b α V2 x*
    x⦇*freeing-node* := (*freeing-node x*) (*t* := *Some* ⦇*pool* = *pool b*, *level* = *lvl x t*, *block* = *bn x t*,

data = *block-ptr* (*mem-pool-info x* (*pool b*)) (*ALIGN4* (*max-sz* (*mem-pool-info x* (*pool b*))) *div 4* ^ *lvl x t*) (*bn x t*)⦈)⦈] **apply** *fast*
done

lemma *mempool-free-stm8-atombody-rest-one-finalstm-h2*:
$V \in mp\text{-}free\text{-}precond8\text{-}3\ t\ b\ \alpha \cap \{\!\!\{ {}' cur = Some\ t \}\!\!\} \Longrightarrow$
$\{free\text{-}stm8\text{-}precond2\ V\ t\ b\} \cap \{\!\!\{NULL < {}'lvl\ t \wedge partner\text{-}bits\ ({}'mem\text{-}pool\text{-}info\ (pool\ b))\ ({}'lvl\ t)\ ({}'bn\ t)\}\!\!\} \neq \{\} \Longrightarrow$
$V2 \in free\text{-}stm8\text{-}precond3\ V\ t\ b \cap \{\!\!\{{}'i\ t = 4\}\!\!\} \Longrightarrow$
$x = free\text{-}stm8\text{-}atombody\text{-}rest\text{-}cond3\ (V2(\!|lvl := (lvl\ V2)(t := lvl\ V2\ t - 1),\ bn := (bn\ V2)(t := bn\ V2\ t\ div\ 4)|\!))\ t\ b \Longrightarrow$
$x(\!|freeing\text{-}node := (freeing\text{-}node\ x)\ (t := Some\ (\!|pool = pool\ b,\ level = lvl\ x\ t,\ block = bn\ x\ t,$
$data = block\text{-}ptr\ (mem\text{-}pool\text{-}info\ x\ (pool\ b))\ (ALIGN4\ (max\text{-}sz\ (mem\text{-}pool\text{-}info\ x\ (pool\ b)))\ div\ 4\ \hat{}\ lvl\ x\ t)\ (bn\ x\ t)|\!))|\!)$
$\in \{\!\!\{{}'allocating\text{-}node\ t = None\}\!\!\}$
by (*simp add*:*Let-def block-ptr-def*)

lemma *mempool-free-stm8-atombody-rest-one-finalstm-h1-2*:
$V \in mp\text{-}free\text{-}precond8\text{-}3\ t\ b\ \alpha \cap \{\!\!\{ {}' cur = Some\ t \}\!\!\} \Longrightarrow$
$\{free\text{-}stm8\text{-}precond2\ V\ t\ b\} \cap \{\!\!\{NULL < {}'lvl\ t \wedge partner\text{-}bits\ ({}'mem\text{-}pool\text{-}info\ (pool\ b))\ ({}'lvl\ t)\ ({}'bn\ t)\}\!\!\} \neq \{\} \Longrightarrow$
$V2 \in free\text{-}stm8\text{-}precond3\ V\ t\ b \cap \{\!\!\{{}'i\ t = 4\}\!\!\} \Longrightarrow$
$x = free\text{-}stm8\text{-}atombody\text{-}rest\text{-}cond3\ (V2(\!|lvl := (lvl\ V2)(t := lvl\ V2\ t - 1),\ bn := (bn\ V2)(t := bn\ V2\ t\ div\ 4)|\!))\ t\ b \Longrightarrow$
$y = x(\!|freeing\text{-}node := freeing\text{-}node\ x(t \mapsto$
$(\!|pool = pool\ b,\ level = lvl\ x\ t,\ block = bn\ x\ t,$
$data = block\text{-}ptr\ (mem\text{-}pool\text{-}info\ x\ (pool\ b))\ (ALIGN4\ (max\text{-}sz\ (mem\text{-}pool\text{-}info\ x\ (pool\ b)))\ div\ 4\ \hat{}\ lvl\ x\ t)\ (bn\ x\ t)|\!))|\!) \Longrightarrow$
$y \in \{\!\!\{{}'(Pair\ V)$
$\in \{(s,\ r).\ (cur\ s \neq Some\ t \longrightarrow gvars\text{-}nochange\ s\ r \wedge lvars\text{-}nochange\ t\ s\ r) \wedge$
$(cur\ s = Some\ t \longrightarrow invariant.inv\ s \longrightarrow invariant.inv\ r) \wedge (\forall t'.\ t' \neq t \longrightarrow lvars\text{-}nochange\ t'\ s\ r)\}\}\!\!\}$
apply(*subgoal-tac* $(cur\ V \neq Some\ t \longrightarrow gvars\text{-}nochange\ V\ y \wedge lvars\text{-}nochange\ t\ V\ y) \wedge$
$(cur\ V = Some\ t \longrightarrow invariant.inv\ V \longrightarrow invariant.inv\ y) \wedge (\forall t'.\ t' \neq t \longrightarrow lvars\text{-}nochange\ t'\ V\ y)$)
prefer *2*
apply(*rule conjI*)
apply(*subgoal-tac cur V = Some t*) **prefer** *2* **apply** *fast* **apply** *fast*
apply(*rule conjI*)
apply(*rule impI*)+ **using** *mempool-free-stm8-atombody-rest-one-finalstm-inv'*[*of V t b α V2 x y*] **apply** *fast*
apply(*rule allI*) **apply**(*rule impI*) **apply**(*simp add*:*lvars-nochange-def Let-def*)

apply *fast*
done

lemma *mempool-free-stm8-atombody-rest-one-finalstm-h1-h1*:
$\forall j.\ j \neq lvl\ V\ t \longrightarrow levels\ (mem\text{-}pool\text{-}info\ V\ (pool\ b))\ !\ j = levels\ (mem\text{-}pool\text{-}info\ V2\ (pool\ b))\ !\ j \Longrightarrow$
$bits\ (levels\ (mem\text{-}pool\text{-}info\ V2\ (pool\ b))\ !\ lvl\ V\ t) =$
$list\text{-}updates\text{-}n\ (bits\ (levels\ (mem\text{-}pool\text{-}info\ V\ (pool\ b))\ !\ lvl\ V\ t)[block\ b\ div\ 4\ \hat{}\ (level\ b - lvl\ V\ t) := FREE])$
$(block\ b\ div\ 4\ \hat{}\ (level\ b - lvl\ V\ t)\ div\ 4 * 4)\ 4\ NOEXIST \Longrightarrow$
$length\ (bits\ (levels\ (mem\text{-}pool\text{-}info\ V\ (pool\ b))\ !\ ia)) =$
$length\ (bits\ (levels\ (mem\text{-}pool\text{-}info\ V2\ (pool\ b))$
$[lvl\ V2\ t - Suc\ NULL := (levels\ (mem\text{-}pool\text{-}info\ V2\ (pool\ b))\ !\ (lvl\ V2\ t - Suc\ NULL))$
$(\!|bits := bits\ (levels\ (mem\text{-}pool\text{-}info\ V2\ (pool\ b))\ !\ (lvl\ V2\ t - Suc\ NULL))[bn\ V2\ t\ div\ 4 := FREEING]|\!)]\ !$
$ia))$
apply(*rule subst*[**where** $s{=}length\ (bits\ (levels\ (mem\text{-}pool\text{-}info\ V2\ (pool\ b))!ia))$
and $t{=}length\ (bits\ (levels\ (mem\text{-}pool\text{-}info\ V2\ (pool\ b))$
$[lvl\ V2\ t - Suc\ 0 := (levels\ (mem\text{-}pool\text{-}info\ V2\ (pool\ b))\ !\ (lvl\ V2\ t - Suc\ 0))$
$(\!|bits := bits\ (levels\ (mem\text{-}pool\text{-}info\ V2\ (pool\ b))\ !\ (lvl\ V2\ t - Suc\ 0))[bn\ V2\ t\ div\ 4 := FREEING]|\!)]\ !$
$ia))$])
apply(*case-tac* $ia = lvl\ V2\ t - Suc\ 0$)
apply(*case-tac* $ia < length\ (levels\ (mem\text{-}pool\text{-}info\ V2\ (pool\ b)))$)
apply *auto*[*1*] **apply** *auto*[*1*] **apply** *auto*[*1*]

apply(*case-tac* $ia = lvl\ V\ t$)
apply(*subgoal-tac* $length\ (list\text{-}updates\text{-}n$
$(bits\ (levels\ (mem\text{-}pool\text{-}info\ V\ (pool\ b))$
$[lvl\ V\ t := (levels\ (mem\text{-}pool\text{-}info\ V\ (pool\ b))\ !\ lvl\ V\ t)$
$(\!|bits := bits\ (levels\ (mem\text{-}pool\text{-}info\ V\ (pool\ b))\ !\ lvl\ V\ t)[block\ b\ div\ 4\ \hat{}\ (level\ b - lvl\ V\ t) := FREE]|\!)]\ !$
$ia))$
$(block\ b\ div\ 4\ \hat{}\ (level\ b - lvl\ V\ t)\ div\ 4 * 4)\ 4\ NOEXIST) = length\ (bits\ (levels\ (mem\text{-}pool\text{-}info\ V\ (pool\ b))$
$[lvl\ V\ t := (levels\ (mem\text{-}pool\text{-}info\ V\ (pool\ b))\ !\ lvl\ V\ t)$
$(\!|bits := bits\ (levels\ (mem\text{-}pool\text{-}info\ V\ (pool\ b))\ !\ lvl\ V\ t)[block\ b\ div\ 4\ \hat{}\ (level\ b - lvl\ V\ t) := FREE]|\!)]\ !$
$ia))$)
prefer *2* **using** *length-list-update-n* **apply** *fast*
apply(*subgoal-tac* $length\ (bits\ (levels\ (mem\text{-}pool\text{-}info\ V\ (pool\ b))$
$[lvl\ V\ t := (levels\ (mem\text{-}pool\text{-}info\ V\ (pool\ b))\ !\ lvl\ V\ t)$
$(\!|bits := bits\ (levels\ (mem\text{-}pool\text{-}info\ V\ (pool\ b))\ !\ lvl\ V\ t)[block\ b\ div\ 4\ \hat{}\ (level\ b - lvl\ V\ t) := FREE]|\!)]\ !$
$ia)) = length\ (bits\ (levels\ (mem\text{-}pool\text{-}info\ V\ (pool\ b))\ !\ ia))$)
prefer *2* **apply**(*case-tac* $ia = lvl\ V\ t$)
apply(*case-tac* $ia < length\ (levels\ (mem\text{-}pool\text{-}info\ V\ (pool\ b)))$)
apply *auto*[*1*] **apply** *auto*[*1*] **apply** *auto*[*1*]
apply *auto*[*1*]

apply(*subgoal-tac length* (*bits* (*levels* (*mem-pool-info V* (*pool b*))
[*lvl V t* := (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)
(|*bits* := *bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)[*block b div 4* ^ (*level b* − *lvl V t*) := *FREE*]|)] !
ia)) = *length* (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *ia*)))
prefer *2* **apply**(*case-tac ia* = *lvl V t*) **apply**(*case-tac ia* < *length* (*levels* (*mem-pool-info V* (*pool b*))))
apply *auto*[*1*] **apply** *auto*[*1*] **apply** *auto*[*1*]
apply *auto*[*1*]
done

lemma *mempool-free-stm8-atombody-rest-one-finalstm-h1*:
V∈*mp-free-precond8-3 t b α* ∩ {|´*cur* = *Some t*|} ⟹
{*free-stm8-precond2 V t b*} ∩ {|*NULL* < ´*lvl t* ∧ *partner-bits* (´*mem-pool-info* (*pool b*)) (´*lvl t*) (´*bn t*)|} ≠ {} ⟹
V2 ∈ *free-stm8-precond3 V t b* ∩ {|´*i t* = *4*|} ⟹
x = *free-stm8-atombody-rest-cond3* (*V2*(|*lvl* := (*lvl V2*)(*t* := *lvl V2 t* − *1*), *bn* := (*bn V2*)(*t* := *bn V2 t div 4*)|)) *t b* ⟹
x(|*freeing-node* := (*freeing-node x*) (*t* := *Some* (|*pool* = *pool b*, *level* = *lvl x t*, *block* = *bn x t*,
data = *block-ptr* (*mem-pool-info x* (*pool b*)) (*ALIGN4* (*max-sz* (*mem-pool-info x* (*pool b*))) *div 4* ^ *lvl x t*) (*bn x t*)|))|)
∈ {|´(*Pair V*) ∈ *Mem-pool-free-guar t*|}
apply(*unfold Mem-pool-free-guar-def*)
apply(*rule pairv-rId*)
apply(*rule pairv-IntI*) **apply**(*rule pairv-IntI*)

apply(*unfold gvars-conf-stable-def gvars-conf-def*)[*1*]
apply *clarify* **apply**(*simp add*:*Let-def set-bit-def*)
apply *clarify* **using** *mempool-free-stm8-atombody-rest-one-finalstm-h1-h1*[*of V t b V2*] **apply** *blast*

using *mempool-free-stm8-atombody-rest-one-finalstm-h1-2*[*of V t b α V2 x*
x(|*freeing-node* := *freeing-node x*(*t* ↦
(|*pool* = *pool b*, *level* = *lvl x t*, *block* = *bn x t*,
data = *block-ptr* (*mem-pool-info x* (*pool b*)) (*ALIGN4* (*max-sz* (*mem-pool-info x* (*pool b*))) *div 4* ^ *lvl x t*)
(*bn x t*)|))|)] **apply** *fast*

apply(*simp add*:*Let-def*)
done

lemma *mempool-free-stm8-atombody-rest-one-finalstm-I1*:
$x \in \{\!|\, ´invariant.inv \,|\!\} \Longrightarrow$
$x \in \{\!|\, ´allocating\text{-}node\ t = None \,|\!\} \Longrightarrow$
$x \in \{\!|\, ´invariant.inv \wedge ´allocating\text{-}node\ t = None \,|\!\}$
by *auto*

lemma *mempool-free-stm8-atombody-rest-one-finalstm-h3-h1*:
inv V $\wedge$
pool b $\in$ *mem-pools V2* $\wedge$
level b $<$ *length* (*levels* (*mem-pool-info V* (*pool b*))) $\wedge$ *lvl V2 t* $\leq$ *level b* $\wedge$ *NULL* $<$ *lvl V2 t* $\Longrightarrow$
mem-pools V = *mem-pools V2* $\wedge$
lvl V t = *lvl V2 t* $\Longrightarrow$
n-max (*mem-pool-info V* (*pool b*)) $*$ *4* ^ (*lvl V2 t* $-$ *Suc NULL*) =
length (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! (*lvl V2 t* $-$ *Suc NULL*)))
apply(*simp add:inv-def inv-mempool-info-def Let-def*) **apply** *auto*[*1*]
done

lemma *mempool-free-stm8-atombody-rest-one-finalstm-h3*:
invariant.inv V $\wedge$
pool b $\in$ *mem-pools V2* $\wedge$
level b $<$ *length* (*levels* (*mem-pool-info V* (*pool b*))) $\wedge$
block b $<$ *length* (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *level b*)) $\wedge$
level b $<$ *length* (*lsizes V2 t*) $\wedge$
bn V t $<$ *length* (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V2 t*)) $\wedge$
lvl V2 t $\leq$ *level b* $\wedge$
NULL $<$ *lvl V2 t* $\Longrightarrow$
mem-pools V = *mem-pools V2* $\wedge$
($\forall$ *p*.
($\forall$ *i*. *length* (*bits* (*levels* (*mem-pool-info V2 p*) ! *i*)) =
length (*bits* (*levels* (*if p* = *pool b*
then mem-pool-info V (*pool b*)
(|*levels* := *levels* (*mem-pool-info V* (*pool b*))
[*lvl V t* := (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)
(|*bits* := *bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)[*block b div 4* ^ (*level b* $-$ *lvl V t*) := *FREE*]|)]|)
else mem-pool-info V p) !
i)))) $\wedge$
($\forall$ *p*. *p* $\neq$ *pool b* $\longrightarrow$ *mem-pool-info V2 p* = *mem-pool-info V p*) $\wedge$
($\forall$ *j*. *j* $\neq$ *lvl V t* $\longrightarrow$ *levels* (*mem-pool-info V* (*pool b*)) ! *j* = *levels* (*mem-pool-info V2* (*pool b*)) ! *j*) $\wedge$
bits (*levels* (*mem-pool-info V2* (*pool b*)) ! *lvl V t*) =
list-updates-n (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)[*block b div 4* ^ (*level b* $-$ *lvl V t*) := *FREE*])
(*block b div 4* ^ (*level b* $-$ *lvl V t*) *div 4* $*$ *4*) (*i V2 t*) *NOEXIST* $\wedge$

block b div 4 ^ (*level b* $-$ *lvl V t*) = *bn V2 t* $\wedge$
lvl V t = *lvl V2 t* $\wedge$ *ALIGN4* (*max-sz* (*mem-pool-info V* (*pool b*))) *div 4* ^ *lvl V*

t = lsz V2 t ∧ *lsizes V t = lsizes V2 t* ∧ *i V2 t* ≤ *4* ∧ *i V2 t = 4* ⟹
x = V2⦇*lvl* := (*lvl V2*)(*t* := *lvl V2 t* − *Suc NULL*), *bn* := (*bn V2*)(*t* := *bn V2 t div 4*),
mem-pool-info := (*mem-pool-info V2*)
(*pool b* := *mem-pool-info V2* (*pool b*)
⦇*levels* := *levels* (*mem-pool-info V2* (*pool b*))
[*lvl V2 t* − *Suc NULL* := (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))
⦇*bits* := *bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))[*bn V2 t div 4* := *FREEING*]⦈]⦈)⦈ ⟹
bn V2 t div 4
< *length* (*bits* (*levels* (*mem-pool-info V2* (*pool b*))
[*lvl V2 t* − *Suc NULL* := (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))
⦇*bits* := *bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))[*bn V2 t div 4* := *FREEING*]⦈] !
(*lvl V2 t* − *Suc NULL*)))
apply(*rule subst*[**where** *s*= *length* (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))) **and** *t* =
length (*bits* (*levels* (*mem-pool-info V2* (*pool b*))
[*lvl V2 t* − *Suc NULL* := (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))
⦇*bits* := *bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))[*bn V2 t div 4* := *FREEING*]⦈] !
(*lvl V2 t* − *Suc NULL*)))])
apply(*subgoal-tac* ∀ *j*. *j* ≠ *lvl V t* ⟶ *levels* (*mem-pool-info V* (*pool b*)) ! *j* = *levels* (*mem-pool-info V2* (*pool b*)) ! *j*)
prefer *2* **apply** *fast*
apply(*subgoal-tac bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! *lvl V t*) =
list-updates-n (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)[*block b div 4* ˆ (*level b* − *lvl V t*) := *FREE*])
(*block b div 4* ˆ (*level b* − *lvl V t*) *div 4* ∗ *4*) (*i V2 t*) *NOEXIST*)
prefer *2* **apply** *fast*
using *mempool-free-stm8-atombody-rest-one-finalstm-h1-h1*[*of V t b V2* (*lvl V2 t* − *Suc NULL*)] **apply** *auto*[*1*]

apply(*rule subst*[**where** *s*=(*n-max* (*mem-pool-info V* (*pool b*))) ∗ *4* ˆ (*lvl V2 t* − *Suc 0*)
and *t*=*length* (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! (*lvl V2 t* − *Suc 0*)))])
using *mempool-free-stm8-atombody-rest-one-finalstm-h3-h1* **apply** *fast*

apply(*subgoal-tac length* (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *level b*))
= (*n-max* (*mem-pool-info V* (*pool b*))) ∗ *4* ˆ *level b*)
prefer *2* **apply**(*simp add*:*inv-def inv-mempool-info-def Let-def*)

apply(*rule lm11*[*of V2 t b V*]) **apply** *simp* **apply** *simp* **apply** *metis*
done

lemma *mempool-free-stm8-atombody-rest-one-finalstm-h4*:
$(\neg$ *free-block-r V2 t* $\longrightarrow$ *freeing-node V t* $=$ *None*$)$ $\wedge$
$\alpha = ($*if* $\exists y$. *freeing-node V t* $=$ *Some y then lvl V t* $+$ *1 else NULL*$)$ $\wedge$
$V \in ($*if NULL* $< \alpha$ *then UNIV else* $\{\})$ $\Longrightarrow$ *free-block-r V2 t*
apply *auto*
done

lemma *mempool-free-stm8-atombody-rest-one-finalstm*:
$V \in$*mp-free-precond8-3 t b* $\alpha \cap \{\!|$ ´*cur* $=$ *Some t*$|\!\} \Longrightarrow$
$\{$*free-stm8-precond2 V t b*$\} \cap \{\!|$*NULL* $<$ ´*lvl t* $\wedge$ *partner-bits* (´*mem-pool-info* (*pool b*)) (´*lvl t*) (´*bn t*)$|\!\} \neq \{\} \Longrightarrow$
$V2 \in$ *free-stm8-precond3 V t b* $\cap \{\!|$´*i t* $= 4|\!\} \Longrightarrow$
$\{$*free-stm8-atombody-rest-cond3* (*V2*(|*lvl* := (*lvl V2*)(*t* := *lvl V2 t* $-$ *1*), *bn* := (*bn V2*)(*t* := *bn V2 t div 4*)|)) *t b*$\}$
$\subseteq \{\!|$´(*freeing-node-update*
(λ-. ´*freeing-node*(*t* $\mapsto$
(|*pool* = *pool b*, *level* = ´*lvl t*, *block* = ´*bn t*,
data = *block-ptr* (´*mem-pool-info* (*pool b*)) (*ALIGN4* (*max-sz* (´*mem-pool-info* (*pool b*))) *div 4* ^ ´*lvl t*) (´*bn t*)|))))
$\in \{\!|$´(*Pair V*) $\in$ *Mem-pool-free-guar t*$|\!\} \cap$ *mp-free-precond8-inv t b* $(\alpha - 1)|\!\}$

apply(*rule subsetI*)
apply(*subgoal-tac x* = *free-stm8-atombody-rest-cond3* (*V2*(|*lvl* := (*lvl V2*)(*t* := *lvl V2 t* $-$ *1*), *bn* := (*bn V2*)(*t* := *bn V2 t div 4*)|)) *t b*)
prefer *2* **apply** *fast*
apply(*subgoal-tac x*(|*freeing-node* := (*freeing-node x*) (*t* := *Some* (|*pool* = *pool b*, *level* = *lvl x t*, *block* = *bn x t*,
data = *block-ptr* (*mem-pool-info x* (*pool b*)) (*ALIGN4* (*max-sz* (*mem-pool-info x* (*pool b*))) *div 4* ^ *lvl x t*) (*bn x t*)|))|)
$\in \{\!|$´(*Pair V*) $\in$ *Mem-pool-free-guar t*$|\!\} \cap$ *mp-free-precond8-inv t b* $(\alpha - 1)$)
apply *blast*

apply(*rule IntI*)

using *mempool-free-stm8-atombody-rest-one-finalstm-h1*[*of V t b* α *V2*] **apply** *meson*

apply(*rule IntI*)
apply(*rule IntI*)
apply(*rule IntI*)

apply(*rule mempool-free-stm8-atombody-rest-one-finalstm-I1*)
using *mempool-free-stm8-atombody-rest-one-finalstm-inv*[*of V t b* α *V2*] **apply** *meson*
using *mempool-free-stm8-atombody-rest-one-finalstm-h2*[*of V t b* α *V2*] **apply** *meson*

apply(*simp add:Let-def gvars-conf-stable-def gvars-conf-def block-ptr-def set-bit-def*)
apply(*subgoal-tac length* (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *level b*)) =
length (*bits* (*levels* (*mem-pool-info V2* (*pool b*))
[*lvl V2 t* − *Suc NULL* := (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))
(|*bits* := *bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! (*lvl V2 t* − *Suc NULL*))[*bn V2 t div 4* := *FREEING*]|)] !
level b)))
prefer *2* **apply**(*subgoal-tac* ∀ *j*. *j* ≠ *lvl V t* ⟶ *levels* (*mem-pool-info V* (*pool b*)) ! *j*
= *levels* (*mem-pool-info V2* (*pool b*)) ! *j*) **prefer** *2* **apply** *fast*
apply(*subgoal-tac bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! *lvl V t*) =
list-updates-n (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)[*block b div 4* ˆ (*level b* − *lvl V t*) := *FREE*])
(*block b div 4* ˆ (*level b* − *lvl V t*) *div 4* ∗ *4*) (*i V2 t*) *NOEXIST*)
prefer *2* **apply** *fast*
using *mempool-free-stm8-atombody-rest-one-finalstm-h1-h1*[*of V t b V2 level b*]
apply *argo*
apply *auto*[*1*]

apply(*simp add:Let-def gvars-conf-stable-def gvars-conf-def block-ptr-def set-bit-def*)
apply(*rule conjI*)
using *mempool-free-stm8-atombody-rest-one-finalstm-h3* **apply** *blast*

apply(*rule conjI*)
apply(*rule subst*[**where** *s*=*lvl V t* **and** *t*=*lvl V2 t*]) **apply** *fast*
apply (*metis Nat.add-diff-assoc div-mult2-eq plus-1-eq-Suc power-add power-commutes power-one-right*)

apply(*rule conjI*)
apply (*metis Suc-pred le-imp-less-Suc nat-le-linear not-less*)

apply(*rule conjI*)
apply(*rule subst*[**where** *s*=*max-sz* (*mem-pool-info V* (*pool b*)) **and** *t*=*ALIGN4* (*max-sz* (*mem-pool-info V* (*pool b*)))])
using *inv-maxsz-align4* **apply** *auto*[*1*]
apply *clarify* **apply**(*rule conjI*)
apply *fast*

apply(*subgoal-tac length* (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *level b*)) = (*n-max* (*mem-pool-info V* (*pool b*))) ∗ *4* ˆ *level b*)
prefer *2* **apply**(*simp add:inv-def inv-mempool-info-def Let-def*)
apply(*subgoal-tac level b* > *0*) **prefer** *2* **apply** *auto*[*1*]
apply(*subgoal-tac block b* < *n-max* (*mem-pool-info V* (*pool b*)) ∗ *4* ˆ *level b*)
prefer *2* **apply** *argo*
apply(*subgoal-tac block b div 4* ˆ (*level b* − *lvl V2 t*) = *bn V2 t*) **prefer** *2* **apply** *metis*
using *lm11*[*of V2 t b V*] **apply** *meson*

using *mempool-free-stm8-atombody-rest-one-finalstm-h4*[*of V2 t V*] **apply** *fast*

apply(*simp add*:*Let-def*)
apply *clarsimp*
apply *auto*
done

term *free-stm8-precond2 V t b*

lemma *mempool-free-stm8-atombody-rest-one*:
$V \in mp\text{-}free\text{-}precond8\text{-}3\ t\ b\ \alpha \cap \{\!\!\{\acute{}cur = Some\ t\}\!\!\} \Longrightarrow$
$\{free\text{-}stm8\text{-}precond2\ V\ t\ b\} \cap \{\!\!\{NULL < \acute{}lvl\ t \wedge partner\text{-}bits\ (\acute{}mem\text{-}pool\text{-}info\ (pool\ b))\ (\acute{}lvl\ t)\ (\acute{}bn\ t)\}\!\!\} \neq \{\} \Longrightarrow$
$V2 \in free\text{-}stm8\text{-}precond3\ V\ t\ b \cap \{\!\!\{\acute{}i\ t = 4\}\!\!\} \Longrightarrow$
$\Gamma \vdash_I Some\ (\acute{}lvl := \acute{}lvl(t := \acute{}lvl\ t - 1);;$
$\acute{}bn := \acute{}bn(t := \acute{}bn\ t\ div\ 4);;$
$\acute{}mem\text{-}pool\text{-}info := set\text{-}bit\text{-}freeing\ \acute{}mem\text{-}pool\text{-}info\ (pool\ b)\ (\acute{}lvl\ t)\ (\acute{}bn\ t);;$
$\acute{}freeing\text{-}node := \acute{}freeing\text{-}node(t \mapsto (\!|pool = pool\ b,\ level = \acute{}lvl\ t,\ block = \acute{}bn\ t,$
$data = block\text{-}ptr\ (\acute{}mem\text{-}pool\text{-}info\ (pool\ b))\ (ALIGN4\ (max\text{-}sz\ (\acute{}mem\text{-}pool\text{-}info\ (pool\ b)))\ div\ 4\ \hat{}\ \acute{}lvl\ t)\ (\acute{}bn\ t)|\!)))$
$sat_p\ [\{V2\},\ \{(s,\ t).\ s = t\},\ UNIV,$
$\{\!\!\{\acute{}(Pair\ V) \in Mem\text{-}pool\text{-}free\text{-}guar\ t\}\!\!\} \cap mp\text{-}free\text{-}precond8\text{-}inv\ t\ b\ (\alpha - 1)]$
apply(*rule Seq*[**where** *mid*={*free-stm8-atombody-rest-cond3* (*free-stm8-atombody-rest-cond2* (*free-stm8-atombody-rest-cond1 V2 t b*) *t b*) *t b*}])
apply(*rule Seq*[**where** *mid*={*free-stm8-atombody-rest-cond2* (*free-stm8-atombody-rest-cond1 V2 t b*) *t b*}])
apply(*rule Seq*[**where** *mid*={*free-stm8-atombody-rest-cond1 V2 t b*}])

apply(*rule Basic*)
apply *fast* **apply** *fast* **using** *stable-id2* **apply** *fast* **using** *stable-id2* **apply** *fast*

apply(*rule Basic*)
apply *fast* **apply** *fast* **using** *stable-id2* **apply** *fast* **using** *stable-id2* **apply** *fast*

apply(*rule Basic*)
apply(*simp add*:*set-bit-def Let-def*) **apply** *fast* **using** *stable-id2* **apply** *fast* **using** *stable-id2* **apply** *fast*

apply(*rule Basic*)
apply(*rule subst*[**where** *s*=*bn V2* **and** *t*=*bn* (*V2*(|*lvl* := (*lvl V2*)(*t* := *lvl V2 t* − *1*)|))])
apply *auto*[*1*]
using *mempool-free-stm8-atombody-rest-one-finalstm*[*of V t b α V2*] **apply** *meson*

apply *fast* **using** *stable-id2* **apply** *fast* **using** *stable-id2* **apply** *fast*
done

lemma *mempool-free-stm8-atombody-rest*:
$V \in mp\text{-}free\text{-}precond8\text{-}3\ t\ b\ \alpha \cap \{\!|\ ´cur = Some\ t|\!\} \Longrightarrow$
$\{free\text{-}stm8\text{-}precond2\ V\ t\ b\} \cap \{\!|NULL < ´lvl\ t \wedge partner\text{-}bits\ (´mem\text{-}pool\text{-}info\ (pool\ b))\ (´lvl\ t)\ (´bn\ t)|\!\} \neq \{\} \Longrightarrow$
$\Gamma \vdash_I Some\ (´lvl := ´lvl(t := ´lvl\ t - 1);;$
$´bn := ´bn(t := ´bn\ t\ div\ 4);;$
$´mem\text{-}pool\text{-}info := set\text{-}bit\text{-}freeing\ ´mem\text{-}pool\text{-}info\ (pool\ b)\ (´lvl\ t)\ (´bn\ t);;$
$´freeing\text{-}node := ´freeing\text{-}node(t \mapsto (\!|pool = pool\ b,\ level = ´lvl\ t,\ block = ´bn\ t,$
$data = block\text{-}ptr\ (´mem\text{-}pool\text{-}info\ (pool\ b))\ (ALIGN4\ (max\text{-}sz\ (´mem\text{-}pool\text{-}info\ (pool\ b)))\ div\ 4\ \hat{}\ ´lvl\ t)\ (´bn\ t)|\!)))$
$sat_p\ [free\text{-}stm8\text{-}precond3\ V\ t\ b \cap \{\!|´i\ t = 4|\!\},\ \{(s, t).\ s = t\},\ UNIV,$
$\{\!|´(Pair\ V) \in Mem\text{-}pool\text{-}free\text{-}guar\ t|\!\} \cap mp\text{-}free\text{-}precond8\text{-}inv\ t\ b\ (\alpha - 1)]$
using *mempool-free-stm8-atombody-rest-one*[*of V t b* α]
Allprecond[**where** $U=free\text{-}stm8\text{-}precond3\ V\ t\ b \cap \{\!|´i\ t = 4|\!\}$ **and**
$P=Some\ (´lvl := ´lvl(t := ´lvl\ t - 1);;$
$´bn := ´bn(t := ´bn\ t\ div\ 4);;$
$´mem\text{-}pool\text{-}info := set\text{-}bit\text{-}freeing\ ´mem\text{-}pool\text{-}info\ (pool\ b)\ (´lvl\ t)\ (´bn\ t);;$
$´freeing\text{-}node := ´freeing\text{-}node(t \mapsto (\!|pool = pool\ b,\ level = ´lvl\ t,\ block = ´bn\ t,$
$data = block\text{-}ptr\ (´mem\text{-}pool\text{-}info\ (pool\ b))\ (ALIGN4\ (max\text{-}sz\ (´mem\text{-}pool\text{-}info\ (pool\ b)))\ div\ 4\ \hat{}\ ´lvl\ t)\ (´bn\ t)|\!)))$ **and**
$rely=\{(x, y).\ x = y\}$ **and**
$guar= UNIV$ **and** $post= \{\!|´(Pair\ V) \in Mem\text{-}pool\text{-}free\text{-}guar\ t|\!\} \cap mp\text{-}free\text{-}precond8\text{-}inv\ t\ b\ (\alpha - 1)]$
apply *meson*
done

abbreviation $free\text{-}stm8\text{-}bd2\text{-}cond1\ V\ t\ b \equiv V(\!|j := (j\ V)(t := lvl\ V\ t)|\!)$
abbreviation $free\text{-}stm8\text{-}bd2\text{-}cond2\ V\ t\ b \equiv V(\!|lbn := (lbn\ V)(t := bn\ V\ t)|\!)$
abbreviation $free\text{-}stm8\text{-}bd2\text{-}cond3\ V\ t\ b \equiv V(\!|lvl := (lvl\ V)(t := j\ V\ t - 1)|\!)$
abbreviation $free\text{-}stm8\text{-}bd2\text{-}cond4\ V\ t\ b \equiv V(\!|bn := (bn\ V)(t := lbn\ V\ t\ div\ 4)|\!)$
abbreviation $free\text{-}stm8\text{-}bd2\text{-}cond5\ V\ t\ b \equiv$
$let\ minf = mem\text{-}pool\text{-}info\ V\ (pool\ b)\ in$
$V(\!|mem\text{-}pool\text{-}info := (mem\text{-}pool\text{-}info\ V)\ (pool\ b := minf\ (\!|levels := (levels\ minf)$

$[lvl\ V\ t := ((levels\ minf)\ !\ (lvl\ V\ t))\ (\!|bits := (bits\ ((levels\ minf)\ !\ (lvl\ V\ t)))\ [bn\ V\ t := FREEING]|\!)]\ |\!))|\!)$

lemma *mempool-free-stm8-atombody-else-blockfit*:
$V \in mp\text{-}free\text{-}precond8\text{-}3\ t\ b\ \alpha \cap \{\!|´cur = Some\ t|\!\} \Longrightarrow$
$free\text{-}stm8\text{-}precond2\ V\ t\ b \in \{\!|block\text{-}fits\ (´mem\text{-}pool\text{-}info\ (pool\ b))\ (´blk\ t)\ (´lsz\ t)|\!\}$
apply(*simp add*:*block-fits-def block-ptr-def buf-size-def set-bit-def*)
apply(*rule subst*[**where** $s=max\text{-}sz\ (mem\text{-}pool\text{-}info\ V\ (pool\ b))$ **and** $t=ALIGN4$

(*max-sz* (*mem-pool-info* *V* (*pool* *b*)))])
apply(*simp add*: *inv-def*) **using** *inv-mempool-info-maxsz-align4*[*rule-format*,*of* *V* *pool* *b*] **apply** *metis*

apply(*subgoal-tac* *length* (*bits* ((*levels* (*mem-pool-info* *V* (*pool* *b*))) ! *level* *b*)) = (*n-max* (*mem-pool-info* *V* (*pool* *b*))) ∗ *4* ˆ (*level* *b*))
prefer *2* **apply**(*simp add*: *inv-def* *inv-mempool-info-def* *Let-def*)

apply(*subgoal-tac* *max-sz* (*mem-pool-info* *V* (*pool* *b*)) *mod* *4* ˆ *lvl* *V* *t* = *0*)
prefer *2* **apply**(*subgoal-tac* ∃ *n*. *max-sz* (*mem-pool-info* *V* (*pool* *b*)) = (*4* ∗ *n*) ∗ (*4* ˆ *n-levels* (*mem-pool-info* *V* (*pool* *b*))))
prefer *2* **apply**(*simp add*:*inv-def*) **using** *inv-mempool-info-def*[*rule-format*, *of* *V*] **apply** *meson*
apply(*subgoal-tac* *length* (*levels* (*mem-pool-info* *V* (*pool* *b*))) = *n-levels* (*mem-pool-info* *V* (*pool* *b*)))
prefer *2* **apply**(*simp add*:*inv-def* *inv-mempool-info-def*) **apply** *metis*
apply(*simp add*: *inv-def* *inv-mempool-info-def*)
using *ge-pow-mod-0*[*of* *lvl* *V* *t* *n-levels* (*mem-pool-info* *V* (*pool* *b*))]
apply (*metis* *add-diff-inverse-nat* *add-lessD1* *ge-pow-mod-0* *le-antisym* *nat-less-le*)

apply(*subgoal-tac* *block* *b* *div* *4* ˆ (*level* *b* − *lvl* *V* *t*) < *n-max* (*mem-pool-info* *V* (*pool* *b*)) ∗ *4* ˆ *lvl* *V* *t*)
prefer *2* **apply** (*metis* (*no-types*, *lifting*) *add-lessD1* *inv-mempool-info-def* *invariant.inv-def* *le-Suc-ex*)

apply(*rule* *block-fits0-h1*[*of* *max-sz* (*mem-pool-info* *V* (*pool* *b*)) *4* ˆ *lvl* *V* *t* *block* *b* *div* *4* ˆ (*level* *b* − *lvl* *V* *t*) *n-max* (*mem-pool-info* *V* (*pool* *b*))])
apply *blast* **apply** *blast*
done

lemma *mempool-free-stm8-atombody-else-inv-mempool-info*:
inv-mempool-info *V* ⟹
inv-mempool-info
(*V*⦇*freeing-node* := (*freeing-node* *V*)(*t* := *None*),
mem-pool-info := (*set-bit-free* (*mem-pool-info* *V*) (*pool* *b*) (*lvl* *V* *t*) (*block* *b* *div* *4* ˆ (*level* *b* − *lvl* *V* *t*)))
(*pool* *b* := *append-free-list* (*set-bit-free* (*mem-pool-info* *V*) (*pool* *b*) (*lvl* *V* *t*) (*block* *b* *div* *4* ˆ (*level* *b* − *lvl* *V* *t*)) (*pool* *b*)) (*lvl* *V* *t*)
(*block-ptr* (*mem-pool-info* *V* (*pool* *b*)) (*ALIGN4* (*max-sz* (*mem-pool-info* *V* (*pool* *b*))) *div* *4* ˆ *lvl* *V* *t*) (*block* *b* *div* *4* ˆ (*level* *b* − *lvl* *V* *t*)))),
free-block-r := (*free-block-r* *V*)(*t* := *False*)⦈)
apply(*simp add*:*inv-mempool-info-def* *append-free-list-def* *set-bit-def*) **apply** *clarify*
apply(*rule* *conjI*) **apply** *meson*
apply(*rule* *conjI*) **apply** *meson*
apply(*rule* *conjI*) **apply** *meson*
apply(*rule* *conjI*) **apply** *meson*
apply(*rule* *conjI*) **apply** *meson*
apply *clarify*

apply(*subgoal-tac* (∀ *i*<*length* (*levels* (*mem-pool-info V* (*pool b*))).
length (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *i*)) = *n-max* (*mem-pool-info V* (*pool b*)) ∗ *4* ˆ *i*))
prefer *2* **apply**(*simp add*:*Let-def*)
apply(*case-tac i* = *lvl V t*)
by *auto*

lemma *mempool-free-stm8-atombody-else-inv-bitmap-freelist*:
inv-mempool-info V ∧ *inv-bitmap-freelist V* ∧ *inv-aux-vars V* ⟹
level b < *length* (*levels* (*mem-pool-info V* (*pool b*))) ⟹
block b < *length* (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *level b*)) ⟹
block b div 4 ˆ (*level b* − *lvl V t*) < *length* (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)) ⟹
lvl V t ≤ *level b* ⟹
freeing-node V t = *Some blka* ⟹
pool blka = *pool b* ⟹
level blka = *lvl V t* ⟹
block blka = *block b div 4* ˆ (*level b* − *lvl V t*) ⟹
inv-bitmap-freelist
(*V*⦇*freeing-node* := (*freeing-node V*)(*t* := *None*),
mem-pool-info := (*set-bit-free* (*mem-pool-info V*) (*pool b*) (*lvl V t*) (*block b div 4* ˆ (*level b* − *lvl V t*)))
(*pool b* := *append-free-list* (*set-bit-free* (*mem-pool-info V*) (*pool b*) (*lvl V t*) (*block b div 4* ˆ (*level b* − *lvl V t*)) (*pool b*)) (*lvl V t*)
(*block-ptr* (*mem-pool-info V* (*pool b*)) (*ALIGN4* (*max-sz* (*mem-pool-info V* (*pool b*))) *div 4* ˆ *lvl V t*) (*block b div 4* ˆ (*level b* − *lvl V t*)))),
free-block-r := (*free-block-r V*)(*t* := *False*)⦈)
apply(*simp add*:*inv-bitmap-freelist-def append-free-list-def set-bit-def block-ptr-def*)
apply *clarify*
apply(*simp add*:*Let-def*)
apply(*rule subst*[**where** *s*=*max-sz* (*mem-pool-info V* (*pool b*)) **and** *t*=*ALIGN4* (*max-sz* (*mem-pool-info V* (*pool b*)))])
apply (*metis inv-mempool-info-maxsz-align4*)

apply(*rule conjI*) **apply** *clarify* **apply**(*rename-tac ii jj*)
apply(*case-tac ii* ≠ *lvl V t*) **apply** *force*
apply(*case-tac jj* = *block b div 4* ˆ (*level b* − *lvl V t*))
apply *clarsimp*

apply(*subgoal-tac bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *ii*) ! *jj* = *bits* (*levels* (*mem-pool-info V* (*pool b*))
[*lvl V t* := (*levels* (*mem-pool-info V* (*pool b*))
[*lvl V t* := (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)
⦇*bits* := *bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)[*block b div 4* ˆ (*level b* − *lvl V t*) := *FREE*]⦈] ! *lvl V t*)
⦇*free-list* := *free-list* (*levels* (*mem-pool-info V* (*pool b*)) [*lvl V t* := (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)

(|*bits* := *bits* (*levels* (*mem-pool-info* *V* (*pool* *b*)) ! *lvl* *V* *t*)[*block* *b* *div* *4* ˆ (*level* *b* − *lvl* *V* *t*) := *FREE*]|)] ! *lvl* *V* *t*) @
[*buf* (*mem-pool-info* *V* (*pool* *b*)) + *ALIGN4* (*max-sz* (*mem-pool-info* *V* (*pool* *b*))) *div* *4* ˆ *lvl* *V* *t* ∗ (*block* *b* *div* *4* ˆ (*level* *b* − *lvl* *V* *t*))]|)] !
ii) ! *jj*)
prefer *2* **apply** *fastforce*
apply(*subgoal-tac* *length* (*bits* (*levels* (*mem-pool-info* *V* (*pool* *b*))!*ii*)) = *length* (*bits* (*levels* (*mem-pool-info* *V* (*pool* *b*))
[*lvl* *V* *t* := (*levels* (*mem-pool-info* *V* (*pool* *b*)) [*lvl* *V* *t* := (*levels* (*mem-pool-info* *V* (*pool* *b*)) ! *lvl* *V* *t*)
(|*bits* := *bits* (*levels* (*mem-pool-info* *V* (*pool* *b*)) ! *lvl* *V* *t*)[*block* *b* *div* *4* ˆ (*level* *b* − *lvl* *V* *t*) := *FREE*]|)] ! *lvl* *V* *t*)
(|*free-list* := *free-list* (*levels* (*mem-pool-info* *V* (*pool* *b*)) [*lvl* *V* *t* := (*levels* (*mem-pool-info* *V* (*pool* *b*)) ! *lvl* *V* *t*)
(|*bits* := *bits* (*levels* (*mem-pool-info* *V* (*pool* *b*)) ! *lvl* *V* *t*)[*block* *b* *div* *4* ˆ (*level* *b* − *lvl* *V* *t*) := *FREE*]|)] ! *lvl* *V* *t*) @
[*buf* (*mem-pool-info* *V* (*pool* *b*)) + *max-sz* (*mem-pool-info* *V* (*pool* *b*)) *div* *4* ˆ *lvl* *V* *t* ∗ (*block* *b* *div* *4* ˆ (*level* *b* − *lvl* *V* *t*))]|)] ! *ii*)))
prefer *2* **apply** *fastforce*
apply(*subgoal-tac* *free-list* (*levels* (*mem-pool-info* *V* (*pool* *b*)) ! *ii*) @
[*buf* (*mem-pool-info* *V* (*pool* *b*)) +
max-sz (*mem-pool-info* *V* (*pool* *b*)) *div* *4* ˆ *lvl* *V* *t* ∗ (*block* *b* *div* *4* ˆ (*level* *b* − *lvl* *V* *t*))] = *free-list* (*levels* (*mem-pool-info* *V* (*pool* *b*))
[*lvl* *V* *t* := (*levels* (*mem-pool-info* *V* (*pool* *b*))
[*lvl* *V* *t* := (*levels* (*mem-pool-info* *V* (*pool* *b*)) ! *lvl* *V* *t*)
(|*bits* := *bits* (*levels* (*mem-pool-info* *V* (*pool* *b*)) ! *lvl* *V* *t*)[*block* *b* *div* *4* ˆ (*level* *b* − *lvl* *V* *t*) := *FREE*]|)] !
lvl *V* *t*)
(|*free-list* := *free-list* (*levels* (*mem-pool-info* *V* (*pool* *b*)) [*lvl* *V* *t* := (*levels* (*mem-pool-info* *V* (*pool* *b*)) ! *lvl* *V* *t*)
(|*bits* := *bits* (*levels* (*mem-pool-info* *V* (*pool* *b*)) ! *lvl* *V* *t*)[*block* *b* *div* *4* ˆ (*level* *b* − *lvl* *V* *t*) := *FREE*]|)] ! *lvl* *V* *t*) @
[*buf* (*mem-pool-info* *V* (*pool* *b*)) +
max-sz (*mem-pool-info* *V* (*pool* *b*)) *div* *4* ˆ *lvl* *V* *t* ∗ (*block* *b* *div* *4* ˆ (*level* *b* − *lvl* *V* *t*))]|)] ! *ii*))
prefer *2* **apply** *clarsimp*
apply(*case-tac* *bits* (*levels* (*mem-pool-info* *V* (*pool* *b*)) ! *ii*) ! *jj* = *FREE*)
apply(*subgoal-tac* (*buf* (*mem-pool-info* *V* (*pool* *b*)) + *jj* ∗ (*max-sz* (*mem-pool-info* *V* (*pool* *b*)) *div* *4* ˆ *ii*)
∈ *set* (*free-list* (*levels* (*mem-pool-info* *V* (*pool* *b*)) ! *ii*))))
prefer *2* **apply** *simp*
apply *clarsimp*

apply(*subgoal-tac* (*buf* (*mem-pool-info* *V* (*pool* *b*)) + *jj* ∗ (*max-sz* (*mem-pool-info* *V* (*pool* *b*)) *div* *4* ˆ *ii*)
∉ *set* (*free-list* (*levels* (*mem-pool-info* *V* (*pool* *b*)) ! *ii*))))
prefer *2* **apply** *simp*
apply(*subgoal-tac* *buf* (*mem-pool-info* *V* (*pool* *b*)) + *max-sz* (*mem-pool-info*

V $(pool\ b))\ div\ 4\ \hat{}\ lvl\ V\ t * (block\ b\ div\ 4\ \hat{}\ (level\ b - lvl\ V\ t))$
$\neq buf\ (mem\text{-}pool\text{-}info\ V\ (pool\ b)) + jj * (max\text{-}sz\ (mem\text{-}pool\text{-}info$
$V\ (pool\ b))\ div\ 4\ \hat{}\ ii))$
prefer *2* **apply**(*subgoal-tac max-sz* (*mem-pool-info V* (*pool b*)) *div 4 ˆ lvl*
V t > *0*)
prefer *2* **apply**(*simp add*:*inv-mempool-info-def Let-def*)
apply(*subgoal-tac* ∃ *n*>*NULL*. *max-sz* (*mem-pool-info V* (*pool*
b)) = *4* ∗ *n* ∗ *4* ˆ *n-levels* (*mem-pool-info V* (*pool b*)))
prefer *2* **apply** *auto*[*1*]
apply(*subgoal-tac lvl V t* < *n-levels* (*mem-pool-info V* (*pool b*)))
prefer *2* **apply** *auto*[*1*]
apply(*metis divisors-zero ge-pow-mod-0 gr0I mod0-div-self*
mult-0-right power-not-zero zero-neq-numeral)
apply *auto*[*1*]

apply(*subgoal-tac buf* (*mem-pool-info V* (*pool b*)) + *jj* ∗ (*max-sz* (*mem-pool-info*
V (*pool b*)) *div 4* ˆ *ii*)
∉ *set* (*free-list* (*levels* (*mem-pool-info V* (*pool b*)) ! *ii*) @
[*buf* (*mem-pool-info V* (*pool b*)) +
max-sz (*mem-pool-info V* (*pool b*)) *div 4* ˆ *lvl V t*
∗ (*block b div 4* ˆ (*level b* − *lvl V t*))]))
prefer *2* **apply** *auto*[*1*]

apply *auto*[*1*]

apply(*rule conjI*)
apply *clarify* **apply**(*rename-tac ii jj*)
apply(*case-tac ii* ≠ *lvl V t*) **apply** *force*
apply(*subgoal-tac free-list* (*levels* (*mem-pool-info V* (*pool b*)) ! *ii*) @
[*buf* (*mem-pool-info V* (*pool b*)) +
max-sz (*mem-pool-info V* (*pool b*)) *div 4* ˆ *lvl V t* ∗
(*block b div 4* ˆ (*level b* − *lvl V t*))] = *free-list* (*levels* (*mem-pool-info V* (*pool b*))
[*lvl V t* := (*levels* (*mem-pool-info V* (*pool b*))
[*lvl V t* := (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)
(|*bits* := *bits* (*levels* (*mem-pool-info V* (*pool b*)) !
lvl V t)[*block b div 4* ˆ (*level b* − *lvl V t*) := *FREE*]|)] !
lvl V t)
(|*free-list* := *free-list* (*levels* (*mem-pool-info V* (*pool*
b)) [*lvl V t* := (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)
(|*bits* := *bits* (*levels* (*mem-pool-info V* (*pool*
b)) ! *lvl V t*)[*block b div 4* ˆ (*level b* − *lvl V t*) := *FREE*]|)] ! *lvl V t*) @
[*buf* (*mem-pool-info V* (*pool b*)) +
max-sz (*mem-pool-info V* (*pool b*)) *div 4* ˆ *lvl V t*
∗ (*block b div 4* ˆ (*level b* − *lvl V t*))]|)] ! *ii*))
prefer *2* **apply** *clarsimp*
apply(*case-tac jj* < *length* (*free-list* (*levels* (*mem-pool-info V* (*pool b*)) ! *ii*)))
apply (*simp add*: *nth-append*)
apply(*case-tac jj* = *length* (*free-list* (*levels* (*mem-pool-info V* (*pool b*)) ! *ii*)))

apply(*subgoal-tac* (*free-list* (*levels* (*mem-pool-info* *V* (*pool* *b*)) ! *ii*) @
[*buf* (*mem-pool-info* *V* (*pool* *b*)) +
max-sz (*mem-pool-info* *V* (*pool* *b*)) *div* *4* ^ *lvl* *V* *t* ∗ (*block* *b* *div* *4* ^ (*level* *b* − *lvl* *V* *t*))]) ! *jj*
= *buf* (*mem-pool-info* *V* (*pool* *b*)) +
max-sz (*mem-pool-info* *V* (*pool* *b*)) *div* *4* ^ *lvl* *V* *t* ∗ (*block* *b* *div* *4* ^ (*level* *b* − *lvl* *V* *t*)))
prefer *2* **apply** *clarsimp*
apply(*subgoal-tac* *block* *b* *div* *4* ^ (*level* *b* − *lvl* *V* *t*) < *n-max* (*mem-pool-info* *V* (*pool* *b*)) ∗ *4* ^ *ii*)
prefer *2* **apply** (*metis* *inv-mempool-info-def*)
apply *auto*[*1*]
apply *auto*[*1*]

apply(*subgoal-tac* *distinct* (*free-list* (*levels* (*mem-pool-info* *V* (*pool* *b*)) ! *i*)))
prefer *2* **apply** *auto*[*1*] **apply**(*rename-tac* *ii*)
apply(*case-tac* *ii* ≠ *lvl* *V* *t*) **apply** *force*
apply(*subgoal-tac* *free-list* (*levels* (*mem-pool-info* *V* (*pool* *b*)) ! *ii*) @
[*buf* (*mem-pool-info* *V* (*pool* *b*)) +
max-sz (*mem-pool-info* *V* (*pool* *b*)) *div* *4* ^ *lvl* *V* *t* ∗ (*block* *b* *div* *4* ^ (*level* *b* − *lvl* *V* *t*))] = *free-list* (*levels* (*mem-pool-info* *V* (*pool* *b*))
[*lvl* *V* *t* := (*levels* (*mem-pool-info* *V* (*pool* *b*))
[*lvl* *V* *t* := (*levels* (*mem-pool-info* *V* (*pool* *b*)) ! *lvl* *V* *t*)
(|*bits* := *bits* (*levels* (*mem-pool-info* *V* (*pool* *b*)) ! *lvl* *V* *t*)[*block* *b* *div* *4* ^ (*level* *b* − *lvl* *V* *t*) := *FREE*]|)] !
lvl *V* *t*)
(|*free-list* := *free-list* (*levels* (*mem-pool-info* *V* (*pool* *b*)) [*lvl* *V* *t* := (*levels* (*mem-pool-info* *V* (*pool* *b*)) ! *lvl* *V* *t*)
(|*bits* := *bits* (*levels* (*mem-pool-info* *V* (*pool* *b*)) ! *lvl* *V* *t*)[*block* *b* *div* *4* ^ (*level* *b* − *lvl* *V* *t*) := *FREE*]|)] ! *lvl* *V* *t*) @
[*buf* (*mem-pool-info* *V* (*pool* *b*)) +
max-sz (*mem-pool-info* *V* (*pool* *b*)) *div* *4* ^ *lvl* *V* *t* ∗ (*block* *b* *div* *4* ^ (*level* *b* − *lvl* *V* *t*))]|)] ! *ii*))
prefer *2* **apply** *clarsimp*

apply(*subgoal-tac* *buf* (*mem-pool-info* *V* (*pool* *b*)) +
max-sz (*mem-pool-info* *V* (*pool* *b*)) *div* *4* ^ *lvl* *V* *t* ∗ (*block* *b* *div* *4* ^ (*level* *b* − *lvl* *V* *t*))
∉ *set* (*free-list* (*levels* (*mem-pool-info* *V* (*pool* *b*)) ! *ii*)))
prefer *2* **apply**(*subgoal-tac* *get-bit* (*mem-pool-info* *V*) (*pool* *b*) (*lvl* *V* *t*) (*block* *b* *div* *4* ^ (*level* *b* − *lvl* *V* *t*)) = *FREEING*)
prefer *2* **apply**(*simp* *add*:*inv-aux-vars-def*) **apply** *metis*
apply (*metis* *BlockState*.*distinct*(*15*) *semiring-normalization-rules*(*7*))
apply *auto*
done

lemma
pool *b* ∈ *mem-pools* *V* ⟹

lvl V t $\leq$ *level b* $\Longrightarrow$
level b $<$ *length* (*levels* (*mem-pool-info V* (*pool b*))) $\Longrightarrow$
block b div 4 ^ (*level b* $-$ *lvl V t*) $<$ *length* (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)) $\Longrightarrow$
V2 = *V*(|*freeing-node* := (*freeing-node V*)(*t* := *None*),
mem-pool-info := (*set-bit-free* (*mem-pool-info V*) (*pool b*) (*lvl V t*) (*block b div 4* ^ (*level b* $-$ *lvl V t*)))
(*pool b* := *append-free-list* (*set-bit-free* (*mem-pool-info V*) (*pool b*) (*lvl V t*) (*block b div 4* ^ (*level b* $-$ *lvl V t*)) (*pool b*)) (*lvl V t*)
(*block-ptr* (*mem-pool-info V* (*pool b*)) (*ALIGN4* (*max-sz* (*mem-pool-info V* (*pool b*))) *div 4* ^ *lvl V t*) (*block b div 4* ^ (*level b* $-$ *lvl V t*)))),
free-block-r := (*free-block-r V*)(*t* := *False*)|) $\Longrightarrow$
$\exists$ *lv bl*. *bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! *lv*) = *bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lv*) [*bl* := *FREE*]
$\wedge$ ($\forall$ *lv'*. *lv* $\neq$ *lv'* $\longrightarrow$ *bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! *lv'*) = *bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lv'*))
apply(*simp add*:*append-free-list-def set-bit-def block-ptr-def*)
apply(*subgoal-tac bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! *lvl V t*) = *bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*) [*block b div 4* ^ (*level b* $-$ *lvl V t*) := *FREE*])
prefer *2* **apply** *auto*[*1*]
apply(*subgoal-tac* $\forall$ *lv'*. *lvl V t* $\neq$ *lv'* $\longrightarrow$ *bits* (*levels* (*mem-pool-info V2* (*pool b*)) ! *lv'*) = *bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lv'*))
prefer *2* **apply** *clarify* **apply** *auto*[*1*]
apply(*rule exI*[**where** *x*=*lvl V t*],*auto*)
done

lemma *mempool-free-stm8-atombody-else-inv-bitmap*:
inv-bitmap V $\wedge$ *inv-aux-vars V* $\Longrightarrow$
pool b $\in$ *mem-pools V* $\Longrightarrow$
level b $<$ *length* (*levels* (*mem-pool-info V* (*pool b*))) $\Longrightarrow$
block b $<$ *length* (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *level b*)) $\Longrightarrow$
block b div 4 ^ (*level b* $-$ *lvl V t*) $<$ *length* (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)) $\Longrightarrow$
lvl V t $\leq$ *level b* $\Longrightarrow$
freeing-node V t = *Some blka* $\Longrightarrow$
pool blka = *pool b* $\Longrightarrow$
level blka = *lvl V t* $\Longrightarrow$
block blka = *block b div 4* ^ (*level b* $-$ *lvl V t*) $\Longrightarrow$
V2 = *V*(|*freeing-node* := (*freeing-node V*)(*t* := *None*),
mem-pool-info := (*set-bit-free* (*mem-pool-info V*) (*pool b*) (*lvl V t*) (*block b div 4* ^ (*level b* $-$ *lvl V t*)))
(*pool b* := *append-free-list* (*set-bit-free* (*mem-pool-info V*) (*pool b*) (*lvl V t*) (*block b div 4* ^ (*level b* $-$ *lvl V t*)) (*pool b*)) (*lvl V t*)
(*block-ptr* (*mem-pool-info V* (*pool b*)) (*ALIGN4* (*max-sz* (*mem-pool-info V* (*pool b*))) *div 4* ^ *lvl V t*) (*block b div 4* ^ (*level b* $-$ *lvl V t*)))),
free-block-r := (*free-block-r V*)(*t* := *False*)|) $\Longrightarrow$
inv-bitmap V2
apply(*unfold inv-bitmap-def*) **apply** *clarify*

apply(*case-tac p = pool b*)
apply(*subgoal-tac* $\exists$ *lv bl. bits (levels (mem-pool-info V (pool b)) ! lv) ! bl = FREEING*
$\wedge$ *bits (levels (mem-pool-info V2 (pool b)) ! lv) = bits (levels (mem-pool-info V (pool b)) ! lv) [bl := FREE]*
$\wedge$ ($\forall$ *lv'. lv* $\neq$ *lv'* $\longrightarrow$ *bits (levels (mem-pool-info V2 (pool b)) ! lv') = bits (levels (mem-pool-info V (pool b)) ! lv')*))
prefer *2* **apply**(*simp add:append-free-list-def set-bit-def block-ptr-def*)
apply(*subgoal-tac bits (levels (mem-pool-info V (pool b)) ! lvl V t) ! (block b div 4 ^ (level b − lvl V t)) = FREEING*)
prefer *2* **apply**(*simp add:inv-aux-vars-def*) **apply** *metis*
apply(*subgoal-tac bits (levels (mem-pool-info V2 (pool b)) ! lvl V t) = bits (levels (mem-pool-info V (pool b)) ! lvl V t) [block b div 4 ^ (level b − lvl V t) := FREE]*)
prefer *2* **apply** *auto*[*1*]
apply(*subgoal-tac* $\forall$ *lv'. lvl V t* $\neq$ *lv'* $\longrightarrow$ *bits (levels (mem-pool-info V2 (pool b)) ! lv') = bits (levels (mem-pool-info V (pool b)) ! lv')*)
prefer *2* **apply** *clarify* **apply** *auto*[*1*]
apply(*rule exI*[**where** *x=lvl V t*]) **apply** *auto*[*1*]

apply(*subgoal-tac length (levels (mem-pool-info V (pool b))) = length (levels (mem-pool-info V2 (pool b))))*
prefer *2* **apply**(*simp add:append-free-list-def set-bit-def block-ptr-def*)

apply(*subgoal-tac inv-bitmap-mp V (pool b)*) **prefer** *2* **apply**(*simp add:inv-bitmap-def*)

apply(*rule subst*[**where** *s=V2* **and** *t=V(|freeing-node := (freeing-node V)(t := None),*
mem-pool-info := (set-bit-free (mem-pool-info V) (pool b) (lvl V t) (block b div 4 ^ (level b − lvl V t)))
(pool b := append-free-list (set-bit-free (mem-pool-info V) (pool b) (lvl V t) (block b div 4 ^ (level b − lvl V t)) (pool b)) (lvl V t)
(block-ptr (mem-pool-info V (pool b)) (ALIGN4 (max-sz (mem-pool-info V (pool b))) div 4 ^ lvl V t) (block b div 4 ^ (level b − lvl V t)))),
free-block-r := (free-block-r V)(t := False)|)]) **apply** *fast*

using *inv-bitmap-freeing2free*[*of V pool b V2*] **apply** *fast*

apply(*subgoal-tac mem-pool-info V p = mem-pool-info*
(V(|freeing-node := (freeing-node V)(t := None),
mem-pool-info := (set-bit-free (mem-pool-info V) (pool b) (lvl V t) (block b div 4 ^ (level b − lvl V t)))
(pool b := append-free-list (set-bit-free (mem-pool-info V) (pool b) (lvl V t) (block b div 4 ^ (level b − lvl V t)) (pool b)) (lvl V t)
(block-ptr (mem-pool-info V (pool b)) (ALIGN4 (max-sz (mem-pool-info V (pool b))) div 4 ^ lvl V t) (block b div 4 ^ (level b − lvl V t)))),

free-block-r := (*free-block-r* V)(t := *False*)⦈) p)
prefer *2* **apply**(*simp add*:*append-free-list-def set-bit-def block-ptr-def*)
apply(*subgoal-tac mem-pools* V = *mem-pools* (V⦇*freeing-node* := (*freeing-node* V)(t := *None*),
mem-pool-info := (*set-bit-free* (*mem-pool-info* V) (*pool* b) (*lvl* V t) (*block* b *div* *4* ^ (*level* b − *lvl* V t)))
(*pool* b := *append-free-list* (*set-bit-free* (*mem-pool-info* V) (*pool* b) (*lvl* V t) (*block* b *div* *4* ^ (*level* b − *lvl* V t)) (*pool* b)) (*lvl* V t)
(*block-ptr* (*mem-pool-info* V (*pool* b)) (*ALIGN4* (*max-sz* (*mem-pool-info* V (*pool* b))) *div* *4* ^ *lvl* V t) (*block* b *div* *4* ^ (*level* b − *lvl* V t)))),
free-block-r := (*free-block-r* V)(t := *False*)⦈))
prefer *2* **apply**(*simp add*:*append-free-list-def set-bit-def block-ptr-def*)
by (*smt BlockState.distinct*(*13*))

lemma *mempool-free-stm8-atombody-else-inv-aux-vars*:
inv-mempool-info V ∧ *inv-aux-vars* V ⟹
allocating-node V t = *None* ⟹
pool b ∈ *mem-pools* V ⟹
level b < *length* (*levels* (*mem-pool-info* V (*pool* b))) ⟹
block b < *length* (*bits* (*levels* (*mem-pool-info* V (*pool* b)) ! *level* b)) ⟹
block b *div* *4* ^ (*level* b − *lvl* V t) < *length* (*bits* (*levels* (*mem-pool-info* V (*pool* b)) ! *lvl* V t)) ⟹
bn V t = *block* b *div* *4* ^ (*level* b − *lvl* V t) ⟹
lvl V t ≤ *level* b ⟹
freeing-node V t = *Some blka* ⟹
pool blka = *pool* b ⟹
level blka = *lvl* V t ⟹
block blka = *block* b *div* *4* ^ (*level* b − *lvl* V t) ⟹
inv-aux-vars
(V⦇*freeing-node* := (*freeing-node* V)(t := *None*),
mem-pool-info := (*set-bit-free* (*mem-pool-info* V) (*pool* b) (*lvl* V t) (*block* b *div* *4* ^ (*level* b − *lvl* V t)))
(*pool* b := *append-free-list* (*set-bit-free* (*mem-pool-info* V) (*pool* b) (*lvl* V t) (*block* b *div* *4* ^ (*level* b − *lvl* V t)) (*pool* b)) (*lvl* V t)
(*block-ptr* (*mem-pool-info* V (*pool* b)) (*ALIGN4* (*max-sz* (*mem-pool-info* V (*pool* b))) *div* *4* ^ *lvl* V t) (*block* b *div* *4* ^ (*level* b − *lvl* V t)))),
free-block-r := (*free-block-r* V)(t := *False*)⦈)
apply(*simp add*:*inv-aux-vars-def append-free-list-def set-bit-def block-ptr-def*) **apply** *clarify*
apply(*rule subst*[**where** *s*=*max-sz* (*mem-pool-info* V (*pool* b)) **and** *t*=*ALIGN4* (*max-sz* (*mem-pool-info* V (*pool* b)))])
apply (*metis inv-mempool-info-maxsz-align4*)

apply(*rule conjI*) **apply** *clarify*
apply(*subgoal-tac* ¬(*pool* n = *pool blka* ∧ *level* n = *level blka* ∧ *block* n = *block blka*))

prefer *2* **apply** *blast*
apply(*case-tac pool n* $\neq$ *pool blka*) **apply** *auto*[*1*]
apply(*case-tac level n* $\neq$ *level blka*) **apply** (*metis* (*no-types*, *lifting*) *nth-list-update-neq*)

apply(*case-tac block n* $\neq$ *block blka*)
apply(*subgoal-tac bits* (*levels* (*mem-pool-info V* (*pool b*))
[*lvl V t* := (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)
(|*bits* := *bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)[*block b div 4* ^ (*level b* − *lvl V t*) := *FREE*],
free-list :=
free-list (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*) @
[*buf* (*mem-pool-info V* (*pool b*)) +
max-sz (*mem-pool-info V* (*pool b*)) *div 4* ^ *lvl V t* ∗ (*block b div 4* ^ (*level b* − *lvl V t*))]|)] !
level n) !
block n = *bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *level n*) ! *block n*)
prefer *2* **apply**(*subgoal-tac bits* (*levels* (*mem-pool-info V* (*pool b*))
[*lvl V t* := (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)
(|*bits* := *bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)[*block b div 4* ^ (*level b* − *lvl V t*) := *FREE*],
free-list :=
free-list (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*) @
[*buf* (*mem-pool-info V* (*pool b*)) +
max-sz (*mem-pool-info V* (*pool b*)) *div 4* ^ *lvl V t* ∗ (*block b div 4* ^ (*level b* − *lvl V t*))]|)] !
level n) = *bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)[*block b div 4* ^ (*level b* − *lvl V t*) := *FREE*])
prefer *2* **apply** *auto*[*1*] **apply** *auto*[*1*]
apply *metis*
apply *fast*

apply(*rule conjI*) **apply** *clarify*
apply(*rule conjI*) **apply** *clarify*
apply(*subgoal-tac bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *level n*) ! *block n* = *FREEING*
∧ (*lvl V t* $\neq$ *level n* ∨ *block b div 4* ^ (*level b* − *lvl V t*) $\neq$ *block n*))
prefer *2* **apply**(*case-tac lvl V t* = *level n*) **apply**(*case-tac block b div 4* ^ (*level b* − *lvl V t*) = *block n*)
apply *clarsimp* **apply** *clarsimp* **apply** *clarsimp*
apply(*subgoal-tac mem-block-addr-valid V n*)
prefer *2* **apply**(*simp add*:*mem-block-addr-valid-def*)
apply(*subgoal-tac blka* $\neq$ *n*)
prefer *2* **apply** *metis*
apply (*metis option.inject*)
apply *clarify*
apply(*subgoal-tac mem-block-addr-valid V n*)
prefer *2* **apply**(*simp add*:*mem-block-addr-valid-def*)

apply (*metis option.inject*)

apply(*rule conjI*) **apply** *clarify*
apply(*subgoal-tac get-bit-s V* (*pool n*) (*level n*) (*block n*) = *ALLOCATING*)
prefer *2* **apply** *blast*
apply(*case-tac lvl V t* = *level n*) **apply**(*case-tac block b div 4* ˆ (*level b* − *lvl V t*) = *block n*)
apply *metis* **apply** *clarsimp* **apply** *clarsimp*

apply *clarify*
apply(*rule conjI*) **apply** *clarify*
apply(*subgoal-tac bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *level n*) ! *block n* = *ALLOCATING*)
prefer *2* **apply**(*case-tac lvl V t* = *level n*) **apply**(*case-tac block b div 4* ˆ (*level b* − *lvl V t*) = *block n*)
apply *clarsimp* **apply** *clarsimp* **apply** *clarsimp*
apply(*subgoal-tac mem-block-addr-valid V n*)
prefer *2* **apply**(*simp add*:*mem-block-addr-valid-def*)
apply *metis*

apply *clarify*
apply(*subgoal-tac mem-block-addr-valid V n*)
prefer *2* **apply**(*simp add*:*mem-block-addr-valid-def*)
apply *metis*
done

lemma *mempool-free-stm8-atombody-else-inv-bitmap0*:
inv-mempool-info V ∧ *inv-bitmap0 V* ⟹
allocating-node V t = *None* ⟹
pool b ∈ *mem-pools V* ⟹
level b < *length* (*levels* (*mem-pool-info V* (*pool b*))) ⟹
block b < *length* (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *level b*)) ⟹
block b div 4 ˆ (*level b* − *lvl V t*) < *length* (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)) ⟹
bn V t = *block b div 4* ˆ (*level b* − *lvl V t*) ⟹
lvl V t ≤ *level b* ⟹
freeing-node V t = *Some blka* ⟹
pool blka = *pool b* ⟹
level blka = *lvl V t* ⟹
block blka = *block b div 4* ˆ (*level b* − *lvl V t*) ⟹
inv-bitmap0
(*V*⦇*freeing-node* := (*freeing-node V*)(*t* := *None*),
mem-pool-info := (*set-bit-free* (*mem-pool-info V*) (*pool b*) (*lvl V t*) (*block b div 4* ˆ (*level b* − *lvl V t*)))
(*pool b* := *append-free-list* (*set-bit-free* (*mem-pool-info V*) (*pool b*) (*lvl V t*) (*block b div 4* ˆ (*level b* − *lvl V t*)) (*pool b*)) (*lvl V t*)
(*block-ptr* (*mem-pool-info V* (*pool b*)) (*ALIGN4* (*max-sz* (*mem-pool-info V* (*pool b*))) *div 4* ˆ *lvl V t*) (*block b div 4* ˆ (*level b* − *lvl V t*)))),

free-block-r := (*free-block-r* *V*)(*t* := *False*)⦈)
apply(*simp add*:*inv-bitmap0-def inv-mempool-info-def append-free-list-def set-bit-def block-ptr-def ALIGN4-def Let-def*) **apply** *clarsimp*
apply(*subgoal-tac get-bit-s V* (*pool b*) *0 i* ≠ *NOEXIST*) **prefer** *2*
apply(*subgoal-tac length* (*bits* (*levels* (*mem-pool-info V* (*pool b*))
[*lvl V t* := (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)
⦇*bits* := *bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)[*block b div 4* ˆ (*level b* − *lvl V t*) := *FREE*],
free-list :=
free-list (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*) @
[*buf* (*mem-pool-info V* (*pool b*)) +
(*max-sz* (*mem-pool-info V* (*pool b*)) + *3*) *div 4* ∗ *4 div 4* ˆ *lvl V t* ∗ (*block b div 4* ˆ (*level b* − *lvl V t*))]⦈] !
0)) = *length* (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *0*)))
prefer *2*
apply(*case-tac lvl V t* = *0*) **apply** *clarsimp*
apply(*rule subst*[**where** *s*=*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *0*)[*block b div 4* ˆ (*level b* − *lvl V t*) := *FREE*]
and *t*=*bits* (*levels* (*mem-pool-info V* (*pool b*))
[*0* := (*levels* (*mem-pool-info V* (*pool b*)) ! *0*)
⦇*bits* := *bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *NULL*)[*block b div 4* ˆ *level b* := *FREE*],
free-list :=
free-list (*levels* (*mem-pool-info V* (*pool b*)) ! *NULL*) @
[*buf* (*mem-pool-info V* (*pool b*)) + (*max-sz* (*mem-pool-info V* (*pool b*)) + *3*) *div 4* ∗ *4* ∗ (*block b div 4* ˆ *level b*)]⦈] !
0)]) **apply** *auto*[*1*] **apply** *auto*[*1*]
apply *auto*[*1*]
apply *auto*[*1*]

apply(*case-tac lvl V t* = *0*)
apply(*case-tac i* = *block b div 4* ˆ (*level b* − *lvl V t*))
apply *auto*[*1*] **apply** *auto*[*1*] **apply** *auto*[*1*]
done

lemma *mempool-free-stm8-atombody-else-inv-bitmapn*:
inv-mempool-info V ∧ *inv-bitmapn V* ⟹
allocating-node V t = *None* ⟹
pool b ∈ *mem-pools V* ⟹
level b < *length* (*levels* (*mem-pool-info V* (*pool b*))) ⟹
block b < *length* (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *level b*)) ⟹
block b div 4 ˆ (*level b* − *lvl V t*) < *length* (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)) ⟹
bn V t = *block b div 4* ˆ (*level b* − *lvl V t*) ⟹
lvl V t ≤ *level b* ⟹
freeing-node V t = *Some blka* ⟹
pool blka = *pool b* ⟹
level blka = *lvl V t* ⟹
block blka = *block b div 4* ˆ (*level b* − *lvl V t*) ⟹

inv-bitmapn
(*V*(|*freeing-node* := (*freeing-node V*)(*t* := *None*),
mem-pool-info := (*set-bit-free* (*mem-pool-info V*) (*pool b*) (*lvl V t*) (*block b div 4* ^ (*level b* − *lvl V t*)))
(*pool b* := *append-free-list* (*set-bit-free* (*mem-pool-info V*) (*pool b*) (*lvl V t*) (*block b div 4* ^ (*level b* − *lvl V t*)) (*pool b*)) (*lvl V t*)
(*block-ptr* (*mem-pool-info V* (*pool b*)) (*ALIGN4* (*max-sz* (*mem-pool-info V* (*pool b*))) *div 4* ^ *lvl V t*) (*block b div 4* ^ (*level b* − *lvl V t*)))),
free-block-r := (*free-block-r V*)(*t* := *False*)|))
apply(*simp add:inv-bitmapn-def inv-mempool-info-def append-free-list-def set-bit-def block-ptr-def ALIGN4-def Let-def*) **apply** *clarsimp*

apply(*subgoal-tac get-bit-s V* (*pool b*) (*length* (*levels* (*mem-pool-info V* (*pool b*))) − *Suc 0*) *i* ≠ *DIVIDED*) **prefer** *2*
apply(*subgoal-tac length* (*bits* (*levels* (*mem-pool-info V* (*pool b*))
[*lvl V t* := (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)
(|*bits* := *bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)[*block b div 4* ^ (*level b* − *lvl V t*) := *FREE*],
free-list :=
free-list (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*) @
[*buf* (*mem-pool-info V* (*pool b*)) +
(*max-sz* (*mem-pool-info V* (*pool b*)) + *3*) *div 4* ∗ *4 div 4* ^ *lvl V t* ∗ (*block b div 4* ^ (*level b* − *lvl V t*))]|)] !
(*length* (*levels* (*mem-pool-info V* (*pool b*))) − *Suc 0*)))
= *length* (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! (*length* (*levels* (*mem-pool-info V* (*pool b*))) − *Suc 0*)))) **prefer** *2*
apply(*case-tac lvl V t* = (*length* (*levels* (*mem-pool-info V* (*pool b*))) − *Suc 0*))
apply *clarsimp*
apply(*rule subst*[**where** *s*=*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! (*length* (*levels* (*mem-pool-info V* (*pool b*))) − *Suc 0*))
[*block b div 4* ^ (*level b* − *lvl V t*) := *FREE*]
and *t*=*bits* (*levels* (*mem-pool-info V* (*pool b*))
[(*length* (*levels* (*mem-pool-info V* (*pool b*))) − *Suc 0*) :=
(*levels* (*mem-pool-info V* (*pool b*)) ! (*length* (*levels* (*mem-pool-info V* (*pool b*))) − *Suc 0*))
(|*bits* := *bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *NULL*)[*block b div 4* ^ *level b* := *FREE*],
free-list :=
free-list (*levels* (*mem-pool-info V* (*pool b*)) ! *NULL*) @
[*buf* (*mem-pool-info V* (*pool b*)) + (*max-sz* (*mem-pool-info V* (*pool b*)) + *3*) *div 4* ∗ *4* ∗ (*block b div 4* ^ *level b*)]|)] !
(*length* (*levels* (*mem-pool-info V* (*pool b*))) − *Suc 0*))]) **apply** *auto*[*1*] **apply** *auto*[*1*]
apply *auto*[*1*]

apply(*case-tac lvl V t* = (*length* (*levels* (*mem-pool-info V* (*pool b*))) − *Suc 0*))
apply(*case-tac i* = *block b div 4* ^ (*level b* − *lvl V t*))
apply *auto*[*1*] **apply** *auto*[*1*] **apply** *auto*[*1*]
done

lemma *mempool-free-stm8-atombody-else-inv-bitmap-not4free*:
lvl V t = NULL ∨
¬ *partner-bits* (*set-bit-free* (*mem-pool-info V*) (*pool b*) (*lvl V t*) (*block b div 4* ^ (*level b* − *lvl V t*)) (*pool b*)) (*lvl V t*)
(*block b div 4* ^ (*level b* − *lvl V t*)) ⟹
inv-mempool-info V ∧ *inv-bitmap-not4free V* ⟹
allocating-node V t = None ⟹
pool b ∈ *mem-pools V* ⟹
level b < *length* (*levels* (*mem-pool-info V* (*pool b*))) ⟹
block b < *length* (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *level b*)) ⟹
block b div 4 ^ (*level b* − *lvl V t*) < *length* (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)) ⟹
bn V t = block b div 4 ^ (*level b* − *lvl V t*) ⟹
lvl V t ≤ *level b* ⟹
freeing-node V t = Some blka ⟹
pool blka = pool b ⟹
level blka = lvl V t ⟹
block blka = block b div 4 ^ (*level b* − *lvl V t*) ⟹
inv-bitmap-not4free
(*V*⦇*freeing-node* := (*freeing-node V*)(*t* := *None*),
mem-pool-info := (*set-bit-free* (*mem-pool-info V*) (*pool b*) (*lvl V t*) (*block b div 4* ^ (*level b* − *lvl V t*)))
(*pool b* := *append-free-list* (*set-bit-free* (*mem-pool-info V*) (*pool b*) (*lvl V t*) (*block b div 4* ^ (*level b* − *lvl V t*)) (*pool b*)) (*lvl V t*)
(*block-ptr* (*mem-pool-info V* (*pool b*)) (*ALIGN4* (*max-sz* (*mem-pool-info V* (*pool b*))) *div 4* ^ *lvl V t*) (*block b div 4* ^ (*level b* − *lvl V t*)))),
free-block-r := (*free-block-r V*)(*t* := *False*)⦈)
apply(*simp add*:*inv-bitmap-not4free-def inv-mempool-info-def append-free-list-def*
set-bit-def block-ptr-def ALIGN4-def Let-def)
apply *clarsimp*

apply(*case-tac lvl V t = 0*)
apply *clarsimp*
apply(*simp add*:*partner-bits-def Let-def*) **apply** *auto*[*1*]

apply *clarsimp*
apply(*case-tac i = lvl V t*)
apply(*simp add*:*partner-bits-def Let-def*)
apply *clarsimp*
apply(*case-tac j div 4 = block b div 4* ^ (*level b* − *lvl V t*) *div 4*)
apply *auto*[*1*]
apply *auto*[*1*]

apply(*simp add*:*partner-bits-def Let-def*)
apply *clarsimp*
done

lemma *mempool-free-stm8-atombody-else-inv*:
lvl V t = *NULL* ∨
¬ *partner-bits* (*set-bit-free* (*mem-pool-info V*) (*pool b*) (*lvl V t*) (*block b div 4* ^ (*level b* − *lvl V t*)) (*pool b*)) (*lvl V t*)
(*block b div 4* ^ (*level b* − *lvl V t*)) ⟹
inv V ⟹
allocating-node V t = *None* ⟹
pool b ∈ *mem-pools V* ⟹
level b < *length* (*levels* (*mem-pool-info V* (*pool b*))) ⟹
block b < *length* (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *level b*)) ⟹
data b = *block-ptr* (*mem-pool-info V* (*pool b*)) (*ALIGN4* (*max-sz* (*mem-pool-info V* (*pool b*))) *div 4* ^ *level b*) (*block b*) ⟹
level b < *length* (*lsizes V t*) ⟹
∀ *ii*<*length* (*lsizes V t*). *lsizes V t* ! *ii* = *ALIGN4* (*max-sz* (*mem-pool-info V* (*pool b*))) *div 4* ^ *ii* ⟹
block b div 4 ^ (*level b* − *lvl V t*) < *length* (*bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)) ⟹
bn V t = *block b div 4* ^ (*level b* − *lvl V t*) ⟹
lvl V t ≤ *level b* ⟹
free-block-r V t ⟹
lsz V t = *ALIGN4* (*max-sz* (*mem-pool-info V* (*pool b*))) *div 4* ^ *lvl V t* ⟹
blk V t = *block-ptr* (*mem-pool-info V* (*pool b*)) (*ALIGN4* (*max-sz* (*mem-pool-info V* (*pool b*))) *div 4* ^ *lvl V t*) (*block b div 4* ^ (*level b* − *lvl V t*)) ⟹
cur V = *Some t* ⟹
data blka = *buf* (*mem-pool-info V* (*pool b*)) + *block b div 4* ^ (*level b* − *lvl V t*) ∗ (*max-sz* (*mem-pool-info V* (*pool b*)) *div 4* ^ *lvl V t*) ⟹
block b div 4 ^ (*level b* − *lvl V t*) < *n-max* (*mem-pool-info V* (*pool b*)) ∗ *4* ^ *lvl V t* ⟹
freeing-node V t = *Some y* ⟹
pool y = *pool b* ⟹
level y = *lvl V t* ⟹
block y = *block b div 4* ^ (*level b* − *lvl V t*) ⟹
inv
(*V*⦇*freeing-node* := (*freeing-node V*)(*t* := *None*),
mem-pool-info := (*set-bit-free* (*mem-pool-info V*) (*pool b*) (*lvl V t*) (*block b div 4* ^ (*level b* − *lvl V t*)))
(*pool b* := *append-free-list* (*set-bit-free* (*mem-pool-info V*) (*pool b*) (*lvl V t*) (*block b div 4* ^ (*level b* − *lvl V t*)) (*pool b*)) (*lvl V t*)
(*block-ptr* (*mem-pool-info V* (*pool b*)) (*ALIGN4* (*max-sz* (*mem-pool-info V* (*pool b*))) *div 4* ^ *lvl V t*) (*block b div 4* ^ (*level b* − *lvl V t*)))),
free-block-r := (*free-block-r V*)(*t* := *False*)⦈)
apply(*simp add*:*inv-def*)
apply(*rule conjI*) **apply**(*simp add*:*inv-cur-def Mem-pool-free-guar-def*)
apply(*rule conjI*) **apply**(*simp add*:*inv-thd-waitq-def append-free-list-def set-bit-def*)
apply *smt*
apply(*rule conjI*) **using** *mempool-free-stm8-atombody-else-inv-mempool-info* **apply** *blast*
apply(*rule conjI*) **using** *mempool-free-stm8-atombody-else-inv-bitmap-freelist* **ap-**

ply *blast*
apply(*rule conjI*) **using** *mempool-free-stm8-atombody-else-inv-bitmap* **apply** *blast*
apply(*rule conjI*) **using** *mempool-free-stm8-atombody-else-inv-aux-vars* **apply** *blast*
apply(*rule conjI*) **using** *mempool-free-stm8-atombody-else-inv-bitmap0* **apply** *blast*
apply(*rule conjI*) **using** *mempool-free-stm8-atombody-else-inv-bitmapn* **apply** *blast*
using *mempool-free-stm8-atombody-else-inv-bitmap-not4free* **apply** *blast*
done

lemma *mp-free-stm8-intI*:
$\{V\} \subseteq \{\!|\, ´(\textit{free-block-r-update}\ (\lambda\text{-}.\ ´\textit{free-block-r}(t := \textit{False}))) \in A |\!\} \Longrightarrow$
$\{V\} \subseteq \{\!|\, ´(\textit{free-block-r-update}\ (\lambda\text{-}.\ ´\textit{free-block-r}(t := \textit{False}))) \in B |\!\} \Longrightarrow$
$\{V\} \subseteq \{\!|\, ´(\textit{free-block-r-update}\ (\lambda\text{-}.\ ´\textit{free-block-r}(t := \textit{False}))) \in A \cap B |\!\}$
by *auto*

lemma *mempool-free-stm8-atombody-else-h1*:
$V \in \{\!|\, ´\textit{free-block-r}\ t |\!\} \cap \textit{mp-free-precond8-3}\ t\ b\ \alpha \cap \{\!|\, ´\textit{cur} = \textit{Some}\ t |\!\} \Longrightarrow$
$\{\textit{free-stm8-precond2}\ V\ t\ b\} \cap - \{\!|\textit{NULL} < ´\textit{lvl}\ t \wedge \textit{partner-bits}\ (´\textit{mem-pool-info}\ (\textit{pool}\ b))\ (´\textit{lvl}\ t)\ (´\textit{bn}\ t)|\!\} \neq \{\} \Longrightarrow$
$\{\textit{let}\ V2 = \textit{free-stm8-precond2}\ V\ t\ b\ \textit{in}\ V2(\!|\textit{mem-pool-info} := (\textit{mem-pool-info}\ V2)$
$(\textit{pool}\ b := \textit{append-free-list}\ (\textit{mem-pool-info}\ V2\ (\textit{pool}\ b))\ (\textit{lvl}\ V2\ t)\ (\textit{blk}\ V2\ t))|\!)\}$
$\subseteq \{\!|\, ´(\textit{free-block-r-update}\ (\lambda\text{-}.\ ´\textit{free-block-r}(t := \textit{False})))$
$\in \{\!|\, ´(\textit{Pair}\ V) \in \textit{Mem-pool-free-guar}\ t |\!\} \cap \textit{mp-free-precond8-inv}\ t\ b\ 0 |\!\}$
apply(*rule mp-free-stm8-intI*)

apply(*simp add*:*Mem-pool-free-guar-def*)
apply(*rule disjI1*) **apply**(*rule conjI*)
apply(*simp add*:*gvars-conf-stable-def gvars-conf-def append-free-list-def set-bit-def block-ptr-def*) **apply** *clarify*
apply(*rename-tac ii blk*)
apply(*case-tac lvl V t = ii*)
apply(*subgoal-tac bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)[*block b div 4* ˆ (*level b* − *lvl V t*) := *FREE*] =
bits (*levels* (*mem-pool-info V* (*pool b*))
[*lvl V t* := (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)
(|*bits* := *bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)[*block b div 4* ˆ (*level b* − *lvl V t*) := *FREE*],
free-list := *free-list* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*) @
[*buf* (*mem-pool-info V* (*pool b*)) + *ALIGN4* (*max-sz* (*mem-pool-info V* (*pool b*)))
div 4 ˆ *lvl V t* ∗ (*block b div 4* ˆ (*level b* − *lvl V t*))]|)] !
ii))
prefer *2* **apply** *auto*[*1*]
apply (*metis length-list-update*)

apply(*subgoal-tac bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *ii*) =
bits (*levels* (*mem-pool-info V* (*pool b*))
[*lvl V t* := (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)
(|*bits* := *bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)[*block b div 4* ^ (*level b* − *lvl V t*) := *FREE*],
free-list := *free-list* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*) @
[*buf* (*mem-pool-info V* (*pool b*)) + *ALIGN4* (*max-sz* (*mem-pool-info V* (*pool b*)))
div 4 ^ *lvl V t* ∗ (*block b div 4* ^ (*level b* − *lvl V t*))]|)] !
ii))
prefer *2* **apply** *auto*[*1*]
apply *auto*[*1*]
apply(*rule conjI*) **apply** *clarsimp*
using *mempool-free-stm8-atombody-else-inv*[*of V t b*] **apply** *metis*

apply *clarify* **apply**(*simp add*:*lvars-nochange-def*)

apply(*rule mp-free-stm8-intI*)
apply *clarsimp*
apply(*rule conjI*)
using *mempool-free-stm8-atombody-else-inv*[*of V t b*] **apply** *metis*
apply(*rule conjI*)
apply(*simp add*:*append-free-list-def set-bit-def block-ptr-def*)
apply(*rule conjI*)
apply(*simp add*:*append-free-list-def set-bit-def block-ptr-def*)
apply(*case-tac lvl V t* = *level b*)
apply(*subgoal-tac bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)[*block b div 4* ^ (*level b* − *lvl V t*) := *FREE*] =
bits (*levels* (*mem-pool-info V* (*pool b*))
[*lvl V t* := (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)
(|*bits* := *bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)[*block b div 4* ^ (*level b* − *lvl V t*) := *FREE*],
free-list :=
free-list (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*) @
[*buf* (*mem-pool-info V* (*pool b*)) +
ALIGN4 (*max-sz* (*mem-pool-info V* (*pool b*))) *div 4* ^ *lvl V t* ∗ (*block b div 4* ^ (*level b* − *lvl V t*))]|)] !
level b))
prefer *2* **apply** *auto*[*1*]
apply (*metis length-list-update*)
apply(*subgoal-tac bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *level b*) =
bits (*levels* (*mem-pool-info V* (*pool b*))
[*lvl V t* := (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)
(|*bits* := *bits* (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*)[*block b div 4* ^ (*level b* − *lvl V t*) := *FREE*],
free-list :=
free-list (*levels* (*mem-pool-info V* (*pool b*)) ! *lvl V t*) @
[*buf* (*mem-pool-info V* (*pool b*)) +

ALIGN4 (*max-sz* (*mem-pool-info* *V* (*pool* *b*))) *div*
4 ˆ *lvl* *V* *t* ∗ (*block* *b* *div* *4* ˆ (*level* *b* − *lvl* *V* *t*))]⦈] !
level *b*))
prefer *2* **apply** *auto*[*1*]
apply *metis*
apply(*rule conjI*)
apply(*simp add*:*append-free-list-def set-bit-def block-ptr-def*)
apply(*rule conjI*)
apply(*simp add*:*append-free-list-def set-bit-def block-ptr-def*)
apply(*simp add*:*append-free-list-def set-bit-def block-ptr-def*)

apply *clarsimp*
done

lemma *mempool-free-stm8-atombody-else'*:
V∈ ⦃´*free-block-r t*⦄ ∩ *mp-free-precond8-3 t b α* ∩ ⦃´*cur* = *Some t*⦄ ⟹
Γ $\vdash_I$ *Some* (*IF block-fits* (´*mem-pool-info* (*pool b*)) (´*blk t*) (´*lsz t*) *THEN*
´*mem-pool-info* := ´*mem-pool-info*(*pool b* := *append-free-list* (´*mem-pool-info*
(*pool b*)) (´*lvl t*) (´*blk t*))
FI;;
´*free-block-r* := ´*free-block-r* (*t* := *False*))
sat_p [{*free-stm8-precond2 V t b*} ∩ − ⦃*NULL* < ´*lvl t* ∧ *partner-bits* (´*mem-pool-info*
(*pool b*)) (´*lvl t*) (´*bn t*)⦄,
{(*s*, *t*). *s* = *t*}, *UNIV*, ⦃´(*Pair V*) ∈ *Mem-pool-free-guar t*⦄
∩ *mp-free-precond8-inv t b 0*]
apply(*case-tac* {*free-stm8-precond2 V t b*} ∩ − ⦃*NULL* < ´*lvl t* ∧ *partner-bits*
(´*mem-pool-info* (*pool b*)) (´*lvl t*) (´*bn t*)⦄ = {})
using *Emptyprecond*[*of Some* (*IF block-fits* (´*mem-pool-info* (*pool b*)) (´*blk t*)
(´*lsz t*) *THEN*
´*mem-pool-info* := ´*mem-pool-info*(*pool b* := *append-free-list* (´*mem-pool-info*
(*pool b*)) (´*lvl t*) (´*blk t*))
FI;;
´*free-block-r* := ´*free-block-r* (*t* := *False*)) {(*s*, *t*). *s* = *t*}
UNIV ⦃´(*Pair V*) ∈ *Mem-pool-free-guar t*⦄ ∩ *mp-free-precond8-inv t b 0*] **apply**
metis

apply(*rule Seq*[**where** *mid*={*let V2* = *free-stm8-precond2 V t b in V2*⦇*mem-pool-info*
:= (*mem-pool-info V2*)
(*pool b* := *append-free-list* (*mem-pool-info V2* (*pool b*)) (*lvl V2 t*) (*blk V2*
t))⦈}])

apply(*rule Cond*)
using *stable-id2* **apply** *fast*

apply(*rule subst*[**where** *s*={} **and** *t*={*free-stm8-precond2 V t b*} ∩ − ⦃*NULL*
< ´*lvl t* ∧ *partner-bits* (´*mem-pool-info* (*pool b*)) (´*lvl t*) (´*bn t*)⦄ ∩
− ⦃*block-fits* (´*mem-pool-info* (*pool b*)) (´*blk t*)
(´*lsz t*)⦄])
using *mempool-free-stm8-atombody-else-blockfit*[*of V t b α*] **apply** *fast*

apply(*rule Basic*) **apply**(*simp add*:*Let-def*)
apply *auto*[*1*] **apply** *fast* **using** *stable-id2* **apply** *fast* **using** *stable-id2* **apply** *fast*

apply(*rule subst*[**where** *s*={} **and** *t*={*free-stm8-precond2 V t b*} ∩ − ⦃*NULL* < ´*lvl t* ∧ *partner-bits* (´*mem-pool-info* (*pool b*)) (´*lvl t*) (´*bn t*)⦄ ∩
− ⦃*block-fits* (´*mem-pool-info* (*pool b*)) (´*blk t*)
(´*lsz t*)⦄])
using *mempool-free-stm8-atombody-else-blockfit*[*of V t b* α] **apply** *fast*
using *Emptyprecond* **apply** *blast*

apply *fast*

apply(*rule Basic*)
using *mempool-free-stm8-atombody-else-h1*[*of V t b* α] **apply** *fast*

apply *fast*
using *stable-id2* **apply** *fast* **using** *stable-id2* **apply** *fast*
done

lemma *mempool-free-stm8-atombody-else*:
V∈ *mp-free-precond8-3 t b* α ∩ ⦃´*cur* = *Some t*⦄ ⟹
Γ $\vdash_I$ *Some* (*IF block-fits* (´*mem-pool-info* (*pool b*)) (´*blk t*) (´*lsz t*) *THEN*
´*mem-pool-info* := ´*mem-pool-info*(*pool b* := *append-free-list* (´*mem-pool-info* (*pool b*)) (´*lvl t*) (´*blk t*))
FI;;
´*free-block-r* := ´*free-block-r* (*t* := *False*))
sat_p [{*free-stm8-precond2 V t b*} ∩ − ⦃*NULL* < ´*lvl t* ∧ *partner-bits* (´*mem-pool-info* (*pool b*)) (´*lvl t*) (´*bn t*)⦄,
{(*s*, *t*). *s* = *t*}, *UNIV*, ⦃´(*Pair V*) ∈ *Mem-pool-free-guar t*⦄ ∩ *mp-free-precond8-inv t b 0*]
apply(*subgoal-tac V*∈⦃´*free-block-r t*⦄ ∩ *mp-free-precond8-3 t b* α ∩ ⦃´*cur* = *Some t*⦄)
prefer *2* **apply**(*subgoal-tac mp-free-precond8-1 t b* α = ⦃´*free-block-r t*⦄ ∩ *mp-free-precond8-1 t b* α)
prefer *2* **using** *mp-free-precond8-1-imp-free-block-r*[*of t b* α] *Int-absorb1*[*of mp-free-precond8-1 t b* α ⦃´*free-block-r t*⦄] **apply** *metis*
apply *auto*[*1*]
using *mempool-free-stm8-atombody-else'*[*of V t b* α] **apply** *metis*
done

lemma *mempool-free-stm8-atombody-rest-extpost*:
V∈*mp-free-precond8-3 t b* α ∩ ⦃´*cur* = *Some t*⦄ ⟹
{*free-stm8-precond2 V t b*} ∩ ⦃*NULL* < ´*lvl t* ∧ *partner-bits* (´*mem-pool-info* (*pool b*)) (´*lvl t*) (´*bn t*)⦄ ≠ {} ⟹
Γ $\vdash_I$ *Some* (´*lvl* := ´*lvl*(*t* := ´*lvl t* − *1*);;

´bn := ´bn(t := ´bn t div 4);;
´mem-pool-info := set-bit-freeing ´mem-pool-info (pool b) (´lvl t) (´bn t);;
´freeing-node := ´freeing-node(t ↦(|pool = pool b, level = ´lvl t, block = ´bn t,
data = block-ptr (´mem-pool-info (pool b)) (ALIGN4 (max-sz (´mem-pool-info (pool b))) div 4 ^ ´lvl t) (´bn t)|)))
sat_p *[free-stm8-precond3 V t b ∩ {|´i t = 4|}, {(s, t). s = t}, UNIV,*
{|´(Pair V) ∈ Mem-pool-free-guar t|} ∩ (mp-free-precond8-inv t b (α − 1) ∪ mp-free-precond8-inv t b 0)]
apply(*rule Conseq[of free-stm8-precond3 V t b ∩ {|´i t = 4|} free-stm8-precond3 V t b ∩ {|´i t = 4|}*
{(s, t). s = t} {(s, t). s = t} UNIV UNIV
{|´(Pair V) ∈ Mem-pool-free-guar t|} ∩ mp-free-precond8-inv t b (α − 1)
{|´(Pair V) ∈ Mem-pool-free-guar t|} ∩ (mp-free-precond8-inv t b (α − 1) ∪ mp-free-precond8-inv t b 0)
Some (´lvl := ´lvl(t := ´lvl t − 1);;
´bn := ´bn(t := ´bn t div 4);;
´mem-pool-info := set-bit-freeing ´mem-pool-info (pool b) (´lvl t) (´bn t);;
´freeing-node := ´freeing-node(t ↦(|pool = pool b, level = ´lvl t, block = ´bn t,
data = block-ptr (´mem-pool-info (pool b)) (ALIGN4 (max-sz (´mem-pool-info (pool b))) div 4 ^ ´lvl t) (´bn t)|))])
apply *fast* **apply** *fast* **apply** *fast* **apply** *auto[1]*
using *mempool-free-stm8-atombody-rest* **apply** *blast*
done

lemma *mempool-free-stm8-atombody-else-extpost*:
V∈ mp-free-precond8-3 t b α ∩ {|´cur = Some t|} ⟹
Γ $\vdash_I$ *Some (IF block-fits (´mem-pool-info (pool b)) (´blk t) (´lsz t) THEN*
´mem-pool-info := ´mem-pool-info(pool b := append-free-list (´mem-pool-info (pool b)) (´lvl t) (´blk t))
FI;;
´free-block-r := ´free-block-r (t := False))
sat_p *[{free-stm8-precond2 V t b} ∩ − {|NULL < ´lvl t ∧ partner-bits (´mem-pool-info (pool b)) (´lvl t) (´bn t)|},*
{(s, t). s = t}, UNIV, {|´(Pair V) ∈ Mem-pool-free-guar t|}
∩ (mp-free-precond8-inv t b (α − 1) ∪ mp-free-precond8-inv t b 0)]
apply(*rule Conseq[of {free-stm8-precond2 V t b} ∩ − {|NULL < ´lvl t ∧ partner-bits (´mem-pool-info (pool b)) (´lvl t) (´bn t)|}*
{free-stm8-precond2 V t b} ∩ − {|NULL < ´lvl t ∧ partner-bits (´mem-pool-info (pool b)) (´lvl t) (´bn t)|}
{(s, t). s = t} {(s, t). s = t} UNIV UNIV
{|´(Pair V) ∈ Mem-pool-free-guar t|} ∩ mp-free-precond8-inv t b 0
{|´(Pair V) ∈ Mem-pool-free-guar t|} ∩ (mp-free-precond8-inv t b (α − 1) ∪ mp-free-precond8-inv t b 0)
Some (IF block-fits (´mem-pool-info (pool b)) (´blk t) (´lsz t) THEN
´mem-pool-info := ´mem-pool-info(pool b := append-free-list

(´*mem-pool-info* (*pool b*)) (´*lvl t*) (´*blk t*))
FI;;
´*free-block-r* := ´*free-block-r* (*t* := *False*))])
apply *fast* **apply** *fast* **apply** *fast* **apply** *auto*[*1*]
using *mempool-free-stm8-atombody-else* **apply** *blast*
done

lemma *mempool-free-stm8-atombody*:
$\Gamma \vdash_I$ *Some* (´*mem-pool-info* := *set-bit-free* ´*mem-pool-info* (*pool b*) (´*lvl t*) (´*bn t*);;
´*freeing-node* := ´*freeing-node*(*t* := *None*);;
IF NULL < ´*lvl t* ∧ *partner-bits* (´*mem-pool-info* (*pool b*)) (´*lvl t*) (´*bn t*)
THEN ´*i* := ´*i*(*t* := *0*);;
WHILE ´*i t* < *4*
DO ´*bb* := ´*bb*(*t* := ´*bn t div 4* ∗ *4* + ´*i t*);;
´*mem-pool-info* := *set-bit-noexist* ´*mem-pool-info* (*pool b*) (´*lvl t*) (´*bb t*);;
´*block-pt* := ´*block-pt*(*t* := *block-ptr* (´*mem-pool-info* (*pool b*)) (´*lsz t*) (´*bb t*));;
IF ´*bn t* ≠ ´*bb t* ∧
block-fits (´*mem-pool-info* (*pool b*)) (´*block-pt t*)
(´*lsz t*) *THEN* ´*mem-pool-info* := ´*mem-pool-info*
(*pool b* := *remove-free-list* (´*mem-pool-info* (*pool b*)) (´*lvl t*) (´*block-pt t*)) *FI*;;
´*i* := ´*i*(*t* := *Suc* (´*i t*))
OD;;
(´*lvl* := ´*lvl*(*t* := ´*lvl t* − *1*);; ´*bn* := ´*bn*(*t* := ´*bn t div 4*);;
´*mem-pool-info* := *set-bit-freeing* ´*mem-pool-info* (*pool b*) (´*lvl t*) (´*bn t*);;
´*freeing-node* := ´*freeing-node*(*t* ↦
(|*pool* = *pool b*, *level* = ´*lvl t*, *block* = ´*bn t*,
data = *block-ptr* (´*mem-pool-info* (*pool b*)) (*ALIGN4* (*max-sz* (´*mem-pool-info* (*pool b*))) *div 4* ˆ ´*lvl t*) (´*bn t*)|)))
ELSE IF block-fits (´*mem-pool-info* (*pool b*)) (´*blk t*)
(´*lsz t*) *THEN* ´*mem-pool-info* := ´*mem-pool-info*
(*pool b* := *append-free-list* (´*mem-pool-info* (*pool b*)) (´*lvl t*) (´*blk t*)) *FI*;;
´*free-block-r* := ´*free-block-r*(*t* := *False*)
FI) sat_p [*mp-free-precond8-3 t b α* ∩ {|´*cur* = *Some t*|} ∩ {*V*} ∩ *UNIV* ∩ {*Va*},
{(*s*, *t*). *s* = *t*}, *UNIV*,
{|´(*Pair Va*) ∈ *UNIV*|} ∩ ({|´(*Pair V*) ∈ *Mem-pool-free-guar t*|}
∩ (*mp-free-precond8-inv t b* (*α* − *1*) ∪ *mp-free-precond8-inv t b 0*))]

apply(*rule subst*[**where** *s*=*mp-free-precond8-3 t b α* ∩ {|´*cur* = *Some t*|} ∩ {*V*} ∩ {*Va*}
and *t*=*mp-free-precond8-3 t b α* ∩ {|´*cur* = *Some t*|} ∩ {*V*} ∩ *UNIV* ∩ {*Va*}])
apply *blast*
apply(*rule subst*[**where** *s*={|´(*Pair V*) ∈ *Mem-pool-free-guar t*|} ∩ (*mp-free-precond8-inv t b* (*α* − *1*))
and *t*={|´(*Pair Va*) ∈ *UNIV*|} ∩ ({|´(*Pair V*) ∈ *Mem-pool-free-guar t*|} ∩

$(mp\text{-}free\text{-}precond8\text{-}inv\ t\ b\ (\alpha - 1)))])$
apply *blast*

apply(*case-tac* $V \neq Va$)
apply(*rule subst*[**where** $s=\{\}$ **and** $t=mp\text{-}free\text{-}precond8\text{-}3\ t\ b\ \alpha \cap \{\!|\ 'cur = Some\ t|\!\} \cap \{V\} \cap \{Va\}$])
apply *fast* **using** *Emptyprecond*[*of* - $\{(s, t).\ s = t\}$ *UNIV*
$\{\!|\ '(Pair\ Va) \in UNIV|\!\} \cap (\{\!|\ '(Pair\ V) \in Mem\text{-}pool\text{-}free\text{-}guar\ t|\!\} \cap (mp\text{-}free\text{-}precond8\text{-}inv\ t\ b\ (\alpha - 1) \cup mp\text{-}free\text{-}precond8\text{-}inv\ t\ b\ 0))$] **apply** *auto*[*1*]

apply(*case-tac* $mp\text{-}free\text{-}precond8\text{-}3\ t\ b\ \alpha \cap \{\!|\ 'cur = Some\ t|\!\} \cap \{V\} \cap \{Va\} = \{\}$)
using *Emptyprecond* **apply** *metis*

apply(*rule subst*[**where** $s=\{V\}$ **and** $t=mp\text{-}free\text{-}precond8\text{-}3\ t\ b\ \alpha \cap \{\!|\ 'cur = Some\ t|\!\} \cap \{V\} \cap \{Va\}$])
using *two-int-one*[*of* $mp\text{-}free\text{-}precond8\text{-}3\ t\ b\ \alpha \cap \{\!|\ 'cur = Some\ t|\!\}\ V\ Va$] **apply** *fast*
apply(*subgoal-tac* $V \in mp\text{-}free\text{-}precond8\text{-}3\ t\ b\ \alpha \cap \{\!|\ 'cur = Some\ t|\!\}$)
prefer *2* **apply** *fast*

apply(*rule Seq*[**where** $mid=\{free\text{-}stm8\text{-}precond2\ V\ t\ b\}$])
apply(*rule Seq*[**where** $mid=\{free\text{-}stm8\text{-}precond1\ V\ t\ b\}$])

apply(*rule Basic*)
apply *force*
apply *fast* **using** *stable-id2* **apply** *fast* **using** *stable-id2* **apply** *fast*

apply(*rule Basic*)
using *mempool-free-stm8-atombody-h1* **apply** *fast*
using *stable-id2* **apply** *fast* **using** *stable-id2* **apply** *fast* **using** *stable-id2* **apply** *fast*

apply(*rule Cond*)
using *stable-id2* **apply** *fast*
apply(*case-tac* $\{free\text{-}stm8\text{-}precond2\ V\ t\ b\} \cap \{\!|NULL < \ 'lvl\ t \wedge partner\text{-}bits\ ('mem\text{-}pool\text{-}info\ (pool\ b))\ ('lvl\ t)\ ('bn\ t)|\!\} = \{\}$)
using *Emptyprecond* **apply** *metis*

apply(*rule Seq*[**where** $mid=free\text{-}stm8\text{-}precond4\ Va\ t\ b$])
apply(*rule Seq*[**where** $mid=free\text{-}stm8\text{-}precond3\ Va\ t\ b$])

apply(*rule Basic*)
apply *simp* **apply**(*simp add*:*gvars-conf-stable-def gvars-conf-def lvars-nochange-def*)

apply *fast* **using** *stable-id2* **apply** *fast* **using** *stable-id2* **apply** *fast*

apply(*rule While*)
using *stable-id2* **apply** *fast* **apply**(*simp add*:*Let-def*) **apply** *auto*[*1*] **using** *stable-id2* **apply** *fast*
using *mempool-free-stm8-set4partbits-while*[*of V Va t b* α] **apply** *fast*
apply *fast*

apply(*rule subst*[**where** *s*=⦃´(*Pair V*) ∈ *Mem-pool-free-guar t*⦄
∩ (*mp-free-precond8-inv t b* (α − *1*) ∪ *mp-free-precond8-inv t b 0*)
and *t*=⦃´(*Pair Va*) ∈ *UNIV*⦄ ∩ (⦃´(*Pair V*) ∈ *Mem-pool-free-guar t*⦄
∩ (*mp-free-precond8-inv t b* (α − *1*) ∪ *mp-free-precond8-inv t b 0*))])
apply *auto*[*1*]
using *mempool-free-stm8-atombody-rest-extpost*[*of V t b* α] **apply** *fast*
apply(*rule subst*[**where** *s*=⦃´(*Pair V*) ∈ *Mem-pool-free-guar t*⦄
∩ (*mp-free-precond8-inv t b* (α − *1*) ∪ *mp-free-precond8-inv t b 0*)
and *t*=⦃´(*Pair Va*) ∈ *UNIV*⦄ ∩ (⦃´(*Pair V*) ∈ *Mem-pool-free-guar t*⦄
∩ (*mp-free-precond8-inv t b* (α − *1*) ∪ *mp-free-precond8-inv t b 0*))])
apply *auto*[*1*]
using *mempool-free-stm8-atombody-else-extpost*[*of V t b* α] **apply** *fast*

apply *fast*
done

lemma {(*s*,*t*). *s* = *t*} = *Id* **by** *auto*

abbreviation *st8-while-body t b* ≡
(*t* ▶ ´*lsz* := ´*lsz* (*t* := ´*lsizes t* ! (´*lvl t*)));;
(*t* ▶ ´*blk* := ´*blk* (*t* := *block-ptr* (´*mem-pool-info* (*pool b*)) (´*lsz t*) (´*bn t*)));;

(*t* ▶ *ATOMIC*

´*mem-pool-info* := *set-bit-free* ´*mem-pool-info* (*pool b*) (´*lvl t*) (´*bn t*);;
´*freeing-node* := ´*freeing-node* (*t* := *None*);; (∗ *remove the freeing node info of the thread* ∗)

IF ´*lvl t* > *0* ∧ *partner-bits* (´*mem-pool-info* (*pool b*)) (´*lvl t*) (´*bn t*) *THEN*
FOR ´*i* := ´*i*(*t* := *0*);
´*i t* < *4*;
´*i* := ´*i*(*t* := ´*i t* + *1*) *DO*
´*bb* := ´*bb* (*t* := (´*bn t div 4*) ∗ *4* + ´*i t*);;
(∗(*t* ▶ ´*mem-pool-info* := *clear-free-bit* ´*mem-pool-info* (*pool b*) (´*lvl t*) (´*bb t*));;∗)

```
´mem-pool-info := set-bit-noexist ´mem-pool-info (pool b) (´lvl t) (´bb t);;
´block-pt := ´block-pt (t := block-ptr (´mem-pool-info (pool b)) (´lsz t) (´bb
t));;
IF ´bn t ≠ ´bb t ∧ block-fits (´mem-pool-info (pool b))
                    (´block-pt t)
                    (´lsz t) THEN

  (* sys-dlist-remove(block-ptr(p, lsz, b)); *)
  ´mem-pool-info := ´mem-pool-info ((pool b) :=
        remove-free-list (´mem-pool-info (pool b)) (´lvl t) (´block-pt t))
FI
ROF;;

(
(*´j := ´j (t := ´lvl t);; (* use lbn and j to store the previous lvl and bn, or
can not give the post condition *)
´lbn := ´lbn (t := ´bn t);; (* since the lbn and j are not used in M-pool-free
*)
´lvl := ´lvl (t := ´j t − 1);;
´bn := ´bn (t := ´lbn t div 4);;*)
´lvl := ´lvl (t := ´lvl t − 1);;
´bn := ´bn (t := ´bn t div 4);;
(* we add this statement. set the parent node from divided to freeing *)
´mem-pool-info := set-bit-freeing ´mem-pool-info (pool b) (´lvl t) (´bn t);;
(*´freeing-node := ´freeing-node (t := Some (|pool = (pool b), level = (´lvl
t),
           block = (´bn t), data = block-ptr (´mem-pool-info (pool b)) (´lsz t)
(´bn t) |))*)
´freeing-node := ´freeing-node (t := Some (|pool = (pool b), level = (´lvl t),
         block = (´bn t),
         data = block-ptr (´mem-pool-info (pool b))
              (((ALIGN4 (max-sz (´mem-pool-info (pool b)))) div (4 ^ (´lvl
t))))
              (´bn t) |))
)

ELSE
  IF block-fits (´mem-pool-info (pool b)) (´blk t) (´lsz t) THEN

    (* sys-dlist-append(&p−>levels[level].free-list, block); *)
    ´mem-pool-info := ´mem-pool-info ((pool b) :=
          append-free-list (´mem-pool-info (pool b)) (´lvl t) (´blk t) )
  FI;;

  ´free-block-r := ´free-block-r (t := False)
FI

END)
```

lemma *mp-free-precond8-inv-0-stb* :
stable (*mp-free-precond8-inv t b* ($\alpha - 1$) $\cup$ *mp-free-precond8-inv t b 0*) (*Mem-pool-free-rely t*)
  apply(*rule stable-un2*)
  using *mp-free-precond8-inv-stb*[*of t b* $\alpha - 1$] **apply** *fast*
  using *mp-free-precond8-inv-stb*[*of t b 0*] **apply** *fast*
done

lemma *mempool-free-stm8-body-terminate*:
$\Gamma \vdash_I$ *Some* (*st8-while-body t b*)
 sat_p [*mp-free-precond8-inv t b* $\alpha \cap \{\!|\alpha > 0|\!\}$, *Mem-pool-free-rely t*, *Mem-pool-free-guar t*,
  mp-free-precond8-inv t b ($\alpha - 1$) $\cup$ *mp-free-precond8-inv t b 0*]
apply(*rule Seq*[**where** *mid=mp-free-precond8-3 t b* α])
apply(*rule Seq*[**where** *mid=mp-free-precond8-2 t b* α])

apply(*unfold stm-def*)[*1*]
apply(*rule Await*)
using *mp-free-precond8-1-stb*[*of t b* α] **apply** *blast*
using *mp-free-precond8-2-stb*[*of t b* α] **apply** *blast*

apply(*rule allI*)
  apply(*rule Basic*)
  apply(*case-tac mp-free-precond8-1 t b* $\alpha \cap \{\!|$´*cur* = *Some t*$|\!\} \cap \{V\} = \{\}$)
  apply *auto*[*1*] **apply** *clarsimp* **apply**(*rule conjI*)
   apply(*simp add*:$Guar_f$-*def gvars-conf-stable-def gvars-conf-def Mem-pool-free-guar-def*)
    apply(*rule disjI1*)
    apply(*rule conjI*)
   apply(*subgoal-tac* (V,V(|*lsz* := (*lsz V*)(*t* := *lsizes V t* ! (*lvl V t*))|))$\in$*lvars-nochange1-4all*)
    using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def*)
    apply(*simp add*:*lvars-nochange-def*)
   apply(*subgoal-tac* (V,V(|*lsz* := (*lsz V*)(*t* := *lsizes V t* ! (*lvl V t*))|))$\in$*lvars-nochange1-4all*)
    using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def*)
  apply *fast* **using** *stable-id2* **apply** *fast* **using** *stable-id2* **apply** *fast*

apply(*unfold stm-def*)[*1*]
apply(*rule Await*)
using *mp-free-precond8-2-stb* **apply** *blast*
using *mp-free-precond8-3-stb* **apply** *blast*
apply(*rule allI*)
  apply(*rule Basic*)
  apply(*case-tac mp-free-precond8-2 t b* $\alpha \cap \{\!|$´*cur* = *Some t*$|\!\} \cap \{V\} = \{\}$)
  apply *auto*[*1*] **apply** *clarsimp* **apply**(*rule conjI*)
 apply(*simp add*:$Guar_f$-*def gvars-conf-stable-def gvars-conf-def Mem-pool-free-guar-def*)

apply(*rule disjI1*)
apply(*rule conjI*)
apply(*subgoal-tac* (V,V(|*blk* := (*blk* V)
(t := *block-ptr* (*mem-pool-info* V (*pool* b)) (*ALIGN4* (*max-sz* (*mem-pool-info* V (*pool* b))) *div* *4* ˆ *lvl* V t)
(*block* b *div* *4* ˆ (*level* b − *lvl* V t)))|))∈*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def*)
apply(*simp add*:*lvars-nochange-def*)
apply(*subgoal-tac* (V,V(|*blk* := (*blk* V)
(t := *block-ptr* (*mem-pool-info* V (*pool* b)) (*ALIGN4* (*max-sz* (*mem-pool-info* V (*pool* b))) *div* *4* ˆ *lvl* V t)
(*block* b *div* *4* ˆ (*level* b − *lvl* V t)))|))∈*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def*)
apply *fast* **using** *stable-id2* **apply** *fast* **using** *stable-id2* **apply** *fast*

apply(*unfold stm-def*)[*1*]
apply(*rule Await*)
using *mp-free-precond8-3-stb* **apply** *blast*
using *mp-free-precond8-inv-0-stb*[*of t b* α] **apply** *fast*
apply(*rule allI*)
apply(*rule Await*)
using *stable-id2* **apply** *blast* **using** *stable-id2* **apply** *blast*
apply *clarify* **using** *mempool-free-stm8-atombody*[*of b t* α] **apply** *auto*[*1*]

done

lemma *loopbody-sat-invterm-imp-inv-post′*:
$\Gamma \vdash_I$ *Some P sat*$_p$ [*mp-free-precond8-inv t b* α ∩ {|$\alpha > 0$|}, *rely*, *guar*, *mp-free-precond8-inv t b* (α − *1*) ∪ *mp-free-precond8-inv t b 0*]
$\Longrightarrow$ $\Gamma \vdash_I$ *Some P sat*$_p$ [*mp-free-precond8-inv t b* α ∩ {|$\alpha > 0$|}, *rely*, *guar*,*mp-free-precond8 t b*]
using *Conseq* [*of mp-free-precond8-inv t b* α ∩ {|$\alpha > 0$|} *mp-free-precond8-inv t b* α ∩ {|$\alpha > 0$|}
rely rely guar guar mp-free-precond8-inv t b (α − *1*) ∪ *mp-free-precond8-inv t b 0*
mp-free-precond8 t b Some P] **by** *blast*

lemma *stm8-inv-imp-prepost2′*:
($\forall \alpha$. $\Gamma \vdash_I$ *Some P sat*$_p$ [*mp-free-precond8-inv t b* α ∩ {|$\alpha > 0$|}, *rely*, *guar*,
mp-free-precond8-inv t b (α − *1*) ∪ *mp-free-precond8-inv t b 0*])
$\Longrightarrow$ $\Gamma \vdash_I$ *Some P sat*$_p$ [*mp-free-precond8 t b* ∩ {|´*free-block-r t*|}, *rely*, *guar*,*mp-free-precond8 t b*]

apply(*rule subst*[**where** *s*=∀ *v*. *v*∈*mp-free-precond8 t b* ∩ {|´*free-block-r t*|} ⟶

$\Gamma \vdash_I$ *Some P sat*$_p$ [$\{v\}$, *rely*, *guar*,*mp-free-precond8 t b*] **and**
t=$\Gamma \vdash_I$ *Some P sat*$_p$ [*mp-free-precond8 t b* $\cap$ $\{\!|$´*free-block-r t*$|\!\}$, *rely*, *guar*,*mp-free-precond8 t b*]])
using *allpre-eq-pre*[*of mp-free-precond8 t b* $\cap$ $\{\!|$´*free-block-r t*$|\!\}$
Some P rely guar mp-free-precond8 t b] **apply** *blast*

apply(*rule allI*) **apply**(*rule impI*)
apply(*subgoal-tac* $\exists\,\alpha$. *v* $\in$ *mp-free-precond8-inv t b* α $\cap$ $\{\!|\alpha > 0|\!\}$)
prefer *2* **using** *looppre-imp-exist-*α*gt0* **apply** *blast*

apply(*erule exE*)

using *sat-pre-imp-allinpre*[*of Some P - rely guar mp-free-precond8 t b*]
loopbody-sat-invterm-imp-inv-post′ **apply** *blast*
done

lemma *mempool-free-stm8-body*:
$\Gamma \vdash_I$ *Some* (*st8-while-body t b*)
sat$_p$ [*mp-free-precond8 t b* $\cap$ $\{\!|$´*free-block-r t*$|\!\}$, *Mem-pool-free-rely t*, *Mem-pool-free-guar t*, *mp-free-precond8 t b*]

using *stm8-inv-imp-prepost2′*[*of* (*st8-while-body t b*) *t b Mem-pool-free-rely t Mem-pool-free-guar t*]
mempool-free-stm8-body-terminate[*of t b*] **apply** *fast*
done

lemma *mempool-free-stm8*:
$\Gamma \vdash_I$ *Some* (*WHILE* ´*free-block-r t DO*
st8-while-body t b
OD)
sat$_p$ [*mp-free-precond8 t b*, *Mem-pool-free-rely t*, *Mem-pool-free-guar t*, *mp-free-precond9 t b*]
apply(*rule While*)
using *mp-free-precond8-stb*[*of t b*] **apply** *blast*
apply *simp-inv* **apply** *auto*[*1*]
using *mp-free-precond9-stb*[*of t b*] **apply** *auto*[*1*]

using *mempool-free-stm8-body*[*of t b*] **apply** *fast*

apply(*simp add*: *Mem-pool-free-guar-def*)

done

9.4 statement 9

abbreviation *stm9-precond-while Va t b*
$\equiv$ $\{$ *V*. *inv V* $\wedge$ *cur V* = *cur Va* $\wedge$ *tick V* = *tick Va* $\wedge$ (*V*,*Va*)$\in$*gvars-conf-stable*

$\wedge$ *freeing-node V t* = *freeing-node Va t* $\wedge$ *allocating-node V t* = *allocating-node*

Va t
  ∧ (∀ *p*. *levels* (*mem-pool-info V p*) = *levels* (*mem-pool-info Va p*))
  ∧ (∀ *p*. *p* ≠ *pool b* ⟶ *mem-pool-info V p* = *mem-pool-info Va p*)
  ∧ (∀ *t'*. *t'* ≠ *t* ⟶ *lvars-nochange t' V Va*)}

lemma *va-precond-while*: *inv Va* ⟹ *Va* ∈ *stm9-precond-while Va t b*
  by (*simp add*:*gvars-conf-stable-def gvars-conf-def lvars-nochange-def*)

lemma *mempool-free-stm9-resch-inv-help*:
  cur V = *Some t* ⟹ *thd-state V t* = *RUNNING* ⟹
   (*SOME ta*. *ta* ≠ *t* ⟶ *thd-state V ta* = *READY*) = *t* ⟹
   V = *V*⦇*cur* := *Some* (*SOME ta*. *ta* ≠ *t* ⟶ *thd-state V ta* = *READY*),
     thd-state := (*thd-state V*)(*t* := *READY*, *SOME ta*. *ta* ≠ *t* ⟶ *thd-state V ta* = *READY* := *RUNNING*)⦈
apply *auto*
proof −
  assume *a1*: *thd-state V t* = *RUNNING*
  assume *a2*: *cur V* = *Some t*
  have (*thd-state V*)(*t* := *RUNNING*) = *thd-state V*
   using *a1* **by** (*metis fun-upd-triv*)
  then show *V* = *V*⦇*cur* := *Some t*, *thd-state* := (*thd-state V*)(*t* := *RUNNING*)⦈
   using *a2* **by** *simp*
qed

lemma *mempool-free-stm9-resch-inv*:
  cur V = *Some t* ⟹ *inv V* ⟹ *inv* (*V*⦇*cur* := *Some* (*SOME ta*. *ta* ≠ *t* ⟶ *thd-state V ta* = *READY*),
    thd-state := (*thd-state V*)(*t* := *READY*, *SOME ta*. *ta* ≠ *t* ⟶ *thd-state V ta* = *READY* := *RUNNING*)⦈)
apply(*subgoal-tac thd-state V t* = *RUNNING*)
  apply(*case-tac* (*SOME ta*. *ta* ≠ *t* ⟶ *thd-state V ta* = *READY*) = *t*)
   apply(*subgoal-tac V* = *V*⦇*cur* := *Some* (*SOME ta*. *ta* ≠ *t* ⟶ *thd-state V ta* = *READY*),
   thd-state := (*thd-state V*)(*t* := *READY*, *SOME ta*. *ta* ≠ *t* ⟶ *thd-state V ta* = *READY* := *RUNNING*)⦈)
   apply *simp* **using** *mempool-free-stm9-resch-inv-help*[*of V t*] **apply** *auto*[*1*]
   apply(*subgoal-tac thd-state V* (*SOME ta*. *ta* ≠ *t* ⟶ *thd-state V ta* = *READY*) = *READY*)
   apply(*simp add*:*inv-def*)
   apply(*rule conjI*) **apply**(*simp add*:*inv-cur-def*) **apply** *auto*[*1*]
   apply(*rule conjI*) **apply**(*simp add*:*inv-thd-waitq-def*) **apply** *auto*[*1*]
   apply(*rule conjI*) **apply**(*simp add*:*inv-mempool-info-def*)
   apply(*rule conjI*) **apply**(*simp add*:*inv-bitmap-freelist-def*)
   apply(*rule conjI*) **apply**(*simp add*:*inv-bitmap-def*)
   apply(*rule conjI*) **apply**(*simp add*: *inv-aux-vars-def mem-block-addr-valid-def*)
   apply(*rule conjI*) **apply**(*simp add*:*inv-bitmap0-def*)
   apply(*rule conjI*) **apply**(*simp add*:*inv-bitmapn-def*)
     apply(*simp add*:*inv-bitmap-not4free-def*)

apply (*metis* (*mono-tags*, *lifting*) *someI-ex*)

apply(*simp add*:*inv-def inv-cur-def*) **apply** *auto*[*1*]
done

lemma *mempool-free-stm9-ifpart-one*:
$Va \in mp\text{-}free\text{-}precond9\ t\ b \cap \{\!|\ ´cur = Some\ t|\!\} \Longrightarrow$
$V \in stm9\text{-}precond\text{-}while\ Va\ t\ b \cap \{\!|wait\text{-}q\ (´mem\text{-}pool\text{-}info\ (pool\ b)) = []|\!\} \Longrightarrow$
$\Gamma \vdash_I Some\ (IF\ ´need\text{-}resched\ t\ THEN\ reschedule\ FI\ )$
$sat_p\ [\{V\},\ \{(x,\ y).\ x = y\},\ UNIV,\ \{\!|´(Pair\ Va) \in Mem\text{-}pool\text{-}free\text{-}guar\ t|\!\} \cap Mem\text{-}pool\text{-}free\text{-}post\ t]$
apply(*rule Cond*)
apply(*simp add*:*stable-def*)

apply(*simp add*:*reschedule-def*)
apply(*rule Seq*[**where** *mid*=$\{V(|thd\text{-}state := (thd\text{-}state\ V)(the\ (cur\ V) := READY)|)$
$(|cur := Some\ (SOME\ t.\ (thd\text{-}state\ (V(|thd\text{-}state := (thd\text{-}state\ V)(the\ (cur\ V) := READY)|)))\ t = READY)|)\}$])
apply(*rule Seq*[**where** *mid*=$\{V(|thd\text{-}state := (thd\text{-}state\ V)(the\ (cur\ V) := READY)|)\}$])
apply(*rule Basic*)
apply *auto*[*1*] **apply**(*simp add*:*stable-def*)+
apply(*rule Basic*)
apply *auto*[*1*] **apply**(*simp add*:*stable-def*)+
apply(*rule Basic*)
apply *auto*[*1*] **apply**(*simp add*:*Mem-pool-free-guar-def*) **apply**(*rule disjI1*)
apply(*rule conjI*) **apply**(*simp add*:*gvars-conf-stable-def gvars-conf-def*)
apply(*rule conjI*) **using** *mempool-free-stm9-resch-inv* **apply** *auto*[*1*]
apply(*simp add*:*lvars-nochange-def*) **apply**(*simp add*:*Mem-pool-free-post-def*)
using *mempool-free-stm9-resch-inv* **apply** *auto*[*1*] **apply** *auto*[*1*] **apply**(*simp add*:*stable-def*)+

apply(*simp add*:*Skip-def*)
apply(*rule Basic*) **apply** *auto*[*1*] **apply**(*simp add*:*Mem-pool-free-guar-def*)
apply(*rule disjI1*) **apply**(*rule conjI*) **apply**(*simp add*:*gvars-conf-stable-def gvars-conf-def*)
apply(*simp add*:*lvars-nochange-def*)
apply(*simp add*:*Mem-pool-free-post-def*)
apply(*simp add*:*stable-def*)+
done

lemma *mempool-free-stm9-ifpart*:
$Va \in mp\text{-}free\text{-}precond9\ t\ b \cap \{\!|\ ´cur = Some\ t|\!\} \Longrightarrow$
$\Gamma \vdash_I Some\ (IF\ ´need\text{-}resched\ t\ THEN\ reschedule\ FI\ )$
$sat_p\ [stm9\text{-}precond\text{-}while\ Va\ t\ b \cap \{\!|wait\text{-}q\ (´mem\text{-}pool\text{-}info\ (pool\ b)) = []|\!\},$
$\{(x,\ y).\ x = y\},\ UNIV,\ \{\!|´(Pair\ Va) \in Mem\text{-}pool\text{-}free\text{-}guar\ t|\!\} \cap Mem\text{-}pool\text{-}free\text{-}post\ t]$

using *mempool-free-stm9-ifpart-one*[*of Va t b*]
Allprecond[**where** *U*=*stm9-precond-while Va t b* ∩ ⦃*wait-q* (´*mem-pool-info* (*pool b*)) = []⦄ **and**
P= *Some* (*IF* ´*need-resched t THEN reschedule FI*) **and** *rely*={(*x*, *y*). *x* = *y*} **and**
guar= *UNIV* **and** *post*= ⦃´(*Pair Va*) ∈ *Mem-pool-free-guar t*⦄ ∩ *Mem-pool-free-post t*]
by *blast*

lemma *mempool-free-stm9-loopbody-one*:
Va ∈ *mp-free-precond9 t b* ∩ ⦃´*cur* = *Some t*⦄ ⟹
Vb ∈ *stm9-precond-while Va t b* ∩ ⦃*wait-q* (´*mem-pool-info* (*pool b*)) ≠ []⦄ ⟹
$\Gamma \vdash_I$ *Some* (´*th* := ´*th*(*t* := *hd* (*wait-q* (´*mem-pool-info* (*pool b*))));;
´*mem-pool-info* := ´*mem-pool-info*
(*pool b* := ´*mem-pool-info* (*pool b*)⦇*wait-q* := *tl* (*wait-q* (´*mem-pool-info* (*pool b*)))⦈);;
´*thd-state* := ´*thd-state*(´*th t* := *READY*);;
´*need-resched* := ´*need-resched*(*t* := *True*))
sat_p [{*Vb*},{(*x*, *y*). *x* = *y*}, *UNIV*, *stm9-precond-while Va t b*]
apply(*rule Seq*[**where** *mid*={*Vb*⦇*th* := (*th Vb*) (*t* := *hd* (*wait-q* ((*mem-pool-info Vb*) (*pool b*))))⦈
⦇*mem-pool-info* := (*mem-pool-info Vb*)
(*pool b* := (*mem-pool-info Vb*) (*pool b*)⦇*wait-q* := *tl* (*wait-q* ((*mem-pool-info Vb*) (*pool b*)))⦈)⦈
⦇*thd-state* := (*thd-state Vb*)(*hd* (*wait-q* ((*mem-pool-info Vb*) (*pool b*))) := *READY*) ⦈}])

apply(*rule Seq*[**where** *mid*={*Vb*⦇*th* := (*th Vb*) (*t* := *hd* (*wait-q* ((*mem-pool-info Vb*) (*pool b*))))⦈
⦇*mem-pool-info* := (*mem-pool-info Vb*)
(*pool b* := (*mem-pool-info Vb*) (*pool b*)⦇*wait-q* := *tl* (*wait-q* ((*mem-pool-info Vb*) (*pool b*)))⦈)⦈}])

apply(*rule Seq*[**where** *mid*={*Vb*⦇*th* := (*th Vb*) (*t* := *hd* (*wait-q* ((*mem-pool-info Vb*) (*pool b*))))⦈}])

apply(*rule Basic*) **apply** *auto*[*1*] **apply** *simp* **apply**(*simp add*:*stable-def*)+
apply(*rule Basic*) **apply** *auto*[*1*] **apply** *simp* **apply**(*simp add*:*stable-def*)+
apply(*rule Basic*) **apply** *auto*[*1*] **apply** *simp* **apply**(*simp add*:*stable-def*)+

apply(*rule Basic*) **apply** *clarify* **apply**(*rule conjI*) **apply**(*simp add*:*gvars-conf-stable-def gvars-conf-def*)
apply(*simp add*:*inv-def*)
apply(*rule conjI*) **apply**(*simp add*:*inv-cur-def inv-thd-waitq-def*)
apply(*rule conjI*) **apply**(*simp add*: *inv-thd-waitq-def*) **apply** *clarify*
apply(*rule conjI*) **apply** *clarify* **apply** (*rule conjI*) **apply** *clarify* **apply**(*rule conjI*) **apply** *clarify*
apply (*smt List.nth-tl Nitpick.size-list-simp*(*2*) *Suc-mono gr-implies-not0 hd-conv-nth in-set-conv-nth length-pos-if-in-set lessI list.set-sel*(*1*))

apply *clarify* **apply** (*meson list.set-sel*(*2*)) **apply** *clarify* **apply** (*metis list.set-sel*(*1*))

apply(*rule conjI*) **apply** *clarify* **apply** (*metis hd-Cons-tl set-ConsD*)
apply(*rule conjI*) **apply** *clarify* **apply** (*metis* (*no-types*, *lifting*) *List.nth-tl Nitpick.size-list-simp*(*2*)
One-nat-def Suc-mono length-tl nat.inject)
apply *clarify* **apply**(*rule conjI*) **apply** *clarify* **apply** (*metis list.set-sel*(*2*))
apply *clarify* **apply**(*rule conjI*) **apply** *clarify* **apply** (*metis list.set-sel*(*2*))
apply *clarify* **apply** *metis*
apply(*rule conjI*) **apply**(*simp add*: *inv-mempool-info-def*) **apply** *auto*[*1*]
apply(*rule conjI*) **apply**(*simp add*: *inv-bitmap-freelist-def*)
apply(*rule conjI*) **apply**(*simp add*: *inv-bitmap-def*)
apply(*rule conjI*) **apply**(*simp add*:*inv-aux-vars-def mem-block-addr-valid-def*)
apply(*rule conjI*) **apply** *metis* **apply** *metis*
apply(*rule conjI*) **apply**(*simp add*:*inv-bitmap0-def*)
apply(*rule conjI*) **apply**(*simp add*:*inv-bitmapn-def*)
apply(*simp add*:*inv-bitmap-not4free-def partner-bits-def*)

apply(*rule conjI*) **apply** *auto*[*1*]
apply(*rule conjI*) **apply** *auto*[*1*]
apply(*rule conjI*) **apply**(*simp add*:*gvars-conf-stable-def gvars-conf-def*)
apply(*rule conjI*) **apply** *auto*[*1*]
apply(*rule conjI*) **apply** *force*
apply(*rule conjI*)
apply *clarify* **apply**(*simp add*:*lvars-nochange-def*)
apply(*simp add*:*lvars-nochange-def*)
by (*simp add*:*stable-def*)+

lemma *mempool-free-stm9-loopbody*:
Va ∈ *mp-free-precond9 t b* ∩ ⦃´*cur* = *Some t*⦄ ⟹
Γ $\vdash_I$ *Some* (´*th* := ´*th*(*t* := *hd* (*wait-q* (´*mem-pool-info* (*pool b*))));;
´*mem-pool-info* := ´*mem-pool-info*
(*pool b* := ´*mem-pool-info* (*pool b*)⦇*wait-q* := *tl* (*wait-q* (´*mem-pool-info* (*pool b*)))⦈));;
´*thd-state* := ´*thd-state*(´*th t* := *READY*);;
´*need-resched* := ´*need-resched*(*t* := *True*))
sat$_p$ [*stm9-precond-while Va t b* ∩ ⦃*wait-q* (´*mem-pool-info* (*pool b*)) ≠ []⦄,
{(*x*, *y*). *x* = *y*}, *UNIV*, *stm9-precond-while Va t b*]
using *mempool-free-stm9-loopbody-one*
Allprecond[**where** *U*=*stm9-precond-while Va t b* ∩ ⦃*wait-q* (´*mem-pool-info* (*pool b*)) ≠ []⦄ **and**
P= *Some* (´*th* := ´*th*(*t* := *hd* (*wait-q* (´*mem-pool-info* (*pool b*))));;
´*mem-pool-info* := ´*mem-pool-info*(*pool b* := ´*mem-pool-info* (*pool b*)⦇*wait-q* := *tl* (*wait-q* (´*mem-pool-info* (*pool b*)))⦈));;
´*thd-state* := ´*thd-state*(´*th t* := *READY*);;
´*need-resched* := ´*need-resched* (*t* := *True*))
and *rely*={(*x*, *y*). *x* = *y*} **and** *guar*= *UNIV* **and** *post*= *stm9-precond-while Va t b*]

by *blast*

lemma *mempool-free-stm9-body-loopinv*:
$Va \in$ *mp-free-precond9 t b* $\cap$ {|´*cur* = *Some t*|} $\Longrightarrow$
$\Gamma \vdash_I$ *Some* (*WHILE wait-q* (´*mem-pool-info* (*pool b*)) $\neq$ []
DO ´*th* := ´*th*(*t* := *hd* (*wait-q* (´*mem-pool-info* (*pool b*))));;
´*mem-pool-info* := ´*mem-pool-info*
(*pool b* := ´*mem-pool-info* (*pool b*)(|*wait-q* := *tl* (*wait-q* (´*mem-pool-info* (*pool b*)))|));;
´*thd-state* := ´*thd-state*(´*th t* := *READY*);;
´*need-resched* := ´*need-resched*(*t* := *True*)
OD;;
IF ´*need-resched t THEN reschedule FI*)
sat_p [*stm9-precond-while Va t b*, {(*x*, *y*). *x* = *y*}, *UNIV*, {*s*. (*Va*,*s*)$\in$*Mem-pool-free-guar t*} $\cap$ *Mem-pool-free-post t*]
apply(*rule Seq*[**where** *mid*=*stm9-precond-while Va t b* $\cap$ {| *wait-q* (´*mem-pool-info* (*pool b*)) = []|}])
apply(*rule While*)
apply(*simp add*:*stable-def*)
apply *auto*[*1*]
apply(*simp add*:*stable-def*)
using *mempool-free-stm9-loopbody*[*of Va t b*] **apply** *simp*
apply *simp*

using *mempool-free-stm9-ifpart* **by** *blast*

lemma *mempool-free-stm9-body*:
mp-free-precond9 t b $\cap$ {|´*inv*|} $\cap$ {|´*cur* = *Some t*|} $\cap$ {*Va*} $\neq$ {} $\Longrightarrow$
$\Gamma \vdash_I$ *Some* (*WHILE wait-q* (´*mem-pool-info* (*pool b*)) $\neq$ []
DO ´*th* := ´*th*(*t* := *hd* (*wait-q* (´*mem-pool-info* (*pool b*))));;
´*mem-pool-info* := ´*mem-pool-info*(*pool b* := ´*mem-pool-info* (*pool b*)(|*wait-q* := *tl* (*wait-q* (´*mem-pool-info* (*pool b*)))|));;
´*thd-state* := ´*thd-state*(´*th t* := *READY*);;
´*need-resched* := ´*need-resched*(*t* := *True*)
OD;;
IF ´*need-resched t THEN reschedule FI*)
sat_p [*mp-free-precond9 t b* $\cap$ {|´*cur* = *Some t*|} $\cap$ {*Va*},
{(*s*, *t*). *s* = *t*}, *UNIV*, {|´(*Pair Va*) $\in$ *Mem-pool-free-guar t*|} $\cap$ *Mem-pool-free-post t*]
apply(*subgoal-tac inv Va*) **prefer** *2* **apply** *simp*
apply(*subgoal-tac Va* $\in$ *mp-free-precond9 t b* $\cap$ {|´*inv*|} $\cap$ {|´*cur* = *Some t*|}) **prefer** *2* **apply** *simp*
using *mempool-free-stm9-body-loopinv*[*of Va t b*] *va-precond-while*[*of Va t b*]
Conseq[**where** *pre*={*Va*} **and** *pre'*=*stm9-precond-while Va t b* **and** *rely*={(*x*, *y*). *x* = *y*} **and** *rely'*={(*x*, *y*). *x* = *y*}
and *guar*=*UNIV* **and** *guar'*=*UNIV* **and** *post'*={|´(*Pair Va*) $\in$ *Mem-pool-free-guar t*|} $\cap$ *Mem-pool-free-post t*
and *post*={|´(*Pair Va*) $\in$ *Mem-pool-free-guar t*|} $\cap$ *Mem-pool-free-post t*

and *P=Some* (*WHILE wait-q* (´*mem-pool-info* (*pool b*)) $\neq$ []
DO ´*th* := ´*th*(*t* := *hd* (*wait-q* (´*mem-pool-info* (*pool b*))));;
´*mem-pool-info* := ´*mem-pool-info*
(*pool b* := ´*mem-pool-info* (*pool b*)(|*wait-q* := *tl* (*wait-q* (´*mem-pool-info* (*pool b*)))|));;
´*thd-state* := ´*thd-state*(´*th t* := *READY*);;
´*need-resched* := ´*need-resched*(*t* := *True*)
OD;;
IF ´*need-resched t THEN reschedule FI*)]
apply *force*
done

lemma *mempool-free-stm9*:
Γ $\vdash_I$ *Some* (*t* ► *ATOMIC*
WHILE wait-q (´*mem-pool-info* (*pool b*)) $\neq$ [] *DO*
´*th* := ´*th* (*t* := *hd* (*wait-q* (´*mem-pool-info* (*pool b*))));;
(∗ *-unpend-thread*(*th*); ∗)
´*mem-pool-info* := ´*mem-pool-info* (*pool b* := ´*mem-pool-info* (*pool b*)
(|*wait-q* := *tl* (*wait-q* (´*mem-pool-info* (*pool b*)))|));;
(∗ *-ready-thread*(*th*); ∗)
´*thd-state* := ´*thd-state* (´*th t* := *READY*);;
´*need-resched* := ´*need-resched*(*t* := *True*)
OD;;

IF ´*need-resched t THEN*
reschedule
FI
END)
sat_p [*mp-free-precond9 t b*, *Mem-pool-free-rely t*, *Mem-pool-free-guar t*, *Mem-pool-free-post t*]
apply(*simp add*:*stm-def*)
apply(*rule Await*)
using *mp-free-precond9-stb* **apply** *auto*[*1*]
apply (*simp add*: *mem-pool-free-post-stb*)

apply(*rule allI*)
apply(*rule Await*)
apply(*simp add*:*stable-def*) **apply**(*simp add*:*stable-def*)
apply(*rule allI*)
apply(*case-tac V = Va*) **apply** *simp*
apply(*case-tac mp-free-precond9 t b* ∩ {|´*inv*|} ∩ {|´*cur* = *Some t*|} ∩ {*Va*} = {})
apply *simp* **apply** (*simp add*: *Emptyprecond stable-id2*)
apply *clarify* **using** *mempool-free-stm9-body* **apply** *force*
apply *simp* **apply** (*simp add*: *Emptyprecond stable-id2*)
done

9.5 final proof

lemma *Mempool-free-satRG*: Γ (*Mem-pool-free t b*) ⊢ *Mem-pool-free-RGCond t b*
  apply(*simp add*:*Evt-sat-RG-def*)
  apply(*simp add*:*body-def* Pre_f*-def* $Post_f$*-def guard-def*
    $Rely_f$*-def* $Guar_f$*-def getrgformula-def*)
  apply (*simp add*: *Mem-pool-free-def Mem-pool-free-RGCond-def*)
  apply(*rule BasicEvt*)
    apply(*simp add*:*body-def* Pre_f*-def* $Post_f$*-def guard-def*
      getrgformula-def)
  apply(*rule Seq*[**where** *mid*=*mp-free-precond9 t b*])
  apply(*rule Seq*[**where** *mid*=*mp-free-precond8 t b*])
  apply(*rule Seq*[**where** *mid*=*mp-free-precond7 t b*])
  apply(*rule Seq*[**where** *mid*=*mp-free-precond6 t b*])
  apply(*rule Seq*[**where** *mid*=*mp-free-precond5 t b*])
  apply(*rule Seq*[**where** *mid*=*mp-free-precond4 t b*])
  apply(*rule Seq*[**where** *mid*=*mp-free-precond3 t b*])
  apply(*rule Seq*[**where** *mid*=*mp-free-precond2 t b*])

  using *mempool-free-stm1*[*of t b*] **apply** *fast*
  using *mempool-free-stm2*[*of t b*] **apply** *fast*
  using *mempool-free-stm3*[*of t b*] **apply** *fast*
  using *mempool-free-stm4*[*of t b*] **apply** *force*
  using *mempool-free-stm5*[*of t b*] **apply** *fast*
  using *mempool-free-stm6*[*of t b*] **apply** *fast*
  using *mempool-free-stm7*[*of t b*] **apply** *fast*
  using *mempool-free-stm8*[*of t b*] **apply** *force*
  using *mempool-free-stm9*[*of t b*] **apply** *force*

  apply(*simp add*:*body-def* Pre_f*-def* $Post_f$*-def guard-def*
    $Rely_f$*-def* $Guar_f$*-def getrgformula-def*)
  using *mem-pool-free-pre-stb* **apply** *fast*

  apply(*simp add*:*getrgformula-def Mem-pool-free-guar-def*)
done

end

theory *func-cor-mempoolalloc*
imports *func-cor-lemma*
begin

10 Functional correctness of $k\_mem\_pool\_alloc$

10.1 intermediate conditions and their stable to rely cond

abbreviation *mp-alloc-precond1 t p tm* ≡
Mem-pool-alloc-pre t ∩ {|*p* ∈ ´*mem-pools* ∧ *tm* ≥ −*1*|}

lemma *mp-alloc-precond1-ext-stb*: *stable* ({|*p* ∈ ´*mem-pools* ∧ *tm* ≥ −*1*|}) (*Mem-pool-alloc-rely t*)
apply(*simp add*:*stable-def*) **apply**(*rule allI*) **apply**(*rule impI*) **apply**(*rule allI*)
apply(*rule impI*)
apply(*simp add*:*Mem-pool-alloc-rely-def*)
apply(*simp add*:*gvars-conf-stable-def*)
unfolding *gvars-conf-def* **apply** *metis*
done

lemma *mp-alloc-precond1-stb*: *stable* (*mp-alloc-precond1 t p tm*) (*Mem-pool-alloc-rely t*)
apply(*rule stable-int2*)
apply(*simp add*:*mem-pool-alloc-pre-stb*)
apply(*simp add*:*mp-alloc-precond1-ext-stb*)
done

abbreviation *mp-alloc-precond2 t p tm* ≡
mp-alloc-precond1 t p tm ∩ {|´*tmout t* = *tm*|}

lemma *mp-alloc-precond2-ext-stb*: *stable* ({|´*tmout t* = *tm*|}) (*Mem-pool-alloc-rely t*)
apply(*simp add*:*stable-def*) **apply**(*rule allI*) **apply**(*rule impI*) **apply**(*rule allI*)
apply(*rule impI*)
apply(*simp add*:*Mem-pool-alloc-rely-def*)
apply(*simp add*:*lvars-nochange-rel-def lvars-nochange-def*) **apply** *smt*
done

lemma *mp-alloc-precond2-stb*: *stable* (*mp-alloc-precond2 t p tm*) (*Mem-pool-alloc-rely t*)
apply(*rule stable-int2*)
apply(*simp add*:*mp-alloc-precond1-stb*)
apply(*simp add*:*mp-alloc-precond2-ext-stb*)
done

abbreviation *mp-alloc-precond3 t p tm* ≡
mp-alloc-precond2 t p tm ∩ {|´*endt t* = *0*|}

lemma *mp-alloc-precond3-ext-stb*: *stable* ({|´*endt t* = *0*|}) (*Mem-pool-alloc-rely t*)
apply(*simp add*:*stable-def*) **apply**(*rule allI*) **apply**(*rule impI*) **apply**(*rule allI*)
apply(*rule impI*)
apply(*simp add*:*Mem-pool-alloc-rely-def*)
apply(*simp add*:*lvars-nochange-rel-def lvars-nochange-def*) **apply** *smt*
done

lemma *mp-alloc-precond3-stb*: *stable* (*mp-alloc-precond3 t p tm*) (*Mem-pool-alloc-rely t*)
  apply(*rule stable-int2*)
  apply(*simp add*:*mp-alloc-precond2-stb*)
  apply(*simp add*:*mp-alloc-precond3-ext-stb*)
done

abbreviation *mp-alloc-precond4 t p tm* ≡
  mp-alloc-precond2 t p tm ∩ {|´*endt t* ≥ *0*|}

lemma *mp-alloc-precond4-ext-stb*: *stable* ({|´*endt t* ≥ *0*|}) (*Mem-pool-alloc-rely t*)
apply(*simp add*:*stable-def*)
done

lemma *mp-alloc-precond4-stb*: *stable* (*mp-alloc-precond4 t p tm*) (*Mem-pool-alloc-rely t*)
  apply(*rule stable-int2*)
  apply(*simp add*:*mp-alloc-precond2-stb*)
  using *mp-alloc-precond4-ext-stb* **apply** *auto*
done

abbreviation *mp-alloc-precond5 t p tm* ≡
  mp-alloc-precond4 t p tm ∩ {|´*mempoolalloc-ret t* = *None*|}

lemma *mp-alloc-precond5-ext-stb*: *stable* ({|´*mempoolalloc-ret t* = *None*|}) (*Mem-pool-alloc-rely t*)
apply(*simp add*:*stable-def*) **apply**(*rule allI*) **apply**(*rule impI*) **apply**(*rule allI*) **apply**(*rule impI*)
  apply(*simp add*:*Mem-pool-alloc-rely-def*)
  apply(*simp add*:*lvars-nochange-rel-def lvars-nochange-def*) **apply** *smt*
done

lemma *mp-alloc-precond5-stb*: *stable* (*mp-alloc-precond5 t p tm*) (*Mem-pool-alloc-rely t*)
  apply(*rule stable-int2*)
  using *mp-alloc-precond4-stb* **apply** *auto*[*1*]
  apply(*simp add*:*mp-alloc-precond5-ext-stb*)
done

abbreviation *mp-alloc-precond6 t p tm* ≡
  mp-alloc-precond5 t p tm ∩ {|´*ret t* = *ESIZEERR*|}

lemma *mp-alloc-precond6-ext-stb*: *stable* ({|´*ret t* = *ESIZEERR*|}) (*Mem-pool-alloc-rely t*)
apply(*simp add*:*stable-def*) **apply**(*rule allI*) **apply**(*rule impI*) **apply**(*rule allI*) **apply**(*rule impI*)
  apply(*simp add*:*Mem-pool-alloc-rely-def*)
  apply(*simp add*:*lvars-nochange-rel-def lvars-nochange-def*) **apply** *smt*

done

lemma *mp-alloc-precond6-stb*: *stable* (*mp-alloc-precond6 t p tm*) (*Mem-pool-alloc-rely t*)
  apply(*rule stable-int2*)
  using *mp-alloc-precond5-stb* **apply** *auto*[*1*]
  apply(*simp add*:*mp-alloc-precond6-ext-stb*)
done

abbreviation *mp-alloc-precond7-ext t p sz timeout* $\equiv$
  $\{s.\ (rf\ s\ t \longrightarrow (timeout = FOREVER \longrightarrow (ret\ s\ t = ESIZEERR \wedge mempoolalloc\text{-}ret\ s\ t = None$
                                $\vee\ ret\ s\ t = OK \wedge (\exists\, mblk.\ mempoolalloc\text{-}ret\ s\ t = Some\ mblk \wedge alloc\text{-}memblk\text{-}valid\ s\ p\ sz\ mblk)))$
        $\wedge\ (timeout = NOWAIT \longrightarrow ((ret\ s\ t = ENOMEM \vee ret\ s\ t = ESIZEERR) \wedge mempoolalloc\text{-}ret\ s\ t = None)$
                                  $\vee\ (ret\ s\ t = OK \wedge (\exists\, mblk.\ mempoolalloc\text{-}ret\ s\ t = Some\ mblk \wedge alloc\text{-}memblk\text{-}valid\ s\ p\ sz\ mblk)))$
          $\wedge\ (timeout > 0 \longrightarrow ((ret\ s\ t = ETIMEOUT \vee ret\ s\ t = ESIZEERR) \wedge mempoolalloc\text{-}ret\ s\ t = None)$
                              $\vee\ (ret\ s\ t = OK \wedge (\exists\, mblk.\ mempoolalloc\text{-}ret\ s\ t = Some\ mblk \wedge alloc\text{-}memblk\text{-}valid\ s\ p\ sz\ mblk))))$
      $\wedge\ (\neg rf\ s\ t \longrightarrow mempoolalloc\text{-}ret\ s\ t = None)$
      $\wedge\ (timeout = FOREVER \longrightarrow tmout\ s\ t = FOREVER)\}$

abbreviation *mp-alloc-precond7 t p sz timeout* $\equiv$
  mp-alloc-precond1 t p timeout $\cap$ *mp-alloc-precond7-ext t p sz timeout*

abbreviation *mp-alloc-precond7-inv t p sz timeout* α $\equiv$
  mp-alloc-precond7 t p sz timeout
    $\cap$ $\{\!|\ \alpha = ($*if* $\acute{}rf\ t \vee \acute{}mempoolalloc\text{-}ret\ t \neq None$ *then* 0 (* *if timeout* $= 0$ (*NOWAIT*), *rf is true* *)
            else if timeout > 0 *then* $\acute{}endt\ t - \acute{}tick$
                (* *in rely cond*, $tick' \geq tick$, *thus convergent* $\beta \leq \alpha$, *not* $< \alpha$, *thus not absolutely convergent* *)
                else 1)
                (* *cannot find convergent* α *for FOREVER*, *so just set* 1 *)$|\!\}$

lemma *mp-alloc-precond7-ext-stb*: *stable* (*mp-alloc-precond7-ext t p sz timeout*) (*Mem-pool-alloc-rely t*)
apply(*simp add*:*stable-def*) **apply**(*rule allI*) **apply**(*rule impI*) **apply**(*rule allI*) **apply**(*rule impI*)
  using *mp-alloc-post-stb*
  apply(*simp add*:*Mem-pool-alloc-rely-def*)
  apply(*simp add*:*lvars-nochange-rel-def lvars-nochange-def*)
  apply(*case-tac* $x = y$)
    apply *simp* **apply** *clarify*
    apply(*simp add*:*alloc-memblk-valid-def gvars-conf-def gvars-conf-stable-def*)

done

lemma *mp-alloc-precond7-stb*: *stable* (*mp-alloc-precond7 t p sz timeout*) (*Mem-pool-alloc-rely t*)
  apply(*rule stable-int2*)
  using *mp-alloc-precond1-stb* **apply** *auto*[*1*]
  apply(*simp add*:*mp-alloc-precond7-ext-stb*)
done

abbreviation *mp-alloc-precond1-0 t p sz tm* ≡
  mp-alloc-precond7 t p sz tm ∩ ⦃¬ ´*rf t*⦄

lemma *mp-alloc-precond1-0-ext-stb*: *stable* ⦃¬ ´*rf t*⦄ (*Mem-pool-alloc-rely t*)
apply(*simp add*:*stable-def*) **apply**(*rule allI*) **apply**(*rule impI*) **apply**(*rule allI*)
apply(*rule impI*)
  apply(*simp add*:*Mem-pool-alloc-rely-def*)
  apply(*simp add*:*lvars-nochange-rel-def lvars-nochange-def*) **apply** *smt*
done

lemma *mp-alloc-precond1-0-stb*: *stable* (*mp-alloc-precond1-0 t p sz tm*) (*Mem-pool-alloc-rely t*)
  apply(*rule stable-int2*)
  using *mp-alloc-precond7-stb* **apply** *auto*[*1*]
  apply(*simp add*:*mp-alloc-precond1-0-ext-stb*)
done

abbreviation *mp-alloc-precond1-1 t p sz tm* ≡
  mp-alloc-precond1-0 t p sz tm (∗∩ ⦃´*blk t* = *0*⦄∗)

lemma *mp-alloc-precond1-1-stb*: *stable* (*mp-alloc-precond1-1 t p sz tm*) (*Mem-pool-alloc-rely t*)
  using *mp-alloc-precond1-0-stb* **by** *simp*

abbreviation *mp-alloc-precond1-2 t p sz tm* ≡
  mp-alloc-precond1-1 t p sz tm ∩ ⦃´*alloc-lsize-r t* = *False*⦄

lemma *mp-alloc-precond1-2-stb*: *stable* (*mp-alloc-precond1-2 t p sz tm*) (*Mem-pool-alloc-rely t*)
  apply(*rule stable-int2*)
  using *mp-alloc-precond1-1-stb* **apply** *auto*[*1*]
  apply(*simp add*:*stable-def*) **apply**(*rule allI*) **apply**(*rule impI*) **apply**(*rule allI*)
apply(*rule impI*)
  apply(*simp add*:*Mem-pool-alloc-rely-def*)
  apply(*simp add*:*lvars-nochange-rel-def lvars-nochange-def*) **apply** *smt*
done

abbreviation *mp-alloc-precond1-3 t p sz tm* ≡
  mp-alloc-precond1-2 t p sz tm ∩ ⦃´*alloc-l t* = −*1*⦄

lemma *mp-alloc-precond1-3-ext-stb*: *stable* $\{\!\!\{$ *´alloc-l t* = -1 $\}\!\!\}$ (*Mem-pool-alloc-rely t*)
apply(*simp add*:*stable-def*) **apply**(*rule allI*) **apply**(*rule impI*) **apply**(*rule allI*) **apply**(*rule impI*)
  apply(*simp add*:*Mem-pool-alloc-rely-def*)
  apply(*simp add*:*lvars-nochange-rel-def lvars-nochange-def*) **apply** *smt*
done

lemma *mp-alloc-precond1-3-stb*: *stable* (*mp-alloc-precond1-3 t p sz tm*) (*Mem-pool-alloc-rely t*)
  apply(*rule stable-int2*)
  using *mp-alloc-precond1-2-stb* **apply** *auto*[*1*]
  apply(*simp add*:*mp-alloc-precond1-3-ext-stb*)
done

abbreviation *mp-alloc-precond1-4 t p sz tm* ≡
  mp-alloc-precond1-3 t p sz tm ∩ $\{\!\!\{$ *´free-l t* = -1 $\}\!\!\}$

lemma *mp-alloc-precond1-4-ext-stb*: *stable* $\{\!\!\{$ *´free-l t* = -1 $\}\!\!\}$ (*Mem-pool-alloc-rely t*)
  apply(*simp add*:*stable-def*) **apply**(*rule allI*) **apply**(*rule impI*) **apply**(*rule allI*) **apply**(*rule impI*)
  apply(*simp add*:*Mem-pool-alloc-rely-def*)
  apply(*simp add*:*lvars-nochange-rel-def lvars-nochange-def*) **apply** *smt*
done

lemma *mp-alloc-precond1-4-stb*: *stable* (*mp-alloc-precond1-4 t p sz tm*) (*Mem-pool-alloc-rely t*)
  apply(*rule stable-int2*)
  using *mp-alloc-precond1-3-stb* **apply** *auto*[*1*]
  apply(*simp add*:*mp-alloc-precond1-4-ext-stb*)
done

abbreviation *mp-alloc-precond1-5 t p sz tm* ≡
  mp-alloc-precond1-4 t p sz tm ∩ $\{\!\!\{$ *´lsizes t* = [*ALIGN4* (*max-sz* (*´mem-pool-info p*))] $\}\!\!\}$

lemma *mp-alloc-precond1-5-ext-stb*: *stable* $\{\!\!\{$ *´lsizes t* = [*ALIGN4* (*max-sz* (*´mem-pool-info p*))] $\}\!\!\}$ (*Mem-pool-alloc-rely t*)
  apply(*simp add*:*stable-def*) **apply**(*rule allI*) **apply**(*rule impI*) **apply**(*rule allI*) **apply**(*rule impI*)
  apply(*simp add*:*Mem-pool-alloc-rely-def*)
  apply(*case-tac x=y*) **apply** *simp* **apply** *clarify*
   apply(*simp add*:*lvars-nochange-rel-def lvars-nochange-def gvars-conf-stable-def gvars-conf-def*)
done

lemma *mp-alloc-precond1-5-stb*: *stable* (*mp-alloc-precond1-5 t p sz tm*) (*Mem-pool-alloc-rely*

t)
  apply(*rule stable-int2*)
  using *mp-alloc-precond1-4-stb* **apply** *auto*[*1*]
  apply(*simp add*:*mp-alloc-precond1-5-ext-stb*)
done

abbreviation *mp-alloc-precond1-6-ext t p sz tm* $\equiv$
  $\{\!|(\forall\, ii<length\ ({}'lsizes\ t).\ {}'lsizes\ t\ !\ ii = (ALIGN4\ (max\text{-}sz\ ({}'mem\text{-}pool\text{-}info\ p)))$ $div\ (4\ \hat{}\ ii))$
    $\wedge\ length\ ({}'lsizes\ t) \leq n\text{-}levels\ ({}'mem\text{-}pool\text{-}info\ p)$
    $\wedge\ ({}'i\ t = 0 \longrightarrow {}'alloc\text{-}l\ t = -1 \wedge {}'free\text{-}l\ t = -1 \wedge length\ ({}'lsizes\ t) = 1)$
    $\wedge\ {}'i\ t \leq n\text{-}levels\ ({}'mem\text{-}pool\text{-}info\ p)$
    $\wedge\ -1 \leq {}'free\text{-}l\ t \wedge {}'free\text{-}l\ t \leq\ int\ ({}'i\ t) - 1 \wedge {}'free\text{-}l\ t \leq {}'alloc\text{-}l\ t$
    $\wedge\ {}'alloc\text{-}l\ t = int\ ({}'i\ t) - 1$
    $\wedge\ ({}'alloc\text{-}l\ t \geq 0 \longrightarrow (\forall\, ii.\ ii \leq nat\ ({}'alloc\text{-}l\ t) \longrightarrow {}'lsizes\ t\ !\ ii \geq sz))$
    $\wedge\ (\neg\ {}'alloc\text{-}lsize\text{-}r\ t \longrightarrow ({}'i\ t = 0 \longrightarrow length\ ({}'lsizes\ t) = 1) \wedge ({}'i\ t > 0 \longrightarrow$ $length\ ({}'lsizes\ t) = {}'i\ t\ ))$
      $\wedge\ ({}'alloc\text{-}lsize\text{-}r\ t \longrightarrow length\ ({}'lsizes\ t) = {}'i\ t + 1 \wedge {}'i\ t < n\text{-}levels$ $({}'mem\text{-}pool\text{-}info\ p) \wedge {}'lsizes\ t\ !\ ({}'i\ t) < sz)|\!\}$

abbreviation *mp-alloc-precond1-6 t p sz tm* $\equiv$
  mp-alloc-precond1-1 t p sz tm $\cap$ *mp-alloc-precond1-6-ext t p sz tm*

abbreviation *mp-alloc-lsizeloop-α-cond t p α* $\equiv$
  $\{\!|\alpha = (if\ \ {}'alloc\text{-}lsize\text{-}r\ t\ (*{}'lsizes\ t\ !\ ({}'i\ t) < sz\ *)$
    $then\ 0\ else\ n\text{-}levels\ ({}'mem\text{-}pool\text{-}info\ p) - {}'i\ t)\ |\!\}$

abbreviation *mp-alloc-lsizestm-loopinv t p sz tm α* $\equiv$
  mp-alloc-precond1-6 t p sz tm $\cap$ *mp-alloc-lsizeloop-α-cond t p α*

abbreviation *mp-alloc-lsizestm-loopcond t p* $\equiv$ $\{\!|\ {}'i\ t < n\text{-}levels\ ({}'mem\text{-}pool\text{-}info$ $p) \wedge \neg\ {}'alloc\text{-}lsize\text{-}r\ t\ |\!\}$

lemma *lsizestm-loopinv-imp-precond*:
mp-alloc-lsizestm-loopinv t p sz tm α $\subseteq$ *mp-alloc-precond1-6 t p sz tm*
by *auto*

lemma *lsizestm-loopinv-αgt0-imp-loopcond*:
mp-alloc-lsizestm-loopinv t p sz tm α $\cap$ $\{\!|\alpha > 0|\!\}$ $\subseteq$ *mp-alloc-lsizestm-loopcond t p*
by *clarsimp*

lemma *lsizestm-loopinv-αeq0-imp-notloopcond*:
mp-alloc-lsizestm-loopinv t p sz tm α $\cap$ $\{\!|\alpha = 0|\!\}$ $\subseteq -$ *mp-alloc-lsizestm-loopcond t p*
by *clarsimp*

lemma *lsizestm-loopinv-αeq0-imp-notloopcond2*:

mp-alloc-lsizestm-loopinv t p sz tm 0 ⊆ − *mp-alloc-lsizestm-loopcond t p*
by *clarsimp*

lemma *lsizestm-pre-loopcond-imp-loopinv-αgt0*:
x∈*mp-alloc-precond1-6 t p sz tm* ∩ *mp-alloc-lsizestm-loopcond t p* ⟹
  ∃ *α*. *x*∈*mp-alloc-lsizestm-loopinv t p sz tm α* ∩ {|*α* > *0*|}
by *clarsimp*

lemma *lsizestm-pre-notloopcond-imp-loopinv-αeq0*:
x∈*mp-alloc-precond1-6 t p sz tm* ∩ − *mp-alloc-lsizestm-loopcond t p* ⟹
  x∈*mp-alloc-lsizestm-loopinv t p sz tm 0*
apply *clarsimp*
apply(*rule conjI*)
  apply *clarify* **apply** *simp*
  apply *clarify* **apply** *simp*
done

lemma *lsizestm-pre-notloopcond-imp-loopinv-αeq0'*:
mp-alloc-precond1-6 t p sz tm ∩ − *mp-alloc-lsizestm-loopcond t p*
    ⊆ *mp-alloc-lsizestm-loopinv t p sz tm 0*
apply *clarsimp*
apply(*rule conjI*) **apply** *clarify*
apply(*rule conjI*) **apply** *clarify*
apply(*rule conjI*) **apply** *clarify*
apply(*rule conjI*) **apply** *clarify*
apply(*rule conjI*) **apply** *clarify*
apply *clarify* **apply** *simp*
done

lemma *lsizestm-pre-notloopcond-eq-loopinv-αeq0*:
mp-alloc-precond1-6 t p sz tm ∩ − *mp-alloc-lsizestm-loopcond t p*
    = *mp-alloc-lsizestm-loopinv t p sz tm 0*
apply(*rule subset-antisym*)
using *lsizestm-pre-notloopcond-imp-loopinv-αeq0'*[*of t p tm sz*] **apply** *blast*
apply(*rule Int-greatest*)
using *lsizestm-loopinv-imp-precond*[*of t p tm sz 0*] **apply** *blast*
using *lsizestm-loopinv-αeq0-imp-notloopcond2*[*of t p tm sz*] **apply** *blast*
done

lemma *lsizeloop-inv-cond-eq-αgt0*:
mp-alloc-lsizestm-loopinv t p sz tm α ∩ *mp-alloc-lsizestm-loopcond t p*
    = *mp-alloc-lsizestm-loopinv t p sz tm α* ∩ {|*α* > *0*|}
apply(*rule subset-antisym*)
apply(*rule Int-greatest*)
  apply *fast*
  apply *clarify* **apply** *auto*[*1*]
apply(*rule Int-greatest*)
  apply *fast*
  apply *clarsimp*

done

lemma *mp-alloc-precond1-6-ext-stb*: *stable* (*mp-alloc-precond1-6-ext t p sz tm*) (*Mem-pool-alloc-rely t*)
  apply(*simp add:stable-def*) **apply** *clarify*
  apply(*simp add:Mem-pool-alloc-rely-def*)
  apply(*case-tac x=y*) **apply** *auto*[*1*] **apply** *clarify*
  apply(*simp add:lvars-nochange-rel-def lvars-nochange-def gvars-conf-stable-def gvars-conf-def*)
 apply(*rule conjI*) **apply** *clarify* **apply**(*simp add:lvars-nochange-rel-def lvars-nochange-def gvars-conf-stable-def gvars-conf-def*)
 apply *clarify* **apply**(*simp add:lvars-nochange-rel-def lvars-nochange-def gvars-conf-stable-def gvars-conf-def*)
done

lemma *mp-alloc-precond1-6-stb*: *stable* (*mp-alloc-precond1-6 t p sz tm*) (*Mem-pool-alloc-rely t*)
  apply(*rule stable-int2*)
  using *mp-alloc-precond1-1-stb* **apply** *auto*[*1*]
  using *mp-alloc-precond1-6-ext-stb* **apply** *auto*
done

lemma *mp-alloc-lsizeloop-α-cond-stb*: *stable* (*mp-alloc-lsizeloop-α-cond t p α*) (*Mem-pool-alloc-rely t*)
apply(*simp add:stable-def*) **apply** *clarify*
apply(*simp add:Mem-pool-alloc-rely-def*) **apply** *auto*
apply(*simp add:lvars-nochange-rel-def lvars-nochange-def gvars-conf-stable-def gvars-conf-def*)+
done

lemma *mp-alloc-lsizestm-loopinv-stb*: *stable* (*mp-alloc-lsizestm-loopinv t p sz tm α*) (*Mem-pool-alloc-rely t*)
apply(*rule stable-int2*)
using *mp-alloc-precond1-6-stb* **apply** *fast*
using *mp-alloc-lsizeloop-α-cond-stb* **apply** *fast*
done

lemma *mp-alloc-lsizestm-loopinv-presv-rely*:
$s \in$ *mp-alloc-lsizestm-loopinv t p sz tm* $\alpha \Longrightarrow (s,r) \in$ *Mem-pool-alloc-rely t* $\Longrightarrow \exists\, \beta \leq \alpha.$
$r \in$ *mp-alloc-lsizestm-loopinv t p sz tm* β
apply(*rule exI*[**where** *x=α*])
apply(*rule conjI*) **apply** *fast*
using *mp-alloc-lsizestm-loopinv-stb*[*of t p tm sz α*] **apply**(*unfold stable-def*) **apply** *meson*
done

abbreviation *mp-alloc-precond1-6-1 t p sz tm α* ≡
  mp-alloc-lsizestm-loopinv t p sz tm α ∩ *mp-alloc-lsizestm-loopcond t p*

lemma *mp-alloc-precond1-6-1-ext-stb*: *stable* (*mp-alloc-lsizestm-loopcond t p*) (*Mem-pool-alloc-rely*

t)
apply(*simp add*:*stable-def*) **apply**(*rule allI*) **apply**(*rule impI*) **apply**(*rule allI*) **apply**(*rule impI*)
apply(*simp add*:*Mem-pool-alloc-rely-def*)
apply(*case-tac x=y*) **apply** *auto*[*1*] **apply** *clarify*
apply(*simp add*:*lvars-nochange-rel-def lvars-nochange-def gvars-conf-stable-def gvars-conf-def*)
done

lemma *mp-alloc-precond1-6-1-stb*: *stable* (*mp-alloc-precond1-6-1 t p sz tm* α) (*Mem-pool-alloc-rely t*)
apply(*rule stable-int2*)
using *mp-alloc-lsizestm-loopinv-stb* **apply** *auto*[*1*]
apply(*simp add*:*mp-alloc-precond1-6-1-ext-stb*)
done

abbreviation *mp-alloc-precond1-6-10 t p sz tm* $\alpha \equiv$
mp-alloc-precond1-6-1 t p sz tm $\alpha \cap \{\!|\,\acute{}i\ t > 0|\!\}$

lemma *mp-alloc-precond1-6-10-ext-stb*: *stable* ($\{\!|\,\acute{}i\ t > 0|\!\}$) (*Mem-pool-alloc-rely t*)
apply(*simp add*:*stable-def*) **apply**(*rule allI*) **apply**(*rule impI*) **apply**(*rule allI*) **apply**(*rule impI*)
apply(*simp add*:*Mem-pool-alloc-rely-def*)
apply(*case-tac x=y*) **apply** *auto*[*1*] **apply** *clarify*
apply(*simp add*:*lvars-nochange-rel-def lvars-nochange-def gvars-conf-stable-def gvars-conf-def*)
done

lemma *mp-alloc-precond1-6-10-stb*: *stable* (*mp-alloc-precond1-6-10 t p sz tm* α) (*Mem-pool-alloc-rely t*)
apply(*rule stable-int2*)
using *mp-alloc-precond1-6-1-stb* **apply** *auto*[*1*]
apply(*simp add*:*mp-alloc-precond1-6-10-ext-stb*)
done

abbreviation *mp-alloc-precond1-6-11 t p sz tm* $\alpha \equiv$
mp-alloc-precond1-6-1 t p sz tm $\alpha \cap - \{\!|\,\acute{}i\ t > 0|\!\}$

lemma *mp-alloc-precond1-6-11-ext-stb*: *stable* ($- \{\!|\,\acute{}i\ t > 0|\!\}$) (*Mem-pool-alloc-rely t*)
apply(*simp add*:*stable-def*) **apply**(*rule allI*) **apply**(*rule impI*) **apply**(*rule allI*) **apply**(*rule impI*)
apply(*simp add*:*Mem-pool-alloc-rely-def*)
apply(*case-tac x=y*) **apply** *auto*[*1*] **apply** *clarify*
apply(*simp add*:*lvars-nochange-rel-def lvars-nochange-def gvars-conf-stable-def gvars-conf-def*)
done

lemma *mp-alloc-precond1-6-11-stb*: *stable* (*mp-alloc-precond1-6-11 t p sz tm* α)

$(Mem\text{-}pool\text{-}alloc\text{-}rely\ t)$
apply(*rule stable-int2*)
using *mp-alloc-precond1-6-1-stb* **apply** *auto*[*1*]
apply(*simp add*:*mp-alloc-precond1-6-11-ext-stb*)
done

abbreviation *mp-alloc-precond1-6-2-ext t p sz tm* $\alpha \equiv$
$\{\!|(\forall\, ii{<}length\ ({}'lsizes\ t).\ {}'lsizes\ t\ !\ ii = (ALIGN4\ (max\text{-}sz\ ({}'mem\text{-}pool\text{-}info\ p)))\ div\ (4\ \hat{}\ ii))$
$\wedge\ length\ ({}'lsizes\ t) \leq n\text{-}levels\ ({}'mem\text{-}pool\text{-}info\ p)$
$\wedge\ ({}'i\ t = 0 \longrightarrow {}'alloc\text{-}l\ t = -1 \wedge {}'free\text{-}l\ t = -1 \wedge length\ ({}'lsizes\ t) = 1)$
$\wedge\ {}'i\ t \leq n\text{-}levels\ ({}'mem\text{-}pool\text{-}info\ p)$
$\wedge\ -1 \leq {}'free\text{-}l\ t \wedge {}'free\text{-}l\ t \leq\ int\ ({}'i\ t) - 1 \wedge {}'free\text{-}l\ t \leq {}'alloc\text{-}l\ t$
$\wedge\ {}'alloc\text{-}l\ t = int\ ({}'i\ t) - 1$
$\wedge\ ({}'alloc\text{-}l\ t \geq 0 \longrightarrow (\forall\, ii.\ ii \leq nat\ ({}'alloc\text{-}l\ t) \longrightarrow {}'lsizes\ t\ !\ ii \geq sz))$
$\wedge\ (\neg\ {}'alloc\text{-}lsize\text{-}r\ t \longrightarrow ({}'i\ t = 0 \longrightarrow length\ ({}'lsizes\ t) = 1) \wedge ({}'i\ t > 0 \longrightarrow length\ ({}'lsizes\ t) = {}'i\ t + 1))$
$(*$ *here* $'i\ t + 1$ *is different from mp-alloc-precond1-6-ext*, $*)$
$\wedge\ ({}'alloc\text{-}lsize\text{-}r\ t \longrightarrow length\ ({}'lsizes\ t) = {}'i\ t + 1 \wedge {}'i\ t < n\text{-}levels\ ({}'mem\text{-}pool\text{-}info\ p) \wedge {}'lsizes\ t\ !\ ({}'i\ t) < sz)|\!\}$
$\cap$ *mp-alloc-lsizeloop-*α*-cond t p* α

abbreviation *mp-alloc-precond1-6-2 t p sz tm* $\alpha \equiv$
mp-alloc-precond1-2 t p sz tm $\cap$ *mp-alloc-precond1-6-2-ext t p sz tm* α

lemma *mp-alloc-precond1-6-2-ext-stb*: *stable* (*mp-alloc-precond1-6-2-ext t p sz tm* α) (*Mem-pool-alloc-rely t*)
apply(*rule stable-int2*)
apply(*simp add*:*stable-def*) **apply** *clarify*
apply(*simp add*:*Mem-pool-alloc-rely-def*)
apply(*case-tac x=y*) **apply** *auto*[*1*] **apply** *clarify*
apply(*simp add*:*lvars-nochange-rel-def lvars-nochange-def gvars-conf-stable-def gvars-conf-def*)
apply(*rule conjI*) **apply** *clarify* **apply**(*simp add*:*lvars-nochange-rel-def lvars-nochange-def gvars-conf-stable-def gvars-conf-def*)
apply *clarify* **apply**(*simp add*:*lvars-nochange-rel-def lvars-nochange-def gvars-conf-stable-def gvars-conf-def*)
using *mp-alloc-lsizeloop-*α*-cond-stb* **apply** *fast*
done

lemma *mp-alloc-precond1-6-2-stb*: *stable* (*mp-alloc-precond1-6-2 t p sz tm* α) (*Mem-pool-alloc-rely t*)
apply(*rule stable-int2*)
using *mp-alloc-precond1-2-stb* **apply** *auto*[*1*]
using *mp-alloc-precond1-6-2-ext-stb* **apply** *auto*
done

abbreviation *mp-alloc-precond1-6-20 t p sz tm* $\alpha \equiv$
mp-alloc-precond1-6-2 t p sz tm $\alpha \cap \{\!|{}'lsizes\ t\ !\ {}'i\ t < sz|\!\}$

lemma *mp-alloc-precond1-6-20-ext-stb*: *stable* ($\{\!|$´*lsizes t* ! ´*i t* < *sz*$|\!\}$) (*Mem-pool-alloc-rely t*)
apply(*simp add*:*stable-def*) **apply** *clarify*
apply(*simp add*:*Mem-pool-alloc-rely-def*)
apply(*case-tac x=y*) **apply** *auto*[*1*] **apply** *clarify*
apply(*simp add*:*lvars-nochange-rel-def lvars-nochange-def gvars-conf-stable-def gvars-conf-def*)
done

lemma *mp-alloc-precond1-6-20-stb*: *stable* (*mp-alloc-precond1-6-20 t p sz tm* α) (*Mem-pool-alloc-rely t*)
apply(*rule stable-int2*)
using *mp-alloc-precond1-6-2-stb* **apply** *auto*[*1*]
apply(*simp add*:*mp-alloc-precond1-6-20-ext-stb*)
done

abbreviation *mp-alloc-precond1-6-21 t p sz tm* $\alpha \equiv$
mp-alloc-precond1-6-2 t p sz tm α $\cap$ $-$ $\{\!|$´*lsizes t* ! ´*i t* < *sz*$|\!\}$

lemma *mp-alloc-precond1-6-21-ext-stb*: *stable* ($-$ $\{\!|$´*lsizes t* ! ´*i t* < *sz*$|\!\}$) (*Mem-pool-alloc-rely t*)
apply(*simp add*:*stable-def*) **apply** *clarify*
apply(*simp add*:*Mem-pool-alloc-rely-def*)
apply(*case-tac x=y*) **apply** *auto*[*1*] **apply** *clarify*
apply(*simp add*:*lvars-nochange-rel-def lvars-nochange-def gvars-conf-stable-def gvars-conf-def*)
done

lemma *mp-alloc-precond1-6-21-stb*: *stable* (*mp-alloc-precond1-6-21 t p sz tm* α) (*Mem-pool-alloc-rely t*)
apply(*rule stable-int2*)
using *mp-alloc-precond1-6-2-stb* **apply** *auto*[*1*]
apply(*simp add*:*mp-alloc-precond1-6-21-ext-stb*)
done

abbreviation *mp-alloc-precond1-6-21-1-ext t p sz tm* $\alpha \equiv$
$\{\!|$($\forall$ *ii*<*length* (´*lsizes t*). ´*lsizes t* ! *ii* = (*ALIGN4* (*max-sz* (´*mem-pool-info p*))) *div* (*4* ˆ *ii*))
$\wedge$ *length* (´*lsizes t*) $\leq$ *n-levels* (´*mem-pool-info p*)
$\wedge$ (´*i t* = *0* $\longrightarrow$ ´*alloc-l t* = *0* $\wedge$ ´*free-l t* = $-$*1* $\wedge$ *length* (´*lsizes t*) = *1*)
$\wedge$ ´*i t* $\leq$ *n-levels* (´*mem-pool-info p*)
$\wedge$ $-$*1* $\leq$ ´*free-l t* $\wedge$ ´*free-l t* $\leq$ *int* (´*i t*) $-$ *1* $\wedge$ ´*free-l t* $\leq$ ´*alloc-l t*
$\wedge$ ´*alloc-l t* = *int* (´*i t*)
$\wedge$ (´*alloc-l t* $\geq$ *0* $\longrightarrow$ ($\forall$ *ii*. *ii* < *nat* (´*alloc-l t*) $\longrightarrow$ ´*lsizes t* ! *ii* $\geq$ *sz*))
$\wedge$ ($\neg$ ´*alloc-lsize-r t* $\longrightarrow$ (´*i t* = *0* $\longrightarrow$ *length* (´*lsizes t*) = *1*) $\wedge$ (´*i t* > *0* $\longrightarrow$ *length* (´*lsizes t*) = ´*i t* + *1*))
$\wedge$ (´*alloc-lsize-r t* $\longrightarrow$ *length* (´*lsizes t*) = ´*i t* + *1* $\wedge$ ´*i t* < *n-levels* (´*mem-pool-info p*) $\wedge$ ´*lsizes t* ! (´*i t*) < *sz*)

$\wedge \neg$ *´lsizes t* ! *´i t* $<$ *sz*$\}$ $\cap$ *mp-alloc-lsizeloop-*α*-cond t p* α

abbreviation *mp-alloc-precond1-6-21-1 t p sz tm* $\alpha \equiv$
mp-alloc-precond1-2 t p sz tm $\cap$ *mp-alloc-precond1-6-21-1-ext t p sz tm* α

lemma *mp-alloc-precond1-6-21-1-ext-stb*: *stable* (*mp-alloc-precond1-6-21-1-ext t p sz tm* α) (*Mem-pool-alloc-rely t*)
apply(*rule stable-int2*)
apply(*simp add*:*stable-def*) **apply** *clarify*
apply(*simp add*:*Mem-pool-alloc-rely-def*)
apply(*case-tac x=y*) **apply** *auto*[*1*] **apply** *clarify*
apply(*simp add*:*lvars-nochange-rel-def lvars-nochange-def gvars-conf-stable-def gvars-conf-def*)
apply *clarify* **apply**(*simp add*:*lvars-nochange-rel-def lvars-nochange-def gvars-conf-stable-def gvars-conf-def*)
using *mp-alloc-lsizeloop-*α*-cond-stb* **apply** *fast*
done

lemma *mp-alloc-precond1-6-21-1-stb*: *stable* (*mp-alloc-precond1-6-21-1 t p sz tm* α) (*Mem-pool-alloc-rely t*)
apply(*rule stable-int2*)
using *mp-alloc-precond1-2-stb* **apply** *auto*[*1*]
using *mp-alloc-precond1-6-21-1-ext-stb* **apply** *auto*
done

abbreviation *mp-alloc-precond1-6-21-2-ext t p sz tm* $\alpha \equiv$
$\{\!|$($\forall$ *ii*$<$*length* (*´lsizes t*). *´lsizes t* ! *ii* = (*ALIGN4* (*max-sz* (*´mem-pool-info p*))) *div* (*4* ^ *ii*))
$\wedge$ *length* (*´lsizes t*) $\leq$ *n-levels* (*´mem-pool-info p*)
$\wedge$ (*´i t* = *0* $\longrightarrow$ *´alloc-l t* = *0* $\wedge$ *length* (*´lsizes t*) = *1*)
$\wedge$ *´i t* $\leq$ *n-levels* (*´mem-pool-info p*)
$\wedge$ -1 $\leq$ *´free-l t* $\wedge$ *´free-l t* $\leq$ *int* (*´i t*) $\wedge$ *´free-l t* $\leq$ *´alloc-l t*
$\wedge$ *´alloc-l t* = *int* (*´i t*)
$\wedge$ (*´alloc-l t* $\geq$ *0* $\longrightarrow$ ($\forall$ *ii*. *ii* $<$ *nat* (*´alloc-l t*) $\longrightarrow$ *´lsizes t* ! *ii* $\geq$ *sz*))
$\wedge$ ($\neg$ *´alloc-lsize-r t* $\longrightarrow$ (*´i t* = *0* $\longrightarrow$ *length* (*´lsizes t*) = *1*) $\wedge$ (*´i t* $>$ *0* $\longrightarrow$ *length* (*´lsizes t*) = *´i t* + *1*))
$\wedge$ (*´alloc-lsize-r t* $\longrightarrow$ *length* (*´lsizes t*) = *´i t* + *1* $\wedge$ *´i t* $<$ *n-levels* (*´mem-pool-info p*) $\wedge$ *´lsizes t* ! (*´i t*) $<$ *sz*)
$\wedge \neg$ *´lsizes t* ! *´i t* $<$ *sz*$|\!\}$ $\cap$ *mp-alloc-lsizeloop-*α*-cond t p* α

abbreviation *mp-alloc-precond1-6-21-2 t p sz tm* $\alpha \equiv$
mp-alloc-precond1-2 t p sz tm $\cap$ *mp-alloc-precond1-6-21-2-ext t p sz tm* α

lemma *mp-alloc-precond1-6-21-2-ext-stb*: *stable* (*mp-alloc-precond1-6-21-2-ext t p sz tm* α) (*Mem-pool-alloc-rely t*)
apply(*rule stable-int2*)
apply(*simp add*:*stable-def*) **apply** *clarify*
apply(*simp add*:*Mem-pool-alloc-rely-def*)
apply(*case-tac x=y*) **apply** *auto*[*1*] **apply** *clarify*

apply(*simp add*:*lvars-nochange-rel-def lvars-nochange-def gvars-conf-stable-def gvars-conf-def*)
apply *clarify* **apply**(*simp add*:*lvars-nochange-rel-def lvars-nochange-def gvars-conf-stable-def gvars-conf-def*)
using *mp-alloc-lsizeloop-α-cond-stb* **apply** *fast*
done

lemma *mp-alloc-precond1-6-21-2-stb*: *stable* (*mp-alloc-precond1-6-21-2 t p sz tm* α) (*Mem-pool-alloc-rely t*)
apply(*rule stable-int2*)
using *mp-alloc-precond1-2-stb* **apply** *auto*[*1*]
using *mp-alloc-precond1-6-21-2-ext-stb* **apply** *auto*
done

abbreviation *mp-alloc-precond1-7 t p sz tm* ≡
mp-alloc-precond1-6 t p sz tm ∩ {|´*i t* ≥ *n-levels* (´*mem-pool-info p*) ∨ ´*alloc-lsize-r t* |}

lemma *mp-alloc-precond1-7-ext-stb*: *stable* ({|´*i t* ≥ *n-levels* (´*mem-pool-info p*) ∨ ´*alloc-lsize-r t* |}) (*Mem-pool-alloc-rely t*)
apply(*simp add*:*stable-def*) **apply** *clarify*
apply(*rule conjI*)
apply *clarify*
apply(*simp add*:*Mem-pool-alloc-rely-def*)
apply(*case-tac x*=*y*) **apply** *simp* **apply** *clarify*
apply(*simp add*:*lvars-nochange-rel-def lvars-nochange-def gvars-conf-stable-def gvars-conf-def*)

apply *clarify*
apply(*simp add*:*Mem-pool-alloc-rely-def*)
apply(*case-tac x*=*y*) **apply** *simp* **apply** *clarify*
apply(*simp add*:*lvars-nochange-rel-def lvars-nochange-def gvars-conf-stable-def gvars-conf-def*)
done

lemma *mp-alloc-precond1-7-stb*: *stable* (*mp-alloc-precond1-7 t p sz tm*) (*Mem-pool-alloc-rely t*)
apply(*rule stable-int2*)
using *mp-alloc-precond1-6-stb* **apply** *auto*[*1*]
apply(*simp add*:*mp-alloc-precond1-7-ext-stb*)
done

abbreviation *mp-alloc-precond1-70-ext t p sz tm* ≡
{|(∀ *ii*<*length* (´*lsizes t*). ´*lsizes t* ! *ii* = (*ALIGN4* (*max-sz* (´*mem-pool-info p*))) *div* (*4* ˆ *ii*))
∧ *length* (´*lsizes t*) ≤ *n-levels* (´*mem-pool-info p*)
∧ ´*alloc-l t* < *int* (*n-levels* (´*mem-pool-info p*))
∧ −*1* ≤ ´*free-l t* ∧ ´*free-l t* ≤ ´*alloc-l t*
∧ (´*alloc-l t* = −*1* ∧ ´*free-l t* = −*1* ∧ *length* (´*lsizes t*) = *1*

∨ (´*alloc-l t* ≥ *0* ∧ (∀ *ii*. *ii* ≤ *nat* (´*alloc-l t*) ⟶ ´*lsizes t* ! *ii* ≥ *sz*)
∧ ((´*alloc-l t* = *int* (*length* (´*lsizes t*)) − *1*) ∧ *length* (´*lsizes t*) = *n-levels* (´*mem-pool-info p*)
∨ ´*alloc-l t* = *int* (*length* (´*lsizes t*)) − *2* ∧ ´*lsizes t* ! *nat* (´*alloc-l t* + *1*) < *sz*)))⦄

abbreviation *mp-alloc-precond1-70 t p sz tm* ≡
mp-alloc-precond1-1 t p sz tm ∩ *mp-alloc-precond1-70-ext t p sz tm*

lemma *mp-alloc-precond1-70-ext-stb*: *stable* (*mp-alloc-precond1-70-ext t p sz tm*) (*Mem-pool-alloc-rely t*)
apply(*simp add*:*stable-def*) **apply** *clarify*
apply(*simp add*:*Mem-pool-alloc-rely-def*)
apply(*case-tac x*=*y*) **apply** *auto*[*1*] **apply** *clarify*
apply(*simp add*:*lvars-nochange-rel-def lvars-nochange-def gvars-conf-stable-def gvars-conf-def*)
done

lemma *mp-alloc-precond1-70-stb*: *stable* (*mp-alloc-precond1-70 t p sz tm*) (*Mem-pool-alloc-rely t*)
apply(*rule stable-int2*)
using *mp-alloc-precond1-1-stb* **apply** *auto*[*1*]
using *mp-alloc-precond1-70-ext-stb* **apply** *auto*
done

lemma *precnd17-bl-170*: *mp-alloc-precond1-7 t p sz tm* ⊆ *mp-alloc-precond1-70 t p sz tm*
apply *clarify* **apply**(*case-tac i x t* = *0*)
apply *clarify* **apply** *auto*[*1*]
apply *clarify*
apply(*rule IntI*) **apply** *auto*[*1*] **apply** *clarify*
apply(*rule conjI*) **apply** *simp*
apply(*rule conjI*) **apply** *simp*
apply(*rule conjI*) **apply** *simp*
apply *simp*
apply(*case-tac alloc-lsize-r x t*) **apply** *auto*
done

abbreviation *mp-alloc-precond1-70-1 t p sz tm* ≡
mp-alloc-precond1-70 t p sz tm ∩ ⦃´*alloc-l t* < *0*⦄

lemma *mp-alloc-precond1-70-1-ext-stb*: *stable* (⦃´*alloc-l t* < *0*⦄) (*Mem-pool-alloc-rely t*)
apply(*simp add*:*stable-def*) **apply** *clarify*
apply(*simp add*:*Mem-pool-alloc-rely-def*)
apply(*case-tac x*=*y*) **apply** *auto*[*1*] **apply** *clarify*
apply(*simp add*:*lvars-nochange-rel-def lvars-nochange-def gvars-conf-stable-def gvars-conf-def*)
done

lemma *mp-alloc-precond1-70-1-stb*: *stable* (*mp-alloc-precond1-70-1 t p sz tm*) (*Mem-pool-alloc-rely t*)
  apply(*rule stable-int2*)
  using *mp-alloc-precond1-70-stb* **apply** *auto*[*1*]
  apply(*simp add*:*mp-alloc-precond1-70-1-ext-stb*)
done

abbreviation *mp-alloc-precond1-70-2 t p sz tm* ≡
  mp-alloc-precond1-70 t p sz tm ∩ − {|´*alloc-l t* < *0*|}

lemma *mp-alloc-precond1-70-2-ext-stb*: *stable* (− {|´*alloc-l t* < *0*|}) (*Mem-pool-alloc-rely t*)
  apply(*simp add*:*stable-def*) **apply** *clarify*
  apply(*simp add*:*Mem-pool-alloc-rely-def*)
  apply(*case-tac x*=*y*) **apply** *auto*[*1*] **apply** *clarify*
  apply(*simp add*:*lvars-nochange-rel-def lvars-nochange-def gvars-conf-stable-def gvars-conf-def*)
done

lemma *mp-alloc-precond1-70-2-stb*: *stable* (*mp-alloc-precond1-70-2 t p sz tm*) (*Mem-pool-alloc-rely t*)
  apply(*rule stable-int2*)
  using *mp-alloc-precond1-70-stb* **apply** *auto*[*1*]
  apply(*simp add*:*mp-alloc-precond1-70-2-ext-stb*)
done

abbreviation *mp-alloc-precond1-70-2-1 t p sz tm* ≡
  mp-alloc-precond1-70-2 t p sz tm ∩ {|´*free-l t* < *0*|}

lemma *mp-alloc-precond1-70-2-1-ext-stb*: *stable* ({|´*free-l t* < *0*|}) (*Mem-pool-alloc-rely t*)
  apply(*simp add*:*stable-def*) **apply** *clarify*
  apply(*simp add*:*Mem-pool-alloc-rely-def*)
  apply(*case-tac x*=*y*) **apply** *auto*[*1*] **apply** *clarify*
  apply(*simp add*:*lvars-nochange-rel-def lvars-nochange-def gvars-conf-stable-def gvars-conf-def*)
done

lemma *mp-alloc-precond1-70-2-1-stb*: *stable* (*mp-alloc-precond1-70-2-1 t p sz tm*) (*Mem-pool-alloc-rely t*)
  apply(*rule stable-int2*)
  using *mp-alloc-precond1-70-2-stb* **apply** *auto*[*1*]
  apply(*simp add*:*mp-alloc-precond1-70-2-1-ext-stb*)
done

abbreviation *mp-alloc-precond1-70-2-2 t p sz tm* ≡
  mp-alloc-precond1-70-2 t p sz tm ∩ − {|´*free-l t* < *0*|}

lemma *mp-alloc-precond1-70-2-2-ext-stb*: *stable* (− {|´*free-l t* < *0*|}) (*Mem-pool-alloc-rely t*)
apply(*simp add*:*stable-def*) **apply** *clarify*
apply(*simp add*:*Mem-pool-alloc-rely-def*)
apply(*case-tac x*=*y*) **apply** *auto*[*1*] **apply** *clarify*
apply(*simp add*:*lvars-nochange-rel-def lvars-nochange-def gvars-conf-stable-def gvars-conf-def*)
done

lemma *mp-alloc-precond1-70-2-2-stb*: *stable* (*mp-alloc-precond1-70-2-2 t p sz tm*) (*Mem-pool-alloc-rely t*)
apply(*rule stable-int2*)
using *mp-alloc-precond1-70-2-stb* **apply** *auto*[*1*]
apply(*simp add*:*mp-alloc-precond1-70-2-2-ext-stb*)
done

lemma *alloc-memblk-data-valid-stb*:
blk x t = *buf* (*mem-pool-info x p*) +
block-num (*mem-pool-info x p*) (*blk x t*) (*lsizes x t* ! *nat* (*free-l x t*)) ∗
(*max-sz* (*mem-pool-info x p*) *div 4* ˆ *nat* (*free-l x t*)) =⇒
block-num (*mem-pool-info x p*) (*blk x t*) (*lsizes x t* ! *nat* (*free-l x t*)) < *n-max* (*mem-pool-info x p*) ∗ *4* ˆ *nat* (*free-l x t*) =⇒
allocating-node x t =
Some (|*pool* = *p*, *level* = *nat* (*free-l x t*), *block* = *block-num* (*mem-pool-info x p*) (*blk x t*) (*lsizes x t* ! *nat* (*free-l x t*)),
data = *blk x t*|) =⇒
(*x*, *y*) ∈ *lvars-nochange-rel t* =⇒
(*x*, *y*) ∈ *gvars-conf-stable* =⇒
alloc-memblk-data-valid y p (*the* (*allocating-node y t*))
apply(*subgoal-tac blk x t* = *blk y t*)
prefer *2* **apply**(*simp add*: *lvars-nochange-rel-def lvars-nochange-def*)
apply(*subgoal-tac buf* (*mem-pool-info x p*) = *buf* (*mem-pool-info y p*))
prefer *2* **apply**(*simp add*: *gvars-conf-stable-def gvars-conf-def*)
apply(*subgoal-tac lsizes x t* = *lsizes y t*)
prefer *2* **apply**(*simp add*: *lvars-nochange-rel-def lvars-nochange-def*)
apply(*subgoal-tac free-l x t* = *free-l y t*)
prefer *2* **apply**(*simp add*: *lvars-nochange-rel-def lvars-nochange-def*)
apply(*subgoal-tac max-sz* (*mem-pool-info x p*) = *max-sz* (*mem-pool-info y p*))
prefer *2* **apply**(*simp add*: *gvars-conf-stable-def gvars-conf-def*)
apply(*subgoal-tac allocating-node x t* = *allocating-node y t*)
prefer *2* **apply**(*simp add*: *lvars-nochange-rel-def lvars-nochange-def*)
apply (*simp add*: *gvars-conf-def gvars-conf-stable-def*)
done

abbreviation *mp-alloc-precond2-1-ext t p sz tm* ≡
{|(´*blk t* = *NULL* ∧ ´*allocating-node t* = *None*)
∨ (´*blk t* > *NULL* ∧ ´*alloc-memblk-data-valid p* (*the* (´*allocating-node t*))
∧ ´*allocating-node t* = *Some* (|*pool* = *p*, *level* = *nat* (´*free-l t*),
block = (*block-num* (´*mem-pool-info p*) (´*blk t*)

((´*lsizes t*)!(*nat* (´*free-l t*)))),
data = ´*blk t* ⦈
∧ (∃ *n*. *n* < *n-max* (´*mem-pool-info p*) ∗ (*4* ^ (*nat* (´*free-l t*)))
∧ ´*blk t* = *buf* (´*mem-pool-info p*) + *n* ∗ (*max-sz* (´*mem-pool-info p*)
div (*4* ^ (*nat* (´*free-l t*))))))⦄

abbreviation *mp-alloc-precond2-1 t p sz tm* ≡
{*s*. *inv s*} ∩ ⦃´*freeing-node t* = *None*⦄ ∩ ⦃*p* ∈ ´*mem-pools* ∧ *tm* ≥ −*1*⦄ ∩
mp-alloc-precond7-ext t p sz tm ∩ ⦃¬ ´*rf t*⦄
∩ *mp-alloc-precond1-70-ext t p sz tm* ∩ − ⦃´*alloc-l t* < *0*⦄
∩ − ⦃´*free-l t* < *0*⦄ ∩ *mp-alloc-precond2-1-ext t p sz tm*

term *mp-alloc-precond2-1 t p sz tm*

lemma *mp-alloc-freenode-stb*:
stable ⦃´*freeing-node t* = *None*⦄ (*Mem-pool-alloc-rely t*)
apply(*simp add*:*stable-def Mem-pool-alloc-rely-def lvars-nochange-rel-def lvars-nochange-def*)
done

lemma *mp-alloc-precond2-1-ext-stb*: *stable* (*mp-alloc-precond2-1-ext t p sz tm*) (*Mem-pool-alloc-rely t*)
apply(*simp add*:*stable-def*) **apply** *clarify*
apply(*rule conjI*) **apply** *clarify* **apply**(*simp add*:*Mem-pool-alloc-rely-def lvars-nochange-rel-def lvars-nochange-def*)
apply *smt*
apply(*rule impI*)+ **apply**(*rule allI*) **apply**(*rule impI*) **apply**(*rule disjI2*)
apply(*subgoal-tac buf* (*mem-pool-info x p*) = *buf* (*mem-pool-info y p*))
prefer *2* **apply**(*simp add*: *Mem-pool-alloc-rely-def gvars-conf-stable-def gvars-conf-def*)
apply *metis*
apply(*subgoal-tac free-l x t* = *free-l y t*)
prefer *2* **apply**(*simp add*: *Mem-pool-alloc-rely-def lvars-nochange-rel-def lvars-nochange-def*)
apply *smt*
apply(*subgoal-tac max-sz* (*mem-pool-info x p*) = *max-sz* (*mem-pool-info y p*))
prefer *2* **apply**(*simp add*: *Mem-pool-alloc-rely-def gvars-conf-stable-def gvars-conf-def*)
apply *smt*
apply(*subgoal-tac blk x t* = *blk y t*)
prefer *2* **apply**(*simp add*: *Mem-pool-alloc-rely-def lvars-nochange-rel-def lvars-nochange-def*)
apply *smt*
apply(*subgoal-tac allocating-node x t* = *allocating-node y t*)
prefer *2* **apply**(*simp add*: *Mem-pool-alloc-rely-def lvars-nochange-rel-def lvars-nochange-def*)
apply *smt*
apply(*subgoal-tac lsizes x t* = *lsizes y t*)
prefer *2* **apply**(*simp add*: *Mem-pool-alloc-rely-def lvars-nochange-rel-def lvars-nochange-def*)
apply *smt*
apply(*subgoal-tac n-max* (*mem-pool-info x p*) = *n-max* (*mem-pool-info y p*))
prefer *2* **apply**(*simp add*: *Mem-pool-alloc-rely-def gvars-conf-stable-def gvars-conf-def*)
apply *smt*
apply(*subgoal-tac block-num* (*mem-pool-info x p*) (*blk x t*) (*lsizes x t* ! *nat* (*free-l x t*))

$= block\text{-}num\ (mem\text{-}pool\text{-}info\ y\ p)\ (blk\ y\ t)\ (lsizes\ y\ t\ !\ nat\ (free\text{-}l\ y\ t)))$
prefer *2* **apply**(*simp add*: *block-num-def Mem-pool-alloc-rely-def lvars-nochange-rel-def lvars-nochange-def*)
apply *smt*
done

lemma *mp-alloc-precond2-1-stb*: *stable* (*mp-alloc-precond2-1 t p sz tm*) (*Mem-pool-alloc-rely t*)
apply(*rule stable-int2*) **apply**(*rule stable-int2*) **apply**(*rule stable-int2*) **apply**(*rule stable-int2*)
apply(*rule stable-int2*) **apply**(*rule stable-int2*) **apply**(*rule stable-int2*) **apply**(*rule stable-int2*)
apply (*simp add*: *stable-inv-alloc-rely1*)
apply (*simp add*: *mp-alloc-freenode-stb*)
apply (*simp add*: *mp-alloc-precond1-ext-stb*)
apply (*simp add*: *mp-alloc-precond7-ext-stb*)
apply (*simp add*: *mp-alloc-precond1-0-ext-stb*)
using *mp-alloc-precond1-70-ext-stb* **apply** *blast*
apply (*simp add*: *mp-alloc-precond1-70-2-ext-stb*)
apply (*simp add*: *mp-alloc-precond1-70-2-2-ext-stb*)
using *mp-alloc-precond2-1-ext-stb* **by** *blast*

abbreviation *mp-alloc-precond2-1-0 t p sz tm* ≡
mp-alloc-precond2-1 t p sz tm ∩ {|´*blk t* = *NULL*|}

lemma *mp-alloc-precond2-1-0-ext-stb*: *stable* ({|´*blk t* = *NULL*|}) (*Mem-pool-alloc-rely t*)
apply(*simp add*:*stable-def*) **apply** *clarify*
apply(*simp add*:*Mem-pool-alloc-rely-def*)
apply(*case-tac x*=*y*) **apply** *auto*[*1*] **apply** *clarify*
apply(*simp add*:*lvars-nochange-rel-def lvars-nochange-def gvars-conf-stable-def gvars-conf-def*)
done

lemma *mp-alloc-precond2-1-0-stb*: *stable* (*mp-alloc-precond2-1-0 t p sz tm*) (*Mem-pool-alloc-rely t*)
apply(*rule stable-int2*)
using *mp-alloc-precond2-1-stb* **apply** *auto*[*1*]
apply(*simp add*:*mp-alloc-precond2-1-0-ext-stb*)
done

abbreviation *mp-alloc-precond2-1-1 t p sz tm* ≡
mp-alloc-precond2-1 t p sz tm ∩ −{|´*blk t* = *NULL*|}

term *mp-alloc-precond2-1-1 t p sz tm*

lemma *mp-alloc-precond2-1-1-ext-stb*: *stable* (−{|´*blk t* = *NULL*|}) (*Mem-pool-alloc-rely t*)

apply(*simp add*:*stable-def*) **apply** *clarify*
apply(*simp add*:*Mem-pool-alloc-rely-def*)
apply(*case-tac x*=*y*) **apply** *auto*[*1*] **apply** *clarify*
apply(*simp add*:*lvars-nochange-rel-def lvars-nochange-def gvars-conf-stable-def gvars-conf-def*)
done

lemma *mp-alloc-precond2-1-1-stb*: *stable* (*mp-alloc-precond2-1-1 t p sz tm*) (*Mem-pool-alloc-rely t*)
apply(*rule stable-int2*)
using *mp-alloc-precond2-1-stb* **apply** *auto*[*1*]
apply(*simp add*:*mp-alloc-precond2-1-1-ext-stb*)
done

abbreviation *mp-alloc-precond2-1-1-loopinv-ext t p sz tm* $\equiv$
$-\{\!|\, ´blk\ t = NULL \,|\!\} \cap \{\!|\, ´from\text{-}l\ t \leq ´alloc\text{-}l\ t \wedge ´from\text{-}l\ t \geq ´free\text{-}l\ t \wedge ´allocating\text{-}node\ t = Some\ (\!|pool = p,\ level = nat\ (´from\text{-}l\ t),$
$block = block\text{-}num\ (´mem\text{-}pool\text{-}info\ p)\ (´blk\ t)\ ((´lsizes\ t)!(nat\ (´from\text{-}l\ t))),$
$data = ´blk\ t\ |\!)$
$\wedge\ ´alloc\text{-}memblk\text{-}data\text{-}valid\ p\ (the\ (´allocating\text{-}node\ t))$
$\wedge\ (\exists\, n.\ n < n\text{-}max\ (´mem\text{-}pool\text{-}info\ p) * (4\ \hat{}\ (nat\ (´from\text{-}l\ t)))$
$\wedge\ ´blk\ t = buf\ (´mem\text{-}pool\text{-}info\ p) + n * (max\text{-}sz\ (´mem\text{-}pool\text{-}info\ p)\ div\ (4\ \hat{}\ (nat\ (´from\text{-}l\ t))))) \,|\!\}$

abbreviation *mp-alloc-precond2-1-1-loopinv t p sz tm* $\equiv$
$\{s.\ inv\ s\} \cap \{\!|\, ´freeing\text{-}node\ t = None \,|\!\} \cap \{\!|\, p \in ´mem\text{-}pools \wedge tm \geq -1 \,|\!\} \cap mp\text{-}alloc\text{-}precond7\text{-}ext\ t\ p\ sz\ tm \cap \{\!|\, \neg\ ´rf\ t \,|\!\}$
$\cap\ mp\text{-}alloc\text{-}precond1\text{-}70\text{-}ext\ t\ p\ sz\ tm \cap -\ \{\!|\, ´alloc\text{-}l\ t < 0 \,|\!\}$
$\cap\ -\ \{\!|\, ´free\text{-}l\ t < 0 \,|\!\} \cap mp\text{-}alloc\text{-}precond2\text{-}1\text{-}1\text{-}loopinv\text{-}ext\ t\ p\ sz\ tm$

lemma *alloc-memblk-data-valid-stb2*:
$blk\ x\ t = buf\ (mem\text{-}pool\text{-}info\ x\ p) +$
$block\text{-}num\ (mem\text{-}pool\text{-}info\ x\ p)\ (blk\ x\ t)\ (lsizes\ x\ t\ !\ nat\ (from\text{-}l\ x\ t)) *$
$(max\text{-}sz\ (mem\text{-}pool\text{-}info\ x\ p)\ div\ 4\ \hat{}\ nat\ (from\text{-}l\ x\ t)) \Longrightarrow$
$block\text{-}num\ (mem\text{-}pool\text{-}info\ x\ p)\ (blk\ x\ t)\ (lsizes\ x\ t\ !\ nat\ (from\text{-}l\ x\ t)) < n\text{-}max\ (mem\text{-}pool\text{-}info\ x\ p) * 4\ \hat{}\ nat\ (from\text{-}l\ x\ t) \Longrightarrow$
$allocating\text{-}node\ x\ t =$
$Some\ (\!|pool = p,\ level = nat\ (from\text{-}l\ x\ t),\ block = block\text{-}num\ (mem\text{-}pool\text{-}info\ x\ p)\ (blk\ x\ t)\ (lsizes\ x\ t\ !\ nat\ (from\text{-}l\ x\ t)),$
$data = blk\ x\ t|\!) \Longrightarrow$
$(x,\ y) \in lvars\text{-}nochange\text{-}rel\ t \Longrightarrow$
$(x,\ y) \in gvars\text{-}conf\text{-}stable \Longrightarrow$
$alloc\text{-}memblk\text{-}data\text{-}valid\ y\ p\ (the\ (allocating\text{-}node\ y\ t))$
apply(*subgoal-tac blk x t* = *blk y t*)
prefer *2* **apply**(*simp add*: *lvars-nochange-rel-def lvars-nochange-def*)
apply(*subgoal-tac buf* (*mem-pool-info x p*) = *buf* (*mem-pool-info y p*))
prefer *2* **apply**(*simp add*: *gvars-conf-stable-def gvars-conf-def*)

apply(*subgoal-tac lsizes x t = lsizes y t*)
prefer *2* **apply**(*simp add*: *lvars-nochange-rel-def lvars-nochange-def*)
apply(*subgoal-tac from-l x t = from-l y t*)
prefer *2* **apply**(*simp add*: *lvars-nochange-rel-def lvars-nochange-def*)
apply(*subgoal-tac max-sz* (*mem-pool-info x p*) = *max-sz* (*mem-pool-info y p*))
prefer *2* **apply**(*simp add*: *gvars-conf-stable-def gvars-conf-def*)
apply(*subgoal-tac allocating-node x t = allocating-node y t*)
prefer *2* **apply**(*simp add*: *lvars-nochange-rel-def lvars-nochange-def*)
apply (*simp add*: *gvars-conf-def gvars-conf-stable-def*)
done

lemma *mp-alloc-precond2-1-1-loopinv-ext-stb*: *stable* (*mp-alloc-precond2-1-1-loopinv-ext t p sz tm*) (*Mem-pool-alloc-rely t*)
apply(*rule stable-int2*)
apply (*simp add*: *mp-alloc-precond2-1-1-ext-stb*)

apply(*simp add*:*stable-def*) **apply** *clarify*
apply(*subgoal-tac buf* (*mem-pool-info x p*) = *buf* (*mem-pool-info y p*))
prefer *2* **apply**(*simp add*: *Mem-pool-alloc-rely-def gvars-conf-stable-def gvars-conf-def*)
apply *metis*
apply(*subgoal-tac from-l x t = from-l y t*)
prefer *2* **apply**(*simp add*: *Mem-pool-alloc-rely-def lvars-nochange-rel-def lvars-nochange-def*)
apply *smt*
apply(*subgoal-tac max-sz* (*mem-pool-info x p*) = *max-sz* (*mem-pool-info y p*))
prefer *2* **apply**(*simp add*: *Mem-pool-alloc-rely-def gvars-conf-stable-def gvars-conf-def*)
apply *smt*
apply(*subgoal-tac blk x t = blk y t*)
prefer *2* **apply**(*simp add*: *Mem-pool-alloc-rely-def lvars-nochange-rel-def lvars-nochange-def*)
apply *smt*
apply(*subgoal-tac allocating-node x t = allocating-node y t*)
prefer *2* **apply**(*simp add*: *Mem-pool-alloc-rely-def lvars-nochange-rel-def lvars-nochange-def*)
apply *smt*
apply(*subgoal-tac lsizes x t = lsizes y t*)
prefer *2* **apply**(*simp add*: *Mem-pool-alloc-rely-def lvars-nochange-rel-def lvars-nochange-def*)
apply *smt*
apply(*subgoal-tac n-max* (*mem-pool-info x p*) = *n-max* (*mem-pool-info y p*))
prefer *2* **apply**(*simp add*: *Mem-pool-alloc-rely-def gvars-conf-stable-def gvars-conf-def*)
apply *smt*
apply(*subgoal-tac block-num* (*mem-pool-info x p*) (*blk x t*) (*lsizes x t* ! *nat* (*from-l x t*))
= *block-num* (*mem-pool-info y p*) (*blk y t*) (*lsizes y t* ! *nat* (*from-l y t*)))
prefer *2* **apply**(*simp add*: *block-num-def Mem-pool-alloc-rely-def lvars-nochange-rel-def lvars-nochange-def*)
apply(*case-tac x=y*) **apply** *auto*[*1*]
apply(*simp add*:*Mem-pool-alloc-rely-def gvars-conf-stable-def gvars-conf-def lvars-nochange-rel-def lvars-nochange-def*)
apply *smt*

done

lemma *mp-alloc-precond2-1-1-loopinv-stb*: *stable* (*mp-alloc-precond2-1-1-loopinv t p sz tm*) (*Mem-pool-alloc-rely t*)
apply(*rule stable-int2*) **apply**(*rule stable-int2*) **apply**(*rule stable-int2*) **apply**(*rule stable-int2*)
apply(*rule stable-int2*) **apply**(*rule stable-int2*) **apply**(*rule stable-int2*) **apply**(*rule stable-int2*)

apply (*simp add*: *stable-inv-alloc-rely1*)
apply (*simp add*: *mp-alloc-freenode-stb*)
apply (*simp add*: *mp-alloc-precond1-ext-stb*)
apply (*simp add*: *mp-alloc-precond7-ext-stb*)
apply (*simp add*: *mp-alloc-precond1-0-ext-stb*)
using *mp-alloc-precond1-70-ext-stb* **apply** *blast*
apply (*simp add*: *mp-alloc-precond1-70-2-ext-stb*)
apply (*simp add*: *mp-alloc-precond1-70-2-2-ext-stb*)
using *mp-alloc-precond2-1-1-loopinv-ext-stb* **apply** *auto*[*1*]
done

abbreviation *mp-alloc-precond2-1-2 t p sz tm* ≡
$\{s.\ inv\ s\} \cap \{\!|\acute{}freeing\text{-}node\ t = None|\!\} \cap \{\!|p \in \acute{}mem\text{-}pools \wedge tm \geq -1|\!\} \cap mp\text{-}alloc\text{-}precond7\text{-}ext\ t\ p\ sz\ tm \cap \{\!|\neg\ \acute{}rf\ t|\!\}$
$\cap\ mp\text{-}alloc\text{-}precond1\text{-}70\text{-}ext\ t\ p\ sz\ tm \cap -\ \{\!|\acute{}alloc\text{-}l\ t < 0|\!\} \cap -\ \{\!|\acute{}free\text{-}l\ t < 0|\!\} \cap -\{\!|\acute{}blk\ t = NULL|\!\}$
$\cap\ \{\!|\acute{}allocating\text{-}node\ t = Some\ (\!|pool = p,\ level = nat\ (\acute{}alloc\text{-}l\ t),$
$block = block\text{-}num\ (\acute{}mem\text{-}pool\text{-}info\ p)\ (\acute{}blk\ t)\ ((\acute{}lsizes\ t)!(nat\ (\acute{}alloc\text{-}l\ t))),$
$data = \acute{}blk\ t\ |\!) \wedge \acute{}alloc\text{-}memblk\text{-}data\text{-}valid\ p\ (the\ (\acute{}allocating\text{-}node\ t))\ |\!\}$

term *mp-alloc-precond2-1-1-loopinv t p sz tm*
term *mp-alloc-precond2-1-2 t p sz tm*

lemma *alloc-memblk-data-valid-stb3*:
blk x t = *buf* (*mem-pool-info x p*) +
block-num (*mem-pool-info x p*) (*blk x t*) (*lsizes x t* ! *nat* (*alloc-l x t*)) ∗
(*max-sz* (*mem-pool-info x p*) *div 4* ˆ *nat* (*alloc-l x t*)) ⟹
block-num (*mem-pool-info x p*) (*blk x t*) (*lsizes x t* ! *nat* (*alloc-l x t*)) < *n-max* (*mem-pool-info x p*) ∗ *4* ˆ *nat* (*alloc-l x t*) ⟹
allocating-node x t =
Some (|*pool* = *p*, *level* = *nat* (*alloc-l x t*), *block* = *block-num* (*mem-pool-info x p*) (*blk x t*) (*lsizes x t* ! *nat* (*alloc-l x t*)),
data = *blk x t*|) ⟹
(*x*, *y*) ∈ *lvars-nochange-rel t* ⟹
(*x*, *y*) ∈ *gvars-conf-stable* ⟹
alloc-memblk-data-valid y p (*the* (*allocating-node y t*))

apply(*subgoal-tac blk x t = blk y t*)
prefer *2* **apply**(*simp add*: *lvars-nochange-rel-def lvars-nochange-def*)
apply(*subgoal-tac buf (mem-pool-info x p) = buf (mem-pool-info y p)*)
prefer *2* **apply**(*simp add*: *gvars-conf-stable-def gvars-conf-def*)
apply(*subgoal-tac lsizes x t = lsizes y t*)
prefer *2* **apply**(*simp add*: *lvars-nochange-rel-def lvars-nochange-def*)
apply(*subgoal-tac alloc-l x t = alloc-l y t*)
prefer *2* **apply**(*simp add*: *lvars-nochange-rel-def lvars-nochange-def*)
apply(*subgoal-tac max-sz (mem-pool-info x p) = max-sz (mem-pool-info y p)*)
prefer *2* **apply**(*simp add*: *gvars-conf-stable-def gvars-conf-def*)
apply(*subgoal-tac allocating-node x t = allocating-node y t*)
prefer *2* **apply**(*simp add*: *lvars-nochange-rel-def lvars-nochange-def*)
apply (*simp add*: *gvars-conf-def gvars-conf-stable-def*)
done

lemma *mp-alloc-precond2-1-2-stb*: *stable* (*mp-alloc-precond2-1-2 t p sz tm*) (*Mem-pool-alloc-rely t*)
apply(*rule stable-int2*) **apply**(*rule stable-int2*) **apply**(*rule stable-int2*) **apply**(*rule stable-int2*)
apply(*rule stable-int2*) **apply**(*rule stable-int2*) **apply**(*rule stable-int2*) **apply**(*rule stable-int2*) **apply**(*rule stable-int2*)
apply (*simp add*: *stable-inv-alloc-rely1*)
apply (*simp add*: *mp-alloc-freenode-stb*)
apply (*simp add*: *mp-alloc-precond1-ext-stb*)
apply (*simp add*: *mp-alloc-precond7-ext-stb*)
apply (*simp add*: *mp-alloc-precond1-0-ext-stb*)
using *mp-alloc-precond1-70-ext-stb* **apply** *blast*
apply (*simp add*: *mp-alloc-precond1-70-2-ext-stb*)
apply (*simp add*: *mp-alloc-precond1-70-2-2-ext-stb*)
apply (*simp add*: *mp-alloc-precond2-1-1-ext-stb*)

apply(*simp add*:*stable-def*) **apply** *clarify*
apply(*simp add*:*Mem-pool-alloc-rely-def lvars-nochange-rel-def lvars-nochange-def*)

apply(*case-tac x=y*) **apply** *auto*[*1*] **apply** *clarify*
apply(*simp add*: *block-num-def lvars-nochange-rel-def lvars-nochange-def gvars-conf-stable-def gvars-conf-def*)
apply(*subgoal-tac blk x t = blk y t*)
prefer *2* **apply**(*simp add*: *lvars-nochange-rel-def lvars-nochange-def*)
apply(*subgoal-tac buf (mem-pool-info x p) = buf (mem-pool-info y p)*)
prefer *2* **apply**(*simp add*: *gvars-conf-stable-def gvars-conf-def*)
apply(*subgoal-tac lsizes x t = lsizes y t*)
prefer *2* **apply**(*simp add*: *lvars-nochange-rel-def lvars-nochange-def*)
apply(*subgoal-tac alloc-l x t = alloc-l y t*)
prefer *2* **apply**(*simp add*: *lvars-nochange-rel-def lvars-nochange-def*)
apply(*subgoal-tac max-sz (mem-pool-info x p) = max-sz (mem-pool-info y p)*)
prefer *2* **apply**(*simp add*: *gvars-conf-stable-def gvars-conf-def*)
apply(*subgoal-tac allocating-node x t = allocating-node y t*)
prefer *2* **apply**(*simp add*: *lvars-nochange-rel-def lvars-nochange-def*)

apply(*subgoal-tac n-max* (*mem-pool-info x p*) = *n-max* (*mem-pool-info y p*))
prefer *2* **apply**(*simp add*: *Mem-pool-alloc-rely-def gvars-conf-stable-def gvars-conf-def*)

apply(*subgoal-tac block-num* (*mem-pool-info x p*) (*blk x t*) (*lsizes x t* ! *nat* (*alloc-l x t*))
= *block-num* (*mem-pool-info y p*) (*blk y t*) (*lsizes y t* ! *nat* (*alloc-l y t*)))
prefer *2* **apply**(*simp add*: *block-num-def Mem-pool-alloc-rely-def lvars-nochange-rel-def lvars-nochange-def*)
by (*metis Mem-block.select-convs*(*2*) *Mem-block.select-convs*(*3*) *Mem-block.select-convs*(*4*) *option.sel*)

abbreviation *mp-alloc-precond2-1-3 t p sz tm* ≡
mp-alloc-precond1-70-2-2 t p sz tm ∩ −⦃´*blk t* = *NULL*⦄
∩ ⦃´*alloc-blk-valid p* (*nat* (´*alloc-l t*)) (*block-num* (´*mem-pool-info p*) (´*blk t*) ((´*lsizes t*)!(*nat* (´*alloc-l t*))))
(´*blk t*) ∧ ´*allocating-node t* = *None*⦄

lemma *mp-alloc-precond2-1-3-stb*: *stable* (*mp-alloc-precond2-1-3 t p sz tm*) (*Mem-pool-alloc-rely t*)
apply(*rule stable-int2*) **apply**(*rule stable-int2*)
using *mp-alloc-precond1-70-2-2-stb* **apply** *auto*[*1*]

apply(*simp add*:*stable-def*) **apply** *clarify*
apply(*simp add*:*Mem-pool-alloc-rely-def*)
apply(*case-tac x*=*y*) **apply** *auto*[*1*] **apply** *clarify*
apply(*simp add*:*lvars-nochange-rel-def lvars-nochange-def gvars-conf-stable-def gvars-conf-def*)

apply(*simp add*:*stable-def*) **apply** *clarify*
apply(*simp add*:*Mem-pool-alloc-rely-def*)
apply(*case-tac x*=*y*) **apply** *auto*[*1*] **apply** *clarify*
apply(*simp add*:*lvars-nochange-rel-def lvars-nochange-def gvars-conf-stable-def gvars-conf-def block-num-def*)
done

abbreviation *mp-alloc-precond2-1-4 t p sz tm* ≡
mp-alloc-precond1 t p tm ∩ ⦃¬´*rf t*⦄ ∩ ⦃ (*tm* = *FOREVER* ⟶ ´*tmout t* = *FOREVER*) ⦄
∩ {*s*. (∃ *mblk*. *mempoolalloc-ret s t* = *Some mblk* ∧ *alloc-memblk-valid s p sz mblk*)}

lemma *mp-alloc-precond2-1-4-stb*: *stable* (*mp-alloc-precond2-1-4 t p sz tm*) (*Mem-pool-alloc-rely t*)
apply(*rule stable-int2*) **apply**(*rule stable-int2*) **apply**(*rule stable-int2*)

using *mp-alloc-precond1-stb* **apply** *auto*[*1*]
apply(*simp add*:*stable-def Mem-pool-alloc-rely-def lvars-nochange-rel-def lvars-nochange-def*)
apply(*simp add*:*stable-def Mem-pool-alloc-rely-def lvars-nochange-rel-def lvars-nochange-def*)
apply *auto*[*1*]
apply(*simp add*:*stable-def*) **apply** *clarify*
apply(*simp add*:*Mem-pool-alloc-rely-def*)
apply(*case-tac x*=*y*) **apply** *simp* **apply** *clarify*
apply(*simp add*:*lvars-nochange-rel-def lvars-nochange-def gvars-conf-stable-def gvars-conf-def*)
apply(*case-tac x*=*y*) **apply** *simp* **apply** *clarify*
apply(*simp add*:*alloc-memblk-valid-def lvars-nochange-rel-def lvars-nochange-def gvars-conf-stable-def gvars-conf-def*)
apply *metis*
done

abbreviation *mp-alloc-precond1-8 t p sz tm* $\equiv$
mp-alloc-precond1 t p tm $\cap$ $\{\!|\neg\,\acute{}rf\ t|\!\}$ $\cap$ $\{\!|$ (*tm* = *FOREVER* $\longrightarrow$ $\acute{}$*tmout t* = *FOREVER*) $|\!\}$
$\cap$ {*s*. (*ret s t* = *OK* $\wedge$ ($\exists$ *mblk*. *mempoolalloc-ret s t* = *Some mblk* $\wedge$ *alloc-memblk-valid s p sz mblk*))
$\vee$ ((*ret s t* = *ESIZEERR* $\vee$ *ret s t* = *EAGAIN* $\vee$ *ret s t* = *ENOMEM*) $\wedge$ *mempoolalloc-ret s t* = *None*) }

lemma *mp-alloc-precond1-8-stb*: *stable* (*mp-alloc-precond1-8 t p sz tm*) (*Mem-pool-alloc-rely t*)
apply(*rule stable-int2*) **apply**(*rule stable-int2*) **apply**(*rule stable-int2*)
using *mp-alloc-precond1-stb* **apply** *auto*[*1*]
apply(*simp add*:*stable-def Mem-pool-alloc-rely-def lvars-nochange-rel-def lvars-nochange-def*)
apply (*smt mem-Collect-eq mp-alloc-precond2-ext-stb stable-def*)

apply(*simp add*:*stable-def*) **apply** *clarify*
apply(*rule conjI*)
apply *clarify*
apply(*rule conjI*)
apply(*simp add*:*Mem-pool-alloc-rely-def*)
apply(*case-tac x*=*y*) **apply** *simp* **apply** *clarify*
apply(*simp add*:*lvars-nochange-rel-def lvars-nochange-def gvars-conf-stable-def gvars-conf-def*)
apply(*simp add*:*Mem-pool-alloc-rely-def*)
apply(*case-tac x*=*y*) **apply** *simp* **apply** *clarify*
apply(*simp add*:*alloc-memblk-valid-def lvars-nochange-rel-def lvars-nochange-def gvars-conf-stable-def gvars-conf-def*)
apply *metis*
apply *clarify*
apply(*simp add*:*Mem-pool-alloc-rely-def*)
apply(*case-tac x*=*y*) **apply** *simp* **apply** *clarify*
apply(*simp add*:*lvars-nochange-rel-def lvars-nochange-def gvars-conf-stable-def gvars-conf-def*)
done

abbreviation *mp-alloc-precond1-8-0 t p sz tm* ≡
mp-alloc-precond1 t p tm ∩ ⦃ (*tm* = *FOREVER* ⟶ ´*tmout t* = *FOREVER*) ⦄
∩ {*s*. (*ret s t* = *OK* ∧ (∃ *mblk*. *mempoolalloc-ret s t* = *Some mblk* ∧ *alloc-memblk-valid s p sz mblk*))
∨ ((*ret s t* = *ESIZEERR* ∨ *ret s t* = *EAGAIN* ∨ *ret s t* = *ENOMEM*) ∧ *mempoolalloc-ret s t* = *None*) }

lemma *mp-alloc-precond1-8-0-stb*: *stable* (*mp-alloc-precond1-8-0 t p sz tm*) (*Mem-pool-alloc-rely t*)
apply(*rule stable-int2*) **apply**(*rule stable-int2*)
using *mp-alloc-precond1-stb* **apply** *auto*[*1*]
apply (*smt mem-Collect-eq mp-alloc-precond2-ext-stb stable-def*)

apply(*simp add*:*stable-def*) **apply** *clarify*
apply(*rule conjI*)
apply *clarify*
apply(*rule conjI*)
apply(*simp add*:*Mem-pool-alloc-rely-def*)
apply(*case-tac x*=*y*) **apply** *simp* **apply** *clarify*
apply(*simp add*:*lvars-nochange-rel-def lvars-nochange-def gvars-conf-stable-def gvars-conf-def*)
apply(*simp add*:*Mem-pool-alloc-rely-def*)
apply(*case-tac x*=*y*) **apply** *simp* **apply** *clarify*
apply(*simp add*:*alloc-memblk-valid-def lvars-nochange-rel-def lvars-nochange-def gvars-conf-stable-def gvars-conf-def*)
apply *metis*
apply *clarify*
apply(*simp add*:*Mem-pool-alloc-rely-def*)
apply(*case-tac x*=*y*) **apply** *simp* **apply** *clarify*
apply(*simp add*:*lvars-nochange-rel-def lvars-nochange-def gvars-conf-stable-def gvars-conf-def*)
done

abbreviation *mp-alloc-precond1-8-1 t p sz tm* ≡
mp-alloc-precond1-8 t p sz tm
∩ ⦃´*ret t* = *OK* ∨ *tm* = *NOWAIT* ∨ ´*ret t* = *ESIZEERR*⦄

lemma *mp-alloc-precond1-8-1-stb*: *stable* (*mp-alloc-precond1-8-1 t p sz tm*) (*Mem-pool-alloc-rely t*)
apply(*rule stable-int2*)
using *mp-alloc-precond1-8-stb* **apply** *auto*[*1*]
apply(*unfold stable-def*) **apply**(*simp add*:*Mem-pool-alloc-rely-def lvars-nochange-rel-def lvars-nochange-def*)
apply *auto*
done

abbreviation *mp-alloc-precond1-8-1-1 t p sz tm* ≡
mp-alloc-precond1-8-0 t p sz tm ∩ ⦃´*ret t* = *OK* ∨ *tm* = *NOWAIT* ∨ ´*ret t* =

ESIZEERR⦄ ∩ ⦃´*rf t* = *True*⦄

lemma *mp-alloc-precond1-8-1-1-stb*: *stable* (*mp-alloc-precond1-8-1-1 t p sz tm*) (*Mem-pool-alloc-rely t*)
  apply(*rule stable-int2*) **apply**(*rule stable-int2*)
  using *mp-alloc-precond1-8-0-stb* **apply** *auto*[*1*]
 apply(*unfold stable-def*) **apply**(*simp add*:*Mem-pool-alloc-rely-def lvars-nochange-rel-def lvars-nochange-def*)
    apply *auto*[*1*]
 apply(*simp add*:*Mem-pool-alloc-rely-def lvars-nochange-rel-def lvars-nochange-def*)

done

abbreviation *mp-alloc-precond1-8-1-2 t p sz tm* ≡
  mp-alloc-precond1-8-1-1 t p sz tm ∩ ⦃´*ret t* = *EAGAIN*⦄

lemma *mp-alloc-precond1-8-1-2-stb*: *stable* (*mp-alloc-precond1-8-1-2 t p sz tm*) (*Mem-pool-alloc-rely t*)
  apply(*rule stable-int2*)
  using *mp-alloc-precond1-8-1-1-stb* **apply** *auto*[*1*]
 apply(*unfold stable-def*) **apply**(*simp add*:*Mem-pool-alloc-rely-def lvars-nochange-rel-def lvars-nochange-def*)
done

abbreviation *mp-alloc-precond1-8-1-3 t p sz tm* ≡
  mp-alloc-precond1-8-1-1 t p sz tm ∩ −⦃´*ret t* = *EAGAIN*⦄

lemma *mp-alloc-precond1-8-1-3-stb*: *stable* (*mp-alloc-precond1-8-1-3 t p sz tm*) (*Mem-pool-alloc-rely t*)
  apply(*rule stable-int2*)
  using *mp-alloc-precond1-8-1-1-stb* **apply** *auto*[*1*]
 apply(*unfold stable-def*) **apply**(*simp add*:*Mem-pool-alloc-rely-def lvars-nochange-rel-def lvars-nochange-def*)
done

abbreviation *mp-alloc-precond1-8-2 t p sz tm* ≡
  mp-alloc-precond1-8 t p sz tm
    ∩ −⦃´*ret t* = *OK* ∨ *tm* = *NOWAIT* ∨ ´*ret t* = *ESIZEERR*⦄

lemma *mp-alloc-precond1-8-2-stb*: *stable* (*mp-alloc-precond1-8-2 t p sz tm*) (*Mem-pool-alloc-rely t*)
  apply(*rule stable-int2*)
  using *mp-alloc-precond1-8-stb* **apply** *auto*[*1*]
 apply(*unfold stable-def*) **apply**(*simp add*:*Mem-pool-alloc-rely-def lvars-nochange-rel-def lvars-nochange-def*)
done

abbreviation *mp-alloc-precond1-8-2-1 t p sz tm* ≡
  mp-alloc-precond1-8-2 t p sz tm ∩ ⦃´*ret t* = *EAGAIN*⦄

lemma *mp-alloc-precond1-8-2-1-stb*: *stable* (*mp-alloc-precond1-8-2-1 t p sz tm*) (*Mem-pool-alloc-rely t*)
  apply(*rule stable-int2*)
  using *mp-alloc-precond1-8-2-stb* **apply** *auto*[*1*]
 apply(*unfold stable-def*) **apply**(*simp add*:*Mem-pool-alloc-rely-def lvars-nochange-rel-def lvars-nochange-def*)
done

abbreviation *mp-alloc-precond1-8-2-2 t p sz tm* ≡
  mp-alloc-precond1-8-2 t p sz tm ∩ −⦃´*ret t* = *EAGAIN*⦄

lemma *mp-alloc-precond1-8-2-2-stb*: *stable* (*mp-alloc-precond1-8-2-2 t p sz tm*) (*Mem-pool-alloc-rely t*)
  apply(*rule stable-int2*)
  using *mp-alloc-precond1-8-2-stb* **apply** *auto*[*1*]
 apply(*unfold stable-def*) **apply**(*simp add*:*Mem-pool-alloc-rely-def lvars-nochange-rel-def lvars-nochange-def*)
done

abbreviation *mp-alloc-precond1-8-2-3 t p sz tm* ≡
  mp-alloc-precond1-8-2-2 t p sz tm ∩ ⦃´*tmout t* ≠ *FOREVER*⦄ ∩ ⦃*tm* > *0*⦄

lemma *mp-pred1823-eq*: *mp-alloc-precond1-8-2-3 t p sz tm* = *mp-alloc-precond1-8-2-2 t p sz tm* ∩ ⦃´*tmout t* ≠ *FOREVER*⦄
  by *auto*

lemma *mp-alloc-precond1-8-2-3-stb*: *stable* (*mp-alloc-precond1-8-2-3 t p sz tm*) (*Mem-pool-alloc-rely t*)
  apply(*rule stable-int2*) **apply**(*rule stable-int2*)
  using *mp-alloc-precond1-8-2-2-stb* **apply** *auto*[*1*]
 apply(*unfold stable-def*) **apply**(*simp add*:*Mem-pool-alloc-rely-def lvars-nochange-rel-def lvars-nochange-def*)
    apply *simp*
done

abbreviation *mp-alloc-precond1-8-2-20 t p sz tm* ≡
  mp-alloc-precond1-8-2-2 t p sz tm ∩ −⦃´*tmout t* ≠ *FOREVER*⦄

lemma *mp-alloc-precond1-8-2-20-stb*: *stable* (*mp-alloc-precond1-8-2-20 t p sz tm*) (*Mem-pool-alloc-rely t*)
  apply(*rule stable-int2*)
  using *mp-alloc-precond1-8-2-2-stb* **apply** *auto*[*1*]
 apply(*unfold stable-def*) **apply**(*simp add*:*Mem-pool-alloc-rely-def lvars-nochange-rel-def lvars-nochange-def*)
done

abbreviation *mp-alloc-precond1-8-2-4 t p sz tm* ≡ *mp-alloc-precond1-8-2-2 t p sz tm* ∩ ⦃*tm* > *0*⦄

lemma *mp-alloc-precond1-8-2-4-stb*: *stable* (*mp-alloc-precond1-8-2-4 t p sz tm*) (*Mem-pool-alloc-rely t*)
apply(*rule stable-int2*)
using *mp-alloc-precond1-8-2-2-stb* **apply** *auto*[*1*]
apply(*unfold stable-def*) **apply** *simp*
done

abbreviation *mp-alloc-precond1-8-2-40 t p sz tm* ≡
mp-alloc-precond1-8-2-4 t p sz tm ∩ {|´*tmout t* < *0*|}

lemma *mp-alloc-precond1-8-2-40-stb*: *stable* (*mp-alloc-precond1-8-2-40 t p sz tm*) (*Mem-pool-alloc-rely t*)
apply(*rule stable-int2*)
using *mp-alloc-precond1-8-2-4-stb* **apply** *blast*
apply(*unfold stable-def*) **apply**(*simp add*:*Mem-pool-alloc-rely-def lvars-nochange-rel-def lvars-nochange-def*)
done

term *mp-alloc-precond1-8-2-40 t p sz tm*

abbreviation *mp-alloc-precond1-8-2-41 t p sz tm* ≡
mp-alloc-precond1-8-2-4 t p sz tm ∩ −{|´*tmout t* < *0*|}

lemma *mp-alloc-precond1-8-2-41-stb*: *stable* (*mp-alloc-precond1-8-2-41 t p sz tm*) (*Mem-pool-alloc-rely t*)
apply(*rule stable-int2*)
using *mp-alloc-precond1-8-2-4-stb* **apply** *blast*
apply(*unfold stable-def*) **apply**(*simp add*:*Mem-pool-alloc-rely-def lvars-nochange-rel-def lvars-nochange-def*)
done

abbreviation *mp-alloc-precond1-8-2-5 t p sz tm* ≡
mp-alloc-precond1-8-0 t p sz tm ∩ {|*tm* > *0*|} ∩ −{|´*ret t* = *OK* ∨ *tm* = *NOWAIT* ∨ ´*ret t* = *ESIZEERR*|} ∩ −{|´*ret t* = *EAGAIN*|}
∩ {|´*tmout t* < *0*|} ∩ {|´*rf t*|}
term *mp-alloc-precond1-8-2-5 t p sz tm*

lemma *mp-alloc-precond1-8-2-5-stb*: *stable* (*mp-alloc-precond1-8-2-5 t p sz tm*) (*Mem-pool-alloc-rely t*)
apply(*rule stable-int2*) **apply**(*rule stable-int2*) **apply**(*rule stable-int2*) **apply**(*rule stable-int2*) **apply**(*rule stable-int2*)
using *mp-alloc-precond1-8-0-stb* **apply** *blast*
apply(*unfold stable-def*) **apply**(*simp add*:*Mem-pool-alloc-rely-def*)
apply(*simp add*:*Mem-pool-alloc-rely-def lvars-nochange-rel-def lvars-nochange-def*)
apply(*simp add*:*Mem-pool-alloc-rely-def lvars-nochange-rel-def lvars-nochange-def*)
apply(*simp add*:*Mem-pool-alloc-rely-def lvars-nochange-rel-def lvars-nochange-def*)
apply(*simp add*:*Mem-pool-alloc-rely-def lvars-nochange-rel-def lvars-nochange-def*)

done

10.2 proof of each statement

10.3 stm1

lemma *mp-alloc-stm1-lm0*:
cur V = Some t $\Longrightarrow$ *inv V* $\Longrightarrow$
V(|*lsizes* := (*lsizes V*)(*t* := *lsizes V t* @ [*ALIGN4* (*lsizes V t* ! (*i V t* − *Suc NULL*) *div 4*)])|)
$\in$ {|´(*Pair V*) $\in$ *Mem-pool-alloc-guar t*|}
apply *auto* **apply**(*simp add*:*Mem-pool-alloc-guar-def gvars-conf-stable-def gvars-conf-def lvars-nochange-def*)
apply(*rule disjI1*)
apply(*subgoal-tac* (*V*,*V*(|*lsizes* := (*lsizes V*)(*t* := *lsizes V t* @ [*ALIGN4* (*lsizes V t* ! (*i V t* − *Suc NULL*) *div 4*)])|))$\in$*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def*)
done

lemma *mp-alloc-stm1-lm1*: *mp-alloc-precond1-6-10 t p sz timeout* α $\cap$ {|´*cur* = *Some t*|} $\cap$ {*V*}
$\subseteq$ {|´(*lsizes-update* (λ-. ´*lsizes*(*t* := ´*lsizes t* @ [*ALIGN4* (´*lsizes t* ! (´*i t* − *Suc NULL*) *div 4*)])))
$\in$ {|´(*Pair V*) $\in$ *Mem-pool-alloc-guar t*|} $\cap$
mp-alloc-precond1-6-2 t p sz timeout α|}
apply *clarify*
apply(*rule IntI*) **using** *mp-alloc-stm1-lm0* **apply** *blast*
apply(*rule IntI*) **prefer** *2* **apply** *clarsimp*
apply(*subgoal-tac lsizes* (*V*(|*lsizes* := (*lsizes V*)(*t* := *lsizes V t* @ [*ALIGN4* (*lsizes V t* ! (*i V t* − *Suc NULL*) *div 4*)])|)) *t*
= *lsizes V t* @ [*ALIGN4* (*lsizes V t* ! (*i V t* − *Suc NULL*) *div 4*)])
prefer *2* **apply** *auto*[*1*]
apply(*simp add*: *subst*[**where** *s*=*lsizes* (*V*(|*lsizes* := (*lsizes V*)(*t* := *lsizes V t* @ [*ALIGN4* (*lsizes V t* ! (*i V t* − *Suc NULL*) *div 4*)])|)) *t*
and *t*=*lsizes V t* @ [*ALIGN4* (*lsizes V t* ! (*i V t* − *Suc NULL*) *div 4*)]]])

apply(*rule conjI*) **apply** *clarify*
apply(*case-tac ii* < *length* (*lsizes V t*)) **apply** (*metis nth-append*)
apply(*case-tac ii* = *length* (*lsizes V t*))
apply(*subgoal-tac* (*lsizes V t* @ [*ALIGN4* (*ALIGN4* (*max-sz* (*mem-pool-info V p*)) *div 4* ^ (*i V t* − *Suc NULL*) *div 4*)]) ! *ii*
= *ALIGN4* (*ALIGN4* (*max-sz* (*mem-pool-info V p*)) *div 4* ^ (*i V t* − *Suc NULL*) *div 4*))
prefer *2* **apply** (*meson nth-append-length*)
apply(*subgoal-tac ALIGN4* (*max-sz* (*mem-pool-info V p*)) *div 4* ^ (*i V t* − *Suc NULL*) *div 4*
= *ALIGN4* (*max-sz* (*mem-pool-info V p*)) *div 4* ^ *ii*)
prefer *2* **apply** (*metis Divides.div-mult2-eq One-nat-def power-minus-mult zero-le-numeral*)

apply(*subgoal-tac* (*ALIGN4* (*max-sz* (*mem-pool-info V p*)) *div 4* ^ *ii*) *mod 4* = *0*)
prefer *2* **apply**(*subgoal-tac* ∃ *n*>*0*. *max-sz* (*mem-pool-info V p*) = (*4* ∗ *n*) ∗ (*4* ^ *n-levels* (*mem-pool-info V p*)))
prefer *2* **apply**(*simp add*:*inv-def inv-mempool-info-def*) **apply** *metis*
apply (*metis* (*no-types*, *lifting*) *inv-maxsz-align4 less-imp-le-nat m-mod-div mod-mult-self1-is-0 mult.assoc pow-mod-0*)
apply (*metis align40*)
apply *linarith*
apply (*simp add*: *le-nat-iff nth-append*)

apply(*rule IntI*) **prefer** *2* **apply** *clarify* **apply** *auto*[*1*]
apply(*rule IntI*) **prefer** *2* **apply** *clarify* **apply** *auto*[*1*]
apply(*rule IntI*) **prefer** *2* **apply** *clarify* **apply** *auto*[*1*]
apply(*rule IntI*) **apply** *simp* **apply** *simp-inv* **apply** *metis*
apply *simp*
done

lemma *mp-alloc-stm1-lm*:
$\Gamma \vdash_I$ *Some* (*IF* ´*i t* > *0 THEN*
(*t* ▶ ´*lsizes* := ´*lsizes*(*t* := ´*lsizes t* @ [*ALIGN4* (´*lsizes t* ! (´*i t* − *1*) *div 4*)]))
FI) sat_p [*mp-alloc-precond1-6-1 t p sz timeout* α, *Mem-pool-alloc-rely t*, *Mem-pool-alloc-guar t*, *mp-alloc-precond1-6-2 t p sz timeout* α]
apply(*rule Cond*)
using *mp-alloc-precond1-6-1-stb* **apply** *simp*

apply(*unfold stm-def*)
apply(*rule Await*)
using *mp-alloc-precond1-6-10-stb*[*of t p timeout sz* α] **apply** *fast*
using *mp-alloc-precond1-6-2-stb* **apply** *simp*
apply(*rule allI*)
apply(*rule Basic*)
apply(*case-tac mp-alloc-precond1-6-10 t p sz timeout* α ∩ ⦃´*cur* = *Some t*⦄ ∩ {*V*} = {})
apply *auto*[*1*]
using *mp-alloc-stm1-lm1*[*of t p timeout sz* α] **apply** *auto*[*1*]
apply *simp* **using** *stable-id2* **apply** *auto*[*1*]
using *stable-id2* **apply** *auto*[*1*]

apply(*unfold Skip-def*)
apply(*rule Basic*) **apply** *clarify*
apply *simp*
apply(*simp add*:*Mem-pool-alloc-guar-def*) **apply** *auto*[*1*]
using *mp-alloc-precond1-6-11-stb*[*of t p timeout sz* α] **apply** *fast*
using *mp-alloc-precond1-6-2-stb* **apply** *fast*
apply(*simp add*:*Mem-pool-alloc-guar-def*)
done

10.4 stm2

lemma *mp-alloc-stm2-lm2-1*:
cur V = Some t ⟹ *inv V* ⟹ *V*⦇*alloc-lsize-r* := (*alloc-lsize-r V*)(*t* := *True*)⦈
∈ ⦃´(*Pair V*) ∈ *Mem-pool-alloc-guar t*⦄
apply *auto* **apply**(*simp add*:*Mem-pool-alloc-guar-def gvars-conf-stable-def gvars-conf-def lvars-nochange-def*)
apply(*rule disjI1*)
apply(*subgoal-tac* (*V*,*V*⦇*alloc-lsize-r* := (*alloc-lsize-r V*)(*t* := *True*)⦈)∈*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def*)
done

lemma *mp-alloc-stm2-lm2*:
mp-alloc-precond1-6-20 t p sz timeout α ∩ ⦃´*cur* = *Some t*⦄ ∩ {*V*}
⊆ ⦃´(*alloc-lsize-r-update* (λ-. ´*alloc-lsize-r*(*t* := *True*)))
∈ ⦃´(*Pair V*) ∈ *Mem-pool-alloc-guar t*⦄ ∩ *mp-alloc-lsizestm-loopinv t p sz timeout 0* ⦄
apply *clarify*
apply(*rule IntI*)
using *mp-alloc-stm2-lm2-1* **apply** *simp*
apply(*rule IntI*) **prefer** *2*
apply(*case-tac i V t = 0*) **apply**(*simp add*:*inv-def inv-mempool-info-def*) **apply** *simp*
apply(*rule IntI*) **prefer** *2* **apply** *auto*[*1*]
apply(*rule IntI*) **prefer** *2* **apply** *simp*
apply(*rule IntI*) **prefer** *2* **apply**(*simp add*:*alloc-memblk-valid-def*)
apply *simp* **apply**(*subgoal-tac* (*V*,*V*⦇*alloc-lsize-r* := (*alloc-lsize-r V*)(*t* := *True*)⦈)∈*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def*)
done

lemma *mp-alloc-stm2-lm4-1*:
cur V = Some t ⟹ *inv V* ⟹ *V*⦇*alloc-l* := (*alloc-l V*)(*t* := *int* (*i V t*))⦈ ∈ ⦃´(*Pair V*) ∈ *Mem-pool-alloc-guar t*⦄
apply *auto* **apply**(*simp add*:*Mem-pool-alloc-guar-def gvars-conf-stable-def gvars-conf-def lvars-nochange-def*)
apply(*rule disjI1*)
apply(*subgoal-tac* (*V*,*V*⦇*alloc-l* := (*alloc-l V*)(*t* := *int* (*i V t*))⦈)∈*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def*)
done

lemma *mp-alloc-stm2-lm4*:
mp-alloc-precond1-6-21 t p sz timeout α ∩ ⦃´*cur* = *Some t*⦄ ∩ {*V*}
⊆ ⦃´(*alloc-l-update* (λ-. ´*alloc-l*(*t* := *int* (´*i t*))))
∈ ⦃´(*Pair V*) ∈ *Mem-pool-alloc-guar t*⦄ ∩
mp-alloc-precond1-6-21-1 t p sz timeout α⦄
apply *clarify*
apply(*rule IntI*)

using *mp-alloc-stm2-lm4-1* **apply** *simp*
apply(*rule IntI*) **prefer** *2*
apply(*case-tac i V t = 0*) **apply** *simp* **apply** *simp*
apply(*rule IntI*) **prefer** *2* **apply** *simp*
apply(*rule IntI*) **prefer** *2* **apply** *simp*
apply(*rule IntI*) **prefer** *2* **apply**(*simp add:alloc-memblk-valid-def*)
apply *simp*
apply(*subgoal-tac* $(V,V(\!|alloc\text{-}l := (alloc\text{-}l\ V)(t := int\ (i\ V\ t))|\!)) \in lvars\text{-}nochange1\text{-}4all$)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add:lvars-nochange1-4all-def lvars-nochange1-def*)

done

lemma *mp-alloc-stm2-lm5-1-1*:
$cur\ V = Some\ t \Longrightarrow inv\ V \Longrightarrow\ V(\!|free\text{-}l := (free\text{-}l\ V)(t := int\ (i\ V\ t))|\!) \in \{\!|\acute{}(Pair\ V) \in Mem\text{-}pool\text{-}alloc\text{-}guar\ t|\!\}$
apply *auto* **apply**(*simp add:Mem-pool-alloc-guar-def gvars-conf-stable-def gvars-conf-def lvars-nochange-def*)
apply(*rule disjI1*)
apply(*subgoal-tac* $(V,V(\!|free\text{-}l := (free\text{-}l\ V)(t := int\ (i\ V\ t))|\!)) \in lvars\text{-}nochange1\text{-}4all$)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add:lvars-nochange1-4all-def lvars-nochange1-def*)
done

lemma *mp-alloc-stm2-lm5-1*:
$mp\text{-}alloc\text{-}precond1\text{-}6\text{-}21\text{-}1\ t\ p\ sz\ timeout\ \alpha \cap \{\!|\acute{}cur = Some\ t|\!\} \cap \{V\}$
$\subseteq \{\!|\acute{}(free\text{-}l\text{-}update\ (\lambda\text{-}.\ \acute{}free\text{-}l(t := int\ (\acute{}i\ t))))$
$\in \{\!|\acute{}(Pair\ V) \in Mem\text{-}pool\text{-}alloc\text{-}guar\ t|\!\} \cap$
$mp\text{-}alloc\text{-}precond1\text{-}6\text{-}21\text{-}2\ t\ p\ sz\ timeout\ \alpha|\!\}$
apply *clarify*
apply(*rule IntI*)
using *mp-alloc-stm2-lm5-1-1* **apply** *simp*
apply(*rule IntI*) **prefer** *2*
apply(*case-tac i V t = 0*) **apply** *simp* **apply** *simp*
apply(*rule IntI*) **prefer** *2* **apply** *simp*
apply(*rule IntI*) **prefer** *2* **apply** *simp*
apply(*rule IntI*) **prefer** *2* **apply**(*simp add:alloc-memblk-valid-def*)
apply *simp*
apply(*subgoal-tac* $(V,V(\!|free\text{-}l := (free\text{-}l\ V)(t := int\ (i\ V\ t))|\!)) \in lvars\text{-}nochange1\text{-}4all$)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add:lvars-nochange1-4all-def lvars-nochange1-def*)

done

lemma *mp-alloc-stm2-lm5*:
$\Gamma \vdash_I Some\ (t \blacktriangleright \acute{}free\text{-}l := \acute{}free\text{-}l\ (t := int\ (\acute{}i\ t)))$
$sat_p\ [mp\text{-}alloc\text{-}precond1\text{-}6\text{-}21\text{-}1\ t\ p\ sz\ timeout\ \alpha,\ Mem\text{-}pool\text{-}alloc\text{-}rely\ t,$
$Mem\text{-}pool\text{-}alloc\text{-}guar\ t,\ mp\text{-}alloc\text{-}precond1\text{-}6\text{-}21\text{-}2\ t\ p\ sz\ timeout\ \alpha]$
apply(*unfold stm-def*)

apply(*rule Await*)
using *mp-alloc-precond1-6-21-1-stb* **apply** *simp*
using *mp-alloc-precond1-6-21-2-stb* **apply** *simp*
apply(*rule allI*)
apply(*rule Basic*)
apply(*case-tac mp-alloc-precond1-6-21-1 t p sz timeout* $\alpha \cap$ {´*cur* = *Some t*} $\cap$ {*V*} = {})
apply *auto*[*1*] **using** *mp-alloc-stm2-lm5-1*[*of t p timeout sz* α] **apply** *auto*[*1*]
apply *simp* **using** *stable-id2* **apply** *auto*[*1*]
using *stable-id2* **apply** *auto*[*1*]
done

lemma *mp-alloc-stm2-lm6*:
$\Gamma \vdash_I$ *Some SKIP sat*$_p$ [*mp-alloc-precond1-6-21-1 t p sz timeout* α, *Mem-pool-alloc-rely t*,
Mem-pool-alloc-guar t, *mp-alloc-precond1-6-21-2 t p sz timeout* α]
apply(*unfold Skip-def*)
apply(*rule Basic*)
apply *clarify* **apply**(*rule IntI*) **apply**(*rule IntI*) **apply**(*rule IntI*) **apply**(*rule IntI*) **apply**(*rule IntI*)
apply *simp+* **apply**(*simp add*:*Mem-pool-alloc-guar-def gvars-conf-stable-def gvars-conf-def lvars-nochange-def*) **apply** *auto*[*1*]
using *mp-alloc-precond1-6-21-1-stb* **apply** *simp*
using *mp-alloc-precond1-6-21-2-stb* **apply** *simp*
done

lemma *mp-alloc-stm2-lm7-1*:
cur V = *Some t* $\Longrightarrow$ *inv V* $\Longrightarrow$ *V*(|*i* := (*i V*)(*t* := (*i V t*) + *1*)|) $\in$ {´(*Pair V*) $\in$ *Mem-pool-alloc-guar t*}
apply *auto* **apply**(*simp add*:*Mem-pool-alloc-guar-def gvars-conf-stable-def gvars-conf-def lvars-nochange-def*)
apply(*rule disjI1*)
apply(*subgoal-tac* (*V*,*V*(|*i* := (*i V*)(*t* := (*i V t*) + *1*)|))$\in$*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def*)
done

lemma *mp-alloc-stm2-lm7*:
mp-alloc-precond1-6-21-2 t p sz timeout $\alpha \cap$ {´*cur* = *Some t*} $\cap$ {*V*}
$\subseteq$ {´(*i-update* (λ-. ´*i*(*t* := *Suc* (´*i t*))))
$\in$ {´(*Pair V*) $\in$ *Mem-pool-alloc-guar t*} $\cap$
mp-alloc-lsizestm-loopinv t p sz timeout (α − *1*)}
apply *clarify*
apply(*rule IntI*)
using *mp-alloc-stm2-lm7-1* **apply** *simp*
apply(*rule IntI*) **prefer** *2*
apply(*case-tac i V t* = *0*) **apply** *simp*
apply(*simp add*:*inv-def inv-mempool-info-def*)
apply(*rule IntI*) **apply**(*rule IntI*) **apply**(*rule IntI*) **apply**(*rule IntI*)

apply *clarsimp* **apply**(*subgoal-tac* $(V, V(\!|i := (i\ V)(t := (i\ V\ t) + 1)|\!)) \in$ *lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def*)
apply *clarsimp*
apply *clarsimp*
apply *clarsimp*
apply *clarsimp*
apply(*rule conjI*) **apply** *auto*[*1*]
apply(*rule conjI*) **apply** *clarify* **apply**(*case-tac* $ii < i\ V\ t$) **apply** *auto*[*1*]
apply(*case-tac* $ii = i\ V\ t$) **apply** *simp* **apply** *simp*
apply *clarsimp*
done

lemma *subset-un-I1*[*intro*]: $A \subseteq B \Longrightarrow A \subseteq B \cup C$ **by** *auto*
lemma *subset-un-I2*[*intro*]: $A \subseteq C \Longrightarrow A \subseteq B \cup C$ **by** *auto*

lemma *mp-alloc-stm2-lm8*:
$P \subseteq \{\!|\acute{}(\textit{alloc-lsize-r-update}\ (\lambda\text{-}.\ \acute{}\textit{alloc-lsize-r}(t := \textit{True})))$
$\in \{\!|\acute{}(\textit{Pair}\ V) \in \textit{Mem-pool-alloc-guar}\ t|\!\} \cap B\ |\!\} \Longrightarrow$
$P \subseteq \{\!|\acute{}(\textit{alloc-lsize-r-update}\ (\lambda\text{-}.\ \acute{}\textit{alloc-lsize-r}(t := \textit{True})))$
$\in \{\!|\acute{}(\textit{Pair}\ V) \in \textit{Mem-pool-alloc-guar}\ t|\!\} \cap (A \cup B)\ |\!\}$
apply *auto*
done

lemma *mp-alloc-stm2-lm9*:
$P \subseteq \{\!|\acute{}(\textit{i-update}\ (\lambda\text{-}.\ \acute{}i(t := \textit{Suc}\ (\acute{}i\ t))))$
$\in \{\!|\acute{}(\textit{Pair}\ V) \in \textit{Mem-pool-alloc-guar}\ t|\!\} \cap A|\!\} \Longrightarrow$
$P \subseteq \{\!|\acute{}(\textit{i-update}\ (\lambda\text{-}.\ \acute{}i(t := \textit{Suc}\ (\acute{}i\ t))))$
$\in \{\!|\acute{}(\textit{Pair}\ V) \in \textit{Mem-pool-alloc-guar}\ t|\!\} \cap (A \cup B)|\!\}$
by *auto*

lemma *mp-alloc-stm2-lm*:
$\Gamma \vdash_I$ *Some* (*IF* $\acute{}$*lsizes* t ! $\acute{}i\ t < sz$ *THEN*
t ▶ $\acute{}$*alloc-lsize-r* := $\acute{}$*alloc-lsize-r*(t := *True*)
ELSE (t ▶ $\acute{}$*alloc-l* := $\acute{}$*alloc-l*(t := *int* ($\acute{}i\ t$)));;
IF $\neg$ *level-empty* ($\acute{}$*mem-pool-info* p) ($\acute{}i\ t$) *THEN*
t ▶ $\acute{}$*free-l* := $\acute{}$*free-l*(t := *int* ($\acute{}i\ t$))
FI;;
(t ▶ $\acute{}i$:= $\acute{}i$(t := *Suc* ($\acute{}i\ t$)))
FI) sat_p [*mp-alloc-precond1-6-2* t p *sz timeout* α, *Mem-pool-alloc-rely* t,
Mem-pool-alloc-guar t, *mp-alloc-lsizestm-loopinv* t p *sz timeout* ($\alpha - 1$)
$\cup$ *mp-alloc-lsizestm-loopinv* t p *sz timeout* *0*]
apply(*rule Cond*)
using *mp-alloc-precond1-6-2-stb* **apply** *simp*

apply(*unfold stm-def*)[*1*]
apply(*rule Await*)
using *mp-alloc-precond1-6-20-stb* **apply** *simp*

apply(*rule stable-un2*)
using *mp-alloc-lsizestm-loopinv-stb* **apply** *fast*
using *mp-alloc-lsizestm-loopinv-stb* **apply** *fast*
apply(*rule allI*)
apply(*rule Basic*)
apply(*case-tac mp-alloc-precond1-6-20 t p sz timeout* $\alpha \cap \{\!|$ *´cur* = *Some t*$|\!\}$ $\cap \{V\} = \{\}$)
apply *auto*[*1*]
apply(*rule mp-alloc-stm2-lm8*[*of - t - mp-alloc-lsizestm-loopinv t p sz timeout 0 mp-alloc-lsizestm-loopinv t p sz timeout* $(\alpha - 1)$])
using *mp-alloc-stm2-lm2*[*of t p timeout sz* α] **apply** *meson*
apply *simp* **using** *stable-id2* **apply** *auto*[*1*]
using *stable-id2* **apply** *auto*[*1*]

apply(*rule Seq*[**where** *mid=mp-alloc-precond1-6-21-2 t p sz timeout* α])
apply(*rule Seq*[**where** *mid=mp-alloc-precond1-6-21-1 t p sz timeout* α])

apply(*unfold stm-def*)[*1*]
apply(*rule Await*)
using *mp-alloc-precond1-6-21-stb* **apply** *simp*
using *mp-alloc-precond1-6-21-1-stb* **apply** *simp*
apply(*rule allI*)
apply(*rule Basic*)
apply(*case-tac mp-alloc-precond1-6-21 t p sz timeout* $\alpha \cap \{\!|$ *´cur* = *Some t*$|\!\}$ $\cap \{V\} = \{\}$)
apply *auto*[*1*] **using** *mp-alloc-stm2-lm4*[*of t p timeout sz*] **apply** *presburger*
apply *simp* **using** *stable-id2* **apply** *fast*
using *stable-id2* **apply** *fast*

apply(*rule Cond*)
using *mp-alloc-precond1-6-21-1-stb* **apply** *simp*

using *Conseq*[**where** *pre=mp-alloc-precond1-6-21-1 t p sz timeout* $\alpha \cap \{\!|\neg$ *level-empty* (*´mem-pool-info p*) (*´i t*)$|\!\}$
and *pre′=mp-alloc-precond1-6-21-1 t p sz timeout* α **and** *rely=Mem-pool-alloc-rely t*
and *rely′=Mem-pool-alloc-rely t* **and** *guar=Mem-pool-alloc-guar t* **and** *guar′=Mem-pool-alloc-guar t*
and *post=mp-alloc-precond1-6-21-2 t p sz timeout* α **and** *post′=mp-alloc-precond1-6-21-2 t p sz timeout* α
and *P=Some* (*t* ▶ *´free-l* := *´free-l* (*t* := *int* (*´i t*)))]
mp-alloc-stm2-lm5[*of t p timeout sz*] **apply** *fast*
using *Conseq*[**where** *pre=mp-alloc-precond1-6-21-1 t p sz timeout* $\alpha \cap - \{\!|\neg$ *level-empty* (*´mem-pool-info p*) (*´i t*)$|\!\}$
and *pre′=mp-alloc-precond1-6-21-1 t p sz timeout* α **and** *rely=Mem-pool-alloc-rely t*
and *rely′=Mem-pool-alloc-rely t* **and** *guar=Mem-pool-alloc-guar t* **and** *guar′=Mem-pool-alloc-guar*

t
  and *post=mp-alloc-precond1-6-21-2 t p sz timeout* α **and** *post'=mp-alloc-precond1-6-21-2 t p sz timeout* α
   and *P=Some SKIP*]
   mp-alloc-stm2-lm6[*of t p timeout sz*] **apply** *fast*
  apply(*simp add*:*Mem-pool-alloc-guar-def*)

 apply(*unfold stm-def*)[*1*]
 apply(*rule Await*)
  using *mp-alloc-precond1-6-21-2-stb* **apply** *simp*
  apply(*rule stable-un2*)
   using *mp-alloc-lsizestm-loopinv-stb* **apply** *fast*
   using *mp-alloc-lsizestm-loopinv-stb* **apply** *fast*
  apply(*rule allI*)
  apply(*rule Basic*)
   apply(*case-tac mp-alloc-precond1-6-21-2 t p sz timeout* α $\cap$ {|´*cur* = *Some t*|} $\cap$ {*V*} = {})
    apply *auto*[*1*]
   apply(*rule mp-alloc-stm2-lm9*[*of - t - mp-alloc-lsizestm-loopinv t p sz timeout* (α − *1*) *mp-alloc-lsizestm-loopinv t p sz timeout 0*])
    using *mp-alloc-stm2-lm7*[*of t p timeout sz* α] **apply** *meson*
   apply *simp* **using** *stable-id2* **apply** *fast*
   using *stable-id2* **apply** *fast*

 apply(*simp add*:*Mem-pool-alloc-guar-def*)
done

term {|´(*Pair Va*) $\in$ *Mem-pool-alloc-guar t*|} $\cap$ *mp-alloc-precond2-1 t p sz timeout*

10.5 lsize while loop

abbreviation *alloc-lsize-loopbody t p sz* $\equiv$
 IF ´*i t* > *0 THEN*
  (*t* ▶ ´*lsizes* := ´*lsizes*(*t* := ´*lsizes t* @ [*ALIGN4* (´*lsizes t* ! (´*i t* − *1*) *div 4*)]))
 FI;;
 IF ´*lsizes t* ! ´*i t* < *sz THEN*
  (*t* ▶ ´*alloc-lsize-r* := ´*alloc-lsize-r*(*t* := *True*))
 ELSE
  (*t* ▶ ´*alloc-l* := ´*alloc-l*(*t* := *int* (´*i t*)));;
  IF ¬ *level-empty* (´*mem-pool-info p*) (´*i t*) *THEN*
   (*t* ▶ ´*free-l* := ´*free-l*(*t* := *int* (´*i t*)))
  FI;;
  (*t* ▶ ´*i* := ´*i*(*t* := ´*i t* + *1*))
 FI

lemma *lsize-loop-body-terminate*:

$\Gamma \vdash_I$ *Some* (*alloc-lsize-loopbody t p sz*)
  sat$_p$ [*mp-alloc-lsizestm-loopinv t p sz tm* $\alpha \cap \{\!\!\{\alpha > 0\}\!\!\}$, *Mem-pool-alloc-rely t*, *Mem-pool-alloc-guar t*,
    mp-alloc-lsizestm-loopinv t p sz tm $(\alpha - 1) \cup$ *mp-alloc-lsizestm-loopinv t p sz tm 0*]

apply(*rule Seq*[**where** *mid=mp-alloc-precond1-6-2 t p sz tm* α])

apply(*rule subst*[**where** *s=mp-alloc-precond1-6-1 t p sz tm* α **and**
  t=mp-alloc-lsizestm-loopinv t p sz tm $\alpha \cap \{\!\!\{\alpha > 0\}\!\!\}$])
  using *lsizeloop-inv-cond-eq-*α*gt0*[*of t p tm sz* α] **apply** *fast*
using *mp-alloc-stm1-lm*[*of t p tm sz* α] **apply** *blast*

using *mp-alloc-stm2-lm* **apply** *simp*

done

lemma *lsizeloopbody-sat-invterm-imp-inv-post*:
$\Gamma \vdash_I$ *Some P sat*$_p$ [*pre*, *rely*, *guar*,
  mp-alloc-lsizestm-loopinv t p sz tm $(\alpha - 1) \cup$ *mp-alloc-lsizestm-loopinv t p sz tm 0*]
$\Longrightarrow$ $\Gamma \vdash_I$ *Some P sat*$_p$ [*pre*, *rely*, *guar*,*mp-alloc-precond1-6 t p sz tm*]
apply(*rule Conseq* [*of pre pre*
  rely rely guar guar mp-alloc-lsizestm-loopinv t p sz tm $(\alpha - 1) \cup$ *mp-alloc-lsizestm-loopinv t p sz tm 0*
  mp-alloc-precond1-6 t p sz tm Some P])
apply *fast+*
done

lemma *lsizeloopbody-term-imp-prepost*:
($\forall \alpha$. $\Gamma \vdash_I$ *Some P sat*$_p$ [*mp-alloc-lsizestm-loopinv t p sz tm* $\alpha \cap \{\!\!\{\alpha > 0\}\!\!\}$, *rely*, *guar*,
  mp-alloc-lsizestm-loopinv t p sz tm $(\alpha - 1) \cup$ *mp-alloc-lsizestm-loopinv t p sz tm 0*])
$\Longrightarrow$ $\Gamma \vdash_I$ *Some P sat*$_p$ [*mp-alloc-precond1-6 t p sz tm* $\cap$ *mp-alloc-lsizestm-loopcond t p*,
  rely, *guar*, *mp-alloc-precond1-6 t p sz tm*]

apply(*rule subst*[**where** *s*=$\forall$ *v*. *v*$\in$*mp-alloc-precond1-6 t p sz tm* $\cap$ *mp-alloc-lsizestm-loopcond t p* $\longrightarrow$
  $\Gamma \vdash_I$ *Some P sat*$_p$ [{*v*}, *rely*, *guar*, *mp-alloc-precond1-6 t p sz tm*] **and**
  t=$\Gamma \vdash_I$ *Some P sat*$_p$ [*mp-alloc-precond1-6 t p sz tm* $\cap$ *mp-alloc-lsizestm-loopcond t p*,
  rely, *guar*, *mp-alloc-precond1-6 t p sz tm*]])
using *allpre-eq-pre*[*of mp-alloc-precond1-6 t p sz tm* $\cap$ *mp-alloc-lsizestm-loopcond t p*
  Some P rely guar mp-alloc-precond1-6 t p sz tm]

apply *meson*

apply(*rule allI*) **apply**(*rule impI*)
apply(*subgoal-tac* $\exists\alpha$. *v* $\in$ *mp-alloc-lsizestm-loopinv t p sz tm* α $\cap$ $\{\!|\alpha > 0|\!\}$)
prefer *2* **using** *lsizestm-pre-loopcond-imp-loopinv-*α*gt0*[*of - t p tm sz*] **apply** *meson*

apply(*erule exE*)
using *sat-pre-imp-allinpre*[*of Some P - rely guar mp-alloc-precond1-6 t p sz tm*]
lsizeloopbody-sat-invterm-imp-inv-post[*of P - rely guar t p tm sz*] **apply** *meson*
done

lemma *lsize-loop-body-satprepost*:
$\Gamma \vdash_I$ *Some* (*alloc-lsize-loopbody t p sz*)
sat$_p$ [*mp-alloc-precond1-6 t p sz timeout* $\cap$ *mp-alloc-lsizestm-loopcond t p*,
Mem-pool-alloc-rely t, *Mem-pool-alloc-guar t*, *mp-alloc-precond1-6 t p sz timeout*]
using *lsizeloopbody-term-imp-prepost*[*of alloc-lsize-loopbody t p sz t p timeout sz*
Mem-pool-alloc-rely t Mem-pool-alloc-guar t]
lsize-loop-body-terminate[*of t sz p timeout*] **apply** *fast*
done

lemma *lsize-loop-stm*:
$\Gamma \vdash_I$ *Some* (*WHILE ´i t* < *n-levels* (*´mem-pool-info p*) $\wedge$ $\neg$ *´alloc-lsize-r t DO*
alloc-lsize-loopbody t p sz
OD) *sat*$_p$ [*mp-alloc-precond1-6 t p sz timeout*, *Mem-pool-alloc-rely t*,
Mem-pool-alloc-guar t, *mp-alloc-precond1-7 t p sz timeout*]
apply(*rule While*)
using *mp-alloc-precond1-6-stb* **apply** *simp*
apply(*rule Int-greatest*) **apply**(*rule Int-greatest*) **apply**(*rule Int-greatest*) **apply**(*rule Int-greatest*)
apply(*rule Int-greatest*)
apply *force+*
using *mp-alloc-precond1-7-stb* **apply** *simp*

using *lsize-loop-body-satprepost*[*of t sz p timeout*] **apply** *fast*

apply(*simp add*:*Mem-pool-alloc-guar-def*)
done

10.6 stm3

lemma *mp-alloc-stm3-lm3-1*: *ii* < *n-levels mp* $\Longrightarrow$
length (*levels mp*) = *n-levels mp* $\Longrightarrow$
free-list (*levels mp* ! *ii*) = [] $\Longrightarrow$
rmhead-free-list mp ii = *mp*
apply(*simp add*:*rmhead-free-list-def*)
by (*metis Mem-pool.surjective Mem-pool.update-convs*(*5*) *Mem-pool-lvl.surjective*

Mem-pool-lvl.update-convs(*2*) *list-update-id*)

lemma *mp-alloc-stm3-lm3-2*:
head-free-list mp ii = *NULL* $\Longrightarrow$
ii < *n-levels mp* $\Longrightarrow$
NULL < *buf mp* $\Longrightarrow$
$\forall$ *i*<*n-levels mp*.
$\forall$ *j*<*length* (*free-list* (*levels mp* ! *i*)).
buf mp $\leq$ *free-list* (*levels mp* ! *i*) ! *j* $\Longrightarrow$
length (*levels mp*) = *n-levels mp* $\Longrightarrow$
free-list (*levels mp* ! *ii*) $\neq$ [] $\Longrightarrow$
False
apply(*simp add*:*head-free-list-def*)
apply(*subgoal-tac hd* (*free-list* (*levels mp* ! *ii*)) $\neq$ *NULL*)
apply *simp*
using *hd-conv-nth* **by** *force*

lemma *mp-alloc-stm3-lm3*:
Va $\in$ *mp-alloc-precond1-70-2-2 t p sz timeout* $\cap$ {|´*cur* = *Some t*|} $\Longrightarrow$
(*if level-empty* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)) *then*
{*V*. *V* = *Va*(|*blk*:=(*blk Va*)(*t*:=*NULL*)|) $\wedge$ *level-empty* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))}
else
{*V*. *V* = *Va*(|*blk*:=(*blk Va*)(*t* := *head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))),
mem-pool-info := (*mem-pool-info Va*)(*p*:=*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))|)
$\wedge$ $\neg$ *level-empty* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))}) $\cap$
$-$ {|´*blk t* $\neq$ *NULL*|}
$\subseteq$ {|´*id* $\in$ {|´(*Pair Va*) $\in$ *Mem-pool-alloc-guar t*|} $\cap$ *mp-alloc-precond2-1 t p sz timeout*|}
apply *clarsimp* **apply** *meson*

apply(*unfold Mem-pool-alloc-guar-def*)[*1*] **apply**(*rule UnI1*) **apply** *simp*
apply(*rule conjI*) **apply**(*simp add*:*gvars-conf-stable-def gvars-conf-def*)
apply(*rule conjI*)
apply(*subgoal-tac* (*Va*,*Va*(|*blk* := (*blk Va*)(*t* := *NULL*)|))$\in$*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def*)
apply(*simp add*:*lvars-nochange-def*)

apply(*subgoal-tac* (*Va*,*Va*(|*blk* := (*blk Va*)(*t* := *NULL*)|))$\in$*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def*)

apply *clarsimp*
apply(*subgoal-tac nat* (*free-l Va t*) $\geq$ *0* $\wedge$ *nat* (*free-l Va t*) < *n-levels* (*mem-pool-info*

Va p))
prefer *2* **apply** *linarith*
apply(*subgoal-tac buf* (*mem-pool-info Va p*) > *0*)
prefer *2* **apply**(*simp add*:*inv-def inv-mempool-info-def inv-bitmap-freelist-def*)
apply *meson*
apply(*subgoal-tac* ∀ *i*<*length* (*levels* (*mem-pool-info Va p*)).
∀ *j*<*length* (*free-list* (*levels* (*mem-pool-info Va p*) ! *i*)).
buf (*mem-pool-info Va p*) ≤ (*free-list* (*levels* (*mem-pool-info Va p*) ! *i*))
! *j*)
prefer *2* **apply**(*simp add*:*inv-def inv-mempool-info-def inv-bitmap-freelist-def*
Let-def)
apply *clarify* **apply** (*metis Suc-leI lessI not-le trans-le-add1*)
apply(*subgoal-tac length* (*levels* (*mem-pool-info Va p*)) = *n-levels* (*mem-pool-info*
Va p))
prefer *2* **apply**(*simp add*:*inv-def inv-mempool-info-def inv-bitmap-freelist-def*)
apply *metis*
apply(*subgoal-tac* ∀ *j*<*length* (*free-list* (*levels* (*mem-pool-info Va p*) ! *nat* (*free-l*
Va t))).
buf (*mem-pool-info Va p*) ≤ *free-list* (*levels* (*mem-pool-info Va p*) ! *nat*
(*free-l Va t*)) ! *j*)
prefer *2* **apply**(*simp add*:*inv-def inv-mempool-info-def inv-bitmap-freelist-def*)
apply(*rule conjI*)
apply(*unfold Mem-pool-alloc-guar-def*)[*1*] **apply**(*rule UnI1*) **apply** *clarsimp*
apply(*rule conjI*)
apply(*simp add*:*gvars-conf-stable-def gvars-conf-def rmhead-free-list-def*) **apply** *clarsimp*
apply(*case-tac nat* (*free-l Va t*) ≠ *i*) **apply** *simp* **apply** *simp*
apply(*rule conjI*)
apply(*case-tac free-list* ((*levels* (*mem-pool-info Va p*)) ! (*nat* (*free-l Va t*)))
= [])
apply(*subgoal-tac rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))
= *mem-pool-info Va p*)
apply *simp* **apply**(*subgoal-tac* (*Va*, *Va*(|*blk* := (*blk Va*)(*t* := *NULL*)|))∈*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def*
lvars-nochange1-def)
using *mp-alloc-stm3-lm3-1*[*of nat* (*free-l Va t*) *mem-pool-info Va p*] **apply**
meson
using *mp-alloc-stm3-lm3-2*[*of mem-pool-info Va p nat* (*free-l Va t*)] **apply**
meson
apply(*simp add*:*lvars-nochange-def*)
apply(*rule conjI*)
apply(*case-tac free-list* ((*levels* (*mem-pool-info Va p*)) ! (*nat* (*free-l Va t*))) =
[])
apply(*subgoal-tac rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)) =
mem-pool-info Va p)
apply *simp* **apply**(*subgoal-tac* (*Va*, *Va*(|*blk* := (*blk Va*)(*t* := *NULL*)|))∈*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def*
lvars-nochange1-def)
using *mp-alloc-stm3-lm3-1*[*of nat* (*free-l Va t*) *mem-pool-info Va p*] **apply**

meson
using *mp-alloc-stm3-lm3-2[of mem-pool-info Va p nat (free-l Va t)]* **apply** *metis*
apply(*rule conjI*) **apply**(*simp add:rmhead-free-list-def*)
apply(*rule conjI*) **apply**(*simp add:rmhead-free-list-def*)
apply(*rule conjI*) **apply**(*simp add:rmhead-free-list-def*) **apply**(*simp add:rmhead-free-list-def*)

apply(*unfold Mem-pool-alloc-guar-def*)[*1*] **apply**(*rule UnI1*) **apply** *simp*
apply(*rule conjI*) **apply**(*simp add:gvars-conf-stable-def gvars-conf-def*)
apply(*rule conjI*)
apply(*subgoal-tac* $(Va, Va(|blk := (blk\ Va)(t := NULL)|)) \in$ *lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add:lvars-nochange1-4all-def lvars-nochange1-def*)
apply(*simp add:lvars-nochange-def*)

apply(*subgoal-tac* $(Va, Va(|blk := (blk\ Va)(t := NULL)|)) \in$ *lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add:lvars-nochange1-4all-def lvars-nochange1-def*)

apply *clarsimp*
apply(*subgoal-tac nat* (*free-l Va t*) $\geq$ *0* $\wedge$ *nat* (*free-l Va t*) $<$ *n-levels* (*mem-pool-info Va p*))
prefer *2* **apply** *linarith*
apply(*subgoal-tac buf* (*mem-pool-info Va p*) $>$ *0*)
prefer *2* **apply**(*simp add:inv-def inv-mempool-info-def inv-bitmap-freelist-def*) **apply** *meson*
apply(*subgoal-tac* $\forall$ *i*$<$*length* (*levels* (*mem-pool-info Va p*)).
$\forall$ *j*$<$*length* (*free-list* (*levels* (*mem-pool-info Va p*) ! *i*)).
buf (*mem-pool-info Va p*) $\leq$ (*free-list* (*levels* (*mem-pool-info Va p*) ! *i*)) ! *j*)
prefer *2* **apply**(*simp add:inv-def inv-mempool-info-def inv-bitmap-freelist-def Let-def*)
apply *clarify* **apply** (*metis Suc-leI lessI not-less trans-le-add1*)
apply(*subgoal-tac length* (*levels* (*mem-pool-info Va p*)) = *n-levels* (*mem-pool-info Va p*))
prefer *2* **apply**(*simp add:inv-def inv-mempool-info-def inv-bitmap-freelist-def*) **apply** *metis*
apply(*subgoal-tac* $\forall$ *j*$<$*length* (*free-list* (*levels* (*mem-pool-info Va p*) ! *nat* (*free-l Va t*))).
buf (*mem-pool-info Va p*) $\leq$ *free-list* (*levels* (*mem-pool-info Va p*) ! *nat* (*free-l Va t*)) ! *j*)
prefer *2* **apply**(*simp add:inv-def inv-mempool-info-def inv-bitmap-freelist-def*)
apply(*rule conjI*)
apply(*unfold Mem-pool-alloc-guar-def*)[*1*] **apply**(*rule UnI1*) **apply** *clarsimp*
apply(*rule conjI*)
apply(*simp add:gvars-conf-stable-def gvars-conf-def rmhead-free-list-def*) **ap-**

ply *clarsimp*
apply(*case-tac nat* (*free-l Va t*) $\neq$ *i*) **apply** *simp* **apply** *simp*
apply(*rule conjI*)
apply(*case-tac free-list* ((*levels* (*mem-pool-info Va p*)) ! (*nat* (*free-l Va t*))) = [])
apply(*subgoal-tac rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)) = *mem-pool-info Va p*)
apply *simp* **apply**(*subgoal-tac* (*Va*, *Va*⦇*blk* := (*blk Va*)(*t* := *NULL*)⦈)∈*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def*)
using *mp-alloc-stm3-lm3-1*[*of nat* (*free-l Va t*) *mem-pool-info Va p*] **apply** *meson*
using *mp-alloc-stm3-lm3-2*[*of mem-pool-info Va p nat* (*free-l Va t*)] **apply** *meson*
apply(*simp add*:*lvars-nochange-def*)
apply(*rule conjI*)
apply(*case-tac free-list* ((*levels* (*mem-pool-info Va p*)) ! (*nat* (*free-l Va t*))) = [])
apply(*subgoal-tac rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)) = *mem-pool-info Va p*)
apply *simp* **apply**(*subgoal-tac* (*Va*, *Va*⦇*blk* := (*blk Va*)(*t* := *NULL*)⦈)∈*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def*)
using *mp-alloc-stm3-lm3-1*[*of nat* (*free-l Va t*) *mem-pool-info Va p*] **apply** *meson*
using *mp-alloc-stm3-lm3-2*[*of mem-pool-info Va p nat* (*free-l Va t*)] **apply** *metis*
apply(*rule conjI*) **apply**(*simp add*:*rmhead-free-list-def*)
apply(*rule conjI*) **apply**(*simp add*:*rmhead-free-list-def*)
apply(*rule conjI*) **apply**(*simp add*:*rmhead-free-list-def*) **apply**(*simp add*:*rmhead-free-list-def*)

apply(*unfold Mem-pool-alloc-guar-def*)[*1*] **apply**(*rule UnI1*) **apply** *simp*
apply(*rule conjI*) **apply**(*simp add*:*gvars-conf-stable-def gvars-conf-def*)
apply(*rule conjI*)
apply(*subgoal-tac* (*Va*, *Va*⦇*blk* := (*blk Va*)(*t* := *NULL*)⦈)∈*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def*)
apply(*simp add*:*lvars-nochange-def*)

apply(*subgoal-tac* (*Va*, *Va*⦇*blk* := (*blk Va*)(*t* := *NULL*)⦈)∈*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def*)

apply *clarsimp*
apply(*subgoal-tac nat* (*free-l Va t*) $\geq$ *0* $\wedge$ *nat* (*free-l Va t*) < *n-levels* (*mem-pool-info Va p*))

prefer *2* **apply** *linarith*
apply(*subgoal-tac buf* (*mem-pool-info Va p*) > *0*)
prefer *2* **apply**(*simp add*:*inv-def inv-mempool-info-def inv-bitmap-freelist-def*)
apply *meson*
apply(*subgoal-tac* ∀ *i*<*length* (*levels* (*mem-pool-info Va p*)).
∀ *j*<*length* (*free-list* (*levels* (*mem-pool-info Va p*) ! *i*)).
buf (*mem-pool-info Va p*) ≤ (*free-list* (*levels* (*mem-pool-info Va p*) ! *i*))
! *j*)
prefer *2* **apply**(*simp add*:*inv-def inv-mempool-info-def inv-bitmap-freelist-def*
Let-def)
apply *clarify* **apply** (*metis Suc-leI lessI not-less trans-le-add1*)
apply(*subgoal-tac length* (*levels* (*mem-pool-info Va p*)) = *n-levels* (*mem-pool-info*
Va p))
prefer *2* **apply**(*simp add*:*inv-def inv-mempool-info-def inv-bitmap-freelist-def*)
apply *metis*
apply(*subgoal-tac* ∀ *j*<*length* (*free-list* (*levels* (*mem-pool-info Va p*) ! *nat* (*free-l*
Va t))).
buf (*mem-pool-info Va p*) ≤ *free-list* (*levels* (*mem-pool-info Va p*) ! *nat*
(*free-l Va t*)) ! *j*)
prefer *2* **apply**(*simp add*:*inv-def inv-mempool-info-def inv-bitmap-freelist-def*)
apply(*rule conjI*)
apply(*unfold Mem-pool-alloc-guar-def*)[*1*] **apply**(*rule UnI1*) **apply** *clarsimp*
apply(*rule conjI*)
apply(*simp add*:*gvars-conf-stable-def gvars-conf-def rmhead-free-list-def*) **ap-
ply** *clarsimp*
apply(*case-tac nat* (*free-l Va t*) ≠ *i*) **apply** *simp* **apply** *simp*
apply(*rule conjI*)
apply(*case-tac free-list* ((*levels* (*mem-pool-info Va p*)) ! (*nat* (*free-l Va t*)))
= [])
apply(*subgoal-tac rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))
= *mem-pool-info Va p*)
apply *simp* **apply**(*subgoal-tac* (*Va*, *Va*(|*blk* := (*blk Va*)(*t* := *NULL*)|))∈*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def*
lvars-nochange1-def)
using *mp-alloc-stm3-lm3-1*[*of nat* (*free-l Va t*) *mem-pool-info Va p*] **apply**
meson
using *mp-alloc-stm3-lm3-2*[*of mem-pool-info Va p nat* (*free-l Va t*)] **apply**
meson
apply(*simp add*:*lvars-nochange-def*)
apply(*rule conjI*)
apply(*case-tac free-list* ((*levels* (*mem-pool-info Va p*)) ! (*nat* (*free-l Va t*))) =
[])
apply(*subgoal-tac rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)) =
mem-pool-info Va p)
apply *simp* **apply**(*subgoal-tac* (*Va*, *Va*(|*blk* := (*blk Va*)(*t* := *NULL*)|))∈*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def*
lvars-nochange1-def)
using *mp-alloc-stm3-lm3-1*[*of nat* (*free-l Va t*) *mem-pool-info Va p*] **apply**
meson

using *mp-alloc-stm3-lm3-2*[*of mem-pool-info Va p nat* (*free-l Va t*)] **apply** *metis*
apply(*rule conjI*) **apply**(*simp add*:*rmhead-free-list-def*)
apply(*rule conjI*) **apply**(*simp add*:*rmhead-free-list-def*)
apply(*simp add*:*rmhead-free-list-def*)

apply(*unfold Mem-pool-alloc-guar-def*)[*1*] **apply**(*rule UnI1*) **apply** *simp*
apply(*rule conjI*) **apply**(*simp add*:*gvars-conf-stable-def gvars-conf-def*)
apply(*rule conjI*)
apply(*subgoal-tac* $(Va, Va(|blk := (blk\ Va)(t := NULL)|)) \in lvars\text{-}nochange1\text{-}4all$)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def*)
apply(*simp add*:*lvars-nochange-def*)

apply(*subgoal-tac* $(Va, Va(|blk := (blk\ Va)(t := NULL)|)) \in lvars\text{-}nochange1\text{-}4all$)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def*)

apply *clarsimp*
apply(*subgoal-tac nat* (*free-l Va t*) $\geq$ *0* $\wedge$ *nat* (*free-l Va t*) $<$ *n-levels* (*mem-pool-info Va p*))
prefer *2* **apply** *linarith*
apply(*subgoal-tac buf* (*mem-pool-info Va p*) $>$ *0*)
prefer *2* **apply**(*simp add*:*inv-def inv-mempool-info-def inv-bitmap-freelist-def*) **apply** *meson*
apply(*subgoal-tac* $\forall$ *i*$<$*length* (*levels* (*mem-pool-info Va p*)).
$\forall$ *j*$<$*length* (*free-list* (*levels* (*mem-pool-info Va p*) ! *i*)).
buf (*mem-pool-info Va p*) $\leq$ (*free-list* (*levels* (*mem-pool-info Va p*) ! *i*)) ! *j*)
prefer *2* **apply**(*simp add*:*inv-def inv-mempool-info-def inv-bitmap-freelist-def Let-def*)
apply *clarify* **apply** (*metis Suc-leI lessI not-less trans-le-add1*)
apply(*subgoal-tac length* (*levels* (*mem-pool-info Va p*)) = *n-levels* (*mem-pool-info Va p*))
prefer *2* **apply**(*simp add*:*inv-def inv-mempool-info-def inv-bitmap-freelist-def*) **apply** *metis*
apply(*subgoal-tac* $\forall$ *j*$<$*length* (*free-list* (*levels* (*mem-pool-info Va p*) ! *nat* (*free-l Va t*))).
buf (*mem-pool-info Va p*) $\leq$ *free-list* (*levels* (*mem-pool-info Va p*) ! *nat* (*free-l Va t*)) ! *j*)
prefer *2* **apply**(*simp add*:*inv-def inv-mempool-info-def inv-bitmap-freelist-def*)
apply(*rule conjI*)
apply(*unfold Mem-pool-alloc-guar-def*)[*1*] **apply**(*rule UnI1*) **apply** *clarsimp*
apply(*rule conjI*)
apply(*simp add*:*gvars-conf-stable-def gvars-conf-def rmhead-free-list-def*) **ap-**

ply *clarsimp*
apply(*case-tac nat* (*free-l Va t*) $\neq$ *i*) **apply** *simp* **apply** *simp*
apply(*rule conjI*)
apply(*case-tac free-list* ((*levels* (*mem-pool-info Va p*)) ! (*nat* (*free-l Va t*)))
= [])
apply(*subgoal-tac rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))
= *mem-pool-info Va p*)
apply *simp* **apply**(*subgoal-tac* (*Va*, *Va*(|*blk* := (*blk Va*)(*t* := *NULL*)|))∈*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def*
lvars-nochange1-def)
using *mp-alloc-stm3-lm3-1*[*of nat* (*free-l Va t*) *mem-pool-info Va p*] **apply**
meson
using *mp-alloc-stm3-lm3-2*[*of mem-pool-info Va p nat* (*free-l Va t*)] **apply**
meson
apply(*simp add*:*lvars-nochange-def*)
apply(*rule conjI*)
apply(*case-tac free-list* ((*levels* (*mem-pool-info Va p*)) ! (*nat* (*free-l Va t*))) =
[])
apply(*subgoal-tac rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)) =
mem-pool-info Va p)
apply *simp* **apply**(*subgoal-tac* (*Va*, *Va*(|*blk* := (*blk Va*)(*t* := *NULL*)|))∈*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def*
lvars-nochange1-def)
using *mp-alloc-stm3-lm3-1*[*of nat* (*free-l Va t*) *mem-pool-info Va p*] **apply**
meson
using *mp-alloc-stm3-lm3-2*[*of mem-pool-info Va p nat* (*free-l Va t*)] **apply**
metis
apply(*rule conjI*) **apply**(*simp add*:*rmhead-free-list-def*)
apply(*rule conjI*) **apply**(*simp add*:*rmhead-free-list-def*) **apply**(*simp add*:*rmhead-free-list-def*)

done

lemma *mp-alloc-stm3-lm2-1*:
length (*bits* (*levels mp* ! *ii*)) =
length (*bits* (*levels mp* [*ii* := (*levels mp* [*ii* := (*levels mp* ! *ii*) (|*free-list* := *fl*|)]
! *ii*)
(|*bits* := *bits* (*levels mp* [*ii* := (*levels mp* ! *ii*) (|*free-list* := *fl*|)] ! *ii*)
[*jj* := *ALLOCATING*]|)] ! *ii*))
apply(*case-tac ii* < *length* (*levels mp*))
apply *simp*
apply *auto*
done

lemma *mp-alloc-stm3-lm2-2*:
length (*bits* (*levels mp* ! *ii*)) =
length (*bits* (*levels mp* [*ii* := (*levels mp* ! *ii*) (|*free-list* := *fl*, *bits* := *bits* (*levels*

mp ! *ii*) [*jj* := *ALLOCATING*]⦈] ! *ii*))
  apply(*case-tac ii* < *length* (*levels mp*))
    apply *simp* **apply** *auto*
done

lemma *mp-alloc-stm3-body-meminfo*:
  pa ≠ *p* ⟹
    set-bit-allocating ((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va* *p*) (*nat* (*free-l Va t*)))) *p*
      (*nat* (*free-l Va t*))
      (*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
        (*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))) (*lsizes Va t* ! *nat* (*free-l Va t*)))
      pa = (*mem-pool-info Va*) *pa*
  by(*simp add*: *set-bit-def*)

lemma *mp-alloc-stm3-body-minf-buf*:
  buf (*set-bit-allocating* ((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info* *Va p*) (*nat* (*free-l Va t*)))) *p*
      (*nat* (*free-l Va t*))
      (*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
        (*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))) (*lsizes Va t* ! *nat* (*free-l Va t*)))
      p) = *buf* (*mem-pool-info Va p*)
  by (*simp add*: *set-bit-def rmhead-free-list-def*)

lemma *mp-alloc-stm3-body-minf-maxsz*:
  max-sz (*set-bit-allocating* ((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info* *Va p*) (*nat* (*free-l Va t*)))) *p*
      (*nat* (*free-l Va t*))
      (*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
        (*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))) (*lsizes Va t* ! *nat* (*free-l Va t*)))
      p) = *max-sz* (*mem-pool-info Va p*)
  by (*simp add*: *set-bit-def rmhead-free-list-def*)

lemma *mp-alloc-stm3-body-minf-nmax*:
  n-max (*set-bit-allocating* ((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info* *Va p*) (*nat* (*free-l Va t*)))) *p*
      (*nat* (*free-l Va t*))
      (*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
        (*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))) (*lsizes Va t* ! *nat* (*free-l Va t*)))
      p) = *n-max* (*mem-pool-info Va p*)
  by (*simp add*: *set-bit-def rmhead-free-list-def*)

lemma *mp-alloc-stm3-body-minf-nlvls*:
  n-levels (*set-bit-allocating* ((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info*

Va p) (*nat* (*free-l Va t*)))) *p*
(*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))) (*lsizes Va t*
! *nat* (*free-l Va t*)))
p) = *n-levels* (*mem-pool-info Va p*)
by (*simp add*: *set-bit-def rmhead-free-list-def*)

lemma *mp-alloc-stm3-body-len-lvls*:
length (*levels* (*set-bit-allocating* ((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p*
(*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))) (*lsizes Va t* ! *nat* (*free-l Va t*)))
p)) = *length* (*levels* (*mem-pool-info Va p*))
by(*simp add*: *set-bit-def rmhead-free-list-def*)

lemma *mp-alloc-stm3-body-len-bits*:
length (*bits* (*levels* (*set-bit-allocating*
((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p*
(*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))) (*lsizes Va t* ! *nat* (*free-l Va t*)))
p) !
ii)) = *length* (*bits* ((*levels* (*mem-pool-info Va p*))!*ii*))
apply(*simp add*: *set-bit-def rmhead-free-list-def block-num-def head-free-list-def*)
by (*smt Mem-pool-lvl.select-convs*(*1*) *Mem-pool-lvl.surjective Mem-pool-lvl.update-convs*(*1*)

Mem-pool-lvl.update-convs(*2*) *list-update-beyond list-updt-samelen not-less nth-list-update-eq nth-list-update-neq*)

lemma *mp-alloc-stm3-body-frlst-otherlvl*:
ii ≠ *nat* (*free-l Va t*) ⟹
free-list (*levels* (*set-bit-allocating* ((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p*
(*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat* (*free-l Va t*)))
p) ! *ii*) = *free-list* (*levels* (*mem-pool-info Va p*) ! *ii*)
by(*simp add*: *set-bit-def rmhead-free-list-def block-num-def head-free-list-def*)

lemma *mp-alloc-stm3-body-frlst-samelvl*:
ii < *length* (*levels* (*mem-pool-info Va p*)) ⟹ *ii* = *nat* (*free-l Va t*) ⟹

free-list (*levels* (*set-bit-allocating* ((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p*
(*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat* (*free-l Va t*)))
p) ! *ii*) = *tl* (*free-list* (*levels* (*mem-pool-info Va p*) ! *ii*))
by(*simp add*: *set-bit-def rmhead-free-list-def block-num-def head-free-list-def*)

lemma *mp-alloc-stm3-lm2-inv-1-1*: (*jj*::*nat*) ≠ (*a* − *b*) *div c* ⟹
(*a* − *b*) *mod c* = *0* ⟹
c ≠ *0* ⟹
b + *jj* ∗ *c* ≠ *a* **by** *auto*

lemma *mp-alloc-stm3-lm2-inv-1-2*:
∃ *n*>*0*. (*a*::*nat*) = *4* ∗ *n* ∗ *4* ^ *b* ⟹
ii < *b* ⟹ *0* < *a div 4* ^ *ii*
by (*smt div-eq-0-iff divisors-zero less-imp-le-nat m-mod-div mod-if not-gr0 pow-mod-0 power-not-zero zero-neq-numeral*)

lemma *mp-alloc-stm3-lm2-inv-1*:
¬ *level-empty* (*mem-pool-info Va p*) *ii* ⟹ *p* ∈ *mem-pools Va* ⟹
inv-mempool-info Va ⟹
∀ *ii*<*length* (*lsizes Va t*). *lsizes Va t* ! *ii* = *ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *ii* ⟹
∀ *p*∈*mem-pools Va*.
∀ *i*<*length* (*levels* (*mem-pool-info Va p*)).
(∀ *j*<*length* (*bits* (*levels* (*mem-pool-info Va p*) ! *i*)).
(*get-bit-s Va p i j* = *FREE*) =
(*buf* (*mem-pool-info Va p*) + *j* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ^ *i*)
∈ *set* (*free-list* (*levels* (*mem-pool-info Va p*) ! *i*)))) ∧
(∀ *j*<*length* (*free-list* (*levels* (*mem-pool-info Va p*) ! *i*)).
∃ *n*<*n-max* (*mem-pool-info Va p*) ∗ *4* ^ *i*.
free-list (*levels* (*mem-pool-info Va p*) ! *i*) ! *j* =
buf (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ^ *i*)) ∧
distinct (*free-list* (*levels* (*mem-pool-info Va p*) ! *i*)) ⟹
length (*lsizes Va t*) ≤ *length* (*levels* (*mem-pool-info Va p*)) ⟹
ii < *length* (*lsizes Va t*) ⟹
length (*bits* (*levels* (*set-bit-allocating* ((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) *ii*)) *p ii*
((*head-free-list* (*mem-pool-info Va p*) *ii* − *buf* (*rmhead-free-list* (*mem-pool-info Va p*) *ii*)) *div*
lsizes Va t ! *ii*)
p) !

ii)) =
length (*bits* (*levels* (*mem-pool-info Va p*) ! *ii*)) $\Longrightarrow$
jj < *length* (*bits* (*levels* (*set-bit-allocating*
(λ*a*. *if a* = *p then rmhead-free-list* (*mem-pool-info Va p*)
ii else mem-pool-info Va a) *p ii*
((*head-free-list* (*mem-pool-info Va p*) *ii* − *buf* (*rmhead-free-list*
(*mem-pool-info Va p*) *ii*)) *div*
lsizes Va t ! *ii*)
p) !
ii)) $\Longrightarrow$
nat (*free-l Va t*) = *ii* $\Longrightarrow$
jj ≠ (*head-free-list* (*mem-pool-info Va p*) *ii* − *buf* (*rmhead-free-list* (*mem-pool-info*
Va p) *ii*)) *div lsizes Va t* ! *ii* $\Longrightarrow$
buf (*mem-pool-info Va p*) + *jj* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ˆ *ii*) ≠
head-free-list (*mem-pool-info Va p*) *ii*
apply(*subgoal-tac buf* (*rmhead-free-list* (*mem-pool-info Va p*) *ii*) = *buf* (*mem-pool-info*
Va p))
prefer *2* **apply**(*simp add*:*rmhead-free-list-def*)
apply(*subgoal-tac lsizes Va t* ! *ii* = *ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div*
4 ˆ *ii*)
prefer *2* **apply** *metis*
apply(*subgoal-tac ALIGN4* (*max-sz* (*mem-pool-info Va p*)) = *max-sz* (*mem-pool-info*
Va p))
prefer *2* **using** *inv-mempool-info-maxsz-align4* **apply** *blast*
apply(*subgoal-tac* (*head-free-list* (*mem-pool-info Va p*) *ii* − *buf* (*rmhead-free-list*
(*mem-pool-info Va p*) *ii*)) *mod lsizes Va t* ! *ii* = *0*)
prefer *2* **apply**(*simp add*:*inv-mempool-info-def head-free-list-def Let-def*)
apply(*subgoal-tac* ∃ *n*. *hd* (*free-list* (*levels* (*mem-pool-info Va p*) ! *ii*)) =
buf (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div*
4 ˆ *ii*))
prefer *2* **apply**(*simp add*:*level-empty-def*)
apply(*subgoal-tac* ∀ *j*<*length* (*free-list* (*levels* (*mem-pool-info Va p*) ! *ii*)).
(∃ *n*<*n-max* (*mem-pool-info Va p*) ∗ *4* ˆ *ii*. *free-list* (*levels* (*mem-pool-info*
Va p) ! *ii*) ! *j* =
buf (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4*
ˆ *ii*)))
prefer *2* **apply** (*simp add*: *linorder-not-less*)
apply (*metis* (*full-types*, *hide-lams*) *hd-conv-nth length-greater-0-conv*)
apply (*metis* (*no-types*, *hide-lams*) *add-diff-cancel-left′ mod-mult-self2-is-0*)

apply(*subgoal-tac lsizes Va t* ! *ii* ≠ *0*)
prefer *2* **apply**(*simp add*:*inv-mempool-info-def Let-def*)
apply(*subgoal-tac* ∃ *n*>*0*. *max-sz* (*mem-pool-info Va p*) = *4* ∗ *n* ∗ *4* ˆ *n-levels*
(*mem-pool-info Va p*))
prefer *2* **apply** *blast*
apply(*subgoal-tac length* (*levels* (*mem-pool-info Va p*)) = *n-levels* (*mem-pool-info*
Va p))
prefer *2* **apply** *simp*
using *mp-alloc-stm3-lm2-inv-1-2*[*of max-sz* (*mem-pool-info Va p*) *n-levels* (*mem-pool-info*

Va p) *ii*]
 apply (*metis* (*no-types*, *lifting*) *add-lessD1 le-eq-less-or-eq less-imp-add-positive*)

using *mp-alloc-stm3-lm2-inv-1-1*[*of jj head-free-list* (*mem-pool-info Va p*) *ii buf* (*rmhead-free-list* (*mem-pool-info Va p*) *ii*) *lsizes Va t* ! *ii*]
apply *auto*[*1*]
done

lemma *mp-alloc-stm3-lm2-inv-2*:
 $(a::nat) + jj * b \neq c \Longrightarrow \exists n.\ a + n * b = c \Longrightarrow$
 $(c - a)\ div\ b \neq jj$ **by** *auto*

lemma *mp-alloc-stm3-lm2-inv-thd-waitq*:
inv-thd-waitq Va ⟹
inv-thd-waitq
 (*Va*(|*blk* := (*blk Va*)(*t* := *head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))),
 mem-pool-info :=
 set-bit-allocating ((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p* (*nat* (*free-l Va t*))
 (*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))) (*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
 (*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat* (*free-l Va t*))),
 allocating-node := *allocating-node Va*(*t* ↦
 (|*pool* = *p*, *level* = *nat* (*free-l Va t*),
 block = *block-num*
 (*set-bit-allocating* ((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p* (*nat* (*free-l Va t*))
 (*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))) (*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
 (*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat* (*free-l Va t*)))
 p)
 (*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))) (*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat* (*free-l Va t*)),
 data = *head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))|))|))
apply(*simp add*:*inv-thd-waitq-def*)
apply(*rule conjI*) **apply**(*simp add*: *rmhead-free-list-def*
 head-free-list-def set-bit-def block-num-def)
apply(*rule conjI*) **apply**(*simp add*: *rmhead-free-list-def*
 head-free-list-def set-bit-def block-num-def) **apply** *metis*
apply(*rule conjI*) **apply**(*simp add*: *rmhead-free-list-def*
 head-free-list-def set-bit-def block-num-def)
apply(*simp add*: *rmhead-free-list-def*
 head-free-list-def set-bit-def block-num-def) **apply** *metis*
done

lemma *mp-alloc-stm3-lm2-inv-aux-vars-1*:
 ¬ (*pool n* = *p* ∧ *level n* = *nat* (*free-l Va t*) ∧ *block n* =
 block-num (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))

(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat* (*free-l Va t*))) ⟹
get-bit-s (*Va*(|*mem-pool-info* :=
set-bit-allocating ((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p*
(*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat* (*free-l Va t*)))|))
(*pool n*) (*level n*) (*block n*) = *get-bit-s Va* (*pool n*) (*level n*) (*block n*)

apply(*rule subst*[**where** *t*= *get-bit-s*
(*Va*(|*mem-pool-info* :=
set-bit-allocating ((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p* (*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat* (*free-l Va t*)))|))
(*pool n*) (*level n*) (*block n*) **and** *s*=*get-bit-s*
(*Va*(|*mem-pool-info* :=
set-bit-allocating (*mem-pool-info Va*) *p* (*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat* (*free-l Va t*)))|))
(*pool n*) (*level n*) (*block n*)])
apply(*simp add*:*rmhead-free-list-def set-bit-def*)
apply (*smt Mem-pool-lvl.select-convs*(*1*) *Mem-pool-lvl.simps*(*4*) *Mem-pool-lvl.surjective*
Mem-pool-lvl.update-convs(*2*) *linorder-not-less list-update-beyond nth-list-update-eq nth-list-update-neq*)
apply(*simp add*:*rmhead-free-list-def set-bit-def*)
apply (*smt Mem-pool-lvl.select-convs*(*1*) *Mem-pool-lvl.simps*(*4*) *Mem-pool-lvl.surjective*
Mem-pool-lvl.update-convs(*2*) *linorder-not-less list-update-beyond nth-list-update-eq nth-list-update-neq*)
done

lemma *mp-alloc-stm3-lm2-inv-aux-vars-2*:
inv-mempool-info Va ⟹
¬ *level-empty* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)) ⟹
p ∈ *mem-pools Va* ⟹
pool n = *p* ∧
level n = *nat* (*free-l Va t*) ∧
block n =
block-num (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))) (*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat* (*free-l Va t*)) ⟹

get-bit-s (*Va*(|*mem-pool-info* :=
set-bit-allocating ((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p*
(*nat* (*free-l Va t*))
(*block-num* (*mem-pool-info Va p*) (*free-list* (*levels* (*mem-pool-info Va p*) ! *nat* (*free-l Va t*)) ! *NULL*)
(*max-sz* (*mem-pool-info Va p*) *div* *4* ˆ *nat* (*free-l Va t*)))|))
(*pool n*) (*level n*) (*block n*) =
get-bit
(*set-bit-allocating* (*mem-pool-info Va*) *p* (*nat* (*free-l Va t*))
(*block-num* (*mem-pool-info Va p*) (*free-list* (*levels* (*mem-pool-info Va p*) ! *nat* (*free-l Va t*)) ! *NULL*)
(*max-sz* (*mem-pool-info Va p*) *div* *4* ˆ *nat* (*free-l Va t*))))
p (*nat* (*free-l Va t*))
(*block-num* (*mem-pool-info Va p*) (*free-list* (*levels* (*mem-pool-info Va p*) ! *nat* (*free-l Va t*)) ! *NULL*)
(*max-sz* (*mem-pool-info Va p*) *div* *4* ˆ *nat* (*free-l Va t*)))
apply(*rule subst*[**where** *t*=*p* **and** *s*=*pool n*]) **apply** *simp*
apply(*rule subst*[**where** *t*=*nat* (*free-l Va t*) **and** *s*=*level n*]) **apply** *simp*
apply(*rule subst*[**where** *t*=(*block-num* (*mem-pool-info Va* (*pool n*)) (*free-list* (*levels* (*mem-pool-info Va* (*pool n*)) ! *level n*) ! *NULL*)
(*max-sz* (*mem-pool-info Va* (*pool n*)) *div* *4* ˆ *level n*)) **and** *s*=*block n*])
apply(*simp add*:*level-empty-def block-num-def rmhead-free-list-def head-free-list-def*)
apply (*metis hd-conv-nth inv-mempool-info-maxsz-align4*)
apply(*rule subst*[**where** *t*=*get-bit-s*
(*Va*(|*mem-pool-info* :=
set-bit-allocating ((*mem-pool-info Va*)(*pool n* := *rmhead-free-list* (*mem-pool-info Va* (*pool n*)) (*level n*))) (*pool n*) (*level n*)
(*block n*)|))
(*pool n*) (*level n*) (*block n*) **and**
s=*get-bit* (*set-bit-allocating* ((*mem-pool-info Va*)(*pool n* := *rmhead-free-list* (*mem-pool-info Va* (*pool n*)) (*level n*))) (*pool n*) (*level n*)
(*block n*)) (*pool n*) (*level n*) (*block n*)]) **apply** *auto*[*1*]
apply(*simp add*:*rmhead-free-list-def set-bit-def*)
apply(*case-tac level n* ≥ *length* (*levels* (*mem-pool-info Va* (*pool n*))))
apply *auto*
done

lemma *mp-alloc-stm3-lm2-inv-aux-vars*:
¬ *level-empty* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)) ⟹
allocating-node Va t = *None* ⟹
freeing-node Va t = *None* ⟹
inv-cur Va ∧ *inv-thd-waitq Va* ∧ *inv-mempool-info Va* ∧ *inv-bitmap-freelist Va* ∧ *inv-bitmap Va* ∧ *inv-aux-vars Va* ⟹
p ∈ *mem-pools Va* ⟹
ETIMEOUT ≤ *timeout* ⟹
timeout = *ETIMEOUT* ⟶ *tmout Va t* = *ETIMEOUT* ⟹
¬ *rf Va t* ⟹
∀ *ii*<*length* (*lsizes Va t*). *lsizes Va t* ! *ii* = *ALIGN4* (*max-sz* (*mem-pool-info Va*

p)) *div 4* ˆ *ii* $\Longrightarrow$
length (*lsizes Va t*) $\leq$ *n-levels* (*mem-pool-info Va p*) $\Longrightarrow$
alloc-l Va t < *int* (*n-levels* (*mem-pool-info Va p*)) $\Longrightarrow$
free-l Va t $\leq$ *alloc-l Va t* $\Longrightarrow$
$\neg$ *free-l Va t* < *OK* $\Longrightarrow$
alloc-l Va t = *int* (*length* (*lsizes Va t*)) − *1* $\wedge$ *length* (*lsizes Va t*) = *n-levels* (*mem-pool-info Va p*) $\vee$
alloc-l Va t = *int* (*length* (*lsizes Va t*)) − *2* $\wedge$ *lsizes Va t* ! *nat* (*alloc-l Va t* + *1*) < *sz* $\Longrightarrow$
length (*lsizes Va t*) $\leq$ *length* (*levels* (*mem-pool-info Va p*)) $\Longrightarrow$
nat (*free-l Va t*) < *length* (*lsizes Va t*) $\Longrightarrow$
inv-aux-vars
(*Va*(|*blk* := (*blk Va*)(*t* := *head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))),
mem-pool-info :=
set-bit-allocating ((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p* (*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *nat* (*free-l Va t*))),
allocating-node := *allocating-node Va*(*t* $\mapsto$
(|*pool* = *p*, *level* = *nat* (*free-l Va t*),
block = *block-num*
(*set-bit-allocating* ((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p*
(*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *nat* (*free-l Va t*)))
p)
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *nat* (*free-l Va t*)),
data = *head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))|))|))
apply(*unfold inv-aux-vars-def*)
apply(*rule conjI*)

apply *clarify*
apply(*subgoal-tac freeing-node*
(*Va*(|*blk* := (*blk Va*)(*t* := *head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))),
mem-pool-info :=
set-bit-allocating ((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p*
(*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))

(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat* (*free-l Va t*))),
allocating-node := *allocating-node Va*(*t* ↦
(|*pool* = *p*, *level* = *nat* (*free-l Va t*),
block = *block-num*
(*set-bit-allocating* ((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))))
p (*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat* (*free-l Va t*)))
p)
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat* (*free-l Va t*)),
data = *head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))|)))|))
= *freeing-node Va*)
prefer *2* **apply** *simp*
apply(*subgoal-tac get-bit-s Va* (*pool n*) (*level n*) (*block n*) = *FREEING*)
prefer *2* **apply** *auto*[*1*]
apply(*case-tac* (*pool n*) = *p* ∧ (*level n*) = *nat* (*free-l Va t*)
∧ (*block n*) = (*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat* (*free-l Va t*))))
apply(*subgoal-tac get-bit-s Va p* (*nat* (*free-l Va t*)) (*block-num* (*mem-pool-info Va p*)
((*free-list* ((*levels* (*mem-pool-info Va p*)) ! (*nat* (*free-l Va t*)))) ! *0*)
(*max-sz* (*mem-pool-info Va p*) *div 4* ^ (*nat* (*free-l Va t*))))
= *FREE*)
prefer *2* **apply**(*simp add*:*level-empty-def*) **using** *inv-bitmap-freelist-fl-FREE*
apply *auto*[*1*]
apply(*subgoal-tac block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat* (*free-l Va t*))
= (*block-num* (*mem-pool-info Va p*) (*free-list* (*levels* (*mem-pool-info Va p*) ! *nat* (*free-l Va t*)) ! *NULL*)
(*max-sz* (*mem-pool-info Va p*) *div 4* ^ *nat* (*free-l Va t*))))
prefer *2* **apply**(*simp add*:*rmhead-free-list-def head-free-list-def block-num-def level-empty-def*)
apply (*metis hd-conv-nth inv-mempool-info-maxsz-align4*)
apply *simp*

apply(*subgoal-tac get-bit-s*
(Va(|blk := (blk Va)(t := head-free-list (mem-pool-info Va p) (nat (free-l Va t))),
mem-pool-info :=
set-bit-allocating ((mem-pool-info Va)(p := rmhead-free-list (mem-pool-info Va p) (nat (free-l Va t)))) p
(nat (free-l Va t))
(block-num (rmhead-free-list (mem-pool-info Va p) (nat (free-l Va t)))
(head-free-list (mem-pool-info Va p) (nat (free-l Va t)))
(ALIGN4 (max-sz (mem-pool-info Va p)) div 4 ^ nat (free-l Va t))),
allocating-node := allocating-node Va(t ↦
(|pool = p, level = nat (free-l Va t),
block = block-num
(set-bit-allocating ((mem-pool-info Va)(p := rmhead-free-list (mem-pool-info Va p) (nat (free-l Va t))))
p (nat (free-l Va t))
(block-num (rmhead-free-list (mem-pool-info Va p) (nat (free-l Va t)))
(head-free-list (mem-pool-info Va p) (nat (free-l Va t)))
(ALIGN4 (max-sz (mem-pool-info Va p)) div 4 ^ nat (free-l Va t)))
p)
(head-free-list (mem-pool-info Va p) (nat (free-l Va t)))
(ALIGN4 (max-sz (mem-pool-info Va p)) div 4 ^ nat (free-l Va t)),
data = head-free-list (mem-pool-info Va p) (nat (free-l Va t))|))|))
(pool n) (level n) (block n) = get-bit-s Va (pool n) (level n) (block n))
apply *simp*
apply(*subgoal-tac get-bit-s*
(Va(|blk := (blk Va)(t := head-free-list (mem-pool-info Va p) (nat (free-l Va t))),
mem-pool-info :=
set-bit-allocating ((mem-pool-info Va)(p := rmhead-free-list (mem-pool-info Va p) (nat (free-l Va t)))) p
(nat (free-l Va t))
(block-num (rmhead-free-list (mem-pool-info Va p) (nat (free-l Va t)))
(head-free-list (mem-pool-info Va p) (nat (free-l Va t)))
(ALIGN4 (max-sz (mem-pool-info Va p)) div 4 ^ nat (free-l Va t))),
allocating-node := allocating-node Va(t ↦
(|pool = p, level = nat (free-l Va t),
block = block-num
(set-bit-allocating ((mem-pool-info Va)(p := rmhead-free-list (mem-pool-info Va p) (nat (free-l Va t))))
p (nat (free-l Va t))

(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat* (*free-l Va t*)))
p)
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat* (*free-l Va t*)),
data = *head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))⦈))⦈))
(*pool n*) (*level n*) (*block n*)
= *get-bit-s* (*Va*⦇ *mem-pool-info* :=
set-bit-allocating ((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p*
(*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat* (*free-l Va t*)))⦈) (*pool n*) (*level n*) (*block n*))
prefer *2* **apply** *force*
apply(*frule mp-alloc-stm3-lm2-inv-aux-vars-1*) **apply** *simp*

apply(*rule conjI*)

apply *clarify*
apply(*subgoal-tac get-bit-s*
(*Va*⦇*blk* := (*blk Va*)(*t* := *head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))),
mem-pool-info :=
set-bit-allocating ((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p*
(*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat* (*free-l Va t*))),
allocating-node := *allocating-node Va*(*t* ↦
⦇*pool* = *p*, *level* = *nat* (*free-l Va t*),
block = *block-num*
(*set-bit-allocating* ((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))))
p (*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat* (*free-l Va t*)))

p)
(head-free-list (mem-pool-info Va p) (nat (free-l Va t)))
(ALIGN4 (max-sz (mem-pool-info Va p)) div 4 ^ nat (free-l Va t)),
data = head-free-list (mem-pool-info Va p) (nat (free-l Va t))⦈)⦈)
(pool n) (level n) (block n)
= get-bit-s (Va⦇ mem-pool-info :=
set-bit-allocating ((mem-pool-info Va)(p := rmhead-free-list (mem-pool-info Va p) (nat (free-l Va t)))) p
(nat (free-l Va t))
(block-num (rmhead-free-list (mem-pool-info Va p) (nat (free-l Va t)))
(head-free-list (mem-pool-info Va p) (nat (free-l Va t)))
(ALIGN4 (max-sz (mem-pool-info Va p)) div 4 ^ nat (free-l Va t)))⦈) (pool n) (level n) (block n))
prefer *2* **apply** *force*
apply(*case-tac (pool n) = p ∧ (level n) = nat (free-l Va t)*
∧ (block n) = (block-num (rmhead-free-list (mem-pool-info Va p) (nat (free-l Va t)))
(head-free-list (mem-pool-info Va p) (nat (free-l Va t)))
(ALIGN4 (max-sz (mem-pool-info Va p)) div 4 ^ nat (free-l Va t))))
apply(*subgoal-tac get-bit-s (Va⦇ mem-pool-info :=*
set-bit-allocating ((mem-pool-info Va)(p := rmhead-free-list (mem-pool-info Va p) (nat (free-l Va t)))) p
(nat (free-l Va t))
(block-num (rmhead-free-list (mem-pool-info Va p) (nat (free-l Va t)))
(head-free-list (mem-pool-info Va p) (nat (free-l Va t)))
(ALIGN4 (max-sz (mem-pool-info Va p)) div 4 ^ nat (free-l Va t)))⦈) (pool n) (level n) (block n) = ALLOCATING)
apply *simp*
apply(*subgoal-tac get-bit (set-bit-allocating (mem-pool-info Va) p (nat (free-l Va t))*
(block-num (mem-pool-info Va p) (free-list (levels (mem-pool-info Va p) ! nat (free-l Va t)) ! NULL)
(max-sz (mem-pool-info Va p) div 4 ^ nat (free-l Va t))))
p (nat (free-l Va t))
(block-num (mem-pool-info Va p) (free-list (levels (mem-pool-info Va p) ! nat (free-l Va t)) ! NULL)
(max-sz (mem-pool-info Va p) div 4 ^ nat (free-l Va t))) = ALLOCATING)
prefer *2*
apply(*rule set-bit-get-bit-eq[of nat (free-l Va t) mem-pool-info Va p*
block-num (mem-pool-info Va p) (free-list (levels (mem-pool-info Va p) ! nat (free-l Va t)) ! NULL)
(max-sz (mem-pool-info Va p) div 4 ^ nat (free-l Va t)) set-bit-allocating (mem-pool-info Va) p (nat (free-l Va t))

(*block-num* (*mem-pool-info Va p*) (*free-list* (*levels* (*mem-pool-info Va p*) ! *nat* (*free-l Va t*)) ! *NULL*)
(*max-sz* (*mem-pool-info Va p*) *div 4* ^ *nat* (*free-l Va t*)))])
apply *simp* **apply**(*simp add*:*level-empty-def*) **using** *inv-bitmap-freelist-fl-bnum-in*[*of Va p nat* (*free-l Va t*) *0*]
apply (*meson le-trans length-greater-0-conv linorder-not-less*) **apply** *simp*

apply(*subgoal-tac get-bit-s* (*Va*(| *mem-pool-info* :=
set-bit-allocating ((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p*
(*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat* (*free-l Va t*)))|)) (*pool n*) (*level n*) (*block n*)
= *get-bit* (*set-bit-allocating* (*mem-pool-info Va*) *p* (*nat* (*free-l Va t*))
(*block-num* (*mem-pool-info Va p*) (*free-list* (*levels* (*mem-pool-info Va p*) ! *nat* (*free-l Va t*)) ! *NULL*)
(*max-sz* (*mem-pool-info Va p*) *div 4* ^ *nat* (*free-l Va t*))))
p (*nat* (*free-l Va t*))
(*block-num* (*mem-pool-info Va p*) (*free-list* (*levels* (*mem-pool-info Va p*) ! *nat* (*free-l Va t*)) ! *NULL*)
(*max-sz* (*mem-pool-info Va p*) *div 4* ^ *nat* (*free-l Va t*))))

apply *simp*
apply(*rule subst*[**where** *t*=*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat* (*free-l Va t*))
and *s*=*block-num* (*mem-pool-info Va p*) (*free-list* (*levels* (*mem-pool-info Va p*) ! *nat* (*free-l Va t*)) ! *NULL*)
(*max-sz* (*mem-pool-info Va p*) *div 4* ^ *nat* (*free-l Va t*))])
apply(*simp add*:*level-empty-def block-num-def rmhead-free-list-def head-free-list-def*)
apply (*metis hd-conv-nth inv-mempool-info-maxsz-align4*)
using *mp-alloc-stm3-lm2-inv-aux-vars-2*[*of Va p t*] **apply** *blast*

apply(*subgoal-tac get-bit* (*mem-pool-info Va*) (*pool n*) (*level n*) (*block n*) = *FREEING*)
prefer *2*
apply(*subgoal-tac get-bit-s*
(*Va*(|*mem-pool-info* :=
set-bit-allocating ((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p*
(*nat* (*free-l Va t*))

(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *nat* (*free-l Va t*)))⦈)
(*pool n*) (*level n*) (*block n*) = *get-bit* (*mem-pool-info Va*) (*pool n*) (*level n*) (*block n*))
prefer *2* **using** *mp-alloc-stm3-lm2-inv-aux-vars-1*[*of - p Va t*] **apply** *blast*
apply *simp*
apply(*subgoal-tac mem-block-addr-valid Va n*)
prefer *2* **apply**(*simp add*:*mem-block-addr-valid-def*)
apply (*metis mp-alloc-stm3-body-meminfo mp-alloc-stm3-body-minf-buf mp-alloc-stm3-body-minf-maxsz*)
apply(*subgoal-tac* ∃ *t*. *freeing-node Va t* = *Some n*) **prefer** *2* **apply** *metis*
apply(*subgoal-tac* ∀ *ta*. *freeing-node Va ta* = *freeing-node*
(*Va*⦇*blk* := (*blk Va*)(*t* := *head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))),
mem-pool-info :=
set-bit-allocating ((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p*
(*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *nat* (*free-l Va t*))),
allocating-node := *allocating-node Va*(*t* ↦
⦇*pool* = *p*, *level* = *nat* (*free-l Va t*),
block = *block-num*
(*set-bit-allocating*
((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p*
(*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *nat* (*free-l Va t*)))
p)
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *nat* (*free-l Va t*)),
data = *head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))⦈)⦈) *ta*)
prefer *2* **apply** *force*
apply *metis*

apply(*rule conjI*)

apply *clarify*

apply(*subgoal-tac get-bit-s*
(*Va*(|*blk* := (*blk Va*)(*t* := *head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))),
mem-pool-info :=
set-bit-allocating ((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p*
(*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div* *4* ^ *nat* (*free-l Va t*))),
allocating-node := *allocating-node Va*(*t* ↦
(|*pool* = *p*, *level* = *nat* (*free-l Va t*),
block = *block-num*
(*set-bit-allocating* ((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))))
p (*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div* *4* ^ *nat* (*free-l Va t*)))
p)
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div* *4* ^ *nat* (*free-l Va t*)),
data = *head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))|))|))
(*pool n*) (*level n*) (*block n*)
= *get-bit-s* (*Va*(| *mem-pool-info* :=
set-bit-allocating ((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p*
(*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div* *4* ^ *nat* (*free-l Va t*)))|)) (*pool n*) (*level n*) (*block n*))
prefer *2* **apply** *force*
apply(*subgoal-tac get-bit-s* (*Va*(| *mem-pool-info* :=
set-bit-allocating ((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p*
(*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div* *4* ^ *nat* (*free-l Va t*)))|)) (*pool n*) (*level n*) (*block n*)
= *get-bit-s* (*Va*(| *mem-pool-info* := *set-bit-allocating* (*mem-pool-info Va*) *p*

(nat (free-l Va t))
(block-num (mem-pool-info Va p)
(head-free-list (mem-pool-info Va p) (nat (free-l Va t)))
(ALIGN4 (max-sz (mem-pool-info Va p)) div 4 ^ nat (free-l Va t)))⦈) (pool n) (level n) (block n))
prefer *2* **apply**(*simp add:rmhead-free-list-def set-bit-def block-num-def*)
apply (*smt Mem-pool-lvl.select-convs(1) Mem-pool-lvl.simps(4) Mem-pool-lvl.surjective*

linorder-not-less list-update-beyond nth-list-update-eq nth-list-update-neq)

apply(*subgoal-tac get-bit-s (Va⦇ mem-pool-info := set-bit-allocating (mem-pool-info Va) p*
(nat (free-l Va t))
(block-num (mem-pool-info Va p)
(head-free-list (mem-pool-info Va p) (nat (free-l Va t)))
(ALIGN4 (max-sz (mem-pool-info Va p)) div 4 ^ nat (free-l Va t)))⦈) (pool n) (level n) (block n) = ALLOCATING)
apply *simp*
apply(*case-tac t = ta*)

apply(*subgoal-tac (pool n) = p ∧ (level n) = nat (free-l Va t)*
∧ (block n) = block-num (mem-pool-info Va p)
(head-free-list (mem-pool-info Va p) (nat (free-l Va t)))
(ALIGN4 (max-sz (mem-pool-info Va p)) div 4 ^ nat (free-l Va t))))
prefer *2* **apply**(*rule conjI*) **apply** *auto[1]* **apply**(*rule conjI*) **apply** *auto[1]*
apply(*subgoal-tac (block n) = block-num (set-bit-allocating ((mem-pool-info Va)(p := rmhead-free-list (mem-pool-info Va p) (nat (free-l Va t))))*
p (nat (free-l Va t))
(block-num (rmhead-free-list (mem-pool-info Va p) (nat (free-l Va t)))
(head-free-list (mem-pool-info Va p) (nat (free-l Va t)))
(ALIGN4 (max-sz (mem-pool-info Va p)) div 4 ^ nat (free-l Va t)))
p)
(head-free-list (mem-pool-info Va p) (nat (free-l Va t)))
(ALIGN4 (max-sz (mem-pool-info Va p)) div 4 ^ nat (free-l Va t))))
prefer *2* **apply** *auto[1]*
apply(*subgoal-tac block-num (set-bit-allocating ((mem-pool-info Va)(p := rmhead-free-list (mem-pool-info Va p) (nat (free-l Va t))))*
p (nat (free-l Va t))
(block-num (rmhead-free-list (mem-pool-info Va p) (nat (free-l Va t)))
(head-free-list (mem-pool-info Va p) (nat (free-l Va t)))
(ALIGN4 (max-sz (mem-pool-info Va p)) div 4 ^

nat (*free-l Va t*)))
p)
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *nat* (*free-l Va t*))
= *block-num* (*mem-pool-info Va p*)
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *nat* (*free-l Va t*)))
prefer *2* **apply**(*simp add:level-empty-def block-num-def set-bit-def rmhead-free-list-def*)
apply *simp*
apply(*subgoal-tac nat* (*free-l Va t*) < *length* (*levels* (*mem-pool-info Va p*)))
prefer *2* **apply** *simp*
apply(*subgoal-tac block-num* (*mem-pool-info Va p*) (*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *nat* (*free-l Va t*))
< *length* (*bits* (*levels* (*mem-pool-info Va p*) ! *nat* (*free-l Va t*))))
prefer *2* **apply**(*rule subst*[**where** *t*=*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) **and** *s*=*max-sz* (*mem-pool-info Va p*)])
apply (*metis inv-mempool-info-maxsz-align4*)
apply(*frule inv-bitmap-freelist-fl-bnum-in*[*of Va p nat* (*free-l Va t*) *0*])
apply *simp* **apply** *simp* **apply** *simp* **apply**(*simp add:level-empty-def*)
apply(*simp add:level-empty-def head-free-list-def*) **apply** (*metis hd-conv-nth*)

using *set-bit-get-bit-eq2*[*of nat* (*free-l Va t*) *Va p block-num* (*mem-pool-info Va p*)
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *nat* (*free-l Va t*)) *ALLOCATING*] **apply** *metis*

apply(*subgoal-tac allocating-node*
(*Va*(|*blk* := (*blk Va*)(*t* := *head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))),
mem-pool-info :=
set-bit-allocating ((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p*
(*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *nat* (*free-l Va t*))),
allocating-node := *allocating-node Va*(*t* ↦
(|*pool* = *p*, *level* = *nat* (*free-l Va t*),

block = *block-num*
(*set-bit-allocating* ((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))))
p (*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat* (*free-l Va t*)))
p)
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat* (*free-l Va t*)),
data = *head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))⦈)⦈)
ta = *allocating-node Va ta*) **prefer** *2* **apply** *force*
apply(*subgoal-tac get-bit* (*mem-pool-info Va*) (*pool n*) (*level n*) (*block n*) = *ALLOCATING*)
prefer *2* **apply** *metis*
apply(*subgoal-tac block-num* (*mem-pool-info Va p*)
((*free-list* ((*levels* (*mem-pool-info Va p*)) ! (*nat* (*free-l Va t*)))) ! *0*)
(*max-sz* (*mem-pool-info Va p*) *div 4* ^ (*nat* (*free-l Va t*)))
= *block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat* (*free-l Va t*)))
prefer *2* **apply**(*simp add:block-num-def rmhead-free-list-def head-free-list-def*)
apply (*simp add*: *hd-conv-nth inv-mempool-info-maxsz-align4 level-empty-def*)
apply(*case-tac* (*pool n*) = *p* ∧ (*level n*) = *nat* (*free-l Va t*)
∧ (*block n*) = (*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat* (*free-l Va t*))))
apply(*subgoal-tac get-bit-s Va p* (*nat* (*free-l Va t*)) (*block-num* (*mem-pool-info Va p*)
((*free-list* ((*levels* (*mem-pool-info Va p*)) ! (*nat* (*free-l Va t*)))) ! *0*)
(*max-sz* (*mem-pool-info Va p*) *div 4* ^ (*nat* (*free-l Va t*))))
= *FREE*)
prefer *2* **apply**(*simp add:level-empty-def*) **using** *inv-bitmap-freelist-fl-FREE*[*of Va p nat* (*free-l Va t*) *0*]
apply *auto*[*1*]

apply *simp*

apply(*subgoal-tac get-bit-s* (*Va*⦇*mem-pool-info* :=
set-bit-allocating (*mem-pool-info Va*) *p* (*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *nat* (*free-l Va t*)))⦈)
(*pool n*) (*level n*) (*block n*) = *get-bit-s Va* (*pool n*) (*level n*) (*block n*))
prefer *2* **apply** (*metis set-bit-get-bit-neq2*)
apply(*rule subst*[**where** *t*=*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) **and** *s*=*max-sz* (*mem-pool-info Va p*)])
apply (*metis inv-mempool-info-maxsz-align4*)
apply (*simp add*: *hd-conv-nth level-empty-def*)
apply (*smt nat-less-iff nth-equalityI set-bit-get-bit-eq set-bit-get-bit-neq set-bit-prev-len zle-int*)

apply(*rule conjI*)

apply *clarify*
apply(*case-tac* (*pool n*) = *p* ∧ (*level n*) = *nat* (*free-l Va t*)
∧ (*block n*) = (*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *nat* (*free-l Va t*))))
apply(*subgoal-tac allocating-node*
(*Va*⦇*blk* := (*blk Va*)(*t* := *head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))),
mem-pool-info :=
set-bit-allocating ((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p*
(*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *nat* (*free-l Va t*))),
allocating-node := *allocating-node Va*(*t* ↦
⦇*pool* = *p*, *level* = *nat* (*free-l Va t*),
block = *block-num*
(*set-bit-allocating*
((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p*
(*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *nat* (*free-l Va t*)))

p)
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat*
(*free-l Va t*)),
data = *head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va*
t))⦈)⦈) *t* =
Some n)
prefer *2* **apply**(*rule subst*[**where** *t*=*allocating-node*
(*Va*⦇*blk* := (*blk Va*)(*t* := *head-free-list* (*mem-pool-info Va p*) (*nat*
(*free-l Va t*))),
mem-pool-info :=
set-bit-allocating ((*mem-pool-info Va*)(*p* := *rmhead-free-list*
(*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p*
(*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l*
Va t)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat* (*free-l*
Va t))),
allocating-node := *allocating-node Va*(*t* ↦
⦇*pool* = *p*, *level* = *nat* (*free-l Va t*),
block = *block-num*
(*set-bit-allocating*
((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info*
Va p) (*nat* (*free-l Va t*)))) *p*
(*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*)
(*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va*
t)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^
nat (*free-l Va t*)))
p)
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat*
(*free-l Va t*)),
data = *head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va*
t))⦈)⦈) *t* **and** *s*=*Some* ⦇*pool* = *p*, *level* = *nat* (*free-l Va t*),
block = *block-num*
(*set-bit-allocating*
((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info*
Va p) (*nat* (*free-l Va t*)))) *p*
(*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*)
(*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va*
t)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^
nat (*free-l Va t*)))

p)
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *nat* (*free-l Va t*)),
data = *head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))⦈])
apply *force*
apply(*rule subst*[**where** *t*=*block-num*
(*set-bit-allocating*
((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p*
(*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *nat* (*free-l Va t*)))
p)
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *nat* (*free-l Va t*))
and *s*=*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *nat* (*free-l Va t*))])
apply(*simp add*: *set-bit-def rmhead-free-list-def block-num-def*)
apply(*simp add*:*mem-block-addr-valid-def*)
apply(*subgoal-tac buf* (*set-bit-allocating* ((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p*
(*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *nat* (*free-l Va t*)))
p) +
block-num (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *nat* (*free-l Va t*)) ∗
(*max-sz* (*set-bit-allocating* ((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p*
(*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *nat* (*free-l Va t*)))
p) *div*
4 ˆ *nat* (*free-l Va t*)) = *head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va*

t)))
apply *auto*[*1*]
apply(*rule subst*[**where** *t=buf* (*set-bit-allocating* ((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p*
(*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *nat* (*free-l Va t*)))
p) **and** *s=buf* (*mem-pool-info Va p*)])
apply(*simp add*:*set-bit-def block-num-def rmhead-free-list-def*)
apply(*rule subst*[**where** *t=block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
((*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *nat* (*free-l Va t*)))
and *s=block-num* (*mem-pool-info Va p*)
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
((*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *nat* (*free-l Va t*)))])
apply(*simp add*:*set-bit-def block-num-def rmhead-free-list-def*)
apply(*rule subst*[**where** *t=max-sz* (*set-bit-allocating* ((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p*
(*nat* (*free-l Va t*))
(*block-num* (*mem-pool-info Va p*) (*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *nat* (*free-l Va t*)))
p) **and** *s=max-sz* (*mem-pool-info Va p*)])
apply(*simp add*:*set-bit-def block-num-def rmhead-free-list-def*)
apply(*rule subst*[**where** *t=ALIGN4* (*max-sz* (*mem-pool-info Va p*)) **and** *s=max-sz* (*mem-pool-info Va p*)])
apply (*metis inv-mempool-info-maxsz-align4*)

apply(*rule ref-byblkn-self*[*of Va p head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)) (*max-sz* (*mem-pool-info Va p*) *div 4* ˆ *nat* (*free-l Va t*))])
apply(*simp add*:*level-empty-def head-free-list-def*)
using *inv-buf-le-fl*[*of Va p nat* (*free-l Va t*) *0*]
apply (*smt hd-conv-nth length-greater-0-conv nat-less-iff zle-int*)
apply(*simp add*:*level-empty-def head-free-list-def*)
using *inv-fl-mod-sz0*[*of Va p nat* (*free-l Va t*) *0*]
apply (*smt hd-conv-nth le-eq-less-or-eq le-trans length-greater-0-conv nat-eq-iff nat-less-iff*)
apply *auto*[*1*]

apply(*subgoal-tac get-bit* (*mem-pool-info Va*) (*pool n*) (*level n*) (*block n*) = *ALLOCATING*)
prefer *2*

apply(*subgoal-tac get-bit-s*
(*Va*(|*mem-pool-info* :=
set-bit-allocating ((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p*
(*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div* *4* ˆ *nat* (*free-l Va t*))))|))
(*pool n*) (*level n*) (*block n*) = *get-bit* (*mem-pool-info Va*) (*pool n*) (*level n*) (*block n*))
prefer *2* **using** *mp-alloc-stm3-lm2-inv-aux-vars-1*[*of - p Va t*] **apply** *blast*
apply *force*

apply(*subgoal-tac* ∃ *ta*. *ta* ≠ *t* ∧ *allocating-node Va ta* = *Some n*)
prefer *2* **apply**(*subgoal-tac mem-block-addr-valid Va n*) **apply** *metis*
apply(*simp add*:*mem-block-addr-valid-def*)
apply (*metis mp-alloc-stm3-body-meminfo mp-alloc-stm3-body-minf-buf mp-alloc-stm3-body-minf-maxsz*)

apply *auto*[*1*]

apply(*rule conjI*)

apply *clarify*
apply(*subgoal-tac* ∀ *t*. *freeing-node*
(*Va*(|*blk* := (*blk Va*)(*t* := *head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))),
mem-pool-info :=
set-bit-allocating ((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p*
(*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div* *4* ˆ *nat* (*free-l Va t*))),
allocating-node := *allocating-node Va*(*t* ↦
(|*pool* = *p*, *level* = *nat* (*free-l Va t*),
block = *block-num*
(*set-bit-allocating*
((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p*
(*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div* *4* ˆ *nat* (*free-l Va t*)))
p)
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div* *4* ˆ *nat* (*free-l*

Va t)),
data = *head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))⦈)⦈)
t = *freeing-node Va t*)
prefer *2* **apply** *force*
apply *auto*[*1*]

apply(*rule conjI*)

apply *clarify*
apply(*case-tac t* = *t1*)
apply(*subgoal-tac get-bit-s Va* (*pool n1*) (*level n1*) (*block n1*) = *FREE*)
prefer *2*
apply(*subgoal-tac pool n1* = *p* ∧ *level n1* = *nat* (*free-l Va t*) ∧ *block n1* = *block-num*
(*set-bit-allocating*
((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p*
(*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *nat* (*free-l Va t*)))
p)
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *nat* (*free-l Va t*)))
prefer *2* **apply** *auto*[*1*]
apply(*subgoal-tac block-num*
(*set-bit-allocating*
((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p*
(*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *nat* (*free-l Va t*)))
p)
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *nat* (*free-l Va t*)) = *block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *nat* (*free-l Va t*)))
prefer *2* **apply**(*simp add*: *set-bit-def rmhead-free-list-def block-num-def*)
apply(*subgoal-tac block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat*

(*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
((*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat* (*free-l Va t*))) =
block-num (*mem-pool-info Va p*)
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
((*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat* (*free-l Va t*))))
prefer *2* **apply**(*simp add*:*set-bit-def block-num-def rmhead-free-list-def*)
apply(*simp add*:*level-empty-def head-free-list-def*)
using *inv-bitmap-freelist-fl-FREE*[*of Va p nat* (*free-l Va t*) *0*]
apply (*smt hd-conv-nth inv-mempool-info-maxsz-align4 le-trans length-greater-0-conv linorder-not-less*)
apply(*subgoal-tac get-bit-s Va* (*pool n2*) (*level n2*) (*block n2*) = *ALLOCATING*)
prefer *2* **apply** *auto*[*1*]
apply *auto*[*1*]

apply(*case-tac t* = *t2*)
apply(*subgoal-tac get-bit-s Va* (*pool n2*) (*level n2*) (*block n2*) = *FREE*)
prefer *2*
apply(*subgoal-tac pool n2* = *p* ∧ *level n2* = *nat* (*free-l Va t*) ∧ *block n2* = *block-num*
(*set-bit-allocating*
((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p*
(*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat* (*free-l Va t*)))
p)
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat* (*free-l Va t*)))
prefer *2* **apply** *auto*[*1*]
apply(*subgoal-tac block-num*
(*set-bit-allocating*
((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p*
(*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat* (*free-l Va t*)))
p)

(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat* (*free-l Va t*)) = *block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat* (*free-l Va t*)))
prefer *2* **apply**(*simp add*: *set-bit-def rmhead-free-list-def block-num-def*)
apply(*subgoal-tac block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
((*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat* (*free-l Va t*))) =
block-num (*mem-pool-info Va p*)
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
((*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat* (*free-l Va t*))))
prefer *2* **apply**(*simp add:set-bit-def block-num-def rmhead-free-list-def*)
apply(*unfold level-empty-def head-free-list-def*)[*1*]
using *inv-bitmap-freelist-fl-FREE*[*of Va p nat* (*free-l Va t*) *0*]
apply (*smt hd-conv-nth inv-mempool-info-maxsz-align4 le-trans length-greater-0-conv linorder-not-less*)
apply(*subgoal-tac get-bit-s Va* (*pool n1*) (*level n1*) (*block n1*) = *ALLOCATING*)
prefer *2* **apply**(*subgoal-tac allocating-node Va t1* = *Some n1*) **prefer** *2* **apply** *auto*[*1*]
apply *blast*
apply *auto*[*1*]

apply(*subgoal-tac allocating-node Va t1* = *Some n1*)
prefer *2* **apply** *auto*[*1*]
apply(*subgoal-tac allocating-node Va t2* = *Some n2*)
prefer *2* **apply** *auto*[*1*]
apply *auto*[*1*]

apply *clarify*
apply(*case-tac t* = *t1*)
apply(*subgoal-tac get-bit-s Va* (*pool n1*) (*level n1*) (*block n1*) = *FREE*)
prefer *2*
apply(*subgoal-tac pool n1* = *p* ∧ *level n1* = *nat* (*free-l Va t*) ∧ *block n1* = *block-num*
(*set-bit-allocating*
((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p*
(*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))

(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat*
(*free-l Va t*)))
p)
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat* (*free-l*
Va t)))
prefer *2* **apply** *auto*[*1*]
apply(*subgoal-tac block-num*
(*set-bit-allocating*
((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info*
Va p) (*nat* (*free-l Va t*)))) *p*
(*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*)
(*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va*
t)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^
nat (*free-l Va t*)))
p)
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat*
(*free-l Va t*)) = *block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va*
t)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat* (*free-l*
Va t)))
prefer *2* **apply**(*simp add*: *set-bit-def rmhead-free-list-def block-num-def*)
apply(*subgoal-tac block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat*
(*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l*
Va t)))
((*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4*
^ *nat* (*free-l Va t*))) =
block-num (*mem-pool-info Va p*)
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
((*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat*
(*free-l Va t*))))
prefer *2* **apply**(*simp add*:*set-bit-def block-num-def rmhead-free-list-def*)
apply(*simp add*:*level-empty-def head-free-list-def*)
using *inv-bitmap-freelist-fl-FREE*[*of Va p nat* (*free-l Va t*) *0*]
apply (*smt hd-conv-nth inv-mempool-info-maxsz-align4 le-trans length-greater-0-conv*
linorder-not-less)
apply(*subgoal-tac get-bit-s Va* (*pool n2*) (*level n2*) (*block n2*) = *FREEING*)
prefer *2* **apply** *auto*[*1*]
apply *auto*[*1*]

apply(*subgoal-tac allocating-node Va t1* = *Some n1*)
prefer *2* **apply** *auto*[*1*]

apply(*subgoal-tac allocating-node Va t2 = Some n2*)
prefer *2* **apply** *auto*[*1*]
apply *auto*[*1*]
done

lemma *mp-alloc-stm3-lm2-inv-bitmap0*:
inv-mempool-info Va ∧ *inv-bitmap0 Va* ⟹
p ∈ *mem-pools Va* ⟹
inv-bitmap0
(*Va*⦇*blk* := (*blk Va*)(*t* := *head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))),
mem-pool-info :=
set-bit-allocating ((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p* (*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))) (*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *nat* (*free-l Va t*))),
allocating-node := *allocating-node Va*(*t* ↦
⦇*pool* = *p*, *level* = *nat* (*free-l Va t*),
block = *block-num*
(*set-bit-allocating* ((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p* (*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))) (*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *nat* (*free-l Va t*)))
p)
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))) (*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *nat* (*free-l Va t*)),
data = *head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))⦈)⦈)
apply(*simp add*:*set-bit-def*)
apply(*rule subst*[**where** *s*=*inv-bitmap0*
(*Va*⦇*mem-pool-info* := (*mem-pool-info Va*)
(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))
⦇*levels* := *levels* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
[*nat* (*free-l Va t*) :=
(*levels* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))) ! *nat* (*free-l Va t*))
⦇*bits* := *bits* (*levels* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))) !
nat (*free-l Va t*))
[*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *nat* (*free-l Va t*)) :=
ALLOCATING]⦈]⦈)⦈)])
apply(*simp add*:*inv-bitmap0-def*)

apply(*subgoal-tac length* (*levels* (*mem-pool-info Va p*)) > *0*) **prefer** *2*
apply(*simp add*:*inv-def inv-mempool-info-def Let-def*) **apply** *fastforce*

apply(*subgoal-tac* ∀ *i*<*length* (*bits* (*levels* (*mem-pool-info Va p*) ! *0*)).
(*bits* (*levels* (*mem-pool-info Va p*) ! *0*)) ! *i* ≠ *NOEXIST*)
prefer *2* **apply**(*simp add*:*inv-def inv-bitmap0-def*) **apply** *metis*

apply(*case-tac nat* (*free-l Va t*) = *0*)
apply(*simp add*:*inv-bitmap0-def Let-def rmhead-free-list-def block-num-def*)
apply *clarsimp*
apply(*case-tac i* = (*head-free-list* (*mem-pool-info Va p*) *NULL* − *buf* (*mem-pool-info Va p*)) *div*
ALIGN4 (*max-sz* (*mem-pool-info Va p*)))
apply(*subgoal-tac bits* (*levels* (*mem-pool-info Va p*)
[*NULL* := (*levels* (*mem-pool-info Va p*)
[*NULL* := (*levels* (*mem-pool-info Va p*) ! *NULL*)
(|*free-list* := *tl* (*free-list* (*levels* (*mem-pool-info Va p*) ! *NULL*))|)] !
NULL)
(|*bits* := *bits* (*levels* (*mem-pool-info Va p*)
[*NULL* := (*levels* (*mem-pool-info Va p*) ! *NULL*)
(|*free-list* := *tl* (*free-list* (*levels* (*mem-pool-info Va p*) ! *NULL*))|)] !
NULL)
[(*head-free-list* (*mem-pool-info Va p*) *NULL* − *buf* (*mem-pool-info Va p*)) *div*
ALIGN4 (*max-sz* (*mem-pool-info Va p*)) :=
ALLOCATING]|)] ! *NULL*) ! *i* = *ALLOCATING*) **prefer** *2*
apply(*rule subst*[**where** *s*=(*bits* (*levels* (*mem-pool-info Va p*) ! *0*))
[(*head-free-list* (*mem-pool-info Va p*) *NULL* − *buf* (*mem-pool-info Va p*)) *div*
ALIGN4 (*max-sz* (*mem-pool-info Va p*)) := *ALLOCATING*]])
apply *fastforce*
apply *simp*
apply *force*

apply(*subgoal-tac bits* (*levels* (*mem-pool-info Va p*) ! *NULL*)
[(*head-free-list* (*mem-pool-info Va p*) *NULL* − *buf* (*mem-pool-info Va p*)) *div*
ALIGN4 (*max-sz* (*mem-pool-info Va p*)) := *ALLOCATING*] ! *i* ≠ *NOEXIST*) **prefer** *2*
apply *force*
apply *simp*

apply(*simp add*:*inv-bitmap0-def Let-def rmhead-free-list-def block-num-def*)
done

lemma *mp-alloc-stm3-lm2-inv-bitmapn*:
inv-mempool-info Va ∧ *inv-bitmapn Va* ⟹
  p ∈ *mem-pools Va* ⟹
  inv-bitmapn
   (*Va*⦇*blk* := (*blk Va*)(*t* := *head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))),
        mem-pool-info :=
        set-bit-allocating ((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p* (*nat* (*free-l Va t*))
            (*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))) (*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
             (*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat* (*free-l Va t*))),
        allocating-node := *allocating-node Va*(*t* ↦
          ⦇*pool* = *p*, *level* = *nat* (*free-l Va t*),
             block = *block-num*
                         (*set-bit-allocating* ((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p* (*nat* (*free-l Va t*))
                       (*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))) (*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
                         (*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat* (*free-l Va t*)))
                       p)
                  (*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))) (*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat* (*free-l Va t*)),
             data = *head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))⦈)⦈)
apply(*simp add*:*set-bit-def*)
apply(*rule subst*[**where** *s*=*inv-bitmapn*
     (*Va*⦇*mem-pool-info* := (*mem-pool-info Va*)
          (*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))
             ⦇*levels* := *levels* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
               [*nat* (*free-l Va t*) :=
                 (*levels* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))) ! *nat* (*free-l Va t*))
                   ⦇*bits* := *bits* (*levels* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))) !
                               nat (*free-l Va t*))
                    [*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
                      (*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
                     (*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat* (*free-l Va t*)) :=
                       ALLOCATING]⦈]⦈)⦈)])
  apply(*simp add*:*inv-bitmapn-def*)
apply(*subgoal-tac length* (*levels* (*mem-pool-info Va p*)) > *0*) **prefer** *2*
  apply(*simp add*:*inv-def inv-mempool-info-def Let-def*) **apply** *fastforce*

apply(*subgoal-tac* ∀ *i*<*length* (*bits* (*levels* (*mem-pool-info Va p*) ! (*length* (*levels* (*mem-pool-info Va p*)) − *Suc 0*))).
                (*bits* (*levels* (*mem-pool-info Va p*) ! (*length* (*levels* (*mem-pool-info

Va p)) − *Suc 0*))) ! *i* ≠ *DIVIDED*)
prefer *2* **apply**(*simp add*:*inv-def inv-bitmapn-def*) **apply** *metis*

apply(*case-tac nat* (*free-l Va t*) = *length* (*levels* (*mem-pool-info Va p*)) − *Suc 0*)
apply(*simp add*:*inv-bitmapn-def Let-def rmhead-free-list-def block-num-def*)
apply *clarsimp*
apply(*case-tac i* = (*head-free-list* (*mem-pool-info Va p*) (*length* (*levels* (*mem-pool-info Va p*)) − *Suc NULL*) −
buf (*mem-pool-info Va p*)) *div*
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ (*length* (*levels* (*mem-pool-info Va p*)) − *Suc NULL*)))
apply(*subgoal-tac bits* (*levels* (*mem-pool-info Va p*) ! (*length* (*levels* (*mem-pool-info Va p*)) − *Suc NULL*))
[(*head-free-list* (*mem-pool-info Va p*) (*length* (*levels* (*mem-pool-info Va p*)) − *Suc NULL*) −
buf (*mem-pool-info Va p*)) *div*
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ (*length* (*levels* (*mem-pool-info Va p*)) − *Suc NULL*)) :=
ALLOCATING] ! *i* ≠ *DIVIDED*) **prefer** *2*
apply(*rule subst*[**where** *s*=(*bits* (*levels* (*mem-pool-info Va p*) ! *0*))
[(*head-free-list* (*mem-pool-info Va p*) *NULL* − *buf* (*mem-pool-info Va p*)) *div*
ALIGN4 (*max-sz* (*mem-pool-info Va p*)) := *ALLOCATING*]])
apply *fastforce*
apply *simp*
apply *force*

apply(*subgoal-tac bits* (*levels* (*mem-pool-info Va p*) ! *NULL*)
[(*head-free-list* (*mem-pool-info Va p*) *NULL* − *buf* (*mem-pool-info Va p*)) *div*
ALIGN4 (*max-sz* (*mem-pool-info Va p*)) := *ALLOCATING*] ! *i* ≠ *DIVIDED*) **prefer** *2*
apply *force*
apply *simp*

apply(*simp add*:*inv-bitmapn-def Let-def rmhead-free-list-def block-num-def*)
done

lemma *mp-alloc-stm3-lm2-inv-bitmap-not4free*:
inv-mempool-info Va ∧ *inv-bitmap-not4free Va* ⟹
p ∈ *mem-pools Va* ⟹
inv-bitmap-not4free
(*Va*(|*blk* := (*blk Va*)(*t* := *head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))),
mem-pool-info :=
set-bit-allocating ((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p* (*nat* (*free-l Va t*))

(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))) (*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat* (*free-l Va t*))),
allocating-node := *allocating-node Va*(*t* ↦
(|*pool* = *p*, *level* = *nat* (*free-l Va t*),
block = *block-num*
(*set-bit-allocating* ((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p* (*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))) (*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat* (*free-l Va t*)))
p)
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))) (*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat* (*free-l Va t*)),
data = *head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))|)|)|))
apply(*rule subst*[**where** *s*=*inv-bitmap-not4free* (*Va*(|*mem-pool-info* :=
set-bit-allocating ((*mem-pool-info Va*)(*p* :=
rmhead-free-list (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p* (*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))) (*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat* (*free-l Va t*)))|)|)])
apply(*simp add*: *inv-bitmap-not4free-def Let-def partner-bits-def set-bit-def rmhead-free-list-def block-num-def*)

apply(*simp add*: *inv-bitmap-not4free-def Let-def partner-bits-def set-bit-def rmhead-free-list-def block-num-def*)
apply *clarsimp*
apply(*case-tac nat* (*free-l Va t*) = *i*) **prefer** *2* **apply** *auto*[*1*]

apply(*subgoal-tac bits* (*levels* (*mem-pool-info Va p*)
[*nat* (*free-l Va t*) :=
(*levels* (*mem-pool-info Va p*)
[*nat* (*free-l Va t*) := (*levels* (*mem-pool-info Va p*) ! *nat* (*free-l Va t*))
(|*free-list* := *tl* (*free-list* (*levels* (*mem-pool-info Va p*) ! *nat* (*free-l Va t*)))|)] !
nat (*free-l Va t*))
(|*bits* := *bits* (*levels* (*mem-pool-info Va p*)
[*nat* (*free-l Va t*) := (*levels* (*mem-pool-info Va p*) ! *nat* (*free-l Va t*))
(|*free-list* := *tl* (*free-list* (*levels* (*mem-pool-info Va p*) ! *nat* (*free-l Va t*)))|)] !
nat (*free-l Va t*))
[(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)) − *buf* (*mem-pool-info Va p*)) *div*

(*ALIGN4* (*max-sz* (*mem-pool-info* *Va* *p*)) *div* *4* ˆ *nat* (*free-l* *Va* *t*)) :=
ALLOCATING]⦈] !
i) = *bits* (*levels* (*mem-pool-info* *Va* *p*) ! *i*) [(*head-free-list* (*mem-pool-info* *Va* *p*) (*nat* (*free-l* *Va* *t*)) − *buf* (*mem-pool-info* *Va* *p*)) *div*
(*ALIGN4* (*max-sz* (*mem-pool-info* *Va* *p*)) *div* *4* ˆ *nat* (*free-l* *Va* *t*)) :=
ALLOCATING]) **prefer** *2* **apply** *simp*
apply *simp*
apply(*case-tac* (*head-free-list* (*mem-pool-info* *Va* *p*) (*nat* (*free-l* *Va* *t*)) − *buf* (*mem-pool-info* *Va* *p*)) *div*
(*ALIGN4* (*max-sz* (*mem-pool-info* *Va* *p*)) *div* *4* ˆ *nat* (*free-l* *Va* *t*)) = *j* *div* *4* ∗ *4*)
apply *auto*[*1*]
apply(*case-tac* (*head-free-list* (*mem-pool-info* *Va* *p*) (*nat* (*free-l* *Va* *t*)) − *buf* (*mem-pool-info* *Va* *p*)) *div*
(*ALIGN4* (*max-sz* (*mem-pool-info* *Va* *p*)) *div* *4* ˆ *nat* (*free-l* *Va* *t*)) = *Suc* (*j* *div* *4* ∗ *4*))
apply(*subgoal-tac* *Suc* (*j* *div* *4* ∗ *4*) < *length* (*bits* (*levels* (*mem-pool-info* *Va* *p*) ! *i*))) **prefer** *2*
apply (*metis* *list-update-beyond* *not-less*)
apply *auto*[*1*]
apply(*case-tac* (*head-free-list* (*mem-pool-info* *Va* *p*) (*nat* (*free-l* *Va* *t*)) − *buf* (*mem-pool-info* *Va* *p*)) *div*
(*ALIGN4* (*max-sz* (*mem-pool-info* *Va* *p*)) *div* *4* ˆ *nat* (*free-l* *Va* *t*)) = *j* *div* *4* ∗ *4* + *2*)
apply(*subgoal-tac* *j* *div* *4* ∗ *4* + *2* < *length* (*bits* (*levels* (*mem-pool-info* *Va* *p*) ! *i*))) **prefer** *2*
apply (*metis* *list-update-beyond* *not-less*)
apply *auto*[*1*]
apply(*case-tac* (*head-free-list* (*mem-pool-info* *Va* *p*) (*nat* (*free-l* *Va* *t*)) − *buf* (*mem-pool-info* *Va* *p*)) *div*
(*ALIGN4* (*max-sz* (*mem-pool-info* *Va* *p*)) *div* *4* ˆ *nat* (*free-l* *Va* *t*)) = *j* *div* *4* ∗ *4* + *3*)
apply(*subgoal-tac* *j* *div* *4* ∗ *4* + *3* < *length* (*bits* (*levels* (*mem-pool-info* *Va* *p*) ! *i*))) **prefer** *2*
apply (*metis* *list-update-beyond* *not-less*)
apply *auto*[*1*]

apply *simp*
done

lemma *mp-alloc-stm3-lm2-inv-mempool-info*:
inv-mempool-info *Va* ∧
p ∈ *mem-pools* *Va* ⟹
∀ *ii*<*length* (*lsizes* *Va* *t*). *lsizes* *Va* *t* ! *ii* = *ALIGN4* (*max-sz* (*mem-pool-info* *Va* *p*)) *div* *4* ˆ *ii* ⟹
length (*lsizes* *Va* *t*) ≤ *n-levels* (*mem-pool-info* *Va* *p*) ⟹
¬ *free-l* *Va* *t* < *OK* ⟹

nat (*free-l* *Va* *t*) < *length* (*lsizes* *Va* *t*) ⟹
inv-mempool-info
(*Va*⦇*blk* := (*blk* *Va*)(*t* := *head-free-list* (*mem-pool-info* *Va* *p*) (*nat* (*free-l* *Va* *t*))),
mem-pool-info :=
set-bit-allocating ((*mem-pool-info* *Va*)(*p* := *rmhead-free-list* (*mem-pool-info* *Va* *p*) (*nat* (*free-l* *Va* *t*)))) *p* (*nat* (*free-l* *Va* *t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info* *Va* *p*) (*nat* (*free-l* *Va* *t*))) (*head-free-list* (*mem-pool-info* *Va* *p*) (*nat* (*free-l* *Va* *t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info* *Va* *p*)) *div* *4* ^ *nat* (*free-l* *Va* *t*))),
allocating-node := *allocating-node* *Va*(*t* ↦
⦇*pool* = *p*, *level* = *nat* (*free-l* *Va* *t*),
block = *block-num*
(*set-bit-allocating* ((*mem-pool-info* *Va*)(*p* := *rmhead-free-list* (*mem-pool-info* *Va* *p*) (*nat* (*free-l* *Va* *t*)))) *p* (*nat* (*free-l* *Va* *t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info* *Va* *p*) (*nat* (*free-l* *Va* *t*))) (*head-free-list* (*mem-pool-info* *Va* *p*) (*nat* (*free-l* *Va* *t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info* *Va* *p*)) *div* *4* ^ *nat* (*free-l* *Va* *t*)))
p)
(*head-free-list* (*mem-pool-info* *Va* *p*) (*nat* (*free-l* *Va* *t*))) (*ALIGN4* (*max-sz* (*mem-pool-info* *Va* *p*)) *div* *4* ^ *nat* (*free-l* *Va* *t*)),
data = *head-free-list* (*mem-pool-info* *Va* *p*) (*nat* (*free-l* *Va* *t*))⦈)⦈)
apply(*simp add*:*inv-mempool-info-def*)
apply(*simp add*: *rmhead-free-list-def*
head-free-list-def set-bit-def block-num-def)
apply(*rule conjI*) **apply** *metis*
apply(*rule conjI*) **apply** *metis*
apply(*rule conjI*) **apply** *metis*
apply(*rule conjI*) **apply** *metis*
apply *clarsimp* **apply**(*simp add*:*Let-def*)
apply(*case-tac nat* (*free-l* *Va* *t*) = *i*)
apply(*subgoal-tac length* (*bits* (*levels* (*mem-pool-info* *Va* *p*) ! (*nat* (*free-l* *Va* *t*))))
= *n-max* (*mem-pool-info* *Va* *p*) ∗ *4* ^ (*nat* (*free-l* *Va* *t*)))
prefer *2* **apply** *metis*
using *mp-alloc-stm3-lm2-2*[**where** *ii*=*nat* (*free-l* *Va* *t*) **and** *mp*=*mem-pool-info* *Va* *p* **and**
fl=*tl* (*free-list* (*levels* (*mem-pool-info* *Va* *p*) ! *nat* (*free-l* *Va* *t*))) **and**
jj=(*hd* (*free-list* (*levels* (*mem-pool-info* *Va* *p*) ! *nat* (*free-l* *Va* *t*))) − *buf* (*mem-pool-info* *Va* *p*)) *div*
lsizes *Va* *t* ! *nat* (*free-l* *Va* *t*)] **apply** *metis*
apply *simp*
done

lemma *mp-alloc-stm3-lm2-inv-bitmap-freelist*:
¬ *level-empty* (*mem-pool-info* *Va* *p*) (*nat* (*free-l* *Va* *t*)) ⟹
inv-bitmap-freelist *Va* ∧ *inv-mempool-info* *Va* ⟹
p ∈ *mem-pools* *Va* ⟹
∀ *ii*<*length* (*lsizes* *Va* *t*). *lsizes* *Va* *t* ! *ii* = *ALIGN4* (*max-sz* (*mem-pool-info* *Va*

p)) div 4 ^ ii ⟹
length (lsizes Va t) ≤ n-levels (mem-pool-info Va p) ⟹
alloc-l Va t < int (n-levels (mem-pool-info Va p)) ⟹
free-l Va t ≤ alloc-l Va t ⟹
¬ free-l Va t < OK ⟹
length (lsizes Va t) ≤ length (levels (mem-pool-info Va p)) ⟹
nat (free-l Va t) < length (lsizes Va t) ⟹
inv-bitmap-freelist
(Va(|blk := (blk Va)(t := head-free-list (mem-pool-info Va p) (nat (free-l Va t))),
mem-pool-info :=
set-bit-allocating ((mem-pool-info Va)(p := rmhead-free-list (mem-pool-info Va p) (nat (free-l Va t)))) p (nat (free-l Va t))
(block-num (rmhead-free-list (mem-pool-info Va p) (nat (free-l Va t))) (head-free-list (mem-pool-info Va p) (nat (free-l Va t)))
(ALIGN4 (max-sz (mem-pool-info Va p)) div 4 ^ nat (free-l Va t))),
allocating-node := allocating-node Va(t ↦
(|pool = p, level = nat (free-l Va t),
block = block-num
(set-bit-allocating ((mem-pool-info Va)(p := rmhead-free-list (mem-pool-info Va p) (nat (free-l Va t)))) p (nat (free-l Va t))
(block-num (rmhead-free-list (mem-pool-info Va p) (nat (free-l Va t))) (head-free-list (mem-pool-info Va p) (nat (free-l Va t)))
(ALIGN4 (max-sz (mem-pool-info Va p)) div 4 ^ nat (free-l Va t)))
p)
(head-free-list (mem-pool-info Va p) (nat (free-l Va t))) (ALIGN4 (max-sz (mem-pool-info Va p)) div 4 ^ nat (free-l Va t)),
data = head-free-list (mem-pool-info Va p) (nat (free-l Va t))|))|))

apply(*simp add:inv-bitmap-freelist-def*)
apply *clarify*
apply(*case-tac pa ≠ p*) **apply**(*simp add:Let-def*)
using *mp-alloc-stm3-body-meminfo* **apply** *smt*
apply(*simp add:Let-def*)
apply(*rule subst*[**where** *t=length (levels (set-bit-allocating ((mem-pool-info Va)(p := rmhead-free-list (mem-pool-info Va p) (nat (free-l Va t)))) p
(nat (free-l Va t))
(block-num (rmhead-free-list (mem-pool-info Va p) (nat (free-l Va t)))
(head-free-list (mem-pool-info Va p) (nat (free-l Va t))) (lsizes Va t ! nat (free-l Va t)))
p))* **and** *s=length (levels (mem-pool-info Va p))*])
using *mp-alloc-stm3-body-len-lvls* **apply** *metis*
apply(*rule subst*[**where** *t=buf (set-bit-allocating ((mem-pool-info Va)(p := rmhead-free-list (mem-pool-info Va p) (nat (free-l Va t)))) p
(nat (free-l Va t))
(block-num (rmhead-free-list (mem-pool-info Va p) (nat (free-l Va t)))
(head-free-list (mem-pool-info Va p) (nat (free-l Va t))) (lsizes Va t ! nat (free-l Va t)))*

p) **and** *s=buf* (*mem-pool-info Va p*)])
using *mp-alloc-stm3-body-minf-buf* **apply** *metis*
apply(*rule subst*[**where** *t=n-max* (*set-bit-allocating* ((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p*
(*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))) (*lsizes Va t* ! *nat* (*free-l Va t*)))
p) **and** *s=n-max* (*mem-pool-info Va p*)])
using *mp-alloc-stm3-body-minf-nmax* **apply** *metis*
apply(*rule subst*[**where** *t=max-sz* (*set-bit-allocating* ((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p*
(*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))) (*lsizes Va t* ! *nat* (*free-l Va t*)))
p) **and** *s=max-sz* (*mem-pool-info Va p*)])
using *mp-alloc-stm3-body-minf-maxsz* **apply** *metis*

apply *clarify* **apply**(*rename-tac pa ii*)
apply(*subgoal-tac length* (*bits* (*levels* (*set-bit-allocating*
((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p*
(*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*lsizes Va t* ! *nat* (*free-l Va t*)))
p) ! *ii*))=*length* (*bits* ((*levels* (*mem-pool-info Va p*))!*ii*)))
prefer *2* **using** *mp-alloc-stm3-body-len-bits* **apply** *metis*

apply(*rule conjI*)
apply *clarify* **apply**(*rule iffI*) **apply**(*rename-tac pa ii jj*)

apply(*case-tac nat* (*free-l Va t*) = *ii*)

apply(*case-tac jj* = (*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))) (*lsizes Va t* ! *nat* (*free-l Va t*))))

apply(*subgoal-tac get-bit* (*set-bit-allocating* ((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p* (*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))) (*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*lsizes Va t* ! *nat* (*free-l Va t*))))
p ii jj = *ALLOCATING*)
prefer *2* **apply**(*simp add*: *set-bit-def rmhead-free-list-def*)
apply (*metis BlockState.distinct*(*17*))

apply(*subgoal-tac get-bit* (*mem-pool-info Va*) *p ii jj* = *FREE*)
prefer *2* **apply**(*simp add*: *set-bit-def rmhead-free-list-def*)
apply(*subgoal-tac buf* (*mem-pool-info Va p*) + *jj* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ˆ *ii*)
∈ *set* (*free-list* (*levels* (*mem-pool-info Va p*) ! *ii*)))
prefer *2* **apply** (*metis mp-alloc-stm3-body-len-lvls*)
apply(*subgoal-tac buf* (*mem-pool-info Va p*) + *jj* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ˆ *ii*)
≠ *head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
prefer *2* **apply**(*simp add*:*block-num-def*) **using** *mp-alloc-stm3-lm2-inv-1*
apply *simp*
apply(*simp add*: *set-bit-def rmhead-free-list-def head-free-list-def*)
using *list-nhd-in-tl-set* **apply** *metis*

apply(*subgoal-tac get-bit* (*mem-pool-info Va*) *p ii jj* = *FREE*)
prefer *2* **apply**(*simp add*: *set-bit-def rmhead-free-list-def*)
apply(*subgoal-tac buf* (*mem-pool-info Va p*) + *jj* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ˆ *ii*)
∈ *set* (*free-list* (*levels* (*mem-pool-info Va p*) ! *ii*)))
prefer *2* **apply** (*metis mp-alloc-stm3-body-len-lvls*)
apply(*simp add*: *set-bit-def rmhead-free-list-def head-free-list-def*)

apply(*rename-tac pa ii jj*)
apply(*subgoal-tac length* (*levels* (*set-bit-allocating* ((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p*
(*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))) (*lsizes Va t* ! *nat* (*free-l Va t*)))
p)) = *length* (*levels* (*mem-pool-info Va p*)))
prefer *2* **using** *mp-alloc-stm3-body-len-lvls* **apply** *metis*
apply(*case-tac nat* (*free-l Va t*) = *ii*)

apply(*subgoal-tac buf* (*mem-pool-info Va p*) + *jj* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ˆ *ii*)
∈ *set* (*tl* (*free-list* (*levels* (*mem-pool-info Va p*) ! *ii*))))
prefer *2* **using** *mp-alloc-stm3-body-minf-buf mp-alloc-stm3-body-minf-maxsz mp-alloc-stm3-body-frlst-samelvl* **apply** *metis*
apply(*subgoal-tac buf* (*mem-pool-info Va p*) + *jj* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ˆ *ii*)
∈ *set* (*free-list* (*levels* (*mem-pool-info Va p*) ! *ii*)))
prefer *2* **apply**(*metis list.set-sel*(*2*) *tl-Nil*)
apply(*subgoal-tac get-bit* (*mem-pool-info Va*) *p ii jj* = *FREE*)
prefer *2* **apply** *metis*
apply(*subgoal-tac block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l*

Va t)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat* (*free-l Va t*)) $\neq$ *jj*)
prefer *2*
apply(*subgoal-tac buf* (*mem-pool-info Va p*) + *jj* $*$ (*max-sz* (*mem-pool-info Va p*) *div 4* ^ *ii*)
$\neq$ *head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
prefer *2* **apply**(*subgoal-tac distinct* (*free-list* (*levels* (*mem-pool-info Va p*) ! *ii*)))
prefer *2* **apply** *metis*
apply(*simp add*:*head-free-list-def*)
using *dist-hd-nin-tl* **apply** (*metis* (*mono-tags*, *hide-lams*) *le-eq-less-or-eq le-trans linorder-not-less*)
apply(*simp add*:*block-num-def*)
apply(*subgoal-tac buf* (*rmhead-free-list* (*mem-pool-info Va p*) *ii*) = *buf* (*mem-pool-info Va p*))
prefer *2* **apply**(*simp add*:*rmhead-free-list-def*)
apply(*subgoal-tac* $\exists$ *n*. *head-free-list* (*mem-pool-info Va p*) *ii* =
buf (*mem-pool-info Va p*) + *n* $*$ (*max-sz* (*mem-pool-info Va p*) *div 4* ^ *ii*))
prefer *2* **apply**(*simp add*:*head-free-list-def level-empty-def*)
apply (*smt add-lessD1 hd-conv-nth le-eq-less-or-eq length-greater-0-conv less-imp-add-positive*)
using *mp-alloc-stm3-lm2-inv-2* **apply** (*metis inv-mempool-info-maxsz-align4*)
apply(*simp add*: *set-bit-def rmhead-free-list-def head-free-list-def*)

apply(*subgoal-tac buf* (*mem-pool-info Va p*) + *jj* $*$ (*max-sz* (*mem-pool-info Va p*) *div 4* ^ *ii*)
$\in$ *set* (*free-list* (*levels* (*mem-pool-info Va p*) ! *ii*)))
prefer *2* **apply** (*metis mp-alloc-stm3-body-frlst-otherlvl mp-alloc-stm3-body-minf-buf mp-alloc-stm3-body-minf-maxsz*)
apply(*subgoal-tac get-bit* (*mem-pool-info Va*) *p ii jj* = *FREE*)
prefer *2* **apply**(*simp add*: *set-bit-def rmhead-free-list-def*)
apply(*simp add*: *set-bit-def rmhead-free-list-def head-free-list-def*)

apply(*rule conjI*)

apply *clarify*
apply(*rename-tac pa ii jj*)
apply(*subgoal-tac length* (*levels* (*set-bit-allocating* ((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p*
(*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))) (*lsizes Va t* ! *nat* (*free-l Va t*)))
p)) = *length* (*levels* (*mem-pool-info Va p*)))

prefer *2* **using** *mp-alloc-stm3-body-len-lvls* **apply** *metis*
apply(*case-tac nat* (*free-l Va t*) = *ii*)

apply(*subgoal-tac* (*free-list*
(*levels* (*set-bit-allocating* ((*mem-pool-info Va*)(*p* := *rmhead-free-list*
(*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p*
(*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l*
Va t)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *nat* (*free-l*
Va t)))
p) ! *ii*)) = (*tl* (*free-list* (*levels* (*mem-pool-info Va p*) ! *ii*))))
prefer *2* **apply**(*simp add*:*level-empty-def set-bit-def rmhead-free-list-def head-free-list-def*)

apply(*subgoal-tac tl* (*free-list* (*levels* (*mem-pool-info Va p*) ! *ii*)) ! *jj* = (*free-list*
(*levels* (*mem-pool-info Va p*) ! *ii*)) ! *Suc jj*)
prefer *2* **apply**(*rule List.nth-tl*)
apply(*subgoal-tac length* (*tl* (*free-list* (*levels* (*mem-pool-info Va p*) ! *ii*))) =
length (*free-list*
(*levels* (*set-bit-allocating* ((*mem-pool-info Va*)(*p* := *rmhead-free-list*
(*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p*
(*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l*
Va t)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *nat* (*free-l*
Va t)))
p) ! *ii*)))
prefer *2* **apply** *simp*
apply *metis*
apply(*subgoal-tac* (∃ *n*. *n* < *n-max* (*mem-pool-info Va p*) ∗ (*4* ˆ *ii*) ∧ (*free-list*
(*levels* (*mem-pool-info Va p*) ! *ii*)) ! *Suc jj* =
buf (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4*
ˆ *ii*)))

using *mp-alloc-stm3-body-minf-buf mp-alloc-stm3-body-minf-nmax mp-alloc-stm3-body-minf-maxsz*
apply *metis*
apply(*subgoal-tac Suc jj* < *length* (*free-list* (*levels* (*mem-pool-info Va p*) !
ii)))
prefer *2* **apply**(*subgoal-tac jj* < *length* (*tl* (*free-list* (*levels* (*mem-pool-info*
Va p) ! *ii*))))
prefer *2* **apply** *metis*
apply(*simp add*:*level-empty-def*)
apply *metis*

using *mp-alloc-stm3-body-minf-buf mp-alloc-stm3-body-minf-maxsz mp-alloc-stm3-body-minf-nmax* *mp-alloc-stm3-body-frlst-otherlvl* **apply** *metis*

apply(*case-tac nat* (*free-l Va t*) = *ii*)
apply(*subgoal-tac* (*free-list*
(*levels* (*set-bit-allocating* ((*mem-pool-info Va*)(*p* := *rmhead-free-list*
(*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p*
(*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l*
Va t)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat* (*free-l*
Va t)))
p) ! *ii*)) = (*tl* (*free-list* (*levels* (*mem-pool-info Va p*) ! *ii*))))
prefer *2* **apply**(*simp add*:*level-empty-def set-bit-def rmhead-free-list-def head-free-list-def*)
apply(*subgoal-tac distinct* (*free-list* (*levels* (*mem-pool-info Va p*) ! *ii*)))
prefer *2* **apply** *simp*
using *distinct-tl* **apply** *metis*

apply(*subgoal-tac distinct* (*free-list* (*levels* (*mem-pool-info Va p*) ! *ii*)))
prefer *2* **apply** (*metis mp-alloc-stm3-body-len-lvls*)
using *mp-alloc-stm3-body-frlst-otherlvl* **apply** *metis*
done

lemma *mp-alloc-stm3-lm2-inv-bitmap*:
¬ *level-empty* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)) ⟹
inv-mempool-info Va ∧ *inv-bitmap-freelist Va* ∧ *inv-bitmap Va* ⟹
p ∈ *mem-pools Va* ⟹
length (*lsizes Va t*) ≤ *n-levels* (*mem-pool-info Va p*) ⟹
alloc-l Va t < *int* (*n-levels* (*mem-pool-info Va p*)) ⟹
free-l Va t ≤ *alloc-l Va t* ⟹
¬ *free-l Va t* < *OK* ⟹
length (*lsizes Va t*) ≤ *length* (*levels* (*mem-pool-info Va p*)) ⟹
nat (*free-l Va t*) < *length* (*lsizes Va t*) ⟹
inv-bitmap
(*Va*(|*blk* := (*blk Va*)(*t* := *head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))),
mem-pool-info :=
set-bit-allocating ((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info*
Va p) (*nat* (*free-l Va t*)))) *p* (*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat* (*free-l Va t*))),
allocating-node := *allocating-node Va*(*t* ↦
(|*pool* = *p*, *level* = *nat* (*free-l Va t*),
block = *block-num*
(*set-bit-allocating* ((*mem-pool-info Va*)(*p* := *rmhead-free-list*
(*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p* (*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l*
Va t))) (*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))

(*ALIGN4* (*max-sz* (*mem-pool-info* *Va* *p*)) *div* *4* ^ *nat* (*free-l* *Va* *t*)))
p)
(*head-free-list* (*mem-pool-info* *Va* *p*) (*nat* (*free-l* *Va* *t*))) (*ALIGN4* (*max-sz* (*mem-pool-info* *Va* *p*)) *div* *4* ^ *nat* (*free-l* *Va* *t*)),
data = *head-free-list* (*mem-pool-info* *Va* *p*) (*nat* (*free-l* *Va* *t*))⦈)⦈)
apply(*subgoal-tac* *inv-bitmap* (*set-bit-s* *Va* *p* (*nat* (*free-l* *Va* *t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info* *Va* *p*) (*nat* (*free-l* *Va* *t*))) (*head-free-list* (*mem-pool-info* *Va* *p*) (*nat* (*free-l* *Va* *t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info* *Va* *p*)) *div* *4* ^ *nat* (*free-l* *Va* *t*)))
ALLOCATING))
prefer *2*
apply(*subgoal-tac* *get-bit-s* *Va* *p* (*nat* (*free-l* *Va* *t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info* *Va* *p*) (*nat* (*free-l* *Va* *t*)))
(*head-free-list* (*mem-pool-info* *Va* *p*) (*nat* (*free-l* *Va* *t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info* *Va* *p*)) *div* *4* ^ *nat* (*free-l* *Va* *t*))) = *FREE*)
prefer *2* **apply**(*simp* *add*:*level-empty-def*)
apply(*subgoal-tac* (*block-num* (*mem-pool-info* *Va* *p*) (*free-list* (*levels* (*mem-pool-info* *Va* *p*) ! *nat* (*free-l* *Va* *t*)) ! *NULL*)
(*max-sz* (*mem-pool-info* *Va* *p*) *div* *4* ^ *nat* (*free-l* *Va* *t*)))
= (*block-num* (*rmhead-free-list* (*mem-pool-info* *Va* *p*) (*nat* (*free-l* *Va* *t*)))
(*head-free-list* (*mem-pool-info* *Va* *p*) (*nat* (*free-l* *Va* *t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info* *Va* *p*)) *div* *4* ^ *nat* (*free-l* *Va* *t*))))
using *inv-bitmap-freelist-fl-FREE*[*of* *Va* *p* *nat* (*free-l* *Va* *t*) *0*] **apply** *simp*
apply(*simp* *add*:*block-num-def* *rmhead-free-list-def* *head-free-list-def*)
apply (*simp* *add*: *hd-conv-nth* *inv-mempool-info-maxsz-align4*)
using *inv-bitmap-presv-setbit*[*of* *Va* *p* (*nat* (*free-l* *Va* *t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info* *Va* *p*) (*nat* (*free-l* *Va* *t*)))
(*head-free-list* (*mem-pool-info* *Va* *p*) (*nat* (*free-l* *Va* *t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info* *Va* *p*)) *div* *4* ^ *nat* (*free-l* *Va* *t*))) *ALLOCATING* *set-bit-s* *Va* *p* (*nat* (*free-l* *Va* *t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info* *Va* *p*) (*nat* (*free-l* *Va* *t*))) (*head-free-list* (*mem-pool-info* *Va* *p*) (*nat* (*free-l* *Va* *t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info* *Va* *p*)) *div* *4* ^ *nat* (*free-l* *Va* *t*)))
ALLOCATING] **apply** *simp*

apply(*rule* *inv-bitmap-presv-mpls-mpi2*[*of* (*set-bit-s* *Va* *p* (*nat* (*free-l* *Va* *t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info* *Va* *p*) (*nat* (*free-l* *Va* *t*))) (*head-free-list* (*mem-pool-info* *Va* *p*) (*nat* (*free-l* *Va* *t*)))
(*ALIGN4* (*max-sz* (*mem-pool-info* *Va* *p*)) *div* *4* ^ *nat* (*free-l* *Va* *t*)))
ALLOCATING) (*Va*⦇*blk* := (*blk* *Va*)(*t* := *head-free-list* (*mem-pool-info* *Va* *p*) (*nat* (*free-l* *Va* *t*))),
mem-pool-info :=
set-bit-allocating ((*mem-pool-info* *Va*)(*p* := *rmhead-free-list* (*mem-pool-info* *Va* *p*) (*nat* (*free-l* *Va* *t*)))) *p* (*nat* (*free-l* *Va* *t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info* *Va* *p*) (*nat* (*free-l* *Va* *t*)))
(*head-free-list* (*mem-pool-info* *Va* *p*) (*nat* (*free-l* *Va* *t*)))

(ALIGN4 (max-sz (mem-pool-info Va p)) div 4 ^ nat (free-l Va t))),
allocating-node := allocating-node Va(t ↦
(|pool = p, level = nat (free-l Va t),
block = block-num
(set-bit-allocating ((mem-pool-info Va)(p := rmhead-free-list (mem-pool-info Va p) (nat (free-l Va t)))) p
(nat (free-l Va t))
(block-num (rmhead-free-list (mem-pool-info Va p) (nat (free-l Va t)))
(head-free-list (mem-pool-info Va p) (nat (free-l Va t)))
(ALIGN4 (max-sz (mem-pool-info Va p)) div 4 ^ nat (free-l Va t)))
p)
(head-free-list (mem-pool-info Va p) (nat (free-l Va t)))
(ALIGN4 (max-sz (mem-pool-info Va p)) div 4 ^ nat (free-l Va t)),
data = head-free-list (mem-pool-info Va p) (nat (free-l Va t))|)|)|)])
apply(*simp add: set-bit-s-def set-bit-def block-num-def rmhead-free-list-def head-free-list-def*)
apply(*simp add: set-bit-s-def set-bit-def block-num-def rmhead-free-list-def head-free-list-def*)
apply *clarsimp* **apply**(*simp add: set-bit-s-def set-bit-def block-num-def rmhead-free-list-def head-free-list-def*)
apply (*smt Mem-pool-lvl.simps(1) Mem-pool-lvl.simps(4) Mem-pool-lvl.surjective Mem-pool-lvl.update-convs(2)*
linorder-not-less list-update-beyond nth-list-update-eq nth-list-update-neq)
by *simp*

lemma *mp-alloc-stm3-lm2-inv*:
(*∗NULL < head-free-list (mem-pool-info Va p) (nat (free-l Va t)) ⟹∗*)
¬ level-empty (mem-pool-info Va p) (nat (free-l Va t)) ⟹
inv Va ⟹
allocating-node Va t = None ⟹
freeing-node Va t = None ⟹
p ∈ mem-pools Va ⟹
ETIMEOUT ≤ timeout ⟹
timeout = ETIMEOUT ⟶ tmout Va t = ETIMEOUT ⟹
¬ rf Va t ⟹
∀ ii<length (lsizes Va t). lsizes Va t ! ii = ALIGN4 (max-sz (mem-pool-info Va p)) div 4 ^ ii ⟹
length (lsizes Va t) ≤ n-levels (mem-pool-info Va p) ⟹
alloc-l Va t < int (n-levels (mem-pool-info Va p)) ⟹
free-l Va t ≤ alloc-l Va t ⟹
¬ free-l Va t < 0 ⟹
alloc-l Va t = int (length (lsizes Va t)) − 1 ∧ length (lsizes Va t) = n-levels (mem-pool-info Va p) ∨
alloc-l Va t = int (length (lsizes Va t)) − 2 ∧ lsizes Va t ! nat (alloc-l Va t + 1) < sz ⟹
inv (Va(|blk := (blk Va)(t := head-free-list (mem-pool-info Va p) (nat (free-l Va t))),

mem-pool-info :=
set-bit-allocating ((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p*
(*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))) (*lsizes Va t* ! *nat* (*free-l Va t*))),
allocating-node := *allocating-node Va*(*t* ↦
(|*pool* = *p*, *level* = *nat* (*free-l Va t*),
block = *block-num*
(*set-bit-allocating* ((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p*
(*nat* (*free-l Va t*))
(*block-num* (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))) (*lsizes Va t* ! *nat* (*free-l Va t*)))
p)
(*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))) (*lsizes Va t* ! *nat* (*free-l Va t*)),
data = *head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))|))|))
apply(*subgoal-tac nat* (*free-l Va t*) < *length* (*levels* (*mem-pool-info Va p*)))
prefer *2* **apply**(*simp add*:*inv-def inv-mempool-info-def Let-def*)
apply(*subgoal-tac length* (*lsizes Va t*) ≤ *length* (*levels* (*mem-pool-info Va p*)))
prefer *2* **apply**(*simp add*:*inv-def inv-mempool-info-def Let-def*)
apply(*subgoal-tac nat* (*free-l Va t*) < *length* (*lsizes Va t*))
prefer *2* **apply** *linarith*
apply(*simp add*:*inv-def*)
apply(*rule conjI*)
apply(*simp add*:*inv-cur-def*)
apply(*rule conjI*)
using *mp-alloc-stm3-lm2-inv-thd-waitq* **apply** *fast*
apply(*rule conjI*)
using *mp-alloc-stm3-lm2-inv-mempool-info* **apply** *fast*
apply(*rule conjI*)
using *mp-alloc-stm3-lm2-inv-bitmap-freelist* **apply** *fast*
apply(*rule conjI*) **using** *mp-alloc-stm3-lm2-inv-bitmap* **apply** *simp*
apply(*rule conjI*) **using** *mp-alloc-stm3-lm2-inv-aux-vars* **apply** *simp*
apply(*rule conjI*) **using** *mp-alloc-stm3-lm2-inv-bitmap0* **apply** *simp*
apply(*rule conjI*) **using** *mp-alloc-stm3-lm2-inv-bitmapn* **apply** *simp*
using *mp-alloc-stm3-lm2-inv-bitmap-not4free* **apply** *simp*
done

lemma *mp-alloc-stm3-lm2-3-1*:
(*a*::*nat*) *mod b* = *0* ⟹ *c* ∗ *b* ∗ (*a div b*) = *c* ∗ *a* **by** *auto*

lemma *mp-alloc-stm3-lm2-3*:

¬ *level-empty* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)) ⟹
inv Va ⟹
alloc-l Va t < *int* (*n-levels* (*mem-pool-info Va p*)) ⟹
free-l Va t ≤ *alloc-l Va t* ⟹
p ∈ *mem-pools Va* ⟹
¬ *free-l Va t* < *0* ⟹
max-sz (*mem-pool-info Va p*) = *ALIGN4* (*max-sz* (*mem-pool-info Va p*)) ⟹
let fl = *hd* (*free-list* (*levels* (*mem-pool-info Va p*) ! *nat* (*free-l Va t*))); *mp* = *mem-pool-info Va p*
in ∃ *n*<*n-max mp* ∗ *4* ^ *nat* (*free-l Va t*). *fl* = *buf mp* + *n* ∗ (*max-sz mp div 4* ^ *nat* (*free-l Va t*)) ⟹
hd (*free-list* (*levels* (*mem-pool-info Va p*) ! *nat* (*free-l Va t*)))
< *buf* (*mem-pool-info Va p*) + *n-max* (*mem-pool-info Va p*) ∗ *max-sz* (*mem-pool-info Va p*)
apply(*subgoal-tac nat* (*free-l Va t*) < *length* (*levels* (*mem-pool-info Va p*)))
prefer *2* **apply**(*simp add:inv-def inv-mempool-info-def Let-def*)
apply(*subgoal-tac hd* (*free-list* (*levels* (*mem-pool-info Va p*) ! *nat* (*free-l Va t*))) ≥ *buf* (*mem-pool-info Va p*))
prefer *2* **apply**(*simp add*: *inv-def*) **using** *inv-buf-le-fl*[*of Va p nat* (*free-l Va t*) *0*]
apply (*simp add*: *hd-conv-nth level-empty-def*)
apply (*simp add*: *hd-conv-nth level-empty-def Let-def*)
apply *clarify*

apply(*subgoal-tac max-sz* (*mem-pool-info Va p*) *mod* (*4* ^ *nat* (*free-l Va t*)) = *0*)
prefer *2* **apply** (*metis ge-pow-mod-0 inv-mempool-info-def inv-def*)
apply(*subgoal-tac n* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ^ *nat* (*free-l Va t*))
< *n-max* (*mem-pool-info Va p*) ∗ *max-sz* (*mem-pool-info Va p*))
prefer *2* **apply**(*subgoal-tac n-max* (*mem-pool-info Va p*) ∗ *4* ^ *nat* (*free-l Va t*) ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ^ *nat* (*free-l Va t*))
= *n-max* (*mem-pool-info Va p*) ∗ *max-sz* (*mem-pool-info Va p*))
prefer *2*
using *mp-alloc-stm3-lm2-3-1*[*of max-sz* (*mem-pool-info Va p*) *4* ^ *nat* (*free-l Va t*) *n-max* (*mem-pool-info Va p*)] **apply** *auto*[*1*]
apply(*subgoal-tac n* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ^ *nat* (*free-l Va t*))
< (*n-max* (*mem-pool-info Va p*) ∗ *4* ^ *nat* (*free-l Va t*)) ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ^ *nat* (*free-l Va t*)))
prefer *2* **apply** (*metis inv-mempool-info-def inv-def mp-alloc-stm3-lm2-inv-1-2 mult-less-mono1*)
apply *linarith*

apply *simp*
done

lemma *mp-alloc-stm3-lm2-5*:
¬ *level-empty* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)) ⟹
inv Va ⟹
alloc-l Va t < *int* (*n-levels* (*mem-pool-info Va p*)) ⟹

free-l Va t ≤ *alloc-l Va t* ⟹
p ∈ *mem-pools Va* ⟹
¬ *free-l Va t* < *0* ⟹
(*hd* (*free-list* (*levels* (*mem-pool-info Va p*) ! *nat* (*free-l Va t*))) − *buf* (*mem-pool-info Va p*)) *div*
(*max-sz* (*mem-pool-info Va p*) *div 4* ˆ *nat* (*free-l Va t*))
< *n-max* (*mem-pool-info Va p*) ∗ *4* ˆ *nat* (*free-l Va t*)
apply(*subgoal-tac nat* (*free-l Va t*) < *length* (*levels* (*mem-pool-info Va p*)))
prefer *2* **apply**(*simp add:inv-def inv-mempool-info-def Let-def*)
apply(*subgoal-tac hd* (*free-list* (*levels* (*mem-pool-info Va p*) ! *nat* (*free-l Va t*))) ≥ *buf* (*mem-pool-info Va p*))
prefer *2* **apply**(*simp add*: *inv-def*) **using** *inv-buf-le-fl*[*of Va p nat* (*free-l Va t*) *0*]
apply (*simp add*: *hd-conv-nth level-empty-def*)
apply (*simp add*: *hd-conv-nth level-empty-def*)
by (*metis block-num-def inv-bitmap-freelist-fl-bnum-in inv-mempool-info-def length-greater-0-conv inv-def*)

lemma *mp-alloc-stm3-lm2-4*:
inv Va ∧
p ∈ *mem-pools Va* ∧
free-list (*levels* (*mem-pool-info Va p*) ! *nat* (*free-l Va t*)) ≠ [] ⟹
nat (*free-l Va t*) < *length* (*levels* (*mem-pool-info Va p*)) ⟹ *NULL* < *hd* (*free-list* (*levels* (*mem-pool-info Va p*) ! *nat* (*free-l Va t*)))
using *inv-imp-fl-lt0* **apply**(*simp add:Let-def*)
by (*simp add*: *hd-conv-nth*)

lemma *mp-alloc-stm3-lm2*:
Va ∈ *mp-alloc-precond1-70-2-2 t p sz timeout* ∩ ⦃´*cur* = *Some t*⦄ ⟹
(∗*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)) ≠ *NULL* ⟹∗)
¬ *level-empty* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)) ⟹
(∗{ *Va*⦇*blk* := (*blk Va*)(*t* := *NULL*)⦈, *Va* ⦇*blk* := (*blk Va*)(*t* := *head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))),
mem-pool-info := (*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))⦈} ∩
⦃*NULL* < ´*blk t*⦄ =
{ *Va*⦇*blk* := (*blk Va*)(*t* := *head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))),
mem-pool-info := (*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))⦈} ⟹∗)
⦃*let vb* = *Va*⦇*blk* := (*blk Va*)(*t* := *head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))),
mem-pool-info := (*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))⦈
in ´(=) (*vb*⦇*mem-pool-info* :=
set-bit-allocating (*mem-pool-info vb*) *p* (*nat* (*free-l vb t*))
(*block-num* (*mem-pool-info vb p*) (*blk vb t*) (*lsizes vb t* ! *nat* (*free-l vb t*)))⦈) ∧

$\neg$ *level-empty* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))⦄
$\subseteq$ ⦃´(*allocating-node-update*
(λ-. ´*allocating-node*(*t* $\mapsto$
⦇*pool* = *p*, *level* = *nat* (´*free-l t*), *block* = *block-num* (´*mem-pool-info p*) (´*blk t*) (´*lsizes t* ! *nat* (´*free-l t*)),
data = ´*blk t*⦈)))
$\in$ ⦃´(*Pair Va*) $\in$ *Mem-pool-alloc-guar t*⦄ $\cap$ *mp-alloc-precond2-1 t p sz timeout*⦄
apply(*subgoal-tac head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)) $\neq$ *NULL*)
prefer *2*
apply(*subgoal-tac* (*nat* (*free-l Va t*)) < *length* (*levels* (*mem-pool-info Va p*)))
prefer *2* **apply**(*simp add*:*inv-def inv-mempool-info-def Let-def*) **apply** *force*
apply(*simp add*:*head-free-list-def level-empty-def*) **using** *mp-alloc-stm3-lm2-4*
apply *simp*

apply *clarsimp*
apply(*rule conjI*)
apply(*simp add*:*Mem-pool-alloc-guar-def gvars-conf-stable-def gvars-conf-def*)
apply(*rule disjI1*)
apply(*rule conjI*)
apply *clarify*
apply(*rule conjI*) **apply**(*simp add*: *rmhead-free-list-def head-free-list-def set-bit-def*)
apply(*rule conjI*) **apply**(*simp add*: *rmhead-free-list-def head-free-list-def set-bit-def*)
apply(*rule conjI*) **apply**(*simp add*: *rmhead-free-list-def head-free-list-def set-bit-def*)
apply(*rule conjI*) **apply**(*simp add*: *rmhead-free-list-def head-free-list-def set-bit-def*)
apply(*rule conjI*) **apply**(*simp add*: *rmhead-free-list-def head-free-list-def set-bit-def*)
apply *clarify* **apply**(*simp add*: *rmhead-free-list-def head-free-list-def set-bit-def block-num-def*)
apply(*case-tac nat* (*free-l Va t*) = *i*)
using *mp-alloc-stm3-lm2-1*[*of mem-pool-info Va p nat* (*free-l Va t*)
tl (*free-list* (*levels* (*mem-pool-info Va p*) ! *nat* (*free-l Va t*)))
(*hd* (*free-list* (*levels* (*mem-pool-info Va p*) ! *nat* (*free-l Va t*))) − *buf* (*mem-pool-info Va p*)) *div*
lsizes Va t ! *nat* (*free-l Va t*)] **apply** *meson*
apply *simp*
apply(*rule conjI*)
using *mp-alloc-stm3-lm2-inv* **apply** *simp*

apply *clarsimp* **apply**(*simp add*:*lvars-nochange-def*)

apply(*rule conjI*)
using *mp-alloc-stm3-lm2-inv* **apply** *simp*

apply(*rule conjI*) **apply** *clarsimp* **apply**(*simp add*: *rmhead-free-list-def head-free-list-def set-bit-def*)
apply(*rule conjI*) **apply**(*simp add*: *rmhead-free-list-def head-free-list-def set-bit-def*)
apply(*rule conjI*) **apply**(*simp add*: *rmhead-free-list-def head-free-list-def set-bit-def*)
apply(*rule conjI*) **apply**(*simp add*: *rmhead-free-list-def head-free-list-def set-bit-def*)

apply(*simp add*:*block-num-def*)
apply(*rule subst*[**where** *t*=*buf* (*set-bit-allocating* ((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p* (*nat* (*free-l Va t*))
((*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)) −
buf (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *div*
lsizes Va t ! *nat* (*free-l Va t*))
p) **and** *s*=*buf* (*mem-pool-info Va p*)])
apply(*simp add*:*head-free-list-def rmhead-free-list-def set-bit-def*)
apply(*rule subst*[**where** *t*=*max-sz* (*set-bit-allocating* ((*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *p*
(*nat* (*free-l Va t*))
((*head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)) −
buf (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *div*
lsizes Va t ! *nat* (*free-l Va t*))
p) **and** *s*=*max-sz* (*mem-pool-info Va p*)])
apply(*simp add*:*head-free-list-def rmhead-free-list-def set-bit-def*)
apply(*simp add*:*head-free-list-def*)
apply(*subgoal-tac lsizes Va t* ! *nat* (*free-l Va t*) = *ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ (*nat* (*free-l Va t*)))
prefer *2* **apply** *auto*[*1*]
apply(*subgoal-tac max-sz* (*mem-pool-info Va p*) = *ALIGN4* (*max-sz* (*mem-pool-info Va p*)))
prefer *2* **apply**(*simp add*:*inv-def inv-mempool-info-def Let-def*)
apply (*metis align40 mod-mult-self1-is-0 semiring-normalization-rules*(*17*))
apply(*subgoal-tac let fl* = *hd* (*free-list* (*levels* (*mem-pool-info Va p*) ! *nat* (*free-l Va t*)));
mp = (*mem-pool-info Va p*) *in*
(∃ *n*. *n* < *n-max mp* ∗ (*4* ˆ (*nat* (*free-l Va t*)))
∧ *fl* = *buf mp* + *n* ∗ (*max-sz mp div* (*4* ˆ (*nat* (*free-l Va t*))))))
prefer *2* **apply**(*simp add*:*inv-def inv-bitmap-freelist-def level-empty-def Let-def*)
apply(*subgoal-tac* (*nat* (*free-l Va t*)) < *length* (*levels* (*mem-pool-info Va p*)))
prefer *2* **apply** (*simp add*: *inv-mempool-info-def Let-def*) **apply** (*smt in-set-conv-nth list.set-sel*(*1*))
apply(*rule conjI*)
apply (*metis add-diff-cancel-left′ div-mult-self-is-m mult-is-0 neq0-conv*)

apply(*rule subst*[**where** *t*=*n-max* (*set-bit-allocating* (λ*a*. *if a* = *p then rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)) *else mem-pool-info Va a*)
p (*nat* (*free-l Va t*))
((*hd* (*free-list* (*levels* (*mem-pool-info Va p*) ! *nat* (*free-l Va t*))) −
buf (*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))) *div*

lsizes Va t ! *nat* (*free-l Va t*))
p) **and** *s*=*n-max* (*mem-pool-info Va p*)])
apply(*simp add*:*rmhead-free-list-def set-bit-def*)

apply(*rule conjI*)
using *mp-alloc-stm3-lm2-5* **apply** *metis*
using *mp-alloc-stm3-lm2-3* **apply**(*simp add*:*Let-def*)
done

lemma *head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)) ≠ *NULL* ⟹
{*Va*(|*blk* := (*blk Va*)(*t* := *NULL*)|), *Va* (|*blk* := (*blk Va*)(*t* := *head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))),
mem-pool-info := (*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))|)} ∩
{|*NULL* < ´*blk t*|} =
{*Va*(|*blk* := (*blk Va*)(*t* := *head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))),
mem-pool-info := (*mem-pool-info Va*)(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))|)}
by *simp*

lemma *mp-alloc-stm3-lm1-1*:
inv Va ⟹ *p* ∈ *mem-pools Va* ⟹ *nat* (*free-l Va t*) < *length* (*levels* (*mem-pool-info Va p*)) ⟹
¬ *level-empty* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)) ⟹
{*V*. *V* = *Va*(|*blk* := (*blk Va*)(*t* := *head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))),
mem-pool-info := (*mem-pool-info Va*)
(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))|)
∧ *blk V t* ≠ *NULL*}
={*V*. *V* = *Va*(|*blk* := (*blk Va*)(*t* := *head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))),
mem-pool-info := (*mem-pool-info Va*)
(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))|)}
apply(*rule equalityI*) **apply**(*rule subsetI*)
apply *clarsimp*
apply(*rule subsetI*)
apply *clarsimp*
apply(*simp add*: *head-free-list-def*)
by (*simp add*: *level-empty-def mp-alloc-stm3-lm2-4*)

lemma *mp-alloc-stm3-lm1*:
mp-alloc-precond1-70-2-2 t p sz timeout ∩ {|´*cur* = *Some t*|} ∩ {*Va*} = {*Va*}
⟹ Γ $\vdash_I$ *Some* (*IF level-empty* (´*mem-pool-info p*) (*nat* (´*free-l t*)) *THEN*
´*blk* := ´*blk*(*t* := *NULL*)
ELSE
´*blk* := ´*blk*(*t* := *head-free-list* (´*mem-pool-info p*) (*nat* (´*free-l t*)));;

(* *sys-dlist-remove*(*node*); *)
´*mem-pool-info* := ´*mem-pool-info* (*p* := *rmhead-free-list* (´*mem-pool-info* *p*) (*nat* (´*free-l t*)))

FI;;
(* ==== *end*: *block* = *sys-dlist-get*(&*p*−>*levels*[*l*].*free-list*); *)

IF ´*blk t* ≠ *NULL THEN*
(* *clear-free-bit*(*p*, *l*, *block-num*(*p*, *block*, *lsz*)); *)
´*mem-pool-info* := *set-bit-allocating* ´*mem-pool-info p* (*nat* (´*free-l t*))
(*block-num* (´*mem-pool-info p*) (´*blk t*) ((´*lsizes t*)!(*nat* (´*free-l t*))));;
(* *set the allocating node info of the thread* *)
´*allocating-node* := ´*allocating-node* (*t* := *Some* (|*pool* = *p*, *level* = *nat* (´*free-l t*),
block = (*block-num* (´*mem-pool-info p*) (´*blk t*) ((´*lsizes t*)!(*nat* (´*free-l t*)))), *data* = ´*blk t* |))
FI) sat_p [*mp-alloc-precond1-70-2-2 t p sz timeout* ∩ {|´*cur* = *Some t*|} ∩ {*Va*},
{(*s*, *t*). *s* = *t*}, *UNIV*, {|´(*Pair Va*) ∈ *Mem-pool-alloc-guar t*|} ∩ *mp-alloc-precond2-1 t p sz timeout*]

apply(*subgoal-tac Va*∈*mp-alloc-precond1-70-2-2 t p sz timeout* ∩ {|´*cur* = *Some t*|})
prefer *2* **apply** *auto*[*1*]
apply(*rule Seq*[**where** *mid*=
if level-empty (*mem-pool-info Va p*) (*nat* (*free-l Va t*)) *then*
{*V*. *V* = *Va*(|*blk*:=(*blk Va*)(*t*:=*NULL*)|) ∧ *level-empty* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))}
else
{*V*. *V* = *Va*(|*blk*:=(*blk Va*)(*t* := *head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))),
mem-pool-info := (*mem-pool-info Va*)(*p*:=*rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))|)
∧ ¬ *level-empty* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))}])

apply(*rule Cond*)
apply(*simp add*:*stable-def*)

apply(*rule Basic*)
apply *auto*[*1*] **apply** *simp* **apply**(*simp add*:*stable-def*) **apply**(*simp add*:*stable-def*)

apply(*rule Seq*[**where** *mid*={*V*. *V* = *Va*(|*blk*:=(*blk Va*)(*t* := *head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))|)
∧ ¬ *level-empty* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))}])

apply(*rule Basic*)
apply *auto*[*1*] **apply** *simp* **apply**(*simp add*:*stable-def*) **apply**(*simp add*:*stable-def*)
apply(*rule Basic*)
apply *clarify*

apply *auto*[*1*] **apply** *simp* **apply**(*simp add*:*stable-def*) **apply**(*simp add*:*stable-def*)

apply *simp*

apply(*rule Cond*)
apply(*simp add*:*stable-def*)
apply(*case-tac* ¬ *level-empty* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))
prefer *2*
apply(*subgoal-tac* {*Va*(|*blk* := (*blk Va*)(*t* := *NULL*)|), *Va*
(|*blk* := (*blk Va*)(*t* := *head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))),
mem-pool-info := (*mem-pool-info Va*)
(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))|)} ∩
{|¬ *level-empty* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))|}
= {})
prefer *2* **apply** *auto*[*1*]
using *Emptyprecond*[**where** *P*=*Some* (´*mem-pool-info* := *set-bit-allocating* ´*mem-pool-info p* (*nat* (´*free-l t*))
(*block-num* (´*mem-pool-info p*) (´*blk t*) ((´*lsizes t*)!(*nat* (´*free-l t*))));;
(∗ *set the allocating node info of the thread* ∗)
´*allocating-node* := ´*allocating-node* (*t* := *Some* (|*pool* = *p*, *level* = *nat* (´*free-l t*),
block = (*block-num* (´*mem-pool-info p*) (´*blk t*) ((´*lsizes t*)!(*nat* (´*free-l t*)))), *data* = ´*blk t* |)))
and *rely*={(*x*,*y*). *x*=*y*} **and** *guar*=*UNIV*
and *post*={|´(*Pair Va*) ∈ *Mem-pool-alloc-guar t*|} ∩ *mp-alloc-precond2-1 t p sz timeout*]
apply *meson* **apply** *auto*[*1*]

apply(*rule subst*[**where** *t*= (*if level-empty* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))
then {|´(=) (*Va*(|*blk* := (*blk Va*)(*t* := *NULL*)|)) ∧ *level-empty* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))|}
else {|´(=) (*Va*(|*blk* := (*blk Va*)(*t* := *head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))),
mem-pool-info := (*mem-pool-info Va*)
(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))|)) ∧
¬ *level-empty* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))|}) ∩
{|´*blk t* ≠

NULL⦈ **and** $s = \{V.\ V = Va⦇blk := (blk\ Va)(t :=$ *head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))),
mem-pool-info := (*mem-pool-info Va*)
(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))⦈
∧ ¬ *level-empty* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))
∧ *blk V t* ≠ *NULL*}])
apply *auto*[*1*]
apply(*rule subst*[**where** *t*= {*V*. *V* = *Va*⦇*blk* := (*blk Va*)(*t* := *head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))),
mem-pool-info := (*mem-pool-info Va*)
(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))⦈
∧ ¬ *level-empty* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))
∧ *blk V t* ≠ *NULL*} **and** *s* = {*V*. *V* = *Va*⦇*blk* := (*blk Va*)(*t* := *head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))),
mem-pool-info := (*mem-pool-info Va*)
(*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))⦈
∧ ¬ *level-empty* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))}])
apply(*subgoal-tac* (*nat* (*free-l Va t*)) < *length* (*levels* (*mem-pool-info Va p*)))
prefer *2* **apply**(*simp add*:*inv-def inv-mempool-info-def Let-def*) **apply** *force*
apply *simp*
using *mp-alloc-stm3-lm1-1* **apply** *force*

apply(*rule Seq*[**where** *mid*=
{*V*. *let vb* = *Va*⦇*blk* := (*blk Va*)(*t* := *head-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))),
mem-pool-info := (*mem-pool-info Va*) (*p* := *rmhead-free-list* (*mem-pool-info Va p*) (*nat* (*free-l Va t*)))⦈
in V = *vb*⦇ *mem-pool-info* := *set-bit-allocating* (*mem-pool-info vb*) *p* (*nat* (*free-l vb t*))
(*block-num* (*mem-pool-info vb p*) (*blk vb t*) ((*lsizes vb t*)!(*nat* (*free-l vb t*))))⦈
∧ ¬ *level-empty* (*mem-pool-info Va p*) (*nat* (*free-l Va t*))}])
apply(*rule Basic*) **apply** *clarsimp*
apply *simp* **apply**(*simp add*:*stable-def*) **apply**(*simp add*:*stable-def*)
apply(*rule Basic*)
using *mp-alloc-stm3-lm2* **apply** *meson*
apply *simp* **apply**(*simp add*:*stable-def*)
using *stable-id2*[*of* ⦃´(*Pair Va*) ∈ *Mem-pool-alloc-guar t*⦄ ∩ *mp-alloc-precond2-1 t p sz timeout*]
apply *meson*

apply(*unfold Skip-def*)

apply(*rule Basic*)
using *mp-alloc-stm3-lm3* **apply** *meson*
apply *simp* **apply**(*simp add*:*stable-def*)
using *stable-id2*[*of* {|´(*Pair Va*) ∈ *Mem-pool-alloc-guar t*|} ∩ *mp-alloc-precond2-1 t p sz timeout*]
apply *meson*
apply *simp*

done

lemma *mp-alloc-stm3-lm*:
$\Gamma \vdash_I$ *Some* (*t* ▶ *ATOMIC*
(∗ ==== *start*: *block* = *sys-dlist-get*(&*p*−>*levels*[*l*].*free-list*); ∗)
IF level-empty (´*mem-pool-info p*) (*nat* (´*free-l t*)) *THEN*
´*blk* := ´*blk*(*t* := *NULL*)
ELSE
´*blk* := ´*blk*(*t* := *head-free-list* (´*mem-pool-info p*) (*nat* (´*free-l t*)));;

(∗ *sys-dlist-remove*(*node*); ∗)
´*mem-pool-info* := ´*mem-pool-info* (*p* := *rmhead-free-list* (´*mem-pool-info p*) (*nat* (´*free-l t*)))

FI;;
(∗ ==== *end*: *block* = *sys-dlist-get*(&*p*−>*levels*[*l*].*free-list*); ∗)

IF ´*blk t* ≠ *NULL THEN*
(∗ *clear-free-bit*(*p*, *l*, *block-num*(*p*, *block*, *lsz*)); ∗)
´*mem-pool-info* := *set-bit-allocating* ´*mem-pool-info p* (*nat* (´*free-l t*))
(*block-num* (´*mem-pool-info p*) (´*blk t*) ((´*lsizes t*)!(*nat* (´*free-l t*))));;
(∗ *set the allocating node info of the thread* ∗)
´*allocating-node* := ´*allocating-node* (*t* := *Some* (|*pool* = *p*, *level* = *nat* (´*free-l t*),
block = (*block-num* (´*mem-pool-info p*) (´*blk t*) ((´*lsizes t*)!(*nat* (´*free-l t*)))), *data* = ´*blk t* |))
FI
END)
sat_p [*mp-alloc-precond1-70-2-2 t p sz timeout*, *Mem-pool-alloc-rely t*, *Mem-pool-alloc-guar t*,
mp-alloc-precond2-1 t p sz timeout]
apply(*simp add*:*stm-def*)
apply(*rule Await*)
using *mp-alloc-precond1-70-2-2-stb* **apply** *simp*
using *mp-alloc-precond2-1-stb* **apply** *simp*
apply(*clarify*)
apply(*rule Await*)
using *stable-id2* **apply** *fast* **using** *stable-id2* **apply** *fast*

apply *clarify*
apply(*case-tac V = Va*) **prefer** *2* **apply** *simp* **using** *Emptyprecond* **apply** *auto*[*1*]
apply *simp*
apply(*case-tac mp-alloc-precond1-70-2-2 t p sz timeout* ∩ {|´*cur* = *Some t*|} ∩ {*Va*} = {})
using *Emptyprecond* **apply** *auto*[*1*]
apply(*subgoal-tac mp-alloc-precond1-70-2-2 t p sz timeout* ∩ {|´*cur* = *Some t*|} ∩ {*Va*} = {*Va*})
prefer *2* **using** *int1-eq*[**where** *P*=*mp-alloc-precond1-70-2-2 t p sz timeout* ∩ {|´*cur* = *Some t*|}] **apply** *meson*
using *mp-alloc-stm3-lm1*[*of t p timeout sz*] **apply** *auto*[*1*]
done

term *mp-alloc-precond1-70-2-2 t p sz timeout*
term *mp-alloc-precond2-1 t p sz timeout*

10.7 stm4

abbreviation *mp-alloc-precond2-1-1-loopinv-0 t p sz tm* ≡
mp-alloc-precond2-1-1-loopinv t p sz tm ∩ {|´*from-l t* < ´*alloc-l t*|}

lemma *mp-alloc-precond2-1-1-loopinv-0-stb*: *stable* (*mp-alloc-precond2-1-1-loopinv-0 t p sz tm*) (*Mem-pool-alloc-rely t*)
apply(*rule stable-int2*)
using *mp-alloc-precond2-1-1-loopinv-stb* **apply** *auto*[*1*]
apply(*simp add*:*stable-def*) **apply** *clarify*
apply(*simp add*:*Mem-pool-alloc-rely-def lvars-nochange-rel-def lvars-nochange-def*)

apply(*case-tac x*=*y*) **apply** *auto*[*1*] **apply** *clarify*
apply(*simp add*: *block-num-def lvars-nochange-rel-def lvars-nochange-def gvars-conf-stable-def gvars-conf-def*)
done

abbreviation *mp-alloc-precond2-1-1-loopinv-1′ t p sz tm* ≡
mp-alloc-precond2-1-1 t p sz tm ∩ {|´*from-l t* ≤ ´*alloc-l t* ∧ ´*from-l t* ≥ ´*free-l t* ∧ ´*allocating-node t* = *Some* (|*pool* = *p*, *level* = *nat* (´*from-l t* + *1*), *block* = *block-num* (´*mem-pool-info p*) (´*blk t*) ((´*lsizes t*)!(*nat* (´*from-l t* + *1*))), *data* = ´*blk t* |) |}

abbreviation *mp-alloc-precond2-1-1-loopinv-1 t p sz tm* ≡
{*s*. *inv s*} ∩ {|´*freeing-node t* = *None*|} ∩ {|*p* ∈ ´*mem-pools* ∧ *tm* ≥ −*1*|} ∩ *mp-alloc-precond7-ext t p sz tm* ∩ {|¬ ´*rf t*|}

$\cap$ *mp-alloc-precond1-70-ext t p sz tm* $\cap$ $-$ ⦃´*alloc-l t* $<$ *0*⦄
$\cap$ $-$ ⦃´*free-l t* $<$ *0*⦄ $\cap$ $-$⦃´*blk t* $=$ *NULL*⦄ $\cap$ ⦃´*from-l t* $<$ ´*alloc-l t*⦄
$\cap$ ⦃´*from-l t* $\leq$ ´*alloc-l t* $\wedge$ ´*from-l t* $\geq$ ´*free-l t*
$\wedge$ ´*allocating-node t* $=$ *Some* (|*pool* $=$ *p*, *level* $=$ *nat* (´*from-l t* $+$ *1*),
block $=$ *block-num* (´*mem-pool-info p*) (´*blk t*) ((´*lsizes t*)!(*nat* (´*from-l t* $+$ *1*))),
data $=$ ´*blk t* |)
$\wedge$ ´*alloc-memblk-data-valid p* (*the* (´*allocating-node t*))
$\wedge$ ($\exists$ *n*. *n* $<$ *n-max* (´*mem-pool-info p*) $*$ (*4* ^ (*nat* (´*from-l t* $+$ *1*)))
$\wedge$ ´*blk t* $=$ *buf* (´*mem-pool-info p*) $+$ *n* $*$ (*max-sz* (´*mem-pool-info p*) *div* (*4* ^ (*nat* (´*from-l t* $+$ *1*))))) ⦄

term *mp-alloc-precond2-1-1-loopinv-0 t p sz tm*
term *mp-alloc-precond2-1-1-loopinv-1 t p sz tm*
term *mp-alloc-precond2-1-1-loopinv-1′ t p sz tm*

lemma *mp-alloc-precond2-1-1-loopinv-1′-stb*: *stable* (*mp-alloc-precond2-1-1-loopinv-1′ t p sz tm*) (*Mem-pool-alloc-rely t*)
apply(*rule stable-int2*)
using *mp-alloc-precond2-1-1-stb* **apply** *auto*[*1*]
apply(*simp add*:*stable-def*) **apply** *clarify*
apply(*simp add*:*Mem-pool-alloc-rely-def lvars-nochange-rel-def lvars-nochange-def*)

apply(*case-tac x*=*y*) **apply** *auto*[*1*] **apply** *clarify*
apply(*simp add*: *block-num-def lvars-nochange-rel-def lvars-nochange-def gvars-conf-stable-def gvars-conf-def*)
done

lemma *mp-alloc-precond2-1-1-loopinv-1-stb*: *stable* (*mp-alloc-precond2-1-1-loopinv-1 t p sz tm*) (*Mem-pool-alloc-rely t*)
apply(*rule stable-int2*) **apply**(*rule stable-int2*) **apply**(*rule stable-int2*)
apply(*rule stable-int2*) **apply**(*rule stable-int2*) **apply**(*rule stable-int2*)
apply(*rule stable-int2*) **apply**(*rule stable-int2*) **apply**(*rule stable-int2*)
apply(*rule stable-int2*)
apply (*simp add*: *stable-inv-alloc-rely1*)
apply (*simp add*: *mp-alloc-freenode-stb*)
apply (*simp add*: *mp-alloc-precond1-ext-stb*)
apply (*simp add*: *mp-alloc-precond7-ext-stb*)
apply (*simp add*: *mp-alloc-precond1-0-ext-stb*)
using *mp-alloc-precond1-70-ext-stb* **apply** *blast*
apply (*simp add*: *mp-alloc-precond1-70-2-ext-stb*)
apply (*simp add*: *mp-alloc-precond1-70-2-2-ext-stb*)
using *mp-alloc-precond2-1-1-ext-stb* **apply** *blast*
apply(*simp add*:*stable-def*) **apply** *clarify*
apply(*simp add*:*Mem-pool-alloc-rely-def lvars-nochange-rel-def lvars-nochange-def*)
apply *smt*

apply(*simp add*:*stable-def*) **apply** *clarify*

apply(*subgoal-tac buf* (*mem-pool-info x p*) = *buf* (*mem-pool-info y p*))
prefer *2* **apply**(*simp add*: *Mem-pool-alloc-rely-def gvars-conf-stable-def gvars-conf-def*)
apply *metis*
apply(*subgoal-tac from-l x t* + *1* = *from-l y t* + *1*)
prefer *2* **apply**(*simp add*: *Mem-pool-alloc-rely-def lvars-nochange-rel-def lvars-nochange-def*)
apply *smt*
apply(*subgoal-tac max-sz* (*mem-pool-info x p*) = *max-sz* (*mem-pool-info y p*))
prefer *2* **apply**(*simp add*: *Mem-pool-alloc-rely-def gvars-conf-stable-def gvars-conf-def*)
apply *metis*
apply(*subgoal-tac blk x t* = *blk y t*)
prefer *2* **apply**(*simp add*: *Mem-pool-alloc-rely-def lvars-nochange-rel-def lvars-nochange-def*)
apply *smt*
apply(*subgoal-tac allocating-node x t* = *allocating-node y t*)
prefer *2* **apply**(*simp add*: *Mem-pool-alloc-rely-def lvars-nochange-rel-def lvars-nochange-def*)
apply *smt*
apply(*subgoal-tac lsizes x t* = *lsizes y t*)
prefer *2* **apply**(*simp add*: *Mem-pool-alloc-rely-def lvars-nochange-rel-def lvars-nochange-def*)
apply *smt*
apply(*subgoal-tac n-max* (*mem-pool-info x p*) = *n-max* (*mem-pool-info y p*))
prefer *2* **apply**(*simp add*: *Mem-pool-alloc-rely-def gvars-conf-stable-def gvars-conf-def*)
apply *metis*
apply(*subgoal-tac block-num* (*mem-pool-info x p*) (*blk x t*) (*lsizes x t* ! *nat* (*from-l x t* + *1*))
= *block-num* (*mem-pool-info y p*) (*blk y t*) (*lsizes y t* ! *nat* (*from-l y t* + *1*)))
prefer *2* **apply**(*simp add*: *block-num-def Mem-pool-alloc-rely-def lvars-nochange-rel-def lvars-nochange-def*)
apply(*case-tac x*=*y*) **apply** *auto*[*1*]
apply(*simp add*:*Mem-pool-alloc-rely-def gvars-conf-stable-def gvars-conf-def lvars-nochange-rel-def lvars-nochange-def*)
apply *smt*

done

abbreviation *mp-alloc-stm4-pre-precond1 Va t p* ≡
Va(|*bn* := (*bn Va*) (*t* := *block-num* (*mem-pool-info Va p*) (*blk Va t*) ((*lsizes Va t*)!(*nat* (*from-l Va t*))))|)
abbreviation *mp-alloc-stm4-pre-precond2 Va t p* ≡
Va(|*mem-pool-info* := *set-bit-divide* (*mem-pool-info Va*) *p* (*nat* (*from-l Va t*)) (*bn Va t*)|)
abbreviation *mp-alloc-stm4-pre-precond3 Va t p* ≡
Va(|*mem-pool-info* := *set-bit-allocating* (*mem-pool-info Va*) *p* (*nat* (*from-l Va t* + *1*)) (*4* ∗ *bn Va t*)|)
abbreviation *mp-alloc-stm4-pre-precond4 Va t p* ≡
Va(|*allocating-node* := (*allocating-node Va*)(*t* := *Some* (|*pool* = *p*, *level* = *nat* (*from-l Va t* + *1*),
block = *4* ∗ *bn Va t*, *data* = *blk Va t* |))|)
abbreviation *mp-alloc-stm4-pre-precond5 Va t p* ≡ *Va*(|*i* := (*i Va*) (*t* := *1*)|)

definition *mp-alloc-stm4-pre-precond-f Va t p* (* *we use definition here. abbreviation leads slow parsing in lemmas* *)
  $\equiv$ *mp-alloc-stm4-pre-precond5*
    (*mp-alloc-stm4-pre-precond4*
    (*mp-alloc-stm4-pre-precond3*
    (*mp-alloc-stm4-pre-precond2*
    (*mp-alloc-stm4-pre-precond1 Va t p*) *t p*) *t p*) *t p*) *t p*

abbreviation *mp-alloc-stm4-loopinv Va t mp*
  $\equiv \{V.\ cur\ V = cur\ Va \wedge tick\ V = tick\ Va \wedge thd\text{-}state\ V = thd\text{-}state\ Va \wedge (V,Va) \in gvars\text{-}conf\text{-}stable$
    $\wedge (\forall p.\ p \neq mp \longrightarrow mem\text{-}pool\text{-}info\ V\ p = mem\text{-}pool\text{-}info\ Va\ p)$
    $\wedge\ wait\text{-}q\ (mem\text{-}pool\text{-}info\ V\ mp) = wait\text{-}q\ (mem\text{-}pool\text{-}info\ Va\ mp)$
    $\wedge (\forall t'.\ t' \neq t \longrightarrow lvars\text{-}nochange\ t'\ V\ Va)$
    $\wedge (\forall jj.\ jj \neq nat\ (from\text{-}l\ Va\ t + 1) \longrightarrow levels\ (mem\text{-}pool\text{-}info\ V\ mp)\ !\ jj = levels\ (mem\text{-}pool\text{-}info\ Va\ mp)\ !\ jj)$
    $\wedge (bits\ (levels\ (mem\text{-}pool\text{-}info\ V\ mp)\ !\ nat\ (from\text{-}l\ Va\ t + 1))$
      $= list\text{-}updates\text{-}n\ (bits\ (levels\ (mem\text{-}pool\text{-}info\ Va\ mp)\ !\ nat\ (from\text{-}l\ Va\ t + 1)))\ (bn\ Va\ t * 4 + 1)\ (i\ V\ t - 1)\ FREE)$
    $\wedge (free\text{-}list\ (levels\ (mem\text{-}pool\text{-}info\ V\ mp)\ !\ nat\ (from\text{-}l\ Va\ t + 1))$
      $= inserts\ (map\ (\lambda ii.\ (lsizes\ Va\ t)\ !\ (nat\ (from\text{-}l\ Va\ t + 1)) * ii + blk\ V\ t)\ [1..<i\ V\ t])$
        $(free\text{-}list\ (levels\ (mem\text{-}pool\text{-}info\ Va\ mp)\ !\ nat\ (from\text{-}l\ Va\ t + 1))))$
    $\wedge\ j\ V = j\ Va \wedge ret\ V = ret\ Va \wedge endt\ V = endt\ Va \wedge rf\ V = rf\ Va \wedge tmout\ V = tmout\ Va$
    $\wedge\ lsizes\ V = lsizes\ Va \wedge alloc\text{-}l\ V = alloc\text{-}l\ Va \wedge free\text{-}l\ V = free\text{-}l\ Va$
    $\wedge\ from\text{-}l\ V = from\text{-}l\ Va \wedge blk\ V = blk\ Va \wedge nodev\ V = nodev\ Va$
    $\wedge\ bn\ V = bn\ Va \wedge alloc\text{-}lsize\text{-}r\ V = alloc\text{-}lsize\text{-}r\ Va \wedge lvl\ V = lvl\ Va \wedge bb\ V = bb\ Va$
    $\wedge\ block\text{-}pt\ V = block\text{-}pt\ Va \wedge th\ V = th\ Va \wedge need\text{-}resched\ V = need\text{-}resched\ Va$
    $\wedge\ mempoolalloc\text{-}ret\ V = mempoolalloc\text{-}ret\ Va \wedge freeing\text{-}node\ V = freeing\text{-}node\ Va$
    $\wedge\ allocating\text{-}node\ V = allocating\text{-}node\ Va$
    $\wedge\ i\ V\ t > 0 \wedge i\ V\ t \leq 4\}$

lemma *in-mp-alloc-stm4-loopinv*: $i\ V\ t = 1 \Longrightarrow V \in mp\text{-}alloc\text{-}stm4\text{-}loopinv\ V\ t\ mp$
  apply *simp*
  apply(*rule conjI*) **apply**(*simp add*:*gvars-conf-stable-def gvars-conf-def*)
  apply(*rule conjI*) **apply**(*simp add*:*lvars-nochange-def*)
  apply(*simp add*:*inserts-def*)
done

abbreviation *mp-alloc-stm4-while-precond1 V t p* $\equiv$
  $V(\!|lbn := (lbn\ V)\ (t := 4 * bn\ V\ t + i\ V\ t)|\!)$
abbreviation *mp-alloc-stm4-while-precond2 V t p* $\equiv$
  $V(\!|lsz := (lsz\ V)\ (t := lsizes\ V\ t\ !\ nat\ (from\text{-}l\ V\ t + 1))|\!)$
abbreviation *mp-alloc-stm4-while-precond3 V t p* $\equiv$

$V(|block2 := (block2\ V)\ (t := lsz\ V\ t * i\ V\ t + blk\ V\ t)|)$
abbreviation *mp-alloc-stm4-while-precond4 V t p* ≡
$V(|mem\text{-}pool\text{-}info := set\text{-}bit\text{-}free\ (mem\text{-}pool\text{-}info\ V)\ p\ (nat\ (from\text{-}l\ V\ t + 1))\ (lbn\ V\ t)|)$
abbreviation *mp-alloc-stm4-while-precond5 V t p* ≡
$V(|mem\text{-}pool\text{-}info := (mem\text{-}pool\text{-}info\ V)\ (p := append\text{-}free\text{-}list\ (mem\text{-}pool\text{-}info\ V\ p)\ (nat\ (from\text{-}l\ V\ t + 1))\ (block2\ V\ t))|)$

lemma *mp-alloc-stm4-pre-in*:
{*mp-alloc-stm4-while-precond4*
(*mp-alloc-stm4-while-precond3*
(*mp-alloc-stm4-while-precond2*
(*mp-alloc-stm4-while-precond1 V t p*) *t p*) *t p*) *t p*} ∩
{|*block-fits* (´*mem-pool-info p*) (´*block2 t*) (´*lsz t*)|}
⊆ {|´*mp-alloc-stm4-while-precond5 t p*
∈ {*mp-alloc-stm4-while-precond5*
(*mp-alloc-stm4-while-precond4*
(*mp-alloc-stm4-while-precond3*
(*mp-alloc-stm4-while-precond2*
(*mp-alloc-stm4-while-precond1 V t p*) *t p*) *t p*) *t p*) *t p*}|}
by *auto*

lemma *mp-alloc-stm4-lsizes*: *lsizes Va t* = *lsizes* (*mp-alloc-stm4-pre-precond-f Va t p*) *t*
by(*simp add*:*mp-alloc-stm4-pre-precond-f-def*)

lemma *mp-alloc-stm4-pre-froml*: *from-l Va t* = *from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t*
by(*simp add*:*mp-alloc-stm4-pre-precond-f-def*)

lemma *mp-alloc-stm4-pre-buf*: *buf* (*mem-pool-info Va q*) = *buf* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *q*)
by (*simp add*:*mp-alloc-stm4-pre-precond-f-def set-bit-def*)

lemma *mp-alloc-stm4-nmax*: *n-max* (*mem-pool-info Va q*) = *n-max* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *q*)
by (*simp add*:*mp-alloc-stm4-pre-precond-f-def set-bit-def*)

lemma *mp-alloc-stm4-pre-maxsz*: *max-sz* (*mem-pool-info Va q*) = *max-sz* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *q*)
by (*simp add*:*mp-alloc-stm4-pre-precond-f-def set-bit-def*)

lemma *mp-alloc-stm4-blk*: *blk Va* = *blk* (*mp-alloc-stm4-pre-precond-f Va t p*)
by (*simp add*:*mp-alloc-stm4-pre-precond-f-def*)

lemma *mp-alloc-stm4-blockfit-help-1*:
$p \in mem\text{-}pools\ Va \Longrightarrow inv\ Va \Longrightarrow$
$(\forall ii<length\ (lsizes\ Va\ t).\ lsizes\ Va\ t\ !\ ii = ALIGN4\ (max\text{-}sz\ (mem\text{-}pool\text{-}info\ Va\ p))\ div\ 4\ \hat{}\ ii) \Longrightarrow$

length (*lsizes Va t*) ≤ *n-levels* (*mem-pool-info Va p*) ⟹
ALIGN4 (*max-sz* (*mem-pool-info Va p*)) = *max-sz* (*mem-pool-info Va p*) ⟹
Suc (*nat* (*from-l Va t*)) < *length* (*lsizes Va t*) ⟹
max-sz (*mem-pool-info Va p*) = *4* ˆ *nat* (*from-l Va t*) ∗ *lsizes Va t* ! *nat* (*from-l Va t*)
apply(*simp add*:*inv-def inv-mempool-info-def*)
apply(*subgoal-tac* ∃ *n*>*0*. *max-sz* (*mem-pool-info Va p*) = *4* ∗ *n* ∗ *4* ˆ *n-levels* (*mem-pool-info Va p*))
prefer *2* **apply** *meson*
apply(*subgoal-tac max-sz* (*mem-pool-info Va p*) *mod* (*4* ˆ *nat* (*from-l Va t*)) = *0*)
prefer *2* **apply** *clarsimp* **using** *ge-pow-mod-0*[*of nat* (*from-l Va t*) *n-levels* (*mem-pool-info Va p*)] **apply** *auto*[*1*]
using *mod0-div-self* [*of max-sz* (*mem-pool-info Va p*) *4* ˆ *nat* (*from-l Va t*)] **apply** *simp*
done

lemma *mp-alloc-stm4-blockfit-help-2*:
ii ≥ *0* ⟹ (*a*::*nat*) *mod* *4* ˆ *nat* (*ii*+*1*) = *0* ⟹ *a div 4* ˆ *nat* (*ii*+*1*) ∗ *4* = *a div 4* ˆ (*nat ii*)
apply(*subgoal-tac nat* (*ii*+*1*) = *nat ii* + *1*) **prefer** *2* **apply** *auto*[*1*]
by *auto*

lemma *mp-alloc-stm4-blockfit-help3*:
inv Va ⟹
p ∈ *mem-pools Va* ⟹
nmax > *0* ⟹
maxsz mod frml = *0* ⟹
n < *nmax* ∗ *frml* ⟹
blk Va t = *buf* (*mem-pool-info Va p*) + *n* ∗ (*maxsz div frml*) ⟹
maxsz div frml + *blk Va t*
≤ *nmax* ∗ *maxsz* + *buf* (*mem-pool-info Va p*)
apply(*case-tac n* = *0*)
apply (*metis* (*no-types*, *lifting*) *Nat.add-0-right add-le-mono1 div-le-dividend div-mult-self1-is-m le-trans mult-is-0*)

apply(*subgoal-tac blk Va t* ≤ *buf* (*mem-pool-info Va p*) + (*nmax* ∗ *frml* − *1*) ∗ (*maxsz div frml*))
prefer *2* **apply** *auto*[*1*]
by (*smt add.left-commute add-diff-cancel-left′ le-diff-conv mp-alloc-stm3-lm2-3-1 mult-eq-if nat-add-left-cancel-le not-less-zero*)

lemma *mp-alloc-stm4-blockfit-help4*:
inv Va ∧
p ∈ *mem-pools Va* ∧
length (*lsizes Va t*) ≤ *n-levels* (*mem-pool-info Va p*) ∧
alloc-l Va t < *int* (*n-levels* (*mem-pool-info Va p*)) ∧

from-l Va t < *alloc-l Va t* ∧
¬ *free-l Va t* < *OK* ∧
free-l Va t ≤ *from-l Va t* ⟹
ALIGN4 (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat* (*from-l Va t* + *1*) ∗ *4* =
ALIGN4 (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat* (*from-l Va t*)
apply(*subgoal-tac ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *mod 4* ^ *nat* (*from-l Va t* + *1*) = *0*)
prefer *2*
apply(*rule subst*[**where** *t*=*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) **and** *s*=*max-sz* (*mem-pool-info Va p*)])
apply (*metis inv-maxsz-align4*)
apply(*subgoal-tac nat* (*from-l Va t* + *1*) < *n-levels* (*mem-pool-info Va p*))
prefer *2* **apply** (*smt nat-less-iff*)
apply(*simp add*:*inv-def inv-mempool-info-def Let-def*)
apply(*subgoal-tac n-levels* (*mem-pool-info Va p*) = *length* (*levels* (*mem-pool-info Va p*)))
prefer *2* **apply** *simp*
apply (*metis ge-pow-mod-0*)
apply(*rule mp-alloc-stm4-blockfit-help-2*)
apply (*metis int-nat-eq linorder-not-less nat-int neq0-conv zless-nat-conj*)
apply *simp*
done

lemma *mp-alloc-stm4-blockfit-help*:
Va ∈ *mp-alloc-precond2-1-1-loopinv-0 t p sz timeout* ∩ ⦃´*cur* = *Some t*⦄ ∧ *ii* < *4*
⟹ *block-fits* (*mem-pool-info Va p*) (*lsizes Va t* ! *nat* (*from-l Va t* + *1*) ∗ *ii* + *blk Va t*)
(*lsizes Va t* ! *nat* (*from-l Va t* + *1*))
apply(*simp add*:*block-fits-def block-num-def buf-size-def*)

apply(*case-tac alloc-l Va t* = *ETIMEOUT* ∧ *free-l Va t* = *ETIMEOUT* ∧ *length* (*lsizes Va t*) = *Suc NULL*)
apply *simp* **apply** *simp*

apply(*subgoal-tac lsizes Va t* ! *nat* (*from-l Va t* + *1*) = *ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat* (*from-l Va t* + *1*))
prefer *2* **apply**(*subgoal-tac nat* (*from-l Va t* + *1*) < *length* (*lsizes Va t*))
prefer *2* **apply** (*smt nat-less-iff*)
apply *simp*
apply(*subgoal-tac lsizes Va t* ! *nat* (*from-l Va t*) = *ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat* (*from-l Va t*))
prefer *2* **apply**(*subgoal-tac nat* (*from-l Va t*) < *length* (*lsizes Va t*))
prefer *2* **apply** (*smt nat-less-iff*)
apply *simp*

apply(*rule subst*[**where** *t*=*lsizes Va t* ! *nat* (*from-l Va t* + *1*) ∗ *ii* + *blk Va t* + *lsizes Va t* ! *nat* (*from-l Va t* + *1*)
and *s*=*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *nat*

(*from-l Va t + 1*) ∗ *ii* + *blk Va t* + *ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *nat* (*from-l Va t + 1*)])
apply *simp*
apply(*subgoal-tac* (*blk Va t* − *buf* (*mem-pool-info Va p*)) *div lsizes Va t* ! *nat* (*from-l Va t*)
< *n-max* (*mem-pool-info Va p*) ∗ *4* ˆ *nat* (*from-l Va t*))
prefer *2* **apply** *force*
apply(*subgoal-tac blk Va t* ≥ *buf* (*mem-pool-info Va p*))
prefer *2* **apply** *force*
apply(*subgoal-tac* (*blk Va t* − *buf* (*mem-pool-info Va p*)) *mod* (*ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *nat* (*from-l Va t*)) = *0*)
prefer *2* **apply** (*metis diff-add-inverse inv-maxsz-align4 mod-mult-self2-is-0*)

apply(*subgoal-tac ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *nat* (*from-l Va t + 1*) ∗ *ii* + *blk Va t* +
ALIGN4 (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *nat* (*from-l Va t + 1*)
≤ *ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *nat* (*from-l Va t + 1*) ∗ *4* + *blk Va t*)
prefer *2* **apply** *simp*

apply(*subgoal-tac ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *nat* (*from-l Va t + 1*) ∗ *4*
= *ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *nat* (*from-l Va t*))
prefer *2* **using** *mp-alloc-stm4-blockfit-help4* **apply** *simp*

apply(*subgoal-tac ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *nat* (*from-l Va t*) + *blk Va t*
≤ *n-max* (*mem-pool-info Va p*) ∗ *max-sz* (*mem-pool-info Va p*) + *buf* (*mem-pool-info Va p*))
apply *simp*
apply *clarify*
apply(*subgoal-tac n-max* (*mem-pool-info Va p*) > *0*)
prefer *2* **apply** (*metis gr0I mult-is-0 not-less-zero*)
apply(*subgoal-tac max-sz* (*mem-pool-info Va p*) *mod 4* ˆ *nat* (*from-l Va t*) = *0*)
prefer *2* **apply**(*subgoal-tac* ∃ *n*>*0*. *max-sz* (*mem-pool-info Va p*) = (*4* ∗ *n*) ∗ (*4* ˆ *n-levels* (*mem-pool-info Va p*)))
prefer *2* **apply** (*metis inv-mempool-info-def inv-def*)
apply (*smt ge-pow-mod-0 of-nat-less-imp-less zless-nat-conj zless-nat-eq-int-zless*)

using *mp-alloc-stm4-blockfit-help3*[*of Va p n-max* (*mem-pool-info Va p*) *max-sz* (*mem-pool-info Va p*) *4* ˆ *nat* (*from-l Va t*) - *t*]
by (*metis inv-maxsz-align4*)

lemma $a \geq b \Longrightarrow c * d \geq e \Longrightarrow (a{::}nat) - b \leq c * d - e \Longrightarrow a + e - b \leq c * d$

by (*simp add*: *Nat.le-diff-conv2*)

lemma $a + b > c \Longrightarrow a + b - c \leq d \Longrightarrow e + f < b \Longrightarrow e + a + f - \mathit{Suc}\ c < d$
by *simp*

lemma $(x::nat) > y \Longrightarrow \exists\, n.\ (4::nat)\ \hat{}\ x = 4\ \hat{}\ y * n$
apply(*subgoal-tac* $4\ \hat{}\ x = 4\ \hat{}\ y * 4\ \hat{}\ (x - y)$) **prefer** *2* **apply** *auto*
by (*metis add-diff-inverse-nat less-imp-le-nat not-less power-add*)

lemma *int-empt1*: $(\forall\, v.\ v \in P \longrightarrow v \notin Q) \Longrightarrow P \cap Q = \{\}$ **by** *auto*

lemma *mp-alloc-stm4-blockfit1-1*:
allocating-node Va t =
Some (|*pool* = *p*, *level* = *nat* (*from-l Va t*), *block* = *block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*)),
data = *blk Va t*|) ∧ *data* (*the* (*allocating-node Va t*)) = *buf* (*mem-pool-info Va p*) +
block (*the* (*allocating-node Va t*)) ∗ (*max-sz* (*mem-pool-info Va p*) *div* *4* ˆ *level* (*the* (*allocating-node Va t*)))
∧ *block* (*the* (*allocating-node Va t*)) < *n-max* (*mem-pool-info Va p*) ∗ *4* ˆ *level* (*the* (*allocating-node Va t*)) ⟹
alloc-blk-valid Va p (*nat* (*from-l Va t*)) (*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*))) (*blk Va t*)
apply(*simp add*:*block-num-def*) **apply** *auto*
done

lemma *mp-alloc-stm4-blockfit1*:
Va ∈ *mp-alloc-precond2-1-1-loopinv-0 t p sz timeout* ∩ {|´*cur* = *Some t*|} ⟹
V ∈ *mp-alloc-stm4-loopinv* (*mp-alloc-stm4-pre-precond-f Va t p*) *t p* ∩ {|´*i t* < *4*|} ⟹
∀ *v*. *v* ∈ {*mp-alloc-stm4-while-precond4*
(*mp-alloc-stm4-while-precond3*
(*mp-alloc-stm4-while-precond2*
(*mp-alloc-stm4-while-precond1 V t p*) *t p*) *t p*) *t p*} ⟶
v ∉ − {|*block-fits* (´*mem-pool-info p*) (´*block2 t*) (´*lsz t*)|}
apply *simp*
apply(*simp add*:*block-fits-def buf-size-def set-bit-def*)
apply(*subgoal-tac lsizes* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* = *lsizes Va t*)
prefer *2* **using** *mp-alloc-stm4-lsizes* **apply** *metis*
apply(*subgoal-tac from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* = *from-l Va t*)
prefer *2* **using** *mp-alloc-stm4-pre-froml* **apply** *metis*
apply(*subgoal-tac buf* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) = *buf* (*mem-pool-info Va p*))
prefer *2* **using** *mp-alloc-stm4-pre-buf* **apply** *metis*
apply(*subgoal-tac n-max* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*)

= *n-max* (*mem-pool-info Va p*))
  prefer *2* **using** *mp-alloc-stm4-nmax* **apply** *metis*
 apply(*subgoal-tac max-sz* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) = *max-sz* (*mem-pool-info Va p*))
  prefer *2* **using** *mp-alloc-stm4-pre-maxsz* **apply** *metis*
 apply(*subgoal-tac blk* (*mp-alloc-stm4-pre-precond-f Va t p*) = *blk Va*)
  prefer *2* **using** *mp-alloc-stm4-blk* **apply** *metis*

 apply(*subgoal-tac buf* (*mem-pool-info V p*) = *buf* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*))
  prefer *2* **apply** (*simp add*:*gvars-conf-stable-def gvars-conf-def*)
 apply(*subgoal-tac n-max* (*mem-pool-info V p*) = *n-max* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*))
  prefer *2* **apply** (*simp add*:*gvars-conf-stable-def gvars-conf-def*)
 apply(*subgoal-tac max-sz* (*mem-pool-info V p*) = *max-sz* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*))
  prefer *2* **apply** (*simp add*:*gvars-conf-stable-def gvars-conf-def*)

 apply(*rule subst*[**where** *t*=*buf* (*mem-pool-info V p*) **and** *s*=*buf* (*mem-pool-info Va p*)]) **apply** *simp*
 apply(*rule subst*[**where** *t*=*n-max* (*mem-pool-info V p*) **and** *s*=*n-max* (*mem-pool-info Va p*)]) **apply** *simp*
 apply(*rule subst*[**where** *t*=*max-sz* (*mem-pool-info V p*) **and** *s*=*max-sz* (*mem-pool-info Va p*)]) **apply** *simp*
 apply(*rule subst*[**where** *t*=*lsizes* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* **and** *s*=*lsizes Va t*]) **apply** *simp*
 apply(*rule subst*[**where** *t*=*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* **and** *s*=*from-l Va t*]) **apply** *simp*
 apply(*rule subst*[**where** *t*=*blk* (*mp-alloc-stm4-pre-precond-f Va t p*) **and** *s*=*blk Va*]) **apply** *simp*

 using *mp-alloc-stm4-blockfit-help* [*of Va t p timeout sz i V t*] **apply**(*unfold block-fits-def buf-size-def*) **apply** *simp*
 apply(*case-tac alloc-l Va t* = *ETIMEOUT* ∧ *free-l Va t* = *ETIMEOUT* ∧ *length* (*lsizes Va t*) = *Suc NULL*)
  apply *simp*
 apply *simp*
 apply(*subgoal-tac alloc-blk-valid Va p* (*nat* (*from-l Va t*)) (*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*))) (*blk Va t*))
  prefer *2* **using** *mp-alloc-stm4-blockfit1-1* [*of Va t p*] **apply** *argo* **apply** *metis*
done

lemma *mp-alloc-stm4-blockfit*:
 Va ∈ *mp-alloc-precond2-1-1-loopinv-0 t p sz timeout* ∩ ⦃´*cur* = *Some t*⦄ ⟹
  V ∈ *mp-alloc-stm4-loopinv* (*mp-alloc-stm4-pre-precond-f Va t p*) *t p* ∩ ⦃´*i t* < *4*⦄ ⟹
   {*mp-alloc-stm4-while-precond4*
    (*mp-alloc-stm4-while-precond3*

(*mp-alloc-stm4-while-precond2*
(*mp-alloc-stm4-while-precond1 V t p*) *t p*) *t p*) *t p*} ∩
− {|*block-fits* (´*mem-pool-info p*) (´*block2 t*) (´*lsz t*)|} = {}
using *mp-alloc-stm4-blockfit1*[*of Va t p timeout sz V*]
int-empt1[*of* {*mp-alloc-stm4-while-precond4*
(*mp-alloc-stm4-while-precond3*
(*mp-alloc-stm4-while-precond2*
(*mp-alloc-stm4-while-precond1 V t p*) *t p*) *t p*) *t p*} − {|*block-fits* (´*mem-pool-info p*) (´*block2 t*) (´*lsz t*)|}]
apply *meson*

done

term *mp-alloc-precond2-1-1-loopinv-0 t p sz timeout* ∩ {|´*cur* = *Some t*|}

term *mp-alloc-precond2-1-1-loopinv-0 t p sz timeout* ∩ {|´*cur* = *Some t*|}
term *mp-alloc-stm4-loopinv* (*mp-alloc-stm4-pre-precond-f Va t p*) *t p* ∩ {|´*i t* < *4*|}
term {*mp-alloc-stm4-while-precond4*
(*mp-alloc-stm4-while-precond3*
(*mp-alloc-stm4-while-precond2*
(*mp-alloc-stm4-while-precond1 V t p*) *t p*) *t p*) *t p*} ∩
− {|*block-fits* (´*mem-pool-info p*) (´*block2 t*) (´*lsz t*)|}

lemma *mp-alloc-stm4-inv-mif-buf*: *buf* (*mem-pool-info Va pa*) = *buf* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *pa*)
apply(*simp add*:*mp-alloc-stm4-pre-precond-f-def*)
apply(*simp add*: *set-bit-def*)
done

lemma *mp-alloc-stm4-inv-mif-mxsz*: *max-sz* (*mem-pool-info Va pa*) = *max-sz* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *pa*)
apply(*simp add*:*mp-alloc-stm4-pre-precond-f-def*)
apply(*simp add*: *set-bit-def*)
done

lemma *mp-alloc-stm4-inv-mif-nmax*: *n-max* (*mem-pool-info Va pa*) = *n-max* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *pa*)
apply(*simp add*:*mp-alloc-stm4-pre-precond-f-def*)
apply(*simp add*: *set-bit-def*)
done

lemma *mp-alloc-stm4-inv-mif-nlvls*: *n-levels* (*mem-pool-info Va pa*) = *n-levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *pa*)
apply(*simp add*:*mp-alloc-stm4-pre-precond-f-def*)
apply(*simp add*: *set-bit-def*)
done

lemma *mp-alloc-stm4-inv-mif-len*: *length* (*levels* (*mem-pool-info Va pa*)) = *length*

(*levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *pa*))
  apply(*simp add*:*mp-alloc-stm4-pre-precond-f-def*)
  apply(*simp add*: *set-bit-def*)
done

lemma *mp-alloc-stm4-inv-bits-len*: *length* (*bits* (*levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *pa*) ! *ii*))
        = *length* (*bits* (*levels* (*mem-pool-info Va pa*) ! *ii*))

  apply(*simp add*:*mp-alloc-stm4-pre-precond-f-def*)
  apply(*case-tac p* $\neq$ *pa*) **apply**(*simp add*: *set-bit-def*)
  apply *simp*
  apply(*case-tac nat* (*from-l Va t*) = *ii*) **apply** *simp*
    apply (*metis set-bit-prev-len set-bit-prev-len2*)
  by (*metis set-bit-prev-len set-bit-prev-len2*)

lemma *inserts-comm*:
inserts ilst lst @ [*v*] = *inserts* (*ilst* @ [*v*]) *lst*
  by (*simp add*: *inserts-def*)

lemma *mp-alloc-stm4-while-isucc''*:
nat (*from-l Vb t* + *1*) < *length* (*levels* (*mem-pool-info Vb p*)) $\Longrightarrow$
  V $\in$ *mp-alloc-stm4-loopinv Vb t p* $\cap$ $\{\!|$´*i t* < *4*$|\!\}$ $\Longrightarrow$
  i V t > *0* $\Longrightarrow$
  $\Gamma \vdash_I$ *Some* (´*i* := ´*i*(*t* := *Suc* (´*i t*))) sat_p
    [{*mp-alloc-stm4-while-precond5*
      (*mp-alloc-stm4-while-precond4*
      (*mp-alloc-stm4-while-precond3*
      (*mp-alloc-stm4-while-precond2*
      (*mp-alloc-stm4-while-precond1 V t p*) *t p*) *t p*) *t p*) *t p*},
    {(*s*, *t*). *s* = *t*}, *UNIV*, *mp-alloc-stm4-loopinv Vb t p*]
apply(*rule Basic*)

apply *clarsimp*
apply(*rule conjI*)
  apply(*simp add*:*gvars-conf-stable-def gvars-conf-def*) **apply** *clarsimp*
  apply(*rule conjI*) **apply** *clarsimp*
    apply(*rule conjI*) **apply**(*simp add*:*append-free-list-def set-bit-def*)
    apply(*rule conjI*) **apply**(*simp add*:*append-free-list-def set-bit-def*)
    apply(*rule conjI*) **apply**(*simp add*:*append-free-list-def set-bit-def*)
    apply(*rule conjI*) **apply**(*simp add*:*append-free-list-def set-bit-def*)
    apply(*rule conjI*) **apply**(*simp add*:*append-free-list-def set-bit-def*)
      apply *clarsimp* **apply**(*simp add*:*append-free-list-def set-bit-def*)
      apply(*case-tac ia* $\neq$ *nat* (*from-l Vb t* + *1*)) **apply** *auto*[*1*]
       apply(*case-tac ia* < *length* (*levels* (*mem-pool-info Vb p*))) **apply** *fastforce*
apply *fastforce*
  apply *clarsimp*

apply(*rule conjI*) **apply**(*simp add*:*append-free-list-def set-bit-def*)
apply(*rule conjI*) **apply**(*simp add*:*append-free-list-def set-bit-def*)
apply(*rule conjI*) **apply**(*simp add*:*append-free-list-def set-bit-def*)
apply(*rule conjI*) **apply**(*simp add*:*append-free-list-def set-bit-def*)
apply(*rule conjI*) **apply**(*simp add*:*append-free-list-def set-bit-def*)
apply *clarsimp* **apply**(*simp add*:*append-free-list-def set-bit-def*)
apply(*rule conjI*)
apply(*simp add*:*append-free-list-def set-bit-def*)
apply(*rule conjI*)
apply(*simp add*:*append-free-list-def set-bit-def*)
apply(*rule conjI*)
apply *clarsimp*
apply(*simp add*:*append-free-list-def set-bit-def lvars-nochange-def*)
apply(*rule conjI*)
apply(*simp add*:*append-free-list-def set-bit-def*)
apply(*subgoal-tac length* (*levels* (*mem-pool-info Vb p*)) = *length* (*levels* (*mem-pool-info V p*)))
prefer *2* **apply**(*simp add*:*gvars-conf-stable-def gvars-conf-def*)
apply(*subgoal-tac nat* (*from-l Vb t* + *1*) < *length* (*levels* (*mem-pool-info Vb p*)))
prefer *2* **apply**(*simp add*:*inv-mempool-info-def*)
apply(*rule conjI*)
apply(*simp add*:*append-free-list-def set-bit-def*)
using *lst-updts-eq-updts-updt*[*of i V t bits* (*levels* (*mem-pool-info Vb p*) ! *nat* (*from-l Vb t* + *1*))
Suc (*bn Vb t* ∗ *4*) *FREE*]
apply (*simp add*: *semiring-normalization-rules*(*7*))
apply(*simp add*:*append-free-list-def set-bit-def*)
using *inserts-comm* **apply** *fast*

apply *fast* **using** *stable-id2* **apply** *fast* **using** *stable-id2* **apply** *fast*
done

lemma *mp-alloc-stm4-while-isucc*:
Va ∈ *mp-alloc-precond2-1-1-loopinv-0 t p sz timeout* ∩ {|´*cur* = *Some t*|} $\Longrightarrow$
V ∈ *mp-alloc-stm4-loopinv* (*mp-alloc-stm4-pre-precond-f Va t p*) *t p* ∩ {|´*i t* < *4*|} $\Longrightarrow$
$\Gamma \vdash_I$ *Some* (´*i* := ´*i*(*t* := *Suc* (´*i t*))) sat_p
[{*mp-alloc-stm4-while-precond5*
(*mp-alloc-stm4-while-precond4*
(*mp-alloc-stm4-while-precond3*
(*mp-alloc-stm4-while-precond2*
(*mp-alloc-stm4-while-precond1 V t p*) *t p*) *t p*) *t p*) *t p*},
{(*s*, *t*). *s* = *t*}, *UNIV*, *mp-alloc-stm4-loopinv* (*mp-alloc-stm4-pre-precond-f Va t p*) *t p*]
apply(*rule mp-alloc-stm4-while-isucc''*)
apply *clarsimp*
apply(*rule subst*[**where** *s*=*from-l Va t* **and** *t*=*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t*])
using *mp-alloc-stm4-pre-froml* **apply** *blast*

apply(*rule subst*[**where** *s*=*length* (*levels* (*mem-pool-info Va p*)) **and**
t=*length* (*levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*))])
using *mp-alloc-stm4-inv-mif-len* **apply** *blast*
apply(*subgoal-tac n-levels* (*mem-pool-info Va p*) = *length* (*levels* (*mem-pool-info Va p*)))
prefer *2* **apply**(*simp add*:*inv-def inv-mempool-info-def*) **apply** *metis*
apply *linarith*

apply *assumption*

apply *clarsimp*
done

lemma *mp-alloc-stm4-while-help*:
$Va \in$ *mp-alloc-precond2-1-1-loopinv-0 t p sz timeout* $\cap$ $\{\!|$´*cur* = *Some t*$|\!\}$ $\Longrightarrow$
$V \in$ *mp-alloc-stm4-loopinv* (*mp-alloc-stm4-pre-precond-f Va t p*) *t p* $\cap$ $\{\!|$´*i t* < *4*$|\!\}$ $\Longrightarrow$
$\Gamma \vdash_I$ *Some* (´*lbn* := ´*lbn*(*t* := *4* ∗ ´*bn t* + ´*i t*);;
´*lsz* := ´*lsz*(*t* := ´*lsizes t* ! *nat* (´*from-l t* + *1*));;
´*block2* := ´*block2*(*t* := ´*lsz t* ∗ ´*i t* + ´*blk t*);;
´*mem-pool-info* := *set-bit-free* ´*mem-pool-info p* (*nat* (´*from-l t* + *1*)) (´*lbn t*);;
IF block-fits (´*mem-pool-info p*) (´*block2 t*) (´*lsz t*) *THEN*
´*mem-pool-info* := ´*mem-pool-info*(*p* := *append-free-list* (´*mem-pool-info p*) (*nat* (´*from-l t* + *1*)) (´*block2 t*))
FI;;
´*i* := ´*i*(*t* := *Suc* (´*i t*)))
sat_p [{*V*}, {(*s*, *t*). *s* = *t*}, *UNIV*, *mp-alloc-stm4-loopinv* (*mp-alloc-stm4-pre-precond-f Va t p*) *t p*]
apply(*rule Seq*[**where** *mid*={*mp-alloc-stm4-while-precond5*
(*mp-alloc-stm4-while-precond4*
(*mp-alloc-stm4-while-precond3*
(*mp-alloc-stm4-while-precond2*
(*mp-alloc-stm4-while-precond1 V t p*) *t p*) *t p*) *t p*) *t p*}])
apply(*rule Seq*[**where** *mid*={*mp-alloc-stm4-while-precond4*
(*mp-alloc-stm4-while-precond3*
(*mp-alloc-stm4-while-precond2*
(*mp-alloc-stm4-while-precond1 V t p*) *t p*) *t p*) *t p*}])
apply(*rule Seq*[**where** *mid*={*mp-alloc-stm4-while-precond3*
(*mp-alloc-stm4-while-precond2*
(*mp-alloc-stm4-while-precond1 V t p*) *t p*) *t p*}])
apply(*rule Seq*[**where** *mid*={*mp-alloc-stm4-while-precond2*
(*mp-alloc-stm4-while-precond1 V t p*) *t p*}])
apply(*rule Seq*[**where** *mid*={*mp-alloc-stm4-while-precond1 V t p*}])

apply(*rule Basic*)
apply *simp* **apply** *simp* **apply**(*simp add*:*stable-def*) **apply**(*simp add*:*stable-def*)

apply(*rule Basic*)
 apply *simp* **apply** *simp* **apply**(*simp add*:*stable-def*) **apply**(*simp add*:*stable-def*)

apply(*rule Basic*)
 apply *simp* **apply** *simp* **apply**(*simp add*:*stable-def*) **apply**(*simp add*:*stable-def*)

apply(*rule Basic*)
 apply *simp* **apply** *simp* **apply**(*simp add*:*stable-def*) **apply**(*simp add*:*stable-def*)

apply(*rule Cond*)
  apply(*simp add*:*stable-def*)
  apply(*rule Basic*)
  using *mp-alloc-stm4-pre-in* **apply** *blast* **apply** *simp* **apply**(*simp add*:*stable-def*)
apply(*simp add*:*stable-def*)

  apply(*unfold Skip-def*)
  apply(*rule subst*[**where** *t*={*mp-alloc-stm4-while-precond4*
   (*mp-alloc-stm4-while-precond3*
   (*mp-alloc-stm4-while-precond2*
   (*mp-alloc-stm4-while-precond1 V t p*) *t p*) *t p*) *t p*} ∩
    − {|*block-fits* (´*mem-pool-info p*) (´*block2 t*) (´*lsz t*)|} **and** *s*={}])
  using *mp-alloc-stm4-blockfit*[*of Va t p timeout sz V*] **apply** *metis*
  using *Emptyprecond* **apply** *metis*
  apply *simp*

using *mp-alloc-stm4-while-isucc*[*of Va t p timeout sz V*] **apply** *fast*
done

lemma *mp-alloc-stm4-while-1*: {|*4* ≤ ´*i t*|} = − {|´*i t* < *4*|} **by** *auto*

term *mp-alloc-precond2-1-1-loopinv-0 t p sz timeout* ∩ {|´*cur* = *Some t*|}

lemma *mp-alloc-stm4-while*:
 Va ∈ *mp-alloc-precond2-1-1-loopinv-0 t p sz timeout* ∩ {|´*cur* = *Some t*|} ⟹
 Γ $\vdash_I$ *Some* (*WHILE* ´*i t* < *4 DO*
   ´*lbn* := ´*lbn* (*t* := *4* ∗ ´*bn t* + ´*i t*);;
   ´*lsz* := ´*lsz* (*t* := (´*lsizes t*) ! (*nat* (´*from-l t* + *1*)));;
   ´*block2* := ´*block2*(*t* := ´*lsz t* ∗ ´*i t* + ´*blk t*);;

   (∗ *set-free-bit*(*p*, *l* + *1*, *lbn*); ∗)
   ´*mem-pool-info* := *set-bit-free* ´*mem-pool-info p* (*nat* (´*from-l t* + *1*)) (´*lbn*
t);;

   IF block-fits (´*mem-pool-info p*) (´*block2 t*) (´*lsz t*) *THEN*

    (∗ *sys-dlist-append*(&*p*−>*levels*[*l* + *1*].*free-list*, *block2*); ∗)
    ´*mem-pool-info* := ´*mem-pool-info* (*p* :=

append-free-list (*´mem-pool-info p*) (*nat* (*´from-l t* + *1*)) (*´block2 t*))
FI;;
´i := *´i*(*t* := *Suc* (*´i t*))
OD) sat_p [*mp-alloc-stm4-loopinv* (*mp-alloc-stm4-pre-precond-f Va t p*) *t p*, {(*s*, *t*). *s* = *t*}, *UNIV*,
mp-alloc-stm4-loopinv (*mp-alloc-stm4-pre-precond-f Va t p*) *t p* ∩ ⦃*´i t* ≥ *4*⦄]
apply(*rule While*)
apply(*simp add*:*stable-def*)
apply(*rule subst*[**where** *t*=− ⦃*´i t* < *4*⦄ **and** *s*=⦃*4* ≤ *´i t*⦄]) **using** *mp-alloc-stm4-while-1*[*of t*] **apply** *simp*
apply *simp*
apply(*simp add*:*stable-def*)
using *mp-alloc-stm4-while-help*[*of Va t p timeout sz*]
Allprecond[*of mp-alloc-stm4-loopinv* (*mp-alloc-stm4-pre-precond-f Va t p*) *t p* ∩⦃*´i t* < *4*⦄
Some (*´lbn* := *´lbn*(*t* := *4* ∗ *´bn t* + *´i t*);;
´lsz := *´lsz*(*t* := *´lsizes t* ! *nat* (*´from-l t* + *1*));;
´block2 := *´block2*(*t* := *´lsz t* ∗ *´i t* + *´blk t*);;
´mem-pool-info := *set-bit-free ´mem-pool-info p* (*nat* (*´from-l t* + *1*)) (*´lbn t*);;
IF block-fits (*´mem-pool-info p*) (*´block2 t*)
(*´lsz t*) *THEN ´mem-pool-info* := *´mem-pool-info*
(*p* := *append-free-list* (*´mem-pool-info p*) (*nat* (*´from-l t* + *1*)) (*´block2 t*)) *FI*;;
´i := *´i*(*t* := *Suc* (*´i t*))) {(*s*, *t*). *s* = *t*} *UNIV*
mp-alloc-stm4-loopinv (*mp-alloc-stm4-pre-precond-f Va t p*) *t p*] **apply** *clarsimp*
apply *force*
done

lemma *mp-alloc-stm4-pre-precond-f-in-mp-alloc-stm4-loopinv*:
mp-alloc-stm4-pre-precond-f Va t p ∈ *mp-alloc-stm4-loopinv* (*mp-alloc-stm4-pre-precond-f Va t p*) *t p*
apply(*subgoal-tac i* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* = *1*)
using *in-mp-alloc-stm4-loopinv* **apply** *meson*
apply(*simp add*:*mp-alloc-stm4-pre-precond-f-def*)
done

lemma *mp-alloc-stm4-mempools*: (*x*, *mp-alloc-stm4-pre-precond-f Va t p*) ∈ *gvars-conf-stable* ⟹ *mem-pools x* = *mem-pools Va*
by (*simp add*:*mp-alloc-stm4-pre-precond-f-def gvars-conf-stable-def gvars-conf-def*)

lemma *mp-alloc-stm4-mempools2*: *mem-pools* (*mp-alloc-stm4-pre-precond-f Va t p*) = *mem-pools x* ⟹ *mem-pools x* = *mem-pools Va*
by (*simp add*:*mp-alloc-stm4-pre-precond-f-def*)

lemma *mp-alloc-stm4-inv-cur*:
cur Va = *Some t* $\Longrightarrow$ $\forall$ *ta*. (*t* = *ta*) = (*thd-state Va ta* = *RUNNING*) $\Longrightarrow$
(*cur* (*mp-alloc-stm4-pre-precond-f Va t p*) = *Some ta*) = (*thd-state* (*mp-alloc-stm4-pre-precond-f Va t p*) *ta* = *RUNNING*)
by (*simp add*:*mp-alloc-stm4-pre-precond-f-def*)

lemma *mp-alloc-stm4-inv-thd-state*: (*x*, *mp-alloc-stm4-pre-precond-f Va t p*) $\in$ *gvars-conf-stable* $\Longrightarrow$
thd-state x = *thd-state* (*mp-alloc-stm4-pre-precond-f Va t p*) $\Longrightarrow$
$\forall$ *pa*. *pa* $\neq$ *p* $\longrightarrow$ *mem-pool-info x pa* = *mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *pa* $\Longrightarrow$
wait-q (*mem-pool-info x p*) = *wait-q* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) $\Longrightarrow$
inv-thd-waitq Va $\Longrightarrow$ *inv-thd-waitq x*
apply(*subgoal-tac* $\forall$ *q*$\in$*mem-pools x*. *wait-q* (*mem-pool-info x q*)
= *wait-q* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *q*))
prefer *2* **apply** *clarify* **apply**(*case-tac q* = *p*) **apply** *simp* **apply** *simp*
apply(*subgoal-tac* $\forall$ *q*$\in$*mem-pools Va*. *wait-q* (*mem-pool-info Va q*)
= *wait-q* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *q*))
prefer *2* **apply** *clarify* **apply**(*simp add*:*mp-alloc-stm4-pre-precond-f-def*)
apply(*simp add*: *set-bit-def*)
apply(*subgoal-tac thd-state* (*mp-alloc-stm4-pre-precond-f Va t p*) = *thd-state Va*)
prefer *2* **apply** (*simp add*:*mp-alloc-stm4-pre-precond-f-def*)
apply(*subgoal-tac mem-pools x* = *mem-pools Va*)
prefer *2* **apply** (*simp add*:*mp-alloc-stm4-pre-precond-f-def gvars-conf-stable-def gvars-conf-def*)
apply(*simp add*:*inv-thd-waitq-def*)
apply *clarify*
by *blast*

lemma *inv-mpinfo-inv-mpinfo-stm4*:
inv-mempool-info Va $\Longrightarrow$ *inv-mempool-info* (*mp-alloc-stm4-pre-precond-f Va t p*)
apply(*simp add*:*inv-mempool-info-def mp-alloc-stm4-pre-precond-f-def*)
apply(*simp add*:*Let-def*) **apply** *clarify*
apply(*rule conjI*)
apply(*simp add*: *set-bit-def*) **apply** *auto*[*1*]
apply(*rule conjI*)
apply(*simp add*: *set-bit-def*)
apply(*rule conjI*) **apply** *clarify* **apply** *auto*[*1*] **apply** *clarify* **apply** *auto*[*1*]
apply(*rule conjI*) **apply**(*simp add*: *set-bit-def*) **apply** *auto*[*1*]
apply(*rule conjI*) **apply**(*simp add*: *set-bit-def*) **apply** *auto*[*1*]
apply(*rule conjI*) **apply**(*simp add*: *set-bit-def*) **apply** *auto*[*1*]
apply *clarify* **apply**(*rename-tac pa ii*)
apply(*subgoal-tac length* (*bits* (*levels* ((*set-bit-divide* (*mem-pool-info Va*) *p* (*nat* (*from-l Va t*))
(*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*)))) *pa*) !
ii)) = *length* (*bits* (*levels* (*mem-pool-info Va pa*) ! *ii*)))

prefer *2* **apply** (*metis* (*no-types*, *lifting*) *fun-upd-apply set-bit-def set-bit-prev-len set-bit-prev-len2*)
apply(*subgoal-tac length* (*bits* (*levels* (*set-bit-allocating* (*set-bit-divide* (*mem-pool-info Va*) *p* (*nat* (*from-l Va t*)) (*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*)))) *p* (*nat* (*from-l Va t* + *1*)) (*4* ∗ *block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*))) *pa*) ! *ii*)) = *length* (*bits* (*levels* ((*set-bit-divide* (*mem-pool-info Va*) *p* (*nat* (*from-l Va t*)) (*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*)))) *pa*) ! *ii*)))
prefer *2* **apply**(*case-tac ii* = *nat* (*from-l Va t* + *1*))
using *set-bit-prev-len set-bit-def* **apply** *auto*[*1*]
using *set-bit-def* **apply** *auto*[*1*]
apply(*subgoal-tac n-max* (*set-bit-allocating* (*set-bit-divide* (*mem-pool-info Va*) *p* (*nat* (*from-l Va t*)) (*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*)))) *p* (*nat* (*from-l Va t* + *1*)) (*4* ∗ *block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*))) *pa*) ∗ *4* ˆ *ii* = *n-max* (*mem-pool-info Va pa*) ∗ *4* ˆ *ii*)
prefer *2* **apply**(*simp add*: *set-bit-def*)
apply(*subgoal-tac length* (*levels* (*set-bit-allocating* (*set-bit-divide* (*mem-pool-info Va*) *p* (*nat* (*from-l Va t*)) (*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*)))) *p* (*nat* (*from-l Va t* + *1*)) (*4* ∗ *block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*))) *pa*)) = *length* (*levels* (*mem-pool-info Va pa*)))
prefer *2* **apply**(*simp add*: *set-bit-def*)
apply *metis*
done

lemma *mp-alloc-stm4-inv-mempool-info*:
(*x*, *mp-alloc-stm4-pre-precond-f Va t p*) ∈ *gvars-conf-stable* ⟹ *inv-mempool-info Va* ⟹ *inv-mempool-info x*
apply(*subgoal-tac inv-mempool-info* (*mp-alloc-stm4-pre-precond-f Va t p*))
prefer *2* **using** *inv-mpinfo-inv-mpinfo-stm4* **apply** *simp*
using *gvars-conf-stb-inv-mpinf* **apply** *simp*
done

lemma *mp-alloc-stm4-lvl-len*:
p ∈ *mem-pools Va* ⟹ (*x*, *mp-alloc-stm4-pre-precond-f Va t p*) ∈ *gvars-conf-stable* ⟹

length (*levels* (*mem-pool-info x pa*)) = *length* (*levels* (*mem-pool-info Va pa*))
apply(*simp add*:*mp-alloc-stm4-pre-precond-f-def gvars-conf-stable-def gvars-conf-def*)

apply(*simp add*: *set-bit-def*)
done

lemma *mp-alloc-stm4-maxsz*:
$p \in$ *mem-pools Va* $\Longrightarrow$ (x, *mp-alloc-stm4-pre-precond-f Va t p*) $\in$ *gvars-conf-stable* $\Longrightarrow$
max-sz (*mem-pool-info x pa*) = *max-sz* (*mem-pool-info Va pa*)
apply(*simp add*:*mp-alloc-stm4-pre-precond-f-def gvars-conf-stable-def gvars-conf-def*)

apply(*simp add*: *set-bit-def*)
done

lemma *mp-alloc-stm4-buf*:
$p \in$ *mem-pools Va* $\Longrightarrow$ (x, *mp-alloc-stm4-pre-precond-f Va t p*) $\in$ *gvars-conf-stable* $\Longrightarrow$
buf (*mem-pool-info x pa*) = *buf* (*mem-pool-info Va pa*)
apply(*simp add*:*mp-alloc-stm4-pre-precond-f-def gvars-conf-stable-def gvars-conf-def*)

apply(*simp add*: *set-bit-def*)
done

lemma *mp-alloc-stm4-pres-mpinfo*:
pa $\neq$ *p* $\longrightarrow$ *mem-pool-info Va pa* = *mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *pa*
apply(*simp add*:*mp-alloc-stm4-pre-precond-f-def set-bit-def*)
done

lemma *mp-alloc-stm4-froml*:
from-l x = *from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) $\Longrightarrow$
from-l x = *from-l Va*
apply(*simp add*:*mp-alloc-stm4-pre-precond-f-def gvars-conf-stable-def gvars-conf-def*)

done

lemma *mp-alloc-stm4-pre-precond-f-lvars-nochange*:
$t' \neq t \Longrightarrow$ *lvars-nochange* t' *Va* (*mp-alloc-stm4-pre-precond-f Va t p*)
apply(*simp add*:*lvars-nochange-def mp-alloc-stm4-pre-precond-f-def*)
done

lemma *mp-alloc-stm4-pre-precond-f-tick*:
tick Va = *tick* (*mp-alloc-stm4-pre-precond-f Va t p*)
by(*simp add*:*mp-alloc-stm4-pre-precond-f-def*)

lemma *mp-alloc-stm4-pre-precond-f-def-frnode*:
freeing-node Va = *freeing-node* (*mp-alloc-stm4-pre-precond-f Va t p*)

by(*simp add*:*mp-alloc-stm4-pre-precond-f-def*)

lemma *mp-alloc-stm4-pre-precond-f-mpls*:
$p \in$ *mem-pools Va* $\Longrightarrow$ (x, *mp-alloc-stm4-pre-precond-f Va t p*) $\in$ *gvars-conf-stable*
$\Longrightarrow p \in$ *mem-pools x*
apply(*simp add*:*mp-alloc-stm4-pre-precond-f-def gvars-conf-stable-def gvars-conf-def*)
done

lemma *mp-alloc-stm4-pre-precond-f-rf*:
rf (*mp-alloc-stm4-pre-precond-f Va t p*) *t* $\Longrightarrow$ *rf Va t*
by(*simp add*:*mp-alloc-stm4-pre-precond-f-def*)

lemma *mp-alloc-stm4-pre-precond-f-ret*:
mempoolalloc-ret Va = *mempoolalloc-ret* (*mp-alloc-stm4-pre-precond-f Va t p*)
by(*simp add*:*mp-alloc-stm4-pre-precond-f-def*)

lemma *mp-alloc-stm4-pre-precond-f-tmout*:
tmout Va = *tmout* (*mp-alloc-stm4-pre-precond-f Va t p*)
by(*simp add*:*mp-alloc-stm4-pre-precond-f-def*)

lemma *mp-alloc-stm4-pre-precond-f-lsz*:
lsizes (*mp-alloc-stm4-pre-precond-f Va t p*) = *lsizes Va*
by(*simp add*:*mp-alloc-stm4-pre-precond-f-def*)

lemma *mp-alloc-stm4-pre-precond-f-allocl*:
alloc-l (*mp-alloc-stm4-pre-precond-f Va t p*) = *alloc-l Va*
by(*simp add*:*mp-alloc-stm4-pre-precond-f-def*)

lemma *mp-alloc-stm4-pre-precond-f-froml*:
from-l (*mp-alloc-stm4-pre-precond-f Va t p*) = *from-l Va*
by(*simp add*:*mp-alloc-stm4-pre-precond-f-def*)

lemma *mp-alloc-stm4-pre-precond-f-freel*:
free-l (*mp-alloc-stm4-pre-precond-f Va t p*) = *free-l Va*
by(*simp add*:*mp-alloc-stm4-pre-precond-f-def*)

lemma *mp-alloc-stm4-pre-precond-f-blk*:
blk (*mp-alloc-stm4-pre-precond-f Va t p*) = *blk Va*
by(*simp add*:*mp-alloc-stm4-pre-precond-f-def*)

lemma *same-level-set-bit-l*:$i1 \neq i' \Longrightarrow$
levels ((*set-bit mp-info p* i' j' *b*) *p*)!*i1* = *levels* (*mp-info p*)!*i1*
unfolding *set-bit-def*
by *auto*

lemma *same-bit-set-bit-l*:$i1 \neq i' \Longrightarrow$
bits (*levels* ((*set-bit mp-info p* i' j' *b*) *p*)!*i1*) = *bits* (*levels* (*mp-info p*)!*i1*)

using *same-level-set-bit-l*
by *auto*

lemma *same-bit-set-bit-j*:
$j1 \neq j' \Longrightarrow$
bits (*levels* ((*set-bit mp-info p i' j' b*) *p*)!*i1*)!*j1* = *bits* (*levels* (*mp-info p*)!*i1*)!*j1*
apply(*simp add*: *set-bit-get-bit-neq set-bit-def*)
by (*metis* (*no-types*, *lifting*) *Mem-pool-lvl.select-convs*(*1*) *Mem-pool-lvl.surjective Mem-pool-lvl.update-convs*(*1*)
list-update-beyond not-less nth-list-update-eq nth-list-update-neq)

lemma *set-bit-set-bit*:
$l1 \neq l2 \vee b1 \neq b2 \Longrightarrow$
set-bit-s (*set-bit-s Va p l1 b1 st1*) *p l2 b2 st2* =
set-bit-s ((*set-bit-s Va p l2 b2 st2*)) *p l1 b1 st1*
unfolding *set-bit-s-def set-bit-def*
apply *auto*
apply (*cases b1*=*b2*) **apply** *auto*
apply (*simp add*: *list-update-swap*)
apply (*simp add*: *list-update-swap*)
apply (*cases l1*=*l2*) **apply** *auto*
apply (*cases l1*< *length* (*levels* (*mem-pool-info Va p*)))
by (*auto simp add*: *list-update-swap*)

lemma *get-bit-set-bit-set-bit*:
assumes *a0*:$l1 \neq l2 \vee b1 \neq b2$ **and**
a1:*l1* < *length* (*levels* ((*mem-pool-info Va*) *p*)) **and**
a2:*b1* < *length* (*bits* (*levels* ((*mem-pool-info Va*) *p*) ! *l1*))
shows *get-bit-s* (*set-bit-s* (*set-bit-s Va p l1 b1 st1*) *p l2 b2 st2*) *p l1 b1* = *st1*
proof−
have *a1'*:*l1* < *length* (*levels* ((*mem-pool-info* (*set-bit-s Va p l2 b2 st2*)) *p*))
using *a1* **unfolding** *set-bit-s-def set-bit-def* **by** *auto*
have *a2'*:*b1* < *length* (*bits* (*levels* ((*mem-pool-info* (*set-bit-s Va p l2 b2 st2*)) *p*) ! *l1*))
using *a2* **unfolding** *set-bit-s-def set-bit-def* **apply** *auto*
by (*metis* (*no-types*, *lifting*) *Mem-pool-lvl.select-convs*(*1*) *Mem-pool-lvl.surjective Mem-pool-lvl.update-convs*(*1*) *a1*
length-list-update nth-list-update-eq nth-list-update-neq)
show *?thesis* **using** *set-bit-get-bit-eq2*[*OF a1' a2'*] *set-bit-set-bit*[*OF a0*] **unfolding** *set-bit-s-def* **by** *auto*
qed

lemma *mp-alloc-stm4-pre-precond-f-same-bits*:**assumes**
a0:*i1*=(*nat* (*from-l Va t*)) **and**
a1:*i2* = (*nat* (*from-l Va t* + *1*)) **and**
a2:*Va'* = *mp-alloc-stm4-pre-precond-f Va t p*
shows $\forall i\ j.\ \neg((i=i1) \vee (i=i2)) \longrightarrow$

get-bit (mem-pool-info Va′) p i j = get-bit (mem-pool-info Va) p i j
using *a0 a1 a2 set-bit-get-bit-neq* **unfolding** *mp-alloc-stm4-pre-precond-f-def*
by *auto*

lemma *same-bit-mp-alloc-stm4-pre-precond-f*:
i1=(nat (from-l Va t)) $\Longrightarrow$
j1= (block-num (mem-pool-info Va p) (blk Va t) (lsizes Va t ! nat (from-l Va t))) $\Longrightarrow$
i2 = (nat (from-l Va t + 1)) $\Longrightarrow$
j2 = (4∗block-num (mem-pool-info Va p) (blk Va t) (lsizes Va t ! nat (from-l Va t))) $\Longrightarrow$
Va′ = mp-alloc-stm4-pre-precond-f Va t p $\Longrightarrow$
∀ i j. ¬((i=i1 ∧ j=j1) ∨(i=i2 ∧ j=j2)) $\longrightarrow$
get-bit (mem-pool-info Va′) p i j = get-bit (mem-pool-info Va) p i j
using *set-bit-get-bit-neq*
apply *(auto simp add:mp-alloc-stm4-pre-precond-f-def)*
by *metis+*

lemma *same-bit-mp-alloc-stm4-pre-precond-f1*:
assumes
a1:¬((l=(nat (from-l Va t)) ∧ b=(block-num (mem-pool-info Va p) (blk Va t) (lsizes Va t ! nat (from-l Va t)))) ∨
(l=(nat (from-l Va t + 1)) ∧ b=(4∗block-num (mem-pool-info Va p) (blk Va t) (lsizes Va t ! nat (from-l Va t)))))
shows *(get-bit-s Va p l b = get-bit-s (mp-alloc-stm4-pre-precond-f Va t p) p l b)*
using *a1 same-bit-mp-alloc-stm4-pre-precond-f* **by** *metis*

lemma *same-bit-mp-alloc-stm4-pre-precond-f11*:
assumes *a0:(l=(nat (from-l Va t)) ∧ b=(block-num (mem-pool-info Va p) (blk Va t) (lsizes Va t ! nat (from-l Va t))))* **and**
a1:l ≥ length (levels (mem-pool-info Va p)) ∨
b≥length (bits (levels (mem-pool-info Va p) ! l))
shows *get-bit-s Va p l b = get-bit-s (mp-alloc-stm4-pre-precond-f Va t p) p l b*
using *a0 a1* **unfolding** *mp-alloc-stm4-pre-precond-f-def set-bit-def*
apply *auto*
by *(metis (no-types, lifting) Mem-pool-lvl.simps(1) Mem-pool-lvl.simps(4) Mem-pool-lvl.surjective*

list-update-beyond not-less nth-list-update-eq nth-list-update-neq)

lemma *same-bit-mp-alloc-stm4-pre-precond-f12*:
assumes *a0:(l=(nat (from-l Va t + 1)) ∧ b=4∗(block-num (mem-pool-info Va p) (blk Va t) (lsizes Va t ! nat (from-l Va t))))* **and**
a1:l ≥ length (levels (mem-pool-info Va p)) ∨
b≥length (bits (levels (mem-pool-info Va p) ! l))
shows *get-bit-s Va p l b = get-bit-s (mp-alloc-stm4-pre-precond-f Va t p) p l b*
using *a0 a1* **unfolding** *mp-alloc-stm4-pre-precond-f-def set-bit-def*
apply *auto*

apply (*metis list-update-beyond nth-list-update-neq*)
by (*smt Mem-pool-lvl.simps*(*1*) *Mem-pool-lvl.simps*(*4*) *Mem-pool-lvl.surjective length-list-update list-update-beyond*
not-less nth-list-update-eq nth-list-update-neq)

lemma *same-bit-mp-alloc-stm4-pre-precond-f2*:
assumes *a1*:*l* ≥ *length* (*levels* (*mem-pool-info Va p*)) ∨
b≥*length* (*bits* (*levels* (*mem-pool-info Va p*) ! *l*))
shows *get-bit-s Va p l b* = *get-bit-s* (*mp-alloc-stm4-pre-precond-f Va t p*) *p l b*
apply (*cases* ¬ ((*l*=(*nat* (*from-l Va t* + *1*)) ∧
b=*4*∗(*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*)))) ∨
(*l*=(*nat* (*from-l Va t*)) ∧ *b*=(*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*))))))
using *same-bit-mp-alloc-stm4-pre-precond-f1* **apply** *fast*
using *a1* **by** (*auto intro*: *same-bit-mp-alloc-stm4-pre-precond-f11 same-bit-mp-alloc-stm4-pre-precond-f12*)

lemma *same-bit-mp-alloc-stm4-pre-precond-divided*:
assumes *a0*:(*l*=(*nat* (*from-l Va t*)) ∧ *b*=(*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*)))) **and**
a1:*l* < *length* (*levels* (*mem-pool-info Va p*)) **and**
a2:*b*<*length* (*bits* (*levels* (*mem-pool-info Va p*) ! *l*)) **and**
a3:(*from-l Va t*) ≥ *0*
shows *get-bit-s* (*mp-alloc-stm4-pre-precond-f Va t p*) *p l b* = *DIVIDED*
proof−
have *l* ≠ *nat* (*from-l Va t* + *1*) ∨ *b* ≠ *4*∗*b* **using** *a0 a3* **by** *fastforce*
then show *?thesis* **using** *a0 a1 a2 a3 set-bit-get-bit-eq2*
unfolding *mp-alloc-stm4-pre-precond-f-def*
using *set-bit-get-bit-neq* **by** *auto*
qed

lemma *same-bit-mp-alloc-stm4-pre-precond-allocating*:
assumes *a0*:(*l*=(*nat* (*from-l Va t* + *1*)) ∧ *b*=*4*∗(*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*)))) **and**
a1:*l* < *length* (*levels* (*mem-pool-info Va p*)) **and**
a2:*b*<*length* (*bits* (*levels* (*mem-pool-info Va p*) ! *l*)) **and**
a3:(*from-l Va t*) ≥ *0*
shows *get-bit-s* (*mp-alloc-stm4-pre-precond-f Va t p*) *p l b* = *ALLOCATING*
proof−
let *?Va* = *set-bit-s Va p* (*nat* (*from-l Va t*))
(*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*))) *DIVIDED*
have *a1'*:*l* < *length* (*levels* (*mem-pool-info ?Va p*))
using *a1* **unfolding** *set-bit-s-def set-bit-def* **by** *auto*
moreover have *a2'*:*b*<*length* (*bits* (*levels* (*mem-pool-info ?Va p*) ! *l*))
using *a2* **unfolding** *set-bit-s-def set-bit-def*
by (*simp add*: *a0 a3 eq-nat-nat-iff*)
ultimately show *?thesis*

using *a0 set-bit-get-bit-eq2 a3*
unfolding *mp-alloc-stm4-pre-precond-f-def set-bit-s-def*
using *set-bit-get-bit-eq* **by** *auto*
qed

lemma *eq-free-list-set-bit-l*:
free-list (*levels* ((*set-bit mp-info p i′ j′ b*) *p*)!*i1*) = *free-list* (*levels* (*mp-info p*)!*i1*)
unfolding *set-bit-def*
apply (*cases i′*< *length* (*levels* (*mp-info p*)); *auto*)
by (*metis* (*no-types*, *lifting*) *Mem-pool-lvl.select-convs*(*2*) *Mem-pool-lvl.surjective Mem-pool-lvl.update-convs*(*1*) *nth-list-update-eq nth-list-update-neq*)

lemma *eq-free-list-mp-alloc-stm4-pre-precond-f*:
free-list (*levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) ! *l*) =
free-list (*levels* (*mem-pool-info Va p*) ! *l*)
unfolding *mp-alloc-stm4-pre-precond-f-def* **using** *eq-free-list-set-bit-l*
by *auto*

lemma *mp-alloc-stm4-pre-precond-f-i*:(*i* (*mp-alloc-stm4-pre-precond-f Va t p*)) *t* = *Suc 0* ∧
(∀ *t′*. *t′*≠*t* ⟶ (*i* (*mp-alloc-stm4-pre-precond-f Va t p*)) *t′* = (*i Va*) *t′*)
unfolding *mp-alloc-stm4-pre-precond-f-def* **by** *force*

lemma *mp-alloc-stm4-pre-precond-f-bn*:
(*bn* (*mp-alloc-stm4-pre-precond-f Va t p*)) *t* =
block-num (*mem-pool-info Va p*) (*blk Va t*) ((*lsizes Va t*)!(*nat* (*from-l Va t*)))
unfolding *mp-alloc-stm4-pre-precond-f-def* **by** *force*

lemma *mp-alloc-stm4-pre-precond-f-allocating*:
(*allocating-node* (*mp-alloc-stm4-pre-precond-f Va t p*)) *t* =
Some (|*pool* = *p*, *level* = *nat* (*from-l Va t* + *1*),
block = *4* ∗ *block-num* (*mem-pool-info Va p*) (*blk Va t*) ((*lsizes Va t*)!(*nat* (*from-l Va t*))),
data = *blk Va t* |)
unfolding *mp-alloc-stm4-pre-precond-f-def*
by *auto*

lemma *get-bit-x-l-b*:
assumes *a0*:(*l*=(*nat* (*from-l* (*Va*::*State*) *t*)) ∧ *b*=(*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*)))) **and**
a1:(*from-l Va t*) ≥ *0* **and**
a2:∀ *jj*. *jj* ≠ *nat* (*from-l Va t* + *1*) ⟶
levels (*mem-pool-info x p*) ! *jj* =
levels (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) ! *jj* **and**
a4:*l* < *length* (*levels* (*mem-pool-info Va p*)) **and**
a5:*b*<*length* (*bits* (*levels* (*mem-pool-info Va p*) ! *l*))
shows *get-bit-s x p l b* = *DIVIDED*

using *a0 a2 a1 a4 a5 same-bit-mp-alloc-stm4-pre-precond-divided* **by** *auto*

lemma *get-bit-x-l1-b4*:
assumes *a0*:(l=(*nat* (*from-l* (*Va*::*State*) t + *1*)) ∧ b=*4*∗(*block-num* (*mem-pool-info* *Va* p) (*blk* *Va* t) (*lsizes* *Va* t ! *nat* (*from-l* *Va* t)))) **and**
a1:(*from-l* *Va* t) ≥ *0* **and**
a3:*bits* (*levels* (*mem-pool-info* x p) ! *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f* *Va* t p) t + *1*)) =
list-updates-n
(*bits* (*levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f* *Va* t p) p) !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f* *Va* t p) t + *1*)))
(*Suc* (*bn* (*mp-alloc-stm4-pre-precond-f* *Va* t p) t ∗ *4*)) *3* *FREE* **and**
a4:l < *length* (*levels* (*mem-pool-info* *Va* p)) **and**
a5:b<*length* (*bits* (*levels* (*mem-pool-info* *Va* p) ! l))
shows *get-bit-s* x p l b = *ALLOCATING*
using *a0 a1 a3 a4 a5 same-bit-mp-alloc-stm4-pre-precond-allocating*
mp-alloc-stm4-pre-precond-f-bn
mp-alloc-stm4-pre-froml
by (*metis lessI list-updates-n-neq mult.commute*)

lemma *get-bit-x-l1-b41*:
assumes *a0*:l=(*nat* (*from-l* (*Va*::*State*) t + *1*)) ∧
(b=*Suc*(*4*∗(*block-num* (*mem-pool-info* *Va* p) (*blk* *Va* t) (*lsizes* *Va* t !
nat (*from-l* *Va* t)))) ∨
b=*Suc*(*Suc*(*4*∗(*block-num* (*mem-pool-info* *Va* p) (*blk* *Va* t) (*lsizes* *Va*
t ! *nat* (*from-l* *Va* t))))) ∨
b=*Suc*(*Suc*(*Suc*(*4*∗(*block-num* (*mem-pool-info* *Va* p) (*blk* *Va* t) (*lsizes*
Va t ! *nat* (*from-l* *Va* t))))))) **and**
a1:(*from-l* *Va* t) ≥ *0* **and**
a2:∀ *jj*. *jj* ≠ *nat* (*from-l* *Va* t + *1*) ⟶
levels (*mem-pool-info* x p) ! *jj* =
levels (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f* *Va* t p) p) ! *jj* **and**
a3:*bits* (*levels* (*mem-pool-info* x p) ! *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f* *Va* t p) t + *1*)) =
list-updates-n
(*bits* (*levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f* *Va* t p) p) !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f* *Va* t p) t + *1*)))
(*Suc* (*bn* (*mp-alloc-stm4-pre-precond-f* *Va* t p) t ∗ *4*)) *3* *FREE* **and**
a4:l < *length* (*levels* (*mem-pool-info* *Va* p)) **and**
a5: b<*length* (*bits* (*levels* (*mem-pool-info* *Va* p) ! l))
shows *get-bit-s* x p l b = *FREE*
using *a0 a1 a3 a4 a5*
apply *auto*
using *mp-alloc-stm4-pre-precond-f-bn*
mp-alloc-stm4-pre-froml mp-alloc-stm4-inv-bits-len Suc-numeral add-2-eq-Suc
add-Suc-right
by (*smt add.commute lessI less-add-same-cancel2*

list-updates-n-eq mult.commute nat-less-le neq0-conv not-le semiring-norm(5))+

lemma *get-bit-x-l1-b41'*:
assumes *a0:l=(nat (from-l (Va::State) t + 1))* ∧
(b=(4∗(block-num (mem-pool-info Va p) (blk Va t) (lsizes Va t ! nat (from-l Va t)))) + 1 ∨
b=(4∗(block-num (mem-pool-info Va p) (blk Va t) (lsizes Va t ! nat (from-l Va t))))+2 ∨
b=(4∗(block-num (mem-pool-info Va p) (blk Va t) (lsizes Va t ! nat (from-l Va t))))+3) **and**
a1:(from-l Va t) ≥ *0* **and**
a2:∀ *jj. jj* ≠ *nat (from-l Va t + 1)* ⟶
levels (mem-pool-info x p) ! jj =
levels (mem-pool-info (mp-alloc-stm4-pre-precond-f Va t p) p) ! jj **and**
a3:bits (levels (mem-pool-info x p) ! nat (from-l (mp-alloc-stm4-pre-precond-f Va t p) t + 1)) =
list-updates-n
(bits (levels (mem-pool-info (mp-alloc-stm4-pre-precond-f Va t p) p) !
nat (from-l (mp-alloc-stm4-pre-precond-f Va t p) t + 1)))
(Suc (bn (mp-alloc-stm4-pre-precond-f Va t p) t ∗ 4)) 3 FREE **and**
a4:l < length (levels (mem-pool-info Va p)) **and**
a5: b<length (bits (levels (mem-pool-info Va p) ! l))
shows *get-bit-s x p l b = FREE*
using *a0 a1 a3 a4 a5*
apply *auto*
using *mp-alloc-stm4-pre-precond-f-bn*
mp-alloc-stm4-pre-froml mp-alloc-stm4-inv-bits-len
apply *(metis add.right-neutral less-not-refl list-updates-n-eq mult.commute nat-add-left-cancel-less not-less zero-less-numeral)*
by *(smt add.commute add-Suc less-Suc-eq less-add-same-cancel2 list-updates-n-eq mp-alloc-stm4-inv-bits-len mp-alloc-stm4-pre-froml mp-alloc-stm4-pre-precond-f-bn mult.commute nat-less-le numeral-3-eq-3) +*

lemma *get-bit-x-stm4-pre-eq*:
assumes
a0:∀ *jj. jj* ≠ *nat (from-l (Va::State) t + 1)* ⟶
levels (mem-pool-info x p) ! jj =
levels (mem-pool-info (mp-alloc-stm4-pre-precond-f Va t p) p) ! jj **and**
a1:bits (levels (mem-pool-info x p) ! nat (from-l (mp-alloc-stm4-pre-precond-f Va t p) t + 1)) =
list-updates-n
(bits (levels (mem-pool-info (mp-alloc-stm4-pre-precond-f Va t p) p) !
nat (from-l (mp-alloc-stm4-pre-precond-f Va t p) t + 1)))
(Suc (bn (mp-alloc-stm4-pre-precond-f Va t p) t ∗ 4)) 3 FREE **and**
a2:l = nat (from-l (mp-alloc-stm4-pre-precond-f Va t p) t + 1) **and**
a3: b1 = (Suc (bn (mp-alloc-stm4-pre-precond-f Va t p) t ∗ 4)) **and**
a4:b2 = Suc (Suc (bn (mp-alloc-stm4-pre-precond-f Va t p) t ∗ 4)) **and**
a5:b3 = Suc (Suc (Suc (bn (mp-alloc-stm4-pre-precond-f Va t p) t ∗ 4)))
shows ∀ *i j.* ¬*((i=l* ∧ *j=b1)* ∨*(i=l* ∧ *j=b2)* ∨ *(i=l* ∧ *j=b3))* ⟶

get-bit-s x p i j = get-bit-s (mp-alloc-stm4-pre-precond-f Va t p) p i j
using *a0 a1 a2 a3 a4 a5* **apply** *clarsimp*
apply (*auto simp add*: *mp-alloc-stm4-pre-precond-f-froml*)
by (*metis* (*no-types*) *add-2-eq-Suc′ add-Suc-right eval-nat-numeral*(*3*) *less-Suc-eq list-updates-n-neq not-less*)

lemma *same-bit-mp-alloc-stm4-pre-precond-f-x*:
assumes *a0*:∀ *jj*. *jj* ≠ *nat* (*from-l* (*Va*::*State*) *t* + *1*) ⟶
levels (*mem-pool-info x p*) ! *jj* =
levels (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) ! *jj* **and**
a1:*bits* (*levels* (*mem-pool-info x p*) ! *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*)) =
list-updates-n
(*bits* (*levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*)))
(*Suc* (*bn* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* ∗ *4*)) *3 FREE* **and**
a2:*i1*=(*nat* (*from-l Va t*)) **and**
a3:*j1*= (*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*))) **and**
a4:*i2* = *nat* (*from-l Va t* + *1*) **and**
a5:*j2* = (*4*∗*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*))) **and**
a6:*j3* = *Suc*(*4*∗(*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*)))) **and**
a7:*j4* = *Suc*(*Suc*(*4*∗(*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*))))) **and**
a8:*j5* = *Suc*(*Suc*(*Suc*(*4*∗(*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*))))))
shows ∀ *i j*. ¬((*i*=*i1* ∧ *j*=*j1*) ∨(*i*=*i2* ∧ *j*=*j2*) ∨ (*i*=*i2* ∧ *j*=*j3*)∨ (*i*=*i2* ∧ *j*=*j4*)∨ (*i*=*i2* ∧ *j*=*j5*)) ⟶
get-bit-s x p i j = *get-bit-s Va p i j*
using *a0 a1 a2 a4 a4 a5 a6 a7 a8 get-bit-x-stm4-pre-eq same-bit-mp-alloc-stm4-pre-precond-f*
proof−
{**fix** *i j*
assume *a00*:¬((*i*=*i1* ∧ *j*=*j1*) ∨(*i*=*i2* ∧ *j*=*j2*) ∨ (*i*=*i2* ∧ *j*=*j3*)∨ (*i*=*i2* ∧ *j*=*j4*)∨ (*i*=*i2* ∧ *j*=*j5*))
then have *get-bit-s Va p i j* =
get-bit-s (*mp-alloc-stm4-pre-precond-f Va t p*) *p i j*
using *same-bit-mp-alloc-stm4-pre-precond-f1 a2 a3 a4 a5*
by *auto*
also have *get-bit-s x p i j* =
get-bit-s (*mp-alloc-stm4-pre-precond-f Va t p*) *p i j*
using *a00 a0 a1 a2 a4 a4 a5 a6 a7 a8 get-bit-x-stm4-pre-eq*[*OF a0 a1*]
mp-alloc-stm4-pre-precond-f-bn
by (*metis mult.commute*)
finally have *get-bit-s x p i j* = *get-bit-s Va p i j* .
} **thus** *?thesis* **by** *fastforce*

qed

lemma *same-bit-mp-alloc-x-va*:
assumes
a0:$\forall$ *jj*. *jj* $\neq$ *nat* (*from-l Va t* + *1*) $\longrightarrow$
levels (*mem-pool-info x p*) ! *jj* =
levels (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) ! *jj* **and**
a1: *bits* (*levels* (*mem-pool-info x p*) ! *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*)) =
list-updates-n
(*bits* (*levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*)))
(*Suc* (*bn* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* ∗ *4*)) *3 FREE* **and**
a2:¬((*l*=(*nat* (*from-l Va t*)) ∧ *b*=(*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*)))) ∨
(*l*=(*nat* (*from-l Va t* + *1*)) ∧ *b*=(*4*∗*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*)))) ∨
(*l*=(*nat* (*from-l Va t* + *1*)) ∧ *b*=*Suc*((*4*∗*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*))))) ∨
(*l*=(*nat* (*from-l Va t* + *1*)) ∧ *b*=*Suc*(*Suc*((*4*∗*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*))))))∨
(*l*=(*nat* (*from-l Va t* + *1*)) ∧ *b*=*Suc*(*Suc*(*Suc*((*4*∗*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*))))))))
shows (*get-bit-s x p l b* = *get-bit-s Va p l b*)
using *same-bit-mp-alloc-stm4-pre-precond-f-x*[*OF a0 a1*] *a2*
by *auto*

lemma *free-list-x*:
assumes *a0*:*free-list* (*levels* (*mem-pool-info x p*) ! *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*)) =
inserts
(*map* (λ*ii*. *lsizes* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*) ∗
ii +
blk (*mp-alloc-stm4-pre-precond-f Va t p*) *t*)
[*Suc NULL*..<*4*])
(*free-list*
(*levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*)))
shows
free-list (*levels* (*mem-pool-info x p*) ! *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*)) = (*free-list*
(*levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*)))@
[*lsizes* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*) ∗

1 +
blk (*mp-alloc-stm4-pre-precond-f Va t p*) *t*,*lsizes* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*) ∗
2 +
blk (*mp-alloc-stm4-pre-precond-f Va t p*) *t*,
lsizes (*mp-alloc-stm4-pre-precond-f Va t p*) *t* !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*) ∗
3 +
blk (*mp-alloc-stm4-pre-precond-f Va t p*) *t*]
using *a0*
by (*simp add*: *numeral-3-eq-3 numeral-Bit0 inserts-def*)
lemma *listx1*:
jj = *length* (*free-list* (*levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*))) ⟹
free-list (*levels* (*mem-pool-info x p*) ! *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*)) =
(*free-list* (*levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*)))@
[*lsizes* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*) ∗ *1* +
blk (*mp-alloc-stm4-pre-precond-f Va t p*) *t*,
lsizes (*mp-alloc-stm4-pre-precond-f Va t p*) *t* !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*) ∗ *2* +
blk (*mp-alloc-stm4-pre-precond-f Va t p*) *t*,
lsizes (*mp-alloc-stm4-pre-precond-f Va t p*) *t* !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*) ∗ *3* +
blk (*mp-alloc-stm4-pre-precond-f Va t p*) *t*] ⟹
free-list (*levels* (*mem-pool-info x p*) ! *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*))! *jj* =
lsizes (*mp-alloc-stm4-pre-precond-f Va t p*) *t* !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*) ∗ *1* +
blk (*mp-alloc-stm4-pre-precond-f Va t p*) *t*
by *auto*

lemma *listx3*:
jj = *Suc*(*Suc* (*length* (*free-list* (*levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*))))) ⟹
free-list (*levels* (*mem-pool-info x p*) ! *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*)) =
(*free-list* (*levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*)))@
[*lsizes* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*) ∗ *1* +
blk (*mp-alloc-stm4-pre-precond-f Va t p*) *t*,

lsizes (*mp-alloc-stm4-pre-precond-f Va t p*) *t* !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*) ∗ *2* +
blk (*mp-alloc-stm4-pre-precond-f Va t p*) *t*,
lsizes (*mp-alloc-stm4-pre-precond-f Va t p*) *t* !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*) ∗ *3* +
blk (*mp-alloc-stm4-pre-precond-f Va t p*) *t*] ⟹
free-list (*levels* (*mem-pool-info x p*) ! *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*))! *jj* =
lsizes (*mp-alloc-stm4-pre-precond-f Va t p*) *t* !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*) ∗ *3* +
blk (*mp-alloc-stm4-pre-precond-f Va t p*) *t*
by (*simp add*: *nth-append*)

lemma *set-free-x-va*: **assumes** *a0*:*free-list* (*levels* (*mem-pool-info x p*) ! *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*)) =
inserts
(*map* (*λii*. *lsizes* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*) ∗
ii +
blk (*mp-alloc-stm4-pre-precond-f Va t p*) *t*)
[*Suc NULL*..<*4*])
(*free-list*
(*levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*)))
shows *set* (*free-list* (*levels* (*mem-pool-info x p*) ! *nat* (*from-l Va t* + *1*))) =
set (*free-list* (*levels* (*mem-pool-info Va p*) ! *nat* (*from-l Va t* + *1*))) ∪
{*lsizes* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*) ∗
1 +
blk (*mp-alloc-stm4-pre-precond-f Va t p*) *t*,*lsizes* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*) ∗
2 +
blk (*mp-alloc-stm4-pre-precond-f Va t p*) *t*,
lsizes (*mp-alloc-stm4-pre-precond-f Va t p*) *t* !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*) ∗
3 +
blk (*mp-alloc-stm4-pre-precond-f Va t p*) *t*}
proof−
have *free-list* (*levels* (*mem-pool-info x p*) ! *nat* (*from-l Va t* + *1*)) =
(*free-list*
(*levels* (*mem-pool-info Va p*) !
nat (*from-l Va t* + *1*)))@
[*lsizes Va t* !
nat (*from-l Va t* + *1*) ∗
1 +
blk (*mp-alloc-stm4-pre-precond-f Va t p*) *t*,*lsizes* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* !
nat (*from-l Va t* + *1*) ∗

2 +
blk (*mp-alloc-stm4-pre-precond-f Va t p*) *t*,
lsizes Va t !
nat (*from-l Va t* + *1*) ∗
3 +
blk (*mp-alloc-stm4-pre-precond-f Va t p*) *t*]
using *free-list-x*[*OF a0*]
by (*metis eq-free-list-mp-alloc-stm4-pre-precond-f mp-alloc-stm4-pre-froml mp-alloc-stm4-pre-precond-f-lsz*)
then show *?thesis* **using** *mp-alloc-stm4-pre-froml mp-alloc-stm4-pre-precond-f-lsz*
by (*metis empty-set list.simps*(*15*) *set-append*)
qed

lemma *free-list-x-subset-va*:
assumes *a0*:*free-list* (*levels* (*mem-pool-info x p*) ! *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*)) =
inserts
(*map* (*λii. lsizes* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*) ∗
ii +
blk (*mp-alloc-stm4-pre-precond-f Va t p*) *t*)
[*Suc NULL..*<*4*])
(*free-list*
(*levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*)))
shows *set* (*free-list* (*levels* (*mem-pool-info Va p*) ! *nat* (*from-l Va t* + *1*))) ⊆
set (*free-list* (*levels* (*mem-pool-info x p*) ! *nat* (*from-l Va t* + *1*)))
proof−
have *free-list* (*levels* (*mem-pool-info x p*) ! *nat* (*from-l Va t* + *1*)) =
(*free-list*
(*levels* (*mem-pool-info Va p*) !
nat (*from-l Va t* + *1*)))@
[*lsizes Va t* !
nat (*from-l Va t* + *1*) ∗
1 +
blk (*mp-alloc-stm4-pre-precond-f Va t p*) *t*,*lsizes* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* !
nat (*from-l Va t* + *1*) ∗
2 +
blk (*mp-alloc-stm4-pre-precond-f Va t p*) *t*,
lsizes Va t !
nat (*from-l Va t* + *1*) ∗
3 +
blk (*mp-alloc-stm4-pre-precond-f Va t p*) *t*]
using *free-list-x*[*OF a0*]
by (*metis eq-free-list-mp-alloc-stm4-pre-precond-f mp-alloc-stm4-pre-froml mp-alloc-stm4-pre-precond-f-lsz*)
then show *?thesis* **by** *auto*
qed

lemma *free-level-x-va*:

assumes
a0:∀ *jj*. *jj* ≠ *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*) ⟶
levels (*mem-pool-info x p*) ! *jj* =
levels (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) ! *jj*
shows ∀ *jj*. *jj* ≠ *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*) ⟶
free-list (*levels* (*mem-pool-info x p*) !*jj*) = *free-list* (*levels* (*mem-pool-info Va p*)! *jj*)
by (*simp add*: *assms eq-free-list-mp-alloc-stm4-pre-precond-f*)

lemma *mp-alloc-stm4-pre-precond-f-bitmap-not-free*:
assumes *a0*:(*get-bit-s Va p l b* ≠ *FREE*) **and**
a1:*l* < *length* (*levels* (*mem-pool-info Va p*)) **and**
a2:*b*<*length* (*bits* (*levels* (*mem-pool-info Va p*) ! *l*)) **and**
a3:(*from-l Va t*) ≥ *0*
shows (*get-bit-s* (*mp-alloc-stm4-pre-precond-f Va t p*) *p l b* ≠ *FREE*)
using *a0 a1 a2 a3 same-bit-mp-alloc-stm4-pre-precond-divided same-bit-mp-alloc-stm4-pre-precond-allocating same-bit-mp-alloc-stm4-pre-precond-f1 BlockState.distinct*(*11*) *BlockState.distinct*(*17*)

proof−
let *?i1*=(*nat* (*from-l Va t*)) **and**
?j1= (*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*))) **and**
?i2 = (*nat* (*from-l Va t* + *1*)) **and**
?j2 = (*4*∗*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*)))
have *i1orj1*:*?i1* ≠ *?i2* ∨ *?j1* ≠ *?j2* **using** *a3* **by** *auto*
{assume *a00*:¬((*l*=*?i1* ∧ *b*=*?j1*) ∨(*l*=*?i2* ∧ *b*=*?j2*))
then have *?thesis* **using** *same-bit-mp-alloc-stm4-pre-precond-f1 a0*
by *auto*
}
moreover {assume *a00*:(*l*=*?i1* ∧ *b*=*?j1*)
have *?thesis*
using *same-bit-mp-alloc-stm4-pre-precond-divided*[*OF a00 a1 a2 a3*]
by *auto*
}
moreover {assume *a00*:(*l*=*?i2* ∧ *b*=*?j2*)
have *?thesis*
using *same-bit-mp-alloc-stm4-pre-precond-allocating*[*OF a00 a1 a2 a3*]
by *auto*
}
ultimately show *?thesis* **by** *auto*
qed

lemma *mp-alloc-stm4-pre-inv-nmax*: *n-max* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *pa*) ∗ *4* ˆ *ii* =
n-max (*mem-pool-info Va pa*) ∗ *4* ˆ *ii*
unfolding *mp-alloc-stm4-pre-precond-f-def set-bit-def*

by *auto*

lemma *allocating-next-notexists*:*inv-bitmap Va* ⟹
p∈*mem-pools Va* ⟹
ii < *length* (*levels* (*mem-pool-info Va p*)) ⟹
jj < *length* (*bits* (*levels* (*mem-pool-info Va p*) ! *ii*)) ⟹
get-bit-s Va p ii jj = *ALLOCATING* ⟹
ii < *length* (*levels* (*mem-pool-info Va p*)) − *1* ⟶ *noexist-bits* (*mem-pool-info Va p*) (*ii* + *1*) (*jj* ∗ *4*)
unfolding *inv-bitmap-def inv-mempool-info-def Let-def*
by *auto*

lemma *block-n*:
assumes
a0:*lsizes Va t* ! *nat* (*from-l Va t*) = *ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *nat* (*from-l Va t*) **and**
a1:*p* ∈ *mem-pools Va* **and**
a2:*inv-mempool-info Va* **and**
a3:*blk Va t* = *buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ˆ *nat* (*from-l Va t*)) **and**
a4:*alloc-l Va t* < *int* (*n-levels* (*mem-pool-info Va p*)) **and**
a5:*from-l Va t* < *alloc-l Va t* **and**
a6:*OK* ≤ *from-l Va t*
shows (*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*))) = *n*
proof−
have (∃ *n*>*NULL*. *max-sz* (*mem-pool-info Va p*) = *4* ∗ *n* ∗ *4* ˆ *n-levels* (*mem-pool-info Va p*)) ∧
NULL < *n-max* (*mem-pool-info Va p*) ∧
NULL < *n-levels* (*mem-pool-info Va p*) ∧ *n-levels* (*mem-pool-info Va p*) = *length* (*levels* (*mem-pool-info Va p*))
using *a2 a1* **unfolding** *inv-mempool-info-def Let-def* **by** *auto*
then show *?thesis* **using** *assms mp-alloc-stm3-lm2-inv-1-2 inv-mempool-info-maxsz-align4*[*OF a2*] *nat-less-iff*
unfolding *block-num-def Let-def* **apply** *auto*
by (*smt less-numeral-extra*(*3*))
qed

definition *addr*::*nat* ⇒ *nat* ⇒ *nat* ⇒ *nat* ⇒ *nat*
where *addr m-size init l n* ≡ *init* + *n* ∗(*m-size div 4* ˆ *l*)

definition *next-addr* :: *nat* ⇒ *nat* ⇒ *nat* ⇒ *nat* ⇒ *nat*
where *next-addr m-size c-addr l n* ≡ (*m-size div 4* ˆ (*l* + *1*))∗*n* + *c-addr*

lemma *next-level-addr*:

assumes
a0:∃ *m*. *m-size* = *m*∗*4*ˆ *p* **and**
a1:*p* > *l*+*1*
shows *next-addr m-size* (*addr m-size init l n*) *l ch* = *addr m-size init* (*l*+*1*) (*n*∗*4* + *ch*)
unfolding *next-addr-def addr-def*
proof(*induct ch*)
case *0*
then show *?case*
apply *auto*
by (*smt One-nat-def a0 a1 dvd-mult-div-cancel dvd-triv-right less-imp-le-nat mult.commute mult.left-commute nonzero-mult-div-cancel-left power-Suc0-right power-add power-le-dvd power-not-zero zero-neq-numeral*)
next
case (*Suc ch*)
obtain *m* **where** *m-size* = *m*∗(*4*::*nat*) ˆ *p* **using** *a0* **by** *auto*
then show *?case* **using** *Suc a1* **by** *auto*
qed

lemma *next-level-addr-eq1*:
assumes
a0:∃ *m*. *m-size* = *m*∗*4*ˆ *p* **and**
a1:*p* > *l*+*1*
shows *next-addr m-size* (*addr m-size init l n*) *l 0* = *addr m-size init l n*
using *next-level-addr*[*OF a0 a1*] **unfolding** *next-addr-def addr-def*
by *linarith*

lemma *next-level-addr-eq*:
assumes
a0:∃ *m*. *m-size* = *m*∗*4*ˆ *p* **and**
a1:*p* > *l*+*1*
shows *addr m-size init* (*l* + *1*) (*n* ∗ *4*) = *addr m-size init l n*
using *next-level-addr*[*OF a0 a1*] *next-level-addr-eq1*[*OF a0 a1*]
by *auto*

lemma *diff-n-m-addr*: **assumes** *a0*:*n*≠*m* **and** *a1*:*m-size* ≥ *4*ˆ*l*
shows *addr m-size init l n* ≠ *addr m-size init l m*
using *a0 a1* **unfolding** *addr-def*
by (*auto simp add*: *Euclidean-Division.div-eq-0-iff*)

lemma *lsizes-addr*:
assumes *a0*:*p* ∈ *mem-pools Va* **and**
a1:∀ *ii*<*length* (*lsizes Va t*). *lsizes Va t* ! *ii* = *ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *ii* **and**
a2:*length* (*lsizes Va t*) ≤ *n-levels* (*mem-pool-info Va p*) **and**
a3:*inv-aux-vars Va* ∧ *inv-bitmap Va* ∧ *inv-mempool-info Va* ∧ *inv-bitmap-freelist Va* **and**
a4:*l*+*1* <*length* (*lsizes Va t*)
shows ∀ *j*. (*lsizes Va t* ! (*l*+*1*)) ∗ *j* +

(*buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ˆ *l*)) =
addr (*max-sz* (*mem-pool-info Va p*)) (*buf* (*mem-pool-info Va p*)) (*l* + *1*)
((*block-num* (*mem-pool-info Va p*) (*buf* (*mem-pool-info Va p*) +
n ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ˆ *l*))
(*lsizes Va t* ! *l*))∗*4* + *j*)
proof−
{**fix** *j*
let *?blk* = (*buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ˆ *l*))
obtain *m* **where** *max-sz*:*max-sz* (*mem-pool-info Va p*) = *4* ∗ *m* ∗ *4* ˆ *n-levels* (*mem-pool-info Va p*)
using *a3 a0* **unfolding** *inv-mempool-info-def Let-def* **by** *auto*
have *b1*:*?blk* =
addr (*max-sz* (*mem-pool-info Va p*)) (*buf* (*mem-pool-info Va p*)) *l*
(*block-num* (*mem-pool-info Va p*) *?blk* (*lsizes Va t* ! *l*))
using *a4 a0 a1 a3*
unfolding *addr-def block-num-def* **apply** *auto*
by (*metis add.commute add-lessD1 div-mult-self-is-m*
inv-mempool-info-maxsz-align4 plus-1-eq-Suc)
have *b2*:(*lsizes Va t* ! (*l*+*1*)) ∗ *j* + *?blk* =
addr (*max-sz* (*mem-pool-info Va p*)) (*buf* (*mem-pool-info Va p*)) (*l*+*1*)
((*block-num* (*mem-pool-info Va p*) *?blk* (*lsizes Va t* ! *l*))∗*4* + *j*)
using *assms a4 inv-mempool-info-maxsz-align4 max-sz b1 next-level-addr*
unfolding *next-addr-def*
by (*smt le-eq-less-or-eq le-less-trans*)
}**thus** *?thesis* **by** *auto*
qed

lemma *free-list-updates-inv1*:
assumes *a0*:*p* ∈ *mem-pools Va* **and**
a1:¬ *free-l Va t* < *OK* **and**
a2:*free-l Va t* ≤ *from-l Va t* **and**
a3:*alloc-l Va t* < *int* (*n-levels* (*mem-pool-info Va p*)) **and**
a4:*from-l Va t* < *alloc-l Va t* **and**
a5:*alloc-l Va t* = *int* (*length* (*lsizes Va t*)) − *1* ∧ *length* (*lsizes Va t*) = *n-levels* (*mem-pool-info Va p*) ∨
alloc-l Va t = *int* (*length* (*lsizes Va t*)) − *2* ∧ *lsizes Va t* ! *nat* (*alloc-l Va t* + *1*) < *sz* **and**
a6:*block-num* (*mem-pool-info Va p*)
(*buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ˆ *nat* (*from-l Va t*)))
(*lsizes Va t* ! *nat* (*from-l Va t*))
< *n-max* (*mem-pool-info Va p*) ∗ *4* ˆ *nat* (*from-l Va t*) **and**
a7:*blk Va t* = *buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ˆ *nat* (*from-l Va t*)) **and**

$a8$:(x, $mp\text{-}alloc\text{-}stm4\text{-}pre\text{-}precond\text{-}f\ Va\ t\ p$) $\in$ $gvars\text{-}conf\text{-}stable$ **and**
$a9$:$from\text{-}l\ x = from\text{-}l\ (mp\text{-}alloc\text{-}stm4\text{-}pre\text{-}precond\text{-}f\ Va\ t\ p)$ **and**
$a10$:$freeing\text{-}node\ x = freeing\text{-}node\ (mp\text{-}alloc\text{-}stm4\text{-}pre\text{-}precond\text{-}f\ Va\ t\ p)$ **and**
$a11$:$allocating\text{-}node\ x = allocating\text{-}node\ (mp\text{-}alloc\text{-}stm4\text{-}pre\text{-}precond\text{-}f\ Va\ t\ p)$ **and**
$a12$:$\forall pa.\ pa \neq p \longrightarrow mem\text{-}pool\text{-}info\ x\ pa = mem\text{-}pool\text{-}info\ (mp\text{-}alloc\text{-}stm4\text{-}pre\text{-}precond\text{-}f$
$Va\ t\ p)\ pa$ **and**
$a13$:$\forall jj.\ jj \neq nat\ (from\text{-}l\ (mp\text{-}alloc\text{-}stm4\text{-}pre\text{-}precond\text{-}f\ Va\ t\ p)\ t + 1) \longrightarrow$
$levels\ (mem\text{-}pool\text{-}info\ x\ p)\ !\ jj = levels\ (mem\text{-}pool\text{-}info\ (mp\text{-}alloc\text{-}stm4\text{-}pre\text{-}precond\text{-}f$
$Va\ t\ p)\ p)\ !\ jj$ **and**
$a14$:$\forall ii<length\ (lsizes\ Va\ t).\ lsizes\ Va\ t\ !\ ii = ALIGN4\ (max\text{-}sz\ (mem\text{-}pool\text{-}info$
$Va\ p))\ div\ 4\ \hat{}\ ii$ **and**
$a15$:$i\ x\ t = 4$ **and**
$a16$:$lsizes\ x = lsizes\ (mp\text{-}alloc\text{-}stm4\text{-}pre\text{-}precond\text{-}f\ Va\ t\ p)$ **and**
$a17$:$length\ (lsizes\ Va\ t) \leq n\text{-}levels\ (mem\text{-}pool\text{-}info\ Va\ p)$ **and**
$a18$:$bits\ (levels\ (mem\text{-}pool\text{-}info\ x\ p)\ !\ nat\ (from\text{-}l\ (mp\text{-}alloc\text{-}stm4\text{-}pre\text{-}precond\text{-}f\ Va$
$t\ p)\ t + 1)) =$
$list\text{-}updates\text{-}n$
$(bits\ (levels\ (mem\text{-}pool\text{-}info\ (mp\text{-}alloc\text{-}stm4\text{-}pre\text{-}precond\text{-}f\ Va\ t\ p)\ p)\ !$
$nat\ (from\text{-}l\ (mp\text{-}alloc\text{-}stm4\text{-}pre\text{-}precond\text{-}f\ Va\ t\ p)\ t + 1)))$
$(Suc\ (bn\ (mp\text{-}alloc\text{-}stm4\text{-}pre\text{-}precond\text{-}f\ Va\ t\ p)\ t * 4))\ 3\ FREE$ **and**
$a19$:
$free\text{-}list\ (levels\ (mem\text{-}pool\text{-}info\ x\ p)\ !\ nat\ (from\text{-}l\ (mp\text{-}alloc\text{-}stm4\text{-}pre\text{-}precond\text{-}f\ Va$
$t\ p)\ t + 1)) =$
$inserts$
$(map\ (\lambda ii.\ lsizes\ (mp\text{-}alloc\text{-}stm4\text{-}pre\text{-}precond\text{-}f\ Va\ t\ p)\ t\ !$
$nat\ (from\text{-}l\ (mp\text{-}alloc\text{-}stm4\text{-}pre\text{-}precond\text{-}f\ Va\ t\ p)\ t + 1) *$
$ii +$
$blk\ (mp\text{-}alloc\text{-}stm4\text{-}pre\text{-}precond\text{-}f\ Va\ t\ p)\ t)$
$[Suc\ NULL..<4])$
$(free\text{-}list$
$(levels\ (mem\text{-}pool\text{-}info\ (mp\text{-}alloc\text{-}stm4\text{-}pre\text{-}precond\text{-}f\ Va\ t\ p)\ p)\ !$
$nat\ (from\text{-}l\ (mp\text{-}alloc\text{-}stm4\text{-}pre\text{-}precond\text{-}f\ Va\ t\ p)\ t + 1)))$ **and**
$a20$:$allocating\text{-}node\ Va\ t =$
$Some\ (\!|pool = p,\ level = nat\ (from\text{-}l\ Va\ t),$
$block = block\text{-}num\ (mem\text{-}pool\text{-}info\ Va\ p)$
$(buf\ (mem\text{-}pool\text{-}info\ Va\ p) + n * (max\text{-}sz\ (mem\text{-}pool\text{-}info\ Va\ p)\ div$
$4\ \hat{}\ nat\ (from\text{-}l\ Va\ t)))$
$(lsizes\ Va\ t\ !\ nat\ (from\text{-}l\ Va\ t)),$
$data = buf\ (mem\text{-}pool\text{-}info\ Va\ p) + n * (max\text{-}sz\ (mem\text{-}pool\text{-}info\ Va\ p)\ div$
$4\ \hat{}\ nat\ (from\text{-}l\ Va\ t))|\!)$ **and**
$a21$:$inv\text{-}aux\text{-}vars\ Va \wedge inv\text{-}bitmap\ Va \wedge inv\text{-}mempool\text{-}info\ Va \wedge inv\text{-}bitmap\text{-}freelist$
Va **and**
$a22$:$ii < length\ (levels\ (mem\text{-}pool\text{-}info\ x\ p))$ **and**
$a23$:$blk\ x = blk\ (mp\text{-}alloc\text{-}stm4\text{-}pre\text{-}precond\text{-}f\ Va\ t\ p)$ **and**
$a24$:$lsizes\ x = lsizes\ (mp\text{-}alloc\text{-}stm4\text{-}pre\text{-}precond\text{-}f\ Va\ t\ p)$
shows $\forall j<length\ (bits\ (levels\ (mem\text{-}pool\text{-}info\ x\ p)\ !\ ii)).$
$(get\text{-}bit\text{-}s\ x\ p\ ii\ j = FREE) =$
$(buf\ (mem\text{-}pool\text{-}info\ x\ p) + j * (max\text{-}sz\ (mem\text{-}pool\text{-}info\ x\ p)\ div\ 4\ \hat{}\ ii)$
$\in set\ (free\text{-}list\ (levels\ (mem\text{-}pool\text{-}info\ x\ p)\ !\ ii)))$

proof−
{
let *?i1*=(*nat* (*from-l Va t*)) **and**
?j1= (*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*))) **and**
?i2 = (*nat* (*from-l Va t* + *1*)) **and**
?j2 = (*4*∗*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*)))
fix *j*
let *?mp* = *mem-pool-info x p*
let *?bts* = *bits* (*levels ?mp* ! *ii*) **and** *?fl* = *free-list* (*levels ?mp* ! *ii*)
assume *a00*:*j*<*length ?bts*
then have *a00′*:*j* < *length* (*bits* (*levels* (*mem-pool-info Va p*) ! *ii*))
using *mp-alloc-stm4-inv-bits-len*
by (*metis a13 a18 length-list-update-n*)
have *inv-bitmap1*:(∀ *j*<*length* (*bits* (*levels* (*mem-pool-info Va p*) ! *ii*)).
(*get-bit-s Va p ii j* = *FREE*) =
(*buf* (*mem-pool-info Va p*) + *j* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ˆ *ii*)
∈ *set* (*free-list* (*levels* (*mem-pool-info Va p*) ! *ii*))))
using *a21 a0 a22 mp-alloc-stm4-lvl-len*[*OF a0 a8*]
unfolding *Let-def inv-bitmap-freelist-def*
by *fastforce*+
have *from-l-gt0*:*0* ≤ *from-l Va t* **using** *a1 a2* **by** *linarith*
have *len-levels*:*length* (*levels* (*mem-pool-info x p*)) = *length* (*levels* (*mem-pool-info Va p*))
using *mp-alloc-stm4-lvl-len*[*OF a0 a8*] **by** *simp*
have *maxsz*:*max-sz* (*mem-pool-info x p*) = *max-sz* (*mem-pool-info Va p*)
using *mp-alloc-stm4-maxsz*[*OF a0 a8*] **by** *simp*
have *buf*:*buf* (*mem-pool-info x p*) = *buf* (*mem-pool-info Va p*)
using *mp-alloc-stm4-buf*[*OF a0 a8*] **by** *simp*
have *from-l*:*from-l x* = *from-l Va*
using *mp-alloc-stm4-froml*[*OF a9*] **by** *auto*
have *mem-pools*:*mem-pools x* = *mem-pools Va* **using** *mp-alloc-stm4-mempools*[*OF a8*] **by** *auto*
have *lsizes-x-va*:*lsizes x* = *lsizes Va*
by (*simp add*: *a16 mp-alloc-stm4-pre-precond-f-lsz*)
have *len-eq*:*length* (*bits* (*levels* (*mem-pool-info x p*) ! *ii*)) =
length (*bits* (*levels* (*mem-pool-info Va p*) ! *ii*))
using *a22 a8 mp-alloc-stm4-inv-bits-len*
unfolding *gvars-conf-stable-def gvars-conf-def*
by *fastforce*
then have *get-bits-va*:(*get-bit-s Va p ii j* = *FREE*) =
(*buf* (*mem-pool-info Va p*) + *j* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ˆ *ii*)
∈ *set* (*free-list* (*levels* (*mem-pool-info Va p*) ! *ii*)))
using *inv-bitmap1 a00* **by** *auto*
have *inv-mempool-info-mp Va p*
using *a21 mem-pools a0* **unfolding** *inv-mempool-info-def* **by** *auto*

note *inv-mempool=this*[*simplified Let-def*]
have *a19′*:*ii* < *length* (*levels* (*mem-pool-info Va p*))
using *a22 mp-alloc-stm4-inv-mif-len*
by (*simp add*: *len-levels*)
{ **assume** *a03*:*ii*≠*?i1* ∧ *ii*≠*?i2*
then have *eq-get-bit-i-j*:*get-bit-s x p ii j* = *get-bit-s Va p ii j*
using *same-bit-mp-alloc-x-va*[*OF a13*[*simplified a9*[*simplified mp-alloc-stm4-froml*[*OF a9*], *THEN sym*]] *a18*] **by** *fast*
moreover have *free-list* (*levels* (*mem-pool-info x p*) ! *ii*) =
free-list (*levels* (*mem-pool-info Va p*) ! *ii*)
using *free-level-x-va*[*OF a13*] *a03 a9 from-l* **by** *metis*
ultimately have (*?bts* ! *j* = *FREE*) = (*buf ?mp* + *j* ∗ (*max-sz ?mp div 4* ˆ *ii*) ∈ *set ?fl*)
using *get-bits-va eq-get-bit-i-j*
by (*simp add*: *buf maxsz*)
}
moreover { **assume** *a03*:*ii*=*?i1*
then have *free*:*free-list* (*levels* (*mem-pool-info x p*) ! *ii*) =
free-list (*levels* (*mem-pool-info Va p*) ! *ii*)
using *free-level-x-va*[*OF a13*] *a03 a9 from-l from-l-gt0* **by** *auto*
{ **assume** *a04*:*j*≠*?j1*
then have *eq-get-bit-i-j*:*get-bit-s x p ii j* = *get-bit-s Va p ii j*
using *same-bit-mp-alloc-x-va*[*OF a13*[*simplified a9*[*simplified from-l*, *THEN sym*]] *a18*]
a03 from-l-gt0
by (*simp add*: *eq-nat-nat-iff*)
then have (*?bts* ! *j* = *FREE*) = (*buf ?mp* + *j* ∗ (*max-sz ?mp div 4* ˆ *ii*) ∈ *set ?fl*)
using *free* **by** (*simp add*: *buf get-bits-va maxsz*)
}
moreover { **assume** *a04*:*j*=*?j1*
then have (*?bts* ! *j* = *DIVIDED*)
using *get-bit-x-l-b a03 a13 a00 a22 len-levels*
len-eq a13 from-l from-l-gt0
by (*simp add*: *a9*)
moreover have *buf* (*mem-pool-info Va p*) + *j* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ˆ *ii*) ∉
set (*free-list* (*levels* (*mem-pool-info Va p*) ! *ii*))
using *get-bits-va a03 a20 a21 a04 a7* **unfolding** *inv-aux-vars-def*
by (*metis BlockState.distinct*(*17*) *Mem-block.select-convs*(*1*) *Mem-block.select-convs*(*2*) *Mem-block.select-convs*(*3*))
ultimately have (*?bts* ! *j* = *FREE*) = (*buf ?mp* + *j* ∗ (*max-sz ?mp div 4* ˆ *ii*) ∈ *set ?fl*)
by (*simp add*: *buf free maxsz*)
}
ultimately have (*?bts* ! *j* = *FREE*) = (*buf ?mp* + *j* ∗ (*max-sz ?mp div 4* ˆ *ii*) ∈ *set ?fl*)
by *auto*
}

moreover { **assume** *a03*:*ii*=*?i2*
then have *block-n*:(*block-num* (*mem-pool-info Va p*)
(*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*))) = *n*
proof−
have *lsizes Va t* ! *nat* (*from-l Va t*) =
ALIGN4 (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ
(*nat* (*from-l Va t*))
using *a14 lsizes-x-va a16 a1 a2 a4 a5 a9 from-l* **by** *auto*
thus *?thesis* **using** *block-n a21 a0 a0 a7 a3 a4 from-l-gt0*
by *blast*
qed
obtain *m* **where** *max-sz*:*max-sz* (*mem-pool-info Va p*) = *4* ∗ *m* ∗ *4* ˆ *n-levels* (*mem-pool-info Va p*)
using *a21 a0* **unfolding** *inv-mempool-info-def Let-def* **by** *auto*
have *ls*:*4* ˆ *ii dvd 4* ∗ *m* ∗ *4* ˆ *n-levels* (*mem-pool-info Va p*) **using** *a03 a22*
by (*metis dvd-triv-right inv-mempool len-levels less-imp-le-nat power-le-dvd*)
have *b0*:*buf* (*mem-pool-info Va p*) + *j* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ˆ *ii*) =
addr (*max-sz* (*mem-pool-info Va p*)) (*buf* (*mem-pool-info Va p*)) *ii j*
unfolding *addr-def* **by** *auto*
have *suc-from-l-lt-lsize*:(*nat* (*from-l Va t*)) + *1*< *length* (*lsizes Va t*)
using *a4 a5 from-l-gt0* **by** *linarith*
have *b2*:∀ *j*. (*lsizes Va t* ! *nat* (*from-l Va t* + *1*)) ∗ *j* + *blk Va t* =
addr (*max-sz* (*mem-pool-info Va p*)) (*buf* (*mem-pool-info Va p*)) (*nat* (*from-l Va t* + *1*))
((*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*)))∗*4* + *j*)
using *lsizes-addr*[*OF a0 a14 a17 a21 suc-from-l-lt-lsize*] *a7 from-l-gt0 block-n*
by (*simp add*: *Suc-nat-eq-nat-zadd1 add.commute*)
then have *b2*:∀ *j*. *lsizes* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*) ∗ *j* +
blk (*mp-alloc-stm4-pre-precond-f Va t p*) *t* =
addr (*max-sz* (*mem-pool-info Va p*)) (*buf* (*mem-pool-info Va p*)) (*nat* (*from-l Va t* + *1*))
((*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*)))∗*4* + *j*)
by (*metis mp-alloc-stm4-blk mp-alloc-stm4-pre-precond-f-froml mp-alloc-stm4-pre-precond-f-lsz*)
{ **assume** *a04*:*j*≠*?j2* ∧ *j*≠*Suc ?j2* ∧ *j*≠*Suc* (*Suc ?j2*) ∧ *j*≠*Suc* (*Suc* (*Suc ?j2*))
then have *eq-get-bit-i-j*:*get-bit-s x p ii j* = *get-bit-s Va p ii j*
using *same-bit-mp-alloc-x-va*[*OF a13*[*simplified a9*[*simplified from-l*, *THEN sym*]] *a18*]
a03 from-l-gt0
by (*simp add*: *eq-nat-nat-iff*)
{ **assume** *get-bit-s Va p ii j* = *FREE*
then have (*buf* (*mem-pool-info Va p*) + *j* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ˆ *ii*)
∈ *set* (*free-list* (*levels* (*mem-pool-info Va p*) ! *ii*)))
using *get-bits-va* **by** *blast*

then have $(buf\ ?mp + j * (max\text{-}sz\ ?mp\ div\ 4\ \hat{}\ ii) \in set\ ?fl)$
using *free-list-x-subset-va[OF a19] a03 buf maxsz* **by** *fastforce*
}
moreover {
assume $get\text{-}bit\text{-}s\ Va\ p\ ii\ j \neq FREE$
then have *not-in-free-Va*: $(buf\ (mem\text{-}pool\text{-}info\ Va\ p) + j * (max\text{-}sz\ (mem\text{-}pool\text{-}info\ Va\ p)\ div\ 4\ \hat{}\ ii)$
$\notin set\ (free\text{-}list\ (levels\ (mem\text{-}pool\text{-}info\ Va\ p)\ !\ ii)))$
using *get-bits-va* **by** *blast*
then have $(buf\ ?mp + j * (max\text{-}sz\ ?mp\ div\ 4\ \hat{}\ ii) \notin set\ ?fl)$
proof−
have $\forall k.\ k<4 \longrightarrow (buf\ ?mp + j * (max\text{-}sz\ ?mp\ div\ 4\ \hat{}\ ii)) \neq$
$lsizes\ (mp\text{-}alloc\text{-}stm4\text{-}pre\text{-}precond\text{-}f\ Va\ t\ p)\ t\ !$
$nat\ (from\text{-}l\ (mp\text{-}alloc\text{-}stm4\text{-}pre\text{-}precond\text{-}f\ Va\ t\ p)\ t + 1) * k +$
$blk\ (mp\text{-}alloc\text{-}stm4\text{-}pre\text{-}precond\text{-}f\ Va\ t\ p)\ t$
using *diff-n-m-addr b2 b0 a03 a04 Euclidean-Division.div-eq-0-iff buf inv-mempool*
ls max-sz maxsz
by *(smt Groups.mult-ac(2) add.right-neutral add-2-eq-Suc′ add-Suc-right*

dvd-div-mult-self less-2-cases less-Suc-eq div-greater-zero-iff
mult-is-0 neq0-conv numeral-Bit0 power-not-zero)
then show *?thesis* **using** *buf maxsz not-in-free-Va set-free-x-va[OF a19, simplified] a03*
apply *auto*
by *presburger*
qed
then have $(buf\ ?mp + j * (max\text{-}sz\ ?mp\ div\ 4\ \hat{}\ ii) \notin set\ ?fl)$
using *a03 buf maxsz a04 set-free-x-va[OF a19]* **by** *auto*
} ultimately have $(?bts\ !\ j = FREE) = (buf\ ?mp + j * (max\text{-}sz\ ?mp\ div\ 4\ \hat{}\ ii) \in set\ ?fl)$
using *eq-get-bit-i-j* **by** *auto*
}
moreover { assume *a04*:$j=?j2$
then have *a03′*:$ii = nat\ (from\text{-}l\ x\ t + 1) \wedge$
$j = 4*block\text{-}num\ (mem\text{-}pool\text{-}info\ x\ p)\ (blk\ x\ t)\ (lsizes\ x\ t\ !\ nat\ (from\text{-}l\ x\ t))$
using *a22 a23 a24 from-l buf from-l a03*
unfolding *block-num-def*
by *(simp add: mp-alloc-stm4-pre-precond-f-blk mp-alloc-stm4-pre-precond-f-lsz)*

have $(?bts\ !\ j = ALLOCATING)$
using *from-l-gt0 a22 a00 a03 len-levels a04 a18 from-l get-bit-x-l1-b4 len-eq*
by *(metis a04 a18 get-bit-x-l1-b4 len-eq)*
then have *bts-j-not-free*:$(?bts\ !\ j \neq FREE)$
by *auto*
moreover have *not-in-free-Va*:$buf\ (mem\text{-}pool\text{-}info\ Va\ p) + j * (max\text{-}sz\ (mem\text{-}pool\text{-}info\ Va\ p)\ div\ 4\ \hat{}\ ii) \notin$

set (*free-list* (*levels* (*mem-pool-info Va p*) ! *ii*))
proof−
have *alloc-i1-j1*:*get-bit-s Va p ?i1 ?j1* = *ALLOCATING*
using *a20 a21 a7* **unfolding** *inv-aux-vars-def*
by (*metis* (*no-types*) *Mem-block.select-convs*(*1*)
Mem-block.select-convs(*2*) *Mem-block.select-convs*(*3*))
have *noexist-bits* (*mem-pool-info Va p*) (*?i1* + *1*) (*?j1* ∗ *4*)
proof−
have *?i1* < *length* (*levels* (*mem-pool-info Va p*))−*1*
using *a19′ from-l-gt0 a3 a4 inv-mempool* **by** *auto*
moreover have *?j1* < *length* (*bits* (*levels* (*mem-pool-info Va p*) ! *?i1*))
using *calculation*
by (*simp add*: *a6 a7 inv-mempool*)
ultimately show *?thesis*
using *alloc-i1-j1 a21 a19′ a00′ a0 a03 a04*
unfolding *Let-def inv-bitmap-def*
by (*smt One-nat-def Suc-pred inv-mempool less-Suc-eq*)
qed
then have (*get-bit-s Va p ii j* = *NOEXIST*)
using *a03 a04 from-l-gt0*
by (*simp add*: *mult.commute nat-add-distrib*)
then show *?thesis* **using** *get-bits-va*
by *simp*
qed
have (*buf ?mp* + *j* ∗ (*max-sz ?mp div 4* ˆ *ii*) ∉ *set ?fl*)
proof−
have ∀ *k*. *k*>*0* ∧ *k*<*4* ⟶ (*buf ?mp* + *j* ∗ (*max-sz ?mp div 4* ˆ *ii*)) ≠
lsizes (*mp-alloc-stm4-pre-precond-f Va t p*) *t* !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*) ∗ *k* +
blk (*mp-alloc-stm4-pre-precond-f Va t p*) *t*
using *diff-n-m-addr b2 b0 a03 a04 buf inv-mempool*
maxsz
by (*metis a19′ add-cancel-right-right div-greater-zero-iff*
mp-alloc-stm3-lm2-inv-1-2 mult.commute neq0-conv)
then show *?thesis* **using** *buf maxsz not-in-free-Va set-free-x-va*[*OF a19*, *simplified*] *a03*
by *auto*
qed
then have (*?bts* ! *j* = *FREE*) = (*buf ?mp* + *j* ∗ (*max-sz ?mp div 4* ˆ *ii*) ∈ *set ?fl*)
using *bts-j-not-free* **by** *auto*
}
moreover {
assume *a04*:*j*=*Suc ?j2* ∨ *j* = *Suc* (*Suc ?j2*) ∨ *j* = *Suc* (*Suc* (*Suc ?j2*))
then have *a04′*:*j*=(*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*)) ∗ *4* + *1*) ∨
j = (*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*)) ∗ *4* + *2*) ∨
j = (*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat*

(*from-l Va t*)) ∗ *4* + *3*)
    by *auto*
   have (*?bts* ! *j* = *FREE*)
    using *get-bit-x-l1-b41*[*OF conjI*[*OF a03 a04*] *from-l-gt0* - *a18 a19′ a00′*]
    using *a13 a9 from-l* **by** *auto*
   moreover have *buf ?mp* + *j* ∗ (*max-sz ?mp div 4* ˆ *ii*) ∈ *set ?fl*
    using *a03 a04′*[*simplified*] *set-free-x-va*[*OF a19*, *simplified b2 buf*[*THEN sym*] *maxsz*[*THEN sym*]]
    using *b0*[*simplified buf*[*THEN sym*] *maxsz*[*THEN sym*] *a03*] **by** *auto*
   ultimately have (*?bts* ! *j* = *FREE*) = (*buf ?mp* + *j* ∗ (*max-sz ?mp div 4* ˆ *ii*) ∈ *set ?fl*) **by** *auto*
  } **ultimately have** (*?bts* ! *j* = *FREE*) = (*buf ?mp* + *j* ∗ (*max-sz ?mp div 4* ˆ *ii*) ∈ *set ?fl*)
   by *auto*
 } **ultimately have** (*?bts* ! *j* = *FREE*) = (*buf ?mp* + *j* ∗ (*max-sz ?mp div 4* ˆ *ii*) ∈ *set ?fl*) **by** *auto*
} **then show** *?thesis* **by** *auto*
qed

lemma *free-list-updates-inv2*:
 assumes *a0*:*p* ∈ *mem-pools Va* **and**
 a1:¬ *free-l Va t* < *OK* **and**
 a2:*free-l Va t* ≤ *from-l Va t* **and**
 a3:*alloc-l Va t* < *int* (*n-levels* (*mem-pool-info Va p*)) **and**
 a4:*from-l Va t* < *alloc-l Va t* **and**
a5: *alloc-l Va t* = *int* (*length* (*lsizes Va t*)) − *1* ∧ *length* (*lsizes Va t*) = *n-levels* (*mem-pool-info Va p*) ∨
 alloc-l Va t = *int* (*length* (*lsizes Va t*)) − *2* ∧ *lsizes Va t* ! *nat* (*alloc-l Va t* + *1*) < *sz* **and**
 a6:*block-num* (*mem-pool-info Va p*)
  (*buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ˆ *nat* (*from-l Va t*)))
  (*lsizes Va t* ! *nat* (*from-l Va t*))
 < *n-max* (*mem-pool-info Va p*) ∗ *4* ˆ *nat* (*from-l Va t*) **and**
 a7:*blk Va t* = *buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ˆ *nat* (*from-l Va t*)) **and**
 a8:(*x*, *mp-alloc-stm4-pre-precond-f Va t p*) ∈ *gvars-conf-stable* **and**
 a9:*from-l x* = *from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
 a10:∀ *jj*. *jj* ≠ *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*) ⟶
  levels (*mem-pool-info x p*) ! *jj* = *levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) ! *jj* **and**
a11:∀ *ii*<*length* (*lsizes Va t*). *lsizes Va t* ! *ii* = *ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *ii* **and**
a12:*lsizes x* = *lsizes* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
a13:*length* (*lsizes Va t*) ≤ *n-levels* (*mem-pool-info Va p*) **and**
a14:
free-list (*levels* (*mem-pool-info x p*) ! *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*)) =
 inserts

$(map\ (\lambda ii.\ lsizes\ (mp\text{-}alloc\text{-}stm4\text{-}pre\text{-}precond\text{-}f\ Va\ t\ p)\ t\ !$
$nat\ (from\text{-}l\ (mp\text{-}alloc\text{-}stm4\text{-}pre\text{-}precond\text{-}f\ Va\ t\ p)\ t + 1) *$
$ii +$
$blk\ (mp\text{-}alloc\text{-}stm4\text{-}pre\text{-}precond\text{-}f\ Va\ t\ p)\ t)$
$[Suc\ NULL..<4])$
$(free\text{-}list$
$(levels\ (mem\text{-}pool\text{-}info\ (mp\text{-}alloc\text{-}stm4\text{-}pre\text{-}precond\text{-}f\ Va\ t\ p)\ p)\ !$
$nat\ (from\text{-}l\ (mp\text{-}alloc\text{-}stm4\text{-}pre\text{-}precond\text{-}f\ Va\ t\ p)\ t + 1)))$ **and**
$a15{:}inv\text{-}aux\text{-}vars\ Va \wedge inv\text{-}bitmap\ Va \wedge inv\text{-}mempool\text{-}info\ Va \wedge inv\text{-}bitmap\text{-}freelist\ Va$ **and**
$a16{:}ii < length\ (levels\ (mem\text{-}pool\text{-}info\ x\ p))$ **and**
$a17{:}blk\ x = blk\ (mp\text{-}alloc\text{-}stm4\text{-}pre\text{-}precond\text{-}f\ Va\ t\ p)$ **and**
$a18{:}lsizes\ x = lsizes\ (mp\text{-}alloc\text{-}stm4\text{-}pre\text{-}precond\text{-}f\ Va\ t\ p)$
shows $\forall j<length\ (free\text{-}list\ (levels\ (mem\text{-}pool\text{-}info\ x\ p)\ !\ ii)).$
$\exists n<n\text{-}max\ (mem\text{-}pool\text{-}info\ x\ p) * 4\ \hat{}\ ii.$
$free\text{-}list\ (levels\ (mem\text{-}pool\text{-}info\ x\ p)\ !\ ii)\ !\ j =$
$buf\ (mem\text{-}pool\text{-}info\ x\ p) + n * (max\text{-}sz\ (mem\text{-}pool\text{-}info\ x\ p)\ div\ 4\ \hat{}\ ii)$

proof−
{ let $?i1=(nat\ (from\text{-}l\ Va\ t))$ **and**
$?j1= (block\text{-}num\ (mem\text{-}pool\text{-}info\ Va\ p)\ (blk\ Va\ t)\ (lsizes\ Va\ t\ !\ nat\ (from\text{-}l\ Va\ t)))$ **and**
$?i2 = (nat\ (from\text{-}l\ Va\ t + 1))$ **and**
$?j2 = (4{*}block\text{-}num\ (mem\text{-}pool\text{-}info\ Va\ p)\ (blk\ Va\ t)\ (lsizes\ Va\ t\ !\ nat\ (from\text{-}l\ Va\ t)))$
let $?mp = mem\text{-}pool\text{-}info\ x\ p$
let $?bts = bits\ (levels\ ?mp\ !\ ii)$ **and** $?fl = free\text{-}list\ (levels\ ?mp\ !\ ii)$
fix j
assume $a00{:}j<length\ ?fl$
have $inv\text{-}bitmap2{:}(\forall j<length\ (free\text{-}list\ (levels\ (mem\text{-}pool\text{-}info\ Va\ p)\ !\ ii)).$
$\exists n<n\text{-}max\ (mem\text{-}pool\text{-}info\ Va\ p) * 4\ \hat{}\ ii.$
$free\text{-}list\ (levels\ (mem\text{-}pool\text{-}info\ Va\ p)\ !\ ii)\ !\ j =$
$buf\ (mem\text{-}pool\text{-}info\ Va\ p) + n * (max\text{-}sz\ (mem\text{-}pool\text{-}info\ Va\ p)\ div\ 4\ \hat{}\ ii))$
using $a15\ a0\ \ a16\ \ mp\text{-}alloc\text{-}stm4\text{-}lvl\text{-}len[OF\ a0\ a8]$
unfolding $Let\text{-}def\ inv\text{-}bitmap\text{-}freelist\text{-}def$
by $fastforce+$
have $from\text{-}l\text{-}gt0{:}0 \le from\text{-}l\ Va\ t$ **using** $a1\ a2$ **by** $linarith$
have $len\text{-}levels{:}length\ (levels\ (mem\text{-}pool\text{-}info\ x\ p)) = length\ (levels\ (mem\text{-}pool\text{-}info\ Va\ p))$
using $mp\text{-}alloc\text{-}stm4\text{-}lvl\text{-}len[OF\ a0\ a8]$ **by** $simp$
have $maxsz{:}max\text{-}sz\ (mem\text{-}pool\text{-}info\ x\ p) = max\text{-}sz\ (mem\text{-}pool\text{-}info\ Va\ p)$
using $mp\text{-}alloc\text{-}stm4\text{-}maxsz[OF\ a0\ a8]$ **by** $simp$
have $buf{:}buf\ (mem\text{-}pool\text{-}info\ x\ p) = buf\ (mem\text{-}pool\text{-}info\ Va\ p)$
using $mp\text{-}alloc\text{-}stm4\text{-}buf[OF\ a0\ a8]$ **by** $simp$
have $from\text{-}l{:}from\text{-}l\ x = from\text{-}l\ Va$
using $mp\text{-}alloc\text{-}stm4\text{-}froml[OF\ a9]$ **by** $auto$
have $mem\text{-}pools{:}mem\text{-}pools\ x = mem\text{-}pools\ Va$ **using** $mp\text{-}alloc\text{-}stm4\text{-}mempools[OF\ a8]$ **by** $auto$

have *lsizes-x-va*:*lsizes x* = *lsizes Va*
by (*simp add*: *a12 mp-alloc-stm4-pre-precond-f-lsz*)
have *len-eq*:*length* (*bits* (*levels* (*mem-pool-info x p*) ! *ii*)) =
length (*bits* (*levels* (*mem-pool-info Va p*) ! *ii*))
using *a16 a8 mp-alloc-stm4-inv-bits-len*
unfolding *gvars-conf-stable-def gvars-conf-def*
by *fastforce*
have *inv-mempool-info-mp Va p*
using *a15 mem-pools a0* **unfolding** *inv-mempool-info-def* **by** *auto*
note *inv-mempool*=*this*[*simplified Let-def*]
have *a15'*:*ii* < *length* (*levels* (*mem-pool-info Va p*))
using *a16 mp-alloc-stm4-inv-mif-len*
by (*simp add*: *len-levels*)
have *nmax*: *n-max* (*mem-pool-info x p*) = *n-max* (*mem-pool-info Va p*)
using *a8* **unfolding** *gvars-conf-stable-def gvars-conf-def* **apply** *auto*
by (*metis mp-alloc-stm4-nmax*)
{ assume *a03*:*ii*≠ *?i2*
then have ∃ *n*<*n-max ?mp* ∗ *4* ˆ *ii*. *?fl* ! *j* = *buf ?mp* + *n* ∗ (*max-sz ?mp div 4* ˆ *ii*)
using *a0 a00 a10 buf eq-free-list-mp-alloc-stm4-pre-precond-f*
inv-bitmap2 maxsz nmax
by (*simp add*: *eq-free-list-mp-alloc-stm4-pre-precond-f mp-alloc-stm4-pre-precond-f-froml*)
}
moreover {
assume *a03*:*ii*= *?i2*
then have *suc-from-l-lt-lsize*:(*nat* (*from-l Va t*)) + *1*< *length* (*lsizes Va t*)
using *a4 a5 from-l-gt0* **by** *linarith*
then have *lsize-i*:*lsizes Va t* ! *nat* (*from-l Va t*) =
ALIGN4 (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ
(*nat* (*from-l Va t*))
using *a11 add-lessD1 suc-from-l-lt-lsize* **by** *blast*
then have *block-n*:(*block-num* (*mem-pool-info Va p*)
(*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*))) = *n*
using *block-n a0 a3 a4 from-l-gt0 a15 a7* **by** *blast*
have *lsize-ii*:*lsizes Va t* ! *ii* =
ALIGN4 (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *ii*
using *a11 from-l-gt0 suc-from-l-lt-lsize*
by (*simp add*: *a03*)
{assume *a04*:*j*< *length* (*free-list* (*levels* (*mem-pool-info Va p*) ! *ii*))
then have *free-list* (*levels* (*mem-pool-info Va p*) ! *ii*) ! *j* = *?fl* ! *j*
using *a14*[*simplified mp-alloc-stm4-pre-precond-f-froml eq-free-list-mp-alloc-stm4-pre-precond-f*] *a03*
unfolding *inserts-def*
by (*simp add*: *nth-append*)
moreover have ∃ *n*<*n-max* (*mem-pool-info Va p*) ∗ *4* ˆ *ii*.
free-list (*levels* (*mem-pool-info Va p*) ! *ii*) ! *j* =
buf (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ˆ *ii*)
using *a04 inv-bitmap2* **by** *fastforce*

ultimately have $\exists$ *n*<*n-max ?mp* ∗ *4* ˆ *ii*. *?fl* ! *j* = *buf ?mp* + *n* ∗ (*max-sz ?mp div 4* ˆ *ii*)
using *buf maxsz nmax* **by** *auto*
}
moreover { assume *a04*:*j* = *length* (*free-list* (*levels* (*mem-pool-info Va p*) ! *ii*))
then have *fl-lsizes*:*?fl* ! *j* = *lsizes Va t* ! *nat* (*from-l Va t* + *1*) ∗ *1* + *blk Va t*
using *free-list-x*[*OF a14*] *a03 a9* *eq-free-list-mp-alloc-stm4-pre-precond-f nth-append-length*
by (*metis* (*no-types*, *lifting*) *a18 from-l lsizes-x-va mp-alloc-stm4-blk*)
let *?nb* = (*block-num* (*mem-pool-info Va p*)
(*buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ˆ *nat* (*from-l Va t*)))
(*lsizes Va t* ! *nat* (*from-l Va t*)) ∗ *4* + *1*)
have *eq-suc-from-l*:*nat* (*from-l Va t* + *1*) = *nat* (*from-l Va t*) + *1* **using** *from-l-gt0* **by** *auto*
from *fl-lsizes*[*simplified a7 this*] **have** *?fl* ! *j* =
addr (*max-sz* (*mem-pool-info Va p*)) (*buf* (*mem-pool-info Va p*)) (*nat* (*from-l Va t*) + *1*)
(*block-num* (*mem-pool-info Va p*)
(*buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ˆ *nat* (*from-l Va t*)))
(*lsizes Va t* ! *nat* (*from-l Va t*)) ∗ *4* + *1*)
using *spec*[*OF lsizes-addr*[*OF a0 a11 a13 a15 suc-from-l-lt-lsize*], *of 1*]
by *auto*
moreover have *length* (*bits* (*levels* (*mem-pool-info x p*) ! *ii*)) =
n-max (*mem-pool-info x p*) ∗ *4* ˆ *ii* **using** *a15 a0*
unfolding *inv-mempool-info-def Let-def*
by (*simp add*: *a15′ len-eq nmax*)
moreover have *?nb* < *n-max* (*mem-pool-info x p*) ∗ *4* ˆ *ii*
using *a6 a03 a0 nmax lsize-ii lsize-i eq-suc-from-l*
inv-mempool-info-maxsz-align4[*OF conjunct1*[*OF conjunct2*[*OF conjunct2*[*OF a15*]]], *simplified a0*]
unfolding *block-num-def*
by *auto*
ultimately have
?nb < *n-max ?mp* ∗ *4* ˆ *ii* ∧ *?fl* ! *j* = *buf ?mp* + *?nb* ∗ (*max-sz ?mp div 4* ˆ *ii*)
using *block-n a7 buf nmax maxsz a6 a03 eq-suc-from-l* **unfolding** *addr-def*
by *auto*
then have $\exists$ *n*<*n-max ?mp* ∗ *4* ˆ *ii*. *?fl* ! *j* = *buf ?mp* + *n* ∗ (*max-sz ?mp div 4* ˆ *ii*) **by** *auto*
}
moreover { assume *a04*:*j* = *Suc* (*length* (*free-list* (*levels* (*mem-pool-info Va p*) ! *ii*)))
then have *fl-lsizes*:*?fl* ! *j* = *lsizes Va t* ! *nat* (*from-l Va t* + *1*) ∗ *2* + *blk Va t*
using *free-list-x*[*OF a14*] *a03 a9* *eq-free-list-mp-alloc-stm4-pre-precond-f*

nth-append-length a18 from-l lsizes-x-va mp-alloc-stm4-blk
by (*metis add.right-neutral add-Suc-right nth-Cons-Suc nth-append-length-plus*)
let *?nb* = (*block-num* (*mem-pool-info Va p*)
(*buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ˆ *nat* (*from-l Va t*)))
(*lsizes Va t* ! *nat* (*from-l Va t*)) ∗ *4* + *2*)
have *eq-suc-from-l*:*nat* (*from-l Va t* + *1*) = *nat* (*from-l Va t*) + *1* **using** *from-l-gt0* **by** *auto*
from *fl-lsizes*[*simplified a7 this*] **have** *?fl* ! *j* =
addr (*max-sz* (*mem-pool-info Va p*)) (*buf* (*mem-pool-info Va p*)) (*nat* (*from-l Va t*) + *1*)
(*block-num* (*mem-pool-info Va p*)
(*buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ˆ *nat* (*from-l Va t*)))
(*lsizes Va t* ! *nat* (*from-l Va t*)) ∗ *4* + *2*)
using *spec*[*OF lsizes-addr*[*OF a0 a11 a13 a15 suc-from-l-lt-lsize*], *of 2 n*]
by *fastforce*
moreover have *length* (*bits* (*levels* (*mem-pool-info x p*) ! *ii*)) =
n-max (*mem-pool-info x p*) ∗ *4* ˆ *ii* **using** *a15 a0*
unfolding *inv-mempool-info-def Let-def*
by (*simp add*: *a15′ len-eq nmax*)
moreover have *?nb* < *n-max* (*mem-pool-info x p*) ∗ *4* ˆ *ii*
using *a6 a03 a0 nmax lsize-ii lsize-i eq-suc-from-l*
inv-mempool-info-maxsz-align4[*OF conjunct1*[*OF conjunct2*[*OF conjunct2*[*OF a15*]]], *simplified a0*]
unfolding *block-num-def*
by *auto*
ultimately have
?nb < *n-max ?mp* ∗ *4* ˆ *ii* ∧ *?fl* ! *j* = *buf ?mp* + *?nb* ∗ (*max-sz ?mp div 4* ˆ *ii*)
using *block-n a7 buf nmax maxsz a6 a03 eq-suc-from-l* **unfolding** *addr-def*
by *auto*
then have ∃ *n*<*n-max ?mp* ∗ *4* ˆ *ii*. *?fl* ! *j* = *buf ?mp* + *n* ∗ (*max-sz ?mp div 4* ˆ *ii*) **by** *auto*
}
moreover { **assume** *a04*:*j* = *Suc* (*Suc* (*length* (*free-list* (*levels* (*mem-pool-info Va p*) ! *ii*))))
then have *fl-lsizes*:*?fl* ! *j* = *lsizes Va t* ! *nat* (*from-l Va t* + *1*) ∗ *3* + *blk Va t*
using *free-list-x*[*OF a14*] *a03 a9 eq-free-list-mp-alloc-stm4-pre-precond-f*
nth-append-length a18 from-l lsizes-x-va mp-alloc-stm4-blk
by (*metis add.right-neutral add-Suc-right nth-Cons-Suc nth-append-length-plus*)
let *?nb* = (*block-num* (*mem-pool-info Va p*)
(*buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ˆ *nat* (*from-l Va t*)))
(*lsizes Va t* ! *nat* (*from-l Va t*)) ∗ *4* + *3*)
have *eq-suc-from-l*:*nat* (*from-l Va t* + *1*) = *nat* (*from-l Va t*) + *1* **using** *from-l-gt0* **by** *auto*
from *fl-lsizes*[*simplified a7 this*] **have** *?fl* ! *j* =

addr (*max-sz* (*mem-pool-info Va p*)) (*buf* (*mem-pool-info Va p*)) (*nat* (*from-l Va t*) + *1*)
(*block-num* (*mem-pool-info Va p*)
(*buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ˆ *nat* (*from-l Va t*)))
(*lsizes Va t* ! *nat* (*from-l Va t*)) ∗ *4* + *3*)
using *spec*[*OF lsizes-addr*[*OF a0 a11 a13 a15 suc-from-l-lt-lsize*], *of 3 n*]
by *fastforce*
moreover have *length* (*bits* (*levels* (*mem-pool-info x p*) ! *ii*)) =
n-max (*mem-pool-info x p*) ∗ *4* ˆ *ii* **using** *a15 a0*
unfolding *inv-mempool-info-def Let-def*
by (*simp add*: *a15′ len-eq nmax*)
moreover have *?nb* < *n-max* (*mem-pool-info x p*) ∗ *4* ˆ *ii*
using *a6 a03 a0 nmax lsize-ii lsize-i eq-suc-from-l*
inv-mempool-info-maxsz-align4[*OF conjunct1*[*OF conjunct2*[*OF conjunct2*[*OF a15*]]], *simplified a0*]
unfolding *block-num-def*
by *auto*
ultimately have
?nb < *n-max ?mp* ∗ *4* ˆ *ii* ∧ *?fl* ! *j* = *buf ?mp* + *?nb* ∗ (*max-sz ?mp div 4* ˆ *ii*)
using *block-n a7 buf nmax maxsz a6 a03 eq-suc-from-l* **unfolding** *addr-def*
by *auto*
then have ∃ *n*<*n-max ?mp* ∗ *4* ˆ *ii*. *?fl* ! *j* = *buf ?mp* + *n* ∗ (*max-sz ?mp div 4* ˆ *ii*) **by** *auto*
} **ultimately have** ∃ *n*<*n-max ?mp* ∗ *4* ˆ *ii*. *?fl* ! *j* = *buf ?mp* + *n* ∗ (*max-sz ?mp div 4* ˆ *ii*)
using *a00 free-list-x*[*OF a14*,
simplified eq-free-list-mp-alloc-stm4-pre-precond-f mp-alloc-stm4-pre-precond-f-froml] *a03*
by *fastforce*
} **ultimately have** ∃ *n*<*n-max ?mp* ∗ *4* ˆ *ii*. *?fl* ! *j* = *buf ?mp* + *n* ∗ (*max-sz ?mp div 4* ˆ *ii*)
by *auto*
} **then show** *?thesis* **by** *auto*
qed

lemma *next-block-less-length-bits*:
assumes
a0:*n* < *length* (*bits* (*levels pi* ! *ii*)) **and**
a1:(*ii*+*1*) < *length* (*levels pi*) **and**
a2:(∀ *i*<*length* (*levels pi*).
length (*bits* (*levels pi* ! *i*)) = *n-max pi* ∗ *4* ˆ *i*)
shows *4*∗*n* + *3* < *length* (*bits* (*levels pi* ! (*ii*+*1*)))
proof−
have *n*< *n-max pi* ∗ *4* ˆ *ii* **using** *a0 a1 a2* **by** *auto*
moreover have *length* (*bits* (*levels pi*!(*ii*+*1*))) = *n-max pi* ∗ *4* ˆ(*ii*+*1*) **using** *a1 a2* **by** *auto*
ultimately show *?thesis* **by** *auto*

qed

lemma *distinct-lists*: **assumes**
a0:*distinct l1* **and**
a1:*distinct l2* **and**
a2:∀ *e*∈*set l2*. *e* ∉ *set l1*
shows *distinct* (*l1* @ *l2*)
  using *assms*
  by(*induct l1*, *auto*)

lemma *free-list-updates-inv3*:
  assumes *a0*:*p* ∈ *mem-pools Va* **and**
 a1:¬ *free-l Va t* < *OK* **and**
 a2:*free-l Va t* ≤ *from-l Va t* **and**
 a3:*alloc-l Va t* < *int* (*n-levels* (*mem-pool-info Va p*)) **and**
 a4:*from-l Va t* < *alloc-l Va t* **and**
a5:*alloc-l Va t* = *int* (*length* (*lsizes Va t*)) − *1* ∧ *length* (*lsizes Va t*) = *n-levels* (*mem-pool-info Va p*) ∨
 alloc-l Va t = *int* (*length* (*lsizes Va t*)) − *2* ∧ *lsizes Va t* ! *nat* (*alloc-l Va t* + *1*) < *sz* **and**
 a6:*block-num* (*mem-pool-info Va p*)
  (*buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ˆ *nat* (*from-l Va t*)))
  (*lsizes Va t* ! *nat* (*from-l Va t*))
 < *n-max* (*mem-pool-info Va p*) ∗ *4* ˆ *nat* (*from-l Va t*) **and**
 a7:*blk Va t* = *buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ˆ *nat* (*from-l Va t*)) **and**
 a8:(*x*, *mp-alloc-stm4-pre-precond-f Va t p*) ∈ *gvars-conf-stable* **and**
 a9:*from-l x* = *from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
 a10:*freeing-node x* = *freeing-node* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
 a11:*allocating-node x* = *allocating-node* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
 a12:∀ *pa*. *pa* ≠ *p* ⟶ *mem-pool-info x pa* = *mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *pa* **and**
 a13:∀ *jj*. *jj* ≠ *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*) ⟶
    levels (*mem-pool-info x p*) ! *jj* = *levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) ! *jj* **and**
a14:∀ *ii*<*length* (*lsizes Va t*). *lsizes Va t* ! *ii* = *ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *ii* **and**
a15:*i x t* = *4* **and**
a16:*lsizes x* = *lsizes* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
a17:*length* (*lsizes Va t*) ≤ *n-levels* (*mem-pool-info Va p*) **and**
a18:*bits* (*levels* (*mem-pool-info x p*) ! *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*)) =
 list-updates-n
  (*bits* (*levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) !
        nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*)))
  (*Suc* (*bn* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* ∗ *4*)) *3 FREE* **and**
a19:

free-list (*levels* (*mem-pool-info x p*) ! *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*)) =
inserts
(*map* (*λii. lsizes* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*) ∗
ii +
blk (*mp-alloc-stm4-pre-precond-f Va t p*) *t*)
[*Suc NULL..<4*])
(*free-list*
(*levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*))) **and**
a20:*allocating-node Va t* =
Some (|*pool* = *p*, *level* = *nat* (*from-l Va t*),
block = *block-num* (*mem-pool-info Va p*)
(*buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div* *4* ˆ *nat* (*from-l Va t*)))
(*lsizes Va t* ! *nat* (*from-l Va t*)),
data = *buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div* *4* ˆ *nat* (*from-l Va t*))|) **and**
a21:*inv-aux-vars Va* ∧ *inv-bitmap Va* ∧ *inv-mempool-info Va* ∧ *inv-bitmap-freelist Va* **and**
a22:*ii* < *length* (*levels* (*mem-pool-info x p*)) **and**
a23:*blk x* = *blk* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
a24:*lsizes x* = *lsizes* (*mp-alloc-stm4-pre-precond-f Va t p*)
shows *distinct* (*free-list* (*levels* (*mem-pool-info x p*) ! *ii*))
proof−
{
let *?i1*=(*nat* (*from-l Va t*)) **and**
?j1= (*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*))) **and**
?i2 = (*nat* (*from-l Va t* + *1*)) **and**
?j2 = (*4*∗*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*)))
let *?mp* = *mem-pool-info x p*
let *?bts* = *bits* (*levels ?mp* ! *ii*) **and** *?fl* = *free-list* (*levels ?mp* ! *ii*)
have *inv-bitmap1*:(∀ *j*<*length* (*bits* (*levels* (*mem-pool-info Va p*) ! *ii*)).
(*get-bit-s Va p ii j* = *FREE*) =
(*buf* (*mem-pool-info Va p*) + *j* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ˆ *ii*)
∈ *set* (*free-list* (*levels* (*mem-pool-info Va p*) ! *ii*)))) **and**
inv-bitmap2:(∀ *j*<*length* (*free-list* (*levels* (*mem-pool-info Va p*) ! *ii*)).
∃ *n*<*n-max* (*mem-pool-info Va p*) ∗ *4* ˆ *ii*.
free-list (*levels* (*mem-pool-info Va p*) ! *ii*) ! *j* =
buf (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ˆ *ii*)) **and**
inv-bitmap3:*distinct* (*free-list* (*levels* (*mem-pool-info Va p*) ! *ii*))
using *a21 a0 a22 mp-alloc-stm4-lvl-len*[*OF a0 a8*]
unfolding *Let-def inv-bitmap-freelist-def*
by *fastforce*+

have *from-l-gt0*:*0* ≤ *from-l Va t* **using** *a1 a2* **by** *linarith*
have *len-levels*:*length* (*levels* (*mem-pool-info x p*)) = *length* (*levels* (*mem-pool-info Va p*))
using *mp-alloc-stm4-lvl-len*[*OF a0 a8*] **by** *simp*
have *maxsz*:*max-sz* (*mem-pool-info x p*) = *max-sz* (*mem-pool-info Va p*)
using *mp-alloc-stm4-maxsz*[*OF a0 a8*] **by** *simp*
have *buf*:*buf* (*mem-pool-info x p*) = *buf* (*mem-pool-info Va p*)
using *mp-alloc-stm4-buf*[*OF a0 a8*] **by** *simp*
have *from-l*:*from-l x* = *from-l Va*
using *mp-alloc-stm4-froml*[*OF a9*] **by** *auto*
have *from-l-suc*:*nat* (*from-l Va t* + *1*) = *nat*(*from-l Va t*) + *1*
using *from-l-gt0* **by** *auto*
have *mem-pools*:*mem-pools x* = *mem-pools Va* **using** *mp-alloc-stm4-mempools*[*OF a8*] **by** *auto*
have *lsizes-x-va*:*lsizes x* = *lsizes Va*
by (*simp add*: *a16 mp-alloc-stm4-pre-precond-f-lsz*)
have *len-eq*:*length* (*bits* (*levels* (*mem-pool-info x p*) ! *ii*)) =
length (*bits* (*levels* (*mem-pool-info Va p*) ! *ii*))
using *a22 a8 mp-alloc-stm4-inv-bits-len*
unfolding *gvars-conf-stable-def gvars-conf-def*
by *fastforce*
have *inv-mempool-info-mp Va p*
using *a21 mem-pools a0* **unfolding** *inv-mempool-info-def* **by** *auto*
note *inv-mempool*=*this*[*simplified Let-def*]
have *a22'*:*ii* < *length* (*levels* (*mem-pool-info Va p*))
using *a22 mp-alloc-stm4-inv-mif-len*
by (*simp add*: *len-levels*)
{ assume *a03*:*ii*≠*?i2*
have *free-list* (*levels* (*mem-pool-info x p*) ! *ii*) =
free-list (*levels* (*mem-pool-info Va p*) ! *ii*)
using *free-level-x-va*[*OF a13*] *a03 a9 from-l* **by** *metis*
then have *distinct* (*free-list* (*levels* (*mem-pool-info x p*) ! *ii*))
using *inv-bitmap3* **by** *auto*
}
moreover { assume *a03*:*ii*=*?i2*
then have *block-n*:(*block-num* (*mem-pool-info Va p*)
(*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*))) = *n*
proof−
have *lsizes Va t* ! *nat* (*from-l Va t*) =
ALIGN4 (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ
(*nat* (*from-l Va t*))
using *a14 lsizes-x-va a16 a1 a2 a4 a5 a9 from-l* **by** *auto*
thus *?thesis* **using** *block-n a21 a0 a0 a7 a3 a4 from-l-gt0*
by *blast*
qed
then have *get-bit-s Va p* (*nat* (*from-l Va t*)) *n* = *ALLOCATING*
using *a20 a13 a21 a7* **unfolding** *inv-aux-vars-def*
by (*metis Mem-block.select-convs*(*1*) *Mem-block.select-convs*(*2*) *Mem-block.select-convs*(*3*))
moreover have *n-len*: *n*< *length* (*bits* (*levels* (*mem-pool-info Va p*) ! *nat*

(*from-l Va t*)))
using *a03 a22′ a6 a7 inv-mempool local.block-n* **by** *auto*
ultimately have *noexist-bits* (*mem-pool-info Va p*) *ii* (*n* ∗ *4*)
using *allocating-next-notexists*[*OF conjunct1*[*OF conjunct2*[*OF a21*]] *a0* -]
a21 a0 a03 a21 a0 a22′
from-l-gt0 from-l-suc inv-mempool **by** *auto*
then have *get-bit-s Va p ii* (*n*∗*4* + *1*) ≠ *FREE* ∧
get-bit-s Va p ii (*n*∗*4* + *2*) ≠ *FREE* ∧
get-bit-s Va p ii (*n*∗*4* + *3*) ≠ *FREE*
by (*simp add*: *mult.commute*)
moreover have *n*∗*4* + *3* < *length* (*bits* (*levels* (*mem-pool-info Va p*) ! *nat*
(*from-l Va t* + *1*)))
using *a03 a22′ n-len inv-mempool from-l-gt0 next-block-less-length-bits from-l-suc*
by *simp*
ultimately have *not-in-freelist*:(*buf* (*mem-pool-info Va p*) + (*n*∗*4* + *1*) ∗
(*max-sz* (*mem-pool-info Va p*) *div 4* ˆ *ii*)
∉ *set* (*free-list* (*levels* (*mem-pool-info Va p*) ! *ii*))) ∧
(*buf* (*mem-pool-info Va p*) + (*n*∗*4* + *2*) ∗ (*max-sz* (*mem-pool-info Va p*)
div 4 ˆ *ii*)
∉ *set* (*free-list* (*levels* (*mem-pool-info Va p*) ! *ii*))) ∧
(*buf* (*mem-pool-info Va p*) + (*n*∗*4* + *3*) ∗ (*max-sz* (*mem-pool-info Va p*)
div 4 ˆ *ii*)
∉ *set* (*free-list* (*levels* (*mem-pool-info Va p*) ! *ii*)))
using *inv-bitmap1 a03*
by (*metis* (*no-types*, *lifting*) *add-lessD1 numeral-3-eq-3*
one-add-one plus-1-eq-Suc semiring-normalization-rules(*21*))
obtain *m* **where** *max-sz*:*max-sz* (*mem-pool-info Va p*) = *4* ∗ *m* ∗ *4* ˆ *n-levels*
(*mem-pool-info Va p*)
using *a21 a0* **unfolding** *inv-mempool-info-def Let-def* **by** *auto*
have *ls*:*4* ˆ *ii dvd 4* ∗ *m* ∗ *4* ˆ *n-levels* (*mem-pool-info Va p*) **using** *a03 a22*
by (*metis dvd-triv-right inv-mempool len-levels less-imp-le-nat power-le-dvd*)

have *suc-from-l-lt-lsize*:(*nat* (*from-l Va t*)) + *1*< *length* (*lsizes Va t*)
using *a4 a5 from-l-gt0* **by** *linarith*
have *b2*:∀ *j*. (*lsizes Va t* ! *nat* (*from-l Va t* + *1*)) ∗ *j* + *blk Va t* =
addr (*max-sz* (*mem-pool-info Va p*)) (*buf* (*mem-pool-info Va p*))
(*nat* (*from-l Va t* + *1*))
((*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat*
(*from-l Va t*)))∗*4* + *j*)
using *lsizes-addr*[*OF a0 a14 a17 a21 suc-from-l-lt-lsize*] *a7 from-l-gt0 block-n*
by (*simp add*: *Suc-nat-eq-nat-zadd1 add.commute*)
then have *b2*:∀ *j*. *lsizes* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*) ∗ *j* +
blk (*mp-alloc-stm4-pre-precond-f Va t p*) *t* =
addr (*max-sz* (*mem-pool-info Va p*)) (*buf* (*mem-pool-info Va p*))
(*nat* (*from-l Va t* + *1*))
((*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat*
(*from-l Va t*)))∗*4* + *j*)
by (*metis mp-alloc-stm4-blk mp-alloc-stm4-pre-precond-f-froml mp-alloc-stm4-pre-precond-f-lsz*)

then have *distinct* (*free-list* (*levels* (*mem-pool-info x p*) ! *ii*))
proof−
have *h1*:*distinct* [*lsizes* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* ! *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*) ∗ *1* +
blk (*mp-alloc-stm4-pre-precond-f Va t p*) *t*,
lsizes (*mp-alloc-stm4-pre-precond-f Va t p*) *t* ! *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*) ∗ *2* +
blk (*mp-alloc-stm4-pre-precond-f Va t p*) *t*,
lsizes (*mp-alloc-stm4-pre-precond-f Va t p*) *t* ! *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*) ∗ *3* +
blk (*mp-alloc-stm4-pre-precond-f Va t p*) *t*] **using** *b2 a03 a22′ inv-mempool mp-alloc-stm3-lm2-inv-1-2* **unfolding** *addr-def*
by (*smt add-diff-cancel-left′ distinct-length-2-or-more*
distinct-singleton mult-cancel-right nat-less-le num.distinct(*3*) *num.distinct*(*5*)
numeral-eq-iff numeral-eq-one-iff semiring-norm(*85*))
have *h2*:∀ *e*∈*set* [*lsizes* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* ! *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*) ∗ *1* +
blk (*mp-alloc-stm4-pre-precond-f Va t p*) *t*,
lsizes (*mp-alloc-stm4-pre-precond-f Va t p*) *t* ! *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*) ∗ *2* +
blk (*mp-alloc-stm4-pre-precond-f Va t p*) *t*,
lsizes (*mp-alloc-stm4-pre-precond-f Va t p*) *t* ! *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*) ∗ *3* +
blk (*mp-alloc-stm4-pre-precond-f Va t p*) *t*].
e ∉ *set* (*free-list* (*levels* (*mem-pool-info Va p*) ! *ii*))
using *b2 a03 not-in-freelist local.block-n* **unfolding** *addr-def* **apply** *auto*
by (*metis* (*no-types*) *not-in-freelist semiring-normalization-rules*(*12*))
show *?thesis*

using *distinct-lists*[*OF inv-bitmap3 h1 h2*] *free-list-x*[*OF a19*]
by (*metis a03 eq-free-list-mp-alloc-stm4-pre-precond-f mp-alloc-stm4-pre-froml*)

qed
} **ultimately have** *distinct* (*free-list* (*levels* (*mem-pool-info x p*) ! *ii*))
by *auto*

} **then show** *?thesis* **by** *auto*
qed

lemma *mp-alloc-stm4-inv-bitmap-freelist*:
assumes *a0*:*p* ∈ *mem-pools Va* **and**
a1:¬ *free-l Va t* < *OK* **and**
a2:*free-l Va t* ≤ *from-l Va t* **and**
a3:*alloc-l Va t* < *int* (*n-levels* (*mem-pool-info Va p*)) **and**

$a4$:*from-l Va t* $<$ *alloc-l Va t* **and**
$a4'$:*alloc-l Va t* $=$ *int* (*length* (*lsizes Va t*)) $-$ *1* $\wedge$ *length* (*lsizes Va t*) $=$ *n-levels* (*mem-pool-info Va p*) $\vee$
alloc-l Va t $=$ *int* (*length* (*lsizes Va t*)) $-$ *2* $\wedge$ *lsizes Va t* ! *nat* (*alloc-l Va t* $+$ *1*) $<$ *sz* **and**
$a5$:*block-num* (*mem-pool-info Va p*)
(*buf* (*mem-pool-info Va p*) $+$ *n* $*$ (*max-sz* (*mem-pool-info Va p*) *div 4* ˆ *nat* (*from-l Va t*)))
(*lsizes Va t* ! *nat* (*from-l Va t*))
$<$ *n-max* (*mem-pool-info Va p*) $*$ *4* ˆ *nat* (*from-l Va t*) **and**
$a6$:*blk Va t* $=$ *buf* (*mem-pool-info Va p*) $+$ *n* $*$ (*max-sz* (*mem-pool-info Va p*) *div* *4* ˆ *nat* (*from-l Va t*)) **and**
$a7$:(*x*, *mp-alloc-stm4-pre-precond-f Va t p*) $\in$ *gvars-conf-stable* **and**
$a8$:*from-l x* $=$ *from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
$a9$:*freeing-node x* $=$ *freeing-node* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
$a10$:*allocating-node x* $=$ *allocating-node* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
$a11$:$\forall$ *pa*. *pa* $\neq$ *p* $\longrightarrow$ *mem-pool-info x pa* $=$ *mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *pa* **and**
$a12$:$\forall$ *jj*. *jj* $\neq$ *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* $+$ *1*) $\longrightarrow$
levels (*mem-pool-info x p*) ! *jj* $=$ *levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) ! *jj* **and**
$a12'$:$\forall$ *ii*$<$*length* (*lsizes Va t*). *lsizes Va t* ! *ii* $=$ *ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *ii* **and**
$a12''$:*i x t* $=$ *4* **and**
$a12'''$:*lsizes x* $=$ *lsizes* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
$a12''''$:*length* (*lsizes Va t*) $\leq$ *n-levels* (*mem-pool-info Va p*) **and**
$a13$:*bits* (*levels* (*mem-pool-info x p*) ! *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* $+$ *1*)) $=$
list-updates-n
(*bits* (*levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* $+$ *1*)))
(*Suc* (*bn* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* $*$ *4*)) *3 FREE* **and**
$a14$:
free-list (*levels* (*mem-pool-info x p*) ! *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* $+$ *1*)) $=$
inserts
(*map* (λ*ii*. *lsizes* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* $+$ *1*) $*$
ii $+$
blk (*mp-alloc-stm4-pre-precond-f Va t p*) *t*)
[*Suc NULL*..$<$*4*])
(*free-list*
(*levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* $+$ *1*))) **and**
$a15$: *inv-mempool-info Va* **and**
$a16$:*inv-bitmap-freelist Va* **and**
$a17$:*allocating-node Va t* $=$
Some ⦇*pool* $=$ *p*, *level* $=$ *nat* (*from-l Va t*),
block $=$ *block-num* (*mem-pool-info Va p*)

(*buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div* *4* ˆ *nat* (*from-l Va t*)))
(*lsizes Va t* ! *nat* (*from-l Va t*)),
data = *buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div* *4* ˆ *nat* (*from-l Va t*))⟧ **and**
a18:*inv-aux-vars Va* ∧ *inv-bitmap Va* ∧ *inv-mempool-info Va* ∧ *inv-bitmap-freelist Va* **and**
a19:*blk x* = *blk* (*mp-alloc-stm4-pre-precond-f Va t p*)
shows*inv-bitmap-freelist x*
proof−
{ **fix** *p*′
assume *a00*:*p*′∈*mem-pools x*
{**assume** *p*≠*p*′
moreover have *mem-pool-info x p*′ = *mem-pool-info Va p*′
using *mp-alloc-stm4-pres-mpinfo*
by (*metis a11 calculation*)
ultimately have *inv-bitmap-freelist-mp x p*′
using *a18 a00 mp-alloc-stm4-lvl-len*[*OF a0 a7*] *mp-alloc-stm4-maxsz*[*OF a0 a7*]
mp-alloc-stm4-buf[*OF a0 a7*] *mp-alloc-stm4-froml*[*OF a8*] *mp-alloc-stm4-mempools*[*OF a7*]
by(*simp add*: *inv-bitmap-freelist-def Let-def*)
}
moreover { **assume** *eq-p*:*p*=*p*′
let *?mp* = *mem-pool-info x p*′
have *inv-mempool-info-mp Va p*′
using *a15 eq-p mp-alloc-stm4-mempools*[*OF a7*] *a00* **unfolding** *inv-mempool-info-def* **by** *auto*
note *inv-mempool*=*this*[*simplified Let-def*]
{**fix** *i*
assume *a01*:*i*<*length* (*levels ?mp*)
then have *inv-bitmap1*:(∀ *j*<*length* (*bits* (*levels* (*mem-pool-info Va p*′) ! *i*)).
(*get-bit-s Va p*′ *i j* = *FREE*) =
(*buf* (*mem-pool-info Va p*′) + *j* ∗ (*max-sz* (*mem-pool-info Va p*′) *div 4* ˆ *i*)
∈ *set* (*free-list* (*levels* (*mem-pool-info Va p*′) ! *i*)))) **and**
inv-bitmap2:(∀ *j*<*length* (*free-list* (*levels* (*mem-pool-info Va p*′) ! *i*)).
∃ *n*<*n-max* (*mem-pool-info Va p*′) ∗ *4* ˆ *i*.
free-list (*levels* (*mem-pool-info Va p*′) ! *i*) ! *j* =
buf (*mem-pool-info Va p*′) + *n* ∗ (*max-sz* (*mem-pool-info Va p*′) *div 4* ˆ *i*)) **and**
inv-bitmap3:*distinct* (*free-list* (*levels* (*mem-pool-info Va p*′) ! *i*))
using *a16 eq-p mp-alloc-stm4-mempools*[*OF a7*] *a00 a01 mp-alloc-stm4-lvl-len*[*OF a0 a7*]
unfolding *Let-def inv-bitmap-freelist-def*
by *fastforce*+
let *?bts* = *bits* (*levels ?mp* ! *i*) **and** *?fl* = *free-list* (*levels ?mp* ! *i*)
have *f1*:(∀ *j*<*length ?bts*. (*?bts* ! *j* = *FREE*) = (*buf ?mp* + *j* ∗ (*max-sz*

?mp div 4 ^ i) ∈ *set ?fl*))
using *assms free-list-updates-inv1 a00 a01 eq-p* **by** *blast*
have *f2*: (∀ *j*<*length ?fl*. ∃ *n*<*n-max ?mp* ∗ *4 ^ i*. *?fl* ! *j* = *buf ?mp* + *n* ∗ (*max-sz ?mp div 4 ^ i*))
using *assms free-list-updates-inv2 a00 a01 eq-p* **by** *blast*
have *f3*:*distinct ?fl* **using** *assms free-list-updates-inv3 a00 a01 eq-p* **by** *blast*

note *conjI*[*OF f1 conjI*[*OF f2 f3*]]
} **then have** *inv-bitmap-freelist-mp x p′* **by** *auto*
}
ultimately have *inv-bitmap-freelist-mp x p′* **by** *auto*
}
thus *?thesis* **unfolding** *inv-bitmap-freelist-def* **by** *auto*
qed

lemma *noexists-eq-bits*: **assumes**
a0:∀ *j*. *j*≥ *jj* ∧ *j*≤ *Suc*(*Suc* (*Suc jj*)) ⟶
get-bit-s x p ii j = *get-bit-s Va p ii j* **and**
a1:*noexist-bits* (*mem-pool-info Va p*) *ii jj*
shows *noexist-bits* (*mem-pool-info x p*) *ii jj*
using *a0 a1*
by *simp*

lemma *mp-alloc-stm4-inv-bitmap1*:
assumes
a0:*inv Va* **and**
a1:*p* ∈ *mem-pools Va* **and**
a2:∀ *ii*<*length* (*lsizes Va t*). *lsizes Va t* ! *ii* = *ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4 ^ ii* **and**
a4:*alloc-l Va t* < *int* (*n-levels* (*mem-pool-info Va p*)) **and**
a5:¬ *free-l Va t* < *OK* **and**
a6:*free-l Va t* ≤ *from-l Va t* **and**
a7:*allocating-node Va t* =
Some (|*pool* = *p*, *level* = *nat* (*from-l Va t*),
block = *block-num* (*mem-pool-info Va p*)
(*buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4 ^ nat* (*from-l Va t*)))
(*lsizes Va t* ! *nat* (*from-l Va t*)),
data = *buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4 ^ nat* (*from-l Va t*))|) **and**
a8:*n* = *block-num* (*mem-pool-info Va p*)
(*buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4 ^ nat* (*from-l Va t*)))
(*lsizes Va t* ! *nat* (*from-l Va t*)) ∨
max-sz (*mem-pool-info Va p*) *div 4 ^ nat* (*from-l Va t*) = *NULL* **and**
a9:*block-num* (*mem-pool-info Va p*)
(*buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4 ^ nat* (*from-l Va t*)))
(*lsizes Va t* ! *nat* (*from-l Va t*))

$<$ *n-max* (*mem-pool-info* *Va* *p*) $*$ *4* ^ *nat* (*from-l* *Va* *t*) **and**
a10:*from-l* *Va* *t* $<$ *alloc-l* *Va* *t* **and**

a11:*n* $<$ *n-max* (*mem-pool-info* *Va* *p*) $*$ *4* ^ *nat* (*from-l* *Va* *t*) **and**
a12:*blk* *Va* *t* = *buf* (*mem-pool-info* *Va* *p*) + *n* $*$ (*max-sz* (*mem-pool-info* *Va* *p*) *div* *4* ^ *nat* (*from-l* *Va* *t*)) **and**
a13:(*x*, *mp-alloc-stm4-pre-precond-f* *Va* *t* *p*) $\in$ *gvars-conf-stable* **and**
a14:$\forall$ *jj*. *jj* $\neq$ *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f* *Va* *t* *p*) *t* + *1*) $\longrightarrow$
levels (*mem-pool-info* *x* *p*) ! *jj* = *levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f* *Va* *t* *p*) *p*) ! *jj* **and**
a15:*bits* (*levels* (*mem-pool-info* *x* *p*) ! *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f* *Va* *t* *p*) *t* + *1*)) =
list-updates-n
(*bits* (*levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f* *Va* *t* *p*) *p*) !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f* *Va* *t* *p*) *t* + *1*)))
(*Suc* (*bn* (*mp-alloc-stm4-pre-precond-f* *Va* *t* *p*) *t* $*$ *4*)) *3* *FREE* **and**
a16:*free-list* (*levels* (*mem-pool-info* *x* *p*) ! *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f* *Va* *t* *p*) *t* + *1*)) =
inserts
(*map* (λ*ii*. *lsizes* (*mp-alloc-stm4-pre-precond-f* *Va* *t* *p*) *t* !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f* *Va* *t* *p*) *t* + *1*) $*$
ii +
blk (*mp-alloc-stm4-pre-precond-f* *Va* *t* *p*) *t*)
[*Suc* *NULL*..<*4*])
(*free-list*
(*levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f* *Va* *t* *p*) *p*) !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f* *Va* *t* *p*) *t* + *1*))) **and**
a17:*lsizes* *x* = *lsizes* (*mp-alloc-stm4-pre-precond-f* *Va* *t* *p*) **and**
a18:*from-l* *x* = *from-l* (*mp-alloc-stm4-pre-precond-f* *Va* *t* *p*) **and**
a01:*ii* $<$ *length* (*levels* (*mem-pool-info* *x* *p*)) **and**
a02:*jj* $<$ *length* (*bits* (*levels* (*mem-pool-info* *x* *p*) ! *ii*))
shows (*get-bit-s* *x* *p* *ii* *jj* = *FREE* $\vee$ *get-bit-s* *x* *p* *ii* *jj* = *FREEING* $\vee$ *get-bit-s* *x* *p* *ii* *jj* = *ALLOCATED* $\vee$ *get-bit-s* *x* *p* *ii* *jj* = *ALLOCATING* $\longrightarrow$
(*NULL* $<$ *ii* $\longrightarrow$ *get-bit-s* *x* *p* (*ii* $-$ *1*) (*jj* *div* *4*) = *DIVIDED*) $\wedge$
(*ii* $<$ *length* (*levels* (*mem-pool-info* *x* *p*)) $-$ *1* $\longrightarrow$ *noexist-bits* (*mem-pool-info* *x* *p*) (*ii* + *1*) (*jj* $*$ *4*)))
proof$-$
let *?mp* = *mem-pool-info* *x* *p*
have *inv*:*inv-aux-vars* *Va* $\wedge$ *inv-bitmap* *Va* $\wedge$ *inv-mempool-info* *Va* $\wedge$ *inv-bitmap-freelist* *Va*
using *a0* **unfolding** *inv-def* **by** *auto*
have *from-l-gt0*:*0* $\leq$ *from-l* *Va* *t* **using** *a6* *a5* **by** *linarith*
have *len-levels*:*length* (*levels* (*mem-pool-info* *x* *p*)) = *length* (*levels* (*mem-pool-info* *Va* *p*))
using *mp-alloc-stm4-lvl-len*[*OF* *a1* *a13*] **by** *simp*
have *maxsz*:*max-sz* (*mem-pool-info* *x* *p*) = *max-sz* (*mem-pool-info* *Va* *p*)
using *mp-alloc-stm4-maxsz*[*OF* *a1* *a13*] **by** *simp*
have *buf*:*buf* (*mem-pool-info* *x* *p*) = *buf* (*mem-pool-info* *Va* *p*)
using *mp-alloc-stm4-buf*[*OF* *a1* *a13*] **by** *simp*

have *from-l*:*from-l x* = *from-l Va*
using *mp-alloc-stm4-froml*[*OF a18*] **by** *auto*
have *from-l-suc*:*nat* (*from-l Va t* + *1*) = *nat*(*from-l Va t*) + *1*
using *from-l-gt0* **by** *auto*
have *mem-pools*:*mem-pools x* = *mem-pools Va* **using** *mp-alloc-stm4-mempools*[*OF a13*] **by** *auto*
have *lsizes-x-va*:*lsizes x* = *lsizes Va* **using** *mp-alloc-stm4-pre-precond-f-lsz a17*
by *auto*
let *?i1*=(*nat* (*from-l Va t*)) **and**
?j1= (*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*))) **and**
?i2 = (*nat* (*from-l Va t* + *1*)) **and**
?j2 = (*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*))∗*4*) **and**
?i1′ = (*nat* (*from-l Va t*)) − *1* **and**
?j1′ = (*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*))) *div 4* **and**
?i2′ = (*nat* (*from-l Va t*)) + *2*
let *?j20′* = *?j2* ∗ *4* **and** *?j21′* = (*?j2*+*1*) ∗ *4* **and** *?j22′* = (*?j2*+*2*)∗*4* **and**
?j23′ = (*?j2*+*3*)∗*4* **and** *?j24′* = (*?j2*+*4*)∗*4*
let *?mp* = *mem-pool-info x p*
have *inv-mempool-info-mp Va p*
using *a1 mem-pools inv* **unfolding** *inv-mempool-info-def* **by** *auto*
note *inv-mempool*=*this*[*simplified Let-def*]
have *i1-len*:*?i1* < *length* (*levels* (*mem-pool-info Va p*))
using *a10 a1 a4 from-l-gt0 inv* **unfolding** *inv-mempool-info-def Let-def*
by *auto*
have *i2-len*:*?i2* < *length* (*levels* (*mem-pool-info Va p*))
using *a10 a1 a4 from-l-gt0 inv* **unfolding** *inv-mempool-info-def Let-def*
by *auto*
have *j1-len*:*?j1* < *length* (*bits* (*levels* (*mem-pool-info Va p*) ! *?i1*))
by (*metis i1-len a9 a12 a1 inv inv-mempool-info-def*)
have *j2-len*:*Suc* (*Suc* (*Suc ?j2*)) < *length* (*bits* (*levels* (*mem-pool-info Va p*) ! *?i2*))
using *i1-len i2-len j1-len inv-mempool from-l-suc*
by *simp*
let *?bts* = *bits* (*levels ?mp* ! *ii*)
let *?btsva* = (*bits* (*levels* (*mem-pool-info Va p*) ! *ii*))
have *a01′*:*ii* < *length* (*levels* (*mem-pool-info Va p*))
using *a01 len-levels* **by** *auto*
then have *inv-bitmap1*:
∀ *j* < *length* (*bits* (*levels* (*mem-pool-info Va p*) ! *ii*)).
(*?btsva* ! *j* = *FREE* ∨ *?btsva* ! *j* = *FREEING* ∨ *?btsva* ! *j* = *ALLOCATED* ∨ *?btsva* ! *j* = *ALLOCATING* ⟶
(*ii* > *0* ⟶ (*bits* (*levels* (*mem-pool-info Va p*) ! (*ii* − *1*))) ! (*j div 4*) = *DIVIDED*)
∧ (*ii* < *length* (*levels* (*mem-pool-info Va p*)) − *1* ⟶ *noexist-bits* (*mem-pool-info Va p*) (*ii*+*1*) (*j*∗*4*)))
∧ (*?btsva* ! *j* = *DIVIDED* ⟶ *ii* > *0* ⟶ (*bits* (*levels* (*mem-pool-info Va*

p) ! (ii − 1))) ! (j div 4) = DIVIDED)
∧ (?btsva ! j = NOEXIST ⟶ ii < length (levels (mem-pool-info Va p)) − 1
⟶ noexist-bits (mem-pool-info Va p) (ii+1) (j∗4))
∧ (?btsva ! j = NOEXIST ∧ ii > 0 ⟶ (bits (levels (mem-pool-info Va p) ! (ii − 1))) ! (j div 4) ≠ DIVIDED)
using *inv mem-pools a1*
unfolding *Let-def inv-bitmap-def*
by *blast*
have *alloc-i1-j1*:*get-bit-s Va p ?i1 ?j1 = ALLOCATING*
using *a7 a0 a12* **unfolding** *inv-aux-vars-def invariant.inv-def*
by (*metis* (*no-types*) *Mem-block.select-convs*(1) *Mem-block.select-convs*(2) *Mem-block.select-convs*(3))

then have *alloc-predi1-j1*:?i1 > 0 ⟶ *get-bit-s Va p* (?i1 − 1) (?j1 div 4) = DIVIDED
using *inv-bitmap1 i1-len j1-len inv a1* **unfolding** *Let-def inv-bitmap-def* **by** *blast*
have *nexisti2*:*noexist-bits* (*mem-pool-info Va p*) ?i2 ?j2
using *a1 conjunct1*[*OF conjunct2*[*OF inv*], *simplified Let-def inv-bitmap-def*] *i1-len j1-len*
alloc-i1-j1 from-l-suc i2-len i1-len j1-len a1
by (*smt One-nat-def Suc-pred add.commute inv-mempool nat-add-left-cancel-less plus-1-eq-Suc*)
have *nexisti3*:?i2 < *length* (*levels* (*mem-pool-info Va p*)) − 1 ⟶
noexist-bits (*mem-pool-info Va p*) ?i2′ ?j20′ ∧
noexist-bits (*mem-pool-info Va p*) ?i2′ ?j21′ ∧
noexist-bits (*mem-pool-info Va p*) ?i2′ ?j22′ ∧
noexist-bits (*mem-pool-info Va p*) ?i2′ ?j23′
proof−
{ assume ?i2 < *length* (*levels* (*mem-pool-info Va p*)) − 1
then have *a00*:∀ j<*length* (*bits* (*levels* (*mem-pool-info Va p*) ! ?i2)).
get-bit-s Va p ?i2 j = NOEXIST ⟶ *noexist-bits* (*mem-pool-info Va p*) ?i2′ (j ∗ 4)
using *a1 conjunct1*[*OF conjunct2*[*OF inv*], *simplified Let-def inv-bitmap-def*] *i2-len*
from-l-suc **by** *auto*
then have *noexist-bits* (*mem-pool-info Va p*) ?i2′ ?j20′ ∧
noexist-bits (*mem-pool-info Va p*) ?i2′ ?j21′ ∧
noexist-bits (*mem-pool-info Va p*) ?i2′ ?j22′ ∧
noexist-bits (*mem-pool-info Va p*) ?i2′ ?j23′
using *j2-len nexisti2 Suc-lessD*
by (*smt One-nat-def add.commute add-2-eq-Suc′ add-Suc-right numeral-3-eq-3 plus-1-eq-Suc*)
}
thus ?*thesis* **by** *fastforce*
qed
let ?bts = *bits* (*levels* ?mp ! *ii*) **and** ?fl = *free-list* (*levels* ?mp ! *ii*)
have *a02′*:*jj* < *length* (*bits* (*levels* (*mem-pool-info Va p*) ! *ii*))
using *a02 a13* **unfolding** *gvars-conf-def gvars-conf-stable-def*

by (*simp add*: *mp-alloc-stm4-inv-bits-len*)
have *eq-len*:*length* (*bits* (*levels* (*mem-pool-info x p*) ! *ii*)) =
length (*bits* (*levels* (*mem-pool-info Va p*) ! *ii*))
using *mp-alloc-stm4-inv-bits-len a14 a15 length-list-update-n*
by *metis*
have *inv-va*:(*?btsva* ! *jj* = *FREE* ∨ *?btsva* ! *jj* = *FREEING* ∨ *?btsva* ! *jj* = *ALLOCATED* ∨ *?btsva* ! *jj* = *ALLOCATING* ⟶
(*ii* > *0* ⟶ (*bits* (*levels* (*mem-pool-info Va p*) ! (*ii* − *1*))) ! (*jj div 4*) = *DIVIDED*)
∧ (*ii* < *length* (*levels* (*mem-pool-info Va p*)) − *1* ⟶ *noexist-bits* (*mem-pool-info Va p*) (*ii*+*1*) (*jj*∗*4*)))
∧ (*?btsva* ! *jj* = *DIVIDED* ⟶ *ii* > *0* ⟶ (*bits* (*levels* (*mem-pool-info Va p*) ! (*ii* − *1*))) ! (*jj div 4*) = *DIVIDED*)
∧ (*?btsva* ! *jj* = *NOEXIST* ⟶ *ii* < *length* (*levels* (*mem-pool-info Va p*)) − *1*
⟶ *noexist-bits* (*mem-pool-info Va p*) (*ii*+*1*) (*jj*∗*4*))
∧ (*?btsva* ! *jj* = *NOEXIST* ∧ *ii* > *0* ⟶ (*bits* (*levels* (*mem-pool-info Va p*) ! (*ii* − *1*))) ! (*jj div 4*) ≠ *DIVIDED*)
using *inv-bitmap1 a02′* **by** *auto*
{ **assume** *a05*:¬((*ii*=*?i1* ∧ *jj*=*?j1*) ∨
(*ii*=*?i2* ∧ *jj*≥ *?j2* ∧ *jj*< *?j2*+*4*) ∨
(*ii*=*?i2′* ∧ *jj*≥ *?j20′* ∧ *jj*< *?j24′*) ∨
(*?i1* >*0* ∧ *ii* = (*?i1* − *1*) ∧ *jj* = *?j1 div 4*))
then have *a050′*:¬(*ii*=*?i1* ∧ *jj*=*?j1*) **and**
a051′: ¬(*ii*=*?i2* ∧ *jj*≥ *?j2* ∧ *jj*<*?j2* + *4*) **and**
a052′:¬(*ii*=*?i2′* ∧ *jj*≥ *?j20′* ∧ *jj*< *?j24′*) **and**
a053′: ¬(*?i1* >*0* ∧ *ii* = (*?i1* − *1*) ∧ *jj* = *?j1 div 4*)
by *force+*
have *eq-get-bit-i-j*:*get-bit-s x p ii jj* = *get-bit-s Va p ii jj*
using *same-bit-mp-alloc-x-va*[*OF a14*[*simplified a18*[*simplified mp-alloc-stm4-froml*[*OF a18*], *THEN sym*]] *a15*, *of ii jj*]
using *a050′ a051′* **by** *auto*
have *eq-get-bit-i1-j1*:*ii*>*0* ⟶ *get-bit-s x p* (*ii*−*1*) (*jj div 4*) = *get-bit-s Va p* (*ii*−*1*) (*jj div 4*)
proof−
{ **assume** *a06*:*ii*>*0*
then have ¬((*ii* − *1*) = *?i1* ∧ *jj div 4* = *?j1*)
using *a050′ a051′ from-l-suc* **by** *fastforce*
moreover have ∀ *j*. *j*≥ *?j2* ∧ *j*≤ *?j2*+*3* ⟶ ¬((*ii* − *1*) = *?i2* ∧ *jj div 4* = *j*)
using *a051′ a052′ from-l-gt0* **by** *fastforce*
ultimately have *get-bit-s x p* (*ii*−*1*) (*jj div 4*) = *get-bit-s Va p* (*ii*−*1*) (*jj div 4*)
using *same-bit-mp-alloc-x-va*[*OF a14*[*simplified a18*[*simplified mp-alloc-stm4-froml*[*OF a18*], *THEN sym*]] *a15*,
of ii −*1 jj div 4*] **by** *auto*
} **thus** *?thesis* **by** *auto* **qed**
have *eq-get-bit-i2-j2*:∀ *j*. *j*≥ (*jj* ∗ *4*) ∧ *j*≤ *Suc*(*Suc* (*Suc* (*jj* ∗ *4*))) ⟶
get-bit-s x p (*ii*+*1*) *j* = *get-bit-s Va p* (*ii*+*1*) *j*

proof−
{ **fix** *j*
assume *a00*:$j \geq (jj * 4) \wedge j \leq (jj * 4)+3$
then have *n1*:$\neg((ii + 1) = ?i1 \wedge j = ?j1)$
using *a053′ from-l-suc* **by** *auto*
have *n2*:$\forall j.\ j \geq ?j2 \wedge j \leq ?j2+3 \longrightarrow \neg((ii + 1) = ?i2 \wedge jj * 4 = j)$
using *a050′ from-l-gt0* **by** *fastforce*
have *get-bit-s x p* $(ii+1)$ *j* = *get-bit-s Va p* $(ii+1)$ *j*
using *same-bit-mp-alloc-x-va*[*OF a14*[*simplified a18*[*simplified mp-alloc-stm4-froml*[*OF a18*], *THEN sym*]] *a15*,
of ii + *1 j*] *n1 n2 a00*
apply (*cases j*=*jj*∗*4*) **by** *auto*
} **thus** *?thesis* **by** *auto*
qed
{ **assume** *a06*: *get-bit-s x p ii jj* = *FREE* ∨
get-bit-s x p ii jj = *FREEING* ∨
get-bit-s x p ii jj = *ALLOCATED* ∨
get-bit-s x p ii jj = *ALLOCATING*
{ **assume** *a07*: *NULL* < *ii*
then have *get-bit-s x p* (*ii* − *1*) (*jj div 4*) = *DIVIDED*
using *a06 a07 eq-get-bit-i1-j1 eq-get-bit-i-j*
using *inv-va* **by** *auto*
}
moreover {
assume *a07*:*ii* < *length* (*levels* (*mem-pool-info x p*)) − *1*
then have *ilen*:*ii* < *length* (*levels* (*mem-pool-info Va p*)) − *1*
by (*simp add*: *len-levels*)
have *get-bit-s Va p ii jj* = *FREE* ∨
get-bit-s Va p ii jj = *FREEING* ∨
get-bit-s Va p ii jj = *ALLOCATED* ∨
get-bit-s Va p ii jj = *ALLOCATING* **using** *eq-get-bit-i-j a06* **by** *auto*

then have *noexist-bits* (*mem-pool-info Va p*) (*ii* + *1*) (*jj* ∗ *4*)
using *ilen inv-va*
by *simp*
then have *noexist-bits* (*mem-pool-info x p*) (*ii* + *1*) (*jj* ∗ *4*)
using *eq-get-bit-i2-j2* **by** (*simp add*: *numeral-3-eq-3*)
}
ultimately have *?thesis* **by** *auto*
} **then have** *?thesis* **by** *auto*
}
moreover {
assume *a06*:$(ii=?i1 \wedge jj=?j1)$
then have *get-bit-s x p ii jj* = *DIVIDED*
using *get-bit-x-l-b a14 a18 from-l from-l-gt0 i1-len j1-len* **by** *presburger*
then have *?thesis* **by** *auto*
}
moreover {
assume *a06*: $(ii=?i2 \wedge jj \geq ?j2 \wedge jj<?j2+4)$

then have *a06'*:*jj*=*?j2* ∨ *jj*=*?j2*+*1* ∨ *jj*=*?j2*+*2* ∨ *jj* =*?j2* + *3* **by** *auto*
{ assume *a07*:*NULL* < *ii*
{ assume *a08*:*jj*=*?j2*
then have *get-bit*:*get-bit-s x p ii jj* = *ALLOCATING*
using *a02 a06 a15 eq-len get-bit-x-l1-b4 i2-len from-l-gt0 i1-len j1-len*
by (*metis mult.commute*)
then have *get-bit-s x p* (*ii*−*1*) (*jj div 4*) = *DIVIDED*
using *a06 a08 get-bit-x-l-b a14 a18 from-l from-l-gt0 i1-len j1-len*
by (*simp add*: *a18 i1-len j1-len from-l-suc*)
}
moreover {
assume *a07*:*jj*≠*?j2*
have *a07'*:*jj div 4* = *?j1* **using** *a06 a07* **by** *auto*
have *get-bit-s x p ii jj* = *FREE*
using *a06 a02 a15 a07 from-l mp-alloc-stm4-inv-bits-len a18 mp-alloc-stm4-pre-precond-f-bn*
by (*auto simp add*: *mp-alloc-stm4-pre-precond-f-bn*)
have *get-bit-s x p* (*ii*−*1*) (*jj div 4*) = *DIVIDED*
using *a06 a07' a14 a18 from-l from-l-gt0 i1-len j1-len*
by (*simp add*: *a18 get-bit-x-l-b i1-len j1-len from-l-suc*)
}
ultimately have *get-bit-s x p* (*ii*−*1*) (*jj div 4*) = *DIVIDED* **by** *fastforce*
}
moreover { assume *a07*:*ii* < *length* (*levels* (*mem-pool-info x p*)) − *1*
then have *get-s*:∀ *j*. *j*≥ (*jj* ∗ *4*) ∧ *j*≤ *Suc*(*Suc* (*Suc* (*jj* ∗ *4*))) ⟶
get-bit-s x p (*ii*+*1*) *j* = *get-bit-s Va p* (*ii*+*1*) *j*
using *same-bit-mp-alloc-x-va*[*OF a14*[*simplified a18*[*simplified mp-alloc-stm4-froml*[*OF a18*],
THEN sym]] *a15*, *of ii* + *1 jj*∗*4*] *a06*
by (*metis Suc-1 Suc-eq-plus1 a14 a18 add.right-neutral add-Suc-right add-left-cancel*
from-l from-l-suc same-bit-mp-alloc-stm4-pre-precond-f1 zero-neq-numeral)
then have *noexist-bits* (*mem-pool-info x p*) (*ii* + *1*) (*jj*∗*4*)
using *a07*[*simplified len-levels*] *a06 inv-va nexisti2*
noexists-eq-bits[*OF get-s*] *a06'*
by *fastforce*
}
ultimately have *?thesis* **by** *fastforce*
}
moreover {
assume *a06*: (*ii*=*?i2'* ∧ *jj*≥ *?j20'* ∧ *jj*< *?j24'*)
then have *a06'*:*jj*=*?j20'* ∨ *jj*=*?j20'*+*1* ∨ *jj*=*?j20'*+*2* ∨ *jj*=*?j20'*+*3* ∨
jj=*?j21'* ∨ *jj*=*?j21'*+*1* ∨ *jj*=*?j21'*+*2* ∨ *jj*=*?j21'*+*3* ∨
jj=*?j22'* ∨ *jj*=*?j22'*+*1* ∨ *jj*=*?j22'*+*2* ∨ *jj*=*?j22'*+*3* ∨
jj=*?j23'* ∨ *jj*=*?j23'*+*1* ∨ *jj*=*?j23'*+*2* ∨ *jj*=*?j23'* + *3*
by *presburger*
then have *eq-get-bit-i-j*:*get-bit-s x p ii jj* = *get-bit-s Va p ii jj*
using *same-bit-mp-alloc-x-va*[*OF a14*[*simplified a18*[*simplified mp-alloc-stm4-froml*[*OF a18*],
THEN sym]] *a15*, *of ii jj*] **using** *a06*

by (*simp add*: *from-l-suc*)
have *i2-lt-length*:*?i2* < *length* (*levels* (*mem-pool-info Va p*)) − *1* **using** *a06* *a01*
by (*simp add*: *len-levels*)
{ assume *a07*: *get-bit-s x p ii jj* = *FREE* ∨
get-bit-s x p ii jj = *FREEING* ∨
get-bit-s x p ii jj = *ALLOCATED* ∨
get-bit-s x p ii jj = *ALLOCATING*
have *get-bit-s Va p ii jj* = *NOEXIST*
using *a07 a06 inv-va nexisti3*[*simplified i2-lt-length*] *a06'*
by *auto*
then have *get-bit-s x p ii jj* = *NOEXIST* **using** *eq-get-bit-i-j* **by** *auto*
} then have *?thesis* **by** *auto*
}
moreover {
assume *a06*: (*?i1* >*0* ∧ *ii* = (*?i1* − *1*) ∧ *jj* = *?j1 div 4*)
then have *eq-get-bit-i-j*:*get-bit-s x p ii jj* = *get-bit-s Va p ii jj*
using *same-bit-mp-alloc-x-va*[*OF a14*[*simplified a18*[*simplified mp-alloc-stm4-froml*[*OF* *a18*],
THEN sym]] *a15*, *of ii jj*]
by *linarith*
then have *get-bit-divided*:*get-bit-s x p ii jj* = *DIVIDED* **using** *a06 alloc-predi1-j1* **by** *simp*
{ assume *a06*: *get-bit-s x p ii jj* = *FREE* ∨ *get-bit-s x p ii jj* = *FREEING* ∨
get-bit-s x p ii jj = *ALLOCATED* ∨ *get-bit-s x p ii jj* = *ALLOCATING*
then have *?thesis* **using** *get-bit-divided* **by** *auto*
} then have *?thesis* **by** *fastforce*
}
ultimately show *?thesis* **by** *fastforce*
qed

lemma *mp-alloc-stm4-inv-bitmap2*:
assumes
a0:*inv Va* **and**
a1:*p* ∈ *mem-pools Va* **and**
a2:∀ *ii*<*length* (*lsizes Va t*). *lsizes Va t* ! *ii* = *ALIGN4* (*max-sz* (*mem-pool-info* *Va p*)) *div 4* ˆ *ii* **and**
a3:*length* (*lsizes Va t*) ≤ *n-levels* (*mem-pool-info Va p*) **and**
a4:*alloc-l Va t* < *int* (*n-levels* (*mem-pool-info Va p*)) **and**
a5:¬ *free-l Va t* < *OK* **and**
a6:*free-l Va t* ≤ *from-l Va t* **and**
a7:*allocating-node Va t* =
Some (|*pool* = *p*, *level* = *nat* (*from-l Va t*),
block = *block-num* (*mem-pool-info Va p*)
(*buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div* *4* ˆ *nat* (*from-l Va t*)))
(*lsizes Va t* ! *nat* (*from-l Va t*)),
data = *buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div* *4* ˆ *nat* (*from-l Va t*))|) **and**

a8:*n* = *block-num* (*mem-pool-info Va p*)
(*buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ˆ *nat* (*from-l Va t*)))
(*lsizes Va t* ! *nat* (*from-l Va t*)) ∨
max-sz (*mem-pool-info Va p*) *div 4* ˆ *nat* (*from-l Va t*) = *NULL* **and**
a9:*block-num* (*mem-pool-info Va p*)
(*buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ˆ *nat* (*from-l Va t*)))
(*lsizes Va t* ! *nat* (*from-l Va t*))
< *n-max* (*mem-pool-info Va p*) ∗ *4* ˆ *nat* (*from-l Va t*) **and**
a10:*from-l Va t* < *alloc-l Va t* **and**
a11:*blk Va t* = *buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ˆ *nat* (*from-l Va t*)) **and**
a12:(*x*, *mp-alloc-stm4-pre-precond-f Va t p*) ∈ *gvars-conf-stable* **and**
a13:∀ *jj*. *jj* ≠ *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*) ⟶
levels (*mem-pool-info x p*) ! *jj* = *levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) ! *jj* **and**
a14:*bits* (*levels* (*mem-pool-info x p*) ! *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*)) =
list-updates-n
(*bits* (*levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*)))
(*Suc* (*bn* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* ∗ *4*)) *3 FREE* **and**
a15:*free-list* (*levels* (*mem-pool-info x p*) ! *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*)) =
inserts
(*map* (λ*ii*. *lsizes* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*) ∗
ii +
blk (*mp-alloc-stm4-pre-precond-f Va t p*) *t*)
[*Suc NULL*..<*4*])
(*free-list*
(*levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*))) **and**
a16:*lsizes x* = *lsizes* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
a17:*from-l x* = *from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
a01:*ii* < *length* (*levels* (*mem-pool-info x p*)) **and**
a02:*jj* < *length* (*bits* (*levels* (*mem-pool-info x p*) ! *ii*)) **and**
a03:*get-bit-s x p ii jj* = *DIVIDED* **and**
a04:*0* < *ii*
shows *get-bit-s x p* (*ii* − *1*) (*jj div 4*) = *DIVIDED*
proof−
let *?mp* = *mem-pool-info x p*
have *inv*:*inv-aux-vars Va* ∧ *inv-bitmap Va* ∧ *inv-mempool-info Va* ∧ *inv-bitmap-freelist Va*
using *a0* **unfolding** *inv-def* **by** *auto*
have *from-l-gt0*:*0* ≤ *from-l Va t* **using** *a6 a5* **by** *linarith*
have *len-levels*:*length* (*levels* (*mem-pool-info x p*)) = *length* (*levels* (*mem-pool-info Va p*))

using *mp-alloc-stm4-lvl-len*[*OF a1 a12*] **by** *simp*
have *maxsz*:*max-sz* (*mem-pool-info x p*) = *max-sz* (*mem-pool-info Va p*)
using *mp-alloc-stm4-maxsz*[*OF a1 a12*] **by** *simp*
have *buf*:*buf* (*mem-pool-info x p*) = *buf* (*mem-pool-info Va p*)
using *mp-alloc-stm4-buf*[*OF a1 a12*] **by** *simp*
have *from-l*:*from-l x* = *from-l Va*
using *mp-alloc-stm4-froml*[*OF a17*] **by** *auto*
have *from-l-suc*:*nat* (*from-l Va t* + *1*) = *nat*(*from-l Va t*) + *1*
using *from-l-gt0* **by** *auto*
have *mem-pools*:*mem-pools x* = *mem-pools Va* **using** *mp-alloc-stm4-mempools*[*OF a12*] **by** *auto*
have *lsizes-x-va*:*lsizes x* = *lsizes Va* **using** *mp-alloc-stm4-pre-precond-f-lsz a16* **by** *auto*
let *?i1*=(*nat* (*from-l Va t*)) **and**
?j1= (*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*))) **and**
?i2 = (*nat* (*from-l Va t* + *1*)) **and**
?j2 = (*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*))∗*4*) **and**
?i1' = (*nat* (*from-l Va t*)) − *1* **and**
?j1' = (*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*))) *div 4* **and**
?i2' = (*nat* (*from-l Va t*)) + *2*
let *?j20'* = *?j2* ∗ *4* **and** *?j21'* = (*?j2*+*1*) ∗ *4* **and** *?j22'* = (*?j2*+*2*)∗*4* **and** *?j23'* = (*?j2*+*3*)∗*4* **and** *?j24'* = (*?j2*+*4*)∗*4*
let *?mp* = *mem-pool-info x p*
have *inv-mempool-info-mp Va p*
using *a1 mem-pools inv* **unfolding** *inv-mempool-info-def* **by** *auto*
note *inv-mempool*=*this*[*simplified Let-def*]
have *i1-len*:*?i1* < *length* (*levels* (*mem-pool-info Va p*))
using *a10 a1 a4 from-l-gt0 inv* **unfolding** *inv-mempool-info-def Let-def*
by *auto*
have *i2-len*:*?i2* < *length* (*levels* (*mem-pool-info Va p*))
using *a10 a1 a4 from-l-gt0 inv* **unfolding** *inv-mempool-info-def Let-def*
by *auto*
have *j1-len*:*?j1* < *length* (*bits* (*levels* (*mem-pool-info Va p*) ! *?i1*))
by (*metis i1-len a9 a11 a1 inv inv-mempool-info-def*)
have *j2-len*:*Suc* (*Suc* (*Suc ?j2*)) < *length* (*bits* (*levels* (*mem-pool-info Va p*) ! *?i2*))
using *i1-len i2-len j1-len inv-mempool from-l-suc*
by *simp*
let *?bts* = *bits* (*levels ?mp* ! *ii*)
let *?btsva* = (*bits* (*levels* (*mem-pool-info Va p*) ! *ii*))
have *a01'*:*ii* < *length* (*levels* (*mem-pool-info Va p*))
using *a01 len-levels* **by** *auto*
then have *inv-bitmap1*:
∀*j* < *length* (*bits* (*levels* (*mem-pool-info Va p*) ! *ii*)).
(*?btsva* ! *j* = *FREE* ∨ *?btsva* ! *j* = *FREEING* ∨ *?btsva* ! *j* = *ALLOCATED* ∨ *?btsva* ! *j* = *ALLOCATING* ⟶

$(ii > 0 \longrightarrow (bits\ (levels\ (mem\text{-}pool\text{-}info\ Va\ p)\ !\ (ii - 1)))\ !\ (j\ div\ 4) = DIVIDED)$
$\wedge\ (ii < length\ (levels\ (mem\text{-}pool\text{-}info\ Va\ p)) - 1 \longrightarrow noexist\text{-}bits\ (mem\text{-}pool\text{-}info\ Va\ p)\ (ii+1)\ (j*4)\ ))$
$\wedge\ (?btsva\ !\ j = DIVIDED \longrightarrow ii > 0 \longrightarrow (bits\ (levels\ (mem\text{-}pool\text{-}info\ Va\ p)\ !\ (ii - 1)))\ !\ (j\ div\ 4) = DIVIDED)$
$\wedge\ (?btsva\ !\ j = NOEXIST \longrightarrow ii < length\ (levels\ (mem\text{-}pool\text{-}info\ Va\ p)) - 1$
$\longrightarrow noexist\text{-}bits\ (mem\text{-}pool\text{-}info\ Va\ p)\ (ii+1)\ (j*4))$
$\wedge\ (?btsva\ !\ j = NOEXIST \wedge ii > 0 \longrightarrow (bits\ (levels\ (mem\text{-}pool\text{-}info\ Va\ p)\ !\ (ii - 1)))\ !\ (j\ div\ 4) \neq DIVIDED)$
using *inv mem-pools a1*
unfolding *Let-def inv-bitmap-def*
by *blast*
have *alloc-i1-j1*:$get\text{-}bit\text{-}s\ Va\ p\ ?i1\ ?j1 = ALLOCATING$
using *a7 a0 a11* **unfolding** *inv-aux-vars-def invariant.inv-def*
by (*metis* (*no-types*) *Mem-block.select-convs*(*1*) *Mem-block.select-convs*(*2*) *Mem-block.select-convs*(*3*))

then have *alloc-predi1-j1*:$?i1 > 0 \longrightarrow get\text{-}bit\text{-}s\ Va\ p\ (?i1 - 1)\ (?j1\ div\ 4) = DIVIDED$
using *inv-bitmap1 i1-len j1-len inv a1* **unfolding** *Let-def inv-bitmap-def* **by** *blast*
have *nexisti2*:$noexist\text{-}bits\ (mem\text{-}pool\text{-}info\ Va\ p)\ ?i2\ ?j2$
using *a1 conjunct1*[*OF conjunct2*[*OF inv*], *simplified Let-def inv-bitmap-def*] *i1-len j1-len*
alloc-i1-j1 from-l-suc i2-len i1-len j1-len a1
by (*smt One-nat-def Suc-pred add.commute inv-mempool nat-add-left-cancel-less plus-1-eq-Suc*)
have *nexisti3*:$?i2 < length\ (levels\ (mem\text{-}pool\text{-}info\ Va\ p)) - 1 \longrightarrow$
$noexist\text{-}bits\ (mem\text{-}pool\text{-}info\ Va\ p)\ ?i2'\ ?j20' \wedge$
$noexist\text{-}bits\ (mem\text{-}pool\text{-}info\ Va\ p)\ ?i2'\ ?j21' \wedge$
$noexist\text{-}bits\ (mem\text{-}pool\text{-}info\ Va\ p)\ ?i2'\ ?j22' \wedge$
$noexist\text{-}bits\ (mem\text{-}pool\text{-}info\ Va\ p)\ ?i2'\ ?j23'$
proof−
{ **assume** $?i2 < length\ (levels\ (mem\text{-}pool\text{-}info\ Va\ p)) - 1$
then have *a00*:$\forall j<length\ (bits\ (levels\ (mem\text{-}pool\text{-}info\ Va\ p)\ !\ ?i2)).$
$get\text{-}bit\text{-}s\ Va\ p\ ?i2\ j = NOEXIST \longrightarrow noexist\text{-}bits\ (mem\text{-}pool\text{-}info\ Va\ p)\ ?i2'\ (j * 4)$
using *a1 conjunct1*[*OF conjunct2*[*OF inv*], *simplified Let-def inv-bitmap-def*] *i2-len*
from-l-suc **by** *auto*
then have $noexist\text{-}bits\ (mem\text{-}pool\text{-}info\ Va\ p)\ ?i2'\ ?j20' \wedge$
$noexist\text{-}bits\ (mem\text{-}pool\text{-}info\ Va\ p)\ ?i2'\ ?j21' \wedge$
$noexist\text{-}bits\ (mem\text{-}pool\text{-}info\ Va\ p)\ ?i2'\ ?j22' \wedge$
$noexist\text{-}bits\ (mem\text{-}pool\text{-}info\ Va\ p)\ ?i2'\ ?j23'$
using *j2-len nexisti2 Suc-lessD*
by (*smt One-nat-def add.commute add-2-eq-Suc′ add-Suc-right numeral-3-eq-3 plus-1-eq-Suc*)
}

thus *?thesis* **by** *fastforce*
qed
let *?bts* = *bits* (*levels* *?mp* ! *ii*) **and** *?fl* = *free-list* (*levels* *?mp* ! *ii*)
have *a02'*:*jj* < *length* (*bits* (*levels* (*mem-pool-info* *Va* *p*) ! *ii*))
using *a02* *a12* **unfolding** *gvars-conf-def* *gvars-conf-stable-def*
by (*simp* *add*: *mp-alloc-stm4-inv-bits-len*)
have *eq-len*:*length* (*bits* (*levels* (*mem-pool-info* *x* *p*) ! *ii*)) =
length (*bits* (*levels* (*mem-pool-info* *Va* *p*) ! *ii*))
using *mp-alloc-stm4-inv-bits-len* *a13* *a14* *length-list-update-n*
by *metis*
have *inv-va*:(*?btsva* ! *jj* = *FREE* ∨ *?btsva* ! *jj* = *FREEING* ∨ *?btsva* ! *jj* = *ALLOCATED* ∨ *?btsva* ! *jj* = *ALLOCATING* ⟶
(*ii* > *0* ⟶ (*bits* (*levels* (*mem-pool-info* *Va* *p*) ! (*ii* − *1*))) ! (*jj* *div* *4*) = *DIVIDED*)
∧ (*ii* < *length* (*levels* (*mem-pool-info* *Va* *p*)) − *1* ⟶ *noexist-bits* (*mem-pool-info* *Va* *p*) (*ii*+*1*) (*jj*∗*4*)))
∧ (*?btsva* ! *jj* = *DIVIDED* ⟶ *ii* > *0* ⟶ (*bits* (*levels* (*mem-pool-info* *Va* *p*) ! (*ii* − *1*))) ! (*jj* *div* *4*) = *DIVIDED*)
∧ (*?btsva* ! *jj* = *NOEXIST* ⟶ *ii* < *length* (*levels* (*mem-pool-info* *Va* *p*)) − *1*
⟶ *noexist-bits* (*mem-pool-info* *Va* *p*) (*ii*+*1*) (*jj*∗*4*))
∧ (*?btsva* ! *jj* = *NOEXIST* ∧ *ii* > *0* ⟶ (*bits* (*levels* (*mem-pool-info* *Va* *p*) ! (*ii* − *1*))) ! (*jj* *div* *4*) ≠ *DIVIDED*)
using *inv-bitmap1* *a02'* **by** *auto*
{ assume *a05*:¬((*ii*=*?i1* ∧ *jj*=*?j1*) ∨
(*ii*=*?i2* ∧ *jj*≥ *?j2* ∧ *jj*< *?j2*+*4*) ∨
(*ii*=*?i2'* ∧ *jj*≥ *?j20'* ∧ *jj*< *?j24'*) ∨
(*?i1* >*0* ∧ *ii* = (*?i1* − *1*) ∧ *jj* = *?j1* *div* *4*))
then have *a050'*:¬(*ii*=*?i1* ∧ *jj*=*?j1*) **and**
a051': ¬(*ii*=*?i2* ∧ *jj*≥ *?j2* ∧ *jj*<*?j2* + *4*) **and**
a052':¬(*ii*=*?i2'* ∧ *jj*≥ *?j20'* ∧ *jj*< *?j24'*) **and**
a053': ¬(*?i1* >*0* ∧ *ii* = (*?i1* − *1*) ∧ *jj* = *?j1* *div* *4*)
by *force*+
have *eq-get-bit-i-j*:*get-bit-s* *x* *p* *ii* *jj* = *get-bit-s* *Va* *p* *ii* *jj*
using *same-bit-mp-alloc-x-va*[*OF* *a13*[*simplified* *a17*[*simplified* *mp-alloc-stm4-froml*[*OF* *a17*], *THEN* *sym*]] *a14*, *of* *ii* *jj*]
using *a050'* *a051'* **by** *auto*
have *eq-get-bit-i1-j1*:*ii*>*0* ⟶ *get-bit-s* *x* *p* (*ii*−*1*) (*jj* *div* *4*) = *get-bit-s* *Va* *p* (*ii*−*1*) (*jj* *div* *4*)
proof−
{ assume *a06*:*ii*>*0*
then have ¬((*ii* − *1*) = *?i1* ∧ *jj* *div* *4* = *?j1*)
using *a050'* *a051'* *from-l-suc* **by** *fastforce*
moreover have ∀ *j*. *j*≥ *?j2* ∧ *j*≤ *?j2*+*3* ⟶ ¬((*ii* − *1*) = *?i2* ∧ *jj* *div* *4* = *j*)
using *a051'* *a052'* *from-l-gt0* **by** *fastforce*
ultimately have *get-bit-s* *x* *p* (*ii*−*1*) (*jj* *div* *4*) = *get-bit-s* *Va* *p* (*ii*−*1*) (*jj* *div* *4*)
using *same-bit-mp-alloc-x-va*[*OF* *a13*[*simplified* *a17*[*simplified* *mp-alloc-stm4-froml*[*OF*

a17], *THEN sym*]] *a14*,
of ii −1 jj div 4] **by** *auto*
} **thus** *?thesis* **by** *auto* **qed**
have *eq-get-bit-i2-j2*:$\forall$ *j*. *j*$\geq$ (*jj* $*$ *4*) $\wedge$ *j*$\leq$ *Suc*(*Suc* (*Suc* (*jj* $*$ *4*))) $\longrightarrow$
get-bit-s x p (*ii*+*1*) *j* = *get-bit-s Va p* (*ii*+*1*) *j*
proof−
{ **fix** *j*
assume *a00*:*j*$\geq$ (*jj* $*$ *4*) $\wedge$ *j*$\leq$ (*jj* $*$ *4*)+*3*
then have *n1*:$\neg$((*ii* + *1*) = *?i1* $\wedge$ *j* = *?j1*)
using *a053′ from-l-suc* **by** *auto*
have *n2*:$\forall$ *j*. *j*$\geq$ *?j2* $\wedge$ *j*$\leq$ *?j2*+*3* $\longrightarrow$ $\neg$((*ii* + *1*) = *?i2* $\wedge$ *jj* $*$ *4* = *j*)
using *a050′ from-l-gt0* **by** *fastforce*
have *get-bit-s x p* (*ii*+*1*) *j* = *get-bit-s Va p* (*ii*+*1*) *j*
using *same-bit-mp-alloc-x-va*[*OF a13*[*simplified a17*[*simplified mp-alloc-stm4-froml*[*OF a17*], *THEN sym*]] *a14*,
of ii + *1 j*] *n1 n2 a00*
apply (*cases j*=*jj*$*$*4*) **by** *auto*
} **thus** *?thesis* **by** *auto*
qed
then have *?thesis*
using *a03 a04 eq-get-bit-i1-j1 eq-get-bit-i-j inv-va* **by** *auto*
}
moreover **{**
assume *a06*:(*ii*=*?i1* $\wedge$ *jj*=*?j1*)
then have *?thesis* **using** *a03 a04*
by (*metis Suc-eq-plus1 Suc-pred a13 a17*
add.commute add-2-eq-Suc′ add-cancel-right-right
alloc-predi1-j1 from-l from-l-suc
same-bit-mp-alloc-stm4-pre-precond-f zero-neq-numeral)
}
moreover **{**
assume *a06*: (*ii*=*?i2* $\wedge$ *jj*$\geq$ *?j2* $\wedge$ *jj*<*?j2*+*4*)
then have *a06′*:*jj*=*?j2* $\vee$ *jj*=*?j2*+*1* $\vee$ *jj*=*?j2*+*2* $\vee$ *jj* =*?j2* + *3* **by** *auto*
have *l1*:(*ii* − *1*) = *?i1* $\wedge$ (*jj div 4*) = *?j1*
using *a2 a03 a04 a01 a02 a02′ a06 a13 a14 a06′*
from-l-gt0 from-l-suc
by (*metis add.commute add-mult-distrib2*
diff-add-inverse2 div-nat-eqI mult.commute
nat-mult-1-right plus-1-eq-Suc)
then have *?thesis* **using** *get-bit-x-l-b*[*OF l1*] *a13*
from-l from-l-gt0 from-l-suc i1-len
by (*simp add*: *a17 j1-len l1*)
}
moreover **{**
assume *a06*: (*ii*=*?i2′* $\wedge$ *jj*$\geq$ *?j20′* $\wedge$ *jj*< *?j24′*)
then have *a06′*:*jj*=*?j20′* $\vee$ *jj*=*?j20′*+*1* $\vee$ *jj*=*?j20′*+*2* $\vee$ *jj*=*?j20′*+*3* $\vee$
jj=*?j21′* $\vee$ *jj*=*?j21′*+*1* $\vee$ *jj*=*?j21′*+*2* $\vee$ *jj*=*?j21′*+*3* $\vee$
jj=*?j22′* $\vee$ *jj*=*?j22′*+*1* $\vee$ *jj*=*?j22′*+*2* $\vee$ *jj*=*?j22′*+*3* $\vee$
jj=*?j23′* $\vee$ *jj*=*?j23′*+*1* $\vee$ *jj*=*?j23′*+*2* $\vee$ *jj*=*?j23′* + *3*

by *presburger*
then have *eq-get-bit-i-j*:*get-bit-s x p ii jj* = *get-bit-s Va p ii jj*
using *same-bit-mp-alloc-x-va*[*OF a13*[*simplified a17*[*simplified mp-alloc-stm4-froml*[*OF a17*],
THEN sym]] *a14*, *of ii jj*] **using** *a06*
by (*simp add*: *from-l-suc*)
moreover have *i2-lt-length*:*?i2* < *length* (*levels* (*mem-pool-info Va p*)) − *1*
using
a06[*simplified len-levels*] *a01*[*simplified len-levels*]
by *simp*
then have *get-bit-s Va p ii jj* = *NOEXIST*
using *a06 a01 a06 inv-va nexisti3 a06′ a06*[*simplified len-levels*] *a01*[*simplified len-levels*]
by *auto*
ultimately have *?thesis*
using *a03* **by** *auto*
}
moreover {
assume *a06*: (*?i1* >*0* ∧ *ii* = (*?i1* − *1*) ∧ *jj* = *?j1 div 4*)
then have *eq-get-bit-i-j*:*get-bit-s x p ii jj* = *get-bit-s Va p ii jj*
using *same-bit-mp-alloc-x-va*[*OF a13*[*simplified a17*[*simplified mp-alloc-stm4-froml*[*OF a17*],
THEN sym]] *a14*, *of ii jj*]
by *linarith*
have *?thesis* **using** *a03 a04 a13 a17 from-l inv-va same-bit-mp-alloc-stm4-pre-precond-f1*
calculation(*1*) *calculation*(*2*) *calculation*(*3*) *calculation*(*4*)
by (*smt Suc-pred add-diff-cancel-left′ int-nat-eq inv-va of-nat-Suc plus-1-eq-Suc*)

}
ultimately show *?thesis* **by** *fastforce*
qed

lemma *mp-alloc-stm4-inv-bitmap3*:
assumes
a0:*inv Va* **and**
a1:*p* ∈ *mem-pools Va* **and**
a2:∀ *ii*<*length* (*lsizes Va t*). *lsizes Va t* ! *ii* = *ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *ii* **and**
a4:*alloc-l Va t* < *int* (*n-levels* (*mem-pool-info Va p*)) **and**
a5:¬ *free-l Va t* < *OK* **and**
a6:*free-l Va t* ≤ *from-l Va t* **and**
a7:*allocating-node Va t* =
Some (|*pool* = *p*, *level* = *nat* (*from-l Va t*),
block = *block-num* (*mem-pool-info Va p*)
(*buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ˆ *nat* (*from-l Va t*)))
(*lsizes Va t* ! *nat* (*from-l Va t*)),
data = *buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ˆ *nat* (*from-l Va t*))|) **and**

a8:*n* = *block-num* (*mem-pool-info* *Va* *p*)
(*buf* (*mem-pool-info* *Va* *p*) + *n* $*$ (*max-sz* (*mem-pool-info* *Va* *p*) *div* *4* ^ *nat* (*from-l* *Va* *t*)))
(*lsizes* *Va* *t* ! *nat* (*from-l* *Va* *t*)) $\vee$
max-sz (*mem-pool-info* *Va* *p*) *div* *4* ^ *nat* (*from-l* *Va* *t*) = *NULL* **and**
a9:*block-num* (*mem-pool-info* *Va* *p*)
(*buf* (*mem-pool-info* *Va* *p*) + *n* $*$ (*max-sz* (*mem-pool-info* *Va* *p*) *div* *4* ^ *nat* (*from-l* *Va* *t*)))
(*lsizes* *Va* *t* ! *nat* (*from-l* *Va* *t*))
< *n-max* (*mem-pool-info* *Va* *p*) $*$ *4* ^ *nat* (*from-l* *Va* *t*) **and**
a10:*from-l* *Va* *t* < *alloc-l* *Va* *t* **and**

a11:*n* < *n-max* (*mem-pool-info* *Va* *p*) $*$ *4* ^ *nat* (*from-l* *Va* *t*) **and**
a12:*blk* *Va* *t* = *buf* (*mem-pool-info* *Va* *p*) + *n* $*$ (*max-sz* (*mem-pool-info* *Va* *p*) *div* *4* ^ *nat* (*from-l* *Va* *t*)) **and**
a13:(*x*, *mp-alloc-stm4-pre-precond-f* *Va* *t* *p*) $\in$ *gvars-conf-stable* **and**
a14:$\forall$ *jj*. *jj* $\neq$ *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f* *Va* *t* *p*) *t* + *1*) $\longrightarrow$
levels (*mem-pool-info* *x* *p*) ! *jj* = *levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f* *Va* *t* *p*) *p*) ! *jj* **and**
a15:*bits* (*levels* (*mem-pool-info* *x* *p*) ! *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f* *Va* *t* *p*) *t* + *1*)) =
list-updates-n
(*bits* (*levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f* *Va* *t* *p*) *p*) !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f* *Va* *t* *p*) *t* + *1*)))
(*Suc* (*bn* (*mp-alloc-stm4-pre-precond-f* *Va* *t* *p*) *t* $*$ *4*)) *3* *FREE* **and**
a16:*free-list* (*levels* (*mem-pool-info* *x* *p*) ! *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f* *Va* *t* *p*) *t* + *1*)) =
inserts
(*map* (λ*ii*. *lsizes* (*mp-alloc-stm4-pre-precond-f* *Va* *t* *p*) *t* !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f* *Va* *t* *p*) *t* + *1*) $*$
ii +
blk (*mp-alloc-stm4-pre-precond-f* *Va* *t* *p*) *t*)
[*Suc* *NULL*..<*4*])
(*free-list*
(*levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f* *Va* *t* *p*) *p*) !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f* *Va* *t* *p*) *t* + *1*))) **and**
a17:*lsizes* *x* = *lsizes* (*mp-alloc-stm4-pre-precond-f* *Va* *t* *p*) **and**
a18:*from-l* *x* = *from-l* (*mp-alloc-stm4-pre-precond-f* *Va* *t* *p*) **and**
a01:*ii* < *length* (*levels* (*mem-pool-info* *x* *p*)) **and**
a02:*jj* < *length* (*bits* (*levels* (*mem-pool-info* *x* *p*) ! *ii*)) **and**
a03:*get-bit-s* *x* *p* *ii* *jj* = *NOEXIST* **and**
a04:*ii* < *length* (*levels* (*mem-pool-info* *x* *p*)) − *1*
shows *noexist-bits* (*mem-pool-info* *x* *p*) (*ii* + *1*) (*jj* $*$ *4*)
proof−
let *?mp* = *mem-pool-info* *x* *p*
have *inv*:*inv-aux-vars* *Va* $\wedge$ *inv-bitmap* *Va* $\wedge$ *inv-mempool-info* *Va* $\wedge$ *inv-bitmap-freelist* *Va*
using *a0* **unfolding** *inv-def* **by** *auto*
have *from-l-gt0*:*0* $\leq$ *from-l* *Va* *t* **using** *a6* *a5* **by** *linarith*

have *len-levels*:*length* (*levels* (*mem-pool-info x p*)) = *length* (*levels* (*mem-pool-info Va p*))
using *mp-alloc-stm4-lvl-len*[*OF a1 a13*] **by** *simp*
have *maxsz*:*max-sz* (*mem-pool-info x p*) = *max-sz* (*mem-pool-info Va p*)
using *mp-alloc-stm4-maxsz*[*OF a1 a13*] **by** *simp*
have *buf*:*buf* (*mem-pool-info x p*) = *buf* (*mem-pool-info Va p*)
using *mp-alloc-stm4-buf*[*OF a1 a13*] **by** *simp*
have *from-l*:*from-l x* = *from-l Va*
using *mp-alloc-stm4-froml*[*OF a18*] **by** *auto*
have *from-l-suc*:*nat* (*from-l Va t* + *1*) = *nat*(*from-l Va t*) + *1*
using *from-l-gt0* **by** *auto*
have *mem-pools*:*mem-pools x* = *mem-pools Va* **using** *mp-alloc-stm4-mempools*[*OF a13*] **by** *auto*
have *lsizes-x-va*:*lsizes x* = *lsizes Va* **using** *mp-alloc-stm4-pre-precond-f-lsz a17*
by *auto*
let *?i1*=(*nat* (*from-l Va t*)) **and**
?j1= (*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*))) **and**
?i2 = (*nat* (*from-l Va t* + *1*)) **and**
?j2 = (*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*))**4*) **and**
?i1' = (*nat* (*from-l Va t*)) − *1* **and**
?j1' = (*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*))) *div 4* **and**
?i2' = (*nat* (*from-l Va t*)) + *2*
let *?j20'* = *?j2* * *4* **and** *?j21'* = (*?j2*+*1*) * *4* **and** *?j22'* = (*?j2*+*2*)**4* **and**
?j23' = (*?j2*+*3*)**4* **and** *?j24'* = (*?j2*+*4*)**4*
let *?mp* = *mem-pool-info x p*
have *inv-mempool-info-mp Va p*
using *a1 mem-pools inv* **unfolding** *inv-mempool-info-def* **by** *auto*
note *inv-mempool*=*this*[*simplified Let-def*]
have *i1-len*:*?i1* < *length* (*levels* (*mem-pool-info Va p*))
using *a10 a1 a4 from-l-gt0 inv* **unfolding** *inv-mempool-info-def Let-def*
by *auto*
have *i2-len*:*?i2* < *length* (*levels* (*mem-pool-info Va p*))
using *a10 a1 a4 from-l-gt0 inv* **unfolding** *inv-mempool-info-def Let-def*
by *auto*
have *j1-len*:*?j1* < *length* (*bits* (*levels* (*mem-pool-info Va p*) ! *?i1*))
by (*metis i1-len a9 a12 a1 inv inv-mempool-info-def*)
have *j2-len*:*Suc* (*Suc* (*Suc ?j2*)) < *length* (*bits* (*levels* (*mem-pool-info Va p*) ! *?i2*))
using *i1-len i2-len j1-len inv-mempool from-l-suc*
by *simp*
let *?bts* = *bits* (*levels ?mp* ! *ii*)
let *?btsva* = (*bits* (*levels* (*mem-pool-info Va p*) ! *ii*))
have *a01'*:*ii* < *length* (*levels* (*mem-pool-info Va p*))
using *a01 len-levels* **by** *auto*
then have *inv-bitmap1*:
∀ *j* < *length* (*bits* (*levels* (*mem-pool-info Va p*) ! *ii*)).

($?btsva ! j = FREE \vee ?btsva ! j = FREEING \vee ?btsva ! j = ALLOCATED$
$\vee$ $?btsva ! j = ALLOCATING \longrightarrow$
$(ii > 0 \longrightarrow (bits (levels (mem\text{-}pool\text{-}info\ Va\ p) ! (ii - 1))) ! (j\ div$
$4) = DIVIDED)$
$\wedge (ii < length (levels (mem\text{-}pool\text{-}info\ Va\ p)) - 1 \longrightarrow noexist\text{-}bits$
$(mem\text{-}pool\text{-}info\ Va\ p) (ii+1) (j*4)))$
$\wedge (?btsva ! j = DIVIDED \longrightarrow ii > 0 \longrightarrow (bits (levels (mem\text{-}pool\text{-}info\ Va$
$p) ! (ii - 1))) ! (j\ div\ 4) = DIVIDED)$
$\wedge (?btsva ! j = NOEXIST \longrightarrow ii < length (levels (mem\text{-}pool\text{-}info\ Va\ p))$
$- 1$
$\longrightarrow noexist\text{-}bits (mem\text{-}pool\text{-}info\ Va\ p) (ii+1) (j*4))$
$\wedge (?btsva ! j = NOEXIST \wedge ii > 0 \longrightarrow (bits (levels (mem\text{-}pool\text{-}info\ Va$
$p) ! (ii - 1))) ! (j\ div\ 4) \neq DIVIDED)$
using *inv mem-pools a1*
unfolding *Let-def inv-bitmap-def*
by *blast*
have *alloc-i1-j1*:*get-bit-s Va p ?i1 ?j1* = *ALLOCATING*
using *a7 a0 a12* **unfolding** *inv-aux-vars-def invariant.inv-def*
by (*metis* (*no-types*) *Mem-block.select-convs*(*1*) *Mem-block.select-convs*(*2*) *Mem-block.select-convs*(*3*))

then have *alloc-predi1-j1*:$?i1 > 0 \longrightarrow get\text{-}bit\text{-}s\ Va\ p\ (?i1 - 1)\ (?j1\ div\ 4) =$
$DIVIDED$
using *inv-bitmap1 i1-len j1-len inv a1* **unfolding** *Let-def inv-bitmap-def* **by**
blast
have *nexisti2*:*noexist-bits* (*mem-pool-info Va p*) *?i2 ?j2*
using *a1 conjunct1*[*OF conjunct2*[*OF inv*], *simplified Let-def inv-bitmap-def*]
i1-len j1-len
alloc-i1-j1 from-l-suc i2-len i1-len j1-len a1
by (*smt One-nat-def Suc-pred add.commute inv-mempool nat-add-left-cancel-less*
plus-1-eq-Suc)
have *nexisti3*:$?i2 < length (levels (mem\text{-}pool\text{-}info\ Va\ p)) - 1 \longrightarrow$
$noexist\text{-}bits (mem\text{-}pool\text{-}info\ Va\ p)\ ?i2'\ ?j20' \wedge$
$noexist\text{-}bits (mem\text{-}pool\text{-}info\ Va\ p)\ ?i2'\ ?j21' \wedge$
$noexist\text{-}bits (mem\text{-}pool\text{-}info\ Va\ p)\ ?i2'\ ?j22' \wedge$
$noexist\text{-}bits (mem\text{-}pool\text{-}info\ Va\ p)\ ?i2'\ ?j23'$
proof−
{ assume $?i2 < length (levels (mem\text{-}pool\text{-}info\ Va\ p)) - 1$
then have *a00*:$\forall j<length (bits (levels (mem\text{-}pool\text{-}info\ Va\ p) ! ?i2)).$
$get\text{-}bit\text{-}s\ Va\ p\ ?i2\ j = NOEXIST \longrightarrow noexist\text{-}bits (mem\text{-}pool\text{-}info\ Va$
$p)\ ?i2' (j * 4)$
using *a1 conjunct1*[*OF conjunct2*[*OF inv*], *simplified Let-def inv-bitmap-def*]
i2-len
from-l-suc **by** *auto*
then have $noexist\text{-}bits (mem\text{-}pool\text{-}info\ Va\ p)\ ?i2'\ ?j20' \wedge$
$noexist\text{-}bits (mem\text{-}pool\text{-}info\ Va\ p)\ ?i2'\ ?j21' \wedge$
$noexist\text{-}bits (mem\text{-}pool\text{-}info\ Va\ p)\ ?i2'\ ?j22' \wedge$
$noexist\text{-}bits (mem\text{-}pool\text{-}info\ Va\ p)\ ?i2'\ ?j23'$
using *j2-len nexisti2 Suc-lessD*
by (*smt One-nat-def add.commute add-2-eq-Suc' add-Suc-right numeral-3-eq-3*

plus-1-eq-Suc)
  }
  thus *?thesis* **by** *fastforce*
 qed
 let *?bts* = *bits* (*levels* *?mp* ! *ii*) **and** *?fl* = *free-list* (*levels* *?mp* ! *ii*)
 have *a02′*:*jj* < *length* (*bits* (*levels* (*mem-pool-info* *Va* *p*) ! *ii*))
  using *a02* *a13* **unfolding** *gvars-conf-def* *gvars-conf-stable-def*
  by (*simp* *add*: *mp-alloc-stm4-inv-bits-len*)
 have *eq-len*:*length* (*bits* (*levels* (*mem-pool-info* *x* *p*) ! *ii*)) =
   length (*bits* (*levels* (*mem-pool-info* *Va* *p*) ! *ii*))
  using *mp-alloc-stm4-inv-bits-len* *a14* *a15* *length-list-update-n*
  by *metis*
 have *inv-va*:(*?btsva* ! *jj* = *FREE* ∨ *?btsva* ! *jj* = *FREEING* ∨ *?btsva* ! *jj* = *ALLOCATED* ∨ *?btsva* ! *jj* = *ALLOCATING* ⟶
   (*ii* > *0* ⟶ (*bits* (*levels* (*mem-pool-info* *Va* *p*) ! (*ii* − *1*))) ! (*jj* *div* *4*) = *DIVIDED*)
   ∧ (*ii* < *length* (*levels* (*mem-pool-info* *Va* *p*)) − *1* ⟶ *noexist-bits* (*mem-pool-info* *Va* *p*) (*ii*+*1*) (*jj*∗*4*)))
  ∧ (*?btsva* ! *jj* = *DIVIDED* ⟶ *ii* > *0* ⟶ (*bits* (*levels* (*mem-pool-info* *Va* *p*) ! (*ii* − *1*))) ! (*jj* *div* *4*) = *DIVIDED*)
  ∧ (*?btsva* ! *jj* = *NOEXIST* ⟶ *ii* < *length* (*levels* (*mem-pool-info* *Va* *p*)) − *1*
   ⟶ *noexist-bits* (*mem-pool-info* *Va* *p*) (*ii*+*1*) (*jj*∗*4*))
  ∧ (*?btsva* ! *jj* = *NOEXIST* ∧ *ii* > *0* ⟶ (*bits* (*levels* (*mem-pool-info* *Va* *p*) ! (*ii* − *1*))) ! (*jj* *div* *4*) ≠ *DIVIDED*)
  using *inv-bitmap1* *a02′* **by** *auto*
 { assume *a05*:¬((*ii*=*?i1* ∧ *jj*=*?j1*) ∨
   (*ii*=*?i2* ∧ *jj*≥ *?j2* ∧ *jj*< *?j2*+*4*) ∨
   (*?i1* >*0* ∧ *ii* = (*?i1* − *1*) ∧ *jj* = *?j1* *div* *4*))
  then have *a050′*:¬(*ii*=*?i1* ∧ *jj*=*?j1*) **and**
   a051′: ¬(*ii*=*?i2* ∧ *jj*≥ *?j2* ∧ *jj*<*?j2* + *4*) **and**
   a053′: ¬(*?i1* >*0* ∧ *ii* = (*?i1* − *1*) ∧ *jj* = *?j1* *div* *4*)
   by *force*+
  have *eq-get-bit-i-j*:*get-bit-s* *x* *p* *ii* *jj* = *get-bit-s* *Va* *p* *ii* *jj*
  using *same-bit-mp-alloc-x-va*[*OF* *a14*[*simplified* *a18*[*simplified* *mp-alloc-stm4-froml*[*OF* *a18*], *THEN* *sym*]] *a15*, *of* *ii* *jj*]
   using *a050′* *a051′* **by** *auto*
  have *eq-get-bit-i2-j2*:∀*j*. *j*≥ (*jj* ∗ *4*) ∧ *j*≤ *Suc*(*Suc* (*Suc* (*jj* ∗ *4*))) ⟶
   get-bit-s *x* *p* (*ii*+*1*) *j* = *get-bit-s* *Va* *p* (*ii*+*1*) *j*
  proof−
   { fix *j*
    assume *a00*:*j*≥ (*jj* ∗ *4*) ∧ *j*≤ (*jj* ∗ *4*)+*3*
    then have *n1*:¬((*ii* + *1*) = *?i1* ∧ *j* = *?j1*)
     using *a053′* *from-l-suc* **by** *auto*
    have *n2*:∀*j*. *j*≥ *?j2* ∧ *j*≤ *?j2*+*3* ⟶ ¬((*ii* + *1*) = *?i2* ∧ *jj* ∗ *4* = *j*)
    using *a050′* *from-l-gt0* **by** *fastforce*
    have *get-bit-s* *x* *p* (*ii*+*1*) *j* = *get-bit-s* *Va* *p* (*ii*+*1*) *j*
   using *same-bit-mp-alloc-x-va*[*OF* *a14*[*simplified* *a18*[*simplified* *mp-alloc-stm4-froml*[*OF* *a18*], *THEN* *sym*]] *a15*,

of ii + 1 j] *n1 n2 a00*
apply (*cases j=jj*4*) **by** *auto*
} **thus** *?thesis* **by** *auto*
qed
then have *?thesis*
using *a04 len-levels eq-get-bit-i-j a03 inv-va* **by** (*simp add*: *numeral-3-eq-3*)
}
moreover {
assume *a06*:(*ii=?i1* ∧ *jj=?j1*)
then have *get-bit-s x p ii jj = DIVIDED*
using *get-bit-x-l-b a14 a18 from-l from-l-gt0 i1-len j1-len* **by** *presburger*
then have *?thesis* **using** *a03* **by** *auto*
}
moreover {
assume *a06*: (*ii=?i2* ∧ *jj*≥ *?j2* ∧ *jj*<*?j2+4*)
then have *a06'*:*jj=?j2* ∨ *jj=?j2+1* ∨ *jj=?j2+2* ∨ *jj =?j2 + 3* **by** *auto*
then have *get-s*:∀ *j*. *j*≥ (*jj* * *4*) ∧ *j*≤ *Suc*(*Suc* (*Suc* (*jj* * *4*))) ⟶
get-bit-s x p (*ii+1*) *j* = *get-bit-s Va p* (*ii+1*) *j*
using *same-bit-mp-alloc-x-va*[*OF a14*[*simplified a18*[*simplified mp-alloc-stm4-froml*[*OF a18*],
THEN sym]] *a15*, *of ii + 1 jj*4*] *a06*
by (*metis Suc-1 Suc-eq-plus1 a14 a18 add.right-neutral add-Suc-right add-left-cancel*

from-l from-l-suc same-bit-mp-alloc-stm4-pre-precond-f1 zero-neq-numeral)
then have *?thesis*
using *a04*[*simplified len-levels*] *a06 inv-va nexisti2*
noexists-eq-bits[*OF get-s*] *a06'*
by *fastforce*
}
moreover {
assume *a06*: (*?i1* >*0* ∧ *ii* = (*?i1* − *1*) ∧ *jj* = *?j1 div 4*)
then have *eq-get-bit-i-j*:*get-bit-s x p ii jj* = *get-bit-s Va p ii jj*
using *same-bit-mp-alloc-x-va*[*OF a14*[*simplified a18*[*simplified mp-alloc-stm4-froml*[*OF a18*],
THEN sym]] *a15*, *of ii jj*]
by *linarith*
then have *get-bit-divided*:*get-bit-s x p ii jj = DIVIDED* **using** *a06 alloc-predi1-j1*
by *simp*
then have *?thesis* **using** *get-bit-divided a03* **by** *auto*
}
ultimately show *?thesis* **by** *fastforce*
qed

lemma *mp-alloc-stm4-inv-bitmap4*:
assumes
a0:*inv Va* **and**
a1:*p* ∈ *mem-pools Va* **and**
a2:∀ *ii*<*length* (*lsizes Va t*). *lsizes Va t* ! *ii* = *ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *ii* **and**

a3:*length* (*lsizes Va t*) ≤ *n-levels* (*mem-pool-info Va p*) **and**
a4:*alloc-l Va t* < *int* (*n-levels* (*mem-pool-info Va p*)) **and**
a5:¬ *free-l Va t* < *OK* **and**
a6:*free-l Va t* ≤ *from-l Va t* **and**
a7:*allocating-node Va t* =
Some (|*pool* = *p*, *level* = *nat* (*from-l Va t*),
block = *block-num* (*mem-pool-info Va p*)
(*buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div* *4* ˆ *nat* (*from-l Va t*)))
(*lsizes Va t* ! *nat* (*from-l Va t*)),
data = *buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div* *4* ˆ *nat* (*from-l Va t*))|) **and**
a8:*n* = *block-num* (*mem-pool-info Va p*)
(*buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div* *4* ˆ *nat* (*from-l Va t*)))
(*lsizes Va t* ! *nat* (*from-l Va t*)) ∨
max-sz (*mem-pool-info Va p*) *div* *4* ˆ *nat* (*from-l Va t*) = *NULL* **and**
a9:*block-num* (*mem-pool-info Va p*)
(*buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div* *4* ˆ *nat* (*from-l Va t*)))
(*lsizes Va t* ! *nat* (*from-l Va t*))
< *n-max* (*mem-pool-info Va p*) ∗ *4* ˆ *nat* (*from-l Va t*) **and**
a10:*from-l Va t* < *alloc-l Va t* **and**
a11:*blk Va t* = *buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div* *4* ˆ *nat* (*from-l Va t*)) **and**
a12:(*x*, *mp-alloc-stm4-pre-precond-f Va t p*) ∈ *gvars-conf-stable* **and**
a13:∀ *jj*. *jj* ≠ *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*) ⟶
levels (*mem-pool-info x p*) ! *jj* = *levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) ! *jj* **and**
a14:*bits* (*levels* (*mem-pool-info x p*) ! *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*)) =
list-updates-n
(*bits* (*levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*)))
(*Suc* (*bn* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* ∗ *4*)) *3* *FREE* **and**
a15:*free-list* (*levels* (*mem-pool-info x p*) ! *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*)) =
inserts
(*map* (λ*ii*. *lsizes* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*) ∗
ii +
blk (*mp-alloc-stm4-pre-precond-f Va t p*) *t*)
[*Suc NULL*..<*4*])
(*free-list*
(*levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*))) **and**
a16:*lsizes x* = *lsizes* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
a17:*from-l x* = *from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
a01:*ii* < *length* (*levels* (*mem-pool-info x p*)) **and**

a02:*jj* < *length* (*bits* (*levels* (*mem-pool-info x p*) ! *ii*)) **and**
a03:*get-bit-s x p ii jj* = *NOEXIST* **and**
a04:*0* < *ii*
shows *get-bit-s x p* (*ii* − *1*) (*jj div 4*) ≠ *DIVIDED*
proof−
let *?mp* = *mem-pool-info x p*
have *inv*:*inv-aux-vars Va* ∧ *inv-bitmap Va* ∧ *inv-mempool-info Va* ∧ *inv-bitmap-freelist Va*
using *a0* **unfolding** *inv-def* **by** *auto*
have *from-l-gt0*:*0* ≤ *from-l Va t* **using** *a6 a5* **by** *linarith*
have *len-levels*:*length* (*levels* (*mem-pool-info x p*)) = *length* (*levels* (*mem-pool-info Va p*))
using *mp-alloc-stm4-lvl-len*[*OF a1 a12*] **by** *simp*
have *maxsz*:*max-sz* (*mem-pool-info x p*) = *max-sz* (*mem-pool-info Va p*)
using *mp-alloc-stm4-maxsz*[*OF a1 a12*] **by** *simp*
have *buf*:*buf* (*mem-pool-info x p*) = *buf* (*mem-pool-info Va p*)
using *mp-alloc-stm4-buf*[*OF a1 a12*] **by** *simp*
have *from-l*:*from-l x* = *from-l Va*
using *mp-alloc-stm4-froml*[*OF a17*] **by** *auto*
have *from-l-suc*:*nat* (*from-l Va t* + *1*) = *nat*(*from-l Va t*) + *1*
using *from-l-gt0* **by** *auto*
have *mem-pools*:*mem-pools x* = *mem-pools Va* **using** *mp-alloc-stm4-mempools*[*OF a12*] **by** *auto*
have *lsizes-x-va*:*lsizes x* = *lsizes Va* **using** *mp-alloc-stm4-pre-precond-f-lsz a16*
by *auto*
let *?i1*=(*nat* (*from-l Va t*)) **and**
?j1= (*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*))) **and**
?i2 = (*nat* (*from-l Va t* + *1*)) **and**
?j2 = (*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*))∗*4*) **and**
?i1' = (*nat* (*from-l Va t*)) − *1* **and**
?j1' = (*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*))) *div 4* **and**
?i2' = (*nat* (*from-l Va t*)) + *2*
let *?j20'* = *?j2* ∗ *4* **and** *?j21'* = (*?j2*+*1*) ∗ *4* **and** *?j22'* = (*?j2*+*2*)∗*4* **and**
?j23' = (*?j2*+*3*)∗*4* **and** *?j24'* = (*?j2*+*4*)∗*4*
let *?mp* = *mem-pool-info x p*
have *inv-mempool-info-mp Va p*
using *a1 mem-pools inv* **unfolding** *inv-mempool-info-def* **by** *auto*
note *inv-mempool*=*this*[*simplified Let-def*]
have *i1-len*:*?i1* < *length* (*levels* (*mem-pool-info Va p*))
using *a10 a1 a4 from-l-gt0 inv* **unfolding** *inv-mempool-info-def Let-def*
by *auto*
have *i2-len*:*?i2* < *length* (*levels* (*mem-pool-info Va p*))
using *a10 a1 a4 from-l-gt0 inv* **unfolding** *inv-mempool-info-def Let-def*
by *auto*
have *j1-len*:*?j1* < *length* (*bits* (*levels* (*mem-pool-info Va p*) ! *?i1*))
by (*metis i1-len a9 a11 a1 inv inv-mempool-info-def*)

have *j2-len*:*Suc* (*Suc* (*Suc* *?j2*)) < *length* (*bits* (*levels* (*mem-pool-info* *Va* *p*) ! *?i2*))
  using *i1-len i2-len j1-len inv-mempool from-l-suc*
  by *simp*
let *?bts* = *bits* (*levels* *?mp* ! *ii*)
let *?btsva* = (*bits* (*levels* (*mem-pool-info* *Va* *p*) ! *ii*))
have *a01'*:*ii* < *length* (*levels* (*mem-pool-info* *Va* *p*))
  using *a01 len-levels* **by** *auto*
then have *inv-bitmap1*:
 $\forall$*j* < *length* (*bits* (*levels* (*mem-pool-info* *Va* *p*) ! *ii*)).
   (*?btsva* ! *j* = *FREE* $\vee$ *?btsva* ! *j* = *FREEING* $\vee$ *?btsva* ! *j* = *ALLOCATED* $\vee$ *?btsva* ! *j* = *ALLOCATING* $\longrightarrow$
     (*ii* > *0* $\longrightarrow$ (*bits* (*levels* (*mem-pool-info* *Va* *p*) ! (*ii* − *1*))) ! (*j* *div* *4*) = *DIVIDED*)
     $\wedge$ (*ii* < *length* (*levels* (*mem-pool-info* *Va* *p*)) − *1* $\longrightarrow$ *noexist-bits* (*mem-pool-info* *Va* *p*) (*ii*+*1*) (*j*∗*4*)))
   $\wedge$ (*?btsva* ! *j* = *DIVIDED* $\longrightarrow$ *ii* > *0* $\longrightarrow$ (*bits* (*levels* (*mem-pool-info* *Va* *p*) ! (*ii* − *1*))) ! (*j* *div* *4*) = *DIVIDED*)
   $\wedge$ (*?btsva* ! *j* = *NOEXIST* $\longrightarrow$ *ii* < *length* (*levels* (*mem-pool-info* *Va* *p*)) − *1*
     $\longrightarrow$ *noexist-bits* (*mem-pool-info* *Va* *p*) (*ii*+*1*) (*j*∗*4*))
   $\wedge$ (*?btsva* ! *j* = *NOEXIST* $\wedge$ *ii* > *0* $\longrightarrow$ (*bits* (*levels* (*mem-pool-info* *Va* *p*) ! (*ii* − *1*))) ! (*j* *div* *4*) $\neq$ *DIVIDED*)
  using *inv mem-pools a1*
  unfolding *Let-def inv-bitmap-def*
  by *blast*
have *alloc-i1-j1*:*get-bit-s* *Va* *p* *?i1* *?j1* = *ALLOCATING*
  using *a7 a0 a11* **unfolding** *inv-aux-vars-def invariant.inv-def*
 by (*metis* (*no-types*) *Mem-block.select-convs*(*1*) *Mem-block.select-convs*(*2*) *Mem-block.select-convs*(*3*))

then have *alloc-predi1-j1*:*?i1* > *0* $\longrightarrow$ *get-bit-s* *Va* *p* (*?i1* − *1*) (*?j1* *div* *4*) = *DIVIDED*
  using *inv-bitmap1 i1-len j1-len inv a1* **unfolding** *Let-def inv-bitmap-def* **by** *blast*
have *nexisti2*:*noexist-bits* (*mem-pool-info* *Va* *p*) *?i2* *?j2*
  using *a1 conjunct1*[*OF conjunct2*[*OF inv*], *simplified Let-def inv-bitmap-def*] *i1-len j1-len*
   alloc-i1-j1 from-l-suc i2-len i1-len j1-len a1
 by (*smt One-nat-def Suc-pred add.commute inv-mempool nat-add-left-cancel-less plus-1-eq-Suc*)
have *nexisti3*:*?i2* < *length* (*levels* (*mem-pool-info* *Va* *p*)) − *1* $\longrightarrow$
   noexist-bits (*mem-pool-info* *Va* *p*) *?i2'* *?j20'* $\wedge$
   noexist-bits (*mem-pool-info* *Va* *p*) *?i2'* *?j21'* $\wedge$
   noexist-bits (*mem-pool-info* *Va* *p*) *?i2'* *?j22'* $\wedge$
   noexist-bits (*mem-pool-info* *Va* *p*) *?i2'* *?j23'*
proof−
 { **assume** *?i2* < *length* (*levels* (*mem-pool-info* *Va* *p*)) − *1*
  then have *a00*:$\forall$*j*<*length* (*bits* (*levels* (*mem-pool-info* *Va* *p*) ! *?i2*)).
    get-bit-s *Va* *p* *?i2* *j* = *NOEXIST* $\longrightarrow$ *noexist-bits* (*mem-pool-info* *Va*

p) *?i2′* (*j* ∗ *4*)
using *a1 conjunct1*[*OF conjunct2*[*OF inv*], *simplified Let-def inv-bitmap-def*] *i2-len*
from-l-suc **by** *auto*
then have *noexist-bits* (*mem-pool-info Va p*) *?i2′ ?j20′* ∧
noexist-bits (*mem-pool-info Va p*) *?i2′ ?j21′* ∧
noexist-bits (*mem-pool-info Va p*) *?i2′ ?j22′* ∧
noexist-bits (*mem-pool-info Va p*) *?i2′ ?j23′*
using *j2-len nexisti2 Suc-lessD*
by (*smt One-nat-def add.commute add-2-eq-Suc′ add-Suc-right numeral-3-eq-3 plus-1-eq-Suc*)
}
thus *?thesis* **by** *fastforce*
qed
let *?bts* = *bits* (*levels ?mp* ! *ii*) **and** *?fl* = *free-list* (*levels ?mp* ! *ii*)
have *a02′*:*jj* < *length* (*bits* (*levels* (*mem-pool-info Va p*) ! *ii*))
using *a02 a12* **unfolding** *gvars-conf-def gvars-conf-stable-def*
by (*simp add*: *mp-alloc-stm4-inv-bits-len*)
have *eq-len*:*length* (*bits* (*levels* (*mem-pool-info x p*) ! *ii*)) =
length (*bits* (*levels* (*mem-pool-info Va p*) ! *ii*))
using *mp-alloc-stm4-inv-bits-len a13 a14 length-list-update-n*
by *metis*
have *inv-va*:(*?btsva* ! *jj* = *FREE* ∨ *?btsva* ! *jj* = *FREEING* ∨ *?btsva* ! *jj* = *ALLOCATED* ∨ *?btsva* ! *jj* = *ALLOCATING* ⟶
(*ii* > *0* ⟶ (*bits* (*levels* (*mem-pool-info Va p*) ! (*ii* − *1*))) ! (*jj div 4*) = *DIVIDED*)
∧ (*ii* < *length* (*levels* (*mem-pool-info Va p*)) − *1* ⟶ *noexist-bits* (*mem-pool-info Va p*) (*ii*+*1*) (*jj*∗*4*)))
∧ (*?btsva* ! *jj* = *DIVIDED* ⟶ *ii* > *0* ⟶ (*bits* (*levels* (*mem-pool-info Va p*) ! (*ii* − *1*))) ! (*jj div 4*) = *DIVIDED*)
∧ (*?btsva* ! *jj* = *NOEXIST* ⟶ *ii* < *length* (*levels* (*mem-pool-info Va p*)) − *1*
⟶ *noexist-bits* (*mem-pool-info Va p*) (*ii*+*1*) (*jj*∗*4*))
∧ (*?btsva* ! *jj* = *NOEXIST* ∧ *ii* > *0* ⟶ (*bits* (*levels* (*mem-pool-info Va p*) ! (*ii* − *1*))) ! (*jj div 4*) ≠ *DIVIDED*)
using *inv-bitmap1 a02′* **by** *auto*
{ assume *a05*:¬((*ii*=*?i1* ∧ *jj*=*?j1*) ∨
(*ii*=*?i2* ∧ *jj*≥ *?j2* ∧ *jj*< *?j2*+*4*) ∨
(*ii*=*?i2′* ∧ *jj*≥ *?j20′* ∧ *jj*< *?j24′*))
then have *a050′*:¬(*ii*=*?i1* ∧ *jj*=*?j1*) **and**
a051′: ¬(*ii*=*?i2* ∧ *jj*≥ *?j2* ∧ *jj*<*?j2* + *4*) **and**
a052′:¬(*ii*=*?i2′* ∧ *jj*≥ *?j20′* ∧ *jj*< *?j24′*)
by *force+*
have *eq-get-bit-i-j*:*get-bit-s x p ii jj* = *get-bit-s Va p ii jj*
using *same-bit-mp-alloc-x-va*[*OF a13*[*simplified a17*[*simplified mp-alloc-stm4-froml*[*OF a17*], *THEN sym*]] *a14*, *of ii jj*]
using *a050′ a051′* **by** *auto*
have *eq-get-bit-i1-j1*:*ii*>*0* ⟶ *get-bit-s x p* (*ii*−*1*) (*jj div 4*) = *get-bit-s Va p* (*ii*−*1*) (*jj div 4*)

proof−
{ **assume** *a06*:*ii*>*0*
then have ¬((*ii* − *1*) = *?i1* ∧ *jj div 4* = *?j1*)
using *a050′ a051′ from-l-suc* **by** *fastforce*
moreover have ∀ *j*. *j*≥ *?j2* ∧ *j*≤ *?j2*+*3* ⟶ ¬((*ii* − *1*) = *?i2* ∧ *jj div 4* = *j*)
using *a051′ a052′ from-l-gt0* **by** *fastforce*
ultimately have *get-bit-s x p* (*ii*−*1*) (*jj div 4*) = *get-bit-s Va p* (*ii*−*1*) (*jj div 4*)
using *same-bit-mp-alloc-x-va*[*OF a13*[*simplified a17*[*simplified mp-alloc-stm4-froml*[*OF a17*], *THEN sym*]] *a14*,
of ii −*1 jj div 4*] **by** *auto*
} **thus** *?thesis* **by** *auto* **qed**
then have *?thesis*
using *a03 a04 eq-get-bit-i1-j1 eq-get-bit-i-j inv-va* **by** *auto*
}
moreover {
assume *a06*:(*ii*=*?i1* ∧ *jj*=*?j1*)
then have *get-bit-s x p ii jj* = *DIVIDED*
using *get-bit-x-l-b a13 a17 from-l from-l-gt0 i1-len j1-len* **by** *presburger*
then have *?thesis* **using** *a03* **by** *auto*
}
moreover {
assume *a06*: (*ii*=*?i2* ∧ *jj*≥ *?j2* ∧ *jj*<*?j2*+*4*)
then have *a06′*:*jj*=*?j2* ∨ *jj*=*?j2*+*1* ∨ *jj*=*?j2*+*2* ∨ *jj* =*?j2* + *3* **by** *auto*
{ **assume** *a08*:*jj*=*?j2*
then have *get-bit*:*get-bit-s x p ii jj* = *ALLOCATING*
using *a02 a06 a14 eq-len get-bit-x-l1-b4 a04 i2-len from-l-gt0 i1-len j1-len*

by (*metis mult.commute*)
then have *?thesis* **using** *a03* **by** *auto*
}
moreover {
assume *a07*:*jj*≠*?j2*
have *a07′*:*jj div 4* = *?j1* **using** *a06 a07* **by** *auto*
have *get-bit-s x p ii jj* = *FREE*
using *a06 a02 a14 a07 from-l mp-alloc-stm4-inv-bits-len a17 mp-alloc-stm4-pre-precond-f-bn*
by (*auto simp add*: *mp-alloc-stm4-pre-precond-f-bn*)
then have *?thesis* **using** *a03* **by** *auto*
}
ultimately have *?thesis* **using** *a06* **by** *fastforce*
}
moreover {
assume *a06*: (*ii*=*?i2′* ∧ *jj*≥ *?j20′* ∧ *jj*< *?j24′*)
then have *a06′*:*jj*=*?j20′* ∨ *jj*=*?j20′*+*1* ∨ *jj*=*?j20′*+*2* ∨ *jj*=*?j20′*+*3* ∨
jj=*?j21′* ∨ *jj*=*?j21′*+*1* ∨ *jj*=*?j21′*+*2* ∨ *jj*=*?j21′*+*3* ∨
jj=*?j22′* ∨ *jj*=*?j22′*+*1* ∨ *jj*=*?j22′*+*2* ∨ *jj*=*?j22′*+*3* ∨
jj=*?j23′* ∨ *jj*=*?j23′*+*1* ∨ *jj*=*?j23′*+*2* ∨ *jj*=*?j23′* + *3*
by *presburger*

have *ij*:(*ii*−*1* = *?i2* ∧ (*jj div 4*)≥ *?j2* ∧ (*jj div 4*) ≤ *?j2* + *3*)
using *a04 a06 from-l-gt0* **by** *auto*
{ **assume** *a08*:(*jj div 4*)=*?j2*
then have *get-bit*:*get-bit-s x p* (*ii*−*1*) (*jj div 4*) = *ALLOCATING*
using *ij a02 a14 eq-len get-bit-x-l1-b4 a04 i2-len from-l-gt0 i1-len j1-len*
by (*metis Suc-lessD j2-len mult.commute*)
then have *?thesis* **using** *a03* **by** *auto*
}
moreover {
assume *a07*:(*jj div 4*)≠*?j2*
then have *ii*−*1* = *?i2* ∧ (*jj div 4* = *Suc ?j2* ∨ *jj div 4* = *Suc* (*Suc ?j2*) ∨
jj div 4 = *Suc* (*Suc* (*Suc?j2*)))
using *ij* **by** *auto*
then have *get-bit-s x p* (*ii*−*1*) (*jj div 4*) = *FREE*
using *ij a01 a02 i2-len j2-len*
get-bit-x-l1-b41[*OF - from-l-gt0*[*simplified from-l a17*]
a13[*simplified a17*[*THEN sym*] *from-l*] *a14*,*of ii*−*1 jj*
div 4]
by (*metis Suc-lessD mult.commute*)
then have *?thesis* **using** *a03* **by** *auto*
}
ultimately have *?thesis* **using** *a06* **by** *fastforce*
}
ultimately show *?thesis* **by** *fastforce*
qed

lemma *mp-alloc-stm4-inv-bitmap*:
assumes
a0:*inv Va* **and**
a1:*freeing-node Va t* = *None* **and**
a2:*p* ∈ *mem-pools Va* **and**
a3:*ETIMEOUT* ≤ *timeout* **and**
a4:*timeout* = *ETIMEOUT* ⟶ *tmout Va t* = *ETIMEOUT* **and**
a5:¬ *rf Va t* **and**
a6:∀ *ii*<*length* (*lsizes Va t*). *lsizes Va t* ! *ii* = *ALIGN4* (*max-sz* (*mem-pool-info*
Va p)) *div 4* ˆ *ii* **and**
a7:*length* (*lsizes Va t*) ≤ *n-levels* (*mem-pool-info Va p*) **and**
a8:*alloc-l Va t* < *int* (*n-levels* (*mem-pool-info Va p*)) **and**
a9:¬ *free-l Va t* < *OK* **and**
a10:*NULL* < *buf* (*mem-pool-info Va p*) ∨ *NULL* < *n* ∧ *NULL* < *max-sz* (*mem-pool-info*
Va p) *div 4* ˆ *nat* (*from-l Va t*) **and**
a11:*free-l Va t* ≤ *from-l Va t* **and**
a12:*allocating-node Va t* =
Some (|*pool* = *p*, *level* = *nat* (*from-l Va t*),
block = *block-num* (*mem-pool-info Va p*)
(*buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div*
4 ˆ *nat* (*from-l Va t*)))
(*lsizes Va t* ! *nat* (*from-l Va t*)),
data = *buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div*

4 ^ nat (from-l Va t))⦈ **and**
a13:*n = block-num (mem-pool-info Va p)*
(buf (mem-pool-info Va p) + n ∗ (max-sz (mem-pool-info Va p) div 4 ^ nat (from-l Va t)))
(lsizes Va t ! nat (from-l Va t)) ∨
max-sz (mem-pool-info Va p) div 4 ^ nat (from-l Va t) = NULL **and**
a14:*block-num (mem-pool-info Va p)*
(buf (mem-pool-info Va p) + n ∗ (max-sz (mem-pool-info Va p) div 4 ^ nat (from-l Va t)))
(lsizes Va t ! nat (from-l Va t))
< n-max (mem-pool-info Va p) ∗ 4 ^ nat (from-l Va t) **and**
a15:*from-l Va t < alloc-l Va t* **and**
a16:*cur Va = Some t* **and**
a17:*n < n-max (mem-pool-info Va p) ∗ 4 ^ nat (from-l Va t)* **and**
a18:*blk Va t = buf (mem-pool-info Va p) + n ∗ (max-sz (mem-pool-info Va p) div 4 ^ nat (from-l Va t))* **and**
a19:*mempoolalloc-ret Va t = None* **and**
a20:∀ *ii*≤*nat (alloc-l Va t). sz ≤ lsizes Va t ! ii* **and**
a21:*alloc-l Va t = int (length (lsizes Va t)) − 1 ∧ length (lsizes Va t) = n-levels (mem-pool-info Va p)* ∨
alloc-l Va t = int (length (lsizes Va t)) − 2 ∧ lsizes Va t ! nat (alloc-l Va t + 1) < sz **and**
a22:*i x t = 4* **and**
a23:*cur x = cur (mp-alloc-stm4-pre-precond-f Va t p)* **and**
a24:*tick x = tick (mp-alloc-stm4-pre-precond-f Va t p)* **and**
a25:*thd-state x = thd-state (mp-alloc-stm4-pre-precond-f Va t p)* **and**
a26:*(x, mp-alloc-stm4-pre-precond-f Va t p) ∈ gvars-conf-stable* **and**
a27:∀ *pa. pa ≠ p ⟶ mem-pool-info x pa = mem-pool-info (mp-alloc-stm4-pre-precond-f Va t p) pa* **and**
a28:*wait-q (mem-pool-info x p) = wait-q (mem-pool-info (mp-alloc-stm4-pre-precond-f Va t p) p)* **and**
a29:∀ *t′. t′ ≠ t ⟶ lvars-nochange t′ x (mp-alloc-stm4-pre-precond-f Va t p)* **and**
a30:∀ *jj. jj ≠ nat (from-l (mp-alloc-stm4-pre-precond-f Va t p) t + 1) ⟶*
levels (mem-pool-info x p) ! jj = levels (mem-pool-info (mp-alloc-stm4-pre-precond-f Va t p) p) ! jj **and**
a31:*bits (levels (mem-pool-info x p) ! nat (from-l (mp-alloc-stm4-pre-precond-f Va t p) t + 1)) =*
list-updates-n
(bits (levels (mem-pool-info (mp-alloc-stm4-pre-precond-f Va t p) p) !
nat (from-l (mp-alloc-stm4-pre-precond-f Va t p) t + 1)))
(Suc (bn (mp-alloc-stm4-pre-precond-f Va t p) t ∗ 4)) 3 FREE **and**
a32:*free-list (levels (mem-pool-info x p) ! nat (from-l (mp-alloc-stm4-pre-precond-f Va t p) t + 1)) =*
inserts
(map (λii. lsizes (mp-alloc-stm4-pre-precond-f Va t p) t !
nat (from-l (mp-alloc-stm4-pre-precond-f Va t p) t + 1) ∗
ii +
blk (mp-alloc-stm4-pre-precond-f Va t p) t)
[Suc NULL..<4])

(*free-list*
(*levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*))) **and**
a33:*j x* = *j* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
a34:*ret x* = *ret* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
a35:*endt x* = *endt* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
a36:*rf x* = *rf* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
a37:*tmout x* = *tmout* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
a38:*lsizes x* = *lsizes* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
a39:*alloc-l x* = *alloc-l* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
a40:*free-l x* = *free-l* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
a41:*from-l x* = *from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
a42:*blk x* = *blk* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
a43:*nodev x* = *nodev* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
a44:*bn x* = *bn* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
a45:*alloc-lsize-r x* = *alloc-lsize-r* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
a46:*lvl x* = *lvl* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
a47:*bb x* = *bb* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
a48:*block-pt x* = *block-pt* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
a49:*th x* = *th* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
a50:*need-resched x* = *need-resched* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
a51:*mempoolalloc-ret x* = *mempoolalloc-ret* (*mp-alloc-stm4-pre-precond-f Va t p*)
and
a52:*freeing-node x* = *freeing-node* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
a53:*allocating-node x* = *allocating-node* (*mp-alloc-stm4-pre-precond-f Va t p*)
shows *inv-bitmap x*
proof−
let *?mp* = *mem-pool-info x p*
have *inv*:*inv-aux-vars Va* ∧ *inv-bitmap Va* ∧ *inv-mempool-info Va* ∧ *inv-bitmap-freelist Va*
using *a0* **unfolding** *inv-def* **by** *auto*
have *from-l-gt0*:*0* ≤ *from-l Va t* **using** *a11 a9* **by** *linarith*
have *len-levels*:*length* (*levels* (*mem-pool-info x p*)) = *length* (*levels* (*mem-pool-info Va p*))
using *mp-alloc-stm4-lvl-len*[*OF a2 a26*] **by** *simp*
have *maxsz*:*max-sz* (*mem-pool-info x p*) = *max-sz* (*mem-pool-info Va p*)
using *mp-alloc-stm4-maxsz*[*OF a2 a26*] **by** *simp*
have *buf*:*buf* (*mem-pool-info x p*) = *buf* (*mem-pool-info Va p*)
using *mp-alloc-stm4-buf*[*OF a2 a26*] **by** *simp*
have *from-l*:*from-l x* = *from-l Va*
using *mp-alloc-stm4-froml*[*OF a41*] **by** *auto*
have *mem-pools*:*mem-pools x* = *mem-pools Va* **using** *mp-alloc-stm4-mempools*[*OF a26*] **by** *auto*
have *lsizes-x-va*:*lsizes x* = *lsizes Va* **using** *mp-alloc-stm4-pre-precond-f-lsz a38*
by *auto*
have *from-l-gt0*:*OK* ≤ *from-l Va t* **using** *a11 a9* **by** *linarith*
{ **fix** *p'*
assume *a00*:*p'*∈*mem-pools x*
let *?i1*=(*nat* (*from-l Va t*)) **and**

?j1= (*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*))) **and**
?i2 = (*nat* (*from-l Va t* + *1*)) **and**
?j2 = (*4*∗*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*))) **and**
?i1′ = (*nat* (*from-l Va t*)) − *1* **and**
?j1′ = (*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*))) *div 4* **and**
?i2′ = (*nat* (*from-l Va t*)) + *2* **and**
?j2′ = (*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*))) ∗ *16*

have *alloc-i1-j1*:*get-bit-s Va p ?i1 ?j1* = *ALLOCATING*
using *a12 a0 a18* **unfolding** *inv-aux-vars-def invariant.inv-def*
by (*metis* (*no-types*) *Mem-block.select-convs*(*1*) *Mem-block.select-convs*(*2*) *Mem-block.select-convs*(*3*))
{**assume** *p*≠*p′*
moreover have *mem-pool-info x p′* = *mem-pool-info Va p′*
using *mp-alloc-stm4-pres-mpinfo*
by (*metis a27 calculation*)
ultimately have *inv-bitmap-mp x p′*
using *a00 inv len-levels maxsz buf from-l mem-pools*
by(*simp add*: *inv-bitmap-def Let-def*)
}
moreover { **assume** *eq-p*:*p*=*p′*
let *?mp* = *mem-pool-info x p*
have *inv-mempool-info-mp Va p*
using *eq-p mem-pools a00 inv* **unfolding** *inv-mempool-info-def* **by** *auto*
note *inv-mempool*=*this*[*simplified Let-def*]
{**fix** *i*
assume *a01*:*i*<*length* (*levels ?mp*)
let *?bts* = *bits* (*levels ?mp* ! *i*)
let *?btsva* = (*bits* (*levels* (*mem-pool-info Va p*) ! *i*))
have *a01′*:*i* < *length* (*levels* (*mem-pool-info Va p*))
using *a01 len-levels* **by** *auto*
then have *inv-bitmap1*:
∀*j* < *length* (*bits* (*levels* (*mem-pool-info Va p*) ! *i*)).
(*?btsva* ! *j* = *FREE* ∨ *?btsva* ! *j* = *FREEING* ∨ *?btsva* ! *j* = *ALLOCATED* ∨ *?btsva* ! *j* = *ALLOCATING* ⟶
(*i* > *0* ⟶ (*bits* (*levels* (*mem-pool-info Va p*) ! (*i* − *1*))) ! (*j div 4*) = *DIVIDED*)
∧ (*i* < *length* (*levels* (*mem-pool-info Va p*)) − *1* ⟶ *noexist-bits* (*mem-pool-info Va p*) (*i*+*1*) (*j*∗*4*)))
∧ (*?btsva* ! *j* = *DIVIDED* ⟶ *i* > *0* ⟶ (*bits* (*levels* (*mem-pool-info Va p*) ! (*i* − *1*))) ! (*j div 4*) = *DIVIDED*)
∧ (*?btsva* ! *j* = *NOEXIST* ⟶ *i* < *length* (*levels* (*mem-pool-info Va p*)) − *1*
⟶ *noexist-bits* (*mem-pool-info Va p*) (*i*+*1*) (*j*∗*4*))
∧ (*?btsva* ! *j* = *NOEXIST* ∧ *i* > *0* ⟶ (*bits* (*levels* (*mem-pool-info*

Va p) ! (*i* − *1*))) ! (*j div 4*) ≠ *DIVIDED*)
using *inv eq-p mem-pools a00*
unfolding *Let-def inv-bitmap-def*
by *blast*
let *?bts* = *bits* (*levels ?mp* ! *i*) **and** *?fl* = *free-list* (*levels ?mp* ! *i*)
have *f1*:∀ *j* < *length ?bts*.
(*?bts* ! *j* = *FREE* ∨ *?bts* ! *j* = *FREEING* ∨ *?bts* ! *j* = *ALLOCATED* ∨ *?bts* ! *j* = *ALLOCATING* ⟶
(*i* > *0* ⟶ (*bits* (*levels* (*mem-pool-info x p*) ! (*i* − *1*))) ! (*j div 4*) = *DIVIDED*)
∧ (*i* < *length* (*levels* (*mem-pool-info x p*)) − *1* ⟶ *noexist-bits* (*mem-pool-info x p*) (*i*+*1*) (*j*∗*4*)))
∧ (*?bts* ! *j* = *DIVIDED* ⟶ *i* > *0* ⟶ (*bits* (*levels* (*mem-pool-info x p*) ! (*i* − *1*))) ! (*j div 4*) = *DIVIDED*)
∧ (*?bts* ! *j* = *NOEXIST* ⟶ *i* < *length* (*levels* (*mem-pool-info x p*)) − *1*
⟶ *noexist-bits* (*mem-pool-info x p*) (*i*+*1*) (*j*∗*4*))
∧ (*?bts* ! *j* = *NOEXIST* ∧ *i* > *0* ⟶ (*bits* (*levels* (*mem-pool-info x p*) ! (*i* − *1*))) ! (*j div 4*) ≠ *DIVIDED*)
proof−
{ fix *j*
assume *a02*:*j*<*length ?bts*
then have *a02'*:*j* < *length* (*bits* (*levels* (*mem-pool-info Va p*) ! *i*))
using *a26* **unfolding** *gvars-conf-def gvars-conf-stable-def*
by (*simp add*: *mp-alloc-stm4-inv-bits-len*)
have *eq-len*:*length* (*bits* (*levels* (*mem-pool-info x p*) ! *i*)) =
length (*bits* (*levels* (*mem-pool-info Va p*) ! *i*))
using *mp-alloc-stm4-inv-bits-len a30 a31 length-list-update-n*
by *metis*
have *inv-va*:(*?btsva* ! *j* = *FREE* ∨ *?btsva* ! *j* = *FREEING* ∨ *?btsva* ! *j* = *ALLOCATED* ∨ *?btsva* ! *j* = *ALLOCATING* ⟶
(*i* > *0* ⟶ (*bits* (*levels* (*mem-pool-info Va p*) ! (*i* − *1*))) ! (*j div 4*) = *DIVIDED*)
∧ (*i* < *length* (*levels* (*mem-pool-info Va p*)) − *1* ⟶ *noexist-bits* (*mem-pool-info Va p*) (*i*+*1*) (*j*∗*4*)))
∧ (*?btsva* ! *j* = *DIVIDED* ⟶ *i* > *0* ⟶ (*bits* (*levels* (*mem-pool-info Va p*) ! (*i* − *1*))) ! (*j div 4*) = *DIVIDED*)
∧ (*?btsva* ! *j* = *NOEXIST* ⟶ *i* < *length* (*levels* (*mem-pool-info Va p*)) − *1*
⟶ *noexist-bits* (*mem-pool-info Va p*) (*i*+*1*) (*j*∗*4*))
∧ (*?btsva* ! *j* = *NOEXIST* ∧ *i* > *0* ⟶ (*bits* (*levels* (*mem-pool-info Va p*) ! (*i* − *1*))) ! (*j div 4*) ≠ *DIVIDED*)
using *inv-bitmap1 a02' eq-p* **by** *auto*
let *?goal1* = (*?bts* ! *j* = *FREE* ∨ *?bts* ! *j* = *FREEING* ∨ *?bts* ! *j* = *ALLOCATED* ∨ *?bts* ! *j* = *ALLOCATING* ⟶
(*i* > *0* ⟶ (*bits* (*levels* (*mem-pool-info x p*) ! (*i* − *1*))) ! (*j div 4*) = *DIVIDED*)
∧ (*i* < *length* (*levels* (*mem-pool-info x p*)) − *1* ⟶ *noexist-bits* (*mem-pool-info x p*) (*i*+*1*) (*j*∗*4*)))

let *?goal2* =(*?bts* ! *j* = *DIVIDED* ⟶ *i* > *0* ⟶ (*bits* (*levels* (*mem-pool-info x p*) ! (*i* − *1*))) ! (*j div 4*) = *DIVIDED*)
let *?goal3* =(*?bts* ! *j* = *NOEXIST* ⟶ *i* < *length* (*levels* (*mem-pool-info x p*)) − *1*
⟶ *noexist-bits* (*mem-pool-info x p*) (*i*+*1*) (*j*∗*4*))
let *?goal4* = (*?bts* ! *j* = *NOEXIST* ∧ *i* > *0* ⟶ (*bits* (*levels* (*mem-pool-info x p*) ! (*i* − *1*))) ! (*j div 4*) ≠ *DIVIDED*)
have *?goal1* **using** *eq-p*
mp-alloc-stm4-inv-bitmap1[*OF a0 a2 a6 a8 a9 a11 a12 a13 a14 a15 a17 a18 a26 a30 a31 a32 a38 a41 a01 a02*]
by *auto*
moreover have *?goal2*
using *mp-alloc-stm4-inv-bitmap2*[*OF a0 a2 a6 a7 a8 a9 a11 a12 a13 a14 a15 a18 a26 a30 a31 a32 a38 a41 a01 a02*]
by *auto*
moreover have *?goal3*
using *mp-alloc-stm4-inv-bitmap3*[*OF a0 a2 a6 a8 a9 a11 a12 a13 a14 a15 a17 a18 a26 a30 a31 a32 a38 a41 a01 a02*]
by *auto*
moreover have *?goal4* **using** *mp-alloc-stm4-inv-bitmap4*[*OF a0 a2 a6 a7 a8 a9 a11 a12 a13 a14 a15 a18 a26 a30 a31 a32 a38 a41 a01 a02*]
by *auto*
ultimately have *?goal1* ∧ *?goal2* ∧ *?goal3* ∧ *?goal4*
by *blast*
} **thus** *?thesis* **by** *auto*
qed
} **then have** *inv-bitmap-mp x p′* **using** *eq-p* **by** *auto*
} **ultimately have** *inv-bitmap-mp x p′* **by** *fastforce*
} **then show** *?thesis* **unfolding** *inv-bitmap-def* **by** *auto*
qed

lemma *mp-alloc-stm4-inv-aux-vars1*:
assumes
a0:*inv Va* **and**
a1:*freeing-node Va t* = *None* **and**
a2:*p* ∈ *mem-pools Va* **and**
a3:∀ *ii*<*length* (*lsizes Va t*). *lsizes Va t* ! *ii* = *ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *ii* **and**
a4:*length* (*lsizes Va t*) ≤ *n-levels* (*mem-pool-info Va p*) **and**
a5:*alloc-l Va t* < *int* (*n-levels* (*mem-pool-info Va p*)) **and**
a6:¬ *free-l Va t* < *OK* **and**
a7:*free-l Va t* ≤ *from-l Va t* **and**
a8:*allocating-node Va t* =
Some (|*pool* = *p*, *level* = *nat* (*from-l Va t*),
block = *block-num* (*mem-pool-info Va p*)
(*buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ˆ *nat* (*from-l Va t*)))
(*lsizes Va t* ! *nat* (*from-l Va t*)),

data = *buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div* *4* ˆ *nat* (*from-l Va t*))⦈ **and**
a9:*block-num* (*mem-pool-info Va p*)
(*buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div* *4* ˆ *nat* (*from-l Va t*)))
(*lsizes Va t* ! *nat* (*from-l Va t*))
< *n-max* (*mem-pool-info Va p*) ∗ *4* ˆ *nat* (*from-l Va t*) **and**
a10:*from-l Va t* < *alloc-l Va t* **and**
a11:*blk Va t* = *buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div* *4* ˆ *nat* (*from-l Va t*)) **and**
a12:(*x*, *mp-alloc-stm4-pre-precond-f Va t p*) ∈ *gvars-conf-stable* **and**
a13:∀ *pa*. *pa* ≠ *p* ⟶ *mem-pool-info x pa* = *mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *pa* **and**
a14:∀ *jj*. *jj* ≠ *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*) ⟶
levels (*mem-pool-info x p*) ! *jj* = *levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) ! *jj* **and**
a15:*bits* (*levels* (*mem-pool-info x p*) ! *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*)) =
list-updates-n
(*bits* (*levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*)))
(*Suc* (*bn* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* ∗ *4*)) *3 FREE* **and**
a16:*free-list* (*levels* (*mem-pool-info x p*) ! *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*)) =
inserts
(*map* (λ*ii*. *lsizes* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*) ∗
ii +
blk (*mp-alloc-stm4-pre-precond-f Va t p*) *t*)
[*Suc NULL*..<*4*])
(*free-list*
(*levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*))) **and**
a17:*lsizes x* = *lsizes* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
a18:*from-l x* = *from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
a19:*freeing-node x* = *freeing-node* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
a20:*allocating-node x* = *allocating-node* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
a21:*freeing-node x t′* = *Some m*
shows *get-bit-s x* (*pool m*) (*level m*) (*block m*) = *FREEING*
proof−
have *inv*:*inv-aux-vars Va* ∧ *inv-bitmap Va* ∧ *inv-mempool-info Va* ∧ *inv-bitmap-freelist Va*
using *a0* **unfolding** *inv-def* **by** *auto*
have *from-l-gt0*:*0* ≤ *from-l Va t* **using** *a7 a6* **by** *linarith*
have *inv-aux-va*:(∀ *t n*. *freeing-node Va t* = *Some n* ⟶
get-bit (*mem-pool-info Va*) (*pool n*) (*level n*) (*block n*) = *FREEING*)
using *a0* **unfolding** *inv-def inv-aux-vars-def*
by *blast*
let *?i1*=(*nat* (*from-l Va t*)) **and**

?j1= (*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*))) **and**
?i2 = (*nat* (*from-l Va t* + 1)) **and**
?j2 = (*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*))*4)
have *mem-pools*:*mem-pools x* = *mem-pools Va* **using** *mp-alloc-stm4-mempools*[*OF a12*] **by** *auto*
have *inv-mempool-info-mp Va p*
using *a2 mem-pools inv* **unfolding** *inv-mempool-info-def Let-def* **by** *auto*
note *inv-mempool*=*this*[*simplified Let-def*]
have *from-l*:*from-l x* = *from-l Va*
using *mp-alloc-stm4-froml*[*OF a18*] **by** *auto*
have *from-l-suc*:*nat* (*from-l Va t* + 1) = *nat*(*from-l Va t*) + 1
using *from-l-gt0* **by** *auto*
have *i1-len*:?i1 < *length* (*levels* (*mem-pool-info Va p*))
using *a10 a2 a5 from-l-gt0 inv* **unfolding** *inv-mempool-info-def Let-def*
by *auto*
have *i2-len*:?i2 < *length* (*levels* (*mem-pool-info Va p*))
using *a10 a2 a5 from-l-gt0 inv* **unfolding** *inv-mempool-info-def Let-def*
by *auto*
have *j1-len*:?j1 < *length* (*bits* (*levels* (*mem-pool-info Va p*) ! ?i1))
by (*metis a0 a2 a9 a11 i1-len inv-mempool-info-def invariant.inv-def*)
have *j2-len*:*Suc* (*Suc* (*Suc* ?j2)) < *length* (*bits* (*levels* (*mem-pool-info Va p*) ! ?i2))
using *i1-len i2-len j1-len inv-mempool from-l-suc*
by *simp*
have *lsizes-x-va*:*lsizes x* = *lsizes Va* **using** *mp-alloc-stm4-pre-precond-f-lsz a17*
by *auto*
{ **assume** t'= *t*
then have ?thesis
using *a1 a19 a21*
by (*metis mp-alloc-stm4-pre-precond-f-def-frnode option.distinct*(1))
}
moreover {**assume** *a00*:$t' \neq t$
then have *freeing-node* (*mp-alloc-stm4-pre-precond-f Va t p*) t' = *freeing-node Va* t'
unfolding *mp-alloc-stm4-pre-precond-f-def* **by** *auto*
then have *eq-alloc*:*freeing-node Va* t' = *freeing-node x* t'
using *a19* **by** *auto*
then have *t2-same-allocating-node-Va*:*freeing-node Va* t' = *Some m*
using *a0 a21 a19*
unfolding *mp-alloc-stm4-pre-precond-f-def invariant.inv-def inv-aux-vars-def*

by *auto*
then have *diff-t*:¬(*pool m* = *p* ∧ *level m* = ?i1 ∧ *block m* = ?j1)
using *a00 a21 a8 inv* **unfolding** *inv-aux-vars-def*
by (*metis Mem-block.simps*(1) *Mem-block.simps*(2) *Mem-block.simps*(3) *a11*)
{
assume *pool m* ≠ *p*

then have *?thesis* **using** *a0 a13 a21 eq-alloc mp-alloc-stm4-pres-mpinfo*
unfolding *inv-aux-vars-def invariant.inv-def*
by *metis*
} **note** *not-pool-p-allocating = this*
moreover {
assume *a01:pool m = p*
have *bit-m-va-alloc:get-bit (mem-pool-info Va) (pool m) (level m) (block m) = FREEING*
using *a21 eq-alloc inv-aux-va* **by** *presburger*
have *maxsz:max-sz (mem-pool-info x p) = max-sz (mem-pool-info Va p)*
using *mp-alloc-stm4-maxsz[OF a2 a12]* **by** *simp*
have *buf:buf (mem-pool-info x p) = buf (mem-pool-info Va p)*
using *mp-alloc-stm4-buf[OF a2 a12]* **by** *simp*
have *alloc-i1-j1:get-bit-s Va p ?i1 ?j1 = ALLOCATING*
using *a8 a0 a11* **unfolding** *inv-aux-vars-def invariant.inv-def*
by *(metis (no-types) Mem-block.select-convs(1) Mem-block.select-convs(2) Mem-block.select-convs(3))*
have *nexisti2:noexist-bits (mem-pool-info Va p) ?i2 ?j2*
using *a2 conjunct1[OF conjunct2[OF inv], simplified Let-def inv-bitmap-def] i1-len j1-len*
alloc-i1-j1 from-l-suc i2-len i1-len j1-len a1
by *(smt One-nat-def Suc-pred add.commute inv-mempool nat-add-left-cancel-less plus-1-eq-Suc)*
{ **assume** *a02:(level m = ?i1 ∧ block m = ?j1)*
then have *?thesis* **using** *diff-t a01* **by** *auto*
}
moreover {
assume *a02:¬(level m = ?i1 ∧ block m = ?j1)*
{ **assume** *a03:¬(level m = ?i2 ∧ (block m)≥ ?j2 ∧ (block m)<?j2 + 4)*
then have *eq-get-bit-i-j:get-bit-s x p (level m) (block m) =*
get-bit-s Va p (level m) (block m)
using *same-bit-mp-alloc-x-va[OF a14[simplified*
a18[simplified mp-alloc-stm4-froml[OF a18], THEN sym]] a15, of level m block m]
a01 a02 **by** *auto*
then have *?thesis* **using** *a01 a20 inv-aux-va not-pool-p-allocating*
a21 eq-alloc inv-aux-va **by** *force*
}
moreover {
assume *a03:(level m = ?i2 ∧ (block m)≥ ?j2 ∧ (block m)<?j2 + 4)*
then have *block m = ?j2 ∨ block m = ?j2 + 1 ∨ block m = ?j2 + 2 ∨ block m = ?j2 + 3*
by *auto*
then have *?thesis* **using** *bit-m-va-alloc nexisti2 a01 a03* **by** *auto*
} **ultimately have** *?thesis* **by** *fastforce*
} **ultimately have** *?thesis* **by** *fastforce*
} **ultimately have** *?thesis* **by** *fastforce*
} **ultimately show** *?thesis* **by** *auto*
qed

lemma *mp-alloc-stm4-inv-aux-vars2*:
assumes
a0:*inv Va* **and**
a1:*freeing-node Va t = None* **and**
a2:*p* ∈ *mem-pools Va* **and**
a3:*alloc-l Va t* < *int* (*n-levels* (*mem-pool-info Va p*)) **and**
a4:¬ *free-l Va t* < *OK* **and**
a5:*free-l Va t* ≤ *from-l Va t* **and**
a6:*block-num* (*mem-pool-info Va p*)
(*buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ˆ *nat* (*from-l Va t*)))
(*lsizes Va t* ! *nat* (*from-l Va t*))
< *n-max* (*mem-pool-info Va p*) ∗ *4* ˆ *nat* (*from-l Va t*) **and**
a7:*from-l Va t* < *alloc-l Va t* **and**
a8:*blk Va t = buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ˆ *nat* (*from-l Va t*)) **and**
a9:(*x*, *mp-alloc-stm4-pre-precond-f Va t p*) ∈ *gvars-conf-stable* **and**
a10:∀ *pa*. *pa* ≠ *p* ⟶ *mem-pool-info x pa = mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *pa* **and**
a11:∀ *jj*. *jj* ≠ *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*) ⟶
levels (*mem-pool-info x p*) ! *jj = levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) ! *jj* **and**
a12:*bits* (*levels* (*mem-pool-info x p*) ! *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*)) =
list-updates-n
(*bits* (*levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*)))
(*Suc* (*bn* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* ∗ *4*)) *3 FREE* **and**
a13:*from-l x = from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
a14:*freeing-node x = freeing-node* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
a54:*get-bit-s x* (*pool m*) (*level m*) (*block m*) = *FREEING* ∧ *mem-block-addr-valid x m*
shows (∃ *t*. *freeing-node x t = Some m*)
proof−
have *inv*:*inv-aux-vars Va* ∧ *inv-bitmap Va* ∧ *inv-mempool-info Va* ∧ *inv-bitmap-freelist Va*
using *a0* **unfolding** *inv-def* **by** *auto*
have *from-l-gt0*:*0* ≤ *from-l Va t* **using** *a5 a4* **by** *linarith*
have *block-valid-va*:*mem-block-addr-valid Va m*
using *a2 a9 a54* *mp-alloc-stm4-buf mp-alloc-stm4-maxsz*
unfolding *mem-block-addr-valid-def* **by** *auto*
have *inv-aux-va*:(∀ *n*. *get-bit* (*mem-pool-info Va*) (*pool n*) (*level n*) (*block n*) = *FREEING* ∧ *mem-block-addr-valid Va n*
⟶ (∃ *t*. *freeing-node Va t = Some n*))
using *a0* **unfolding** *inv-def inv-aux-vars-def*
by *blast*
{**assume** (*pool m*) ≠ *p*

then have *get-bit-s Va* (*pool m*) (*level m*) (*block m*) = *get-bit-s x* (*pool m*) (*level m*) (*block m*)
using *a10*
by (*metis mp-alloc-stm4-pres-mpinfo*)
then have *?thesis* **using** *a54 inv-aux-va block-valid-va a14 mp-alloc-stm4-pre-precond-f-def-frnode*
by *metis*
}
moreover{
assume *a01*:*pool m* = *p*
let *?i1*=(*nat* (*from-l Va t*)) **and**
?j1= (*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*))) **and**
?i2 = (*nat* (*from-l Va t* + *1*)) **and**
?j2 = (*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*))∗*4*)
have *mem-pools*:*mem-pools x* = *mem-pools Va* **using** *mp-alloc-stm4-mempools*[*OF a9*] **by** *auto*
have *inv-mempool-info-mp Va p*
using *a2 mem-pools inv* **unfolding** *inv-mempool-info-def Let-def* **by** *auto*
note *inv-mempool*=*this*[*simplified Let-def*]
have *from-l*:*from-l x* = *from-l Va*
using *mp-alloc-stm4-froml*[*OF a13*] **by** *auto*
have *from-l-suc*:*nat* (*from-l Va t* + *1*) = *nat*(*from-l Va t*) + *1*
using *from-l-gt0* **by** *auto*
have *i1-len*:*?i1* < *length* (*levels* (*mem-pool-info Va p*))
using *a7 a2 a3 from-l-gt0 inv* **unfolding** *inv-mempool-info-def Let-def*
by *auto*
have *i2-len*:*?i2* < *length* (*levels* (*mem-pool-info Va p*))
using *a7 a2 a3 from-l-gt0 inv* **unfolding** *inv-mempool-info-def Let-def*
by *auto*
have *j1-len*:*?j1* < *length* (*bits* (*levels* (*mem-pool-info Va p*) ! *?i1*))
by (*metis a0 a2 a6 a8 i1-len inv-mempool-info-def invariant.inv-def*)
have *j2-len*:*Suc* (*Suc* (*Suc ?j2*)) < *length* (*bits* (*levels* (*mem-pool-info Va p*) ! *?i2*))
using *i1-len i2-len j1-len inv-mempool from-l-suc*
by *simp*
{ assume *a02*:¬(((*level m*)=*?i1* ∧ (*block m*)=*?j1*) ∨
((*level m*)=*?i2* ∧ (*block m*)≥ *?j2* ∧ (*block m*)< *?j2*+*4*))
then have *a020′*:¬((*level m*)=*?i1* ∧ (*block m*)=*?j1*) **and**
a021′: ¬((*level m*)=*?i2* ∧ (*block m*)≥ *?j2* ∧ (*block m*)<*?j2* + *4*)
by *force*+
then have *eq-get-bit-i-j*:*get-bit-s x p* (*level m*) (*block m*) = *get-bit-s Va p* (*level m*) (*block m*)
using *same-bit-mp-alloc-x-va*[*OF a11*[*simplified a13*[*simplified mp-alloc-stm4-froml*[*OF a13*], *THEN sym*]] *a12*, *of level m block m*]
using *a020′ a021′* **by** *auto*
then have *?thesis* **using** *a01 a54 inv-aux-va*
block-valid-va a14 mp-alloc-stm4-pre-precond-f-def-frnode

by *metis*
}
moreover{
assume *a02*:((*level m*)=*?i1* ∧ (*block m*)=*?j1*)
then have *get-bit-s x p ?i1 ?j1* = *DIVIDED*
by (*simp add*: *a11 from-l-gt0 from-l-suc i1-len j1-len mp-alloc-stm4-pre-precond-f-froml*

same-bit-mp-alloc-stm4-pre-precond-divided)
then have *?thesis* **using** *a54 a02 a01* **by** *auto*
}
moreover{
assume *a02*:(*level m*)=*?i2* ∧ (*block m*)≥ *?j2* ∧ (*block m*)< *?j2*+*4*
then have *get-bit-s x p ?i2 ?j2* = *ALLOCATING*
using *i2-len j2-len a12 get-bit-x-l1-b4*[*OF - from-l-gt0 a12*, *of ?i2 ?j2*]
by (*metis* (*no-types*, *lifting*) *add-2-eq-Suc′ add-Suc-right add-lessD1 mult.commute*)
moreover {
assume *a07*:(*block m*)≠*?j2*
then have (*level m*) = *?i2* ∧ ((*block m*) = *Suc ?j2* ∨
(*block m*) = *Suc* (*Suc ?j2*) ∨ (*block m*) = *Suc* (*Suc* (*Suc?j2*)))
using *a02* **by** *auto*
then have *get-bit-s x p* (*level m*) (*block m*) = *FREE*
using *a02 i2-len j2-len*
get-bit-x-l1-b41[*OF - from-l-gt0*[*simplified from-l a13*]
a11[*simplified a13*[*THEN sym*] *from-l*] *a12*,*of level m*
block m]
by (*metis Suc-lessD mult.commute*)
}
ultimately have *?thesis* **using** *a54 a02 a01* **by** *fastforce*
} ultimately have *?thesis* **by** *auto*
} ultimately show *?thesis* **by** *auto*
qed

lemma *mp-alloc-stm4-inv-aux-vars3*:
assumes
a0:*inv Va* **and**
a1:*freeing-node Va t* = *None* **and**
a2:*p* ∈ *mem-pools Va* **and**
a3:*alloc-l Va t* < *int* (*n-levels* (*mem-pool-info Va p*)) **and**
a4:¬ *free-l Va t* < *OK* **and**
a5:*free-l Va t* ≤ *from-l Va t* **and**
a6:*allocating-node Va t* =
Some ⦇*pool* = *p*, *level* = *nat* (*from-l Va t*),
block = *block-num* (*mem-pool-info Va p*)
(*buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div*
4 ˆ *nat* (*from-l Va t*)))
(*lsizes Va t* ! *nat* (*from-l Va t*)),
data = *buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div*
4 ˆ *nat* (*from-l Va t*))⦈ **and**
a7:*block-num* (*mem-pool-info Va p*)

(*buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ^ *nat* (*from-l Va t*)))
(*lsizes Va t* ! *nat* (*from-l Va t*))
< *n-max* (*mem-pool-info Va p*) ∗ *4* ^ *nat* (*from-l Va t*) **and**
a8:*from-l Va t* < *alloc-l Va t* **and**
a9:*blk Va t* = *buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ^ *nat* (*from-l Va t*)) **and**
a10:(*x*, *mp-alloc-stm4-pre-precond-f Va t p*) ∈ *gvars-conf-stable* **and**
a11:∀ *pa*. *pa* ≠ *p* ⟶ *mem-pool-info x pa* = *mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *pa* **and**
a12:∀ *jj*. *jj* ≠ *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*) ⟶
levels (*mem-pool-info x p*) ! *jj* = *levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) ! *jj* **and**
a13:*bits* (*levels* (*mem-pool-info x p*) ! *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*)) =
list-updates-n
(*bits* (*levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*)))
(*Suc* (*bn* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* ∗ *4*)) *3 FREE* **and**
a14:*free-list* (*levels* (*mem-pool-info x p*) ! *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*)) =
inserts
(*map* (λ*ii*. *lsizes* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*) ∗
ii +
blk (*mp-alloc-stm4-pre-precond-f Va t p*) *t*)
[*Suc NULL..<4*])
(*free-list*
(*levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*))) **and**
a15:*lsizes x* = *lsizes* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
a16:*from-l x* = *from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
a17:*allocating-node x* = *allocating-node* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
a18:*allocating-node x t'* = *Some m*
shows *get-bit-s x* (*pool m*) (*level m*) (*block m*) = *ALLOCATING*
proof−
have *inv*:*inv-aux-vars Va* ∧ *inv-bitmap Va* ∧ *inv-mempool-info Va* ∧ *inv-bitmap-freelist Va*
using *a0* **unfolding** *inv-def* **by** *auto*
have *from-l-gt0*:*0* ≤ *from-l Va t* **using** *a5 a4* **by** *linarith*
have *inv-aux-va*:(∀ *t n*. *allocating-node Va t* = *Some n* ⟶
get-bit (*mem-pool-info Va*) (*pool n*) (*level n*) (*block n*) = *ALLOCATING*)
using *a0* **unfolding** *inv-def inv-aux-vars-def*
by *blast*
let *?i1*=(*nat* (*from-l Va t*)) **and**
?j1= (*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*))) **and**
?i2 = (*nat* (*from-l Va t* + *1*)) **and**
?j2 = (*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va*

t))∗*4*)
have *mem-pools*:*mem-pools x* = *mem-pools Va* **using** *mp-alloc-stm4-mempools*[*OF a10*] **by** *auto*
have *inv-mempool-info-mp Va p*
using *a2 mem-pools inv* **unfolding** *inv-mempool-info-def Let-def* **by** *auto*
note *inv-mempool*=*this*[*simplified Let-def*]
have *from-l*:*from-l x* = *from-l Va*
using *mp-alloc-stm4-froml*[*OF a16*] **by** *auto*
have *from-l-suc*:*nat* (*from-l Va t* + *1*) = *nat*(*from-l Va t*) + *1*
using *from-l-gt0* **by** *auto*
have *i1-len*:*?i1* < *length* (*levels* (*mem-pool-info Va p*))
using *a8 a2 a3 from-l-gt0 inv* **unfolding** *inv-mempool-info-def Let-def*
by *auto*
have *i2-len*:*?i2* < *length* (*levels* (*mem-pool-info Va p*))
using *a8 a2 a3 from-l-gt0 inv* **unfolding** *inv-mempool-info-def Let-def*
by *auto*
have *j1-len*:*?j1* < *length* (*bits* (*levels* (*mem-pool-info Va p*) ! *?i1*))
by (*metis a0 a2 a7 a9 i1-len inv-mempool-info-def invariant.inv-def*)
have *j2-len*:*Suc* (*Suc* (*Suc ?j2*)) < *length* (*bits* (*levels* (*mem-pool-info Va p*) ! *?i2*))
using *i1-len i2-len j1-len inv-mempool from-l-suc*
by *simp*
have *lsizes-x-va*:*lsizes x* = *lsizes Va* **using** *mp-alloc-stm4-pre-precond-f-lsz a15*
by *auto*
{**assume** *a00*:*t′* ≠ *t*
then have *allocating-node* (*mp-alloc-stm4-pre-precond-f Va t p*) *t′* = *allocating-node Va t′*
unfolding *mp-alloc-stm4-pre-precond-f-def* **by** *auto*
then have *eq-alloc*:*allocating-node Va t′* = *allocating-node x t′*
using *a17* **by** *auto*
then have *diff-t*:¬(*pool m* = *p* ∧ *level m* = *?i1* ∧ *block m* = *?j1*)
using *a00 a18 a6 inv* **unfolding** *inv-aux-vars-def*
by (*metis Mem-block.simps*(*1*) *Mem-block.simps*(*2*) *Mem-block.simps*(*3*) *a9*)
{
assume *pool m* ≠ *p*
then have *?thesis*
by (*metis a11 a18 eq-alloc inv-aux-va mp-alloc-stm4-pres-mpinfo*)
} **note** *not-pool-p-allocating* = *this*
moreover {
assume *a01*:*pool m* = *p*
have *bit-m-va-alloc*:*get-bit* (*mem-pool-info Va*) (*pool m*) (*level m*) (*block m*) = *ALLOCATING*
using *a18 eq-alloc inv-aux-va* **by** *presburger*
have *maxsz*:*max-sz* (*mem-pool-info x p*) = *max-sz* (*mem-pool-info Va p*)
using *mp-alloc-stm4-maxsz*[*OF a2 a10*] **by** *simp*
have *buf*:*buf* (*mem-pool-info x p*) = *buf* (*mem-pool-info Va p*)
using *mp-alloc-stm4-buf*[*OF a2 a10*] **by** *simp*
have *alloc-i1-j1*:*get-bit-s Va p ?i1 ?j1* = *ALLOCATING*
using *a6 a0 a9* **unfolding** *inv-aux-vars-def invariant.inv-def*

by (*metis* (*no-types*) *Mem-block.select-convs*(*1*) *Mem-block.select-convs*(*2*) *Mem-block.select-convs*(*3*))
have *nexisti2*:*noexist-bits* (*mem-pool-info Va p*) *?i2 ?j2*
using *a2 conjunct1*[*OF conjunct2*[*OF inv*], *simplified Let-def inv-bitmap-def*] *i1-len j1-len*
alloc-i1-j1 from-l-suc i2-len i1-len j1-len a1
by (*smt One-nat-def Suc-pred add.commute inv-mempool nat-add-left-cancel-less plus-1-eq-Suc*)
{ assume *a02*:(*level m* = *?i1* ∧ *block m* = *?j1*)
then have *?thesis* **using** *diff-t a01* **by** *auto*
}
moreover {
assume *a02*:¬(*level m* = *?i1* ∧ *block m* = *?j1*)
{ assume *a03*:¬(*level m* = *?i2* ∧ (*block m*)≥ *?j2* ∧ (*block m*)<*?j2* + *4*)
then have *eq-get-bit-i-j*:*get-bit-s x p* (*level m*) (*block m*) =
get-bit-s Va p (*level m*) (*block m*)
using *same-bit-mp-alloc-x-va*[*OF a12*[*simplified*
a16[*simplified mp-alloc-stm4-froml*[*OF a16*], *THEN sym*]] *a13*, *of level m block m*]
a01 a02 **by** *auto*
then have *?thesis* **using** *a01 a17 inv-aux-va not-pool-p-allocating*
a18 eq-alloc inv-aux-va **by** *force*
}
moreover {
assume *a03*:(*level m* = *?i2* ∧ (*block m*)≥ *?j2* ∧ (*block m*)<*?j2* + *4*)
then have *block m* = *?j2* ∨ *block m* = *?j2* + *1* ∨ *block m* = *?j2* + *2* ∨ *block m* = *?j2* + *3*
by *auto*
then have *?thesis* **using** *bit-m-va-alloc nexisti2 a01 a03* **by** *auto*
}
ultimately have *?thesis* **by** *fastforce*
} ultimately have *?thesis* **by** *fastforce*
} ultimately have *?thesis* **by** *auto*
}
moreover {
assume *t′*=*t*
then have (*pool m*) = *p* ∧ (*level m*) = *?i2* ∧ (*block m*) = *?j2*
by (*metis Mem-block.simps*(*1*) *Mem-block.simps*(*2*) *Mem-block.simps*(*3*) *a17 a18*
mp-alloc-stm4-pre-precond-f-allocating mult.commute option.sel)
then have *?thesis* **using** *get-bit-x-l1-b4*[*OF - from-l-gt0 a13 i2-len*] *j2-len*
by (*metis Suc-lessD mult.commute*)
}
ultimately show *?thesis* **by** *auto*
qed

lemma *mp-alloc-stm4-inv-aux-vars4*:
assumes
a0:*inv Va* **and**

$a1$:*freeing-node Va t = None* **and**
$a2$:*p* ∈ *mem-pools Va* **and**
$a3$:∀ *ii*<*length* (*lsizes Va t*). *lsizes Va t* ! *ii* = *ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *ii* **and**
$a4$:*alloc-l Va t* < *int* (*n-levels* (*mem-pool-info Va p*)) **and**
$a5$:¬ *free-l Va t* < *OK* **and**
$a6$:*free-l Va t* ≤ *from-l Va t* **and**
$a7$:*allocating-node Va t* =
Some (|*pool* = *p*, *level* = *nat* (*from-l Va t*),
block = *block-num* (*mem-pool-info Va p*)
(*buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ^ *nat* (*from-l Va t*)))
(*lsizes Va t* ! *nat* (*from-l Va t*)),
data = *buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ^ *nat* (*from-l Va t*))|) **and**
$a8$:*block-num* (*mem-pool-info Va p*)
(*buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ^ *nat* (*from-l Va t*)))
(*lsizes Va t* ! *nat* (*from-l Va t*))
< *n-max* (*mem-pool-info Va p*) ∗ *4* ^ *nat* (*from-l Va t*) **and**
$a9$:*from-l Va t* < *alloc-l Va t* **and**
$a10$:*blk Va t* = *buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ^ *nat* (*from-l Va t*)) **and**
$a11$:*alloc-l Va t* = *int* (*length* (*lsizes Va t*)) − *1* ∧ *length* (*lsizes Va t*) = *n-levels* (*mem-pool-info Va p*) ∨
alloc-l Va t = *int* (*length* (*lsizes Va t*)) − *2* ∧ *lsizes Va t* ! *nat* (*alloc-l Va t* + *1*) < *sz* **and**
$a12$:(*x*, *mp-alloc-stm4-pre-precond-f Va t p*) ∈ *gvars-conf-stable* **and**
$a13$:∀ *pa*. *pa* ≠ *p* ⟶ *mem-pool-info x pa* = *mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *pa* **and**
$a14$:∀ *jj*. *jj* ≠ *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*) ⟶
levels (*mem-pool-info x p*) ! *jj* = *levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) ! *jj* **and**
$a15$:*bits* (*levels* (*mem-pool-info x p*) ! *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*)) =
list-updates-n
(*bits* (*levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*)))
(*Suc* (*bn* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* ∗ *4*)) *3 FREE* **and**
$a16$:*free-list* (*levels* (*mem-pool-info x p*) ! *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*)) =
inserts
(*map* (λ*ii*. *lsizes* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*) ∗
ii +
blk (*mp-alloc-stm4-pre-precond-f Va t p*) *t*)
[*Suc NULL*..<*4*])
(*free-list*
(*levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) !

nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*))) **and**
a17:*lsizes x* = *lsizes* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
a18:*from-l x* = *from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
a19:*blk x* = *blk* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
a20:*allocating-node x* = *allocating-node* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
a21:*get-bit-s x* (*pool m*) (*level m*) (*block m*) = *ALLOCATING* ∧ *mem-block-addr-valid x m*
shows (∃ *t*. *allocating-node x t* = *Some m*)
proof−
have *inv*:*inv-aux-vars Va* ∧ *inv-bitmap Va* ∧ *inv-mempool-info Va* ∧ *inv-bitmap-freelist Va*
using *a0* **unfolding** *inv-def* **by** *auto*
have *from-l-gt0*:*0* ≤ *from-l Va t* **using** *a6 a5* **by** *linarith*
have *block-valid-va*:*mem-block-addr-valid Va m*
using *a2 a12 a21 mp-alloc-stm4-buf mp-alloc-stm4-maxsz*
unfolding *mem-block-addr-valid-def* **by** *auto*
have *data-m*:*data m* =
buf (*mem-pool-info x* (*pool m*)) + (*block m*) ∗ ((*max-sz* (*mem-pool-info x* (*pool m*))) *div* (*4* ˆ (*level m*)))
using *a21* **unfolding** *mem-block-addr-valid-def* **by** *auto*
have *inv-aux-va*:(∀ *n*. *get-bit* (*mem-pool-info Va*) (*pool n*) (*level n*) (*block n*) = *ALLOCATING* ∧ *mem-block-addr-valid Va n*
⟶ (∃ *t*. *allocating-node Va t* = *Some n*))
using *a0* **unfolding** *inv-def inv-aux-vars-def*
by *blast*
{ assume *a00*:(*pool m*) ≠ *p*
then obtain *t*′ **where** *allocating-node Va t*′ = *Some m* **using** *inv-aux-va*
by (*metis a13 a21 block-valid-va mp-alloc-stm4-pres-mpinfo*)
moreover have *t*′≠*t* **using** *a2* **unfolding** *inv-def inv-aux-vars-def*
using *a00 a7 calculation* **by** *auto*
then have *allocating-node* (*mp-alloc-stm4-pre-precond-f Va t p*) *t*′ = *allocating-node Va t*′
unfolding *mp-alloc-stm4-pre-precond-f-def* **by** *auto*
then have *eq-alloc*:*allocating-node Va t*′ = *allocating-node x t*′
using *a20* **by** *auto*
ultimately have *?thesis* **by** *auto*
}
moreover{
assume *a01*:*pool m* = *p*
let *?i1*=(*nat* (*from-l Va t*)) **and**
?j1= (*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*))) **and**
?i2 = (*nat* (*from-l Va t* + *1*)) **and**
?j2 = (*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*))∗*4*)
have *mem-pools*:*mem-pools x* = *mem-pools Va* **using** *mp-alloc-stm4-mempools*[*OF a12*] **by** *auto*
have *inv-mempool-info-mp Va p*
using *a2 mem-pools inv* **unfolding** *inv-mempool-info-def Let-def* **by** *auto*

note *inv-mempool=this*[*simplified Let-def*]
have *from-l*:*from-l x = from-l Va*
  using *mp-alloc-stm4-froml*[*OF a18*] **by** *auto*
have *from-l-suc*:*nat* (*from-l Va t + 1*) = *nat*(*from-l Va t*) + *1*
  using *from-l-gt0* **by** *auto*
have *i1-len*:*?i1 < length* (*levels* (*mem-pool-info Va p*))
  using *a9 a2 a4 from-l-gt0 inv* **unfolding** *inv-mempool-info-def Let-def*
   by *auto*
have *i2-len*:*?i2 < length* (*levels* (*mem-pool-info Va p*))
  using *a9 a2 a4 from-l-gt0 inv* **unfolding** *inv-mempool-info-def Let-def*
  by *auto*
have *j1-len*:*?j1 < length* (*bits* (*levels* (*mem-pool-info Va p*) ! *?i1*))
  by (*metis a0 a2 a8 a10 i1-len inv-mempool-info-def invariant.inv-def*)
have *j2-len*:*Suc* (*Suc* (*Suc ?j2*)) < *length* (*bits* (*levels* (*mem-pool-info Va p*) ! *?i2*))
  using *i1-len i2-len j1-len inv-mempool from-l-suc*
  by *simp*
have *lsizes-x-va*:*lsizes x = lsizes Va*
  by (*simp add*: *a17 mp-alloc-stm4-pre-precond-f-lsz*)
have *buf*:*buf* (*mem-pool-info x p*) = *buf* (*mem-pool-info Va p*)
  using *mp-alloc-stm4-buf*[*OF a2 a12*] **by** *simp*
have *maxsz*:*max-sz* (*mem-pool-info x p*) = *max-sz* (*mem-pool-info Va p*)
  using *mp-alloc-stm4-maxsz*[*OF a2 a12*] **by** *simp*
{ assume *a02*:¬(((*level m*)=*?i1* ∧ (*block m*)=*?j1*) ∨
      ((*level m*)=*?i2* ∧ (*block m*)≥ *?j2* ∧ (*block m*)< *?j2+4*))
  then have *a020′*:¬((*level m*)=*?i1* ∧ (*block m*)=*?j1*) **and**
      a021′: ¬((*level m*)=*?i2* ∧ (*block m*)≥ *?j2* ∧ (*block m*)<*?j2 + 4*)
   by *force+*
   then have *eq-get-bit-i-j*:*get-bit-s x p* (*level m*) (*block m*) = *get-bit-s Va p* (*level m*) (*block m*)
    using *same-bit-mp-alloc-x-va*[*OF a14*[*simplified a18*[*simplified mp-alloc-stm4-froml*[*OF a18*], *THEN sym*]] *a15*, *of level m block m*]
    using *a020′ a021′* **by** *auto*
   then have *get-bit-s Va* (*pool m*) (*level m*) (*block m*) = *ALLOCATING* ∧ *mem-block-addr-valid Va m*
    using *a01 a21 block-valid-va* **by** *auto*
  then obtain *t′* **where** *allocating-node Va t′* = *Some m* **using** *inv-aux-va* **by** *auto*
  moreover have *t′≠t* **using** *a02 a7 a10 calculation* **by** *auto*
 then have *allocating-node* (*mp-alloc-stm4-pre-precond-f Va t p*) *t′* = *allocating-node Va t′*
    unfolding *mp-alloc-stm4-pre-precond-f-def* **by** *auto*
  then have *eq-alloc*:*allocating-node Va t′* = *allocating-node x t′*
    using *a20* **by** *auto*
  ultimately have *?thesis* **by** *auto*
}
moreover{
  assume *a02*:((*level m*)=*?i1* ∧ (*block m*)=*?j1*)

```
    then have get-bit-s x p ?i1 ?j1 = DIVIDED
    by (simp add: a14 from-l-gt0 from-l-suc i1-len j1-len mp-alloc-stm4-pre-precond-f-froml

            same-bit-mp-alloc-stm4-pre-precond-divided)
    then have ?thesis using a21 a02 a01 by auto
  }
  moreover{
    assume a02:(level m)=?i2 ∧ (block m)≥ ?j2 ∧ (block m)< ?j2+4
    then have block-n:(block-num (mem-pool-info Va p)
          (blk Va t) (lsizes Va t ! nat (from-l Va t))) = n
    proof−
      have lsizes Va t ! nat (from-l Va t) =
           ALIGN4 (max-sz (mem-pool-info Va p)) div 4 ^
            (nat (from-l Va t))
        using a3 lsizes-x-va  a5 a6 a9 a11 a5 from-l by auto
      thus ?thesis using block-n inv a2  a10   a4 a9 from-l-gt0
        by blast
    qed
    then have get-bit-s x p ?i2 ?j2 = ALLOCATING
      using i2-len j2-len  a15 get-bit-x-l1-b4[OF - from-l-gt0 a15, of ?i2 ?j2]
    by (metis (no-types, lifting) add-2-eq-Suc' add-Suc-right add-lessD1 mult.commute)

    { assume a03:block m = ?j2
      then have m = (|pool = p, level = ?i2, block = ?j2,
                    data = buf (mem-pool-info x p) +
                        ?j2 ∗ (max-sz (mem-pool-info x p) div 4 ^ ?i2)
                  |) using data-m a03 a02 a01 by auto
      moreover have  blk x t = buf (mem-pool-info x p) +
                        ?j1 ∗ ((max-sz (mem-pool-info x p) div 4 ^ ?i1))
          using a10[simplified buf[THEN sym] maxsz[THEN sym]] block-n a19
mp-alloc-stm4-blk
        by metis
      then have allocating-node x t = Some (|pool = p, level = ?i2, block = ?j2,
                               data = buf (mem-pool-info x p) +
                                 ?j1 ∗  (max-sz (mem-pool-info x p) div 4 ^
?i1)|)
        using a20 a19 mp-alloc-stm4-blk mp-alloc-stm4-pre-precond-f-allocating
        by (metis mult.commute)
       ultimately have  ?thesis using buf maxsz next-level-addr-eq unfolding
addr-def
        by (metis from-l-suc i2-len inv-mempool)
    }
    moreover {
      assume a07:(block m)≠?j2
      then have  (level m) = ?i2 ∧ ((block m) = Suc ?j2 ∨
            (block m) = Suc (Suc ?j2) ∨ (block m) = Suc (Suc (Suc?j2)))
        using a02 by auto
      then have  get-bit-s x p (level m) (block m) = FREE
        using a02 i2-len j2-len
```

get-bit-x-l1-b41[*OF* - *from-l-gt0*[*simplified from-l a18*]
a14[*simplified a18*[*THEN sym*] *from-l*] *a15*,*of level m*
block m]
by (*metis Suc-lessD mult.commute*)
then have *?thesis* **using** *a21 a01* **by** *auto*
}
ultimately have *?thesis* **by** *auto*
}
ultimately have *?thesis* **by** *auto*
} **ultimately show** *?thesis* **by** *auto*
qed

lemma *mp-alloc-stm4-inv-aux-vars5*:
assumes
a0:*inv Va* **and**
a1:*freeing-node x* = *freeing-node* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
a2: $t1 \neq t2$ ∧ *freeing-node x t1* = *Some n1* ∧ *freeing-node x t2* = *Some n2*
shows ¬ (*pool n1* = *pool n2* ∧ *level n1* = *level n2* ∧ *block n1* = *block n2*)
proof−
have $t1 \neq t2$ ∧ *freeing-node Va t1* = *Some n1* ∧ *freeing-node Va t2* = *Some n2*
using *a1 a2*
by (*metis mp-alloc-stm4-pre-precond-f-def-frnode*)
then have ¬ (*pool n1* = *pool n2* ∧ *level n1* = *level n2* ∧ *block n1* = *block n2*)
using *a0* **unfolding** *inv-def inv-aux-vars-def* **by** *auto*
then show *?thesis*
by *blast*
qed

lemma *mp-alloc-stm4-inv-aux-vars6*:
assumes
a0:*inv Va* **and**
a1:*freeing-node Va t* = *None* **and**
a2:*p* ∈ *mem-pools Va* **and**
a3:*length* (*lsizes Va t*) ≤ *n-levels* (*mem-pool-info Va p*) **and**
a4:*alloc-l Va t* < *int* (*n-levels* (*mem-pool-info Va p*)) **and**
a5:¬ *free-l Va t* < *OK* **and**
a6:*free-l Va t* ≤ *from-l Va t* **and**
a7:*allocating-node Va t* =
Some (|*pool* = *p*, *level* = *nat* (*from-l Va t*),
block = *block-num* (*mem-pool-info Va p*)
(*buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div*
4 ˆ *nat* (*from-l Va t*)))
(*lsizes Va t* ! *nat* (*from-l Va t*)),
data = *buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div*
4 ˆ *nat* (*from-l Va t*))|) **and**
a8:*block-num* (*mem-pool-info Va p*)
(*buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ˆ *nat*
(*from-l Va t*)))
(*lsizes Va t* ! *nat* (*from-l Va t*))

< *n-max* (*mem-pool-info Va p*) * *4* ^ *nat* (*from-l Va t*) **and**
a9:*from-l Va t* < *alloc-l Va t* **and**
a10:*blk Va t* = *buf* (*mem-pool-info Va p*) + *n* * (*max-sz* (*mem-pool-info Va p*) *div 4* ^ *nat* (*from-l Va t*)) **and**
a11:(*x*, *mp-alloc-stm4-pre-precond-f Va t p*) ∈ *gvars-conf-stable* **and**
a12:∀ *pa*. *pa* ≠ *p* ⟶ *mem-pool-info x pa* = *mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *pa* **and**
a13:∀ *jj*. *jj* ≠ *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*) ⟶
levels (*mem-pool-info x p*) ! *jj* = *levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) ! *jj* **and**
a14:*bits* (*levels* (*mem-pool-info x p*) ! *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*)) =
list-updates-n
(*bits* (*levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*)))
(*Suc* (*bn* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* * *4*)) *3 FREE* **and**
a15:*free-list* (*levels* (*mem-pool-info x p*) ! *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*)) =
inserts
(*map* (λ*ii*. *lsizes* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*) *
ii +
blk (*mp-alloc-stm4-pre-precond-f Va t p*) *t*)
[*Suc NULL*..<*4*])
(*free-list*
(*levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*))) **and**
a16:*lsizes x* = *lsizes* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
a17:*from-l x* = *from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
a18:*allocating-node x* = *allocating-node* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
a19:*t1* ≠ *t2* ∧ *allocating-node x t1* = *Some n1* ∧ *allocating-node x t2* = *Some n2*
shows ¬ (*pool n1* = *pool n2* ∧ *level n1* = *level n2* ∧ *block n1* = *block n2*)
proof−
have *inv*:*inv-aux-vars Va* ∧ *inv-bitmap Va* ∧ *inv-mempool-info Va* ∧ *inv-bitmap-freelist Va*
using *a0* **unfolding** *inv-def* **by** *auto*
have *from-l-gt0*:*0* ≤ *from-l Va t* **using** *a6 a5* **by** *linarith*
let *?i1*=(*nat* (*from-l Va t*)) **and**
?j1= (*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*))) **and**
?i2 = (*nat* (*from-l Va t* + *1*)) **and**
?j2 = (*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*))**4*)
have *mem-pools*:*mem-pools x* = *mem-pools Va* **using** *mp-alloc-stm4-mempools*[*OF a11*] **by** *auto*
have *inv-mempool-info-mp Va p*
using *a2 mem-pools inv* **unfolding** *inv-mempool-info-def Let-def* **by** *auto*
note *inv-mempool*=*this*[*simplified Let-def*]
have *from-l*:*from-l x* = *from-l Va*

using *mp-alloc-stm4-froml*[*OF a17*] **by** *auto*
have *from-l-suc*:*nat* (*from-l Va t* + *1*) = *nat*(*from-l Va t*) + *1*
using *from-l-gt0* **by** *auto*
have *i1-len*:*?i1* < *length* (*levels* (*mem-pool-info Va p*))
using *a9 a2 a4 from-l-gt0 inv* **unfolding** *inv-mempool-info-def Let-def*
by *auto*
have *i2-len*:*?i2* < *length* (*levels* (*mem-pool-info Va p*))
using *a9 a2 a4 from-l-gt0 inv* **unfolding** *inv-mempool-info-def Let-def*
by *auto*
have *j1-len*:*?j1* < *length* (*bits* (*levels* (*mem-pool-info Va p*) ! *?i1*))
by (*metis a0 a2 a8 a10 i1-len inv-mempool-info-def invariant.inv-def*)
have *j2-len*:*Suc* (*Suc* (*Suc ?j2*)) < *length* (*bits* (*levels* (*mem-pool-info Va p*) ! *?i2*))
using *i1-len i2-len j1-len inv-mempool from-l-suc*
by *simp*
have *lsizes-x-va*:*lsizes x* = *lsizes Va* **using** *mp-alloc-stm4-pre-precond-f-lsz a16*
by *auto*
have *maxsz*:*max-sz* (*mem-pool-info x p*) = *max-sz* (*mem-pool-info Va p*)
using *mp-alloc-stm4-maxsz*[*OF a2 a11*] **by** *simp*
have *buf*:*buf* (*mem-pool-info x p*) = *buf* (*mem-pool-info Va p*)
using *mp-alloc-stm4-buf*[*OF a2 a11*] **by** *simp*
have *alloc-i1-j1*:*get-bit-s Va p ?i1 ?j1* = *ALLOCATING*
using *a7 a0 a10* **unfolding** *inv-aux-vars-def invariant.inv-def*
by (*metis* (*no-types*) *Mem-block.select-convs*(*1*) *Mem-block.select-convs*(*2*) *Mem-block.select-convs*(*3*))

have *nexisti2*:*noexist-bits* (*mem-pool-info Va p*) *?i2 ?j2*
using *a2 conjunct1*[*OF conjunct2*[*OF inv*], *simplified Let-def inv-bitmap-def*] *i1-len j1-len*
alloc-i1-j1 from-l-suc i2-len i1-len j1-len a1
by (*smt One-nat-def Suc-pred add.commute inv-mempool nat-add-left-cancel-less plus-1-eq-Suc*)
{ assume *t*≠*t1* **and** *t*≠*t2*
then have *?thesis*
using *a0 a19 a18 inv-aux-vars-def*
unfolding *mp-alloc-stm4-pre-precond-f-def invariant.inv-def* **by** *force*
}
moreover {
assume *a00*:*t*=*t1*
then have *t2*≠*t* **using** *a19* **by** *auto*
then have *t2-same-allocating-node-Va*:*allocating-node Va t2* = *Some n2*
using *a0 a19 a18*
unfolding *mp-alloc-stm4-pre-precond-f-def invariant.inv-def inv-aux-vars-def*
by *force*
then have *get-bit-n2*:*get-bit-s Va* (*pool n2*) (*level n2*) (*block n2*) = *ALLOCATING*
using *a0 a19 a18 inv-aux-vars-def*
unfolding *mp-alloc-stm4-pre-precond-f-def invariant.inv-def* **by** *force*
have ¬ (*pool n1* = *pool n2* ∧ *level n1* = *level n2* ∧ *block n1* = *block n2*) =
(*pool n1* = *pool n2* ⟶ ¬(*level n1* = *level n2* ∧ *block n1* = *block n2*))

by *auto*
moreover {
assume *a02*:*pool n1* = *pool n2*
have *n1* = (|*pool* = *p*, *level* = *?i2*, *block* = *?j2*,
data = *blk Va t*
|) **using** *a19*
by (*simp add*: *a00 a18 mp-alloc-stm4-pre-precond-f-allocating mult.commute*)

then have ¬(*level n1* = *level n2* ∧ *block n1* = *block n2*)
using *get-bit-n2 a02 nexisti2* **by** *auto*
}
then have *?thesis* **by** *auto*
}
moreover {
assume *a00*:*t*=*t2*
then have *t1*≠*t* **using** *a19* **by** *auto*
then have *t2-same-allocating-node-Va*:*allocating-node Va t1* = *Some n1*
using *a0 a19 a18 inv-aux-vars-def*
unfolding *mp-alloc-stm4-pre-precond-f-def invariant.inv-def* **by** *force*
then have *get-bit-n2*:*get-bit-s Va* (*pool n1*) (*level n1*) (*block n1*) = *ALLOCAT-ING*
using *a0 a19 a18 inv-aux-vars-def*
unfolding *mp-alloc-stm4-pre-precond-f-def invariant.inv-def* **by** *force*
have ¬ (*pool n1* = *pool n2* ∧ *level n1* = *level n2* ∧ *block n1* = *block n2*) =
(*pool n1* = *pool n2* ⟶ ¬(*level n1* = *level n2* ∧ *block n1* = *block n2*))
by *auto*
moreover {
assume *a02*:*pool n1* = *pool n2*
have *n2* = (|*pool* = *p*, *level* = *?i2*, *block* = *?j2*,
data = *blk Va t*
|) **using** *a19*
by (*simp add*: *a00 a18 mp-alloc-stm4-pre-precond-f-allocating mult.commute*)

then have ¬(*level n1* = *level n2* ∧ *block n1* = *block n2*)
using *get-bit-n2 a02 nexisti2* **by** *auto*
}
then have *?thesis* **by** *auto*
}
ultimately show *?thesis* **by** *auto*
qed

lemma *mp-alloc-stm4-inv-aux-vars7*:
assumes
a0:*inv Va* **and**
a1:*freeing-node Va t* = *None* **and**
a2:*p* ∈ *mem-pools Va* **and**
a3:∀ *ii*<*length* (*lsizes Va t*). *lsizes Va t* ! *ii* = *ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ^ *ii* **and**
a4:*length* (*lsizes Va t*) ≤ *n-levels* (*mem-pool-info Va p*) **and**

$a5$:*alloc-l Va t* $<$ *int* (*n-levels* (*mem-pool-info Va p*)) **and**
$a6$:$\neg$ *free-l Va t* $<$ *OK* **and**
$a7$:*free-l Va t* $\leq$ *from-l Va t* **and**
$a8$:*allocating-node Va t* =
Some (|*pool* = *p*, *level* = *nat* (*from-l Va t*),
block = *block-num* (*mem-pool-info Va p*)
(*buf* (*mem-pool-info Va p*) + *n* $*$ (*max-sz* (*mem-pool-info Va p*) *div* *4* ^ *nat* (*from-l Va t*)))
(*lsizes Va t* ! *nat* (*from-l Va t*)),
data = *buf* (*mem-pool-info Va p*) + *n* $*$ (*max-sz* (*mem-pool-info Va p*) *div* *4* ^ *nat* (*from-l Va t*))|) **and**
$a9$:*n* = *block-num* (*mem-pool-info Va p*)
(*buf* (*mem-pool-info Va p*) + *n* $*$ (*max-sz* (*mem-pool-info Va p*) *div* *4* ^ *nat* (*from-l Va t*)))
(*lsizes Va t* ! *nat* (*from-l Va t*)) $\vee$
max-sz (*mem-pool-info Va p*) *div* *4* ^ *nat* (*from-l Va t*) = *NULL* **and**
$a10$:*block-num* (*mem-pool-info Va p*)
(*buf* (*mem-pool-info Va p*) + *n* $*$ (*max-sz* (*mem-pool-info Va p*) *div* *4* ^ *nat* (*from-l Va t*)))
(*lsizes Va t* ! *nat* (*from-l Va t*))
$<$ *n-max* (*mem-pool-info Va p*) $*$ *4* ^ *nat* (*from-l Va t*) **and**
$a11$:*from-l Va t* $<$ *alloc-l Va t* **and**
$a12$:*blk Va t* = *buf* (*mem-pool-info Va p*) + *n* $*$ (*max-sz* (*mem-pool-info Va p*) *div* *4* ^ *nat* (*from-l Va t*)) **and**
$a23$:(*x*, *mp-alloc-stm4-pre-precond-f Va t p*) $\in$ *gvars-conf-stable* **and**
$a14$:$\forall$ *pa*. *pa* $\neq$ *p* $\longrightarrow$ *mem-pool-info x pa* = *mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *pa* **and**
$a15$:$\forall$ *jj*. *jj* $\neq$ *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*) $\longrightarrow$
levels (*mem-pool-info x p*) ! *jj* = *levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) ! *jj* **and**
$a16$:*bits* (*levels* (*mem-pool-info x p*) ! *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*)) =
list-updates-n
(*bits* (*levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*)))
(*Suc* (*bn* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* $*$ *4*)) *3 FREE* **and**
$a17$:*free-list* (*levels* (*mem-pool-info x p*) ! *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*)) =
inserts
(*map* (λ*ii*. *lsizes* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*) $*$
ii +
blk (*mp-alloc-stm4-pre-precond-f Va t p*) *t*)
[*Suc NULL*..<*4*])
(*free-list*
(*levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*))) **and**
$a18$:*lsizes x* = *lsizes* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
$a19$:*from-l x* = *from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**

a20:*freeing-node x* = *freeing-node* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
a21:*allocating-node x* = *allocating-node* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
a22:*allocating-node x t1* = *Some n1* ∧ *freeing-node x t2* = *Some n2*
shows ¬ (*pool n1* = *pool n2* ∧ *level n1* = *level n2* ∧ *block n1* = *block n2*)
proof−
{**assume** *pool n1* = *pool n2*
moreover have *get-bit-s x* (*pool n1*) (*level n1*) (*block n1*) = *ALLOCATING*
using *mp-alloc-stm4-inv-aux-vars3 assms* **by** *blast*
moreover have *get-bit-s x* (*pool n2*) (*level n2*) (*block n2*) = *FREEING*
using *mp-alloc-stm4-inv-aux-vars1 assms* **by** *blast*
ultimately have *?thesis* **by** *auto*
} **thus** *?thesis* **by** *auto*
qed

lemma *mp-alloc-stm4-inv-aux-vars*:
assumes
a0:*inv Va* **and**
a1:*freeing-node Va t* = *None* **and**
a2:*p* ∈ *mem-pools Va* **and**
a3:∀ *ii*<*length* (*lsizes Va t*). *lsizes Va t* ! *ii* = *ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *ii* **and**
a4:*length* (*lsizes Va t*) ≤ *n-levels* (*mem-pool-info Va p*) **and**
a5:*alloc-l Va t* < *int* (*n-levels* (*mem-pool-info Va p*)) **and**
a6:¬ *free-l Va t* < *OK* **and**
a7:*free-l Va t* ≤ *from-l Va t* **and**
a8:*allocating-node Va t* =
Some (|*pool* = *p*, *level* = *nat* (*from-l Va t*),
block = *block-num* (*mem-pool-info Va p*)
(*buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div* *4* ˆ *nat* (*from-l Va t*)))
(*lsizes Va t* ! *nat* (*from-l Va t*)),
data = *buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div* *4* ˆ *nat* (*from-l Va t*))|) **and**
a9:*n* = *block-num* (*mem-pool-info Va p*)
(*buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ˆ *nat* (*from-l Va t*)))
(*lsizes Va t* ! *nat* (*from-l Va t*)) ∨
max-sz (*mem-pool-info Va p*) *div 4* ˆ *nat* (*from-l Va t*) = *NULL* **and**
a10:*block-num* (*mem-pool-info Va p*)
(*buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ˆ *nat* (*from-l Va t*)))
(*lsizes Va t* ! *nat* (*from-l Va t*))
< *n-max* (*mem-pool-info Va p*) ∗ *4* ˆ *nat* (*from-l Va t*) **and**
a11:*from-l Va t* < *alloc-l Va t* **and**
a12:*blk Va t* = *buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ˆ *nat* (*from-l Va t*)) **and**
a13:(*x*, *mp-alloc-stm4-pre-precond-f Va t p*) ∈ *gvars-conf-stable* **and**
a14:∀ *pa*. *pa* ≠ *p* ⟶ *mem-pool-info x pa* = *mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *pa* **and**

a15:∀ *jj*. *jj* ≠ *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*) ⟶
levels (*mem-pool-info x p*) ! *jj* = *levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) ! *jj* **and**
a16:*bits* (*levels* (*mem-pool-info x p*) ! *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*)) =
list-updates-n
(*bits* (*levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*)))
(*Suc* (*bn* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* ∗ *4*)) *3 FREE* **and**
a17:*free-list* (*levels* (*mem-pool-info x p*) ! *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*)) =
inserts
(*map* (λ*ii*. *lsizes* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*) ∗
ii +
blk (*mp-alloc-stm4-pre-precond-f Va t p*) *t*)
[*Suc NULL*..<*4*])
(*free-list*
(*levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*))) **and**
a18:*lsizes x* = *lsizes* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
a19:*from-l x* = *from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
a20:*freeing-node x* = *freeing-node* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
a21:*allocating-node x* = *allocating-node* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
a22:*alloc-l Va t* = *int* (*length* (*lsizes Va t*)) − *1* ∧ *length* (*lsizes Va t*) = *n-levels* (*mem-pool-info Va p*) ∨
alloc-l Va t = *int* (*length* (*lsizes Va t*)) − *2* ∧ *lsizes Va t* ! *nat* (*alloc-l Va t* + *1*) < *sz* **and**
a23:*blk x* = *blk* (*mp-alloc-stm4-pre-precond-f Va t p*)
shows *inv-aux-vars x* **unfolding** *inv-aux-vars-def*
using
mp-alloc-stm4-inv-aux-vars1[*OF assms*(*1*−*9*,*11*−*22*)]
mp-alloc-stm4-inv-aux-vars2[*OF assms*(*1*−*3*,*6*−*8*,*11*−*17*,*20*−*21*)]
mp-alloc-stm4-inv-aux-vars3[*OF assms*(*1*−*3*,*6*−*7*,*8*−*9*,*11*−*20*,*22*)]
mp-alloc-stm4-inv-aux-vars4[*OF assms*(*1*−*4*,*6*−*9*,*11*−*13*,*23*,*14*−*20*,*24*,*22*)]
mp-alloc-stm4-inv-aux-vars5[*OF assms*(*1*,*21*)]
mp-alloc-stm4-inv-aux-vars6[*OF assms*(*1*−*3*,*5*−*9*,*11*−*20*,*22*)] *mp-alloc-stm4-inv-aux-vars7*[*OF assms*(*1*−*22*)]
by *auto*

lemma *mp-alloc-stm4-inv-bitmap0*:
assumes
a0:*inv Va* **and**
a1:*freeing-node Va t* = *None* **and**
a2:*p* ∈ *mem-pools Va* **and**
a3:∀ *ii*<*length* (*lsizes Va t*). *lsizes Va t* ! *ii* = *ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *ii* **and**
a4:*length* (*lsizes Va t*) ≤ *n-levels* (*mem-pool-info Va p*) **and**
a5:*alloc-l Va t* < *int* (*n-levels* (*mem-pool-info Va p*)) **and**

$a6$:$\neg$ *free-l Va t* $<$ *OK* **and**
$a7$:*free-l Va t* $\leq$ *from-l Va t* **and**
$a8$:*allocating-node Va t* $=$
Some (|*pool* $=$ *p*, *level* $=$ *nat* (*from-l Va t*),
block $=$ *block-num* (*mem-pool-info Va p*)
(*buf* (*mem-pool-info Va p*) $+$ *n* $*$ (*max-sz* (*mem-pool-info Va p*) *div* *4* ^ *nat* (*from-l Va t*)))
(*lsizes Va t* ! *nat* (*from-l Va t*)),
data $=$ *buf* (*mem-pool-info Va p*) $+$ *n* $*$ (*max-sz* (*mem-pool-info Va p*) *div* *4* ^ *nat* (*from-l Va t*))|) **and**
$a9$:*n* $=$ *block-num* (*mem-pool-info Va p*)
(*buf* (*mem-pool-info Va p*) $+$ *n* $*$ (*max-sz* (*mem-pool-info Va p*) *div 4* ^ *nat* (*from-l Va t*)))
(*lsizes Va t* ! *nat* (*from-l Va t*)) $\vee$
max-sz (*mem-pool-info Va p*) *div 4* ^ *nat* (*from-l Va t*) $=$ *NULL* **and**
$a10$:*block-num* (*mem-pool-info Va p*)
(*buf* (*mem-pool-info Va p*) $+$ *n* $*$ (*max-sz* (*mem-pool-info Va p*) *div 4* ^ *nat* (*from-l Va t*)))
(*lsizes Va t* ! *nat* (*from-l Va t*))
$<$ *n-max* (*mem-pool-info Va p*) $*$ *4* ^ *nat* (*from-l Va t*) **and**
$a11$:*from-l Va t* $<$ *alloc-l Va t* **and**
$a12$:*blk Va t* $=$ *buf* (*mem-pool-info Va p*) $+$ *n* $*$ (*max-sz* (*mem-pool-info Va p*) *div 4* ^ *nat* (*from-l Va t*)) **and**
$a13$:(*x*, *mp-alloc-stm4-pre-precond-f Va t p*) $\in$ *gvars-conf-stable* **and**
$a14$:$\forall$ *pa*. *pa* $\neq$ *p* $\longrightarrow$ *mem-pool-info x pa* $=$ *mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *pa* **and**
$a15$:$\forall$ *jj*. *jj* $\neq$ *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* $+$ *1*) $\longrightarrow$
levels (*mem-pool-info x p*) ! *jj* $=$ *levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) ! *jj* **and**
$a16$:*bits* (*levels* (*mem-pool-info x p*) ! *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* $+$ *1*)) $=$
list-updates-n
(*bits* (*levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* $+$ *1*)))
(*Suc* (*bn* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* $*$ *4*)) *3 FREE* **and**
$a17$:*free-list* (*levels* (*mem-pool-info x p*) ! *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* $+$ *1*)) $=$
inserts
(*map* (λ*ii*. *lsizes* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* $+$ *1*) $*$
ii $+$
blk (*mp-alloc-stm4-pre-precond-f Va t p*) *t*)
[*Suc NULL*..<*4*])
(*free-list*
(*levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* $+$ *1*))) **and**
$a18$:*lsizes x* $=$ *lsizes* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
$a19$:*from-l x* $=$ *from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
$a20$:*freeing-node x* $=$ *freeing-node* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**

a21:*allocating-node x* = *allocating-node* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
a22:*alloc-l Va t* = *int* (*length* (*lsizes Va t*)) − *1* ∧ *length* (*lsizes Va t*) = *n-levels* (*mem-pool-info Va p*) ∨
alloc-l Va t = *int* (*length* (*lsizes Va t*)) − *2* ∧ *lsizes Va t* ! *nat* (*alloc-l Va t* + *1*) < *sz* **and**
a23:*blk x* = *blk* (*mp-alloc-stm4-pre-precond-f Va t p*)
shows *inv-bitmap0 x*
proof(*simp add*: *inv-bitmap0-def Let-def*)
{ fix *p′ j*
assume *a00*:*p′* ∈ *mem-pools x*
assume *a01*:*j*<*length* (*bits* (*levels* (*mem-pool-info x p′*) ! *NULL*))
{ assume *p′*≠ *p*
then have *get-bit-s x p′ NULL j* ≠ *NOEXIST*
by (*metis a0 a00 a01 a13 a14 inv-bitmap0-def*
invariant.inv-def mp-alloc-stm4-mempools mp-alloc-stm4-pres-mpinfo)
}
moreover { assume *a02*:*p′* = *p*
let *?i1*=(*nat* (*from-l Va t*)) **and**
?j1= (*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*))) **and**
?i2 = (*nat* (*from-l Va t* + *1*)) **and**
?j2 = (*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*))∗*4*)
have *from-l-gt0*:(*nat* (*from-l Va t* + *1*)) > *0*
using *a6 a7* **by** *linarith*
have *zero-lt-len-levels*:*0* < *length* (*levels* (*mem-pool-info x p*))
by (*metis a0 a13 a2 inv-mempool-info-def invariant.inv-def mp-alloc-stm4-lvl-len*)
then have *len-eq*:*length* (*bits* (*levels* (*mem-pool-info x p*) ! *0*)) =
length (*bits* (*levels* (*mem-pool-info Va p*) ! *0*))
using *a13 mp-alloc-stm4-inv-bits-len*
unfolding *gvars-conf-stable-def gvars-conf-def*
by *fastforce*
have *from-l-gt0*:*0* ≤ *from-l Va t* **using** *a7 a6* **by** *linarith*
{ assume *a04*:*j* = *?j1*
then have *get-bit-s x p′ NULL j* ≠ *NOEXIST* **using** *a00 a01 a02*
get-bit-x-l-b a13
mp-alloc-stm4-lvl-len[*OF a2 a13*] *len-eq mp-alloc-stm4-froml*[*OF a19*]
from-l-gt0 a19 a0 a15 a2 same-bit-mp-alloc-stm4-pre-precond-divided
unfolding *inv-bitmap0-def inv-def* **apply** *auto*
using *mp-alloc-stm4-pre-precond-f-same-bits zero-lt-len-levels*
by (*smt BlockState.distinct*(*19*) *nat-0-iff*)
}
moreover {
assume *a04*:*j*≠*?j1*
then have *eq-get-bit-i-j*:*get-bit-s x p 0 j* = *get-bit-s Va p 0 j*
using *same-bit-mp-alloc-x-va*[*OF a15*[*simplified a19*[*simplified mp-alloc-stm4-froml*[*OF a19*], *THEN sym*]] *a16*, *of 0 j*]
using *from-l-gt0* **by** *auto*
then have *get-bit-s x p′ NULL j* ≠ *NOEXIST*

using *a0* **unfolding** *inv-def inv-bitmap0-def a00 a01*
by (*metis a01 a02 a2 len-eq*)
} **ultimately have** *get-bit-s x p′ NULL j* $\neq$ *NOEXIST* **by** *auto*
} **ultimately have** *get-bit-s x p′ NULL j* $\neq$ *NOEXIST* **by** *auto*
} **then show** $\forall$ *p*$\in$*mem-pools x*.
$\forall$ *i*<*length* (*bits* (*levels* (*mem-pool-info x p*) ! *NULL*)).
get-bit-s x p NULL i $\neq$ *NOEXIST* **by** *auto*
qed

lemma *mp-alloc-stm4-inv-bitmapn*:
assumes
a0:*inv Va* **and**
a1:*p* $\in$ *mem-pools Va* **and**
a2:*alloc-l Va t* < *int* (*n-levels* (*mem-pool-info Va p*)) **and**
a3:$\neg$ *free-l Va t* < *OK* **and**
a4:*free-l Va t* $\leq$ *from-l Va t* **and**
a5:*block-num* (*mem-pool-info Va p*)
(*buf* (*mem-pool-info Va p*) + *n* $*$ (*max-sz* (*mem-pool-info Va p*) *div 4* ^ *nat* (*from-l Va t*)))
(*lsizes Va t* ! *nat* (*from-l Va t*))
< *n-max* (*mem-pool-info Va p*) $*$ *4* ^ *nat* (*from-l Va t*) **and**
a6:*from-l Va t* < *alloc-l Va t* **and**
a7:*blk Va t* = *buf* (*mem-pool-info Va p*) + *n* $*$ (*max-sz* (*mem-pool-info Va p*) *div 4* ^ *nat* (*from-l Va t*)) **and**
a8:(*x*, *mp-alloc-stm4-pre-precond-f Va t p*) $\in$ *gvars-conf-stable* **and**
a9:$\forall$ *pa*. *pa* $\neq$ *p* $\longrightarrow$ *mem-pool-info x pa* = *mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *pa* **and**
a10:$\forall$ *jj*. *jj* $\neq$ *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*) $\longrightarrow$
levels (*mem-pool-info x p*) ! *jj* = *levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) ! *jj* **and**
a11:*bits* (*levels* (*mem-pool-info x p*) ! *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*)) =
list-updates-n
(*bits* (*levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*)))
(*Suc* (*bn* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* $*$ *4*)) *3 FREE* **and**
a12:*free-list* (*levels* (*mem-pool-info x p*) ! *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*)) =
inserts
(*map* (λ*ii*. *lsizes* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*) $*$
ii +
blk (*mp-alloc-stm4-pre-precond-f Va t p*) *t*)
[*Suc NULL*..<*4*])
(*free-list*
(*levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*))) **and**
a13:*from-l x* = *from-l* (*mp-alloc-stm4-pre-precond-f Va t p*)
shows *inv-bitmapn x*

proof(*simp add*: *inv-bitmapn-def Let-def*)
 { **fix** *p′ j*
  let *?k* = (*length* (*levels* (*mem-pool-info x p′*)) − *Suc 0*)
 assume *a00*:*p′* ∈ *mem-pools x*
 assume *a01*:*j*<*length* (*bits* (*levels* (*mem-pool-info x p′*) ! *?k*))
 { **assume** *p′*≠ *p*
  then have *get-bit-s x p′ ?k j* ≠ *DIVIDED*
   using *a00 a01 a0 a8 a9 mp-alloc-stm4-mempools mp-alloc-stm4-pres-mpinfo*
   unfolding *inv-bitmapn-def inv-def*
   by (*metis One-nat-def*)
 }
 moreover { **assume** *a02*:*p′* = *p*
  let *?i1*=(*nat* (*from-l Va t*)) **and**
  ?j1= (*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*))) **and**
  ?i2 = (*nat* (*from-l Va t* + *1*)) **and**
  ?j2 = (*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*))∗*4*)
  have *from-l-gt0*:(*nat* (*from-l Va t* + *1*)) > *0*
   using *a3 a4* **by** *linarith*
  have *zero-lt-len-levels*:*0* < *length* (*levels* (*mem-pool-info x p*))
  by (*metis a0 a8 a1 inv-mempool-info-def invariant.inv-def mp-alloc-stm4-lvl-len*)
  then have *len-eq*:*length* (*bits* (*levels* (*mem-pool-info x p*) ! *0*)) =
   length (*bits* (*levels* (*mem-pool-info Va p*) ! *0*))
   using *a8 mp-alloc-stm4-inv-bits-len*
    unfolding *gvars-conf-stable-def gvars-conf-def*
    by *fastforce*
 have *mem-pools*:*mem-pools x* = *mem-pools Va* **using** *mp-alloc-stm4-mempools*[*OF a8*] **by** *auto*
  have *inv-mempool-info-mp Va p*
   using *a1 mem-pools a0* **unfolding** *inv-def inv-mempool-info-def Let-def* **by** *auto*
  note *inv-mempool*=*this*[*simplified Let-def*]
  have *from-l*:*from-l x* = *from-l Va*
   using *mp-alloc-stm4-froml*[*OF a13*] **by** *auto*
  have *from-l-suc*:*nat* (*from-l Va t* + *1*) = *nat*(*from-l Va t*) + *1*
   using *from-l-gt0* **by** *auto*
  have *i1-len*:*?i1* < *length* (*levels* (*mem-pool-info Va p*))
   using *a6 a1 a2 from-l-gt0 a0* **unfolding** *inv-def inv-mempool-info-def Let-def*
    by *auto*
  have *i2-len*:*?i2* < *length* (*levels* (*mem-pool-info Va p*))
   using *a0 a6 a1 a2 from-l-gt0* **unfolding** *inv-def inv-mempool-info-def Let-def*
   by *auto*
  have *j1-len*:*?j1* < *length* (*bits* (*levels* (*mem-pool-info Va p*) ! *?i1*))
   by (*metis a0 a1 a5 a7 i1-len inv-mempool-info-def invariant.inv-def*)
  have *j2-len*:*Suc* (*Suc* (*Suc ?j2*)) < *length* (*bits* (*levels* (*mem-pool-info Va p*) ! *?i2*))
   using *i1-len i2-len j1-len inv-mempool from-l-suc*
   by *simp*

have *from-l-gt0*:*0* ≤ *from-l Va t* **using** *a4 a3* **by** *linarith*
{ assume *a03*:*?i2* = *?k*
{ assume *a04*:*j*≥ *?j2* ∧ *j*<*?j2*+*4*
{ assume *a05*:*j*=*?j2*
then have *get-bit-s x p′ ?i2 j* = *ALLOCATING* **using** *a00 a01 a02 a03*
get-bit-x-l1-b4[*OF* - *from-l-gt0 a11 i2-len*,*of ?j2*] *a8 zero-lt-len-levels*
mp-alloc-stm4-lvl-len[*OF a1 a8*] *len-eq mp-alloc-stm4-froml*[*OF a13*]
from-l-gt0 a13 j2-len
by (*meson Suc-lessD mult.commute*)
then have *get-bit-s x p′ ?i2 j* ≠ *DIVIDED* **by** *auto*
}
moreover {
assume *a05*:*j*≥ *?j2* + *1* ∧ *j*<*?j2*+*4*
then have *get-bit-s x p′ ?i2 j* = *FREE* **using** *a00 a01 a02 a03*
get-bit-x-l1-b41[*OF* - *from-l-gt0 a10*[*simplified from-l a13*[*THEN sym*]]
a11 i2-len,*of ?j2*] *a8 zero-lt-len-levels*
mp-alloc-stm4-lvl-len[*OF a1 a8*] *len-eq mp-alloc-stm4-froml*[*OF a13*]
from-l-gt0 a13 j2-len a11 mp-alloc-stm4-pre-precond-f-bn
by (*smt One-nat-def add.commute add-Suc-shift length-list-update-n*
list-updates-n-eq
numeral-2-eq-2 numeral-3-eq-3 numeral-Bit0 plus-1-eq-Suc)
then have *get-bit-s x p′ ?i2 j* ≠ *DIVIDED* **by** *auto*
} ultimately have *get-bit-s x p′ ?i2 j* ≠ *DIVIDED* **using** *a04* **by** *fastforce*
}
moreover{
assume ¬(*j*≥ *?j2* ∧ *j*<*?j2*+*4*)
moreover have *eq-get-bit-i-j*:*get-bit-s x p ?i2 j* = *get-bit-s Va p ?i2 j*
using *a03 from-l-suc same-bit-mp-alloc-x-va*[*OF*
a10[*simplified a13*[*simplified mp-alloc-stm4-froml*[*OF a13*], *THEN sym*]]
a11, *of ?i2 j*]
from-l-gt0 calculation
by *force*
ultimately have *get-bit-s x p′ ?i2 j* ≠ *DIVIDED*
using *a0 a02 a03* **unfolding** *inv-def inv-bitmapn-def Let-def*
by (*metis One-nat-def a01 a8 a10 a11 a1 length-list-update-n*
mp-alloc-stm4-inv-bits-len mp-alloc-stm4-lvl-len)
} ultimately have *get-bit-s x p′ ?k j* ≠ *DIVIDED* **using** *a03* **by** *auto*
}
moreover {
assume *?i2* ≠ *?k*
moreover have *?i2*<*?k*
using *calculation a00 a02 a8 i2-len mem-pools mp-alloc-stm4-lvl-len* **by** *auto*
then have *?i1*≠*?k*
by *linarith*
ultimately have *eq-get-bit-i-j*:*get-bit-s x p ?k j* = *get-bit-s Va p ?k j*
using *from-l-suc same-bit-mp-alloc-x-va*[*OF*
a10[*simplified a13*[*simplified mp-alloc-stm4-froml*[*OF a13*], *THEN sym*]]
a11, *of ?i2 j*]
from-l-gt0

by (*metis a10 a13 from-l same-bit-mp-alloc-stm4-pre-precond-f1*)
then have *get-bit-s x p' ?k j* $\neq$ *DIVIDED*
using *a0 a02* **unfolding** *inv-def inv-bitmapn-def Let-def*
by (*metis One-nat-def a01 a8 a10 a11 a1 length-list-update-n mp-alloc-stm4-inv-bits-len mp-alloc-stm4-lvl-len*)
} **ultimately have** *get-bit-s x p' ?k j* $\neq$ *DIVIDED* **by** *auto*
} **ultimately have** *get-bit-s x p' ?k j* $\neq$ *DIVIDED* **by** *auto*
}
then show $\forall$ *p*$\in$*mem-pools x*.
$\forall$ *i*<*length* (*bits* (*levels* (*mem-pool-info x p*) ! (*length* (*levels* (*mem-pool-info x p*)) − *Suc NULL*))).
get-bit-s x p (*length* (*levels* (*mem-pool-info x p*)) − *Suc NULL*) *i* $\neq$ *DIVIDED* **by** *auto*

qed

lemma *mp-alloc-stm4-inv-bitmap4free*:
assumes
a0:*inv Va* **and**
a1:*freeing-node Va t* = *None* **and**
a2:*p* $\in$ *mem-pools Va* **and**
a3:$\forall$ *ii*<*length* (*lsizes Va t*). *lsizes Va t* ! *ii* = *ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *ii* **and**
a4:*length* (*lsizes Va t*) $\leq$ *n-levels* (*mem-pool-info Va p*) **and**
a5:*alloc-l Va t* < *int* (*n-levels* (*mem-pool-info Va p*)) **and**
a6:¬ *free-l Va t* < *OK* **and**
a7:*free-l Va t* $\leq$ *from-l Va t* **and**
a8:*allocating-node Va t* =
Some (|*pool* = *p*, *level* = *nat* (*from-l Va t*),
block = *block-num* (*mem-pool-info Va p*)
(*buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ˆ *nat* (*from-l Va t*)))
(*lsizes Va t* ! *nat* (*from-l Va t*)),
data = *buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ˆ *nat* (*from-l Va t*))|) **and**
a9:*n* = *block-num* (*mem-pool-info Va p*)
(*buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ˆ *nat* (*from-l Va t*)))
(*lsizes Va t* ! *nat* (*from-l Va t*)) $\vee$
max-sz (*mem-pool-info Va p*) *div 4* ˆ *nat* (*from-l Va t*) = *NULL* **and**
a10:*block-num* (*mem-pool-info Va p*)
(*buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ˆ *nat* (*from-l Va t*)))
(*lsizes Va t* ! *nat* (*from-l Va t*))
< *n-max* (*mem-pool-info Va p*) ∗ *4* ˆ *nat* (*from-l Va t*) **and**
a11:*from-l Va t* < *alloc-l Va t* **and**
a12:*blk Va t* = *buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4* ˆ *nat* (*from-l Va t*)) **and**
a13:(*x*, *mp-alloc-stm4-pre-precond-f Va t p*) $\in$ *gvars-conf-stable* **and**

a14: $\forall$ *pa*. *pa* $\neq$ *p* $\longrightarrow$ *mem-pool-info x pa* = *mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *pa* **and**
a15: $\forall$ *jj*. *jj* $\neq$ *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*) $\longrightarrow$
levels (*mem-pool-info x p*) ! *jj* = *levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) ! *jj* **and**
a16: *bits* (*levels* (*mem-pool-info x p*) ! *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*)) =
list-updates-n
(*bits* (*levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*)))
(*Suc* (*bn* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* ∗ *4*)) *3 FREE* **and**
a17: *free-list* (*levels* (*mem-pool-info x p*) ! *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*)) =
inserts
(*map* (λ*ii*. *lsizes* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*) ∗
ii +
blk (*mp-alloc-stm4-pre-precond-f Va t p*) *t*)
[*Suc NULL*..<*4*])
(*free-list*
(*levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*) !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*))) **and**
a18: *lsizes x* = *lsizes* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
a19: *from-l x* = *from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
a20: *freeing-node x* = *freeing-node* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
a21: *allocating-node x* = *allocating-node* (*mp-alloc-stm4-pre-precond-f Va t p*) **and**
a22: *alloc-l Va t* = *int* (*length* (*lsizes Va t*)) − *1* ∧ *length* (*lsizes Va t*) = *n-levels* (*mem-pool-info Va p*) ∨
alloc-l Va t = *int* (*length* (*lsizes Va t*)) − *2* ∧ *lsizes Va t* ! *nat* (*alloc-l Va t* + *1*) < *sz* **and**
a23: *blk x* = *blk* (*mp-alloc-stm4-pre-precond-f Va t p*)
shows *inv-bitmap-not4free x*
proof−
{ **fix** *p′ i j*
assume *a00*: *p′* ∈ *mem-pools x* **and**
a01: *i*<*length* (*levels* (*mem-pool-info x p′*)) **and**
a02: *j*<*length* (*bits* (*levels* (*mem-pool-info x p′*) ! *i*))
{ **assume** *a03*: *0* < *i* **and**
a04: *get-bit-s x p′ i* (*Suc* (*Suc* (*j div 4* ∗ *4*))) = *FREE* **and**
a05: *get-bit-s x p′ i* (*Suc* (*j div 4* ∗ *4*)) = *FREE* **and**
a06: *get-bit-s x p′ i* (*j div 4* ∗ *4*) = *FREE*
{ **assume** *p′*≠ *p*
then have *get-bit-s x p′ i* (*j div 4* ∗ *4* + *3*) ≠ *FREE*
using *a00 a01 a0 a8* **using** *a00 a01 a0 a8 a9 mp-alloc-stm4-mempools mp-alloc-stm4-pres-mpinfo*
unfolding *inv-bitmap-not4free-def inv-def*
by (*metis a02 a03 a04 a05 a06 a13 a14 add.commute*
add-2-eq-Suc′ partner-bits-def plus-1-eq-Suc)
} **note** *not-p* = *this*

moreover{
assume *a07*:$p' = p$
let *?i1*=(*nat* (*from-l Va t*)) **and**
?j1= (*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*))) **and**
?i2 = (*nat* (*from-l Va t* + *1*)) **and**
?j2 = (*block-num* (*mem-pool-info Va p*) (*blk Va t*) (*lsizes Va t* ! *nat* (*from-l Va t*))∗*4*)
have *from-l-gt0*:(*nat* (*from-l Va t* + *1*)) > *0*
using *a6 a7* **by** *linarith*
have *zero-lt-len-levels*:*0* < *length* (*levels* (*mem-pool-info x p*))
using *a0 a2 mp-alloc-stm4-lvl-len* **unfolding** *inv-mempool-info-def inv-def*
using *a01 a07 gr-implies-not0* **by** *blast*
then have *len-eq*:*length* (*bits* (*levels* (*mem-pool-info x p*) ! *0*)) =
length (*bits* (*levels* (*mem-pool-info Va p*) ! *0*))
using *a13 mp-alloc-stm4-inv-bits-len*
unfolding *gvars-conf-stable-def gvars-conf-def*
by *fastforce*
have *mem-pools*:*mem-pools x* = *mem-pools Va* **using** *mp-alloc-stm4-mempools*[*OF a13*] **by** *auto*
have *inv-mempool-info-mp Va p*
using *a2 mem-pools a0* **unfolding** *inv-def inv-mempool-info-def Let-def* **by** *auto*
note *inv-mempool*=*this*[*simplified Let-def*]
have *from-l*:*from-l x* = *from-l Va*
using *mp-alloc-stm4-froml*[*OF a19*] **by** *auto*
have *from-l-suc*:*nat* (*from-l Va t* + *1*) = *nat*(*from-l Va t*) + *1*
using *from-l-gt0* **by** *auto*
have *i1-len*:*?i1* < *length* (*levels* (*mem-pool-info Va p*))
using *a2 a11 a5 from-l-gt0 a0* **unfolding** *inv-def inv-mempool-info-def Let-def*
by *auto*
have *i2-len*:*?i2* < *length* (*levels* (*mem-pool-info Va p*))
using *a0 a5 a2 a11 from-l-gt0* **unfolding** *inv-def inv-mempool-info-def Let-def*
by *auto*
have *j1-len*:*?j1* < *length* (*bits* (*levels* (*mem-pool-info Va p*) ! *?i1*))
using *assms*(*11*) *assms*(*13*) *i1-len inv-mempool* **by** *presburger*
have *j2-len*:*Suc* (*Suc* (*Suc ?j2*)) < *length* (*bits* (*levels* (*mem-pool-info Va p*) ! *?i2*))
using *i1-len i2-len j1-len inv-mempool from-l-suc*
by *simp*

have *from-l-gt0*:*0* ≤ *from-l Va t*
using *a6 a7* **by** *linarith*
{ **assume** *a08*:*i*≠*?i1* ∧ *i*≠*?i2*
then have *eq-get-bit-i-j*:*get-bit-s x p i* (*j div 4* ∗ *4* + *3*) = *get-bit-s Va p i* (*j div 4* ∗ *4* + *3*)
using *same-bit-mp-alloc-x-va*

[*OF a15*[*simplified a19*[*simplified mp-alloc-stm4-froml*[*OF a19*],
THEN sym]] *a16*, *of i* (*j div 4 * 4 + 3*)]
from-l-gt0 **by** *auto*
moreover have *eq-get-bit-i-j*:*get-bit-s x p i* (*j div 4 * 4*) = *get-bit-s Va p i* (*j div 4 * 4*)
using *same-bit-mp-alloc-x-va*
[*OF a15*[*simplified a19*[*simplified mp-alloc-stm4-froml*[*OF a19*],
THEN sym]] *a16*, *of i* (*j div 4 * 4*)] *a08*
from-l-gt0 **by** *auto*
moreover have *eq-get-bit-i-j*:*get-bit-s x p i* (*Suc* (*j div 4 * 4*)) = *get-bit-s Va p i* (*Suc* (*j div 4 * 4*))
using *same-bit-mp-alloc-x-va*
[*OF a15*[*simplified a19*[*simplified mp-alloc-stm4-froml*[*OF a19*],
THEN sym]] *a16*, *of i* (*Suc* (*j div 4 * 4*))] *a08*
from-l-gt0 **by** *auto*
moreover have *eq-get-bit-i-j*:*get-bit-s x p i* (*Suc* (*Suc* (*j div 4 * 4*))) = *get-bit-s Va p i* (*Suc* (*Suc* (*j div 4 * 4*)))
using *same-bit-mp-alloc-x-va*
[*OF a15*[*simplified a19*[*simplified mp-alloc-stm4-froml*[*OF a19*],
THEN sym]] *a16*, *of i* (*Suc* (*Suc* (*j div 4 * 4*)))] *a08*
from-l-gt0 **by** *auto*
ultimately have *get-bit-s x p′ i* (*j div 4 * 4 + 3*) ≠ *FREE*
using *a07 a03 a04 a05 a06 a01 a02 a0 a13 a15 a16 a2 mp-alloc-stm4-inv-bits-len mp-alloc-stm4-lvl-len*
unfolding *inv-bitmap-not4free-def inv-def Let-def partner-bits-def*
by (*metis add.commute add-2-eq-Suc′ length-list-update-n plus-1-eq-Suc*)

}
moreover { assume *i=?i1*
then have *get-bit-s x p′ i* (*j div 4 * 4 + 3*) ≠ *FREE*
using *not-p a0 a02 a03 a04 a05 a06 a15 a19 a2 from-l from-l-gt0 from-l-suc i1-len j1-len*
mp-alloc-stm4-inv-bits-len mp-alloc-stm4-pre-precond-f-bitmap-not-free
same-bit-mp-alloc-stm4-pre-precond-f1
unfolding *inv-bitmap-not4free-def invariant.inv-def partner-bits-def*
by (*smt add-2-eq-Suc′ add-eq-self-zero le-zero-eq nat-int-add not-one-le-zero*

plus-1-eq-Suc)
}note *i1= this*
moreover {
assume *a08*:*i=?i2*
{ assume *j* ≥ *?j2* ∧ *j* ≤ *?j2 + 3*
then have *j* = *?j2* ∨ *j*= *?j2 + 1* ∨ *j*= *?j2 + 2* ∨ *j* = *?j2 + 3*
by *auto*
then have *j div 4 * 4* = *?j2* **by** *auto*
moreover have *get-bit-s x p′ i ?j2* = *ALLOCATING*
using *get-bit-x-l1-b4*[*OF - from-l-gt0 a16 i2-len*] *a08 a07 mult.commute j2-len*
by (*metis Suc-lessD*)

ultimately have *get-bit-s x p′ i* (*j div 4* ∗ *4* + *3*) ≠ *FREE* **using** *a06* **by** *auto*

}

moreover {

assume ¬(*j*≥*?j2* ∧ *j*≤ *?j2* + *3*)

then have *j* < *?j2* ∨ *j* > *?j2* + *3*

by *auto*

moreover { **assume** *j*<*?j2*

then have *j div 4* ∗ *4* + *3* < *?j2*

by *presburger*

moreover have *get-bit-s x p i* (*j div 4*∗*4*) = *get-bit-s Va p i* (*j div 4*∗*4*)

using *same-bit-mp-alloc-x-va*[*OF a15*[*simplified from-l a19*[*THEN sym*]] *a16*, *of i* (*j div 4*∗*4*)]

a01 a02 a03 a04 a05 a06 a07 a08 a00 calculation

a0 a16 a19 a2 from-l i2-len mp-alloc-stm4-inv-bits-len from-l-suc

by (*auto simp add*: *a16*)

moreover have *get-bit-s x p i* (*j div 4*∗*4*+*1*) = *get-bit-s Va p i* (*j div 4*∗*4*+*1*)

using *same-bit-mp-alloc-x-va*[*OF a15*[*simplified from-l a19*[*THEN sym*]] *a16*, *of i* (*j div 4*∗*4*+*1*)]

a01 a02 a03 a04 a05 a06 a07 a08 a00 calculation

a0 a16 a19 a2 from-l i2-len mp-alloc-stm4-inv-bits-len from-l-suc

by (*auto simp add*: *a16*)

moreover have *get-bit-s x p i* (*j div 4*∗*4*+*2*) = *get-bit-s Va p i* (*j div 4*∗*4*+*2*)

using *same-bit-mp-alloc-x-va*[*OF a15*[*simplified from-l a19*[*THEN sym*]] *a16*, *of i* (*j div 4*∗*4*+*2*)]

a01 a02 a03 a04 a05 a06 a07 a08 a00 calculation

a0 a16 a19 a2 from-l i2-len mp-alloc-stm4-inv-bits-len from-l-suc

by (*auto simp add*: *a16*)

moreover have *get-bit-s x p i* (*j div 4*∗*4*+*3*) = *get-bit-s Va p i* (*j div 4*∗*4*+*3*)

using *same-bit-mp-alloc-x-va*[*OF a15*[*simplified from-l a19*[*THEN sym*]] *a16*, *of i* (*j div 4*∗*4*+*3*)]

a01 a02 a03 a04 a05 a06 a07 a08 a00 calculation

a0 a16 a19 a2 from-l i2-len mp-alloc-stm4-inv-bits-len from-l-suc

by (*auto simp add*: *a16*)

ultimately have *get-bit-s x p′ i* (*j div 4* ∗ *4* + *3*) ≠ *FREE*

using *same-bit-mp-alloc-x-va*[*OF - a16*] *a15 a01 a02 a03 a04 a05 a06 a07 a08 a00*

a0 a16 a19 a2 from-l i2-len mp-alloc-stm4-inv-bits-len i1

unfolding *inv-def inv-bitmap-not4free-def partner-bits-def*

by (*smt add.commute add-2-eq-Suc′ length-list-update-n plus-1-eq-Suc*)

}

moreover{

assume *j*>*?j2* + *3*

then have *j div 4* ∗ *4* > *?j2* + *3*

by *presburger*
moreover have *get-bit-s x p i (j div 4∗4) = get-bit-s Va p i (j div 4∗4)*
using *same-bit-mp-alloc-x-va[OF a15[simplified from-l a19[THEN sym]] a16, of i (j div 4∗4)]*
a01 a02 a03 a04 a05 a06 a07 a08 a00 calculation
a0 a16 a19 a2 from-l i2-len mp-alloc-stm4-inv-bits-len from-l-suc
by (*auto simp add*: *a16*)
moreover have *get-bit-s x p i (j div 4∗4+1) = get-bit-s Va p i (j div 4∗4+1)*
using *same-bit-mp-alloc-x-va[OF a15[simplified from-l a19[THEN sym]] a16, of i (j div 4∗4+1)]*
a01 a02 a03 a04 a05 a06 a07 a08 a00 calculation
a0 a16 a19 a2 from-l i2-len mp-alloc-stm4-inv-bits-len from-l-suc
by (*auto simp add*: *a16*)
moreover have *get-bit-s x p i (j div 4∗4+2) = get-bit-s Va p i (j div 4∗4+2)*
using *same-bit-mp-alloc-x-va[OF a15[simplified from-l a19[THEN sym]] a16, of i (j div 4∗4+2)]*
a01 a02 a03 a04 a05 a06 a07 a08 a00 calculation
a0 a16 a19 a2 from-l i2-len mp-alloc-stm4-inv-bits-len from-l-suc
by (*auto simp add*: *a16*)
moreover have *get-bit-s x p i (j div 4∗4+3) = get-bit-s Va p i (j div 4∗4+3)*
using *same-bit-mp-alloc-x-va[OF a15[simplified from-l a19[THEN sym]] a16, of i (j div 4∗4+3)]*
a01 a02 a03 a04 a05 a06 a07 a08 a00 calculation
a0 a16 a19 a2 from-l i2-len mp-alloc-stm4-inv-bits-len from-l-suc
by (*auto simp add*: *a16*)
ultimately have *get-bit-s x p′ i (j div 4 ∗ 4 + 3) ≠ FREE*
using *same-bit-mp-alloc-x-va[OF - a16] a15 a01 a02 a03 a04 a05 a06 a07 a08 a00*
a0 a16 a19 a2 from-l i2-len mp-alloc-stm4-inv-bits-len i1
unfolding *inv-def inv-bitmap-not4free-def partner-bits-def*
by (*smt add.commute add-2-eq-Suc′ length-list-update-n plus-1-eq-Suc*)
}
ultimately have *get-bit-s x p′ i (j div 4 ∗ 4 + 3) ≠ FREE* **by** *auto*
} **ultimately have** *get-bit-s x p′ i (j div 4 ∗ 4 + 3) ≠ FREE* **by** *auto*
} **ultimately have** *get-bit-s x p′ i (j div 4 ∗ 4 + 3) ≠ FREE* **by** *auto*
} **ultimately have** *get-bit-s x p′ i (j div 4 ∗ 4 + 3) ≠ FREE* **by** *auto*
}
} **then show** *inv-bitmap-not4free x*
unfolding *inv-bitmap-not4free-def Let-def partner-bits-def*
by *auto*
qed

lemma *mp-alloc-stm4-whlpst-in-post-inv*:
inv Va ⟹
freeing-node Va t = None ⟹

$p \in mem\text{-}pools\ Va \Longrightarrow$
$ETIMEOUT \leq timeout \Longrightarrow$
$timeout = ETIMEOUT \longrightarrow tmout\ Va\ t = ETIMEOUT \Longrightarrow$
$\neg\ rf\ Va\ t \Longrightarrow$
$\forall\, ii{<}length\ (lsizes\ Va\ t).\ lsizes\ Va\ t\ !\ ii = ALIGN4\ (max\text{-}sz\ (mem\text{-}pool\text{-}info\ Va\ p))\ div\ 4\ \hat{}\ ii \Longrightarrow$
$length\ (lsizes\ Va\ t) \leq n\text{-}levels\ (mem\text{-}pool\text{-}info\ Va\ p) \Longrightarrow$
$alloc\text{-}l\ Va\ t < int\ (n\text{-}levels\ (mem\text{-}pool\text{-}info\ Va\ p)) \Longrightarrow$
$\neg\ free\text{-}l\ Va\ t < OK \Longrightarrow$
$NULL < buf\ (mem\text{-}pool\text{-}info\ Va\ p) \vee NULL < n \wedge NULL < max\text{-}sz\ (mem\text{-}pool\text{-}info\ Va\ p)\ div\ 4\ \hat{}\ nat\ (from\text{-}l\ Va\ t) \Longrightarrow$
$free\text{-}l\ Va\ t \leq from\text{-}l\ Va\ t \Longrightarrow$
$allocating\text{-}node\ Va\ t =$
$Some\ (\!|pool = p,\ level = nat\ (from\text{-}l\ Va\ t),$
$block = block\text{-}num\ (mem\text{-}pool\text{-}info\ Va\ p)$
$(buf\ (mem\text{-}pool\text{-}info\ Va\ p) + n * (max\text{-}sz\ (mem\text{-}pool\text{-}info\ Va\ p)\ div\ 4\ \hat{}\ nat\ (from\text{-}l\ Va\ t)))$
$(lsizes\ Va\ t\ !\ nat\ (from\text{-}l\ Va\ t)),$
$data = buf\ (mem\text{-}pool\text{-}info\ Va\ p) + n * (max\text{-}sz\ (mem\text{-}pool\text{-}info\ Va\ p)\ div\ 4\ \hat{}\ nat\ (from\text{-}l\ Va\ t))|\!) \Longrightarrow$
$n = block\text{-}num\ (mem\text{-}pool\text{-}info\ Va\ p)$
$(buf\ (mem\text{-}pool\text{-}info\ Va\ p) + n * (max\text{-}sz\ (mem\text{-}pool\text{-}info\ Va\ p)\ div\ 4\ \hat{}\ nat\ (from\text{-}l\ Va\ t)))$
$(lsizes\ Va\ t\ !\ nat\ (from\text{-}l\ Va\ t)) \vee$
$max\text{-}sz\ (mem\text{-}pool\text{-}info\ Va\ p)\ div\ 4\ \hat{}\ nat\ (from\text{-}l\ Va\ t) = NULL \Longrightarrow$
$block\text{-}num\ (mem\text{-}pool\text{-}info\ Va\ p)$
$(buf\ (mem\text{-}pool\text{-}info\ Va\ p) + n * (max\text{-}sz\ (mem\text{-}pool\text{-}info\ Va\ p)\ div\ 4\ \hat{}\ nat\ (from\text{-}l\ Va\ t)))$
$(lsizes\ Va\ t\ !\ nat\ (from\text{-}l\ Va\ t))$
$< n\text{-}max\ (mem\text{-}pool\text{-}info\ Va\ p) * 4\ \hat{}\ nat\ (from\text{-}l\ Va\ t) \Longrightarrow$
$from\text{-}l\ Va\ t < alloc\text{-}l\ Va\ t \Longrightarrow$
$cur\ Va = Some\ t \Longrightarrow$
$n < n\text{-}max\ (mem\text{-}pool\text{-}info\ Va\ p) * 4\ \hat{}\ nat\ (from\text{-}l\ Va\ t) \Longrightarrow$
$blk\ Va\ t = buf\ (mem\text{-}pool\text{-}info\ Va\ p) + n * (max\text{-}sz\ (mem\text{-}pool\text{-}info\ Va\ p)\ div\ 4\ \hat{}\ nat\ (from\text{-}l\ Va\ t)) \Longrightarrow$
$mempoolalloc\text{-}ret\ Va\ t = None \Longrightarrow$
$\forall\, ii{\leq}nat\ (alloc\text{-}l\ Va\ t).\ sz \leq lsizes\ Va\ t\ !\ ii \Longrightarrow$
$alloc\text{-}l\ Va\ t = int\ (length\ (lsizes\ Va\ t)) - 1 \wedge length\ (lsizes\ Va\ t) = n\text{-}levels\ (mem\text{-}pool\text{-}info\ Va\ p) \vee$
$alloc\text{-}l\ Va\ t = int\ (length\ (lsizes\ Va\ t)) - 2 \wedge lsizes\ Va\ t\ !\ nat\ (alloc\text{-}l\ Va\ t + 1) < sz \Longrightarrow$
$i\ x\ t = 4 \Longrightarrow$
$cur\ x = cur\ (mp\text{-}alloc\text{-}stm4\text{-}pre\text{-}precond\text{-}f\ Va\ t\ p) \Longrightarrow$
$tick\ x = tick\ (mp\text{-}alloc\text{-}stm4\text{-}pre\text{-}precond\text{-}f\ Va\ t\ p) \Longrightarrow$
$thd\text{-}state\ x = thd\text{-}state\ (mp\text{-}alloc\text{-}stm4\text{-}pre\text{-}precond\text{-}f\ Va\ t\ p) \Longrightarrow$
$(x,\ mp\text{-}alloc\text{-}stm4\text{-}pre\text{-}precond\text{-}f\ Va\ t\ p) \in gvars\text{-}conf\text{-}stable \Longrightarrow$
$\forall\, pa.\ pa \neq p \longrightarrow mem\text{-}pool\text{-}info\ x\ pa = mem\text{-}pool\text{-}info\ (mp\text{-}alloc\text{-}stm4\text{-}pre\text{-}precond\text{-}f\ Va\ t\ p)\ pa \Longrightarrow$
$wait\text{-}q\ (mem\text{-}pool\text{-}info\ x\ p) = wait\text{-}q\ (mem\text{-}pool\text{-}info\ (mp\text{-}alloc\text{-}stm4\text{-}pre\text{-}precond\text{-}f$

$Va\ t\ p)\ p) \Longrightarrow$
$\forall\, t'.\ t' \neq t \longrightarrow$ *lvars-nochange* $t'\ x$ (*mp-alloc-stm4-pre-precond-f* $Va\ t\ p) \Longrightarrow$
$\forall\, jj.\ jj \neq$ *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f* $Va\ t\ p)\ t + 1) \longrightarrow$
levels (*mem-pool-info* $x\ p$) ! jj = *levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f* $Va\ t\ p)\ p)$! $jj \Longrightarrow$
bits (*levels* (*mem-pool-info* $x\ p$) ! *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f* $Va\ t\ p$) $t + 1$)) =
list-updates-n
(*bits* (*levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f* $Va\ t\ p$) p) !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f* $Va\ t\ p$) $t + 1$)))
(*Suc* (*bn* (*mp-alloc-stm4-pre-precond-f* $Va\ t\ p$) $t * 4$)) *3 FREE* $\Longrightarrow$
free-list (*levels* (*mem-pool-info* $x\ p$) ! *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f* $Va\ t\ p$) $t + 1$)) =
inserts
(*map* ($\lambda ii.$ *lsizes* (*mp-alloc-stm4-pre-precond-f* $Va\ t\ p$) t !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f* $Va\ t\ p$) $t + 1$) $*$
ii +
blk (*mp-alloc-stm4-pre-precond-f* $Va\ t\ p$) t)
[*Suc NULL..*<*4*])
(*free-list*
(*levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f* $Va\ t\ p$) p) !
nat (*from-l* (*mp-alloc-stm4-pre-precond-f* $Va\ t\ p$) $t + 1$))) $\Longrightarrow$
j x = *j* (*mp-alloc-stm4-pre-precond-f* $Va\ t\ p) \Longrightarrow$
ret x = *ret* (*mp-alloc-stm4-pre-precond-f* $Va\ t\ p) \Longrightarrow$
endt x = *endt* (*mp-alloc-stm4-pre-precond-f* $Va\ t\ p) \Longrightarrow$
rf x = *rf* (*mp-alloc-stm4-pre-precond-f* $Va\ t\ p) \Longrightarrow$
tmout x = *tmout* (*mp-alloc-stm4-pre-precond-f* $Va\ t\ p) \Longrightarrow$
lsizes x = *lsizes* (*mp-alloc-stm4-pre-precond-f* $Va\ t\ p) \Longrightarrow$
alloc-l x = *alloc-l* (*mp-alloc-stm4-pre-precond-f* $Va\ t\ p) \Longrightarrow$
free-l x = *free-l* (*mp-alloc-stm4-pre-precond-f* $Va\ t\ p) \Longrightarrow$
from-l x = *from-l* (*mp-alloc-stm4-pre-precond-f* $Va\ t\ p) \Longrightarrow$
blk x = *blk* (*mp-alloc-stm4-pre-precond-f* $Va\ t\ p) \Longrightarrow$
nodev x = *nodev* (*mp-alloc-stm4-pre-precond-f* $Va\ t\ p) \Longrightarrow$
bn x = *bn* (*mp-alloc-stm4-pre-precond-f* $Va\ t\ p) \Longrightarrow$
alloc-lsize-r x = *alloc-lsize-r* (*mp-alloc-stm4-pre-precond-f* $Va\ t\ p) \Longrightarrow$
lvl x = *lvl* (*mp-alloc-stm4-pre-precond-f* $Va\ t\ p) \Longrightarrow$
bb x = *bb* (*mp-alloc-stm4-pre-precond-f* $Va\ t\ p) \Longrightarrow$
block-pt x = *block-pt* (*mp-alloc-stm4-pre-precond-f* $Va\ t\ p) \Longrightarrow$
th x = *th* (*mp-alloc-stm4-pre-precond-f* $Va\ t\ p) \Longrightarrow$
need-resched x = *need-resched* (*mp-alloc-stm4-pre-precond-f* $Va\ t\ p) \Longrightarrow$
mempoolalloc-ret x = *mempoolalloc-ret* (*mp-alloc-stm4-pre-precond-f* $Va\ t\ p) \Longrightarrow$
freeing-node x = *freeing-node* (*mp-alloc-stm4-pre-precond-f* $Va\ t\ p) \Longrightarrow$
allocating-node x = *allocating-node* (*mp-alloc-stm4-pre-precond-f* $Va\ t\ p) \Longrightarrow$ *inv* x
apply(*simp add*:*inv-def*)
apply(*rule conjI*) **apply**(*simp add*:*inv-cur-def*) **apply** *clarify* **using** *mp-alloc-stm4-inv-cur*
apply *metis*
apply(*rule conjI*) **using** *mp-alloc-stm4-inv-thd-state* **apply** *metis*
apply(*rule conjI*) **using** *mp-alloc-stm4-inv-mempool-info* **apply** *metis*

apply(*rule conjI*) **using** *mp-alloc-stm4-inv-bitmap-freelist* **apply** *blast*
apply(*rule conjI*) **using** *mp-alloc-stm4-inv-bitmap* **unfolding** *inv-def* **apply** *blast*
apply(*rule conjI*) **using** *mp-alloc-stm4-inv-aux-vars* **unfolding** *inv-def* **apply** *blast*
apply(*rule conjI*) **using** *mp-alloc-stm4-inv-bitmap0* **unfolding** *inv-def* **apply** *blast*
apply (*rule conjI*) **using** *mp-alloc-stm4-inv-bitmapn* **unfolding** *inv-def* **apply** *blast*
using *mp-alloc-stm4-inv-bitmap4free* **unfolding** *inv-def* **by** *blast*

lemma *mp-alloc-stm4-whlpst-in-post-h1*:
$p \in \textit{mem-pools Va} \Longrightarrow$
$\textit{inv Va} \Longrightarrow$
$\textit{alloc-l Va t} < \textit{int} (\textit{n-levels} (\textit{mem-pool-info Va p})) \Longrightarrow$
$\textit{from-l Va t} < \textit{alloc-l Va t} \Longrightarrow$
$\neg \textit{free-l Va t} < 0 \Longrightarrow$
$\textit{free-l Va t} \leq \textit{from-l Va t} \Longrightarrow$
$\forall ii<\textit{length} (\textit{lsizes Va t}).\ \textit{lsizes Va t} ! ii = \textit{ALIGN4} (\textit{max-sz} (\textit{mem-pool-info Va p})) \textit{ div } 4 \hat{}\ ii \Longrightarrow$
$\textit{length} (\textit{lsizes Va t}) \leq \textit{n-levels} (\textit{mem-pool-info Va p}) \Longrightarrow$
$\textit{alloc-l Va t} = \textit{int} (\textit{length} (\textit{lsizes Va t})) - 1 \wedge \textit{length} (\textit{lsizes Va t}) = \textit{n-levels} (\textit{mem-pool-info Va p}) \vee$
$\textit{alloc-l Va t} = \textit{int} (\textit{length} (\textit{lsizes Va t})) - 2 \wedge \textit{lsizes Va t} ! \textit{nat} (\textit{alloc-l Va t} + 1) < sz \Longrightarrow$
$\textit{blk Va t} = \textit{buf} (\textit{mem-pool-info Va p}) + n * (\textit{max-sz} (\textit{mem-pool-info Va p}) \textit{ div } 4 \hat{}\ \textit{nat} (\textit{from-l Va t})) \Longrightarrow$
$(x, \textit{mp-alloc-stm4-pre-precond-f Va t p}) \in \textit{gvars-conf-stable} \Longrightarrow$
$\textit{allocating-node} (\textit{mp-alloc-stm4-pre-precond-f Va t p})\ t =$
$\textit{Some} (\!|\textit{pool} = p, \textit{level} = \textit{nat} (\textit{from-l} (\textit{mp-alloc-stm4-pre-precond-f Va t p})\ t + 1),$
$\textit{block} = \textit{block-num} (\textit{mem-pool-info } x\ p) (\textit{blk} (\textit{mp-alloc-stm4-pre-precond-f Va t p})\ t)$
$(\textit{lsizes} (\textit{mp-alloc-stm4-pre-precond-f Va t p})\ t\ !$
$\textit{nat} (\textit{from-l} (\textit{mp-alloc-stm4-pre-precond-f Va t p})\ t + 1)),$
$\textit{data} = \textit{blk} (\textit{mp-alloc-stm4-pre-precond-f Va t p})\ t|\!)$
apply(*simp add*:*mp-alloc-stm4-pre-precond-f-def block-num-def*)
apply(*rule subst*[**where** *s*=*buf* (*mem-pool-info Va p*) **and** *t*=*buf* (*mem-pool-info x p*)])
apply(*simp add*:*gvars-conf-stable-def gvars-conf-def set-bit-def*)

apply(*rule subst*[**where** $s=n * (\textit{max-sz} (\textit{mem-pool-info Va p}) \textit{ div } 4 \hat{}\ \textit{nat} (\textit{from-l Va t}))$ **and** $t=\textit{buf} (\textit{mem-pool-info Va p}) + n * (\textit{max-sz} (\textit{mem-pool-info Va p}) \textit{ div } 4 \hat{}\ \textit{nat} (\textit{from-l Va t})) - \textit{buf} (\textit{mem-pool-info Va p})$])
apply *arith*

apply(*subgoal-tac* $\forall ii<\textit{length} (\textit{lsizes Va t}).\ \textit{lsizes Va t} ! ii = (\textit{max-sz} (\textit{mem-pool-info}$

Va p)) *div 4 ^ ii*)
  prefer *2* **using** *inv-maxsz-align4*[*of Va*] **apply** *metis*

apply(*rule subst*[**where** *s=lsizes Va t ! nat* (*from-l Va t*) *div 4* **and** *t=lsizes Va t ! nat* (*from-l Va t + 1*)])
  apply (*smt div-mult-self1-is-m mp-alloc-stm4-blockfit-help4 nat-less-iff semiring-normalization-rules*(*7*) *zero-less-numeral*)
  by (*smt div-eq-0-iff m-mod-div mod-mult-self2-is-0 mp-alloc-stm4-blockfit-help4 nat-less-iff nonzero-mult-div-cancel-right semiring-normalization-rules*(*7*))

lemma *mp-alloc-stm4-whlpst-in-post-h2*:
$p \in$ *mem-pools Va* $\Longrightarrow$
inv Va $\Longrightarrow$
alloc-l Va t < *int* (*n-levels* (*mem-pool-info Va p*)) $\Longrightarrow$
from-l Va t < *alloc-l Va t* $\Longrightarrow$
$\neg$ *free-l Va t* < *0* $\Longrightarrow$
free-l Va t $\leq$ *from-l Va t* $\Longrightarrow$
$\forall$ *ii*<*length* (*lsizes Va t*). *lsizes Va t ! ii = ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4 ^ ii* $\Longrightarrow$
length (*lsizes Va t*) $\leq$ *n-levels* (*mem-pool-info Va p*) $\Longrightarrow$
alloc-l Va t = int (*length* (*lsizes Va t*)) − *1* $\wedge$ *length* (*lsizes Va t*) = *n-levels* (*mem-pool-info Va p*) $\vee$
alloc-l Va t = int (*length* (*lsizes Va t*)) − *2* $\wedge$ *lsizes Va t ! nat* (*alloc-l Va t + 1*) < *sz* $\Longrightarrow$
blk Va t = buf (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4 ^ nat* (*from-l Va t*)) $\Longrightarrow$
(*x*, *mp-alloc-stm4-pre-precond-f Va t p*) $\in$ *gvars-conf-stable* $\Longrightarrow$
data (*the* (*allocating-node* (*mp-alloc-stm4-pre-precond-f Va t p*) *t*)) =
buf (*mem-pool-info x p*) +
block (*the* (*allocating-node* (*mp-alloc-stm4-pre-precond-f Va t p*) *t*)) ∗
(*max-sz* (*mem-pool-info x p*) *div 4 ^ level* (*the* (*allocating-node* (*mp-alloc-stm4-pre-precond-f Va t p*) *t*)))
apply(*simp add*:*mp-alloc-stm4-pre-precond-f-def block-num-def*)

apply(*rule subst*[**where** *s=buf* (*mem-pool-info Va p*) **and** *t=buf* (*mem-pool-info x p*)])
  apply(*simp add*:*gvars-conf-stable-def gvars-conf-def set-bit-def*)

apply(*rule subst*[**where** *s=n* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4 ^ nat* (*from-l Va t*)) **and** *t=buf* (*mem-pool-info Va p*) + *n* ∗ (*max-sz* (*mem-pool-info Va p*) *div 4 ^ nat* (*from-l Va t*)) −
buf (*mem-pool-info Va p*)])
  apply *arith*

apply(*subgoal-tac* $\forall$ *ii*<*length* (*lsizes Va t*). *lsizes Va t ! ii* = (*max-sz* (*mem-pool-info Va p*)) *div 4 ^ ii*)
  prefer *2* **using** *inv-maxsz-align4*[*of Va*] **apply** *metis*

apply(*rule subst*[**where** *s=lsizes Va t ! nat (from-l Va t) div 4* **and** *t=lsizes Va t ! nat (from-l Va t + 1)*])
  apply (*smt div-mult-self1-is-m mp-alloc-stm4-blockfit-help4 nat-less-iff semiring-normalization-rules(7) zero-less-numeral*)
apply(*rule subst*[**where** *s=max-sz (mem-pool-info Va p)* **and** *t=max-sz (mem-pool-info x p)*])
  apply(*simp add:gvars-conf-stable-def gvars-conf-def set-bit-def*)
apply(*rule subst*[**where** *s=max-sz (mem-pool-info Va p) div 4 ^ nat (from-l Va t) div 4*
  and *t=max-sz (mem-pool-info Va p) div 4 ^ nat (from-l Va t + 1)*])
  apply (*metis inv-maxsz-align4 mp-alloc-stm4-blockfit-help4 nonzero-mult-div-cancel-right zero-neq-numeral*)
apply(*rule subst*[**where** *s=max-sz (mem-pool-info Va p) div 4 ^ nat (from-l Va t)*
  and *t=lsizes Va t ! nat (from-l Va t)*])
  apply (*smt nat-less-iff*)
apply(*subgoal-tac* $\exists$ *m>0. max-sz (mem-pool-info Va p) = (4 * m) * (4 ^ n-levels (mem-pool-info Va p))*)
  prefer *2* **apply**(*simp add:inv-def inv-mempool-info-def Let-def*) **apply** *metis*
  by (*smt add-left-cancel inv-maxsz-align4 mp-alloc-stm4-blockfit-help4 mult.assoc mult-is-0 nonzero-mult-div-cancel-left semiring-normalization-rules(7)*)

lemma *mp-alloc-stm4-whlpst-in-post-h3-1*:
from-l Va t $\geq$ *0* $\Longrightarrow$ *n < n-max (mem-pool-info Va p) * 4 ^ nat (from-l Va t)* $\Longrightarrow$
  *4 * n < n-max (mem-pool-info Va p) * 4 ^ nat (from-l Va t + 1)*
  by (*smt mult.assoc Divides.div-mult2-eq Suc-nat-eq-nat-zadd1 div-eq-0-iff div-mult-mult1-if gr-implies-not0 mult.commute mult-eq-0-iff power-Suc semiring-normalization-rules(7) zero-less-numeral*)

lemma *mp-alloc-stm4-whlpst-in-post-h3*:
p $\in$ *mem-pools Va* $\Longrightarrow$
inv Va $\Longrightarrow$
alloc-l Va t < int (n-levels (mem-pool-info Va p)) $\Longrightarrow$
from-l Va t < alloc-l Va t $\Longrightarrow$
$\neg$ *free-l Va t < 0* $\Longrightarrow$
free-l Va t $\leq$ *from-l Va t* $\Longrightarrow$
$\forall$ *ii<length (lsizes Va t). lsizes Va t ! ii = ALIGN4 (max-sz (mem-pool-info Va p)) div 4 ^ ii* $\Longrightarrow$
length (lsizes Va t) $\leq$ *n-levels (mem-pool-info Va p)* $\Longrightarrow$
alloc-l Va t = int (length (lsizes Va t)) − 1 $\wedge$ *length (lsizes Va t) = n-levels (mem-pool-info Va p)* $\vee$
  alloc-l Va t = int (length (lsizes Va t)) − 2 $\wedge$ *lsizes Va t ! nat (alloc-l Va t + 1) < sz* $\Longrightarrow$
*n < n-max (mem-pool-info Va p) * 4 ^ nat (from-l Va t)* $\Longrightarrow$
(x, mp-alloc-stm4-pre-precond-f Va t p) $\in$ *gvars-conf-stable* $\Longrightarrow$
*blk Va t = buf (mem-pool-info Va p) + n * (max-sz (mem-pool-info Va p) div 4 ^ nat (from-l Va t))* $\Longrightarrow$

block (*the* (*allocating-node* (*mp-alloc-stm4-pre-precond-f Va t p*) *t*))
$<$ *n-max* (*mem-pool-info x p*) $*$ *4* ˆ *level* (*the* (*allocating-node* (*mp-alloc-stm4-pre-precond-f Va t p*) *t*))
apply(*simp add:mp-alloc-stm4-pre-precond-f-def block-num-def*)
apply(*rule subst*[**where** *s=max-sz* (*mem-pool-info Va p*) *div 4* ˆ *nat* (*from-l Va t*)
and *t=lsizes Va t* ! *nat* (*from-l Va t*)])
using *inv-maxsz-align4* **apply** *auto*[*1*]
apply(*rule subst*[**where** *s=n-max* (*mem-pool-info Va p*) **and** *t=n-max* (*mem-pool-info x p*)])
apply(*simp add:mp-alloc-stm4-pre-precond-f-def set-bit-def gvars-conf-stable-def gvars-conf-def*)
apply(*subgoal-tac* $\exists$ *m*>*0*. *max-sz* (*mem-pool-info Va p*) = (*4* $*$ *m*) $*$ (*4* ˆ *n-levels* (*mem-pool-info Va p*)))
prefer *2* **apply**(*simp add:inv-def inv-mempool-info-def Let-def*) **apply** *metis*
apply(*subgoal-tac nat* (*from-l Va t*) $<$ *n-levels* (*mem-pool-info Va p*)) **prefer** *2*
apply *linarith*

apply(*rule subst*[**where** *s=n* **and** *t=*(*n* $*$ (*max-sz* (*mem-pool-info Va p*) *div 4* ˆ *nat* (*from-l Va t*)) *div*
(*max-sz* (*mem-pool-info Va p*) *div 4* ˆ *nat* (*from-l Va t*)))])
apply (*simp add*: *mp-alloc-stm3-lm2-inv-1-2*)
apply *clarsimp*

apply(*rule mp-alloc-stm4-whlpst-in-post-h3-1*)
apply *arith* **apply** *blast*

done

lemma *mp-alloc-stm4-whlpst-in-post-h4*:
p $\in$ *mem-pools Va* $\Longrightarrow$
inv Va $\Longrightarrow$
alloc-l Va t $<$ *int* (*n-levels* (*mem-pool-info Va p*)) $\Longrightarrow$
from-l Va t $<$ *alloc-l Va t* $\Longrightarrow$
$\neg$ *free-l Va t* $<$ *0* $\Longrightarrow$
free-l Va t $\leq$ *from-l Va t* $\Longrightarrow$
$\forall$ *ii*<*length* (*lsizes Va t*). *lsizes Va t* ! *ii* = *ALIGN4* (*max-sz* (*mem-pool-info Va p*)) *div 4* ˆ *ii* $\Longrightarrow$
length (*lsizes Va t*) $\leq$ *n-levels* (*mem-pool-info Va p*) $\Longrightarrow$
alloc-l Va t = *int* (*length* (*lsizes Va t*)) − *1* $\wedge$ *length* (*lsizes Va t*) = *n-levels* (*mem-pool-info Va p*) $\vee$
alloc-l Va t = *int* (*length* (*lsizes Va t*)) − *2* $\wedge$ *lsizes Va t* ! *nat* (*alloc-l Va t* + *1*) $<$ *sz* $\Longrightarrow$
n $<$ *n-max* (*mem-pool-info Va p*) $*$ *4* ˆ *nat* (*from-l Va t*) $\Longrightarrow$
blk Va t = *buf* (*mem-pool-info Va p*) + *n* $*$ (*max-sz* (*mem-pool-info Va p*) *div 4* ˆ *nat* (*from-l Va t*)) $\Longrightarrow$
(*x*, *mp-alloc-stm4-pre-precond-f Va t p*) $\in$ *gvars-conf-stable* $\Longrightarrow$
($\exists$ *n*<*n-max* (*mem-pool-info x p*) $*$ *4* ˆ *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*).

blk (*mp-alloc-stm4-pre-precond-f Va t p*) *t* =
buf (*mem-pool-info x p*) +
n ∗ (*max-sz* (*mem-pool-info x p*) *div 4* ^ *nat* (*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* + *1*)))
apply(*rule subst*[**where** *s*=*n-max* (*mem-pool-info Va p*) **and** *t*=*n-max* (*mem-pool-info x p*)])
apply(*simp add*:*gvars-conf-stable-def gvars-conf-def mp-alloc-stm4-pre-precond-f-def set-bit-def*)
apply(*rule subst*[**where** *s*=*buf* (*mem-pool-info Va p*) **and** *t*=*buf* (*mem-pool-info x p*)])
apply(*simp add*:*gvars-conf-stable-def gvars-conf-def mp-alloc-stm4-pre-precond-f-def set-bit-def*)
apply(*rule subst*[**where** *s*=*from-l Va* **and** *t*=*from-l* (*mp-alloc-stm4-pre-precond-f Va t p*)])
apply(*simp add*:*mp-alloc-stm4-pre-precond-f-def*)
apply(*rule subst*[**where** *s*=*blk Va* **and** *t*=*blk* (*mp-alloc-stm4-pre-precond-f Va t p*)])
apply(*simp add*:*mp-alloc-stm4-pre-precond-f-def*)
apply(*rule subst*[**where** *s*=*max-sz* (*mem-pool-info Va p*) **and** *t*=*max-sz* (*mem-pool-info x p*)])
apply(*simp add*:*gvars-conf-stable-def gvars-conf-def mp-alloc-stm4-pre-precond-f-def set-bit-def*)
apply(*rule exI*[**where** *x*=*4* ∗ *n*])
by (*smt inv-maxsz-align4 mp-alloc-stm4-blockfit-help4 mp-alloc-stm4-whlpst-in-post-h3-1*
mult.assoc semiring-normalization-rules(*7*))

lemma *mp-alloc-stm4-whlpst-in-post*:
Va ∈ *mp-alloc-precond2-1-1-loopinv-0 t p sz timeout* ∩ {|´*cur* = *Some t*|} ⟹
mp-alloc-stm4-loopinv (*mp-alloc-stm4-pre-precond-f Va t p*) *t p* ∩ {|´*i t* ≥ *4*|}
⊆ {|´(*Pair Va*) ∈ *Mem-pool-alloc-guar t*|} ∩ *mp-alloc-precond2-1-1-loopinv-1 t p sz timeout*
apply *clarsimp*
apply(*rule conjI*)
apply(*simp add*:*Mem-pool-alloc-guar-def*) **apply** *clarsimp*
apply(*rule conjI*)
apply(*simp add*:*gvars-conf-stable-def gvars-conf-def*)
apply(*rule conjI*) **using** *mp-alloc-stm4-mempools2* **apply** *metis*
apply *clarify*
apply(*rule conjI*) **using** *mp-alloc-stm4-inv-mif-buf* **apply** *metis*
apply(*rule conjI*) **using** *mp-alloc-stm4-inv-mif-mxsz* **apply** *metis*
apply(*rule conjI*) **using** *mp-alloc-stm4-inv-mif-nmax* **apply** *metis*
apply(*rule conjI*) **using** *mp-alloc-stm4-inv-mif-nlvls* **apply** *metis*
apply(*rule conjI*) **using** *mp-alloc-stm4-inv-mif-len* **apply** *metis*
apply *clarify* **using** *mp-alloc-stm4-inv-bits-len* **apply** *metis*
apply(*rule conjI*)
using *mp-alloc-stm4-whlpst-in-post-inv*[*of Va t p timeout - sz -*] **apply** *auto*[*1*]
apply(*rule conjI*)

apply *clarsimp*
apply(*subgoal-tac lvars-nochange t′ x* (*mp-alloc-stm4-pre-precond-f Va t p*))
prefer *2* **apply** *metis*
apply(*subgoal-tac lvars-nochange t′ Va* (*mp-alloc-stm4-pre-precond-f Va t p*))
prefer *2* **using** *mp-alloc-stm4-pre-precond-f-lvars-nochange*[*of - t Va p*] **apply** *metis*
using *lvars-nochange-trans*[*of - Va mp-alloc-stm4-pre-precond-f Va t p -*]
lvars-nochange-sym **apply** *metis*
using *mp-alloc-stm4-pre-precond-f-tick* **apply** *metis*

apply(*rule conjI*)
apply *clarsimp*
using *mp-alloc-stm4-whlpst-in-post-inv*[*of Va t p timeout - sz -*] **apply** *auto*[*1*]

apply(*rule conjI*)
apply *clarsimp*
using *mp-alloc-stm4-pre-precond-f-def-frnode* **apply** *metis*

apply(*rule conjI*)
apply *clarsimp*
using *mp-alloc-stm4-pre-precond-f-mpls* **apply** *metis*

apply(*rule conjI*)
apply *clarsimp*
apply(*rule conjI*) **apply** *clarsimp*
apply(*subgoal-tac rf Va t*) **prefer** *2* **using** *mp-alloc-stm4-pre-precond-f-rf* **apply** *metis*
apply *fast*
apply(*rule conjI*) **apply** *clarsimp*
using *mp-alloc-stm4-pre-precond-f-ret* **apply** *metis*
apply *clarsimp* **using** *mp-alloc-stm4-pre-precond-f-tmout* **apply** *metis*

apply(*rule conjI*)
apply *clarsimp*
apply(*subgoal-tac rf Va t*) **prefer** *2* **using** *mp-alloc-stm4-pre-precond-f-rf* **apply** *metis*
apply *fast*

apply(*rule conjI*)
apply *clarsimp*
apply(*rule conjI*)
apply *clarsimp*
apply(*subgoal-tac max-sz* (*mem-pool-info x p*) = *max-sz* (*mem-pool-info Va p*))
prefer *2* **apply**(*subgoal-tac max-sz* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*)
= *max-sz* (*mem-pool-info Va p*))
prefer *2* **using** *mp-alloc-stm4-pre-maxsz* **apply** *metis*
apply(*simp add*:*gvars-conf-stable-def gvars-conf-def*)
apply(*subgoal-tac lsizes* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* = *lsizes Va t*)

prefer *2* **using** *mp-alloc-stm4-pre-precond-f-lsz* **apply** *metis*
apply *metis*
apply(*subgoal-tac n-levels* (*mem-pool-info x p*) = *n-levels* (*mem-pool-info Va p*))
prefer *2* **apply**(*subgoal-tac n-levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*)
= *n-levels* (*mem-pool-info Va p*))
prefer *2* **using** *mp-alloc-stm4-inv-mif-nlvls* **apply** *metis*
apply(*simp add*:*gvars-conf-stable-def gvars-conf-def*)
apply(*rule conjI*)
apply(*subgoal-tac lsizes* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* = *lsizes Va t*)
prefer *2* **using** *mp-alloc-stm4-pre-precond-f-lsz* **apply** *metis*
apply *metis*
apply(*rule conjI*)
apply(*subgoal-tac alloc-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* = *alloc-l Va t*)

prefer *2* **using** *mp-alloc-stm4-pre-precond-f-allocl* **apply** *metis*
apply *metis*
apply(*rule conjI*)
apply(*rule subst*[**where** *t*= *free-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* **and** *s*= *free-l Va t*])
using *mp-alloc-stm4-pre-precond-f-freel* **apply** *metis*
apply *linarith*
apply(*rule conjI*)
apply(*subgoal-tac alloc-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* = *alloc-l Va t*)

prefer *2* **using** *mp-alloc-stm4-pre-precond-f-allocl* **apply** *metis*
apply(*rule subst*[**where** *t*= *free-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* **and** *s*= *free-l Va t*])
using *mp-alloc-stm4-pre-precond-f-freel* **apply** *metis*
apply *linarith*

apply(*rule disjI2*)
apply(*rule subst*[**where** *s*=*alloc-l Va* **and** *t*=*alloc-l* (*mp-alloc-stm4-pre-precond-f Va t p*)])
using *mp-alloc-stm4-pre-precond-f-allocl* **apply** *metis*
apply(*rule subst*[**where** *s*=*lsizes Va* **and** *t*=*lsizes* (*mp-alloc-stm4-pre-precond-f Va t p*)])
using *mp-alloc-stm4-pre-precond-f-lsz* **apply** *metis*
apply(*rule conjI*) **apply** *linarith*
apply(*rule conjI*) **apply** *blast*
apply(*subgoal-tac n-levels* (*mem-pool-info x p*) = *n-levels* (*mem-pool-info Va p*))
prefer *2* **apply**(*subgoal-tac n-levels* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p*) *p*)
= *n-levels* (*mem-pool-info Va p*))
prefer *2* **using** *mp-alloc-stm4-inv-mif-nlvls* **apply** *metis*
apply(*simp add*:*gvars-conf-stable-def gvars-conf-def*)
apply *metis*

apply(*rule conjI*)

apply *clarsimp*
apply(*subgoal-tac alloc-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* = *alloc-l Va t*)
prefer *2* **using** *mp-alloc-stm4-pre-precond-f-allocl* **apply** *metis*
apply *arith*

apply(*rule conjI*)
apply *clarsimp*
apply(*subgoal-tac free-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* = *free-l Va t*)
prefer *2* **using** *mp-alloc-stm4-pre-precond-f-freel* **apply** *metis*
apply *arith*

apply(*rule conjI*)
apply *clarsimp*
apply(*subgoal-tac blk Va t* > *0*) **prefer** *2*
apply(*simp add*:*inv-def inv-mempool-info-def*)
apply(*subgoal-tac blk* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* = *blk Va t*)
prefer *2* **using** *mp-alloc-stm4-pre-precond-f-blk* **apply** *metis*
apply *arith*

apply(*rule conjI*)
apply *clarsimp*
apply(*subgoal-tac alloc-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* = *alloc-l Va t*)
prefer *2* **using** *mp-alloc-stm4-pre-precond-f-allocl* **apply** *metis*
apply(*subgoal-tac from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* = *from-l Va t*)
prefer *2* **using** *mp-alloc-stm4-pre-precond-f-froml* **apply** *metis*
apply *arith*

apply *clarsimp*
apply(*rule conjI*)
apply(*subgoal-tac alloc-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* = *alloc-l Va t*)
prefer *2* **using** *mp-alloc-stm4-pre-precond-f-allocl* **apply** *metis*
apply(*subgoal-tac from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* = *from-l Va t*)
prefer *2* **using** *mp-alloc-stm4-pre-precond-f-froml* **apply** *metis*
apply *arith*

apply(*rule conjI*)
apply(*subgoal-tac from-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* = *from-l Va t*)
prefer *2* **using** *mp-alloc-stm4-pre-precond-f-froml* **apply** *metis*
apply(*subgoal-tac free-l* (*mp-alloc-stm4-pre-precond-f Va t p*) *t* = *free-l Va t*)
prefer *2* **using** *mp-alloc-stm4-pre-precond-f-freel* **apply** *metis*
apply *arith*

apply(*rule conjI*)
using *mp-alloc-stm4-whlpst-in-post-h1* **apply** *blast*

apply(*rule conjI*)
using *mp-alloc-stm4-whlpst-in-post-h2* **apply** *blast*

apply(*rule conjI*)

using *mp-alloc-stm4-whlpst-in-post-h3* **apply** *blast*

using *mp-alloc-stm4-whlpst-in-post-h4* **apply** *blast*
done

lemma *thd-state* (*mp-alloc-stm4-pre-precond-f Va t p*) = *thd-state Va*
by(*simp add*:*mp-alloc-stm4-pre-precond-f-def*)

lemma *thd-state* (*mp-alloc-stm4-pre-precond-f Va t p*) = *thd-state Va*
by(*simp add*:*mp-alloc-stm4-pre-precond-f-def*)

lemma ∀ *p*∈*mem-pools Va*. *wait-q* (*mem-pool-info Va p*) = *wait-q* (*mem-pool-info* (*mp-alloc-stm4-pre-precond-f Va t p1*) *p*)
apply *clarify*
apply(*simp add*:*mp-alloc-stm4-pre-precond-f-def*)
apply(*simp add*: *set-bit-def*)
done

term *mp-alloc-precond2-1-1-loopinv-0 t p sz timeout*
term *mp-alloc-precond2-1-1-loopinv-1 t p sz timeout*

lemma *mp-alloc-stm4-lm1-1*:
Va ∈ *mp-alloc-precond2-1-1-loopinv-0 t p sz timeout* ∩ ⦃´*cur* = *Some t*⦄ ⟹
Γ $\vdash_I$ *Some* (´*bn* := ´*bn* (*t* := *block-num* (´*mem-pool-info p*) (´*blk t*) ((´*lsizes t*)!(*nat* (´*from-l t*))));;
´*mem-pool-info* := *set-bit-divide* ´*mem-pool-info p* (*nat* (´*from-l t*)) (´*bn t*);;
´*mem-pool-info* := *set-bit-allocating* ´*mem-pool-info p* (*nat* (´*from-l t* + *1*)) (*4* ∗ ´*bn t*);;
´*allocating-node* := ´*allocating-node* (*t* := *Some* (|*pool* = *p*, *level* = *nat* (´*from-l t* + *1*),
block = *4* ∗ ´*bn t*, *data* = ´*blk t* |));;
FOR ´*i* := ´*i* (*t* := *Suc 0*);
´*i t* < *4*;
´*i* := ´*i* (*t* := *Suc* (´*i t*)) *DO*
´*lbn* := ´*lbn* (*t* := *4* ∗ ´*bn t* + ´*i t*);;
´*lsz* := ´*lsz* (*t* := (´*lsizes t*) ! (*nat* (´*from-l t* + *1*)));;
´*block2* := ´*block2*(*t* := ´*lsz t* ∗ ´*i t* + ´*blk t*);;
´*mem-pool-info* := *set-bit-free* ´*mem-pool-info p* (*nat* (´*from-l t* + *1*)) (´*lbn t*);;

IF block-fits (´*mem-pool-info p*) (´*block2 t*) (´*lsz t*) *THEN*
´*mem-pool-info* := ´*mem-pool-info* (*p* :=
append-free-list (´*mem-pool-info p*) (*nat* (´*from-l t* + *1*)) (´*block2 t*))
FI
ROF) sat_p [{*Va*}, {(*s*, *t*). *s* = *t*}, *UNIV*,
⦃´(*Pair Va*) ∈ *Mem-pool-alloc-guar t*⦄ ∩ *mp-alloc-precond2-1-1-loopinv-1 t p sz timeout*]

apply(*rule Seq*[**where** *mid*={*mp-alloc-stm4-pre-precond4*
(*mp-alloc-stm4-pre-precond3*
(*mp-alloc-stm4-pre-precond2*
(*mp-alloc-stm4-pre-precond1 Va t p*) *t p*) *t p*) *t p*}])
apply(*rule Seq*[**where** *mid*={*mp-alloc-stm4-pre-precond3*
(*mp-alloc-stm4-pre-precond2*
(*mp-alloc-stm4-pre-precond1 Va t p*) *t p*) *t p*}])
apply(*rule Seq*[**where** *mid*={*mp-alloc-stm4-pre-precond2*
(*mp-alloc-stm4-pre-precond1 Va t p*) *t p*}])
apply(*rule Seq*[**where** *mid*={*mp-alloc-stm4-pre-precond1 Va t p*}])

apply(*rule Basic*)
apply *simp* **apply** *simp* **apply**(*simp add*:*stable-def*) **apply**(*simp add*:*stable-def*)

apply(*rule Basic*)
apply *simp* **apply** *simp* **apply**(*simp add*:*stable-def*) **apply**(*simp add*:*stable-def*)

apply(*rule Basic*)
apply *simp* **apply** *simp* **apply**(*simp add*:*stable-def*) **apply**(*simp add*:*stable-def*)

apply(*rule Basic*)
apply *simp* **apply** *simp* **apply**(*simp add*:*stable-def*) **apply**(*simp add*:*stable-def*)

apply(*rule Seq*[**where** *mid*={*mp-alloc-stm4-pre-precond-f Va t p*}])
apply(*rule Basic*)
apply(*simp add*:*mp-alloc-stm4-pre-precond-f-def*) **apply** *simp* **apply**(*simp add*:*stable-def*)
apply(*simp add*:*stable-def*)

apply(*rule Conseq*[**where** *pre*={*mp-alloc-stm4-pre-precond-f Va t p*}
and *pre′*=*mp-alloc-stm4-loopinv* (*mp-alloc-stm4-pre-precond-f Va t p*) *t p*
and *rely*={(*s*, *t*).*s* = *t*} **and** *rely′*={(*s*, *t*).*s* = *t*} **and** *guar*=*UNIV* **and** *guar′*=*UNIV*

and *post′*=*mp-alloc-stm4-loopinv* (*mp-alloc-stm4-pre-precond-f Va t p*) *t p* ∩ ⦃´*i t* ≥ *4*⦄])
using *mp-alloc-stm4-pre-precond-f-in-mp-alloc-stm4-loopinv* **apply** *auto*[*1*]
apply *simp* **apply** *simp* **using** *mp-alloc-stm4-whlpst-in-post*[*of Va t p timeout sz*] **apply** *argo*
using *mp-alloc-stm4-while*[*of Va t p timeout sz*] **apply** *fastforce*
done

term *mp-alloc-precond2-1-1-loopinv-0 t p sz timeout* ∩ ⦃´*cur* = *Some t*⦄
term *mp-alloc-precond2-1-1-loopinv-1 t p sz timeout*
term ⦃´(*Pair Va*) ∈ *Mem-pool-alloc-guar t*⦄ ∩ *mp-alloc-precond2-1-1-loopinv-1 t p sz timeout*

lemma *mp-alloc-stm4-lm1*:

mp-alloc-precond2-1-1-loopinv-0 t p sz timeout ∩ {|´*cur* = *Some t*|} ∩ {*Va*} = {*Va*} ⟹
  Γ $\vdash_I$ *Some* (´*bn* := ´*bn*(*t* := *block-num* (´*mem-pool-info p*) (´*blk t*) (´*lsizes t* ! *nat* (´*from-l t*)));;
    ´*mem-pool-info* := *set-bit-divide* ´*mem-pool-info p* (*nat* (´*from-l t*)) (´*bn t*);;
    ´*mem-pool-info* := *set-bit-allocating* ´*mem-pool-info p* (*nat* (´*from-l t* + *1*)) (*4* ∗ ´*bn t*);;
    ´*allocating-node* := ´*allocating-node*(*t* ↦ (|*pool* = *p*, *level* = *nat* (´*from-l t* + *1*), *block* = *4* ∗ ´*bn t*, *data* = ´*blk t*|));;
    (´*i* := ´*i*(*t* := *Suc NULL*);;
     WHILE ´*i t* < *4*
     DO ´*lbn* := ´*lbn*(*t* := *4* ∗ ´*bn t* + ´*i t*);; ´*lsz* := ´*lsz*(*t* := ´*lsizes t* ! *nat* (´*from-l t* + *1*));;
       ´*block2* := ´*block2*(*t* := ´*lsz t* ∗ ´*i t* + ´*blk t*);;
        ´*mem-pool-info* := *set-bit-free* ´*mem-pool-info p* (*nat* (´*from-l t* + *1*)) (´*lbn t*);;
       IF block-fits (´*mem-pool-info p*) (´*block2 t*)
         (´*lsz t*) *THEN* ´*mem-pool-info* := ´*mem-pool-info*
              (*p* := *append-free-list* (´*mem-pool-info p*) (*nat* (´*from-l t* + *1*)) (´*block2 t*)) *FI*;;
       ´*i* := ´*i*(*t* := *Suc* (´*i t*))
     OD)) sat_p [*mp-alloc-precond2-1-1-loopinv-0 t p sz timeout* ∩ {|´*cur* = *Some t*|} ∩ {*Va*},
        {(*s*, *t*). *s* = *t*}, *UNIV*,
      {|´(*Pair Va*) ∈ *Mem-pool-alloc-guar t*|} ∩ *mp-alloc-precond2-1-1-loopinv-1 t p sz timeout*]
   apply(*rule subst*[**where** *t*=*mp-alloc-precond2-1-1-loopinv-0 t p sz timeout* ∩ {|´*cur* = *Some t*|} ∩ {*Va*} **and** *s*={*Va*}])
  apply *metis*
  apply(*subgoal-tac Va* ∈ *mp-alloc-precond2-1-1-loopinv-0 t p sz timeout* ∩ {|´*cur* = *Some t*|})
   prefer *2* **apply** *auto*[*1*]
  using *mp-alloc-stm4-lm1-1* **apply** *meson*
done

term *mp-alloc-precond2-1-1-loopinv t p sz timeout*
term *mp-alloc-precond2-1-2 t p sz timeout*

lemma *mp-alloc-stm4-lm*:
  Γ $\vdash_I$ *Some* (*WHILE* ´*from-l t* < ´*alloc-l t DO*
    (∗ ==== *start*: *blk* = *break-block*(*p*, *blk*, *from-l*, *lsizes*); ∗)
   (*t* ► *ATOMIC*
     ´*bn* := ´*bn* (*t* := *block-num* (´*mem-pool-info p*) (´*blk t*) ((´*lsizes t*)!(*nat* (´*from-l t*))));;

    ´*mem-pool-info* := *set-bit-divide* ´*mem-pool-info p* (*nat* (´*from-l t*)) (´*bn t*);;

    ´*mem-pool-info* := *set-bit-allocating* ´*mem-pool-info p* (*nat* (´*from-l t* + *1*)) (*4* ∗ ´*bn t*);;

(* *set the allocating node info of the thread* *)
´allocating-node := *´allocating-node* (t := *Some* (|*pool* = p, *level* = *nat* (*´from-l* t + *1*),
block = *4* ∗ *´bn* t, *data* = *´blk* t |));;

FOR ´i := *´i* (t := *1*);
´i t < *4*;
´i := *´i* (t := *´i* t + *1*) *DO*
´lbn := *´lbn* (t := *4* ∗ *´bn* t + *´i* t);;
´lsz := *´lsz* (t := (*´lsizes* t) ! (*nat* (*´from-l* t + *1*)));;
´block2 := *´block2*(t := *´lsz* t ∗ *´i* t + *´blk* t);;

(* *set-free-bit*(p, l + *1*, *lbn*); *)
´mem-pool-info := *set-bit-free ´mem-pool-info* p (*nat* (*´from-l* t + *1*)) (*´lbn* t);;

IF block-fits (*´mem-pool-info* p) (*´block2* t) (*´lsz* t) *THEN*

(* *sys-dlist-append*(&p−>*levels*[l + *1*].*free-list*, *block2*); *)
´mem-pool-info := *´mem-pool-info* (p :=
append-free-list (*´mem-pool-info* p) (*nat* (*´from-l* t + *1*)) (*´block2* t))
FI
ROF

END);;
(t ► *´from-l* := *´from-l*(t := *´from-l* t + *1*))
OD) sat_p [*mp-alloc-precond2-1-1-loopinv* t p *sz timeout*, *Mem-pool-alloc-rely* t, *Mem-pool-alloc-guar* t,
mp-alloc-precond2-1-2 t p *sz timeout*]
apply(*rule While*)
using *mp-alloc-precond2-1-1-loopinv-stb* **apply** *simp*
apply(*rule Int-greatest*) **apply**(*rule Int-greatest*) **apply**(*rule Int-greatest*)
apply(*rule Int-greatest*) **apply**(*rule Int-greatest*) **apply**(*rule Int-greatest*)
apply(*rule Int-greatest*) **apply**(*rule Int-greatest*)
apply *auto*[*1*] **apply** *auto*[*1*] **apply** *auto*[*1*] **apply** *auto*[*1*] **apply** *auto*[*1*]
apply *auto*[*1*] **apply** *auto*[*1*] **apply** *auto*[*1*] **apply** *clarify* **apply** *auto*[*1*] **apply** *auto*[*1*]
apply(*rule subst*[**where** t={|*´from-l* t ≤ *´alloc-l* t ∧ *´allocating-node* t =
Some (|*pool* = p, *level* = *nat* (*´from-l* t), *block* = *block-num* (*´mem-pool-info* p) (*´blk* t) (*´lsizes* t ! *nat* (*´from-l* t)),
data = *´blk* t|)|} **and** s={|*´from-l* t ≤ *´alloc-l* t|} ∩ {|*´allocating-node* t =
Some (|*pool* = p, *level* = *nat* (*´from-l* t), *block* = *block-num* (*´mem-pool-info* p) (*´blk* t) (*´lsizes* t ! *nat* (*´from-l* t)),
data = *´blk* t|)|}]) **apply** *auto*[*1*]

using *mp-alloc-precond2-1-2-stb* **apply** *simp*

apply(*rule Seq*[**where** *mid=mp-alloc-precond2-1-1-loopinv-1 t p sz timeout*])

apply(*unfold stm-def*)[*1*]
apply(*rule Await*)
using *mp-alloc-precond2-1-1-loopinv-0-stb* **apply** *auto*[*1*]
using *mp-alloc-precond2-1-1-loopinv-1-stb* **apply** *simp*
apply *clarify*
apply(*rule Await*)
using *stable-id2* **apply** *fast* **using** *stable-id2* **apply** *fast*
apply *clarify*
apply(*case-tac V = Va*) **prefer** *2* **apply** *simp* **using** *Emptyprecond* **apply** *auto*[*1*]
apply *simp*
apply(*case-tac mp-alloc-precond2-1-1-loopinv-0 t p sz timeout* ∩ {|´*cur* = *Some t*|} ∩ {*Va*} = {})
using *Emptyprecond*[*of* - {(*s*, *t*). *s* = *t*} *UNIV*] **apply** *auto*[*1*]
apply(*subgoal-tac mp-alloc-precond2-1-1-loopinv-0 t p sz timeout* ∩ {|´*cur* = *Some t*|} ∩ {*Va*} = {*Va*})
prefer *2* **using** *int1-eq*[**where** *P=mp-alloc-precond2-1-1-loopinv-0 t p sz timeout* ∩ {|´*cur* = *Some t*|}] **apply** *meson*
using *mp-alloc-stm4-lm1*[*of t p timeout sz*] **apply** *auto*[*1*]

apply(*unfold stm-def*)
apply(*rule Await*)
using *mp-alloc-precond2-1-1-loopinv-1-stb* **apply** *simp*
using *mp-alloc-precond2-1-1-loopinv-stb* **apply** *auto*[*1*]
apply *clarify*
apply(*rule Basic*)
apply(*case-tac mp-alloc-precond2-1-1-loopinv-1 t p sz timeout* ∩ {|´*cur* = *Some t*|} ∩ {*V*} = {})
apply *auto*[*1*]
apply(*subgoal-tac mp-alloc-precond2-1-1-loopinv-1 t p sz timeout* ∩ {|´*cur* = *Some t*|} ∩ {*V*} = {*V*})
prefer *2* **using** *int1-eq*[**where** *P=mp-alloc-precond2-1-1-loopinv-1 t p sz timeout* ∩ {|´*cur* = *Some t*|}] **apply** *meson*
apply *simp*
apply(*rule conjI*) **apply**(*simp add:Mem-pool-alloc-guar-def*) **apply**(*rule disjI1*)
apply(*rule conjI*) **apply**(*simp add:gvars-conf-stable-def gvars-conf-def*)
apply(*rule conjI*) **apply**(*subgoal-tac* (*V*,*V*(|*from-l* := (*from-l V*)(*t* := *from-l V t* + *1*)|))∈*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add:lvars-nochange1-4all-def lvars-nochange1-def*)
apply(*simp add:lvars-nochange-def*)
apply(*rule conjI*) **apply**(*subgoal-tac* (*V*,*V*(|*from-l* := (*from-l V*)(*t* := *from-l V t* + *1*)|))∈*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add:lvars-nochange1-4all-def lvars-nochange1-def*)

apply(*rule conjI*) **apply** *auto*[*1*]
apply(*rule conjI*) **apply** (*metis less-minus-one-simps*(*1*))
apply(*rule conjI*) **apply** *smt*
apply (*metis* (*no-types*, *hide-lams*) *Mem-block.simps*(*2*) *Mem-block.simps*(*3*) *Mem-block.simps*(*4*) *option.sel*)

apply *simp* **using** *stable-id2* **apply** *blast* **using** *stable-id2* **apply** *blast*

apply(*simp add*:*Mem-pool-alloc-guar-def*)
done

10.8 stm5

lemma *mp-alloc-stm5-lm-1-inv-mempool-info*:
free-l V t $\leq$ *alloc-l V t* $\Longrightarrow$
alloc-l V t $<$ *int* (*n-levels* (*mem-pool-info V p*)) $\Longrightarrow$
p $\in$ *mem-pools V* $\Longrightarrow$
inv-mempool-info V $\Longrightarrow$
$\neg$ *free-l V t* $<$ *OK* $\Longrightarrow$
NULL $<$ *blk V t* $\Longrightarrow$
inv-mempool-info
(*V*(|*mem-pool-info* := (*mem-pool-info V*)
(*p* := *mem-pool-info V p*
(|*levels* := *levels* (*mem-pool-info V p*)
[*nat* (*alloc-l V t*) := (*levels* (*mem-pool-info V p*) ! *nat* (*alloc-l V t*))
(|*bits* := *bits* (*levels* (*mem-pool-info V p*) ! *nat* (*alloc-l V t*))
[(*blk V t* $-$ *buf* (*mem-pool-info V p*)) *div lsizes V t* ! *nat* (*alloc-l V t*) := *ALLOCATED*]|)]|)),
allocating-node := (*allocating-node V*)(*t* := *None*)|))
apply(*simp add*:*inv-mempool-info-def*)
apply(*rule conjI*) **apply** *metis*
apply(*rule conjI*) **apply** *metis*
apply(*rule conjI*) **apply** *metis*
apply(*rule conjI*) **apply** *metis*
apply *clarify* **apply**(*rename-tac ii*) **apply**(*subgoal-tac length* (*bits* (*levels* (*mem-pool-info V p*)
[*nat* (*alloc-l V t*) := (*levels* (*mem-pool-info V p*) ! *nat* (*alloc-l V t*))
(|*bits* := *bits* (*levels* (*mem-pool-info V p*) ! *nat* (*alloc-l V t*))
[(*blk V t* $-$ *buf* (*mem-pool-info V p*)) *div lsizes V t* ! *nat* (*alloc-l V t*) := *ALLOCATED*]|)] !
ii))=*length* (*bits* (*levels* (*mem-pool-info V p*) ! *ii*)))
prefer *2* **apply**(*case-tac ii* = *nat* (*alloc-l V t*)) **apply** *force* **apply** *force*
apply *metis*
done

lemma *mp-alloc-stm5-lm-1-inv-bitmap-h1*:
allocating-node V t =
Some (|*pool* = *p*, *level* = *nat* (*alloc-l V t*), *block* = (*blk V t* $-$ *buf* (*mem-pool-info*

V p)) *div lsizes V t* ! *nat* (*alloc-l V t*), *data* = *blk V t*⦈ ⟹
  ∀ *t n*. *allocating-node V t* = *Some n* ⟶ *get-bit-s V* (*pool n*) (*level n*) (*block n*) = *ALLOCATING* ⟹
    get-bit-s V p (*nat* (*alloc-l V t*)) ((*blk V t* − *buf* (*mem-pool-info V p*)) *div lsizes V t* ! *nat* (*alloc-l V t*)) = *ALLOCATING*
by *fastforce*

lemma *mp-alloc-stm5-lm-1-inv-bitmap-freelist*:
allocating-node V t =
 Some ⦇*pool* = *p*, *level* = *nat* (*alloc-l V t*), *block* = (*blk V t* − *buf* (*mem-pool-info V p*)) *div lsizes V t* ! *nat* (*alloc-l V t*), *data* = *blk V t*⦈ ⟹
 alloc-l V t < *int* (*n-levels* (*mem-pool-info V p*)) ⟹
 p ∈ *mem-pools V* ⟹
 inv-mempool-info V ∧ *inv-aux-vars V* ∧ *inv-bitmap-freelist V* ⟹
 inv-bitmap-freelist
  (*V*⦇*mem-pool-info* := (*mem-pool-info V*)
    (*p* := *mem-pool-info V p*
     ⦇*levels* := *levels* (*mem-pool-info V p*)
      [*nat* (*alloc-l V t*) := (*levels* (*mem-pool-info V p*) ! *nat* (*alloc-l V t*))
       ⦇*bits* := *bits* (*levels* (*mem-pool-info V p*) ! *nat* (*alloc-l V t*))
        [(*blk V t* − *buf* (*mem-pool-info V p*)) *div lsizes V t* ! *nat* (*alloc-l V t*) := *ALLOCATED*]⦈]⦈),
    allocating-node := (*allocating-node V*)(*t* := *None*)⦈)
apply(*rule subst*[**where** *s*=*inv-bitmap-freelist*
  (*V*⦇*mem-pool-info* := (*mem-pool-info V*)
    (*p* := *mem-pool-info V p*
     ⦇*levels* := *levels* (*mem-pool-info V p*)
      [*nat* (*alloc-l V t*) := (*levels* (*mem-pool-info V p*) ! *nat* (*alloc-l V t*))
       ⦇*bits* := *bits* (*levels* (*mem-pool-info V p*) ! *nat* (*alloc-l V t*))
        [(*blk V t* − *buf* (*mem-pool-info V p*)) *div lsizes V t* ! *nat* (*alloc-l V t*) := *ALLOCATED*]⦈]⦈)⦈)])
 apply(*simp add*:*inv-bitmap-freelist-def*)
apply(*rule subst*[**where** *s*=*inv-bitmap-freelist* (*set-bit-s V p* (*nat* (*alloc-l V t*))
  ((*blk V t* − *buf* (*mem-pool-info V p*)) *div lsizes V t* ! *nat* (*alloc-l V t*)) *ALLOCATED*)])
 apply(*unfold set-bit-s-def set-bit-def*)[*1*] **apply** *blast*
apply(*subgoal-tac get-bit-s V p* (*nat* (*alloc-l V t*))
        ((*blk V t* − *buf* (*mem-pool-info V p*)) *div lsizes V t* ! *nat* (*alloc-l V t*)) = *ALLOCATING*) **prefer** *2*
 apply(*subgoal-tac* ∀ *t n*. *allocating-node V t* = *Some n* ⟶ *get-bit-s V* (*pool n*) (*level n*) (*block n*) = *ALLOCATING*) **prefer** *2*
  apply(*simp add*:*inv-aux-vars-def Let-def*)
 using *mp-alloc-stm5-lm-1-inv-bitmap-h1* **apply** *blast*

using *inv-bitmap-freelist-presv-setbit-notfree*[*of p V ALLOCATED nat* (*alloc-l V t*)
  (*blk V t* − *buf* (*mem-pool-info V p*)) *div lsizes V t* ! *nat* (*alloc-l V t*)]
 apply *fastforce*

done

lemma *mp-alloc-stm5-lm-1-inv-bitmap*:
allocating-node V t =
Some (|*pool* = *p*, *level* = *nat* (*alloc-l V t*), *block* = (*blk V t* − *buf* (*mem-pool-info V p*)) *div lsizes V t* ! *nat* (*alloc-l V t*), *data* = *blk V t*|) ⟹
p ∈ *mem-pools V* ⟹
inv-bitmap V ∧ *inv-aux-vars V* ⟹
inv-bitmap
(*V*(|*mem-pool-info* := (*mem-pool-info V*)
(*p* := *mem-pool-info V p*
(|*levels* := *levels* (*mem-pool-info V p*)
[*nat* (*alloc-l V t*) := (*levels* (*mem-pool-info V p*) ! *nat* (*alloc-l V t*))
(|*bits* := *bits* (*levels* (*mem-pool-info V p*) ! *nat* (*alloc-l V t*))
[(*blk V t* − *buf* (*mem-pool-info V p*)) *div lsizes V t* ! *nat* (*alloc-l V t*) := *ALLOCATED*]|)]|)),
allocating-node := (*allocating-node V*)(*t* := *None*)|))
apply(*rule subst*[**where** *s*=*inv-bitmap*
(*V*(|*mem-pool-info* := (*mem-pool-info V*)
(*p* := *mem-pool-info V p*
(|*levels* := *levels* (*mem-pool-info V p*)
[*nat* (*alloc-l V t*) := (*levels* (*mem-pool-info V p*) ! *nat* (*alloc-l V t*))
(|*bits* := *bits* (*levels* (*mem-pool-info V p*) ! *nat* (*alloc-l V t*))
[(*blk V t* − *buf* (*mem-pool-info V p*)) *div lsizes V t* ! *nat* (*alloc-l V t*) := *ALLOCATED*]|)]|))|)])
apply(*simp add*:*inv-bitmap-def*)

apply(*rule subst*[**where** *s*=*inv-bitmap* (*set-bit-s V p* (*nat* (*alloc-l V t*))
((*blk V t* − *buf* (*mem-pool-info V p*)) *div lsizes V t* ! *nat* (*alloc-l V t*)) *ALLOCATED*)])
apply(*unfold set-bit-s-def set-bit-def*)[*1*] **apply** *blast*

apply(*subgoal-tac get-bit-s V p* (*nat* (*alloc-l V t*))
((*blk V t* − *buf* (*mem-pool-info V p*)) *div lsizes V t* ! *nat* (*alloc-l V t*)) = *ALLOCATING*) **prefer** *2*
apply(*subgoal-tac* ∀ *t n*. *allocating-node V t* = *Some n* ⟶ *get-bit-s V* (*pool n*) (*level n*) (*block n*) = *ALLOCATING*) **prefer** *2*
apply(*simp add*:*inv-aux-vars-def Let-def*)
using *mp-alloc-stm5-lm-1-inv-bitmap-h1* **apply** *blast*

using *inv-bitmap-presv-setbit*[*of V p nat* (*alloc-l V t*) (*blk V t* − *buf* (*mem-pool-info V p*)) *div lsizes V t* ! *nat* (*alloc-l V t*)
ALLOCATED set-bit-s V p (*nat* (*alloc-l V t*)) ((*blk V t* − *buf* (*mem-pool-info V p*)) *div lsizes V t* ! *nat* (*alloc-l V t*)) *ALLOCATED*]
apply *blast*
done

lemma *mp-alloc-stm5-lm-1-inv-aux-vars*:

(*blk V t* − *buf* (*mem-pool-info V p*)) *div lsizes V t* ! *nat* (*alloc-l V t*) < *n-max*
(*mem-pool-info V p*) ∗ *4* ˆ *nat* (*alloc-l V t*) ⟹
blk V t =
buf (*mem-pool-info V p*) +
(*blk V t* − *buf* (*mem-pool-info V p*)) *div lsizes V t* ! *nat* (*alloc-l V t*) ∗ (*max-sz*
(*mem-pool-info V p*) *div 4* ˆ *nat* (*alloc-l V t*)) ⟹
0 < *blk V t* ⟹
allocating-node V t =
Some (|*pool* = *p*, *level* = *nat* (*alloc-l V t*), *block* = (*blk V t* − *buf* (*mem-pool-info*
V p)) *div lsizes V t* ! *nat* (*alloc-l V t*),
data = *blk V t*|) ⟹
alloc-l V t < *int* (*n-levels* (*mem-pool-info V p*)) ⟹
p ∈ *mem-pools V* ⟹
inv-mempool-info V ∧ *inv-aux-vars V* ⟹
∀ *ii*<*length* (*lsizes V t*). *lsizes V t* ! *ii* = *ALIGN4* (*max-sz* (*mem-pool-info V p*))
div 4 ˆ *ii* ⟹
inv-aux-vars
(*V*(|*mem-pool-info* := (*mem-pool-info V*)
(*p* := *mem-pool-info V p*
(|*levels* := *levels* (*mem-pool-info V p*)
[*nat* (*alloc-l V t*) := (*levels* (*mem-pool-info V p*) ! *nat* (*alloc-l V t*))
(|*bits* := *bits* (*levels* (*mem-pool-info V p*) ! *nat* (*alloc-l V t*))
[(*blk V t* − *buf* (*mem-pool-info V p*)) *div lsizes V t* ! *nat* (*alloc-l*
V t) := *ALLOCATED*]|)]|)),
allocating-node := (*allocating-node V*)(*t* := *None*)|))
apply(*unfold inv-aux-vars-def*)
apply(*subgoal-tac get-bit-s V p* (*nat* (*alloc-l V t*))
((*blk V t* − *buf* (*mem-pool-info V p*)) *div lsizes V t* ! *nat* (*alloc-l*
V t)) = *ALLOCATING*) **prefer** *2*
using *mp-alloc-stm5-lm-1-inv-bitmap-h1* **apply** *presburger*

apply(*subgoal-tac mem-block-addr-valid V* ((|*pool* = *p*, *level* = *nat* (*alloc-l V t*),
block = (*blk V t* − *buf* (*mem-pool-info V p*)) *div lsizes V t* ! *nat*
(*alloc-l V t*), *data* = *blk V t*|))) **prefer** *2*
apply(*simp add*:*mem-block-addr-valid-def*)

apply(*rule conjI*)
apply *clarify*
apply(*subgoal-tac freeing-node V ta* = *Some n*) **prefer** *2* **apply** *force*

apply(*subgoal-tac* ¬(*pool n* = *p* ∧ *level n* = *nat* (*alloc-l V t*)
∧ *block n* = (*blk V t* − *buf* (*mem-pool-info V p*)) *div lsizes V t* ! *nat* (*alloc-l*
V t)))
prefer *2* **apply** *metis*
apply(*subgoal-tac get-bit-s V* (*pool n*) (*level n*) (*block n*) = *FREEING*) **prefer** *2*
apply *presburger*
apply(*subgoal-tac get-bit-s V* (*pool n*) (*level n*) (*block n*) = *get-bit-s*
(*V*(|*mem-pool-info* := (*mem-pool-info V*)
(*p* := *mem-pool-info V p*

(|*levels* := *levels* (*mem-pool-info* *V* *p*)
[*nat* (*alloc-l* *V* *t*) := (*levels* (*mem-pool-info* *V* *p*) ! *nat* (*alloc-l* *V* *t*))
(|*bits* := *bits* (*levels* (*mem-pool-info* *V* *p*) ! *nat* (*alloc-l* *V* *t*))
[(*blk* *V* *t* − *buf* (*mem-pool-info* *V* *p*)) *div* *lsizes* *V* *t* ! *nat* (*alloc-l* *V* *t*) := *ALLOCATED*]|)]|)),
allocating-node := (*allocating-node* *V*)(*t* := *None*)|)) (*pool* *n*) (*level* *n*) (*block* *n*)) **prefer** *2*
apply(*case-tac* *pool* *n* ≠ *p*) **apply** *force*
apply(*case-tac* *level* *n* ≠ *nat* (*alloc-l* *V* *t*)) **apply** *force*
apply(*case-tac* *block* *n* ≠ (*blk* *V* *t* − *buf* (*mem-pool-info* *V* *p*)) *div* *lsizes* *V* *t* ! *nat* (*alloc-l* *V* *t*))
apply(*case-tac* *level* *n* ≥ *length* (*levels* (*mem-pool-info* *V* (*pool* *n*)))) **apply** *fastforce*
apply *force* **apply** *blast*
apply *argo*

apply(*rule* *conjI*)
apply *clarify*
apply(*subgoal-tac* ∃ *ta*. *freeing-node* *V* *ta* = *Some* *n*) **prefer** *2*
apply(*subgoal-tac* *get-bit-s* *V* (*pool* *n*) (*level* *n*) (*block* *n*) = *FREEING*) **prefer** *2*
apply(*case-tac* *pool* *n* ≠ *p*) **apply** *force*
apply(*case-tac* *level* *n* ≠ *nat* (*alloc-l* *V* *t*)) **apply** *force*
apply(*case-tac* *block* *n* ≠ (*blk* *V* *t* − *buf* (*mem-pool-info* *V* *p*)) *div* *lsizes* *V* *t* ! *nat* (*alloc-l* *V* *t*))
apply(*case-tac* *level* *n* ≥ *length* (*levels* (*mem-pool-info* *V* (*pool* *n*))))
apply *fastforce* **apply** *force*
apply(*case-tac* *level* *n* ≥ *length* (*levels* (*mem-pool-info* *V* (*pool* *n*))))
apply *fastforce*
apply(*case-tac* *block* *n* ≥ *length* (*bits* (*levels* (*mem-pool-info* *V* *p*) ! *nat* (*alloc-l* *V* *t*))))
apply *fastforce* **apply** *fastforce*
apply(*subgoal-tac* *mem-block-addr-valid* *V* *n*) **prefer** *2*
apply(*simp* *add*:*mem-block-addr-valid-def*)
apply *blast*
apply *force*

apply(*rule* *conjI*)
apply *clarify*
apply(*subgoal-tac* *t* ≠ *ta*) **prefer** *2* **apply** *fastforce*
apply(*subgoal-tac* *allocating-node* *V* *ta* = *Some* *n*) **prefer** *2* **apply** *force*

apply(*subgoal-tac* ¬(*pool* *n* = *p* ∧ *level* *n* = *nat* (*alloc-l* *V* *t*)
∧ *block* *n* = (*blk* *V* *t* − *buf* (*mem-pool-info* *V* *p*)) *div* *lsizes* *V* *t* ! *nat* (*alloc-l* *V* *t*)))
prefer *2* **apply** (*metis* *Mem-block.select-convs*(*1*) *Mem-block.select-convs*(*2*) *Mem-block.select-convs*(*3*))
apply(*subgoal-tac* *get-bit-s* *V* (*pool* *n*) (*level* *n*) (*block* *n*) = *ALLOCATING*) **prefer** *2* **apply** *presburger*

apply(*subgoal-tac get-bit-s V (pool n) (level n) (block n) = get-bit-s*
(*V*⦇*mem-pool-info* := (*mem-pool-info V*)
(*p* := *mem-pool-info V p*
⦇*levels* := *levels* (*mem-pool-info V p*)
[*nat* (*alloc-l V t*) := (*levels* (*mem-pool-info V p*) ! *nat* (*alloc-l V t*))
⦇*bits* := *bits* (*levels* (*mem-pool-info V p*) ! *nat* (*alloc-l V t*))
[(*blk V t* − *buf* (*mem-pool-info V p*)) *div lsizes V t* ! *nat* (*alloc-l V t*) := *ALLOCATED*]⦈]⦈),
allocating-node := (*allocating-node V*)(*t* := *None*)⦈) (*pool n*) (*level n*) (*block n*)) **prefer** *2*
apply(*case-tac pool n* ≠ *p*) **apply** *force*
apply(*case-tac level n* ≠ *nat* (*alloc-l V t*)) **apply** *force*
apply(*case-tac block n* ≠ (*blk V t* − *buf* (*mem-pool-info V p*)) *div lsizes V t* ! *nat* (*alloc-l V t*))
apply(*case-tac level n* ≥ *length* (*levels* (*mem-pool-info V* (*pool n*)))) **apply** *fastforce*
apply *force* **apply** *blast*
apply *argo*

apply(*rule conjI*)
apply *clarify*
apply(*subgoal-tac nat* (*alloc-l V t*) < *length* (*levels* (*mem-pool-info V p*))) **prefer** *2*
apply(*simp add:inv-mempool-info-def Let-def*)
apply (*metis int-nat-eq of-nat-0-less-iff of-nat-less-imp-less*)
apply(*subgoal-tac* (*blk V t* − *buf* (*mem-pool-info V p*)) *div lsizes V t* ! *nat* (*alloc-l V t*)
< *length* (*bits* (*levels* (*mem-pool-info V p*) ! *nat* (*alloc-l V t*))))
prefer *2*
apply(*simp add:inv-mempool-info-def Let-def*)
apply(*subgoal-tac* ¬(*pool n* = *p* ∧ *level n* = *nat* (*alloc-l V t*)
∧ *block n* = (*blk V t* − *buf* (*mem-pool-info V p*)) *div lsizes V t* ! *nat* (*alloc-l V t*)))
prefer *2*
apply(*case-tac pool n* ≠ *p*) **apply** *fastforce*
apply(*case-tac level n* ≠ *nat* (*alloc-l V t*)) **apply** *fastforce*
apply(*case-tac block n* ≠ (*blk V t* − *buf* (*mem-pool-info V p*)) *div lsizes V t* ! *nat* (*alloc-l V t*)) **apply** *fastforce*
apply *simp*

apply(*subgoal-tac* ∃ *ta. ta* ≠ *t* ∧ *allocating-node V ta* = *Some n*) **prefer** *2*
apply(*subgoal-tac get-bit-s V (pool n) (level n) (block n)* = *ALLOCATING*) **prefer** *2*
apply(*case-tac pool n* ≠ *p*) **apply** *force*
apply(*case-tac level n* ≠ *nat* (*alloc-l V t*)) **apply** *force*
apply(*case-tac block n* ≠ (*blk V t* − *buf* (*mem-pool-info V p*)) *div lsizes V t* ! *nat* (*alloc-l V t*))
apply(*case-tac level n* ≥ *length* (*levels* (*mem-pool-info V* (*pool n*))))

apply *fastforce* **apply** *force*
apply(*case-tac level n* ≥ *length* (*levels* (*mem-pool-info V* (*pool n*))))
apply *fastforce*
apply(*case-tac block n* ≥ *length* (*bits* (*levels* (*mem-pool-info V p*) ! *nat* (*alloc-l V t*))))
apply *fastforce* **apply** *fastforce*
apply(*subgoal-tac mem-block-addr-valid V n*) **prefer** *2*
apply(*simp add*:*mem-block-addr-valid-def*)
apply (*metis Mem-block.select-convs*(*1*) *Mem-block.select-convs*(*2*) *Mem-block.select-convs*(*3*) *option.sel*)

apply *auto*[*1*]

apply(*rule conjI*)
apply *clarify*
apply *auto*[*1*]

apply(*rule conjI*)
apply *clarify*
apply(*subgoal-tac allocating-node V t1* = *Some n1*) **prefer** *2*
apply(*case-tac t* = *t1*) **apply** *force* **apply** *force*
apply(*subgoal-tac allocating-node V t2* = *Some n1*) **prefer** *2*
apply(*case-tac t* = *t2*) **apply** *force* **apply** *force*
apply *metis*

apply *clarify*
apply(*subgoal-tac allocating-node V t1* = *Some n1*) **prefer** *2*
apply(*case-tac t* = *t1*) **apply** *force* **apply** *force*
apply(*subgoal-tac freeing-node V t2* = *Some n1*) **prefer** *2* **apply** *force*
apply *metis*
done

lemma *mp-alloc-stm5-lm-1-inv-bitmap0*:
p ∈ *mem-pools V* ⟹
inv-mempool-info V ∧ *inv-bitmap0 V* ⟹
inv-bitmap0
(*V*⦇*mem-pool-info* := (*mem-pool-info V*)
(*p* := *mem-pool-info V p*
⦇*levels* := *levels* (*mem-pool-info V p*)
[*nat* (*alloc-l V t*) := (*levels* (*mem-pool-info V p*) ! *nat* (*alloc-l V t*))
⦇*bits* := *bits* (*levels* (*mem-pool-info V p*) ! *nat* (*alloc-l V t*))
[(*blk V t* − *buf* (*mem-pool-info V p*)) *div lsizes V t* ! *nat* (*alloc-l V t*) := *ALLOCATED*]⦈]⦈),
allocating-node := (*allocating-node V*)(*t* := *None*)⦈)
apply(*simp add*:*inv-bitmap0-def Let-def*)
apply *clarsimp*
apply(*subgoal-tac length* (*levels* (*mem-pool-info V p*)) > *0*) **prefer** *2*
apply(*simp add*:*inv-mempool-info-def Let-def*) **apply** *fastforce*

apply(*case-tac nat* (*alloc-l V t*) = *0*)
  apply(*case-tac i* = (*blk V t* − *buf* (*mem-pool-info V p*)) *div lsizes V t* ! *nat* (*alloc-l V t*))
    apply *fastforce* **apply** *force*
by *fastforce*

lemma *mp-alloc-stm5-lm-1-inv-bitmapn*:
$p \in$ *mem-pools V* $\Longrightarrow$
  inv-mempool-info V $\wedge$ *inv-bitmapn V* $\Longrightarrow$
  inv-bitmapn
   (*V*(|*mem-pool-info* := (*mem-pool-info V*)
       (*p* := *mem-pool-info V p*
         (|*levels* := *levels* (*mem-pool-info V p*)
           [*nat* (*alloc-l V t*) := (*levels* (*mem-pool-info V p*) ! *nat* (*alloc-l V t*))
             (|*bits* := *bits* (*levels* (*mem-pool-info V p*) ! *nat* (*alloc-l V t*))
               [(*blk V t* − *buf* (*mem-pool-info V p*)) *div lsizes V t* ! *nat* (*alloc-l V t*) := *ALLOCATED*]|)]|)),
       allocating-node := (*allocating-node V*)(*t* := *None*)|))
apply(*simp add*:*inv-bitmapn-def Let-def*)
apply *clarsimp*
apply(*subgoal-tac length* (*levels* (*mem-pool-info V p*)) > *0*) **prefer** *2*
 apply(*simp add*:*inv-mempool-info-def Let-def*) **apply** *fastforce*

apply(*case-tac nat* (*alloc-l V t*) = *length* (*levels* (*mem-pool-info V p*)) − *Suc 0*)
  apply(*case-tac i* = (*blk V t* − *buf* (*mem-pool-info V p*)) *div lsizes V t* ! *nat* (*alloc-l V t*))
    apply *fastforce* **apply** *force*
by *fastforce*

lemma *mp-alloc-stm5-lm-1-inv-bitmap-not4free*:
(*blk V t* − *buf* (*mem-pool-info V p*)) *div lsizes V t* ! *nat* (*alloc-l V t*) < *n-max* (*mem-pool-info V p*) ∗ *4* ^ *nat* (*alloc-l V t*) $\Longrightarrow$
  alloc-l V t < *int* (*n-levels* (*mem-pool-info V p*)) $\Longrightarrow$
  $p \in$ *mem-pools V* $\Longrightarrow$
  inv-mempool-info V $\wedge$ *inv-bitmap-not4free V* $\Longrightarrow$
  inv-bitmap-not4free
   (*V*(|*mem-pool-info* := (*mem-pool-info V*)
       (*p* := *mem-pool-info V p*
         (|*levels* := *levels* (*mem-pool-info V p*)
           [*nat* (*alloc-l V t*) := (*levels* (*mem-pool-info V p*) ! *nat* (*alloc-l V t*))
             (|*bits* := *bits* (*levels* (*mem-pool-info V p*) ! *nat* (*alloc-l V t*))
               [(*blk V t* − *buf* (*mem-pool-info V p*)) *div lsizes V t* ! *nat* (*alloc-l V t*) := *ALLOCATED*]|)]|)),
       allocating-node := (*allocating-node V*)(*t* := *None*)|))
apply(*subgoal-tac length* (*levels* (*mem-pool-info V p*)) > *0*) **prefer** *2*
 apply(*simp add*:*inv-mempool-info-def Let-def*) **apply** *fastforce*
apply(*subgoal-tac nat* (*alloc-l V t*) < *length* (*levels* (*mem-pool-info V p*))) **prefer** *2*

apply(*simp add*:*inv-mempool-info-def Let-def*)
apply (*metis int-nat-eq of-nat-0-less-iff of-nat-less-imp-less*)

apply(*simp add*:*inv-bitmap-not4free-def partner-bits-def Let-def*)
apply *clarsimp*

apply(*subgoal-tac* (*blk V t* $-$ *buf* (*mem-pool-info V p*)) *div lsizes V t* ! *nat* (*alloc-l V t*)
$<$ *length* (*bits* (*levels* (*mem-pool-info V p*) ! *nat* (*alloc-l V t*))))
prefer *2*
apply(*simp add*:*inv-mempool-info-def Let-def*)

apply(*case-tac nat* (*alloc-l V t*) $<$ *length* (*levels* (*mem-pool-info V p*)))
apply(*case-tac i* $=$ *nat* (*alloc-l V t*))
apply(*case-tac* (*blk V t* $-$ *buf* (*mem-pool-info V p*)) *div lsizes V t* ! *nat* (*alloc-l V t*) $=$ *j div 4* $*$ *4*)
apply *fastforce*
apply(*case-tac* (*blk V t* $-$ *buf* (*mem-pool-info V p*)) *div lsizes V t* ! *nat* (*alloc-l V t*) $=$ *Suc* (*j div 4* $*$ *4*))
apply *fastforce*
apply(*case-tac* (*blk V t* $-$ *buf* (*mem-pool-info V p*)) *div lsizes V t* ! *nat* (*alloc-l V t*) $=$ *Suc* (*Suc* (*j div 4* $*$ *4*)))
apply *fastforce*
apply(*case-tac* (*blk V t* $-$ *buf* (*mem-pool-info V p*)) *div lsizes V t* ! *nat* (*alloc-l V t*) $=$ *j div 4* $*$ *4* $+$ *3*)
apply *fastforce*
apply *fastforce*
apply *force*
by *blast*

lemma *mp-alloc-stm5-lm-1-inv*:
(*blk V t* $-$ *buf* (*mem-pool-info V p*)) *div lsizes V t* ! *nat* (*alloc-l V t*) $<$ *n-max* (*mem-pool-info V p*) $*$ *4* ˆ *nat* (*alloc-l V t*) $\Longrightarrow$
blk V t $=$ *buf* (*mem-pool-info V p*) $+$ (*blk V t* $-$ *buf* (*mem-pool-info V p*)) *div lsizes V t* ! *nat* (*alloc-l V t*) $*$
(*max-sz* (*mem-pool-info V p*) *div 4* ˆ *nat* (*alloc-l V t*)) $\Longrightarrow$
allocating-node V t $=$
Some (|*pool* $=$ *p*, *level* $=$ *nat* (*alloc-l V t*), *block* $=$ (*blk V t* $-$ *buf* (*mem-pool-info V p*)) *div lsizes V t* ! *nat* (*alloc-l V t*),
data $=$ *blk V t*|) $\Longrightarrow$
free-l V t $\leq$ *alloc-l V t* $\Longrightarrow$
alloc-l V t $<$ *int* (*n-levels* (*mem-pool-info V p*)) $\Longrightarrow$
length (*lsizes V t*) $\leq$ *n-levels* (*mem-pool-info V p*) $\Longrightarrow$
p $\in$ *mem-pools V* $\Longrightarrow$
inv V $\Longrightarrow$
$\forall$ *ii*$<$*length* (*lsizes V t*). *lsizes V t* ! *ii* $=$ *ALIGN4* (*max-sz* (*mem-pool-info V p*) *div 4* ˆ *ii* $\Longrightarrow$
$\neg$ *free-l V t* $<$ *OK* $\Longrightarrow$
NULL $<$ *blk V t* $\Longrightarrow$

$\forall ii \le nat\ (alloc\text{-}l\ V\ t).\ sz \le lsizes\ V\ t\ !\ ii \Longrightarrow$
$alloc\text{-}l\ V\ t = int\ (length\ (lsizes\ V\ t)) - 1 \wedge length\ (lsizes\ V\ t) = n\text{-}levels\ (mem\text{-}pool\text{-}info\ V\ p) \vee$
$alloc\text{-}l\ V\ t = int\ (length\ (lsizes\ V\ t)) - 2 \wedge lsizes\ V\ t\ !\ nat\ (int\ (length\ (lsizes\ V\ t)) - 1) < sz \Longrightarrow$
$inv\ (V(\!|mem\text{-}pool\text{-}info := (mem\text{-}pool\text{-}info\ V)$
$(p := mem\text{-}pool\text{-}info\ V\ p$
$(\!|levels := levels\ (mem\text{-}pool\text{-}info\ V\ p)$
$[nat\ (alloc\text{-}l\ V\ t) := (levels\ (mem\text{-}pool\text{-}info\ V\ p)\ !\ nat\ (alloc\text{-}l\ V\ t))$
$(\!|bits := bits\ (levels\ (mem\text{-}pool\text{-}info\ V\ p)\ !\ nat\ (alloc\text{-}l\ V\ t))$
$[(blk\ V\ t - buf\ (mem\text{-}pool\text{-}info\ V\ p))\ div\ lsizes\ V\ t\ !\ nat\ (alloc\text{-}l\ V\ t) := ALLOCATED]|\!)]|\!)),$
$allocating\text{-}node := (allocating\text{-}node\ V)(t := None)|\!))$
apply(*simp add*:*inv-def*)
apply(*rule conjI*) **apply**(*simp add*:*inv-cur-def*)
apply(*rule conjI*) **apply**(*simp add*:*inv-thd-waitq-def*)
apply(*rule conjI*) **apply** *metis* **apply** *metis*
apply(*rule conjI*) **using** *mp-alloc-stm5-lm-1-inv-mempool-info* **apply** *blast*
apply(*rule conjI*) **using** *mp-alloc-stm5-lm-1-inv-bitmap-freelist* **apply** *blast*
apply(*rule conjI*) **using** *mp-alloc-stm5-lm-1-inv-bitmap* **apply** *blast*
apply(*rule conjI*) **using** *mp-alloc-stm5-lm-1-inv-aux-vars* **apply** *blast*
apply(*rule conjI*) **using** *mp-alloc-stm5-lm-1-inv-bitmap0* **apply** *blast*
apply(*rule conjI*) **using** *mp-alloc-stm5-lm-1-inv-bitmapn* **apply** *blast*
using *mp-alloc-stm5-lm-1-inv-bitmap-not4free* **apply** *blast*
done

term $mp\text{-}alloc\text{-}precond2\text{-}1\text{-}2\ t\ p\ sz\ timeout \cap \{\!|\acute{}cur = Some\ t|\!\}$

lemma *mp-alloc-stm5-lm-1*:
$mp\text{-}alloc\text{-}precond2\text{-}1\text{-}2\ t\ p\ sz\ timeout \cap \{\!|\acute{}cur = Some\ t|\!\} \cap \{V\} \neq \{\} \Longrightarrow$
$\Gamma \vdash_I Some\ (\acute{}mem\text{-}pool\text{-}info :=$
$set\text{-}bit\text{-}alloc\ \acute{}mem\text{-}pool\text{-}info\ p\ (nat\ (\acute{}alloc\text{-}l\ t))\ (block\text{-}num\ (\acute{}mem\text{-}pool\text{-}info\ p)\ (\acute{}blk\ t)\ (\acute{}lsizes\ t\ !\ nat\ (\acute{}alloc\text{-}l\ t)));;$
$\acute{}allocating\text{-}node := \acute{}allocating\text{-}node\ (t := None)\ )$
$sat_p\ [mp\text{-}alloc\text{-}precond2\text{-}1\text{-}2\ t\ p\ sz\ timeout \cap \{\!|\acute{}cur = Some\ t|\!\} \cap \{V\},$
$\{(s, t).\ s = t\},\ UNIV, \{\!|\acute{}(Pair\ V) \in Mem\text{-}pool\text{-}alloc\text{-}guar\ t|\!\} \cap mp\text{-}alloc\text{-}precond2\text{-}1\text{-}3\ t\ p\ sz\ timeout]$
apply(*subgoal-tac* $mp\text{-}alloc\text{-}precond2\text{-}1\text{-}2\ t\ p\ sz\ timeout \cap \{\!|\acute{}cur = Some\ t|\!\} \cap \{V\}=\{V\}$)
prefer *2* **using** *int1-eq*[**where** $P=mp\text{-}alloc\text{-}precond2\text{-}1\text{-}2\ t\ p\ sz\ timeout \cap \{\!|\acute{}cur = Some\ t|\!\}$] **apply** *meson*

apply *simp*

apply(*rule Seq*[**where** $mid=\{V(\!|mem\text{-}pool\text{-}info := set\text{-}bit\text{-}alloc\ (mem\text{-}pool\text{-}info\ V)\ p\ (nat\ (alloc\text{-}l\ V\ t))$
$(block\text{-}num\ ((mem\text{-}pool\text{-}info\ V)\ p)\ (blk\ V\ t)\ (lsizes\ V\ t\ !\ nat\ (alloc\text{-}l\ V\ t)))|\!)\}$])

apply(*rule Basic*)
apply *simp* **apply** *simp* **apply**(*simp add*:*stable-def*) **apply**(*simp add*:*stable-def*)

apply(*rule Basic*)
apply *clarsimp* **apply**(*simp add*: *set-bit-def block-num-def*)
apply(*rule conjI*)
apply(*simp add*:*Mem-pool-alloc-guar-def*) **apply**(*rule disjI1*)
apply(*rule conjI*)
apply(*simp add*:*gvars-conf-stable-def gvars-conf-def*) **apply** *clarsimp*
apply(*case-tac* $i = nat\ (alloc\text{-}l\ V\ t)$) **apply**(*case-tac* $i < length\ (levels\ (mem\text{-}pool\text{-}info\ V\ p))$)
apply *auto*[*1*] **apply** *auto*[*1*] **apply** *auto*[*1*]
apply(*rule conjI*) **using** *mp-alloc-stm5-lm-1-inv* **apply** *clarsimp*
apply(*simp add*:*lvars-nochange-def*)
apply(*rule conjI*) **using** *mp-alloc-stm5-lm-1-inv* **apply** *clarsimp*
apply(*case-tac* $alloc\text{-}l\ V\ t = int\ (length\ (lsizes\ V\ t)) - 1 \wedge length\ (lsizes\ V\ t) = n\text{-}levels\ (mem\text{-}pool\text{-}info\ V\ p)$)
apply *simp* **apply** *simp*

apply *simp* **apply**(*simp add*:*stable-def*) **using** *stable-id2* **apply** *metis*
done

lemma *mp-alloc-stm5-lm*:
$\Gamma \vdash_I Some\ (t \blacktriangleright\ ´mem\text{-}pool\text{-}info := set\text{-}bit\text{-}alloc\ ´mem\text{-}pool\text{-}info\ p\ (nat\ (´alloc\text{-}l\ t))$
$(block\text{-}num\ (´mem\text{-}pool\text{-}info\ p)\ (´blk\ t)\ ((´lsizes\ t)!(nat\ (´alloc\text{-}l\ t))));;$
$´allocating\text{-}node := ´allocating\text{-}node\ (t := None)$
$)\ sat_p\ [mp\text{-}alloc\text{-}precond2\text{-}1\text{-}2\ t\ p\ sz\ timeout,\ Mem\text{-}pool\text{-}alloc\text{-}rely\ t,\ Mem\text{-}pool\text{-}alloc\text{-}guar\ t,$
$mp\text{-}alloc\text{-}precond2\text{-}1\text{-}3\ t\ p\ sz\ timeout]$
apply(*simp add*:*stm-def*)
apply(*rule Await*)
using *mp-alloc-precond2-1-2-stb* **apply** *auto*[*1*]
using *mp-alloc-precond2-1-3-stb* **apply** *auto*[*1*]

apply *clarify*
apply(*case-tac* $mp\text{-}alloc\text{-}precond2\text{-}1\text{-}2\ t\ p\ sz\ timeout \cap \{\!|´cur = Some\ t|\!\} \cap \{V\} = \{\}$)
apply *simp* **using** *Emptyprecond* **apply** *metis*
using *mp-alloc-stm5-lm-1*[*of t p timeout sz*] **apply** *clarsimp*
done

term *mp-alloc-precond2-1-2 t p sz timeout*
term *mp-alloc-precond2-1-3 t p sz timeout*

10.9 stm6

lemma *mp-alloc-stm6-lm*:

$\Gamma \vdash_I$ *Some* (t ▶ ´*mempoolalloc-ret* := ´*mempoolalloc-ret* (t :=
Some ⦇*pool* = *p*, *level* = *nat* (´*alloc-l t*),
block = *block-num* (´*mem-pool-info p*) (´*blk t*) ((´*lsizes t*)!(*nat* (´*alloc-l t*))),
data = ´*blk t* ⦈))
sat$_p$ [*mp-alloc-precond2-1-3 t p sz timeout*, *Mem-pool-alloc-rely t*, *Mem-pool-alloc-guar t*,
mp-alloc-precond2-1-4 t p sz timeout]
apply(*simp add*:*stm-def*)
apply(*rule Await*)
using *mp-alloc-precond2-1-3-stb* **apply** *simp*
using *mp-alloc-precond2-1-4-stb* **apply** *simp*
apply *clarify*
apply(*rule Basic*)
apply *clarsimp*
apply(*rule conjI*)
apply(*simp add*:*Mem-pool-alloc-guar-def*) **apply**(*rule disjI1*)
apply(*rule conjI*)
apply(*simp add*:*gvars-conf-stable-def gvars-conf-def*)
apply(*rule conjI*)
apply(*subgoal-tac* (*V*, *V*⦇*mempoolalloc-ret* := *mempoolalloc-ret V*(*t* ↦
⦇*pool* = *p*, *level* = *nat* (*alloc-l V t*), *block* = *block-num* (*mem-pool-info V p*) (*blk V t*) (*lsizes V t* ! *nat* (*alloc-l V t*)),
data = *blk V t*⦈)⦈)∈*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def*)
apply(*simp add*:*lvars-nochange-def*)
apply(*rule conjI*)
apply(*subgoal-tac* (*V*, *V*⦇*mempoolalloc-ret* := *mempoolalloc-ret V*(*t* ↦
⦇*pool* = *p*, *level* = *nat* (*alloc-l V t*), *block* = *block-num* (*mem-pool-info V p*) (*blk V t*) (*lsizes V t* ! *nat* (*alloc-l V t*)),
data = *blk V t*⦈)⦈)∈*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def*)
apply(*simp add*:*alloc-memblk-valid-def*)
apply(*rule conjI*)
apply (*smt int-nat-eq inv-maxsz-align4 less-imp-le-nat not-less of-nat-less-iff*)

apply *clarify*
apply(*subgoal-tac* ¬(*alloc-l V t* = *int* (*length* (*lsizes V t*)) − *1* ∧ *length* (*lsizes V t*) = *n-levels* (*mem-pool-info V p*)))
prefer *2* **apply** *auto*[*1*]
apply *simp* **apply** (*smt Suc-nat-eq-nat-zadd1 inv-maxsz-align4 lessI nat-int power-Suc*)

apply *simp* **using** *stable-id2* **apply** *metis* **using** *stable-id2* **apply** *metis*
done

10.10 stm7

abbreviation *mp-alloc-stm7-precond1 Va* ≡ *Va*⦇*thd-state* := (*thd-state Va*)(*the* (*cur Va*) := *BLOCKED*)⦈
abbreviation *mp-alloc-stm7-precond3 Va t p* ≡
Va⦇*mem-pool-info* := (*mem-pool-info Va*)(*p* := (*mem-pool-info Va p*)⦇*wait-q* := (*wait-q* (*mem-pool-info Va p*))@ [*the* (*cur Va*)]⦈)⦈

lemma *mp-alloc-stm7-lm-2-1*: (λ*a*. *if a* = *p then mem-pool-info Va p*⦇*wait-q* := *wait-q* (*mem-pool-info Va p*) @ [*t*]⦈
else mem-pool-info (*Va*⦇*thd-state* := (*thd-state Va*)(*t* := *BLOCKED*)⦈) *a*) *x*
= (λ*a*. *if a* = *p then mem-pool-info Va p*⦇*wait-q* := *wait-q* (*mem-pool-info Va p*) @ [*t*]⦈
else mem-pool-info Va a) *x*
apply(*case-tac x* = *p*)
apply *auto*
done

lemma *mp-alloc-stm7-lm-2-2*:
cur Va = *Some t* ⟹
(λ*a*. *if a* = *p*
then mem-pool-info Va p⦇*wait-q* := *wait-q* (*mem-pool-info Va p*) @ [*the* (*cur* (*Va*⦇*thd-state* := (*thd-state Va*)(*t* := *BLOCKED*)⦈))]⦈
else mem-pool-info (*Va*⦇*thd-state* := (*thd-state Va*)(*t* := *BLOCKED*)⦈) *a*)
=
(λ*a*. *if a* = *p then mem-pool-info Va p*⦇*wait-q* := *wait-q* (*mem-pool-info Va p*) @ [*t*]⦈ *else mem-pool-info Va a*)
using *mp-alloc-stm7-lm-2-1* **by** *auto*

lemma *mp-alloc-stm7-lm-2*:
cur Va = *Some t* ⟹
(λ*a*. *if a* = *p then*
mem-pool-info (*Va*⦇*thd-state* := (*thd-state Va*)(*t* := *BLOCKED*)⦈) *p*
⦇*wait-q* := *wait-q* (*mem-pool-info* (*Va*⦇*thd-state* := (*thd-state Va*)(*t* := *BLOCKED*)⦈) *p*)
@ [*the* (*cur* (*Va*⦇*thd-state* := (*thd-state Va*)(*t* := *BLOCKED*)⦈))]⦈
else mem-pool-info (*Va*⦇*thd-state* := (*thd-state Va*)(*t* := *BLOCKED*)⦈) *a*)
=
(*mem-pool-info Va*)(*p* := *mem-pool-info Va p*⦇*wait-q* := *wait-q* (*mem-pool-info Va p*) @ [*t*]⦈)
apply(*rule subst*[**where** *t*=*mem-pool-info* (*Va*⦇*thd-state* := (*thd-state Va*)(*t* := *BLOCKED*)⦈) *p*
and *s*=*mem-pool-info Va p*]) **apply** *simp*
apply(*simp add*:*fun-upd-def*)
using *mp-alloc-stm7-lm-2-2* **apply** *auto*
done

lemma *mp-alloc-stm7-swap-ifbody-inv*:
$p \in$ *mem-pools Va* $\Longrightarrow$
inv Va $\Longrightarrow$
cur Va = *Some t* $\Longrightarrow$
(*if ta* = *t then BLOCKED else thd-state Va ta*) = *READY* $\Longrightarrow$
inv (*mp-alloc-stm7-precond3 Va t p*
(|*cur* := *Some* (*SOME ta*. *ta* $\neq$ *t* $\wedge$ (*ta* $\neq$ *t* $\longrightarrow$ *thd-state Va ta* = *READY*)),
thd-state :=
λx. *if x* = (*SOME ta*. *ta* $\neq$ *t* $\wedge$ (*ta* $\neq$ *t* $\longrightarrow$ *thd-state Va ta* = *READY*)) *then RUNNING*
else thd-state
(*Va*(|*thd-state* := (*thd-state Va*)(*t* := *BLOCKED*),
mem-pool-info := (*mem-pool-info Va*)
(*p* := *mem-pool-info Va p*(|*wait-q* := *wait-q* (*mem-pool-info Va p*) @ [*t*]|)),
cur := *Some* (*SOME ta*. (*ta* = *t* $\longrightarrow$ *BLOCKED* = *READY*) $\wedge$ (*ta* $\neq$ *t* $\longrightarrow$ *thd-state Va ta* = *READY*))|))
x|))
apply(*subgoal-tac thd-state Va t* = *RUNNING*)
prefer *2* **apply**(*simp add*:*inv-def inv-cur-def*) **apply** *auto*[*1*]
apply(*subgoal-tac ta* $\neq$ *t* $\wedge$ *thd-state Va ta* = *READY*)
prefer *2* **apply** *auto*[*1*] **using** *Thread-State-Type.distinct*(*3*) **apply** *presburger*

apply(*subgoal-tac* (*SOME ta*. *ta* $\neq$ *t* $\wedge$ (*ta* $\neq$ *t* $\longrightarrow$ *thd-state Va ta* = *READY*)) $\neq$ *t*)
prefer *2* **using** *exE-some*[**where** *P*=λ*tb*. *tb* $\neq$ *t* $\wedge$ (*tb* $\neq$ *t* $\longrightarrow$ *thd-state Va tb* = *READY*)
and *c*=*SOME tb*. *tb* $\neq$ *t* $\wedge$ (*tb* $\neq$ *t* $\longrightarrow$ *thd-state Va tb* = *READY*)]
apply *auto*[*1*]
apply(*subgoal-tac thd-state Va* (*SOME ta*. *ta* $\neq$ *t* $\wedge$ (*ta* $\neq$ *t* $\longrightarrow$ *thd-state Va ta* = *READY*)) = *READY*)
prefer *2* **using** *exE-some*[**where** *P*=λ*tb*. *tb* $\neq$ *t* $\wedge$ (*tb* $\neq$ *t* $\longrightarrow$ *thd-state Va tb* = *READY*)
and *c*=*SOME tb*. *tb* $\neq$ *t* $\wedge$ (*tb* $\neq$ *t* $\longrightarrow$ *thd-state Va tb* = *READY*)]
apply *auto*[*1*]

apply(*simp add*:*inv-def*)
apply(*rule conjI*) **apply**(*simp add*:*inv-cur-def*) **apply** *auto*[*1*]
apply(*rule conjI*) **apply**(*simp add*:*inv-thd-waitq-def*)
apply(*rule conjI*) **apply** *auto*[*1*]
apply(*rule conjI*) **apply** *auto*[*1*]
apply(*rule conjI*) **apply** (*metis* (*no-types*, *lifting*) *Thread-State-Type.distinct*(*5*) *diff-is-0-eq'*
less-Suc-eq less-Suc-eq-le nth-Cons-0 nth-append nth-mem)
apply *auto*[*1*]
apply(*rule conjI*) **apply**(*simp add*:*inv-mempool-info-def*) **apply** *meson*
apply(*rule conjI*) **apply**(*simp add*:*inv-bitmap-freelist-def*) **apply** *meson*
apply(*rule conjI*) **apply**(*simp add*:*inv-bitmap-def*) **apply**(*simp add*:*Let-def*)

apply(*rule conjI*) **apply**(*simp add*: *inv-aux-vars-def mem-block-addr-valid-def*)
apply *meson*
apply(*rule conjI*) **apply**(*simp add*:*inv-bitmap0-def*)
apply(*rule conjI*) **apply**(*simp add*:*inv-bitmapn-def*)
apply(*simp add*:*inv-bitmap-not4free-def partner-bits-def*) **apply** *meson*
done

lemma *mp-alloc-stm7-swap-elsebody-inv*:
$p \in mem\text{-}pools\ Va \Longrightarrow$
$inv\ Va \Longrightarrow$
$cur\ Va = Some\ t \Longrightarrow$
$(if\ ta = t\ then\ BLOCKED\ else\ thd\text{-}state\ Va\ ta) \neq READY \Longrightarrow$
$inv\ (cur\text{-}update\ Map.empty$
$(Va(\!|thd\text{-}state := (thd\text{-}state\ Va)(t := BLOCKED),$
$mem\text{-}pool\text{-}info := (mem\text{-}pool\text{-}info\ Va)(p := mem\text{-}pool\text{-}info\ Va\ p(\!|wait\text{-}q := wait\text{-}q\ (mem\text{-}pool\text{-}info\ Va\ p)\ @\ [t]|\!))|\!)))$
apply(*subgoal-tac thd-state Va t = RUNNING*)
prefer *2* **apply**(*simp add*:*inv-def inv-cur-def*) **apply** *auto*[*1*]

apply(*simp add*:*inv-def*)
apply(*rule conjI*) **apply**(*simp add*:*inv-cur-def*) **apply** *auto*[*1*]
apply(*rule conjI*) **apply**(*simp add*:*inv-thd-waitq-def*)
apply(*rule conjI*) **apply** *auto*[*1*]
apply(*rule conjI*) **apply** (*metis Thread-State-Type.distinct*(*6*) *diff-is-0-eq′ less-Suc-eq*
less-Suc-eq-le nth-Cons-0 nth-append nth-mem)
apply (*metis* (*no-types*, *lifting*) *Thread-State-Type.distinct*(*5*))
apply(*rule conjI*) **apply**(*simp add*:*inv-mempool-info-def*) **apply** *meson*
apply(*rule conjI*) **apply**(*simp add*:*inv-bitmap-freelist-def*) **apply** *meson*
apply(*rule conjI*) **apply**(*simp add*:*inv-bitmap-def*) **apply**(*simp add*:*Let-def*)
apply(*rule conjI*) **apply**(*simp add*: *inv-aux-vars-def mem-block-addr-valid-def*)
apply *meson*
apply(*rule conjI*) **apply**(*simp add*:*inv-bitmap0-def*)
apply(*rule conjI*) **apply**(*simp add*:*inv-bitmapn-def*)
apply(*simp add*:*inv-bitmap-not4free-def partner-bits-def*) **apply** *meson*

done

lemma *mp-alloc-stm7-lm-1*:
$mp\text{-}alloc\text{-}precond1\text{-}8\text{-}2\text{-}2\ t\ p\ sz\ timeout \cap \{\!|\acute{}cur = Some\ t|\!\} \cap \{V\} \cap UNIV \cap \{Va\} \neq \{\} \Longrightarrow$
$\Gamma \vdash_I Some\ (\acute{}thd\text{-}state := \acute{}thd\text{-}state(the\ \acute{}cur := BLOCKED);;$
$\acute{}mem\text{-}pool\text{-}info := \acute{}mem\text{-}pool\text{-}info(p := \acute{}mem\text{-}pool\text{-}info\ p(\!|wait\text{-}q := wait\text{-}q\ (\acute{}mem\text{-}pool\text{-}info\ p)\ @\ [the\ \acute{}cur]\ |\!));;$
$swap\ )$
$sat_p\ [mp\text{-}alloc\text{-}precond1\text{-}8\text{-}2\text{-}2\ t\ p\ sz\ timeout \cap \{\!|\acute{}cur = Some\ t|\!\} \cap \{V\} \cap UNIV$

$\cap$ {*Va*},
{(*s*, *t*). *s* = *t*}, *UNIV*, ⦃´(*Pair Va*) ∈ *UNIV*⦄ ∩ (⦃´(*Pair V*) ∈ *Mem-pool-alloc-guar t*⦄ ∩
(*mp-alloc-precond1-8-2-2 t p sz timeout*))]
apply(*subgoal-tac V = Va*)
prefer *2* **apply** *simp*
apply(*subgoal-tac mp-alloc-precond1-8-2-2 t p sz timeout* ∩ ⦃´*cur* = *Some t*⦄ ∩ {*V*} ∩ *UNIV* ∩ {*Va*} = {*Va*})
prefer *2* **apply** *auto*[*1*]
apply(*rule subst*[**where** *t*=*mp-alloc-precond1-8-2-2 t p sz timeout* ∩ ⦃´*cur* = *Some t*⦄ ∩ {*V*} ∩ *UNIV* ∩ {*Va*} **and** *s*={*V*}])
apply *simp*
apply *clarsimp*

apply(*rule Seq*[**where** *mid*={*mp-alloc-stm7-precond3* (*mp-alloc-stm7-precond1 Va*) *t p*}])
apply(*rule Seq*[**where** *mid*={*mp-alloc-stm7-precond1 Va*}])

apply(*rule Basic*)
apply(*simp add*:*fun-upd-def*)
apply *simp* **apply**(*simp add*:*stable-def*) **apply**(*simp add*:*stable-def*)

apply(*rule Basic*)
apply *simp* **using** *mp-alloc-stm7-lm-2*[*of Va t p*] **apply** *metis*
apply *simp* **apply**(*simp add*:*stable-def*) **apply**(*simp add*:*stable-def*)

apply(*simp add*:*swap-def*)
apply(*rule Cond*)
apply(*simp add*:*stable-def*)

apply(*case-tac* {*Va*⦇*thd-state* := (*thd-state Va*)(*t* := *BLOCKED*),
mem-pool-info := (*mem-pool-info Va*)
(*p* := *mem-pool-info Va p*⦇*wait-q* := *wait-q* (*mem-pool-info Va p*) @ [*t*]⦈)⦈} ∩
⦃∃ *t*. ´*thd-state t* = *READY*⦄ = {})
apply *simp* **using** *Emptyprecond* **apply** *metis*
apply(*rule subst*[**where** *t*={*Va*⦇*thd-state* := (*thd-state Va*)(*t* := *BLOCKED*),
mem-pool-info := (*mem-pool-info Va*)
(*p* := *mem-pool-info Va p*⦇*wait-q* := *wait-q* (*mem-pool-info Va p*) @ [*t*]⦈)⦈} ∩
⦃∃ *t*. ´*thd-state t* = *READY*⦄ **and** *s*={*Va*⦇*thd-state* := (*thd-state Va*)(*t* := *BLOCKED*),
mem-pool-info := (*mem-pool-info Va*)
(*p* := *mem-pool-info Va p*⦇*wait-q* := *wait-q* (*mem-pool-info Va p*) @ [*t*]⦈)⦈}])
apply *simp*
apply(*rule Seq*[**where** *mid*={*let V* = *mp-alloc-stm7-precond3* (*mp-alloc-stm7-precond1 Va*) *t p in*

$V(\!|cur := Some\ (SOME\ t.\ (thd\text{-}state\ V)\ t = READY)|\!)\}])$

apply(*rule Basic*)
apply *auto*[*1*] **apply** *simp* **apply**(*simp add*:*stable-def*) **apply**(*simp add*:*stable-def*)

apply(*rule Basic*)
apply *auto*[*1*]
apply(*simp add*:*Mem-pool-alloc-guar-def*)
apply(*rule disjI1*)
apply(*rule conjI*)
apply(*simp add*:*gvars-conf-stable-def gvars-conf-def*)
apply(*rule conjI*)
using *mp-alloc-stm7-swap-ifbody-inv* **apply** *auto*[*1*]
apply(*simp add*:*lvars-nochange-def*)
using *mp-alloc-stm7-swap-ifbody-inv* **apply** *auto*[*1*]
apply(*simp add*:*Mem-pool-alloc-guar-def*)
apply(*rule disjI1*)
apply(*rule conjI*)
apply(*simp add*:*gvars-conf-stable-def gvars-conf-def*)
apply(*rule conjI*)
using *mp-alloc-stm7-swap-ifbody-inv* **apply** *auto*[*1*]
apply(*simp add*:*lvars-nochange-def*)
using *mp-alloc-stm7-swap-ifbody-inv* **apply** *auto*[*1*]

apply *simp* **apply**(*simp add*:*stable-def*) **using** *stable-id2* **apply** *metis*

apply(*rule Basic*)
apply *auto*[*1*]
apply(*simp add*:*Mem-pool-alloc-guar-def*)
apply(*rule disjI1*)
apply(*rule conjI*)
apply(*simp add*:*gvars-conf-stable-def gvars-conf-def*)
apply(*rule conjI*)
using *mp-alloc-stm7-swap-elsebody-inv* **apply** *auto*[*1*]
apply(*simp add*:*lvars-nochange-def*)
using *mp-alloc-stm7-swap-elsebody-inv* **apply** *auto*[*1*]
apply(*simp add*:*Mem-pool-alloc-guar-def*)
apply(*rule disjI1*)
apply(*rule conjI*)
apply(*simp add*:*gvars-conf-stable-def gvars-conf-def*)
apply(*rule conjI*)
using *mp-alloc-stm7-swap-elsebody-inv* **apply** *auto*[*1*]
apply(*simp add*:*lvars-nochange-def*)
using *mp-alloc-stm7-swap-elsebody-inv* **apply** *auto*[*1*]

apply *simp* **apply**(*simp add*:*stable-def*) **using** *stable-id2* **apply** *metis*
apply *simp*

done

lemma *mp-alloc-stm7-lm*:
$\Gamma \vdash_I$ *Some* (t ▶ *ATOMIC*
´*thd-state* := ´*thd-state*(*the* ´*cur* := *BLOCKED*);;
´*mem-pool-info* := ´*mem-pool-info*(p := ´*mem-pool-info* p⦇*wait-q* := *wait-q*
(´*mem-pool-info* p) @ [*the* ´*cur*] ⦈);;
swap
END) sat_p [*mp-alloc-precond1-8-2-2* t p *sz timeout*, *Mem-pool-alloc-rely* t,
Mem-pool-alloc-guar t,
mp-alloc-precond1-8-2-2 t p *sz timeout*]
apply(*simp add*:*stm-def*)
apply(*rule Await*)
using *mp-alloc-precond1-8-2-2-stb* **apply** *simp*
using *mp-alloc-precond1-8-2-2-stb* **apply** *simp*
apply *clarify*
apply(*rule Await*)
using *stable-id2* **apply** *metis*
using *stable-id2* **apply** *metis*
apply *clarify*
apply(*case-tac mp-alloc-precond1-8-2-2 t p sz timeout* ∩
⦃´*cur* = *Some* t⦄ ∩ {V} ∩ *UNIV* ∩ {*Va*} = {})
using *Emptyprecond* **apply** *metis*
using *mp-alloc-stm7-lm-1* **apply** *meson*
done

term *mp-alloc-precond1-8-2-2 t p sz timeout*

10.11 final proof

lemma *mp-alloc-stm8-guar*:
cur V = *Some* t ⟹ *inv* V ⟹ V⦇*rf* := (*rf* V)(t := *True*)⦈ ∈ ⦃´(*Pair* V) ∈
Mem-pool-alloc-guar t⦄
apply *auto* **apply**(*simp add*:*Mem-pool-alloc-guar-def gvars-conf-stable-def gvars-conf-def*
lvars-nochange-def)
apply(*rule disjI1*)
apply(*subgoal-tac* (V,V⦇*rf* := (*rf* V)(t := *True*)⦈)∈*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def*
lvars-nochange1-def)
done

lemma *mp-alloc-stm9-guar*:
cur V = *Some* t ⟹ *inv* V ⟹ V⦇*ret* := (*ret* V)(t := *ETIMEOUT*)⦈ ∈ ⦃´(*Pair*
V) ∈ *Mem-pool-alloc-guar* t⦄
apply *auto* **apply**(*simp add*:*Mem-pool-alloc-guar-def gvars-conf-stable-def gvars-conf-def*
lvars-nochange-def)
apply(*rule disjI1*)

```
 apply(subgoal-tac (V,V(|ret := (ret V)(t := ETIMEOUT)|))∈lvars-nochange1-4all)
  using glnochange-inv0 apply auto[1] apply(simp add:lvars-nochange1-4all-def
lvars-nochange1-def)
done

lemma Mempool-alloc-satRG: Γ (Mem-pool-alloc t p sz timeout) ⊢ Mem-pool-alloc-RGCond
t p sz timeout
  apply(simp add: Mem-pool-alloc-RGCond-def getrgformula-def)
  apply(simp add:Evt-sat-RG-def Pre_f-def Post_f-def Rely_f-def Guar_f-def)
  apply(unfold Mem-pool-alloc-def)
  apply(rule BasicEvt)
  apply(unfold body-def guard-def snd-conv fst-conv)

  apply(rule Seq[where mid=mp-alloc-precond7 t p sz timeout])
  apply(rule Seq[where mid=mp-alloc-precond6 t p timeout])
  apply(rule Seq[where mid=mp-alloc-precond5 t p timeout])
  apply(rule Seq[where mid=mp-alloc-precond4 t p timeout])
  apply(rule Seq[where mid=mp-alloc-precond3 t p timeout])
  apply(rule Seq[where mid=mp-alloc-precond2 t p timeout])


  apply(simp add:stm-def)
  apply(rule Await)
    using mp-alloc-precond1-stb apply auto[1]
    using mp-alloc-precond2-stb apply simp
    apply(rule allI)
      apply(rule Basic)
      apply(case-tac mp-alloc-precond1 t p timeout ∩ {|´cur = Some t|} ∩ {V} =
{})
        apply auto[1] apply simp
        apply(rule conjI)
          apply(simp add:Mem-pool-alloc-guar-def) apply(rule disjI1)
          apply(rule conjI) apply(simp add:gvars-conf-stable-def gvars-conf-def)
          apply(rule conjI)
         apply(subgoal-tac (V,V(|tmout := (tmout V)(t := timeout)|))∈lvars-nochange1-4all)
         using glnochange-inv0 apply auto[1] apply(simp add:lvars-nochange1-4all-def
lvars-nochange1-def)
            apply(simp add:lvars-nochange-def)
       apply(subgoal-tac (V,V(|tmout := (tmout V)(t := timeout)|))∈lvars-nochange1-4all)
         using glnochange-inv0 apply auto[1] apply(simp add:lvars-nochange1-4all-def
lvars-nochange1-def)
        apply(simp add:stable-def)+


  apply(simp add:stm-def)
  apply(rule Await)
    using mp-alloc-precond2-stb apply simp
    using mp-alloc-precond3-stb apply simp
```

apply(*rule allI*)
apply(*rule Basic*)
apply(*case-tac mp-alloc-precond2 t p timeout* $\cap$ $\{\!|\,{}^{\prime}cur = Some\ t|\!\}$ $\cap$ $\{V\}$ = $\{\}$)
apply *auto*[*1*] **apply** *simp*
apply(*rule conjI*)
apply(*simp add*:*Mem-pool-alloc-guar-def*) **apply**(*rule disjI1*)
apply(*rule conjI*) **apply**(*simp add*:*gvars-conf-stable-def gvars-conf-def*)
apply(*rule conjI*)
apply(*subgoal-tac* $(V, V(\!|endt := (endt\ V)(t := NULL)|\!)) \in$ *lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def*)
apply(*simp add*:*lvars-nochange-def*)
apply(*subgoal-tac* $(V, V(\!|endt := (endt\ V)(t := NULL)|\!)) \in$ *lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def*)
apply *simp* **apply**(*simp add*:*stable-def*) **apply**(*simp add*:*stable-def*)

apply(*unfold stm-def*)[*1*]
apply(*rule Await*)
using *mp-alloc-precond3-stb* **apply** *simp*
using *mp-alloc-precond4-stb* **apply** *simp*
apply *clarify*
apply(*rule Cond*)
apply(*simp add*:*stable-def*)
apply(*rule Basic*)
apply(*case-tac mp-alloc-precond3 t p timeout* $\cap$ $\{\!|\,{}^{\prime}cur = Some\ t|\!\}$ $\cap$ $\{V\}$ = $\{\}$)
apply *auto*[*1*] **apply** *auto*[*1*]
apply(*simp add*:*Mem-pool-alloc-guar-def*) **apply** *auto*[*1*]
apply(*simp add*:*gvars-conf-stable-def gvars-conf-def*)
apply(*subgoal-tac* $(V, V(\!|endt := (endt\ V)(t := tick\ V + nat\ (tmout\ V\ t))|\!)) \in$ *lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def*)
apply(*simp add*:*lvars-nochange-def*)
apply(*subgoal-tac* $(V, V(\!|endt := (endt\ V)(t := tick\ V + nat\ (tmout\ V\ t))|\!)) \in$ *lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def*)
apply *simp* **apply**(*simp add*:*stable-def*) **apply**(*simp add*:*stable-def*)

apply(*unfold Skip-def*)[*1*]
apply(*rule Basic*)
apply(*case-tac mp-alloc-precond3 t p timeout* $\cap$ $\{\!|\,{}^{\prime}cur = Some\ t|\!\}$ $\cap$ $\{V\}$ $\cap$ $-$ $\{\!|OK < timeout|\!\}$ = $\{\}$)
apply *auto*[*1*] **apply** *auto*[*1*]
apply(*simp add*:*Mem-pool-alloc-guar-def*)+

apply(*simp add*:*stable-def*)+

apply(*simp add*:*stm-def*)
apply(*rule Await*)
using *mp-alloc-precond4-stb* **apply** *simp*
using *mp-alloc-precond5-stb* **apply** *simp*
apply(*rule allI*)
apply(*rule Basic*)
apply(*case-tac mp-alloc-precond2 t p timeout* $\cap$ ⦃´*cur* = *Some t*⦄ $\cap$ { *V* } = {})
apply *auto*[*1*] **apply** *simp*
apply(*rule conjI*)
apply(*simp add*:*Mem-pool-alloc-guar-def*) **apply**(*rule disjI1*)
apply(*rule conjI*) **apply**(*simp add*:*gvars-conf-stable-def gvars-conf-def*)
apply(*rule conjI*)
apply(*subgoal-tac* (*V*,*V*⦇*mempoolalloc-ret* := (*mempoolalloc-ret V*)(*t* := *None*)⦈)∈*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def*)
apply(*simp add*:*lvars-nochange-def*)
apply(*subgoal-tac* (*V*,*V*⦇*mempoolalloc-ret* := (*mempoolalloc-ret V*)(*t* := *None*)⦈)∈*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def*)
apply(*simp add*:*stable-def*)+

apply(*simp add*:*stm-def*)
apply(*rule Await*)
using *mp-alloc-precond5-stb* **apply** *simp*
using *mp-alloc-precond6-stb* **apply** *simp*
apply(*rule allI*)
apply(*rule Basic*)
apply(*case-tac mp-alloc-precond5 t p timeout* $\cap$ ⦃´*cur* = *Some t*⦄ $\cap$ { *V* } = {})
apply *auto*[*1*] **apply** *simp*
apply(*rule conjI*)
apply(*simp add*:*Mem-pool-alloc-guar-def*) **apply**(*rule disjI1*)
apply(*rule conjI*) **apply**(*simp add*:*gvars-conf-stable-def gvars-conf-def*)
apply(*rule conjI*)
apply(*subgoal-tac* (*V*,*V*⦇*ret* := (*ret V*)(*t* := *ESIZEERR*)⦈)∈*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def*)
apply(*simp add*:*lvars-nochange-def*)
apply(*subgoal-tac* (*V*,*V*⦇*ret* := (*ret V*)(*t* := *ESIZEERR*)⦈)∈*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def*)
apply(*simp add*:*stable-def*)+

apply(*simp add*:*stm-def*)
apply(*rule Await*)
using *mp-alloc-precond6-stb* **apply** *simp*
using *mp-alloc-precond7-stb* **apply** *simp*
apply(*rule allI*)
apply(*rule Basic*)
apply(*case-tac mp-alloc-precond6 t p timeout* $\cap$ $\{\!|$ ´*cur* = *Some t*$|\!\}$ $\cap$ $\{V\}$ = $\{\}$)
apply *auto*[*1*] **apply** *simp*
apply(*rule conjI*)
apply(*simp add*:*Mem-pool-alloc-guar-def*) **apply**(*rule disjI1*)
apply(*rule conjI*) **apply**(*simp add*:*gvars-conf-stable-def gvars-conf-def*)
apply(*rule conjI*)
apply(*subgoal-tac* $(V, V(\!|rf := (rf\ V)(t := False)|\!)) \in$ *lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def*)
apply(*simp add*:*lvars-nochange-def*)
apply(*subgoal-tac* $(V, V(\!|rf := (rf\ V)(t := False)|\!)) \in$ *lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def*)
apply(*simp add*:*stable-def*)+

apply(*rule While*)
using *mp-alloc-precond7-stb* **apply** *simp*
apply(*simp add*:*Mem-pool-alloc-post-def*) **apply** *auto*[*1*]
using *mp-alloc-post-stb* **apply** *simp*

prefer *2* **apply**(*simp add*:*Mem-pool-alloc-guar-def*)
prefer *2* **apply** (*simp add*: *mem-pool-alloc-pre-stb*)
prefer *2* **apply**(*simp add*:*Mem-pool-alloc-guar-def*)

apply(*rule Seq*[**where** *mid*=*mp-alloc-precond1-8 t p sz timeout*])
apply(*rule Seq2*[**where** *mida*=*mp-alloc-precond1-7 t p sz timeout* **and** *midb*=*mp-alloc-precond1-70 t p sz timeout*])

apply(*rule Seq*[**where** *mid*=*mp-alloc-precond1-6 t p sz timeout*])
apply(*rule Seq*[**where** *mid*=*mp-alloc-precond1-5 t p sz timeout*])
apply(*rule Seq*[**where** *mid*=*mp-alloc-precond1-4 t p sz timeout*])
apply(*rule Seq*[**where** *mid*=*mp-alloc-precond1-3 t p sz timeout*])
apply(*rule Seq*[**where** *mid*=*mp-alloc-precond1-2 t p sz timeout*])
apply(*rule Seq*[**where** *mid*=*mp-alloc-precond1-1 t p sz timeout*])

apply(*simp add*:*stm-def*)
apply(*rule Await*)

using *mp-alloc-precond1-0-stb* **apply** *simp*
using *mp-alloc-precond1-1-stb* **apply** *simp*
apply(*rule allI*)
apply(*rule Basic*)
apply(*case-tac mp-alloc-precond1-0 t p sz timeout* $\cap$ $\{\!\!|$ ´*cur* = *Some t*$|\!\!\}$ $\cap$ $\{V\} = \{\}$)
apply *auto*[*1*] **apply** *clarify*
apply(*rule IntI*) **apply** *auto*[*1*]
apply(*simp add*:*Mem-pool-alloc-guar-def lvars-nochange1-def lvars-nochange-def*

gvars-conf-stable-def gvars-conf-def)
apply(*subgoal-tac* $(V, V(\!|$*blk* := (*blk V*)(*t* := *NULL*)$|\!))\in$*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*]
apply(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def lvars-nochange-def*)
apply(*simp add*:*Mem-pool-alloc-guar-def lvars-nochange1-def lvars-nochange-def*

gvars-conf-stable-def gvars-conf-def)
apply(*subgoal-tac* $(V, V(\!|$*blk* := (*blk V*)(*t* := *NULL*)$|\!))\in$*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*]
apply(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def lvars-nochange-def*)
apply(*simp add*:*alloc-memblk-valid-def*)
apply(*subgoal-tac* $(V, V(\!|$*blk* := (*blk V*)(*t* := *NULL*)$|\!))\in$*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def*)
apply(*simp add*:*stable-def*)+

apply(*simp add*:*stm-def*)
apply(*rule Await*)
using *mp-alloc-precond1-1-stb* **apply** *simp*
using *mp-alloc-precond1-2-stb* **apply** *simp*
apply(*rule allI*)
apply(*rule Basic*)
apply(*case-tac mp-alloc-precond1-1 t p sz timeout* $\cap$ $\{\!\!|$ ´*cur* = *Some t*$|\!\!\}$ $\cap$ $\{V\} = \{\}$)
apply *auto*[*1*] **apply** *clarify*
apply(*rule IntI*) **apply** *auto*[*1*]
apply(*simp add*:*Mem-pool-alloc-guar-def lvars-nochange1-def lvars-nochange-def*

gvars-conf-stable-def gvars-conf-def)
apply(*subgoal-tac* $(V, V(\!|$*alloc-lsize-r* := (*alloc-lsize-r V*)(*t* := *False*)$|\!))\in$*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*]
apply(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def lvars-nochange-def*)
apply(*simp add*:*Mem-pool-alloc-guar-def lvars-nochange1-def lvars-nochange-def*

gvars-conf-stable-def gvars-conf-def)
apply(*subgoal-tac* $(V, V(\!|$*alloc-lsize-r* := (*alloc-lsize-r V*)(*t* := *False*)$|\!))\in$*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*]
apply(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def lvars-nochange-def*)

apply(*simp add*:*alloc-memblk-valid-def*)
apply(*subgoal-tac* (V,V(|*alloc-lsize-r* := (*alloc-lsize-r* V)(t := *False*)|))∈*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def*
lvars-nochange1-def)
apply(*simp add*:*stable-def*)+

apply(*simp add*:*stm-def*)
apply(*rule Await*)
using *mp-alloc-precond1-2-stb* **apply** *simp*
using *mp-alloc-precond1-3-stb* **apply** *simp*
apply(*rule allI*)
apply(*rule Basic*)
apply(*case-tac mp-alloc-precond1-2 t p sz timeout* ∩ {|´*cur* = *Some t*|} ∩
{V} = {})
apply *auto*[*1*] **apply** *clarify*
apply(*rule IntI*) **apply** *auto*[*1*]
apply(*simp add*:*Mem-pool-alloc-guar-def lvars-nochange1-def lvars-nochange-def*

gvars-conf-stable-def gvars-conf-def)
apply(*subgoal-tac* (V,V(|*alloc-l* := (*alloc-l* V)(t := *ETIMEOUT*)|))∈*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*]
apply(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def lvars-nochange-def*)
apply(*simp add*:*Mem-pool-alloc-guar-def lvars-nochange1-def lvars-nochange-def*

gvars-conf-stable-def gvars-conf-def)
apply(*subgoal-tac* (V,V(|*alloc-l* := (*alloc-l* V)(t := *ETIMEOUT*)|))∈*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*]
apply(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def lvars-nochange-def*)
apply(*simp add*:*alloc-memblk-valid-def*)
apply(*subgoal-tac* (V,V(|*alloc-l* := (*alloc-l* V)(t := *ETIMEOUT*)|))∈*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def*
lvars-nochange1-def)
apply(*simp add*:*stable-def*)+

apply(*simp add*:*stm-def*)
apply(*rule Await*)
using *mp-alloc-precond1-3-stb* **apply** *simp*
using *mp-alloc-precond1-4-stb* **apply** *simp*
apply(*rule allI*)
apply(*rule Basic*)
apply(*case-tac mp-alloc-precond1-3 t p sz timeout* ∩ {|´*cur* = *Some t*|} ∩
{V} = {})
apply *auto*[*1*] **apply** *clarify*
apply(*rule IntI*) **apply** *auto*[*1*]
apply(*simp add*:*Mem-pool-alloc-guar-def lvars-nochange1-def lvars-nochange-def*

gvars-conf-stable-def gvars-conf-def)

apply(*subgoal-tac* $(V, V(\!|$*free-l* := (*free-l* V)(t := *ETIMEOUT*)$|\!)) \in$ *lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*]
apply(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def lvars-nochange-def*)
apply(*simp add*:*Mem-pool-alloc-guar-def lvars-nochange1-def lvars-nochange-def*

gvars-conf-stable-def gvars-conf-def)
apply(*subgoal-tac* $(V, V(\!|$*free-l* := (*free-l* V)(t := *ETIMEOUT*)$|\!)) \in$ *lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*]
apply(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def lvars-nochange-def*)
apply(*simp add*:*alloc-memblk-valid-def*)
apply(*subgoal-tac* $(V, V(\!|$*free-l* := (*free-l* V)(t := *ETIMEOUT*)$|\!)) \in$ *lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def*)
apply(*simp add*:*stable-def*)+

apply(*simp add*:*stm-def*)
apply(*rule Await*)
using *mp-alloc-precond1-4-stb* **apply** *simp*
using *mp-alloc-precond1-5-stb* **apply** *simp*
apply(*rule allI*)
apply(*rule Basic*)
apply(*case-tac mp-alloc-precond1-4 t p sz timeout* $\cap$ $\{\!|$´*cur* = *Some t*$|\!\}$ $\cap$ $\{V\} = \{\}$)
apply *auto*[*1*] **apply** *clarify*
apply(*rule IntI*) **apply** *auto*[*1*]
apply(*simp add*:*Mem-pool-alloc-guar-def lvars-nochange1-def lvars-nochange-def*

gvars-conf-stable-def gvars-conf-def)
apply(*subgoal-tac* $(V, V(\!|$*lsizes* := (*lsizes* V)(t := [*ALIGN4* (*max-sz* (*mem-pool-info* V p))])$|\!)) \in$ *lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*]
apply(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def lvars-nochange-def*)
apply(*simp add*:*Mem-pool-alloc-guar-def lvars-nochange1-def lvars-nochange-def*

gvars-conf-stable-def gvars-conf-def)
apply(*subgoal-tac* $(V, V(\!|$*lsizes* := (*lsizes* V)(t := [*ALIGN4* (*max-sz* (*mem-pool-info* V p))])$|\!)) \in$ *lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*]
apply(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def lvars-nochange-def*)
apply(*simp add*:*alloc-memblk-valid-def*)
apply(*subgoal-tac* $(V, V(\!|$*lsizes* := (*lsizes* V)(t := [*ALIGN4* (*max-sz* (*mem-pool-info* V p))])$|\!)) \in$ *lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def*)
apply(*simp add*:*stable-def*)+

apply(*simp add*:*stm-def*)

apply(*rule Await*)
using *mp-alloc-precond1-5-stb* **apply** *simp*
using *mp-alloc-precond1-6-stb* **apply** *simp*
apply(*rule allI*)
apply(*rule Basic*)
apply(*case-tac mp-alloc-precond1-5 t p sz timeout* $\cap$ $\{\!|$´*cur* = *Some t*$|\!\}$ $\cap$ $\{V\} = \{\}$)
apply *auto*[*1*] **apply** *clarify*
apply(*rule IntI*) **apply** *auto*[*1*]
apply(*simp add*:*Mem-pool-alloc-guar-def lvars-nochange1-def lvars-nochange-def*

gvars-conf-stable-def gvars-conf-def)
apply(*subgoal-tac* $(V, V(\!|i := (i\ V)(t := 0)|\!))\in$*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*]
apply(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def lvars-nochange-def*)
apply(*simp add*:*Mem-pool-alloc-guar-def lvars-nochange1-def lvars-nochange-def*

gvars-conf-stable-def gvars-conf-def)
apply(*subgoal-tac* $(V, V(\!|i := (i\ V)(t := 0)|\!))\in$*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*]
apply(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def lvars-nochange-def*)
apply(*simp add*:*alloc-memblk-valid-def*)
apply(*rule conjI*)
apply(*subgoal-tac* $(V, V(\!|i := (i\ V)(t := 0)|\!))\in$*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def*)
apply(*simp add*:*inv-def inv-mempool-info-def*) **apply** (*meson Suc-leI*)
apply *simp* **apply**(*simp add*:*stable-def*) **using** *stable-id2* **apply** *auto*[*1*]

using *lsize-loop-stm*[*of t p sz timeout*] **apply** *clarsimp*

using *precnd17-bl-170* **apply** *simp*

apply(*rule Cond*)
using *mp-alloc-precond1-70-stb* **apply** *simp*

apply(*simp add*:*stm-def*)
apply(*rule Await*)
using *mp-alloc-precond1-70-1-stb* **apply** *simp*
using *mp-alloc-precond1-8-stb* **apply** *auto*[*1*]

apply(*rule allI*)
apply(*rule Basic*)
apply(*case-tac mp-alloc-precond1-70-1 t p sz timeout* $\cap$ $\{\!|$´*cur* = *Some t*$|\!\}$ $\cap$ $\{V\} = \{\}$)

apply *auto*[*1*] **apply** *clarify*
apply(*rule IntI*) **apply** *simp*
apply(*simp add*:*Mem-pool-alloc-guar-def gvars-conf-stable-def gvars-conf-def lvars-nochange-def*)
apply(*rule disjI1*)
apply(*subgoal-tac* $(V, V(\!|ret := (ret\ V)(t := ESIZEERR)|\!)) \in lvars\text{-}nochange1\text{-}4all$)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def*)
apply(*rule IntI*) **prefer** *2*
apply(*case-tac* $i\ V\ t = 0$) **apply**(*simp add*:*inv-def inv-mempool-info-def*)
apply *simp*
apply(*rule IntI*) **prefer** *2* **apply** *simp*
apply(*subgoal-tac* $(V, V(\!|ret := (ret\ V)(t := ESIZEERR)|\!)) \in lvars\text{-}nochange1\text{-}4all$)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def*)
apply *simp* **using** *stable-id2* **apply** *auto*[*1*] **using** *stable-id2* **apply** *auto*[*1*]

apply(*rule Cond*)
using *mp-alloc-precond1-70-2-stb* **apply** *simp*

apply(*simp add*:*stm-def*)
apply(*rule Await*)
using *mp-alloc-precond1-70-2-1-stb* **apply** *simp*
using *mp-alloc-precond1-8-stb* **apply** *auto*[*1*]

apply(*rule allI*)
apply(*rule Basic*)
apply(*case-tac* $mp\text{-}alloc\text{-}precond1\text{-}70\text{-}2\text{-}1\ t\ p\ sz\ timeout \cap \{\!|\,\acute{}cur = Some\ t|\!\} \cap \{V\} = \{\}$)
apply *auto*[*1*] **apply** *clarify*
apply(*rule IntI*) **apply** *simp*
apply(*simp add*:*Mem-pool-alloc-guar-def gvars-conf-stable-def gvars-conf-def lvars-nochange-def*)
apply(*rule disjI1*)
apply(*subgoal-tac* $(V, V(\!|ret := (ret\ V)(t := ENOMEM)|\!)) \in lvars\text{-}nochange1\text{-}4all$)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def*)
apply(*rule IntI*) **prefer** *2*
apply(*case-tac* $i\ V\ t = 0$) **apply**(*simp add*:*inv-def inv-mempool-info-def*)
apply *simp*
apply(*rule IntI*) **prefer** *2* **apply** *simp*
apply(*subgoal-tac* $(V, V(\!|ret := (ret\ V)(t := ENOMEM)|\!)) \in lvars\text{-}nochange1\text{-}4all$)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def*)
apply *simp* **using** *stable-id2* **apply** *auto*[*1*] **using** *stable-id2* **apply** *auto*[*1*]

apply(*rule Seq*[**where** *mid=mp-alloc-precond2-1 t p sz timeout*])

using *mp-alloc-stm3-lm* **apply** *simp*

apply(*rule Cond*)
using *mp-alloc-precond2-1-stb* **apply** *simp*

apply(*simp add:stm-def*)
apply(*rule Await*)
using *mp-alloc-precond2-1-0-stb* **apply** *simp*
using *mp-alloc-precond1-8-stb* **apply** *auto*[*1*]

apply(*rule allI*)
apply(*rule Basic*)
apply(*case-tac mp-alloc-precond2-1-0 t p sz timeout* $\cap$ $\{\!|$*´cur = Some t*$|\!\}$ $\cap$ $\{V\} = \{\}$)
apply *auto*[*1*] **apply** *clarify*
apply(*rule IntI*) **apply** *simp*
apply(*simp add:Mem-pool-alloc-guar-def gvars-conf-stable-def gvars-conf-def lvars-nochange-def*)
apply(*rule disjI1*)
apply(*subgoal-tac* $(V, V(\!|ret := (ret\ V)(t := EAGAIN)|\!)) \in$ *lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add:lvars-nochange1-4all-def lvars-nochange1-def*)
apply(*rule IntI*) **prefer** *2*
apply(*case-tac i V t = 0*) **apply**(*simp add:inv-def inv-mempool-info-def*)
apply *simp*
apply(*rule IntI*) **prefer** *2* **apply** *simp*
apply(*subgoal-tac* $(V, V(\!|ret := (ret\ V)(t := EAGAIN)|\!)) \in$ *lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add:lvars-nochange1-4all-def lvars-nochange1-def*)
apply *simp* **using** *stable-id2* **apply** *auto*[*1*] **using** *stable-id2* **apply** *auto*[*1*]

apply(*rule Seq*[**where** *mid=mp-alloc-precond2-1-4 t p sz timeout*])
apply(*rule Seq*[**where** *mid=mp-alloc-precond2-1-3 t p sz timeout*])
apply(*rule Seq*[**where** *mid=mp-alloc-precond2-1-2 t p sz timeout*])
apply(*rule Seq*[**where** *mid=mp-alloc-precond2-1-1-loopinv t p sz timeout*])

apply(*simp add:stm-def*)
apply(*rule Await*)
using *mp-alloc-precond2-1-1-stb* **apply** *simp*
using *mp-alloc-precond2-1-1-loopinv-stb* **apply** *simp*

apply(*rule allI*)
apply(*rule Basic*)
apply(*case-tac mp-alloc-precond2-1-1 t p sz timeout* ∩ {|´*cur* = *Some t*|} ∩ {*V*} = {})
apply *auto*[*1*] **apply** *clarify*
apply(*rule IntI*) **apply** *simp*
apply(*simp add*:*Mem-pool-alloc-guar-def gvars-conf-stable-def gvars-conf-def lvars-nochange-def*)
apply(*rule disjI1*)
apply(*subgoal-tac* (*V*, *V*(|*from-l* := (*from-l V*)(*t* := *free-l V t*)|))∈*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def*)
apply(*rule IntI*) **prefer** *2*
apply(*case-tac i V t* = *0*) **apply**(*simp add*:*inv-def inv-mempool-info-def*)
apply *simp*
apply(*rule IntI*) **prefer** *2* **apply** *simp*
apply(*subgoal-tac* (*V*, *V*(|*from-l* := (*from-l V*)(*t* := *free-l V t*)|))∈*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def*)
apply *simp* **using** *stable-id2* **apply** *auto*[*1*] **using** *stable-id2* **apply** *auto*[*1*]

using *mp-alloc-stm4-lm* **apply** *simp*

using *mp-alloc-stm5-lm* **apply** *simp*

using *mp-alloc-stm6-lm* **apply** *simp*

apply(*simp add*:*stm-def*)
apply(*rule Await*)
using *mp-alloc-precond2-1-4-stb* **apply** *simp*
using *mp-alloc-precond1-8-stb* **apply** *auto*[*1*]

apply(*rule allI*)
apply(*rule Basic*)
apply(*case-tac mp-alloc-precond2-1-4 t p sz timeout* ∩ {|´*cur* = *Some t*|} ∩ {*V*} = {})
apply *auto*[*1*] **apply** *clarify*
apply(*rule IntI*) **apply**(*simp add*:*Mem-pool-alloc-guar-def gvars-conf-stable-def gvars-conf-def lvars-nochange-def*)
apply(*rule disjI1*)
apply(*subgoal-tac* (*V*, *V*(|*ret* := (*ret V*)(*t* := *OK*)|))∈*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def*

lvars-nochange1-def)
apply(*rule IntI*) **apply**(*rule IntI*) **apply**(*rule IntI*) **apply**(*rule IntI*)
apply(*subgoal-tac* $(V, V(\!|ret := (ret\ V)(t := OK)|\!)) \in$ *lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def*)
apply *auto*[*1*] **apply** *auto*[*1*] **apply** *auto*[*1*]
apply *auto*[*1*] **apply**(*simp add*:*alloc-memblk-valid-def*) **apply** *auto*[*1*]
apply(*simp add*:*alloc-memblk-valid-def*) **apply** *auto*[*1*]
apply *simp* **using** *stable-id2* **apply** *auto*[*1*] **using** *stable-id2* **apply** *auto*[*1*]

apply(*simp add*:*Mem-pool-alloc-guar-def*)
apply(*simp add*:*Mem-pool-alloc-guar-def*)
apply(*simp add*:*Mem-pool-alloc-guar-def*)

apply(*rule Cond*)
using *mp-alloc-precond1-8-stb* **apply** *simp*

apply(*rule Seq*[**where** *mid*=*mp-alloc-precond1-8-1-1 t p sz timeout*])

apply(*simp add*:*stm-def*)
apply(*rule Await*)
using *mp-alloc-precond1-8-1-stb* **apply** *auto*[*1*]
using *mp-alloc-precond1-8-1-1-stb* **apply** *auto*[*1*]
apply(*rule allI*)
apply(*rule Basic*)
apply(*case-tac mp-alloc-precond1-8-1 t p sz timeout* $\cap$ $\{\!|\,\acute{}cur = Some\ t|\!\}$ $\cap \{V\} = \{\}$)
apply *auto*[*1*] **apply** *clarify*
apply(*rule IntI*)
apply(*simp add*:*Mem-pool-alloc-guar-def lvars-nochange1-def lvars-nochange-def*

gvars-conf-stable-def gvars-conf-def) **apply**(*rule disjI1*)
apply(*subgoal-tac* $(V, V(\!|rf := (rf\ V)(t := True)|\!)) \in$ *lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*]
apply(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def lvars-nochange-def*)
apply(*rule IntI*) **apply**(*rule IntI*) **apply**(*rule IntI*)
apply(*simp add*:*Mem-pool-alloc-guar-def lvars-nochange1-def lvars-nochange-def*

gvars-conf-stable-def gvars-conf-def)
apply(*subgoal-tac* $(V, V(\!|rf := (rf\ V)(t := True)|\!)) \in$ *lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*]
apply(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def lvars-nochange-def*)
apply *simp*
apply(*simp add*:*alloc-memblk-valid-def*) **apply** *simp*

apply(*simp add*:*stable-def*)+

apply(*rule Cond*)
using *mp-alloc-precond1-8-1-1-stb* **apply** *auto*[*1*]

apply(*simp add*:*stm-def*)
apply(*rule Await*)
using *mp-alloc-precond1-8-1-2-stb* **apply** *auto*[*1*]
using *mp-alloc-precond7-stb* **apply** *auto*[*1*]
apply(*rule allI*)
apply(*rule Basic*)
apply(*case-tac mp-alloc-precond1-8-1-2 t p sz timeout* $\cap$ ⦃´*cur* = *Some t*⦄ $\cap$ $\{V\} = \{\}$)
apply *auto*[*1*] **apply** *auto*[*1*]
apply(*simp add*:*Mem-pool-alloc-guar-def gvars-conf-stable-def gvars-conf-def lvars-nochange-def*)
apply(*rule disjI1*)
apply(*subgoal-tac* $(V, V(\!|ret := (ret\ V)(t := ENOMEM)|\!))\in$*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def*)
apply(*subgoal-tac* $(V, V(\!|ret := (ret\ V)(t := ENOMEM)|\!))\in$*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*] **apply**(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def*)
apply *simp*
apply(*simp add*:*stable-def*) **apply** *auto*[*1*]
apply(*simp add*:*stable-def*)

apply(*simp add*:*Skip-def*)
apply(*rule Basic*)
apply *auto*[*1*] **apply**(*simp add*:*Mem-pool-alloc-guar-def*) **apply** *auto*[*1*]

using *mp-alloc-precond1-8-1-3-stb* **apply** *auto*[*1*]
using *mp-alloc-precond7-stb* **apply** *auto*[*1*]

apply(*simp add*:*Mem-pool-alloc-guar-def*)

apply(*rule Cond*)
using *mp-alloc-precond1-8-2-stb* **apply** *simp*

apply(*simp add*:*Skip-def*)
apply(*rule Basic*) **apply** *auto*[*1*]
apply(*simp add*:*Mem-pool-alloc-guar-def*) **apply** *auto*[*1*]
using *mp-alloc-precond1-8-2-1-stb* **apply** *simp*
using *mp-alloc-precond7-stb* **apply** *simp*

apply(*rule Seq*[**where** *mid=mp-alloc-precond1-8-2-2 t p sz timeout*])

using *mp-alloc-stm7-lm* **apply** *simp*

apply(*rule Cond*)
using *mp-alloc-precond1-8-2-2-stb* **apply** *auto*[*1*]
apply(*rule Seq*[**where** *mid=mp-alloc-precond1-8-2-4 t p sz timeout*])

apply(*unfold stm-def*)[*1*]
apply(*rule Await*)
using *mp-alloc-precond1-8-2-3-stb mp-pred1823-eq* **apply** *auto*[*1*]
using *mp-alloc-precond1-8-2-4-stb* **apply** *blast*
apply(*rule allI*)
apply(*rule Basic*)
apply(*case-tac mp-alloc-precond1-8-2-3 t p sz timeout* ∩ {|´*cur* = *Some t*|} ∩ {*V*} = {})
apply *auto*[*1*] **apply** *auto*[*1*]
apply(*simp add*:*Mem-pool-alloc-guar-def*) **apply**(*rule disjI1*)
apply(*simp add*:*Mem-pool-alloc-guar-def lvars-nochange1-def lvars-nochange-def gvars-conf-stable-def gvars-conf-def*)
apply(*subgoal-tac* (*V*,*V*(|*tmout* := (*tmout V*)(*t* := *int* (*endt V t*) − *int* (*tick V*))|))∈*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*]
apply(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def lvars-nochange-def*)
apply(*subgoal-tac* (*V*,*V*(|*tmout* := (*tmout V*)(*t* := *int* (*endt V t*) − *int* (*tick V*))|))∈*lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*]
apply(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def lvars-nochange-def*) **apply** *auto*[*1*]
apply(*simp add*:*stable-def*) **apply** *auto*[*1*] **apply**(*simp add*:*stable-def*)
apply *auto*[*1*]

apply(*rule Cond*)
using *mp-alloc-precond1-8-2-4-stb* **apply** *blast*

apply(*rule Seq*[**where** *mid=mp-alloc-precond1-8-2-5 t p sz timeout*])

apply(*unfold stm-def*)[*1*]
apply(*rule Await*)
using *mp-alloc-precond1-8-2-40-stb* **apply** *blast*
using *mp-alloc-precond1-8-2-5-stb* **apply** *blast*
apply(*rule allI*)
apply(*rule Basic*)
apply(*case-tac mp-alloc-precond1-8-2-40 t p sz timeout* ∩ {|´*cur* = *Some t*|} ∩ {*V*} = {})

apply *auto*[*1*] **apply** *auto*[*1*]
using *mp-alloc-stm8-guar* **apply** *simp*
apply(*subgoal-tac* $(V, V(\!|rf := (rf\ V)(t := True)|\!)) \in$ *lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*]
apply(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def lvars-nochange-def*)
using *mp-alloc-stm8-guar* **apply** *simp*
apply(*subgoal-tac* $(V, V(\!|rf := (rf\ V)(t := True)|\!)) \in$ *lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*]
apply(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def lvars-nochange-def*)
using *mp-alloc-stm8-guar* **apply** *simp*
apply *simp* **apply**(*simp add*:*stable-def*) **apply** *auto*[*1*] **apply**(*simp add*:*stable-def*) **apply** *auto*[*1*]

apply(*unfold stm-def*)[*1*]
apply(*rule Await*)
using *mp-alloc-precond1-8-2-5-stb* **apply** *blast*
using *mp-alloc-precond7-stb* **apply** *blast*
apply(*rule allI*)
apply(*rule Basic*)
apply(*case-tac mp-alloc-precond1-8-2-5 t p sz timeout* $\cap$ $\{\!|$ ´*cur* = *Some t*$|\!\}$ $\cap$ $\{V\} = \{\}$)
apply *auto*[*1*] **apply** *auto*[*1*]
using *mp-alloc-stm9-guar* **apply** *simp*
apply(*subgoal-tac* $(V, V(\!|ret := (ret\ V)(t := ETIMEOUT)|\!)) \in$ *lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*]
apply(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def lvars-nochange-def*)
using *mp-alloc-stm9-guar* **apply** *simp*
apply(*subgoal-tac* $(V, V(\!|ret := (ret\ V)(t := ETIMEOUT)|\!)) \in$ *lvars-nochange1-4all*)
using *glnochange-inv0* **apply** *auto*[*1*]
apply(*simp add*:*lvars-nochange1-4all-def lvars-nochange1-def lvars-nochange-def*)
apply *simp* **apply**(*simp add*:*stable-def*) **apply** *auto*[*1*] **apply**(*simp add*:*stable-def*)

apply(*unfold Skip-def*)[*1*]
apply(*rule Basic*)
apply *auto*[*1*]
prefer *2* **using** *mp-alloc-precond1-8-2-41-stb* **apply** *fast*
prefer *2* **using** *mp-alloc-precond7-stb* **apply** *blast*
apply(*simp add*:*Mem-pool-alloc-guar-def*) **apply** *auto*[*1*]

apply(*simp add*:*Mem-pool-alloc-guar-def*)

apply(*unfold Skip-def*)[*1*]
apply(*rule Basic*)

apply *auto*[*1*]
apply(*simp add*:*Mem-pool-alloc-guar-def*) **apply** *auto*[*1*]
using *mp-alloc-precond1-8-2-20-stb* **apply** *fast*
using *mp-alloc-precond7-stb* **apply** *blast*

apply(*simp add*:*Mem-pool-alloc-guar-def*)+
done

end

theory *memory-manage-sys*
imports *rg-cond func-cor-other func-cor-mempoolfree func-cor-mempoolalloc*
begin

11 formal specification of Zephyr memory management

definition *Mem-pool-alloc-RGF* :: *Thread* ⇒ *mempool-ref* ⇒ *nat* ⇒ *int* ⇒ (*EventLabel*, *Core*, *State*, *State com option*) *rgformula-e*
where *Mem-pool-alloc-RGF t p sz timeout*
≡ (*Mem-pool-alloc t p sz timeout*, *Mem-pool-alloc-RGCond t p sz timeout*)

definition *Mem-pool-free-RGF* :: *Thread* ⇒ *Mem-block* ⇒ (*EventLabel*, *Core*, *State*, *State com option*) *rgformula-e*
where *Mem-pool-free-RGF t b* ≡ (*Mem-pool-free t b*, *Mem-pool-free-RGCond t b*)

definition *Schedule-RGF* :: *Thread* ⇒ (*EventLabel*, *Core*, *State*, *State com option*) *rgformula-e*
where *Schedule-RGF t* ≡ (*Schedule t*, *Schedule-RGCond t*)

definition *Tick-RGF* :: (*EventLabel*, *Core*, *State*, *State com option*) *rgformula-e*
where *Tick-RGF* ≡ (*Tick*, *Tick-RGCond*)

definition *Thread-RGF* :: *Thread* ⇒ (*EventLabel*, *Core*, *State*, *State com option*) *rgformula-es*
where *Thread-RGF t* ≡ (*rgf-EvtSys* ((⋃(*p*, *sz*, *timeout*).{*Mem-pool-alloc-RGF t p sz timeout*}) ∪
(⋃*b*. {*Mem-pool-free-RGF t b*})),
RG[(*Mem-pool-free-pre t* ∩ *Mem-pool-alloc-pre t*),
(*Mem-pool-free-rely t* ∩ *Mem-pool-alloc-rely t*),
(*Mem-pool-free-guar t* ∪ *Mem-pool-alloc-guar t*),
(*Mem-pool-free-post t* ∪ (⋃(*p*, *sz*, *timeout*).*Mem-pool-alloc-post t p sz timeout*))])

definition *Scheduler-RGF* :: (*EventLabel*, *Core*, *State*, *State com option*) *rgformula-es*
where *Scheduler-RGF* ≡ (*rgf-EvtSys* (⋃*t*. {*Schedule-RGF t*}),
RG[{*s*. *inv s*}, *Schedule-rely*, *Schedule-guar*, {*s*. *inv s*}])

definition *Timer-RGF* :: (*EventLabel*, *Core*, *State*, *State com option*) *rgformula-es*
where *Timer-RGF* ≡ (*rgf-EvtSys* {*Tick-RGF*},
    RG[⦃*True*⦄, *Tick-rely*, *Tick-guar*, ⦃*True*⦄])

definition *Memory-manage-system-Spec* :: (*EventLabel*, *Core*, *State*, *State com option*) *rgformula-par*
where *Memory-manage-system-Spec k* ≡
  case k of ($\mathcal{T}$ *t*) ⇒ *Thread-RGF t*
    | $\mathcal{S}$ ⇒ *Scheduler-RGF*
    | *Timer* ⇒ *Timer-RGF*

12 functional correctness of the whole specification

definition *sys-rely* ≡ {}

definition *sys-guar* ≡ *Tick-guar* ∪ *Schedule-guar* ∪ ($\bigcup$ *t*. (*Mem-pool-free-guar t* ∪ *Mem-pool-alloc-guar t*))

lemma *scheduler-esys-sat*: Γ ⊢ *fst* (*Memory-manage-system-Spec* $\mathcal{S}$)
  sat_s [Pre_{es} (*Memory-manage-system-Spec* $\mathcal{S}$),
    $Rely_{es}$ (*Memory-manage-system-Spec* $\mathcal{S}$),
    $Guar_{es}$ (*Memory-manage-system-Spec* $\mathcal{S}$),
    $Post_{es}$ (*Memory-manage-system-Spec* $\mathcal{S}$)]
apply(*simp add*:*Memory-manage-system-Spec-def Scheduler-RGF-def Schedule-RGF-def*)
 apply(*rule EvtSys-h*)
  apply *auto*[*1*] **apply**(*simp add*:E_e*-def* Pre_e*-def* $Rely_e$*-def* $Guar_e$*-def* $Post_e$*-def*)
   using *Schedule-satRG* **apply**(*simp add*:*Schedule-RGCond-def Evt-sat-RG-def* Pre_f*-def* $Rely_f$*-def* $Guar_f$*-def* $Post_f$*-def*)
   apply *fast*
   apply(*simp add*:Pre_{es}*-def* Pre_e*-def Schedule-RGCond-def*)
   apply(*simp add*:$Rely_{es}$*-def* $Rely_e$*-def Schedule-RGCond-def*)
   apply(*simp add*:$Guar_{es}$*-def* $Guar_e$*-def Schedule-RGCond-def*)
   apply(*simp add*:$Post_{es}$*-def* $Post_e$*-def Schedule-RGCond-def*)
   apply(*simp add*:$Post_e$*-def* Pre_e*-def Schedule-RGCond-def getrgformula-def*)
   apply(*simp add*:Pre_{es}*-def* $Rely_{es}$*-def getrgformula-def*)
    using *stable-inv-sched-rely* **apply**(*simp add*:*stable-def*)
   apply(*simp add*:$Guar_{es}$*-def getrgformula-def Schedule-guar-def*)
done

lemma *thread-esys-sat*: Γ ⊢ *fst* (*Memory-manage-system-Spec* ($\mathcal{T}$ *x*))
  sat_s [Pre_{es} (*Memory-manage-system-Spec* ($\mathcal{T}$ *x*)),
    $Rely_{es}$ (*Memory-manage-system-Spec* ($\mathcal{T}$ *x*)),
    $Guar_{es}$ (*Memory-manage-system-Spec* ($\mathcal{T}$ *x*)),
    $Post_{es}$ (*Memory-manage-system-Spec* ($\mathcal{T}$ *x*))]

apply(*simp add*:*Memory-manage-system-Spec-def Thread-RGF-def Mem-pool-alloc-RGF-def Mem-pool-free-RGF-def*)
apply(*rule EvtSys-h*)
apply *auto*[*1*]
apply(*simp add*:E_e*-def* Pre_e*-def* $Rely_e$*-def* $Guar_e$*-def* $Post_e$*-def getrgformula-def Mem-pool-alloc-RGCond-def*)
using *Mempool-alloc-satRG* **apply**(*simp add*:*Evt-sat-RG-def Mem-pool-alloc-RGCond-def*

getrgformula-def Pre_f*-def* $Rely_f$*-def* $Guar_f$*-def* $Post_f$*-def*) **apply** *fast*
apply(*simp add*:E_e*-def* Pre_e*-def* $Rely_e$*-def* $Guar_e$*-def* $Post_e$*-def getrgformula-def Mem-pool-free-RGCond-def*)
using *Mempool-free-satRG* **apply**(*simp add*:*Evt-sat-RG-def Mem-pool-free-RGCond-def*

getrgformula-def Pre_f*-def* $Rely_f$*-def* $Guar_f$*-def* $Post_f$*-def*) **apply** *fast*
apply *auto*[*1*]
apply(*simp add*:Pre_{es}*-def* Pre_e*-def getrgformula-def Mem-pool-alloc-RGCond-def*)
apply(*simp add*:Pre_{es}*-def* Pre_e*-def getrgformula-def Mem-pool-free-RGCond-def*)
apply *auto*[*1*]
apply(*simp add*:$Rely_{es}$*-def* $Rely_e$*-def getrgformula-def Mem-pool-alloc-RGCond-def*)
apply(*simp add*:$Rely_{es}$*-def* $Rely_e$*-def getrgformula-def Mem-pool-free-RGCond-def*)
apply *auto*[*1*]
apply(*simp add*:$Guar_{es}$*-def* $Guar_e$*-def getrgformula-def Mem-pool-alloc-RGCond-def*)
apply(*simp add*:$Guar_{es}$*-def* $Guar_e$*-def getrgformula-def Mem-pool-free-RGCond-def*)
apply *auto*[*1*]
apply(*simp add*:$Post_{es}$*-def* $Post_e$*-def getrgformula-def Mem-pool-alloc-RGCond-def*)
apply *auto*[*1*]
apply(*simp add*:$Post_{es}$*-def* $Post_e$*-def getrgformula-def Mem-pool-free-RGCond-def*)
apply *auto*[*1*]
apply(*simp add*:$Post_e$*-def* Pre_e*-def Mem-pool-alloc-RGCond-def getrgformula-def Mem-pool-alloc-post-def*)
apply(*simp add*:$Post_e$*-def* Pre_e*-def Mem-pool-alloc-RGCond-def Mem-pool-free-RGCond-def*

getrgformula-def Mem-pool-alloc-post-def)
apply(*simp add*:$Post_e$*-def* Pre_e*-def Mem-pool-alloc-RGCond-def Mem-pool-free-RGCond-def*

getrgformula-def Mem-pool-free-post-def)
apply(*simp add*:$Post_e$*-def* Pre_e*-def Mem-pool-free-RGCond-def getrgformula-def Mem-pool-free-post-def*)
apply(*simp add*:Pre_{es}*-def* $Rely_{es}$*-def getrgformula-def Mem-pool-free-rely-def Mem-pool-alloc-rely-def*)

defer *1*
apply(*simp add*:$Guar_{es}$*-def getrgformula-def Mem-pool-free-guar-def*)
using *mem-pool-free-pre-stb* **apply**(*simp add*:*Mem-pool-free-rely-def*)
done

lemma *timer-esys-sat*: $\Gamma \vdash$ *fst* (*Memory-manage-system-Spec Timer*)
sat_s [Pre_{es} (*Memory-manage-system-Spec Timer*),

Rely$_{es}$ (*Memory-manage-system-Spec Timer*),
Guar$_{es}$ (*Memory-manage-system-Spec Timer*),
Post$_{es}$ (*Memory-manage-system-Spec Timer*)]
apply(*simp add:Memory-manage-system-Spec-def Timer-RGF-def Tick-RGF-def*)
apply(*rule EvtSys-h*)
apply *auto*[*1*] **apply**(*simp add:E*$_e$*-def Pre*$_e$*-def Rely*$_e$*-def Guar*$_e$*-def Post*$_e$*-def*)
using *Tick-satRG* **apply**(*simp add:Tick-RGCond-def Evt-sat-RG-def Pre*$_f$*-def Rely*$_f$*-def Guar*$_f$*-def Post*$_f$*-def*)
apply *fast*
apply(*simp add:Pre*$_{es}$*-def Pre*$_e$*-def Tick-RGCond-def*)
apply(*simp add:Rely*$_{es}$*-def Rely*$_e$*-def Tick-RGCond-def*)
apply(*simp add:Guar*$_{es}$*-def Guar*$_e$*-def Tick-RGCond-def*)
apply(*simp add:Post*$_{es}$*-def Post*$_e$*-def Tick-RGCond-def*)
apply(*simp add:Post*$_e$*-def Pre*$_e$*-def Tick-RGCond-def getrgformula-def*)
apply(*simp add:Pre*$_{es}$*-def Rely*$_{es}$*-def getrgformula-def*)
using *stable-inv-sched-rely* **apply**(*simp add:stable-def*)
apply(*simp add:Guar*$_{es}$*-def getrgformula-def Tick-guar-def*)
done

lemma *esys-sat*: $\Gamma \vdash$ *fst* (*Memory-manage-system-Spec k*)
sat$_s$ [*Pre*$_{es}$ (*Memory-manage-system-Spec k*),
Rely$_{es}$ (*Memory-manage-system-Spec k*),
Guar$_{es}$ (*Memory-manage-system-Spec k*),
Post$_{es}$ (*Memory-manage-system-Spec k*)]
apply(*induct k*)
using *scheduler-esys-sat* **apply** *fast*
using *thread-esys-sat* **apply** *fast*
using *timer-esys-sat* **apply** *fast*
done

lemma *s0-esys-pre*: $\{s0\} \subseteq$ *Pre*$_{es}$ (*Memory-manage-system-Spec k*)
apply(*induct k*)
apply(*simp add:Memory-manage-system-Spec-def Pre*$_{es}$*-def Scheduler-RGF-def getrgformula-def*)
using *s0-inv* **apply** *fast*
apply(*simp add:Memory-manage-system-Spec-def Pre*$_{es}$*-def Thread-RGF-def getrgformula-def*)
using *s0-inv s0a4 s0a10* **apply** *auto*[*1*]
apply(*simp add:Memory-manage-system-Spec-def Pre*$_{es}$*-def Timer-RGF-def getrgformula-def*)
done

lemma *alloc-free-eq-guar*: *Mem-pool-free-guar x* = *Mem-pool-alloc-guar x*
by(*simp add:Mem-pool-free-guar-def Mem-pool-alloc-guar-def*)

lemma *alloc-free-eq-rely*: *Mem-pool-free-rely x* = *Mem-pool-alloc-rely x*
by(*simp add:Mem-pool-free-rely-def Mem-pool-alloc-rely-def*)

lemma *esys-guar-in-other*:
jj $\neq$ *k* $\longrightarrow$ *Guar*$_{es}$ (*Memory-manage-system-Spec jj*) $\subseteq$ *Rely*$_{es}$ (*Memory-manage-system-Spec k*)

apply *auto*
apply(*induct jj*)
apply(*induct k*)
apply *simp*
apply(*simp add*:$Guar_{es}$*-def* $Rely_{es}$*-def Memory-manage-system-Spec-def Scheduler-RGF-def Thread-RGF-def getrgformula-def*)
using *schedguar-in-allocrely* **apply**(*simp add*:*Mem-pool-free-rely-def Mem-pool-alloc-rely-def*)
apply *auto*[*1*]
apply(*simp add*:$Guar_{es}$*-def* $Rely_{es}$*-def Memory-manage-system-Spec-def Scheduler-RGF-def Timer-RGF-def getrgformula-def*)
using *schedguar-in-tickrely* **apply** *auto*[*1*]
apply(*induct k*)
apply(*simp add*:$Guar_{es}$*-def* $Rely_{es}$*-def Memory-manage-system-Spec-def Scheduler-RGF-def Thread-RGF-def getrgformula-def*)
apply *auto*[*1*]
using *allocguar-in-schedrely alloc-free-eq-guar* **apply** *fast*
using *allocguar-in-schedrely* **apply** *fast*
apply(*simp add*:$Guar_{es}$*-def* $Rely_{es}$*-def Memory-manage-system-Spec-def Thread-RGF-def getrgformula-def*)
apply *auto*[*1*]
using *allocguar-in-allocrely alloc-free-eq-guar alloc-free-eq-rely* **apply** *fast+*
apply(*simp add*:$Guar_{es}$*-def* $Rely_{es}$*-def Memory-manage-system-Spec-def Timer-RGF-def Thread-RGF-def getrgformula-def*)
apply *auto*[*1*]
using *allocguar-in-tickrely alloc-free-eq-guar alloc-free-eq-rely* **apply** *fast+*
apply(*induct k*)
apply(*simp add*:$Guar_{es}$*-def* $Rely_{es}$*-def Memory-manage-system-Spec-def Scheduler-RGF-def Timer-RGF-def getrgformula-def*)
using *tickguar-in-schedrely* **apply** *fast*
apply(*simp add*:$Guar_{es}$*-def* $Rely_{es}$*-def Memory-manage-system-Spec-def Thread-RGF-def Timer-RGF-def getrgformula-def*)
apply *auto*[*1*]
using *tickguar-in-allocrely alloc-free-eq-guar alloc-free-eq-rely* **apply** *fast+*
done

lemma *esys-guar-in-sys*: $Guar_{es}$ (*Memory-manage-system-Spec k*) $\subseteq$ *sys-guar*
apply(*induct k*)
apply(*simp add*:$Guar_{es}$*-def Memory-manage-system-Spec-def Scheduler-RGF-def getrgformula-def sys-guar-def*) **apply** *auto*[*1*]
apply(*simp add*:$Guar_{es}$*-def Memory-manage-system-Spec-def Thread-RGF-def getrgformula-def sys-guar-def*) **apply** *auto*[*1*]
apply(*simp add*:$Guar_{es}$*-def Memory-manage-system-Spec-def Timer-RGF-def getrgformula-def sys-guar-def*) **apply** *auto*[*1*]
done

lemma *mem-sys-sat*: $\Gamma \vdash$ *Memory-manage-system-Spec SAT* [{*s0*}, *sys-rely*, *sys-guar*, *UNIV*]
apply(*rule ParallelESys*[*of* Γ *Memory-manage-system-Spec*{*s0*} *sys-rely sys-guar UNIV*])

apply *clarify* **using** *esys-sat* **apply** *fast*
using *s0-esys-pre* **apply** *fast*
apply(*simp add*:*sys-rely-def*)
using *esys-guar-in-other* **apply** *fast*
using *esys-guar-in-sys* **apply** *fast*
apply *simp*
done

end

theory *memory-management-inv*
imports *memory-manage-sys*
begin

13 invariant verification

theorem *invariant-presv-pares* Γ *inv* (*paresys-spec Memory-manage-system-Spec*) {*s0*} *sys-rely*
apply(*rule invariant-theorem*[**where** *G*=*sys-guar* **and** *pst* = *UNIV*])
using *mem-sys-sat* **apply** *fast*
apply(*simp add*:*sys-rely-def stable-def*)
apply(*simp add*:*sys-guar-def*)
apply(*rule stable-un-R*) **apply**(*rule stable-un-R*)
using *tick-guar-stb-inv* **apply**(*simp add*:*stable-def*)
using *sched-guar-stb-inv* **apply**(*simp add*:*stable-def*)
apply(*rule stable-un-S*) **apply** *clarify* **apply**(*rule stable-un-R*)
using *alloc-guar-stb-inv alloc-free-eq-guar* **apply**(*simp add*:*stable-def*)
using *alloc-guar-stb-inv* **apply**(*simp add*:*stable-def*)
using *s0-inv* **apply** *simp*
done

end