A Theory of Architectural Design Patterns

Diego Marmsoler

December 14, 2021

Abstract

The following document formalizes and verifies several architectural design patterns [1]. Each pattern specification is formalized in terms of a locale where the locale assumptions correspond to the assumptions which a pattern poses on an architecture. Thus, pattern specifications may build on top of each other by interpreting the corresponding locale. A pattern is verified using the framework provided by the AFP entry *Dynamic Architectures* [3].

Currently, the document consists of formalizations of 4 different patterns: the singleton, the publisher subscriber, the blackboard pattern, and the blockchain pattern. Thereby, the publisher component of the publisher subscriber pattern is modeled as an instance of the singleton pattern and the blackboard pattern is modeled as an instance of the publisher subscriber pattern.

In general, this entry provides the first steps towards an overall theory of architectural design patterns [2].

Contents

1 A Theory of Singletons **2**
- 1.1 Singletons . . . 2
  - 1.1.1 Calculus Interpretation . . . 2
  - 1.1.2 Architectural Guarantees . . . 2

2 A Theory of Publisher-Subscriber Architectures **5**
- 2.1 Subscriptions . . . 5
- 2.2 Publisher-Subscriber Architectures . . . 5
  - 2.2.1 Calculus Interpretation . . . 5
  - 2.2.2 Results from Singleton . . . 6
  - 2.2.3 Architectural Guarantees . . . 6

3 A Theory of Blackboard Architectures **6**
- 3.1 Problems and Solutions . . . 6
- 3.2 Blackboard Architectures . . . 7
  - 3.2.1 Calculus Interpretation . . . 8
  - 3.2.2 Results from Singleton . . . 8
  - 3.2.3 Results from Publisher Subscriber . . . 8
  - 3.2.4 Knowledge Sources . . . 8
  - 3.2.5 Architectural Guarantees . . . 8

4 Some Auxiliary Results **16**

5 Relative Frequency LTL **18**

6 Blockchain Architectures **28**
6.1 Blockchains . . . 28
6.2 Blockchain Architectures . . . 29
6.2.1 Component Behavior . . . 31
6.2.2 Maximal Honest Blockchains . . . 36
6.2.3 Honest Proof of Work . . . 36
6.2.4 History . . . 40
6.2.5 Blockchain Development . . . 48

1 A Theory of Singletons

In the following, we formalize the specification of the singleton pattern as described in [4].

theory *Singleton*
imports *DynamicArchitectures.Dynamic-Architecture-Calculus*
begin

1.1 Singletons

In the following we formalize a variant of the Singleton pattern.

locale *singleton* = *dynamic-component cmp active*
  for *active* :: *'id* ⇒ *cnf* ⇒ *bool* (‖-‖_ [*0*,*110*]*60*)
  and *cmp* :: *'id* ⇒ *cnf* ⇒ *'cmp* (*σ*_(-) [*0*,*110*]*60*) +
assumes *alwaysActive*: ⋀*k*. ∃ *id*. ‖*id*‖$_k$
  and *unique*: ∃ *id*. ∀ *k*. ∀ *id'*. (‖*id'*‖$_k$ ⟶ *id* = *id'*)
begin

1.1.1 Calculus Interpretation

baIA: ⟦∃ *i*≥*n*. ‖*c*‖$_{t\ i}$; *φ* (*σ*$_c$*t* ⟨*c* → *t*⟩$_n$)⟧ ⟹ *eval c t t' n* [*φ*]$_b$

baIN1: ⟦∃ *i*. ‖*c*‖$_{t\ i}$; ¬ (∃ *i*≥*n*. ‖*c*‖$_{t\ i}$); *φ* (*t'* (*n* − ⟨*c* ∧ *t*⟩ − *1*))⟧ ⟹ *eval c t t' n* [*φ*]$_b$

baIN2: ⟦∄ *i*. ‖*c*‖$_{t\ i}$; *φ* (*t' n*)⟧ ⟹ *eval c t t' n* [*φ*]$_b$

1.1.2 Architectural Guarantees

definition *the-singleton* ≡ *THE id*. ∀ *k*. ∀ *id'*. ‖*id'*‖$_k$ ⟶ *id'* = *id*

theorem *ts-prop*:
  fixes *k*::*cnf*
  shows ⋀*id*. ‖*id*‖$_k$ ⟹ *id* = *the-singleton*
    and ‖*the-singleton*‖$_k$
proof −
  { fix *id*
    assume *a1*: ‖*id*‖$_k$
  have (*THE id*. ∀ *k*. ∀ *id'*. ‖*id'*‖$_k$ ⟶ *id'* = *id*) = *id*
  proof (*rule the-equality*)
    show ∀ *k id'*. ‖*id'*‖$_k$ ⟶ *id'* = *id*
    proof
      fix *k* **show** ∀ *id'*. ‖*id'*‖$_k$ ⟶ *id'* = *id*
      proof
        fix *id'* **show** ‖*id'*‖$_k$ ⟶ *id'* = *id*
        proof

```
        assume ‖id′‖_k
        from unique have ∃ id. ∀ k. ∀ id′. (‖id′‖_k ⟶ id = id′) .
        then obtain i′′ where ∀ k. ∀ id′. (‖id′‖_k ⟶ i′′ = id′) by auto
          with ‹‖id′‖_k› have id=i′′ and id′=i′′ using a1 by auto
        thus id′ = id by simp
      qed
    qed
  qed
  next
    fix i′′ show ∀ k id′. ‖id′‖_k ⟶ id′ = i′′ ⟹ i′′ = id using a1 by auto
  qed
  hence ‖id‖_k ⟹ id = the-singleton by (simp add: the-singleton-def)
  } note g1 = this
  thus ⋀id. ‖id‖_k ⟹ id = the-singleton by simp

  from alwaysActive obtain id where ‖id‖_k by blast
  with g1 have id = the-singleton by simp
  with ‹‖id‖_k› show ‖the-singleton‖_k by simp
qed
declare ts-prop(2)[simp]

lemma lNact-active[simp]:
  fixes cid t n
  shows ⟨the-singleton ⇐ t⟩_n = n
  using lNact-active ts-prop(2) by auto

lemma lNxt-active[simp]:
  fixes cid t n
  shows ⟨the-singleton → t⟩_n = n
by (simp add: nxtAct-active)

lemma baI[intro]:
  fixes t n a
  assumes φ (σ_the-singleton(t n))
  shows eval the-singleton t t′ n [φ]_b using assms by (simp add: baIANow)

lemma baE[elim]:
  fixes t n a
  assumes eval the-singleton t t′ n [φ]_b
  shows φ (σ_the-singleton(t n)) using assms by (simp add: baEANow)

lemma evtE[elim]:
  fixes t id n a
  assumes eval the-singleton t t′ n (◇_b γ)
  shows ∃ n′≥n. eval the-singleton t t′ n′ γ
proof −
  have ‖the-singleton‖_t n by simp
  with assms obtain n′ where n′≥⟨the-singleton → t⟩_n and (∃ i≥n′. ‖the-singleton‖_t i ∧
    (∀ n′′≥⟨the-singleton ⇐ t⟩_n′. n′′ ≤ ⟨the-singleton → t⟩_n′ ⟶ eval the-singleton t t′ n′′ γ)) ∨
    ¬ (∃ i≥n′. ‖the-singleton‖_t i) ∧ eval the-singleton t t′ n′ γ using evtEA[of n the-singleton t] by blast
  moreover have ‖the-singleton‖_t n′ by simp
  ultimately have
    ∀ n′′≥⟨the-singleton ⇐ t⟩_n′. n′′ ≤ ⟨the-singleton → t⟩_n′ ⟶ eval the-singleton t t′ n′′ γ by auto
  hence eval the-singleton t t′ n′ γ by simp
  moreover from ‹n′≥⟨the-singleton → t⟩_n› have n′≥n by (simp add: nxtAct-active)
```

ultimately show *?thesis* **by** *auto*
qed

lemma *globE*[*elim*]:
fixes t *id* n a
assumes *eval the-singleton* t t' n $(\Box_b\ \gamma)$
shows $\forall\, n'{\geq}n.$ *eval the-singleton* t t' n' γ
proof
fix n' **show** $n \leq n' \longrightarrow$ *eval the-singleton* t t' n' γ
proof
assume $n{\leq}n'$
hence $\langle$*the-singleton* $\Leftarrow t\rangle_n \leq n'$ **by** *simp*
moreover have $\|$*the-singleton*$\|_{t\ n}$ **by** *simp*
ultimately show *eval the-singleton* t t' n' γ
using ‹*eval the-singleton* t t' n $(\Box_b\ \gamma)$› *globEA* **by** *blast*
qed
qed

lemma *untilI*[*intro*]:
fixes t::*nat* $\Rightarrow$ *cnf*
and t'::*nat* $\Rightarrow$ *'cmp*
and n::*nat*
and n'::*nat*
assumes $n'{\geq}n$
and *eval the-singleton* t t' n' γ
and $\bigwedge n''.\ [\![n{\leq}n'';\ n''{<}n']\!] \Longrightarrow$ *eval the-singleton* t t' n'' γ'
shows *eval the-singleton* t t' n $(\gamma'\ \mathfrak{U}_b\ \gamma)$
proof −
have $\|$*the-singleton*$\|_{t\ n}$ **by** *simp*
moreover from ‹$n'{\geq}n$› **have** $\langle$*the-singleton* $\Leftarrow t\rangle_n \leq n'$ **by** *simp*
moreover have $\|$*the-singleton*$\|_{t\ n'}$ **by** *simp*
moreover have
$\exists\, n''{\geq}\langle$*the-singleton* $\Leftarrow t\rangle_{n'}.\ n'' \leq \langle$*the-singleton* $\rightarrow t\rangle_{n'} \wedge$ *eval the-singleton* t t' n'' $\gamma \wedge$
$(\forall\, n'''{\geq}\langle$*the-singleton* $\rightarrow t\rangle_n.\ n''' < \langle$*the-singleton* $\Leftarrow t\rangle_{n''} \longrightarrow$
$(\exists\, n''''{\geq}\langle$*the-singleton* $\Leftarrow t\rangle_{n'''}.\ n'''' \leq \langle$*the-singleton* $\rightarrow t\rangle_{n'''} \wedge$ *eval the-singleton* t t' n'''' $\gamma'))$
proof −
have $n'{\geq}\langle$*the-singleton* $\Leftarrow t\rangle_{n'}$ **by** *simp*
moreover have $n' \leq \langle$*the-singleton* $\rightarrow t\rangle_{n'}$ **by** *simp*
moreover from *assms*(*3*) **have** $(\forall\, n''{\geq}\langle$*the-singleton* $\rightarrow t\rangle_n.\ n'' < \langle$*the-singleton* $\Leftarrow t\rangle_{n'} \longrightarrow$
$(\exists\, n'''{\geq}\langle$*the-singleton* $\Leftarrow t\rangle_{n''}.\ n''' \leq \langle$*the-singleton* $\rightarrow t\rangle_{n''} \wedge$ *eval the-singleton* t t' n''' $\gamma'))$
by *auto*
ultimately show *?thesis* **using** ‹*eval the-singleton* t t' n' γ› **by** *auto*
qed
ultimately show *?thesis* **using** *untilIA*[*of n the-singleton t n' t' γ γ'*] **by** *blast*
qed

lemma *untilE*[*elim*]:
fixes t *id* n γ' γ
assumes *eval the-singleton* t t' n $(\gamma'\ \mathfrak{U}_b\ \gamma)$
shows $\exists\, n'{\geq}n.$ *eval the-singleton* t t' n' $\gamma \wedge (\forall\, n''{\geq}n.\ n'' < n' \longrightarrow$ *eval the-singleton* t t' n'' $\gamma')$
proof −
have $\|$*the-singleton*$\|_{t\ n}$ **by** *simp*
with ‹*eval the-singleton* t t' n $(\gamma'\ \mathfrak{U}_b\ \gamma)$› **obtain** n' **where** $n'{\geq}\langle$*the-singleton* $\rightarrow t\rangle_n$ **and**
$(\exists\, i{\geq}n'.\ \|$*the-singleton*$\|_{t\ i}) \wedge$
$(\forall\, n''{\geq}\langle$*the-singleton* $\Leftarrow t\rangle_{n'}.\ n'' \leq \langle$*the-singleton* $\rightarrow t\rangle_{n'} \longrightarrow$ *eval the-singleton* t t' n'' $\gamma) \wedge$

$(\forall\, n''{\geq}\langle \mathit{the\text{-}singleton} \Leftarrow t\rangle_n.\ n'' < \langle \mathit{the\text{-}singleton} \Leftarrow t\rangle_{n'} \longrightarrow \mathit{eval\ the\text{-}singleton\ t\ t'\ n''\ \gamma'}) \vee$
$\neg\ (\exists\, i{\geq}n'.\ \|\mathit{the\text{-}singleton}\|_{t\ i}) \wedge$
$\mathit{eval\ the\text{-}singleton\ t\ t'\ n'\ \gamma} \wedge (\forall\, n''{\geq}\langle \mathit{the\text{-}singleton} \Leftarrow t\rangle_n.\ n'' < n' \longrightarrow \mathit{eval\ the\text{-}singleton\ t\ t'\ n''\ \gamma'})$
using *untilEA*[*of n the-singleton t t' γ' γ*] **by** *auto*
moreover have $\|\mathit{the\text{-}singleton}\|_{t\ n'}$ **by** *simp*
ultimately have
$(\forall\, n''{\geq}\langle \mathit{the\text{-}singleton} \Leftarrow t\rangle_{n'}.\ n'' \leq \langle \mathit{the\text{-}singleton} \rightarrow t\rangle_{n'} \longrightarrow \mathit{eval\ the\text{-}singleton\ t\ t'\ n''\ \gamma}) \wedge$
$(\forall\, n''{\geq}\langle \mathit{the\text{-}singleton} \Leftarrow t\rangle_n.\ n'' < \langle \mathit{the\text{-}singleton} \Leftarrow t\rangle_{n'} \longrightarrow \mathit{eval\ the\text{-}singleton\ t\ t'\ n''\ \gamma'})$ **by** *auto*
hence $\mathit{eval\ the\text{-}singleton\ t\ t'\ n'\ \gamma}$ **and** $(\forall\, n''{\geq}n.\ n'' < n' \longrightarrow \mathit{eval\ the\text{-}singleton\ t\ t'\ n''\ \gamma'})$ **by** *auto*
with ‹*eval the-singleton t t' n' γ*› ‹$n'{\geq}\langle \mathit{the\text{-}singleton} \rightarrow t\rangle_n$› **show** *?thesis* **by** *auto*
qed
end

end

2 A Theory of Publisher-Subscriber Architectures

In the following, we formalize the specification of the publisher subscriber pattern as described in [4].

theory *Publisher-Subscriber*
imports *Singleton*
begin

2.1 Subscriptions

datatype *'evt subscription = sub 'evt | unsub 'evt*

2.2 Publisher-Subscriber Architectures

locale *publisher-subscriber* =
pb: *singleton pbactive pbcmp* +
sb: *dynamic-component sbcmp sbactive*
for *pbactive* :: *'pid ⇒ cnf ⇒ bool* (‖-‖_ [0,110]60)
and *pbcmp* :: *'pid ⇒ cnf ⇒ 'PB* (σ_(-) [0,110]60)
and *sbactive* :: *'sid ⇒ cnf ⇒ bool* (‖-‖_ [0,110]60)
and *sbcmp* :: *'sid ⇒ cnf ⇒ 'SB* (σ_(-) [0,110]60) +
fixes *pbsb* :: *'PB ⇒ ('evt set) subscription set*
and *pbnt* :: *'PB ⇒ ('evt × 'msg)*
and *sbnt* :: *'SB ⇒ ('evt × 'msg) set*
and *sbsb* :: *'SB ⇒ ('evt set) subscription*
assumes *conn1*: $\bigwedge k\ pid.\ \|pid\|_k$
$\Longrightarrow pbsb\ (\sigma_{pid}(k)) = (\bigcup sid{\in}\{sid.\ \|sid\|_k\}.\ \{sbsb\ (\sigma_{sid}(k))\})$
and *conn2*: $\bigwedge t\ n\ n''\ sid\ pid\ E\ e\ m.$
$[\![t \in arch;\ \|pid\|_{t\ n};\ \|sid\|_{t\ n};\ sub\ E = sbsb\ (\sigma_{sid}(t\ n));\ n''{\geq}\ n;\ e \in E;$
$\nexists\, n'\ E'.\ n' \geq n \wedge n' \leq n'' \wedge \|sid\|_{t\ n'} \wedge$
$unsub\ E' = sbsb\ (\sigma_{sid}(t\ n')) \wedge e \in E';$
$(e,\ m) = pbnt\ (\sigma_{pid}(t\ n''));\ \|sid\|_{t\ n''}]\!]$
$\Longrightarrow pbnt\ (\sigma_{pid}(t\ n'')) \in sbnt\ (\sigma_{sid}(t\ n''))$
begin

2.2.1 Calculus Interpretation

pb.baIA: $[\![\exists\, i{\geq}n.\ \|c\|_{t\ i};\ \varphi\ (\sigma_c t\ (pb.nxtAct\ c\ t\ n))]\!] \Longrightarrow pb.eval\ c\ t\ t'\ n\ [\varphi]_b$

sb.baIA: $[\![\exists\, i{\geq}n.\ \|c\|_{t\ i};\ \varphi\ (\sigma_c t\ (sb.nxtAct\ c\ t\ n))]\!] \Longrightarrow sb.eval\ c\ t\ t'\ n\ [\varphi]_b$

2.2.2 Results from Singleton

abbreviation *the-pb* :: *'pid* **where**
the-pb ≡ *pb.the-singleton*

pb.ts-prop (*1*): $\|id\|_k \Longrightarrow id = the\text{-}pb$

pb.ts-prop (*2*): $\|the\text{-}pb\|_k$

2.2.3 Architectural Guarantees

The following theorem ensures that a subscriber indeed receives all messages associated with an event for which he is subscribed.

theorem *msgDelivery*:
  fixes $t\ n\ n''$ **and** *sid*::*'sid* **and** $E\ e\ m$
  assumes $t \in arch$
    and $\|sid\|_{t\ n}$
    and $sub\ E = sbsb\ (\sigma_{sid}(t\ n))$
    and $n'' \geq n$
    and $\nexists n'\ E'.\ n' \geq n \wedge n' \leq n'' \wedge \|sid\|_{t\ n'} \wedge unsub\ E' = sbsb(\sigma_{sid}(t\ n'))$
        $\wedge\ e \in E'$
    and $e \in E$
    and $(e,m) = pbnt\ (\sigma_{the\text{-}pb}(t\ n''))$
    and $\|sid\|_{t\ n''}$
  shows $(e,m) \in sbnt\ (\sigma_{sid}(t\ n''))$
  using *assms conn2 pb.ts-prop*(*2*) **by** *simp*

Since a publisher is actually a singleton, we can provide an alternative version of constraint *conn1*.

lemma *conn1A*:
  fixes *k*
  shows $pbsb\ (\sigma_{the\text{-}pb}(k)) = (\bigcup sid{\in}\{sid.\ \|sid\|_k\}.\ \{sbsb\ (\sigma_{sid}(k))\})$
  using *conn1*[*OF pb.ts-prop*(*2*)] **.**
end

end

3 A Theory of Blackboard Architectures

In the following, we formalize the specification of the blackboard pattern as described in [4].

theory *Blackboard*
imports *Publisher-Subscriber*
begin

3.1 Problems and Solutions

Blackboards work with problems and solutions for them.

typedecl *PROB*
consts *sb* :: (*PROB* × *PROB*) *set*
axiomatization where *sbWF*: *wf sb*

typedecl *SOL*
consts *solve*:: $PROB \Rightarrow SOL$

3.2 Blackboard Architectures

In the following, we describe the locale for the blackboard pattern.

locale *blackboard = publisher-subscriber bbactive bbcmp ksactive kscmp bbrp bbcs kscs ksrp*
for *bbactive* :: $'bid \Rightarrow cnf \Rightarrow bool$ $(\|\text{-}\|_{\text{-}}\ [0,110]60)$
and *bbcmp* :: $'bid \Rightarrow cnf \Rightarrow 'BB$ $(\sigma_{\text{-}}(\text{-})\ [0,110]60)$
and *ksactive* :: $'kid \Rightarrow cnf \Rightarrow bool$ $(\|\text{-}\|_{\text{-}}\ [0,110]60)$
and *kscmp* :: $'kid \Rightarrow cnf \Rightarrow 'KS$ $(\sigma_{\text{-}}(\text{-})\ [0,110]60)$
and *bbrp* :: $'BB \Rightarrow (PROB\ set)\ subscription\ set$
and *bbcs* :: $'BB \Rightarrow (PROB \times SOL)$
and *kscs* :: $'KS \Rightarrow (PROB \times SOL)\ set$
and *ksrp* :: $'KS \Rightarrow (PROB\ set)\ subscription$ +
fixes *bbns* :: $'BB \Rightarrow (PROB \times SOL)\ set$
and *ksns* :: $'KS \Rightarrow (PROB \times SOL)$
and *bbop* :: $'BB \Rightarrow PROB$
and *ksop* :: $'KS \Rightarrow PROB\ set$
and *prob* :: $'kid \Rightarrow PROB$
assumes
ks1: $\forall p.\ \exists ks.\ p{=}prob\ ks$ — Component Parameter
— Assertions about component behavior.
and *bhvbb1*: $\bigwedge t\ t'\ bId\ p\ s.\ [\![t \in arch]\!] \Longrightarrow pb.eval\ bId\ t\ t'\ 0$
$(\Box_b\ ([\lambda bb.\ (p,s){\in}bbns\ bb]_b$
$\longrightarrow^b\ (\Diamond_b\ [\lambda bb.\ (p,s) = bbcs\ bb]_b)))$
and *bhvbb2*: $\bigwedge t\ t'\ bId\ P\ q.\ [\![t{\in}arch]\!] \Longrightarrow pb.eval\ bId\ t\ t'\ 0$
$(\Box_b\ ([\lambda bb.\ sub\ P \in bbrp\ bb \wedge q \in P]_b \longrightarrow^b$
$(\Diamond_b\ [\lambda bb.\ q = bbop\ bb]_b)))$
and *bhvbb3*: $\bigwedge t\ t'\ bId\ p\ .\ [\![t{\in}arch]\!] \Longrightarrow pb.eval\ bId\ t\ t'\ 0$
$(\Box_b\ ([\lambda bb.\ p = bbop(bb)]_b \longrightarrow^b$
$([\lambda bb.\ p{=}bbop(bb)]_b\ \mathfrak{W}_b\ [\lambda bb.\ (p,solve(p)) = bbcs(bb)]_b)))$
and *bhvks1*: $\bigwedge t\ t'\ kId\ p\ P.\ [\![t{\in}arch;\ p = prob\ kId]\!] \Longrightarrow sb.eval\ kId\ t\ t'\ 0$
$(\Box_b\ ([\lambda ks.\ sub\ P = ksrp\ ks]_b \wedge^b$
$(\forall_b\ q.\ ((sb.pred\ (q{\in}P)) \longrightarrow^b\ (\Diamond_b\ ([\lambda ks.\ (q,solve(q)) \in kscs\ ks]_b))))$
$\longrightarrow^b\ (\Diamond_b\ [\lambda ks.\ (p,\ solve\ p) = ksns\ ks]_b)))$
and *bhvks2*: $\bigwedge t\ t'\ kId\ p\ P\ q.\ [\![t \in arch;p = prob\ kId]\!] \Longrightarrow sb.eval\ kId\ t\ t'\ 0$
$(\Box_b\ [\lambda ks.\ sub\ P = ksrp\ ks \wedge q \in P \longrightarrow (q,p) \in sb]_b)$
and *bhvks3*: $\bigwedge t\ t'\ kId\ p.\ [\![t{\in}arch;p = prob\ kId]\!] \Longrightarrow sb.eval\ kId\ t\ t'\ 0$
$(\Box_b\ ([\lambda ks.\ p{\in}ksop\ ks]_b \longrightarrow^b\ (\Diamond_b\ [\lambda ks.\ (\exists P.\ sub\ P = ksrp\ ks)]_b)))$
and *bhvks4*: $\bigwedge t\ t'\ kId\ p\ P.\ [\![t{\in}arch;\ p{\in}P]\!] \Longrightarrow sb.eval\ kId\ t\ t'\ 0$
$(\Box_b\ ([\lambda ks.\ sub\ P = ksrp\ ks]_b \longrightarrow^b$
$((\neg^b\ (\exists_b\ P'.\ (sb.pred\ (p{\in}P') \wedge^b\ [\lambda ks.\ unsub\ P' = ksrp\ ks]_b)))\ \mathfrak{W}_b$
$[\lambda ks.\ (p,solve\ p) \in kscs\ ks]_b)))$

— Assertions about component activation.
and *actks*:
$\bigwedge t\ n\ kid\ p.\ [\![t \in arch;\ \|kid\|_{t\ n};\ p{=}prob\ kid;\ p{\in}ksop\ (\sigma_{kid}(t\ n))]\!]$
$\Longrightarrow (\exists n'{\geq}n.\ \|kid\|_{t\ n'} \wedge (p,\ solve\ p) = ksns\ (\sigma_{kid}(t\ n')) \wedge$
$(\forall n''{\geq}n.\ n''{<}n' \longrightarrow \|kid\|_{t\ n''}))$
$\vee\ (\forall n'{\geq}n.\ (\|kid\|_{t\ n'} \wedge (\neg(p,\ solve\ p) = ksns\ (\sigma_{kid}(t\ n')))))$

— Assertions about connections.
and *conn1*: $\bigwedge k\ bid.\ \|bid\|_k$
$\Longrightarrow bbns\ (\sigma_{bid}(k)) = (\bigcup kid{\in}\{kid.\ \|kid\|_k\}.\ \{ksns\ (\sigma_{kid}(k))\})$

and *conn2*: $\bigwedge k\ kid.\ \|kid\|_k$
$\Longrightarrow ksop\ (\sigma_{kid}(k)) = (\bigcup bid{\in}\{bid.\ \|bid\|_k\}.\ \{bbop\ (\sigma_{bid}(k))\})$
begin
notation *sb.lNAct* $(\langle\text{-} \Leftarrow \text{-}\rangle_\text{-})$
notation *sb.nxtAct* $(\langle\text{-} \rightarrow \text{-}\rangle_\text{-})$
notation *pb.lNAct* $(\langle\text{-} \Leftarrow \text{-}\rangle_\text{-})$
notation *pb.nxtAct* $(\langle\text{-} \rightarrow \text{-}\rangle_\text{-})$

3.2.1 Calculus Interpretation

pb.baIA: $[\![\exists\, i{\geq}n.\ \|c\|_{t\ i};\ \varphi\ (\sigma_c t\ \langle c \rightarrow t\rangle_n)]\!] \Longrightarrow pb.eval\ c\ t\ t'\ n\ [\varphi]_b$

sb.baIA: $[\![\exists\, i{\geq}n.\ \|c\|_{t\ i};\ \varphi\ (\sigma_c t\ \langle c \rightarrow t\rangle_n)]\!] \Longrightarrow sb.eval\ c\ t\ t'\ n\ [\varphi]_b$

3.2.2 Results from Singleton

abbreviation *the-bb* $\equiv$ *the-pb*

pb.ts-prop (1): $\|id\|_k \Longrightarrow id = the\text{-}bb$

pb.ts-prop (2): $\|the\text{-}bb\|_k$

3.2.3 Results from Publisher Subscriber

msgDelivery: $[\![t \in arch;\ \|sid\|_{t\ n};\ sub\ E = ksrp\ (\sigma_{sid} t\ n);\ n \leq n'';\ \nexists\, n'\ E'.\ n \leq n' \wedge n' \leq n'' \wedge \|sid\|_{t\ n'} \wedge unsub\ E' = ksrp\ (\sigma_{sid} t\ n') \wedge e \in E';\ e \in E;\ (e,\ m) = bbcs\ (\sigma_{the\text{-}bb} t\ n'');\ \|sid\|_{t\ n''}]\!]$
$\Longrightarrow (e,\ m) \in kscs\ (\sigma_{sid} t\ n'')$

lemma *conn2-bb*:
fixes k **and** *kid*::*'kid*
assumes $\|kid\|_k$
shows $bbop\ (\sigma_{the\text{-}bb}(k)){\in}ksop\ (\sigma_{kid}(k))$
proof −
from *assms* **have** $ksop\ (\sigma_{kid}(k)) = (\bigcup bid{\in}\{bid.\ \|bid\|_k\}.\ \{bbop\ (\sigma_{bid}(k))\})$ **using** *conn2* **by** *simp*
moreover have $(\bigcup bid.\{bid.\ \|bid\|_k\}){=}\{the\text{-}bb\}$ **using** *pb.ts-prop(1)* **by** *auto*
hence $(\bigcup bid{\in}\{bid.\ \|bid\|_k\}.\ \{bbop\ (\sigma_{bid}(k))\}) = \{bbop\ (\sigma_{the\text{-}bb}(k))\}$ **by** *auto*
ultimately show *?thesis* **by** *simp*
qed

3.2.4 Knowledge Sources

In the following we introduce an abbreviation for knowledge sources which are able to solve a specific problem.

definition *sKs*:: *PROB* $\Rightarrow$ *'kid* **where**
sKs p $\equiv$ (*SOME kid. p = prob kid*)

lemma *sks-prob*:
$p = prob\ (sKs\ p)$
using *sKs-def someI-ex*[*of* $\lambda kid.\ p = prob\ kid$] *ks1* **by** *auto*

3.2.5 Architectural Guarantees

The following theorem verifies that a problem is eventually solved by the pattern even if no knowledge source exist which can solve the problem on its own. It assumes, however, that for

every open sub problem, a corresponding knowledge source able to solve the problem will be eventually activated.

lemma *pSolved-Ind*:
fixes t **and** t'::*nat* $\Rightarrow$'*BB* **and** p **and** t''::*nat* $\Rightarrow$'*KS*
assumes $t \in arch$ **and**
$\forall\, n.\ (\exists\, n' \geq n.\ \|sKs\ (bbop(\sigma_{the\text{-}bb}(t\ n)))\|_{t\ n'})$
shows
$\forall\, n.\ (\exists\, P.\ sub\ P \in bbrp(\sigma_{the\text{-}bb}(t\ n)) \wedge p \in P) \longrightarrow$
$(\exists\, m \geq n.\ (p, solve(p)) = bbcs\ (\sigma_{the\text{-}bb}(t\ m)))$
— The proof is by well-founded induction over the subproblem relation *sb*
proof (*rule wf-induct*[**where** r=*sb*])
— We first show that the subproblem relation is indeed well-founded ...
show *wf sb* **by** (*simp add*: *sbWF*)
next
— ... then we show that a problem p is indeed solved
— if all its sub-problems p' are eventually solved
fix p **assume** *indH*: $\forall\, p'.\ (p', p) \in sb \longrightarrow (\forall\, n.\ (\exists\, P.\ sub\ P \in bbrp\ (\sigma_{the\text{-}bb}(t\ n)) \wedge p' \in P)$
$\longrightarrow (\exists\, m \geq n.\ (p', solve(p')) = bbcs\ (\sigma_{the\text{-}bb}(t\ m))))$
show $\forall\, n.\ (\exists\, P.\ sub\ P \in bbrp\ (\sigma_{the\text{-}bb}(t\ n)) \wedge p \in P)$
$\longrightarrow (\exists\, m \geq n.\ (p, solve(p)) = bbcs\ (\sigma_{the\text{-}bb}(t\ m)))$
proof
fix n_0 **show** $(\exists\, P.\ sub\ P \in bbrp\ (\sigma_{the\text{-}bb}(t\ n_0)) \wedge p \in P) \longrightarrow$
$(\exists\, m \geq n_0.\ (p, solve(p)) = bbcs\ (\sigma_{the\text{-}bb}(t\ m)))$
proof
assume $\exists\, P.\ sub\ P \in bbrp\ (\sigma_{the\text{-}bb}(t\ n_0)) \wedge p \in P$
moreover have $(\exists\, P.\ sub\ P \in bbrp\ (\sigma_{the\text{-}bb}(t\ n_0)) \wedge p \in P) \longrightarrow (\exists\, n' \geq n_0.\ p = bbop(\sigma_{the\text{-}bb}(t\ n')))$
proof
assume $\exists\, P.\ sub\ P \in bbrp\ (\sigma_{the\text{-}bb}(t\ n_0)) \wedge p \in P$
then obtain P **where** $sub\ P \in bbrp\ (\sigma_{the\text{-}bb}(t\ n_0))$ **and** $p \in P$ **by** *auto*
hence $pb.eval\ the\text{-}bb\ t\ t'\ n_0\ [\lambda bb.\ sub\ P \in bbrp\ bb \wedge p \in P]_b$ **using** *pb.baI* **by** *simp*
moreover from *pb.globE*[*OF bhvbb2*] **have**
$pb.eval\ the\text{-}bb\ t\ t'\ n_0\ ([\lambda bb.\ sub\ P \in bbrp\ bb \wedge p \in P]_b \longrightarrow^b \Diamond_b\ [\lambda bb.\ p = bbop\ bb]_b)$
using ‹$t \in arch$› **by** *simp*
ultimately have $pb.eval\ the\text{-}bb\ t\ t'\ n_0\ (\Diamond_b\ [\lambda bb.\ p = bbop\ bb]_b)$ **using** *pb.impE* **by** *blast*
then obtain n' **where** $n' \geq n_0$ **and** $pb.eval\ the\text{-}bb\ t\ t'\ n'\ [\lambda bb.\ p = bbop\ bb]_b$
using *pb.evtE* **by** *blast*
hence $p = bbop(\sigma_{the\text{-}bb}(t\ n'))$ **using** *pb.baE* **by** *auto*
with ‹$n' \geq n_0$› **show** $\exists\, n' \geq n_0.\ p = bbop(\sigma_{the\text{-}bb}(t\ n'))$ **by** *auto*
qed
ultimately obtain n **where** $n \geq n_0$ **and** $p = bbop(\sigma_{the\text{-}bb}(t\ n))$ **by** *auto*

— Problem p is provided at the output of the blackboard until it is solved
— or forever...
from *pb.globE*[*OF bhvbb3*] **have**
$pb.eval\ the\text{-}bb\ t\ t'\ n\ ([\lambda\ bb.\ p = bbop(bb)]_b \longrightarrow^b$
$([\lambda\ bb.\ p = bbop(bb)]_b\ \mathfrak{W}_b\ [\lambda bb.\ (p, solve(p)) = bbcs(bb)]_b))$
using ‹$t \in arch$› **by** *auto*
moreover from ‹$p = bbop\ (\sigma_{the\text{-}bb}(t\ n))$› **have**
$pb.eval\ the\text{-}bb\ t\ t'\ n\ [\lambda\ bb.\ p = bbop\ bb]_b$
using ‹$t \in arch$› *pb.baI* **by** *simp*
ultimately have $pb.eval\ the\text{-}bb\ t\ t'\ n$
$([\lambda\ bb.\ p = bbop(bb)]_b\ \mathfrak{W}_b\ [\lambda\ bb.\ (p, solve(p)) = bbcs(bb)]_b)$
using *pb.impE* **by** *blast*
hence $pb.eval\ the\text{-}bb\ t\ t'\ n\ (([\lambda\ bb.\ p = bbop\ bb]_b\ \mathfrak{U}_b$

$[\lambda\ bb.\ (p,solve(p)) = bbcs\ bb]_b) \vee^b (\Box_b\ [\lambda\ bb.\ p{=}bbop\ bb]_b))$
using *pb.wuntil-def* **by** *simp*
hence *pb.eval the-bb t t′ n*
$([\lambda bb.\ p{=}bbop\ bb]_b\ \mathfrak{U}_b\ [\lambda bb.\ (p,solve(p)) = bbcs\ bb]_b) \vee$
$(pb.eval\ the\text{-}bb\ t\ t'\ n\ (\Box_b\ [\lambda\ bb.\ p{=}bbop\ bb]_b))$
using *pb.disjE* **by** *simp*
thus $\exists\, m{\geq}n_0.\ (p,solve\ p) = bbcs(\sigma_{the\text{-}bb}(t\ m))$
— We need to consider both cases, the case in which the problem is eventually
— solved and the case in which the problem is always provided as an output
proof
— First we consider the case in which the problem is eventually solved:
assume *pb.eval the-bb t t′ n*
$([\lambda bb.\ p{=}bbop\ bb]_b\ \mathfrak{U}_b\ [\lambda bb.\ (p,solve(p)) = bbcs\ bb]_b)$
hence $\exists\, i{\geq}n.$ *(pb.eval the-bb t t′ i*
$[\lambda bb.\ (p,solve(p)) = bbcs\ bb]_b \wedge$
$(\forall\, k{\geq}n.\ k{<}i \longrightarrow pb.eval\ the\text{-}bb\ t\ t'\ k\ [\lambda bb.\ p = bbop\ bb]_b))$
using ‹*t∈arch*› *pb.untilE* **by** *simp*
then obtain *i* **where** *i≥n* **and**
pb.eval the-bb t t′ i $[\lambda bb.\ (p,solve(p)) = bbcs\ bb]_b$ **by** *auto*
hence $(p,solve(p)) = bbcs(\sigma_{the\text{-}bb}(t\ i))$
using ‹*t∈arch*› *pb.baEA* **by** *auto*
moreover from ‹*i≥n*› ‹$n{\geq}n_0$› **have** $i{\geq}n_0$ **by** *simp*
ultimately show *?thesis* **by** *auto*
next
— Now we consider the case in which p is always provided at the output
— of the blackboard:
assume *pb.eval the-bb t t′ n*
$(\Box_b\ [\lambda bb.\ p{=}bbop\ bb]_b)$
hence $\forall\, n'{\geq}n.$ *(pb.eval the-bb t t′ n′* $[\lambda bb.\ p = bbop\ bb]_b)$
using ‹*t∈arch*› *pb.globE* **by** *auto*
hence *outp*: $\forall\, n'{\geq}n.\ (p = bbop\ (\sigma_{the\text{-}bb}(t\ n')))$
using ‹*t∈arch*› *pb.baE* **by** *blast*

— thus, by assumption there exists a KS which is able to solve p and which
— is active at n'...
with *assms(2)* **have** $\exists\, n'{\geq}n.\ \|sKs\ p\|_{t\ n'}$ **by** *auto*
then obtain n_k **where** $n_k{\geq}n$ **and** $\|sKs\ p\|_{t\ n_k}$ **by** *auto*
— ... and get the problem as its input.
moreover from ‹$n_k{\geq}n$› **have** $p = bbop\ (\sigma_{the\text{-}bb}(t\ n_k))$
using *outp* **by** *simp*
ultimately have $p{\in}ksop(\sigma_{sKs\ p}(t\ n_k))$ **using** *conn2-bb[of sKs p t* n_k*]* **by** *simp*

— thus the ks will either solve the problem or not solve it and
— be activated forever
hence $(\exists\, n'{\geq}n_k.\ \|sKs\ p\|_{t\ n'} \wedge$
$(p,\ solve\ p) = ksns\ (\sigma_{sKs\ p}(t\ n')) \wedge$
$(\forall\, n''{\geq}n_k.\ n''{<}n' \longrightarrow \|sKs\ p\|_{t\ n''})) \vee$
$(\forall\, n'{\geq}n_k.\ (\|sKs\ p\|_{t\ n'} \wedge$
$(\neg(p,\ solve\ p) = ksns\ (\sigma_{sKs\ p}(t\ n')))))$
using ‹$\|sKs\ p\|_{t\ n_k}$› *actks[of t sKs p]* ‹*t∈arch*› *sks-prob* **by** *simp*
thus *?thesis*
proof
— if the ks solves it
assume $\exists\, n'{\geq}n_k.\ \|sKs\ p\|_{t\ n'} \wedge (p,\ solve\ p) = ksns\ (\sigma_{sKs\ p} t\ n')$
$\wedge\ (\forall\, n''{\geq}n_k.\ n'' < n' \longrightarrow \|sKs\ p\|_{t\ n''})$

— it is forwarded to the blackboard
then obtain n_s **where** $n_s{\geq}n_k$ **and** $\|sKs\ p\|_{t\ n_s}$
and $(p,\ solve\ p) = ksns\ (\sigma_{sKs\ p} t\ n_s)$ **by** *auto*
moreover have $\langle sKs\ p \rightarrow t\rangle_{n_s} = n_s$
by (*simp add*: ‹$\|sKs\ p\|_{t\ n_s}$› *sb.nxtAct-active*)
ultimately have
$(p,solve(p)) \in bbns\ (\sigma_{the\text{-}bb}(t\ (\langle sKs\ p \rightarrow t\rangle_{n_s})))$
using *conn1*[*OF pb.ts-prop*(*2*)] ‹$\|sKs\ p\|_{t\ n_s}$› **by** *auto*

— finally, the blackboard will forward the solution which finishes the proof.
with *bhvbb1* **have** *pb.eval the-bb t t'* $(\langle sKs\ p \rightarrow t\rangle_{n_s})$
$(\Diamond_b\ [\lambda bb.\ (p,\ solve\ p) = bbcs\ bb]_b)$
using ‹*t*∈*arch*› *pb.globE pb.impE*[*of the-bb t t'*] **by** *blast*
then obtain n_f **where** $n_f{\geq}\langle sKs\ p \rightarrow t\rangle_{n_s}$ **and**
pb.eval the-bb t t' n_f $[\lambda bb.\ (p,\ solve\ p) = bbcs\ bb]_b$
using ‹*t*∈*arch*› *pb.evtE*[*of t t'* $\langle sKs\ p \rightarrow t\rangle_{n_s}$] **by** *auto*
hence $(p,\ solve\ p) = bbcs\ (\sigma_{the\text{-}bb}(t\ n_f))$
using ‹$t \in arch$› *pb.baEA* **by** *auto*
moreover have $n_f{\geq}n_0$
proof −
from ‹$\|sKs\ p\|_{t\ n_k}$› **have** $\langle sKs\ p \rightarrow t\rangle_{n_k}{\geq}n_k$
using *sb.nxtActI* **by** *blast*
with ‹$\langle sKs\ p \rightarrow t\rangle_{n_s} = n_s$› **show** *?thesis*
using ‹$n_f{\geq}\langle sKs\ p \rightarrow t\rangle_{n_s}$› ‹$n_s{\geq}n_k$› ‹$n_k{\geq}n$› ‹$n{\geq}n_0$› **by** *arith*
qed
ultimately show *?thesis* **by** *auto*
next
— otherwise, we derive a contradiction
assume *case-ass*: $\forall\, n'{\geq}n_k.\ \|sKs\ p\|_{t\ n'} \wedge \neg(p,\ solve\ p) = ksns\ (\sigma_{sKs\ p} t\ n')$

— first, the KS will eventually register for the subproblems P it requires to solve p...
from ‹$\|sKs\ p\|_{t\ n_k}$› **have** $\exists\, i{\geq}0.\ \|sKs\ p\|_{t\ i}$ **by** *auto*
moreover have $\langle sKs\ p \Leftarrow t\rangle_0 \leq n_k$ **by** *simp*
ultimately have *sb.eval* $(sKs\ p)\ t\ t''\ n_k$
$([\lambda ks.\ p{\in}ksop\ ks]_b \longrightarrow^b (\Diamond_b\ [\lambda ks.\ \exists\, P.\ sub\ P = ksrp\ ks]_b))$
using *sb.globEA*[*OF - bhvks3*[*of t p sKs p t''*]] ‹*t*∈*arch*› *sks-prob* **by** *simp*
moreover have *sb.eval* $(sKs\ p)\ t\ t''\ n_k\ [\lambda ks.\ p \in ksop\ ks]_b$
proof −
from ‹$\|sKs\ p\|_{t\ n_k}$› **have** $\exists\, n'{\geq}n_k.\ \|sKs\ p\|_{t\ n'}$ **by** *auto*
moreover have $p \in ksop\ (\sigma_{sKs\ p}(t\ (\langle sKs\ p \rightarrow t\rangle_{n_k})))$
proof −
from ‹$\|sKs\ p\|_{t\ n_k}$› **have** $\langle sKs\ p \rightarrow t\rangle_{n_k}{=}n_k$
using *sb.nxtAct-active* **by** *blast*
with ‹$p{\in}ksop(\sigma_{sKs\ p}(t\ n_k))$› **show** *?thesis* **by** *simp*
qed
ultimately show *?thesis* **using** *sb.baIA*[*of* n_k *sKs p t*] **by** *blast*
qed
ultimately have *sb.eval* $(sKs\ p)\ t\ t''\ n_k$ $(\Diamond_b\ [\lambda ks.\ \exists\, P.\ sub\ P = ksrp\ ks]_b)$
using *sb.impE* **by** *blast*
then obtain n_r **where** $n_r{\geq}\langle sKs\ p \rightarrow t\rangle_{n_k}$ **and**
$\exists\, i{\geq}n_r.\ \|sKs\ p\|_{t\ i} \wedge$
$(\forall\, n''{\geq}\langle sKs\ p \Leftarrow t\rangle_{n_r}.\ n'' \leq \langle sKs\ p \rightarrow t\rangle_{n_r}$
$\longrightarrow$ *sb.eval* $(sKs\ p)\ t\ t''\ n''\ [\lambda ks.\ \exists\, P.\ sub\ P = ksrp\ ks]_b) \vee$
$\neg\ (\exists\, i{\geq}n_r.\ \|sKs\ p\|_{t\ i}) \wedge$
sb.eval $(sKs\ p)\ t\ t''\ n_r\ [\lambda ks.\ \exists\, P.\ sub\ P = ksrp\ ks]_b$

using ‹$\|sKs\ p\|_{t\ n_k}$› *sb.evtEA*[*of* n_k *sKs p t*] **by** *blast*
moreover from *case-ass* **have** $\langle sKs\ p \to t\rangle_{n_k} \geq n_k$ **using** *sb.nxtActI* **by** *blast*
with ‹$n_r \geq \langle sKs\ p \to t\rangle_{n_k}$› **have** $n_r \geq n_k$ **by** *arith*
hence $\exists\, i \geq n_r.\ \|sKs\ p\|_{t\ i}$ **using** *case-ass* **by** *auto*
hence $n_r \leq \langle sKs\ p \to t\rangle_{n_r}$ **using** *sb.nxtActLe* **by** *simp*
moreover have $n_r \geq \langle sKs\ p \Leftarrow t\rangle_{n_r}$ **by** *simp*
ultimately have
sb.eval (*sKs p*) *t t''* n_r $[\lambda ks.\ \exists\, P.\ sub\ P = ksrp\ ks]_b$ **by** *blast*
with ‹$\exists\, i \geq n_r.\ \|sKs\ p\|_{t\ i}$› **obtain** *P* **where**
sub P = *ksrp* ($\sigma_{sKs\ p}$(*t* ($\langle sKs\ p \to t\rangle_{n_r}$)))
using *sb.baEA* **by** *blast*
hence *sb.eval* (*sKs p*) *t t''* n_r $[\lambda ks.\ sub\ P = ksrp\ ks]_b$
using ‹$\exists\, i \geq n_r.\ \|sKs\ p\|_{t\ i}$› *sb.baIA sks-prob* **by** *blast*

— the knowledgesource will eventually get a solution for each required subproblem:
moreover have *sb.eval* (*sKs p*) *t t''* n_r ($\forall_b\ p'$. (*sb.pred* ($p' \in P$) $\longrightarrow^b$
($\Diamond_b$ $[\lambda ks.\ (p', solve\ p') \in kscs\ ks]_b$)))
proof −
have $\forall\, p'$. *sb.eval* (*sKs p*) *t t''* n_r (*sb.pred* ($p' \in P$) $\longrightarrow^b$
($\Diamond_b$ $[\lambda ks.\ (p', solve\ p') \in kscs\ ks]_b$))
proof
— by induction hypothesis, the blackboard will eventually provide solutions for subproblems
fix p'
have *sb.eval* (*sKs p*) *t t''* n_r (*sb.pred* ($p' \in P$)) $\longrightarrow$
(*sb.eval* (*sKs p*) *t t''* n_r
($\Diamond_b$ $[\lambda ks.\ (p', solve\ p') \in kscs\ ks]_b$))
proof
assume *sb.eval* (*sKs p*) *t t''* n_r (*sb.pred* ($p' \in P$))
hence $p' \in P$ **using** *sb.predE* **by** *blast*
thus (*sb.eval* (*sKs p*) *t t''* n_r ($\Diamond_b$ $[\lambda ks.\ (p', solve\ p') \in kscs\ ks]_b$))
proof −
have $\langle sKs\ p \Leftarrow t\rangle_0 \leq n_r$ **by** *simp*
moreover from ‹$\|sKs\ p\|_{t\ n_k}$› **have** $\exists\, i \geq 0.\ \|sKs\ p\|_{t\ i}$ **by** *auto*
ultimately have *sb.eval* (*sKs p*) *t t''* n_r ($[\lambda ks.\ sub\ P = ksrp\ ks]_b$
$\longrightarrow^b$ (($\neg^b$ ($\exists_b\ P'$. (*sb.pred* ($p' \in P'$) $\wedge^b$ $[\lambda ks.\ unsub\ P' = ksrp\ ks]_b$))) $\mathfrak{W}_b$
$[\lambda ks.\ (p', solve\ p') \in kscs\ ks]_b$))
using *sb.globEA*[*OF - bhvks4*[*of t p' P sKs p t''*]]
‹*t*∈*arch*› ‹$\|sKs\ p\|_{t\ n_k}$› ‹$p' \in P$› **by** *simp*
with ‹*sb.eval* (*sKs p*) *t t''* n_r $[\lambda ks.\ sub\ P = ksrp\ ks]_b$› **have**
sb.eval (*sKs p*) *t t''* n_r (($\neg^b$ ($\exists_b\ P'$. (*sb.pred* ($p' \in P'$) $\wedge^b$
$[\lambda ks.\ unsub\ P' = ksrp\ ks]_b$))) $\mathfrak{W}_b$ $[\lambda ks.\ (p', solve\ p') \in kscs\ ks]_b$)
using *sb.impE*[*of* (*sKs p*) *t t''* n_r $[\lambda ks.\ sub\ P = ksrp\ ks]_b$] **by** *blast*
hence *sb.eval* (*sKs p*) *t t''* n_r (($\neg^b$ ($\exists_b\ P'$. (*sb.pred* ($p' \in P'$) $\wedge^b$
$[\lambda ks.\ unsub\ P' = ksrp\ ks]_b$))) $\mathfrak{U}_b$ $[\lambda ks.\ (p', solve\ p') \in kscs\ ks]_b$) $\vee$
sb.eval (*sKs p*) *t t''* n_r ($\Box_b$ ($\neg^b$ ($\exists_b\ P'$. (*sb.pred* ($p' \in P'$) $\wedge^b$
$[\lambda ks.\ unsub\ P' = ksrp\ ks]_b$)))) **using** *sb.wuntil-def* **by** *auto*
thus (*sb.eval* (*sKs p*) *t t''* n_r ($\Diamond_b$ $[\lambda ks.\ (p', solve\ p') \in kscs\ ks]_b$))
proof
let *?γ'*=$\neg^b$ ($\exists_b\ P'$. (*sb.pred* ($p' \in P'$) $\wedge^b$ ($[\lambda ks.\ unsub\ P' = ksrp\ ks]_b$)))
let *?γ*=$[\lambda ks.\ (p', solve\ p') \in kscs\ ks]_b$
assume *sb.eval* (*sKs p*) *t t''* n_r (*?γ'* $\mathfrak{U}_b$ *?γ*)
with ‹$\exists\, i \geq n_r.\ \|sKs\ p\|_{t\ i}$› **obtain** n' **where** $n' \geq \langle sKs\ p \to t\rangle_{n_r}$ **and**
lass: ($\exists\, i \geq n'.\ \|sKs\ p\|_{t\ i}$) $\wedge$ ($\forall\, n'' \geq \langle sKs\ p \Leftarrow t\rangle_{n'}.\ n'' \leq \langle sKs\ p \to t\rangle_{n'}$
$\longrightarrow$ *sb.eval* (*sKs p*) *t t''* *n''* *?γ*) $\wedge$
($\forall\, n'' \geq \langle sKs\ p \Leftarrow t\rangle_{n_r}.\ n'' < \langle sKs\ p \Leftarrow t\rangle_{n'}$

$\longrightarrow$ *sb.eval* (*sKs p*) *t t''* *n''* *?γ'*) ∨
¬ (∃ *i*≥*n'*. ‖*sKs p*‖$_{t\ i}$) ∧ *sb.eval* (*sKs p*) *t t''* *n'* *?γ* ∧
(∀ *n''*≥⟨*sKs p* ⇐ *t*⟩$_{n_r}$. *n''* < *n'* $\longrightarrow$ *sb.eval* (*sKs p*) *t t''* *n''* *?γ'*)
using *sb.untilEA*[*of* n_r *sKs p t t''*] ‹∃ *i*≥n_r. ‖*sKs p*‖$_{t\ i}$› **by** *blast*
thus *?thesis*
proof *cases*
assume ∃ *i*≥*n'*. ‖*sKs p*‖$_{t\ i}$
with *lass* **have** ∀ *n''*≥⟨*sKs p* ⇐ *t*⟩$_{n'}$. *n''* ≤ ⟨*sKs p* → *t*⟩$_{n'}$
$\longrightarrow$ *sb.eval* (*sKs p*) *t t''* *n''* *?γ* **by** *auto*
moreover have *n'*≥⟨*sKs p* ⇐ *t*⟩$_{n'}$ **by** *simp*
moreover have *n'* ≤ ⟨*sKs p* → *t*⟩$_{n'}$
using ‹∃ *i*≥*n'*. ‖*sKs p*‖$_{t\ i}$› *sb.nxtActLe* **by** *simp*
ultimately have *sb.eval* (*sKs p*) *t t''* *n'* *?γ* **by** *simp*
moreover have ⟨*sKs p* ⇐ *t*⟩$_{n_r}$ ≤ *n'* **using** ‹n_r ≤ ⟨*sKs p* → *t*⟩$_{n_r}$›
‹⟨*sKs p* ⇐ *t*⟩$_{n_r}$ ≤ n_r› ‹⟨*sKs p* → *t*⟩$_{n_r}$ ≤ *n'*› **by** *linarith*
ultimately show *?thesis* **using** ‹∃ *i*≥n_r. ‖*sKs p*‖$_{t\ i}$› ‹∃ *i*≥*n'*. ‖*sKs p*‖$_{t\ i}$›
‹*n'*≥⟨*sKs p* ⇐ *t*⟩$_{n'}$› ‹*n'* ≤ ⟨*sKs p* → *t*⟩$_{n'}$›
sb.evtIA[*of* n_r *sKs p t n' t''* *?γ*] **by** *blast*
next
assume ¬ (∃ *i*≥*n'*. ‖*sKs p*‖$_{t\ i}$)
with *lass* **have** *sb.eval* (*sKs p*) *t t''* *n'* *?γ* ∧
(∀ *n''*≥⟨*sKs p* ⇐ *t*⟩$_{n_r}$. *n''* < *n'* $\longrightarrow$ *sb.eval* (*sKs p*) *t t''* *n''* *?γ'*) **by** *auto*
moreover have ⟨*sKs p* ⇐ *t*⟩$_{n_r}$ ≤ *n'*
using ‹n_r ≤ ⟨*sKs p* → *t*⟩$_{n_r}$› ‹⟨*sKs p* ⇐ *t*⟩$_{n_r}$ ≤ n_r›
‹⟨*sKs p* → *t*⟩$_{n_r}$ ≤ *n'*› **by** *linarith*
ultimately show *?thesis* **using** ‹∃ *i*≥n_r. ‖*sKs p*‖$_{t\ i}$› ‹¬ (∃ *i*≥*n'*. ‖*sKs p*‖$_{t\ i}$)›
sb.evtIA[*of* n_r *sKs p t n' t''* *?γ*] **by** *blast*
qed
next
assume *cass*: *sb.eval* (*sKs p*) *t t''* n_r
(□$_b$ (¬^{b} (∃$_b$ *P'*. (*sb.pred* (*p'*∈*P'*) ∧^{b} [λ*ks*. *unsub P'* = *ksrp ks*]$_b$))))

have *sub P* = *ksrp* ($\sigma_{sKs\ p}$(*t* (⟨*sKs p* → *t*⟩$_{n_r}$))) ∧
p' ∈ *P* $\longrightarrow$ (*p'*, *p*) ∈ *sb*
proof −
have ∃ *i*≥*0*. ‖*sKs p*‖$_{t\ i}$ **using** ‹∃ *i*≥*0*. ‖*sKs p*‖$_{t\ i}$› **by** *auto*
moreover have ⟨*sKs p* ⇐ *t*⟩$_0$ ≤ (⟨*sKs p* → *t*⟩$_{n_r}$) **by** *simp*
ultimately have *sb.eval* (*sKs p*) *t t''* (⟨*sKs p* → *t*⟩$_{n_r}$)
[λ*ks*. *sub P* = *ksrp ks* ∧ *p'* ∈ *P* $\longrightarrow$ (*p'*, *p*) ∈ *sb*]$_b$
using *sb.globEA*[*OF - bhvks2*[*of t p sKs p t''* *P*]] ‹*t* ∈ *arch*› *sks-prob* **by** *blast*
moreover from ‹∃ *i*≥n_r. ‖*sKs p*‖$_{t\ i}$› **have**
‖*sKs p*‖$_{t\ (⟨sKs\ p \to t⟩_{n_r})}$ **using** *sb.nxtActI* **by** *blast*
ultimately show *?thesis*
using *sb.baEANow*[*of sKs p t t''* ⟨*sKs p* → *t*⟩$_{n_r}$] **by** *simp*
qed
with ‹*p'* ∈ *P*› **have** (*p'*, *p*) ∈ *sb*
using ‹*sub P* = *ksrp* ($\sigma_{sKs\ p}$(*t* (⟨*sKs p* → *t*⟩$_{n_r}$)))›
sks-prob **by** *simp*
moreover from ‹∃ *i*≥n_r. ‖*sKs p*‖$_{t\ i}$› **have** ‖*sKs p*‖$_{t\ (⟨sKs\ p \to t⟩_{n_r})}$
using *sb.nxtActI* **by** *blast*
with ‹*sub P* = *ksrp* ($\sigma_{sKs\ p}$(*t* (⟨*sKs p* → *t*⟩$_{n_r}$)))›
have *sub P* ∈ *bbrp* ($\sigma_{the\text{-}bb}$(*t* (⟨*sKs p* → *t*⟩$_{n_r}$)))
using *conn1A* **by** *auto*
with ‹ *p'* ∈ *P*› **have** *sub P* ∈ *bbrp* ($\sigma_{the\text{-}bb}$*t* (⟨*sKs p* → *t*⟩$_{n_r}$)) ∧ *p'* ∈ *P* **by** *auto*
ultimately obtain *m* **where** *m*≥⟨*sKs p* → *t*⟩$_{n_r}$ **and** (*p'*, *solve p'*) = *bbcs* ($\sigma_{the\text{-}bb}$(*t*

m))

using *indH* **by** *auto*

— and due to the publisher subscriber property,
— the knowledge source will receive them
moreover have
$\nexists\, n\ P.\ \langle sKs\ p \rightarrow t\rangle_{n_r} \leq n \wedge n \leq m \wedge \|sKs\ p\|_{t\ n} \wedge$
$unsub\ P = ksrp\ (\sigma_{sKs\ p}(t\ n)) \wedge p' \in P$
proof
assume $\exists\, n\ P'.\ \langle sKs\ p \rightarrow t\rangle_{n_r} \leq n \wedge n \leq m \wedge \|sKs\ p\|_{t\ n} \wedge$
$unsub\ P' = ksrp\ (\sigma_{sKs\ p}(t\ n)) \wedge p' \in P'$
then obtain $n\ P'$ **where**
$\|sKs\ p\|_{t\ n}$ **and** $\langle sKs\ p \rightarrow t\rangle_{n_r} \leq n$ **and** $n \leq m$ **and**
$unsub\ P' = ksrp\ (\sigma_{sKs\ p}(t\ n))$ **and** $p' \in P'$ **by** *auto*
hence $sb.eval\ (sKs\ p)\ t\ t''\ n\ (\exists_b\ P'.\ sb.pred\ (p' \in P') \wedge^b$
$[\lambda ks.\ unsub\ P' = ksrp\ ks]_b)$ **by** *blast*
moreover have $\langle sKs\ p \Leftarrow t\rangle_{n_r} \leq n$
using $\langle n_r \leq \langle sKs\ p \rightarrow t\rangle_{n_r}\rangle$ $\langle\langle sKs\ p \Leftarrow t\rangle_{n_r} \leq n_r\rangle$
$\langle\langle sKs\ p \rightarrow t\rangle_{n_r} \leq n\rangle$ **by** *linarith*
with *cass* **have** $sb.eval\ (sKs\ p)\ t\ t''\ n\ (\neg^b\ (\exists_b\ P'.\ (sb.pred\ (p' \in P')$
$\wedge^b\ [\lambda ks.\ unsub\ P' = ksrp\ ks]_b)))$
using $sb.globEA[of\ n_r\ sKs\ p\ t\ t''$
$\neg^b\ (\exists_b P'.\ sb.pred\ (p' \in P') \wedge^b\ [\lambda ks.\ unsub\ P' = ksrp\ ks]_b)\ n]$
$\langle\exists\, i \geq n_r.\ \|sKs\ p\|_{t\ i}\rangle$ **by** *auto*
ultimately show *False* **using** *sb.negE* **by** *auto*
qed
moreover from $\langle\exists\, i \geq n_r.\ \|sKs\ p\|_{t\ i}\rangle$ **have**
$\|sKs\ p\|_{t\ (\langle sKs\ p \rightarrow t\rangle_{n_r})}$ **using** *sb.nxtActI* **by** *blast*
moreover have $sub\ P = ksrp\ (\sigma_{sKs\ p}(t\ (\langle sKs\ p \rightarrow t\rangle_{n_r})))$
using $\langle sub\ P = ksrp\ (\sigma_{sKs\ p}(t\ (\langle sKs\ p \rightarrow t\rangle_{n_r})))\rangle$ **.**
moreover from $\langle m \geq \langle sKs\ p \rightarrow t\rangle_{n_r}\rangle$ **have** $\langle sKs\ p \rightarrow t\rangle_{n_r} \leq m$ **by** *simp*
moreover from $\langle\exists\, i \geq n_r.\ \|sKs\ p\|_{t\ i}\rangle$
have $\langle sKs\ p \rightarrow t\rangle_{n_r} \geq n_r$ **using** *sb.nxtActI* **by** *blast*
hence $m \geq n_k$ **using** $\langle\langle sKs\ p \rightarrow t\rangle_{n_r} \leq m\rangle$ $\langle\langle sKs\ p \rightarrow t\rangle_{n_k} \leq n_r\rangle$
$\langle\langle sKs\ p \rightarrow t\rangle_{n_k} \geq n_k\rangle$ **by** *simp*
with *case-ass* **have** $\|sKs\ p\|_{t\ m}$ **by** *simp*
ultimately have $(p', solve\ p') \in kscs\ (\sigma_{sKs\ p}(t\ m))$
and $\|sKs\ p\|_{t\ m}$
using $\langle t \in arch\rangle$ $msgDelivery[of\ t\ sKs\ p\ \langle sKs\ p \rightarrow t\rangle_{n_r}\ P\ m\ p'\ solve\ p']$
$\langle p' \in P\rangle$ **by** *auto*
hence $sb.eval\ (sKs\ p)\ t\ t''\ m\ [\lambda ks.\ (p', solve\ p') \in kscs\ ks]_b$
using *sb.baIANow* **by** *simp*
moreover have $m \geq \langle sKs\ p \Leftarrow t\rangle_m$ **by** *simp*
moreover from $\langle\|sKs\ p\|_{t\ m}\rangle$ **have** $m \leq \langle sKs\ p \rightarrow t\rangle_m$
using *sb.nxtActLe* **by** *auto*
moreover from $\langle\exists\, i \geq n_r.\ \|sKs\ p\|_{t\ i}\rangle$ **have**
$\langle sKs\ p \Leftarrow t\rangle_{n_r} \leq \langle sKs\ p \rightarrow t\rangle_{n_r}$ **by** *simp*
with $\langle\langle sKs\ p \rightarrow t\rangle_{n_r} \leq m\rangle$ **have** $\langle sKs\ p \Leftarrow t\rangle_{n_r} \leq m$ **by** *arith*
ultimately show $sb.eval\ (sKs\ p)\ t\ t''\ n_r$
$(\Diamond_b\ [\lambda ks.\ (p', solve\ p') \in kscs\ ks]_b)$
using $\langle\exists\, i \geq n_r.\ \|sKs\ p\|_{t\ i}\rangle$ *sb.evtIA* **by** *blast*
qed
qed
qed
thus $sb.eval\ (sKs\ p)\ t\ t''\ n_r\ (sb.pred\ (p' \in P) \longrightarrow^b$

$(\Diamond_b\ [\lambda ks.\ (p', solve\ p') \in kscs\ ks]_b))$
using *sb.impI* **by** *auto*
qed
thus *?thesis* **using** *sb.allI* **by** *blast*
qed

— Thus, the knowlege source will eventually solve the problem at hand...
ultimately have $sb.eval\ (sKs\ p)\ t\ t''\ n_r$
$([\lambda ks.\ sub\ P = ksrp\ ks]_b \wedge^b$
$(\forall_b\ q.\ (sb.pred\ (q \in P) \longrightarrow^b \Diamond_b\ [\lambda ks.\ (q,\ solve\ q) \in kscs\ ks]_b)))$
using *sb.conjI* **by** *simp*
moreover from $\langle \exists\, i{\geq}n_r.\ \|sKs\ p\|_{t\ i}\rangle$ **have** $\exists\, i{\geq}0.\ \|sKs\ p\|_{t\ i}$ **by** *blast*
hence $sb.eval\ (sKs\ p)\ t\ t''\ n_r$
$(([\lambda ks.\ sub\ P = ksrp\ ks]_b \wedge^b$
$(\forall_b\ q.\ (sb.pred\ (q \in P) \longrightarrow^b$
$\Diamond_b\ [\lambda ks.\ (q,\ solve\ q) \in kscs\ ks]_b))) \longrightarrow^b$
$(\Diamond_b\ [\lambda ks.\ (p,\ solve\ p) = ksns\ ks]_b))$ **using** $\langle t \in arch\rangle$
$sb.globEA[OF\text{ - }bhvks1[of\ t\ p\ sKs\ p\ t''\ P]]\ sks\text{-}prob$ **by** *simp*
ultimately have $sb.eval\ (sKs\ p)\ t\ t''\ n_r$
$(\Diamond_b\ [\lambda ks.\ (p,solve(p))=ksns(ks)]_b)$
using $sb.impE[of\ sKs\ p\ t\ t''\ n_r]$ **by** *blast*

— and forward it to the blackboard
then obtain n_s **where** $n_s{\geq}\langle sKs\ p \rightarrow t\rangle_{n_r}$ **and**
$(\exists\, i{\geq}n_s.\ \|sKs\ p\|_{t\ i} \wedge$
$(\forall\, n''{\geq}\langle sKs\ p \Leftarrow t\rangle_{n_s}.\ n'' \leq \langle sKs\ p \rightarrow t\rangle_{n_s} \longrightarrow$
$sb.eval\ (sKs\ p)\ t\ t''\ n''\ [\lambda ks.\ (p,solve(p))=ksns(ks)]_b)) \vee$
$\neg\ (\exists\, i{\geq}n_s.\ \|sKs\ p\|_{t\ i}) \wedge$
$sb.eval\ (sKs\ p)\ t\ t''\ n_s\ [\lambda ks.\ (p,solve(p))=ksns(ks)]_b$
using $sb.evtEA[of\ n_r\ sKs\ p\ t]\ \langle \exists\, i{\geq}n_r.\ \|sKs\ p\|_{t\ i}\rangle$ **by** *blast*
moreover from $\langle\langle sKs\ p \rightarrow t\rangle_{n_r} \geq n_r\rangle\ \langle n_r{\geq}n_k\rangle\ \langle n_s{\geq}\langle sKs\ p \rightarrow t\rangle_{n_r}\rangle$
have $n_s{\geq}n_k$ **by** *arith*
with *case-ass* **have** $\exists\, i{\geq}n_s.\ \|sKs\ p\|_{t\ i}$ **by** *auto*
moreover have $n_s{\geq}\langle sKs\ p \Leftarrow t\rangle_{n_s}$ **by** *simp*
moreover from $\langle \exists\, i{\geq}n_s.\ \|sKs\ p\|_{t\ i}\rangle$ **have** $n_s \leq \langle sKs\ p \rightarrow t\rangle_{n_s}$
using *sb.nxtActLe* **by** *simp*
ultimately have $sb.eval\ (sKs\ p)\ t\ t''\ n_s\ [\lambda ks.\ (p,solve(p))=ksns(ks)]_b$
using $sb.evtEA[of\ n_r\ sKs\ p\ t]\ \langle \exists\, i{\geq}n_r.\ \|sKs\ p\|_{t\ i}\rangle$ **by** *blast*
with $\langle \exists\, i{\geq}n_s.\ \|sKs\ p\|_{t\ i}\rangle$ **have**
$(p,solve(p)) = ksns\ (\sigma_{sKs\ p}(t\ (\langle sKs\ p \rightarrow t\rangle_{n_s})))$
using $sb.baEA[of\ n_s\ sKs\ p\ t\ t''\ \lambda ks.\ (p,\ solve\ p) = ksns\ ks]$ **by** *auto*
moreover from $\langle \exists\, i{\geq}n_s.\ \|sKs\ p\|_{t\ i}\rangle$
have $\|sKs\ p\|_{t\ (\langle sKs\ p \rightarrow t\rangle_{n_s})}$ **using** *sb.nxtActI* **by** *simp*
ultimately have $(p,solve(p)) \in bbns\ (\sigma_{the\text{-}bb}(t\ (\langle sKs\ p \rightarrow t\rangle_{n_s})))$
using $conn1[OF\ pb.ts\text{-}prop(2)[of\ t\ (\langle sKs\ p \rightarrow t\rangle_{n_s})]]$ **by** *auto*
hence $pb.eval\ the\text{-}bb\ t\ t'\ \langle sKs\ p \rightarrow t\rangle_{n_s}\ [\lambda bb.\ (p,solve(p)) \in bbns\ bb]_b$
using $\langle t{\in}arch\rangle\ pb.baI$ **by** *simp*

— finally, the blackboard will forward the solution which finishes the proof.
with *bhvbb1* **have** $pb.eval\ the\text{-}bb\ t\ t'\ \langle sKs\ p \rightarrow t\rangle_{n_s}$
$(\Diamond_b\ [\lambda bb.\ (p,\ solve\ p) = bbcs\ bb]_b)$
using $\langle t{\in}arch\rangle\ pb.globE\ pb.impE[of\ the\text{-}bb\ t\ t']$ **by** *blast*
then obtain n_f **where** $n_f{\geq}\langle sKs\ p \rightarrow t\rangle_{n_s}$ **and**
$pb.eval\ the\text{-}bb\ t\ t'\ n_f\ [\lambda bb.\ (p,\ solve\ p) = bbcs\ bb]_b$
using $\langle t{\in}arch\rangle\ pb.evtE[of\ t\ t'\ \langle sKs\ p \rightarrow t\rangle_{n_s}]$ **by** *auto*

```
          hence (p, solve p) = bbcs (σ_the-bb(t n_f))
            using ‹t ∈ arch› pb.baEA by auto
          moreover have n_f≥n_0
          proof −
            from ‹∃ n'''≥n_s. ‖sKs p‖_{t n'''}› have ⟨sKs p → t⟩_{n_s}≥n_s
              using sb.nxtActLe by simp
            moreover from ‹n_k≥n› and ‹‖sKs p‖_{t n_k}› have ⟨sKs p → t⟩_{n_k}≥n_k
              using sb.nxtActI by blast
            ultimately show ?thesis
              using ‹n_f≥⟨sKs p → t⟩_{n_s}› ‹n_s≥⟨sKs p → t⟩_{n_r}›
              ‹⟨sKs p → t⟩_{n_r}≥n_r› ‹n_r≥⟨sKs p → t⟩_{n_k}› ‹n_k≥n› ‹n≥n_0› by arith
          qed
          ultimately show ?thesis by auto
        qed
      qed
    qed
  qed

  theorem pSolved:
    fixes t and t'::nat ⇒'BB and t''::nat ⇒'KS
    assumes t∈arch and
      ∀ n. (∃ n'≥n. ‖sKs (bbop(σ_the-bb(t n)))‖_{t n'})
    shows
      ∀ n. (∀ P. (sub P ∈ bbrp(σ_the-bb(t n))
        ⟶ (∀ p ∈ P. (∃ m≥n. (p,solve(p)) = bbcs (σ_the-bb(t m))))))
    using assms pSolved-Ind by blast
end

end
```

4 Some Auxiliary Results

```
theory Auxiliary imports Main
begin

lemma disjE3: P ∨ Q ∨ R ⟹ (P ⟹ S) ⟹ (Q ⟹ S) ⟹ (R ⟹ S) ⟹ S by auto

lemma ge-induct[consumes 1, case-names step]:
  fixes i::nat and j::nat and P::nat ⇒ bool
  shows i ≤ j ⟹ (⋀n. i ≤ n ⟹ ((∀ m ≥ i.  m<n ⟶ P m) ⟹ P n)) ⟹ P j
proof −
  assume a0: i ≤ j and a1: (⋀n. i ≤ n ⟹ ((∀ m ≥ i.  m<n ⟶ P m) ⟹ P n))
  have (⋀n. ∀ m<n. i ≤ m ⟶ P m ⟹ i ≤ n ⟶ P n)
  proof
    fix n
    assume a2: ∀ m<n. i ≤ m ⟶ P m
    show i ≤ n ⟹ P n
    proof −
      assume i ≤ n
      with a1 have (∀ m ≥ i.  m<n ⟶ P m) ⟹ P n by simp
      moreover from a2 have ∀ m ≥ i.  m<n ⟶ P m by simp
      ultimately show P n by simp
    qed
  qed
```

```
  with nat-less-induct[of λj. i ≤ j ⟶ P j j] have i ≤ j ⟶ P j .
  with a0 show ?thesis by simp
qed

lemma my-induct[consumes 1, case-names base step]:
  fixes P::nat⇒bool
assumes less: i ≤ j
    and base: P j
    and step: ⋀n. i ≤ n ⟹ n < j ⟹ (∀ n'>n. n'≤j ⟶ P n') ⟹ P n
  shows P i
proof cases
  assume j=0
  thus ?thesis using less base by simp
next
  assume ¬ j=0
  have j − (j − i) ≥ i ⟶ P (j − (j − i))
  proof (rule less-induct[of λn::nat. j−n ≥ i ⟶ P (j−n) j−i])
    fix x assume asmp: ⋀y. y < x ⟹ i ≤ j − y ⟶ P (j − y)
    show i ≤ j − x ⟶ P (j − x)
    proof cases
      assume x=0
      with base show ?thesis by simp
    next
      assume ¬ x=0
      with ‹j ≠ 0› have j − x < j by simp
      show ?thesis
      proof
        assume i ≤ j − x
        moreover have ∀ n'>j−x. n'≤j ⟶ P n'
        proof
          fix n'
          show n'>j−x ⟶ n'≤j ⟶ P n'
          proof (rule HOL.impI[OF HOL.impI])
            assume j − x < n' and n' ≤ j
            hence j − n' < x by simp
           moreover from ‹i ≤ j − x› ‹j − x < n'› have i ≤ n' using le-less-trans less-imp-le-nat by
blast
            with ‹n' ≤ j› have i ≤ j − (j − n') by simp
            ultimately have  P (j − (j − n')) using asmp by simp
            moreover from ‹n' ≤ j› have j − (j − n') = n' by simp
            ultimately show P n' by simp
          qed
        qed
        ultimately show P (j − x) using ‹j−x<j› step[of j−x] by simp
      qed
    qed
  qed
  moreover from less have j − (j − i) = i by simp
  ultimately show ?thesis by simp
qed

lemma Greatest-ex-le-nat: assumes ∃ k. P k ∧ (∀ k'. P k' ⟶ k' ≤ k) shows ¬(∃ n'>Greatest P. P n')
  by (metis Greatest-equality assms less-le-not-le)

lemma cardEx: assumes finite A and finite B and card A > card B shows ∃ x∈A. ¬ x∈B
```

proof *cases*
assume $A \subseteq B$
with *assms* **have** *card* $A \leq$*card* B **using** *card-mono* **by** *blast*
with *assms* **have** *False* **by** *simp*
thus *?thesis* **by** *simp*
next
assume $\neg\ A \subseteq B$
thus *?thesis* **by** *auto*
qed

lemma *cardshift*: *card* $\{i$::*nat*. $i>n \wedge i \leq n' \wedge p\ (n'' + i)\}$ = *card* $\{i.\ i>(n + n'') \wedge i \leq (n' + n'') \wedge p\ i\}$
proof −
let *?f*=$\lambda i.\ i+n''$
have *bij-betw ?f* $\{i$::*nat*. $i>n \wedge i \leq n' \wedge p\ (n'' + i)\}$ $\{i.\ i>(n + n'') \wedge i \leq (n' + n'') \wedge p\ i\}$
proof (*rule bij-betwI'*)
fix $x\ y$ **assume** $x \in \{i.\ n < i \wedge i \leq n' \wedge p\ (n'' + i)\}$ **and** $y \in \{i.\ n < i \wedge i \leq n' \wedge p\ (n'' + i)\}$
show $(x + n'' = y + n'') = (x = y)$ **by** *simp*
next
fix x::*nat* **assume** $x \in \{i.\ n < i \wedge i \leq n' \wedge p\ (n'' + i)\}$
hence $n<x$ **and** $x \leq n'$ **and** $p(n''+x)$ **by** *auto*
moreover have $n''+x=x+n''$ **by** *simp*
ultimately have $n + n'' < x + n''$ **and** $x + n'' \leq n' + n''$ **and** $p\ (x + n'')$ **by** *auto*
thus $x + n'' \in \{i.\ n + n'' < i \wedge i \leq n' + n'' \wedge p\ i\}$ **by** *auto*
next
fix y::*nat* **assume** $y \in \{i.\ n + n'' < i \wedge i \leq n' + n'' \wedge p\ i\}$
hence $n+n''<y$ **and** $y \leq n'+n''$ **and** $p\ y$ **by** *auto*
then obtain x **where** $x=y-n''$ **by** *simp*
with ‹$n+n''<y$› **have** $y=x+n''$ **by** *simp*
moreover from ‹$x=y-n''$› ‹$n+n''<y$› **have** $x>n$ **by** *simp*
moreover from ‹$x=y-n''$› ‹$y \leq n'+n''$› **have** $x \leq n'$ **by** *simp*
moreover from ‹$y=x+n''$› **have** $y=n''+x$ **by** *simp*
with ‹$p\ y$› **have** $p\ (n'' + x)$ **by** *simp*
ultimately show $\exists x \in \{i.\ n < i \wedge i \leq n' \wedge p\ (n'' + i)\}.\ y = x + n''$ **by** *auto*
qed
thus *?thesis* **using** *bij-betw-same-card* **by** *auto*
qed

end

5 Relative Frequency LTL

theory *RF-LTL*
imports *Main HOL−Library.Sublist Auxiliary DynamicArchitectures.Dynamic-Architecture-Calculus*
begin

type-synonym *'s seq* = *nat* ⇒ *'s*

abbreviation *ccard n n' p* ≡ *card* $\{i.\ i>n \wedge i \leq n' \wedge p\ i\}$

lemma *ccard-same*:
assumes $\neg\ p$ (*Suc n'*)
shows *ccard n n' p* = *ccard n* (*Suc n'*) *p*
proof −
have $\{i.\ i > n \wedge i \leq$ *Suc n'* $\wedge\ p\ i\} = \{i.\ i>n \wedge i \leq n' \wedge p\ i\}$

```
  proof
    show {i. n < i ∧ i ≤ Suc n' ∧ p i} ⊆ {i. n < i ∧ i ≤ n' ∧ p i}
    proof
      fix x assume x ∈ {i. n < i ∧ i ≤ Suc n' ∧ p i}
      hence n<x and x≤Suc n' and p x by auto
      with assms (1) have x≠Suc n' by auto
      with ‹x≤Suc n'› have x ≤ n' by simp
      with ‹n<x› ‹p x› show x ∈ {i. n < i ∧ i ≤ n' ∧ p i} by simp
    qed
  next
    show {i. n < i ∧ i ≤ n' ∧ p i} ⊆ {i. n < i ∧ i ≤ Suc n' ∧ p i} by auto
  qed
  thus ?thesis by simp
qed

lemma ccard-zero[simp]:
  fixes n::nat
  shows ccard n n p = 0
  by auto

lemma ccard-inc:
  assumes p (Suc n')
    and n' ≥ n
  shows ccard n (Suc n') p = Suc (ccard n n' p)
proof −
  let ?A = {i. i > n ∧ i ≤ n' ∧ p i}
  have finite ?A by simp
  moreover have Suc n' ∉ ?A by simp
  ultimately have card (insert (Suc n') ?A) = Suc (card ?A) using card-insert-disjoint[of ?A] by simp
  moreover have insert (Suc n') ?A = {i. i>n ∧ i ≤ (Suc n') ∧ p i}
  proof
    show insert (Suc n') ?A ⊆ {i. n < i ∧ i ≤ Suc n' ∧ p i}
    proof
      fix x assume x ∈ insert (Suc n') {i. n < i ∧ i ≤ n' ∧ p i}
      hence x=Suc n' ∨ n < x ∧ x ≤ n' ∧ p x by simp
      thus x ∈ {i. n < i ∧ i ≤ Suc n' ∧ p i}
      proof
        assume x = Suc n'
        with assms (1) assms (2) show ?thesis by simp
      next
        assume n < x ∧ x ≤ n' ∧ p x
        thus ?thesis by simp
      qed
    qed
  next
    show {i. n < i ∧ i ≤ Suc n' ∧ p i} ⊆ insert (Suc n') ?A by auto
  qed
  ultimately show ?thesis by simp
qed

lemma ccard-mono:
  assumes n'≥n
  shows n''≥n' ⟹ ccard n (n''::nat) p ≥ ccard n n' p
proof (induction n'' rule: dec-induct)
  case base
```

```
  then show ?case ..
next
  case (step n'')
  then show ?case
  proof cases
    assume p (Suc n'')
    moreover from step.hyps assms have n≤n'' by simp
    ultimately have ccard n (Suc n'') p = Suc (ccard n n'' p) using ccard-inc[of p n'' n] by simp
    also have ... ≥ ccard n n' p using step.IH by simp
    finally show ?case .
  next
    assume ¬ p (Suc n'')
    moreover from step.hyps assms have n≤n'' by simp
    ultimately have ccard n (Suc n'') p = ccard n n'' p using ccard-same[of p n'' n] by simp
    also have ... ≥ ccard n n' p using step.IH by simp
    finally show ?case by simp
  qed
qed

lemma ccard-ub[simp]:
  ccard n n' p ≤ Suc n' − n
proof −
  have {i. i>n ∧ i ≤ n' ∧ p i} ⊆ {i. i≥n ∧ i ≤ n'} by auto
  hence ccard n n' p ≤ card {i. i≥n ∧ i ≤ n'} by (simp add: card-mono)
  moreover have {i. i≥n ∧ i ≤ n'} = {n..n'} by auto
  hence card {i. i≥n ∧ i ≤ n'} = Suc n' − n by simp
  ultimately show ?thesis by simp
qed

lemma ccard-sum:
  fixes n::nat
  assumes n'≥n''
    and n''≥n
  shows ccard n n' P = ccard n n'' P + ccard n'' n' P
proof −
  have ccard n n' P = card {i. i>n ∧ i ≤ n' ∧ P i} by simp
  moreover have {i. i>n ∧ i ≤ n' ∧ P i} =
    {i. i>n ∧ i ≤ n'' ∧ P i} ∪ {i. i>n'' ∧ i ≤ n' ∧ P i} (is ?LHS = ?RHS)
  proof
    show ?LHS ⊆ ?RHS by auto
  next
    show ?RHS ⊆ ?LHS
    proof
      fix x
      assume x∈?RHS
      hence x>n ∧ x ≤ n'' ∧ P x ∨ x>n'' ∧ x ≤ n' ∧ P x by auto
      thus x∈?LHS
      proof
        assume n < x ∧ x ≤ n'' ∧ P x
        with assms show ?thesis by simp
      next
        assume n'' < x ∧ x ≤ n' ∧ P x
        with assms show ?thesis by simp
      qed
    qed
```

```
  qed
  hence card ?LHS = card ?RHS by simp
  ultimately have ccard n n' P = card ?RHS by simp
  moreover have card ?RHS = card {i. i>n ∧ i ≤ n'' ∧ P i} + card {i. i>n'' ∧ i ≤ n' ∧ P i}
  proof (rule card-Un-disjoint)
    show finite {i. n < i ∧ i ≤ n'' ∧ P i} by simp
    show finite {i. n'' < i ∧ i ≤ n' ∧ P i} by simp
    show {i. n < i ∧ i ≤ n'' ∧ P i} ∩ {i. n'' < i ∧ i ≤ n' ∧ P i} = {} by auto
  qed
  moreover have ccard n n'' P = card {i. i>n ∧ i ≤ n'' ∧ P i} by simp
  moreover have ccard n'' n' P= card {i. i>n'' ∧ i ≤ n' ∧ P i} by simp
  ultimately show ?thesis by simp
qed

lemma ccard-ex:
  fixes n::nat
  shows c≥1 ⟹ c < ccard n n'' P ⟹ ∃ n'<n''. n'>n ∧ ccard n n' P = c
proof (induction c rule: dec-induct)
  let ?l = LEAST i::nat. n < i ∧ i < n'' ∧ P i
  case base
  moreover have ccard n n'' P ≤ Suc (card {i. n < i ∧ i < n'' ∧ P i})
  proof −
    from ‹ccard n n'' P > 1› have n''>n using less-le-trans by force
    then obtain n' where Suc n' = n'' and Suc n' ≥ n by (metis lessE less-imp-le-nat)
    moreover have {i. n < i ∧ i < Suc n' ∧ P i} = {i. n < i ∧ i ≤ n' ∧ P i} by auto
    hence card {i. n < i ∧ i < Suc n' ∧ P i} = card {i. n < i ∧ i ≤ n' ∧ P i} by simp
    moreover have card {i. n < i ∧ i ≤ Suc n' ∧ P i} ≤ Suc (card {i. n < i ∧ i ≤ n' ∧ P i})
    proof cases
      assume P (Suc n')
      moreover from ‹n''>n› ‹Suc n'=n''› have n'≥n by simp
      ultimately show ?thesis using ccard-inc[of P n' n] by simp
    next
      assume ¬ P (Suc n')
      moreover from ‹n''>n› ‹Suc n'=n''› have n'≥n by simp
      ultimately show ?thesis using ccard-same[of P n' n] by simp
    qed
    ultimately show ?thesis by simp
  qed
  ultimately have card {i. n < i ∧ i < n'' ∧ P i} ≥ 1 by simp
  hence {i. n < i ∧ i < n'' ∧ P i} ≠ {} by fastforce
  hence ∃ i. n < i ∧ i < n'' ∧ P i by auto
  hence ?l>n and ?l<n'' and P ?l using LeastI-ex[of λi::nat. n < i ∧ i < n'' ∧ P i] by auto
  moreover have {i. n < i ∧ i ≤ ?l ∧ P i} = {?l}
  proof
    show {i. n < i ∧ i ≤ ?l ∧ P i} ⊆ {?l}
    proof
      fix i
      assume i∈{i. n < i ∧ i ≤ ?l ∧ P i}
      hence n < i and i ≤ ?l and P i by auto
      with ‹∃ i. n < i ∧ i < n'' ∧ P i› have i=?l
        using Least-le[of λi. n < i ∧ i < n'' ∧ P i] by (meson antisym le-less-trans)
      thus i∈{?l} by simp
    qed
  next
    show {?l} ⊆ {i. n < i ∧ i ≤ ?l ∧ P i}
```

```
    proof
      fix i
      assume i∈{?l}
      hence i=?l by simp
      with ‹?l>n› ‹?l<n''› ‹P ?l› show i∈{i. n < i ∧ i ≤ ?l ∧ P i} by simp
    qed
  qed
  hence ccard n ?l P = 1 by simp
  ultimately show ?case by auto
next
  case (step c)
  moreover from step.prems have Suc c<ccard n n'' P by simp
  ultimately obtain n' where n'<n'' and n < n' and ccard n n' P = c by auto
  hence ccard n n'' P = ccard n n' P + ccard n' n'' P using ccard-sum[of n' n'' n] by simp
  with ‹Suc c<ccard n n'' P› ‹ccard n n' P = c› have ccard n' n'' P>1 by simp
  moreover have ccard n' n'' P ≤ Suc (card {i. n' < i ∧ i < n'' ∧ P i})
  proof −
    from ‹ccard n' n'' P > 1› have n''>n' using less-le-trans by force
    then obtain n''' where Suc n''' = n'' and Suc n''' ≥ n' by (metis lessE less-imp-le-nat)
    moreover have {i. n' < i ∧ i < Suc n''' ∧ P i} = {i. n' < i ∧ i ≤ n''' ∧ P i} by auto
    hence card {i. n' < i ∧ i < Suc n''' ∧ P i} = card {i. n' < i ∧ i ≤ n''' ∧ P i} by simp
    moreover have card {i. n' < i ∧ i ≤ Suc n''' ∧ P i} ≤ Suc (card {i. n' < i ∧ i ≤ n''' ∧ P i})
    proof cases
      assume P (Suc n''')
      moreover from ‹n''>n'› ‹Suc n'''=n''› have n'''≥n' by simp
      ultimately show ?thesis using ccard-inc[of P n''' n'] by simp
    next
      assume ¬ P (Suc n''')
      moreover from ‹n''>n'› ‹Suc n'''=n''› have n'''≥n' by simp
      ultimately show ?thesis using ccard-same[of P n''' n'] by simp
    qed
    ultimately show ?thesis by simp
  qed
  ultimately have card {i. n' < i ∧ i < n'' ∧ P i} ≥ 1 by simp
  hence {i. n' < i ∧ i < n'' ∧ P i} ≠ {} by fastforce
  hence ∃ i. n' < i ∧ i < n'' ∧ P i by auto
  let ?l = LEAST i::nat. n' < i ∧ i < n'' ∧ P i
  from ‹∃ i. n' < i ∧ i < n'' ∧ P i› have n' < ?l
    using LeastI-ex[of λi::nat. n' < i ∧ i < n'' ∧ P i] by auto
  with ‹n < n'› have ccard n ?l P = ccard n n' P + ccard n' ?l P using ccard-sum[of n' ?l n] by simp
  moreover have {i. n' < i ∧ i ≤ ?l ∧ P i} = {?l}
  proof
    show {i. n' < i ∧ i ≤ ?l ∧ P i} ⊆ {?l}
    proof
      fix i
      assume i∈{i. n' < i ∧ i ≤ ?l ∧ P i}
      hence n' < i and i ≤ ?l and P i by auto
      with ‹∃ i. n' < i ∧ i < n'' ∧ P i› have i=?l
        using Least-le[of λi. n' < i ∧ i < n'' ∧ P i] by (meson antisym le-less-trans)
      thus i∈{?l} by simp
    qed
  next
    show {?l} ⊆ {i. n' < i ∧ i ≤ ?l ∧ P i}
    proof
      fix i
```

```
      assume i∈{?l}
      hence i=?l by simp
      moreover from ‹∃ i. n' < i ∧ i < n'' ∧ P i› have ?l<n'' and P ?l
        using LeastI-ex[of λi. n' < i ∧ i < n'' ∧ P i] by auto
      ultimately show i∈{i. n' < i ∧ i ≤ ?l ∧ P i} using ‹?l>n'› by simp
    qed
  qed
  hence ccard n' ?l P = 1 by simp
  ultimately have card {i. n < i ∧ i ≤ ?l ∧ P i} = Suc c using ‹ccard n n' P = c› by simp
  moreover from ‹∃ i. n' < i ∧ i < n'' ∧ P i› have n' < ?l and ?l < n'' and P ?l
    using LeastI-ex[of λi::nat. n' < i ∧ i < n'' ∧ P i] by auto
  with ‹n < n'› have n<?l and ?l<n'' by auto
  ultimately show ?case by auto
qed

lemma ccard-freq:
  assumes (n'::nat)≥n
    and ccard n n' P > ccard n n' Q + cnf
  shows ∃ n' n''. ccard n' n'' P > cnf ∧ ccard n' n'' Q ≤ cnf
proof cases
  assume cnf = 0
  with assms(2) have ccard n n' P > ccard n n' Q by simp
  hence card {i. n < i ∧ i ≤ n' ∧ P i}>card {i. n < i ∧ i ≤ n' ∧ Q i} (is card ?LHS>card ?RHS)
    by simp
  then obtain i where i∈?LHS and ¬ i ∈ ?RHS and i>0 using cardEx[of ?LHS ?RHS] by auto
  hence P i and ¬ Q i by auto
  with ‹i>0› obtain n'' where P (Suc n'') and ¬Q (Suc n'') using gr0-implies-Suc by auto
  hence ccard n'' (Suc n'') P = 1 using ccard-inc by auto
  with ‹cnf = 0› have ccard n'' (Suc n'') P > cnf by simp
  moreover from ‹¬Q (Suc n'')› have ccard n'' (Suc n'') Q = 0 using ccard-same[of Q n'' n'] by
auto
  with ‹cnf = 0› have ccard n'' (Suc n'') Q ≤ cnf by simp
  ultimately show ?thesis by auto
next
  assume ¬ cnf = 0
  show ?thesis
  proof (rule ccontr)
    assume ¬ (∃ n' n''. ccard n' n'' P > cnf ∧ ccard n' n'' Q ≤ cnf)
    hence hyp: ∀ n' n''. ccard n' n'' Q ≤ cnf ⟶ ccard n' n'' P ≤ cnf
      using leI less-imp-le-nat by blast
    show False
    proof cases
      assume ccard n n' Q ≤ cnf
      with hyp have ccard n n' P ≤ cnf by simp
      with assms show False by simp
    next
      let ?gcond=λn''. n''≥n ∧ n''≤n' ∧ (∃ x≥1. ccard n n'' Q = x * cnf)
      let ?g= GREATEST n''. ?gcond n''
      assume ¬ ccard n n' Q ≤ cnf
      hence ccard n n' Q > cnf by simp
      hence ∃ n''. ?gcond n''
      proof −
        from ‹ccard n n' Q > cnf› ‹¬cnf=0› obtain n'' where n''>n and n''≤n' and ccard n n'' Q
= cnf
          using ccard-ex[of cnf n n' Q ] by auto
```

moreover from ‹*ccard n n'' Q = cnf*› **have** $\exists x{\geq}1.$ *ccard n n'' Q = x * cnf* **by** *auto*
ultimately show *?thesis* **using** *less-imp-le-nat* **by** *blast*
qed
moreover have $\forall n''{>}n'.\ \neg$ *?gcond n''* **by** *simp*
ultimately have *gex*: $\exists n''.$ *?gcond n''* $\wedge$ $(\forall n'''.$ *?gcond n'''* $\longrightarrow$ $n'''{\leq}n'')$
using *boundedGreatest*[*of ?gcond - n'*] **by** *blast*
hence $\exists x{\geq}1.$ *ccard n ?g Q = x * cnf* **and** *?g* $\geq$ *n* **using** *GreatestI-ex-nat*[*of ?gcond*] **by** *auto*
moreover {fix *n''*
have $n''{\geq}n \Longrightarrow \exists x{\geq}1.$ *ccard n n'' Q = x * cnf* $\Longrightarrow$ *ccard n n'' P* $\leq$ *ccard n n'' Q*
proof (*induction n'' rule*: *ge-induct*)
case (*step n'*)
from *step.prems* **obtain** *x* **where** *x≥1* **and** *cas*: *ccard n n' Q = x * cnf* **by** *auto*
then show *?case*
proof *cases*
assume *x=1*
with *cas* **have** *ccard n n' Q = cnf* **by** *simp*
with *hyp* **have** *ccard n n' P* $\leq$ *cnf* **by** *simp*
with ‹*ccard n n' Q = cnf*› **show** *?thesis* **by** *simp*
next
assume $\neg$*x=1*
with ‹*x≥1*› **have** *x>1* **by** *simp*
hence *x−1* $\geq$ *1* **by** *simp*
moreover from ‹*cnf≠0*› ‹*x−1* $\geq$ *1*› **have** $(x{-}1) * cnf < x * cnf \wedge (x - 1) * cnf \neq 0$ **by** *auto*
with ‹*x−1* $\geq$ *1*› ‹*cnf≠0*›‹*ccard n n' Q = x * cnf*› **obtain** *n''*
where $n''{>}n$ **and** $n''{<}n'$ **and** *ccard n n'' Q = (x−1) * cnf*
using *ccard-ex*[*of* (*x−1*)*cnf n n' Q*] **by** *auto*
ultimately have $\exists x{\geq}1.$ *ccard n n'' Q = x * cnf* **and** $n''{\geq}n$ **by** *auto*
with ‹$n''{\geq}n$› ‹$n''{<}n'$› **have** *ccard n n'' P* $\leq$ *ccard n n'' Q* **using** *step.IH* **by** *simp*
moreover have *ccard n'' n' Q = cnf*
proof −
from ‹*x−1* $\geq$ *1*› **have** *x*cnf = ((x−1) * cnf) + cnf*
using *semiring-normalization-rules*(*2*)[*of* (*x* − *1*) *cnf*] **by** *simp*
with ‹*ccard n n'' Q = (x−1) * cnf*› ‹*ccard n n' Q = x * cnf*›
have *ccard n n' Q = ccard n n'' Q + cnf* **by** *simp*
moreover from ‹$n''{\geq}n$› ‹$n''{<}n'$› **have** *ccard n n' Q = ccard n n'' Q + ccard n'' n' Q*
using *ccard-sum*[*of n'' n' n*] **by** *simp*
ultimately show *?thesis* **by** *simp*
qed
moreover from ‹*ccard n'' n' Q = cnf*› **have** *ccard n'' n' P* $\leq$ *cnf* **using** *hyp* **by** *simp*
ultimately show *?thesis* **using** ‹$n''{\geq}n$› ‹$n''{<}n'$› *ccard-sum*[*of n'' n' n*] **by** *simp*
qed
qed } note *geq = this*
ultimately have *ccard n ?g P* $\leq$ *ccard n ?g Q* **by** *simp*
moreover have *ccard ?g n' P* $\leq$ *cnf*
proof (*rule ccontr*)
assume $\neg$ *ccard ?g n' P* $\leq$ *cnf*
hence *ccard ?g n' P > cnf* **by** *simp*
have *ccard ?g n' Q > cnf*
proof (*rule ccontr*)
assume $\neg$*ccard ?g n' Q > cnf*
hence *ccard ?g n' Q* $\leq$ *cnf* **by** *simp*
with ‹*ccard ?g n' P > cnf*› **show** *False*
using ‹$\neg$ $(\exists n'\ n''.$ *ccard n' n'' P > cnf* $\wedge$ *ccard n' n'' Q* $\leq$ *cnf*)› **by** *simp*
qed

```
      with ‹¬ cnf=0› obtain n'' where n''>?g and n''<n' and ccard ?g n'' Q = cnf
        using ccard-ex[of cnf ?g n' Q] by auto
      moreover have ∃x≥1. ccard n n'' Q = x * cnf
      proof −
        from ‹∃x≥1. ccard n ?g Q = x * cnf› obtain x where x≥1 and ccard n ?g Q = x * cnf by
auto
        from ‹n''>?g› ‹?g≥n› have ccard n n'' Q = ccard n ?g Q + ccard ?g n'' Q
          using ccard-sum[of ?g n'' n Q] by simp
        with ‹ccard n ?g Q = x * cnf› have ccard n n'' Q = x * cnf + ccard ?g n'' Q by simp
        with ‹ccard ?g n'' Q = cnf› have ccard n n'' Q = Suc x * cnf by simp
        thus ?thesis by auto
      qed
      moreover from ‹n''>?g› ‹?g≥n› have n''≥n by simp
      ultimately have ∃n''>?g. ?gcond n'' by auto
       moreover from gex have ∀n'''. ?gcond n''' ⟶ n'''≤?g using Greatest-le-nat[of ?gcond] by
auto
      ultimately show False by auto
    qed
    moreover from gex have n'≥?g using GreatestI-ex-nat[of ?gcond] by auto
    ultimately have ccard n n' P≤ccard n n' Q + cnf using ccard-sum[of ?g n' n] using ‹?g ≥ n›
by simp
    with assms show False by simp
  qed
 qed
qed

locale honest =
 fixes bc:: ('a list) seq
   and n::nat
 assumes growth: n'≠0 ⟹ n'≤n ⟹ bc n' = bc (n'−1) ∨ (∃b. bc n' = bc (n' − 1) @ b)
begin
end

locale dishonest =
 fixes bc:: ('a list) seq
   and mining::bool seq
 assumes growth: ⋀n::nat. prefix (bc (Suc n)) (bc n) ∨ (∃b::'a. bc (Suc n) = bc n @ [b]) ∧ mining
(Suc n)
begin

lemma prefix-save:
 assumes prefix sbc (bc n')
   and ∀n'''>n'. n'''≤n'' ⟶ length (bc n''') ≥ length sbc
 shows n''≥n' ⟹ prefix sbc (bc n'')
proof (induction n'' rule: dec-induct)
 case base
 with assms(1) show ?case by simp
next
 case (step n)
 from growth[of n] show ?case
 proof
   assume prefix (bc (Suc n)) (bc n)
   moreover from step.hyps have length (bc (Suc n)) ≥ length sbc using assms(2) by simp
   ultimately show ?thesis using step.IH using prefix-length-prefix by auto
 next
```

```
    assume (∃ b. bc (Suc n) = bc n @ [b]) ∧ mining (Suc n)
    with step.IH show ?thesis by auto
  qed
qed

theorem prefix-length:
  assumes prefix sbc (bc n') and ¬ prefix sbc (bc n'') and n'≤n''
  shows ∃ n'''>n'. n'''≤n'' ∧ length (bc n''') < length sbc
proof (rule ccontr)
  assume ¬ (∃ n'''>n'. n'''≤n'' ∧ length (bc n''') < length sbc)
  hence ∀ n'''>n'. n'''≤n'' ⟶ length (bc n''') ≥ length sbc by auto
  with assms have prefix sbc (bc n'') using prefix-save[of sbc n' n''] by simp
  with assms (2) show False by simp
qed

theorem grow-mining:
  assumes length (bc n) < length (bc (Suc n))
  shows mining (Suc n)
  using assms growth leD prefix-length-le by blast

lemma length-suc-length:
  length (bc (Suc n)) ≤ Suc (length (bc n))
  by (metis eq-iff growth le-SucI length-append-singleton prefix-length-le)

end

locale dishonest-growth =
  fixes bc:: nat seq
    and mining:: nat ⇒ bool
  assumes as1: ⋀n::nat. bc (Suc n) ≤ Suc (bc n)
    and as2: ⋀n::nat. bc (Suc n) > bc n ⟹ mining (Suc n)
begin

end

sublocale dishonest ⊆ dishonest-growth λn. length (bc n) using grow-mining length-suc-length by unfold-locales auto

context dishonest-growth
begin
  theorem ccard-diff-lgth:
    n'≥n ⟹ ccard n n' (λn. mining n) ≥ (bc n' − bc n)
  proof (induction n' rule: dec-induct)
    case base
    then show ?case by simp
    next
    case (step n')
    from as1 have bc (Suc n') < Suc (bc n') ∨ bc (Suc n') = Suc (bc n')
      using le-neq-implies-less by blast
    then show ?case
    proof
      assume bc (Suc n') < Suc (bc n')
      hence bc (Suc n') − bc n ≤ bc n' − bc n by simp
      moreover from step.hyps have ccard n (Suc n') (λn. mining n) ≥ ccard n n' (λn. mining n)
        using ccard-mono[of n n' Suc n'] by simp
```

```
    ultimately show ?thesis using step.IH by simp
  next
    assume bc (Suc n') = Suc (bc n')
    hence bc (Suc n') − bc n ≤ Suc (bc n' − bc n) by simp
    moreover from ‹bc (Suc n') = Suc (bc n')› have mining (Suc n') using as2 by simp
    with step.hyps have ccard n (Suc n') (λn. mining n) ≥ Suc (ccard n n' (λn. mining n))
      using ccard-inc by simp
    ultimately show ?thesis using step.IH by simp
  qed
 qed
end

locale honest-growth =
 fixes bc:: nat seq
  and mining:: nat ⇒ bool
  and init:: nat
 assumes as1: ⋀n::nat. bc (Suc n) ≥ bc n
  and as2: ⋀n::nat. mining (Suc n) ⟹ bc (Suc n) > bc n
begin
 lemma grow-mono: n'≥n⟹bc n'≥bc n
 proof (induction n' rule: dec-induct)
  case base
  then show ?case by simp
 next
  case (step n')
  then show ?case using as1[of n'] by simp
 qed

 theorem ccard-diff-lgth:
  shows n'≥n ⟹ bc n' − bc n ≥ ccard n n' (λn. mining n)
 proof (induction n' rule: dec-induct)
  case base
  then show ?case by simp
 next
  case (step n')
  then show ?case
  proof cases
   assume mining (Suc n')
   with as2 have bc (Suc n') > bc n' by simp
   moreover from step.hyps have bc n'≥bc n using grow-mono by simp
   ultimately have bc (Suc n') − bc n > bc n' − bc n by simp
   moreover from as1 have bc (Suc n') − bc n ≥ bc n' − bc n by (simp add: diff-le-mono)
   moreover from ‹mining (Suc n')› step.hyps
    have ccard n (Suc n') (λn. mining n) ≤ Suc (ccard n n' (λn. mining n))
    using ccard-inc by simp
   ultimately show ?thesis using step.IH by simp
  next
   assume ¬ mining (Suc n')
   hence ccard n (Suc n') (λn. mining n) ≤ (ccard n n' (λn. mining n)) using ccard-same by simp
   moreover from as1 have bc (Suc n') − bc n ≥ bc n' − bc n by (simp add: diff-le-mono)
   ultimately show ?thesis using step.IH by simp
  qed
 qed
end
```

locale *bounded-growth* = *hg*: *honest-growth hbc hmining* + *dg*: *dishonest-growth dbc dmining*
  for *hbc*:: *nat seq*
  and *dbc*:: *nat seq*
  and *hmining*:: *nat* ⇒ *bool*
  and *dmining*:: *nat* ⇒ *bool*
  and *sbc*::*nat*
  and *cnf*::*nat* +
 assumes *fair*: ⋀*n n'*. *ccard n n'* (λ*n*. *dmining n*) > *cnf* ⟹ *ccard n n'* (λ*n*. *hmining n*) > *cnf*
  and *a2*: *hbc 0* ≥ *sbc*+*cnf*
  and *a3*: *dbc 0* < *sbc*
begin

theorem *hn-upper-bound*: **shows** *dbc n* < *hbc n*
proof (*rule ccontr*)
 assume ¬ *dbc n* < *hbc n*
 hence *dbc n* ≥ *hbc n* **by** *simp*
 moreover from *a2 a3* **have** *hbc 0* > *dbc 0* + *cnf* **by** *simp*
 moreover have *hbc n*≥*hbc 0* **using** *hg.grow-mono* **by** *simp*
 ultimately have *dbc n* − *dbc 0* > *hbc n* − *hbc 0* + *cnf* **by** *simp*
 moreover have *ccard 0 n* (λ*n*. *hmining n*) ≤ *hbc n* − *hbc 0* **using** *hg.ccard-diff-lgth* **by** *simp*
 moreover have *dbc n* − *dbc 0* ≤ *ccard 0 n* (λ*n*. *dmining n*) **using** *dg.ccard-diff-lgth* **by** *simp*
 ultimately have *ccard 0 n* (λ*n*. *dmining n*) > *ccard 0 n* (λ*n*. *hmining n*) + *cnf* **by** *simp*
 hence ∃ *n' n''*. *ccard n' n''* (λ*n*. *dmining n*) > *cnf* ∧ *ccard n' n''* (λ*n*. *hmining n*) ≤ *cnf*
  using *ccard-freq* **by** *blast*
 with *fair* **show** *False* **using** *leD* **by** *blast*
qed

end

end

6 Blockchain Architectures

theory *Blockchain* **imports** *Auxiliary DynamicArchitectures.Dynamic-Architecture-Calculus RF-LTL*
begin

6.1 Blockchains

A blockchain itself is modeled as a simple list.

type-synonym *'a BC* = *'a list*

abbreviation *max-cond*:: (*'a BC*) *set* ⇒ *'a BC* ⇒ *bool*
 where *max-cond B b* ≡ *b* ∈ *B* ∧ (∀ *b'*∈*B*. *length b'* ≤ *length b*)

no-syntax
 -MAX1 :: *pttrns* ⇒ *'b* ⇒ *'b* ((*3MAX -./ -*) [*0*, *10*] *10*)
 -MAX :: *pttrn* ⇒ *'a set* ⇒ *'b* ⇒ *'b* ((*3MAX -:-./ -*) [*0*, *0*, *10*] *10*)
 -MAX1 :: *pttrns* ⇒ *'b* ⇒ *'b* ((*3MAX -./ -*) [*0*, *10*] *10*)
 -MAX :: *pttrn* ⇒ *'a set* ⇒ *'b* ⇒ *'b* ((*3MAX -∈-./ -*) [*0*, *0*, *10*] *10*)

definition *MAX*:: (*'a BC*) *set* ⇒ *'a BC*
 where *MAX B* = (*SOME b*. *max-cond B b*)

lemma *max-ex*:

fixes *XS*::('*a BC*) *set*
assumes *XS* ≠ {}
and *finite XS*
shows ∃ *xs*∈*XS*. (∀ *ys*∈*XS*. *length ys* ≤ *length xs*)
proof (*rule Finite-Set.finite-ne-induct*)
show *finite XS* **using** *assms* **by** *simp*
next
from *assms* **show** *XS* ≠ {} **by** *simp*
next
fix *x*::'*a BC*
show ∃ *xs*∈{*x*}. ∀ *ys*∈{*x*}. *length ys* ≤ *length xs* **by** *simp*
next
fix *zs*::'*a BC* **and** *F*::('*a BC*) *set*
assume *finite F* **and** *F* ≠ {} **and** *zs* ∉ *F* **and** ∃ *xs*∈*F*. ∀ *ys*∈*F*. *length ys* ≤ *length xs*
then obtain *xs* **where** *xs*∈*F* **and** ∀ *ys*∈*F*. *length ys* ≤ *length xs* **by** *auto*
show ∃ *xs*∈*insert zs F*. ∀ *ys*∈*insert zs F*. *length ys* ≤ *length xs*
proof (*cases*)
assume *length zs* ≥ *length xs*
with ‹∀ *ys*∈*F*. *length ys* ≤ *length xs*› **show** *?thesis* **by** *auto*
next
assume ¬ *length zs* ≥ *length xs*
hence *length zs* ≤ *length xs* **by** *simp*
with ‹*xs* ∈ *F*› **show** *?thesis* **using** ‹∀ *ys*∈*F*. *length ys* ≤ *length xs*› **by** *auto*
qed
qed

lemma *max-prop*:
fixes *XS*::('*a BC*) *set*
assumes *XS* ≠ {}
and *finite XS*
shows *MAX XS* ∈ *XS*
and ∀ *b'*∈*XS*. *length b'* ≤ *length* (*MAX XS*)
proof −
from *assms* **have** ∃ *xs*∈*XS*. ∀ *ys*∈*XS*. *length ys* ≤ *length xs* **using** *max-ex*[*of XS*] **by** *auto*
with *MAX-def*[*of XS*] **show** *MAX XS* ∈ *XS* **and** ∀ *b'*∈*XS*. *length b'* ≤ *length* (*MAX XS*)
using *someI-ex*[*of* λ*b*. *b* ∈ *XS* ∧ (∀ *b'*∈*XS*. *length b'* ≤ *length b*)] **by** *auto*
qed

lemma *max-less*:
fixes *b*::'*a BC* **and** *b'*::'*a BC* **and** *B*::('*a BC*) *set*
assumes *b*∈*B*
and *finite B*
and *length b* > *length b'*
shows *length* (*MAX B*) > *length b'*
proof −
from *assms* **have** ∃ *xs*∈*B*. ∀ *ys*∈*B*. *length ys* ≤ *length xs* **using** *max-ex*[*of B*] **by** *auto*
with *MAX-def*[*of B*] **have** ∀ *b'*∈*B*. *length b'* ≤ *length* (*MAX B*)
using *someI-ex*[*of* λ*b*. *b* ∈ *B* ∧ (∀ *b'*∈*B*. *length b'* ≤ *length b*)] **by** *auto*
with ‹*b*∈*B*› **have** *length b* ≤ *length* (*MAX B*) **by** *simp*
with ‹*length b* > *length b'*› **show** *?thesis* **by** *simp*
qed

6.2 Blockchain Architectures

In the following we describe the locale for blockchain architectures.

locale *Blockchain* = *dynamic-component cmp active*
  for *active* :: *'nid* ⇒ *cnf* ⇒ *bool* (‖-‖_ [0,110]60)
    and *cmp* :: *'nid* ⇒ *cnf* ⇒ *'ND* (σ_(-) [0,110]60) +
  fixes *pin* :: *'ND* ⇒ (*'nid BC*) *set*
    and *pout* :: *'ND* ⇒ *'nid BC*
    and *bc* :: *'ND* ⇒ *'nid BC*
    and *mining* :: *'ND* ⇒ *bool*
    and *honest* :: *'nid* ⇒ *bool*
    and *actHn* :: *cnf* ⇒ *'nid set*
    and *actDn* :: *cnf* ⇒ *'nid set*
    and *PoW*:: *trace* ⇒ *nat* ⇒ *nat*
    and *hmining*:: *trace* ⇒ *nat* ⇒ *bool*
    and *dmining*:: *trace* ⇒ *nat* ⇒ *bool*
    and *cb*:: *nat*
  defines *actHn k* ≡ {*nid*. ‖*nid*‖_*k* ∧ *honest nid*}
    and *actDn k* ≡ {*nid*. ‖*nid*‖_*k* ∧ ¬ *honest nid*}
    and *PoW t n* ≡ (*LEAST x*. ∀ *nid*∈*actHn* (*t n*). *length* (*bc* (σ_*nid*(*t n*))) ≤ *x*)
    and *hmining t* ≡ (λ*n*. ∃ *nid*∈*actHn* (*t n*). *mining* (σ_*nid*(*t n*)))
    and *dmining t* ≡ (λ*n*. ∃ *nid*∈*actDn* (*t n*). *mining* (σ_*nid*(*t n*)))
  assumes *consensus*: ⋀*nid t t' bc'*::(*'nid BC*). ⟦*honest nid*⟧ ⟹ *eval nid t t' 0*
    (□_*b* ([λ*nd*. *bc'* = (*if* (∃ *b*∈*pin nd*. *length b* > *length* (*bc nd*)) *then* (*MAX* (*pin nd*)) *else* (*bc nd*))]_*b*
      ⟶^*b* ○_*b* [λ*nd*.(¬ *mining nd* ∧ *bc nd* = *bc'* ∨ *mining nd* ∧ (∃ *b*. *bc nd* = *bc'* @ [*b*]))]_*b*))
    and *attacker*: ⋀*nid t t' bc'*. ⟦¬ *honest nid*⟧ ⟹ *eval nid t t' 0*
    (□_*b* ([λ*nd*. *bc'* = (*SOME b*. *b* ∈ (*pin nd* ∪ {*bc nd*}))]_*b* ⟶^*b*
      ○_*b* [λ*nd*.(¬ *mining nd* ∧ *prefix* (*bc nd*) *bc'* ∨ *mining nd* ∧ (∃ *b*. *bc nd* = *bc'* @ [*b*]))]_*b*))
    and *forward*: ⋀*nid t t'*. *eval nid t t' 0* (□_*b* [λ*nd*. *pout nd* = *bc nd*]_*b*)
    — At each time point a node will forward its blockchain to the network
    and *init*: ⋀*nid t t'*. *eval nid t t' 0* [λ*nd*. *bc nd*=[]]_*b*
    and *conn*: ⋀*k nid*. ⟦‖*nid*‖_*k*; *honest nid*⟧
      ⟹ *pin* (*cmp nid k*) = (⋃ *nid'*∈*actHn k*. {*pout* (*cmp nid' k*)})
    and *act*: ⋀*t n*::*nat*. *finite* {*nid*::*'nid*. ‖*nid*‖_*t n*}
    and *actHn*: ⋀*t n*::*nat*. ∃ *nid*. *honest nid* ∧ ‖*nid*‖_*t n* ∧ ‖*nid*‖_*t* (*Suc n*)
    and *fair*: ⋀*n n'*. *ccard n n'* (*dmining t*) > *cb* ⟹ *ccard n n'* (*hmining t*) > *cb*
    and *closed*: ⋀*t nid b n*::*nat*. ⟦‖*nid*‖_*t n*; *b* ∈ *pin* (σ_*nid*(*t n*))⟧ ⟹ ∃ *nid'*. ‖*nid'*‖_*t n* ∧ *bc* (σ_*nid'*(*t n*))
= *b*
    and *mine*: ⋀*t nid n*::*nat*. ⟦*honest nid*; ‖*nid*‖_*t* (*Suc n*); *mining* (σ_*nid*(*t* (*Suc n*)))⟧ ⟹ ‖*nid*‖_*t n*
begin

lemma *init-model*:
  assumes ¬ (∃ *n'*. *latestAct-cond nid t n n'*)
    and ‖*nid*‖_*t n*
  shows *bc* (σ_*nid*t *n*) = []
proof −
  from *assms*(*2*) **have** ∃ *i*≥*0*. ‖*nid*‖_*t i* **by** *auto*
  with *init* **have** *bc* (σ_*nid*t ⟨*nid* → *t*⟩_*0*) = [] **using** *baEA*[*of 0 nid t*] **by** *blast*
  moreover from *assms* **have** *n*=⟨*nid* → *t*⟩_*0* **using** *nxtAct-eq* **by** *simp*
  ultimately show *?thesis* **by** *simp*
qed

lemma *fwd-bc*:
  fixes *nid* **and** *t*::*nat* ⇒ *cnf* **and** *t'*::*nat* ⇒ *'ND*
  assumes ‖*nid*‖_*t n*
  shows *pout* (σ_*nid*t *n*) = *bc* (σ_*nid*t *n*)
  using *assms forward globEANow*[*THEN baEANow*[*of nid t t' n*]] **by** *blast*

lemma *finite-input*:
fixes t n *nid*
assumes $\|nid\|_{t\ n}$
defines *dep nid'* $\equiv$ *pout* $(\sigma_{nid'}(t\ n))$
shows *finite* (*pin* (*cmp nid* $(t\ n)$))
proof −
have *finite* $\{nid'.\ \|nid'\|_{t\ n}\}$ **using** *act* **by** *auto*
moreover have *pin* (*cmp nid* $(t\ n)) \subseteq$ *dep* ‘ $\{nid'.\ \|nid'\|_{t\ n}\}$
proof
fix x **assume** $x \in$ *pin* (*cmp nid* $(t\ n)$)
show $x \in$ *dep* ‘ $\{nid'.\ \|nid'\|_{t\ n}\}$
proof −
from *assms* **obtain** *nid'* **where** $\|nid'\|_{t\ n}$ **and** *bc* $(\sigma_{nid'}(t\ n)) = x$
using *closed* ‹$x \in$ *pin* (*cmp nid* $(t\ n))$› **by** *blast*
hence *pout* $(\sigma_{nid'}(t\ n)) = x$ **using** *fwd-bc* **by** *auto*
hence *x*=*dep nid'* **using** *dep-def* **by** *simp*
moreover from ‹$\|nid'\|_{t\ n}$› **have** *nid'* $\in \{nid'.\ \|nid'\|_{t\ n}\}$ **by** *simp*
ultimately show *?thesis* **using** *image-eqI* **by** *simp*
qed
qed
ultimately show *?thesis* **using** *finite-surj* **by** *metis*
qed

lemma *nempty-input*:
fixes t n *nid*
assumes $\|nid\|_{t\ n}$
and *honest nid*
shows *pin* (*cmp nid* $(t\ n))\neq\{\}$ **using** *conn*[*of nid t n*] *act assms actHn-def* **by** *auto*

lemma *onlyone*:
assumes $\exists\, n'\geq n.\ \|tid\|_{t\ n'}$
and $\exists\, n'<n.\ \|tid\|_{t\ n'}$
shows $\exists!i.\ \langle tid \leftarrow t\rangle_n \leq i \wedge i < \langle tid \rightarrow t\rangle_n \wedge \|tid\|_{t\ i}$
proof
show $\langle tid \leftarrow t\rangle_n \leq \langle tid \leftarrow t\rangle_n \wedge \langle tid \leftarrow t\rangle_n < \langle tid \rightarrow t\rangle_n \wedge \|tid\|_{t\ \langle tid \leftarrow t\rangle_n}$
by (*metis assms dynamic-component.nxtActI latestAct-prop*(*1*) *latestAct-prop*(*2*) *less-le-trans order-refl*)
next
fix i
show $\langle tid \leftarrow t\rangle_n \leq i \wedge i < \langle tid \rightarrow t\rangle_n \wedge \|tid\|_{t\ i} \Longrightarrow i = \langle tid \leftarrow t\rangle_n$
by (*metis latestActless*(*1*) *leI le-less-Suc-eq le-less-trans nxtActI order-refl*)
qed

6.2.1 Component Behavior

lemma *bhv-hn-ex*:
fixes t **and** t'::*nat* ⇒ *'ND* **and** *tid*
assumes *honest tid*
and $\exists\, n'\geq n.\ \|tid\|_{t\ n'}$
and $\exists\, n'<n.\ \|tid\|_{t\ n'}$
and $\exists\, b\in$*pin* $(\sigma_{tid}t\ \langle tid \leftarrow t\rangle_n).$ *length* $b >$ *length* $(bc\ (\sigma_{tid}t\ \langle tid \leftarrow t\rangle_n))$
shows ¬ *mining* $(\sigma_{tid}t\ \langle tid \rightarrow t\rangle_n) \wedge bc\ (\sigma_{tid}t\ \langle tid \rightarrow t\rangle_n) =$
Blockchain.MAX (*pin* $(\sigma_{tid}t\ \langle tid \leftarrow t\rangle_n)) \vee$ *mining* $(\sigma_{tid}t\ \langle tid \rightarrow t\rangle_n) \wedge$
$(\exists\, b.\ bc\ (\sigma_{tid}t\ \langle tid \rightarrow t\rangle_n) =$ *Blockchain.MAX* (*pin* $(\sigma_{tid}t\ \langle tid \leftarrow t\rangle_n))$ @ $[b])$
proof −
let *?cond* = $\lambda nd.$ *MAX* (*pin* $(\sigma_{tid}t\ \langle tid \leftarrow t\rangle_n)) =$

(*if* (∃ *b*∈*pin nd*. *length b* > *length* (*bc nd*)) *then* (*MAX* (*pin nd*)) *else* (*bc nd*))
let *?check* = λ*nd*. ¬ *mining nd* ∧ *bc nd* = *MAX* (*pin* (σ_{tid}*t* ⟨*tid* ← *t*⟩*n*)) ∨ *mining nd* ∧
(∃ *b*. *bc nd* = *MAX* (*pin* (σ_{tid}*t* ⟨*tid* ← *t*⟩*n*)) @ [*b*])
from ‹*honest tid*› **have** *eval tid t t′ 0* (□*b*([*?cond*]*b* ⟶*b* ○*b* [*?check*]*b*))
using *consensus*[*of tid - - MAX* (*pin* (σ_{tid}*t* ⟨*tid* ← *t*⟩*n*))] **by** *simp*
moreover from *assms* **have** ∃ *i*≥*0*. ‖*tid*‖*t i* **by** *auto*
moreover have ⟨*tid* ⇐ *t*⟩*0* ≤ ⟨*tid* ← *t*⟩*n* **by** *simp*
ultimately have *eval tid t t′* ⟨*tid* ← *t*⟩*n* ([*?cond*]*b* ⟶*b* ○*b* [*?check*]*b*)
using *globEA*[*of 0 tid t t′* ([*?cond*]*b* ⟶*b* ○*b* [*?check*]*b*) ⟨*tid* ← *t*⟩*n*] **by** *fastforce*
moreover have *eval tid t t′* ⟨*tid* ← *t*⟩*n* [*?cond*]*b*
proof (*rule baIA*)
from ‹∃ *n′*<*n*. ‖*tid*‖*t n′*› **show** ∃ *i*≥⟨*tid* ← *t*⟩*n*. ‖*tid*‖*t i* **using** *latestAct-prop*(*1*) **by** *blast*
from *assms*(*3*) *assms*(*4*) **show** *?cond* (σ_{tid}*t* ⟨*tid* → *t*⟩⟨*tid* ← *t*⟩*n*) **using** *latestActNxt* **by** *simp*
qed
ultimately have *eval tid t t′* ⟨*tid* ← *t*⟩*n* (○*b* [*?check*]*b*)
using *impE*[*of tid t t′ -* [*?cond*]*b* ○*b* [*?check*]*b*] **by** *simp*
moreover have ∃ *i*>⟨*tid* → *t*⟩⟨*tid* ← *t*⟩*n*. ‖*tid*‖*t i*
proof −
from *assms* **have** ⟨*tid* → *t*⟩*n*>⟨*tid* ← *t*⟩*n* **using** *latestActNxtAct* **by** *simp*
with *assms*(*3*) **have** ⟨*tid* → *t*⟩*n*>⟨*tid* → *t*⟩⟨*tid* ← *t*⟩*n* **using** *latestActNxt* **by** *simp*
moreover from ‹∃ *n′*≥*n*. ‖*tid*‖*t n′*› **have** ‖*tid*‖*t* ⟨*tid* → *t*⟩*n* **using** *nxtActI* **by** *simp*
ultimately show *?thesis* **by** *auto*
qed
moreover from *assms* **have** ⟨*tid* ← *t*⟩*n* ≤ ⟨*tid* → *t*⟩*n*
using *latestActNxtAct* **by** (*simp add*: *order.strict-implies-order*)
moreover from *assms* **have** ∃!*i*. ⟨*tid* ← *t*⟩*n* ≤ *i* ∧ *i* < ⟨*tid* → *t*⟩*n* ∧ ‖*tid*‖*t i*
using *onlyone* **by** *simp*
ultimately have *eval tid t t′* ⟨*tid* → *t*⟩*n* [*?check*]*b*
using *nxtEA1*[*of tid t* ⟨*tid* ← *t*⟩*n t′* [*?check*]*b* ⟨*tid* → *t*⟩*n*] **by** *simp*
moreover from ‹∃ *n′*≥*n*. ‖*tid*‖*t n′*› **have** ‖*tid*‖*t* ⟨*tid* → *t*⟩*n* **using** *nxtActI* **by** *simp*
ultimately show *?thesis* **using** *baEANow*[*of tid t t′* ⟨*tid* → *t*⟩*n ?check*] **by** *simp*
qed

lemma *bhv-hn-in*:
fixes *t* **and** *t′*::*nat* ⇒ *'ND* **and** *tid*
assumes *honest tid*
and ∃ *n′*≥*n*. ‖*tid*‖*t n′*
and ∃ *n′*<*n*. ‖*tid*‖*t n′*
and ¬ (∃ *b*∈*pin* (σ_{tid}*t* ⟨*tid* ← *t*⟩*n*). *length b* > *length* (*bc* (σ_{tid}*t* ⟨*tid* ← *t*⟩*n*)))
shows ¬ *mining* (σ_{tid}*t* ⟨*tid* → *t*⟩*n*) ∧ *bc* (σ_{tid}*t* ⟨*tid* → *t*⟩*n*) = *bc* (σ_{tid}*t* ⟨*tid* ← *t*⟩*n*) ∨
mining (σ_{tid}*t* ⟨*tid* → *t*⟩*n*) ∧ (∃ *b*. *bc* (σ_{tid}*t* ⟨*tid* → *t*⟩*n*) = *bc* (σ_{tid}*t* ⟨*tid* ← *t*⟩*n*) @ [*b*])
proof −
let *?cond* = λ*nd*. *bc* (σ_{tid}*t* ⟨*tid* ← *t*⟩*n*) = (*if* (∃ *b*∈*pin nd*. *length b* > *length* (*bc nd*)) *then* (*MAX* (*pin nd*)) *else* (*bc nd*))
let *?check* = λ*nd*. ¬ *mining nd* ∧ *bc nd* = *bc* (σ_{tid}*t* ⟨*tid* ← *t*⟩*n*) ∨ *mining nd* ∧ (∃ *b*. *bc nd* = *bc* (σ_{tid}*t* ⟨*tid* ← *t*⟩*n*) @ [*b*])
from ‹*honest tid*› **have** *eval tid t t′ 0* ((□*b*([*?cond*]*b* ⟶*b* ○*b* [*?check*]*b*)))
using *consensus*[*of tid - - bc* (σ_{tid}*t* ⟨*tid* ← *t*⟩*n*)] **by** *simp*
moreover from *assms* **have** ∃ *i*≥*0*. ‖*tid*‖*t i* **by** *auto*
moreover have ⟨*tid* ⇐ *t*⟩*0* ≤ ⟨*tid* ← *t*⟩*n* **by** *simp*
ultimately have *eval tid t t′* ⟨*tid* ← *t*⟩*n* ([*?cond*]*b* ⟶*b* ○*b* [*?check*]*b*)
using *globEA*[*of 0 tid t t′* [*?cond*]*b* ⟶*b* ○*b* [*?check*]*b* ⟨*tid* ← *t*⟩*n*] **by** *fastforce*
moreover have *eval tid t t′* ⟨*tid* ← *t*⟩*n* [*?cond*]*b*
proof (*rule baIA*)
from ‹∃ *n′*<*n*. ‖*tid*‖*t n′*› **show** ∃ *i*≥⟨*tid* ← *t*⟩*n*. ‖*tid*‖*t i* **using** *latestAct-prop*(*1*) **by** *blast*

from *assms(3) assms(4)* **show** *?cond* ($\sigma_{tid}t\ \langle tid \rightarrow t\rangle_{\langle tid \leftarrow t\rangle_n}$) **using** *latestActNxt* **by** *simp*
qed
ultimately have *eval tid t t'* $\langle tid \leftarrow t\rangle_n$ ($\bigcirc_b$ [*?check*]$_b$)
using *impE*[*of tid t t'* - [*?cond*]$_b$ $\bigcirc_b$ [*?check*]$_b$] **by** *simp*
moreover have $\exists\, i{>}\langle tid \rightarrow t\rangle_{\langle tid \leftarrow t\rangle_n}.\ \|tid\|_{t\ i}$
proof −
from *assms* **have** $\langle tid \rightarrow t\rangle_n{>}\langle tid \leftarrow t\rangle_n$ **using** *latestActNxtAct* **by** *simp*
with *assms(3)* **have** $\langle tid \rightarrow t\rangle_n{>}\langle tid \rightarrow t\rangle_{\langle tid \leftarrow t\rangle_n}$ **using** *latestActNxt* **by** *simp*
moreover from $\langle\exists\, n'{\geq}n.\ \|tid\|_{t\ n'}\rangle$ **have** $\|tid\|_{t\ \langle tid \rightarrow t\rangle_n}$ **using** *nxtActI* **by** *simp*
ultimately show *?thesis* **by** *auto*
qed
moreover from *assms* **have** $\langle tid \leftarrow t\rangle_n \leq \langle tid \rightarrow t\rangle_n$
using *latestActNxtAct* **by** (*simp add*: *order.strict-implies-order*)
moreover from *assms* **have** $\exists!i.\ \langle tid \leftarrow t\rangle_n \leq i \wedge i < \langle tid \rightarrow t\rangle_n \wedge \|tid\|_{t\ i}$
using *onlyone* **by** *simp*
ultimately have *eval tid t t'* $\langle tid \rightarrow t\rangle_n$ [*?check*]$_b$
using *nxtEA1*[*of tid t* $\langle tid \leftarrow t\rangle_n$ *t'* [*?check*]$_b$ $\langle tid \rightarrow t\rangle_n$] **by** *simp*
moreover from $\langle\exists\, n'{\geq}n.\ \|tid\|_{t\ n'}\rangle$ **have** $\|tid\|_{t\ \langle tid \rightarrow t\rangle_n}$ **using** *nxtActI* **by** *simp*
ultimately show *?thesis* **using** *baEANow*[*of tid t t'* $\langle tid \rightarrow t\rangle_n$ *?check*] **by** *simp*
qed

lemma *bhv-hn-context*:
assumes *honest tid*
and $\|tid\|_{t\ n}$
and $\exists\, n'{<}n.\ \|tid\|_{t\ n'}$
shows $\exists\, nid'.\ \|nid'\|_{t\ \langle tid \leftarrow t\rangle_n} \wedge$ (*mining* ($\sigma_{tid}t\ n$) $\wedge$ ($\exists\, b.\ bc\ (\sigma_{tid}t\ n) = bc\ (\sigma_{nid'}t\ \langle tid \leftarrow t\rangle_n)$ @ [b]) $\vee$
$\neg$ *mining* ($\sigma_{tid}t\ n$) $\wedge$ *bc* ($\sigma_{tid}t\ n$) = *bc* ($\sigma_{nid'}t\ \langle tid \leftarrow t\rangle_n$))
proof *cases*
assume *casmp*: $\exists\, b{\in}pin\ (\sigma_{tid}t\ \langle tid \leftarrow t\rangle_n).\ length\ b > length\ (bc\ (\sigma_{tid}t\ \langle tid \leftarrow t\rangle_n))$
moreover from *assms(2)* **have** $\exists\, n'{\geq}n.\ \|tid\|_{t\ n'}$ **by** *auto*
moreover from *assms(3)* **have** $\exists\, n'{<}n.\ \|tid\|_{t\ n'}$ **by** *auto*
ultimately have $\neg$ *mining* ($\sigma_{tid}t\ \langle tid \rightarrow t\rangle_n$) $\wedge$ *bc* ($\sigma_{tid}t\ \langle tid \rightarrow t\rangle_n$) = *Blockchain.MAX* (*pin* ($\sigma_{tid}t$ $\langle tid \leftarrow t\rangle_n$)) $\vee$
mining ($\sigma_{tid}t\ \langle tid \rightarrow t\rangle_n$) $\wedge$ ($\exists\, b.$ *bc* ($\sigma_{tid}t\ \langle tid \rightarrow t\rangle_n$) = *Blockchain.MAX* (*pin* ($\sigma_{tid}t\ \langle tid \leftarrow t\rangle_n$)) @ [$b$])
using *assms(1) bhv-hn-ex* **by** *auto*
moreover from *assms(2)* **have** $\langle tid \rightarrow t\rangle_n = n$ **using** *nxtAct-active* **by** *simp*
ultimately have $\neg$ *mining* ($\sigma_{tid}t\ \langle tid \rightarrow t\rangle_n$) $\wedge$ *bc* ($\sigma_{tid}t\ n$) = *Blockchain.MAX* (*pin* ($\sigma_{tid}t$ $\langle tid \leftarrow t\rangle_n$)) $\vee$
mining ($\sigma_{tid}t\ \langle tid \rightarrow t\rangle_n$) $\wedge$ ($\exists\, b.$ *bc* ($\sigma_{tid}t\ n$) = *Blockchain.MAX* (*pin* ($\sigma_{tid}t\ \langle tid \leftarrow t\rangle_n$)) @ [$b$]) **by** *simp*
moreover from *assms(2)* **have** $\langle tid \rightarrow t\rangle_n = n$ **using** *nxtAct-active* **by** *simp*
ultimately have $\neg$ *mining* ($\sigma_{tid}t\ n$) $\wedge$ *bc* ($\sigma_{tid}t\ n$) = *Blockchain.MAX* (*pin* ($\sigma_{tid}t\ \langle tid \leftarrow t\rangle_n$)) $\vee$
mining ($\sigma_{tid}t\ n$) $\wedge$ ($\exists\, b.$ *bc* ($\sigma_{tid}t\ n$) = *Blockchain.MAX* (*pin* ($\sigma_{tid}t\ \langle tid \leftarrow t\rangle_n$)) @ [$b$]) **by** *simp*
moreover have *Blockchain.MAX* (*pin* ($\sigma_{tid}t\ \langle tid \leftarrow t\rangle_n$)) $\in$ *pin* ($\sigma_{tid}t\ \langle tid \leftarrow t\rangle_n$)
proof −
from $\langle\exists\, n'{<}n.\ \|tid\|_{t\ n'}\rangle$ **have** $\|tid\|_{t\ \langle tid \leftarrow t\rangle_n}$ **using** *latestAct-prop(1)* **by** *simp*
hence *finite* (*pin* (σ_{tid}($t\ \langle tid \leftarrow t\rangle_n$))) **using** *finite-input*[*of tid t* $\langle tid \leftarrow t\rangle_n$] **by** *simp*
moreover from *casmp* **obtain** b **where** $b \in pin\ (\sigma_{tid}t\ \langle tid \leftarrow t\rangle_n)$ **and** $length\ b > length\ (bc\ (\sigma_{tid}t$ $\langle tid \leftarrow t\rangle_n))$ **by** *auto*
ultimately show *?thesis* **using** *max-prop(1)* **by** *auto*
qed
with $\langle\exists\, n'{<}n.\ \|tid\|_{t\ n'}\rangle$ **obtain** *nid* **where** $\|nid\|_{t\ \langle tid \leftarrow t\rangle n}$

and $bc\ (\sigma_{nid} t\ \langle tid \leftarrow t\rangle_n) = Blockchain.MAX\ (pin\ (\sigma_{tid} t\ \langle tid \leftarrow t\rangle_n))$ **using**
$closed[of\ tid\ t\ \langle tid \leftarrow t\rangle_n\ MAX\ (pin\ (\sigma_{tid} t\ \langle tid \leftarrow t\rangle_n))]\ latestAct\text{-}prop(1)$ **by** *auto*
ultimately show *?thesis* **by** *auto*
next
assume $\neg\ (\exists\, b{\in}pin\ (\sigma_{tid} t\ \langle tid \leftarrow t\rangle_n).\ length\ b > length\ (bc\ (\sigma_{tid} t\ \langle tid \leftarrow t\rangle_n)))$
moreover from $assms(2)$ **have** $\exists\, n'{\geq}n.\ \|tid\|_{t\ n'}$ **by** *auto*
moreover from $assms(3)$ **have** $\exists\, n'{<}n.\ \|tid\|_{t\ n'}$ **by** *auto*
ultimately have $\neg\ mining\ (\sigma_{tid} t\ \langle tid \rightarrow t\rangle_n) \wedge bc\ (\sigma_{tid} t\ \langle tid \rightarrow t\rangle_n) = bc\ (\sigma_{tid} t\ \langle tid \leftarrow t\rangle_n) \vee$
$mining\ (\sigma_{tid} t\ \langle tid \rightarrow t\rangle_n) \wedge (\exists\, b.\ bc\ (\sigma_{tid} t\ \langle tid \rightarrow t\rangle_n) = bc\ (\sigma_{tid} t\ \langle tid \leftarrow t\rangle_n)\ @\ [b])$
using $assms(1)\ bhv\text{-}hn\text{-}in[of\ tid\ n\ t]$ **by** *auto*
moreover from $assms(2)$ **have** $\langle tid \rightarrow t\rangle_n = n$ **using** $nxtAct\text{-}active$ **by** *simp*
ultimately have $\neg\ mining\ (\sigma_{tid} t\ n) \wedge bc\ (\sigma_{tid} t\ n) = bc\ (\sigma_{tid} t\ \langle tid \leftarrow t\rangle_n) \vee$
$mining\ (\sigma_{tid} t\ n) \wedge (\exists\, b.\ bc\ (\sigma_{tid} t\ n) = bc\ (\sigma_{tid} t\ \langle tid \leftarrow t\rangle_n)\ @\ [b])$ **by** *simp*
moreover from $\langle \exists\, n'.\ latestAct\text{-}cond\ tid\ t\ n\ n' \rangle$ **have** $\|tid\|_{t\ \langle tid \leftarrow t\rangle_n}$
using $latestAct\text{-}prop(1)$ **by** *simp*
ultimately show *?thesis* **by** *auto*
qed

lemma *bhv-dn*:
fixes t **and** $t'{::}nat \Rightarrow 'ND$ **and** uid
assumes $\neg\ honest\ uid$
and $\exists\, n'{\geq}n.\ \|uid\|_{t\ n'}$
and $\exists\, n'{<}n.\ \|uid\|_{t\ n'}$
shows $\neg\ mining\ (\sigma_{uid} t\ \langle uid \rightarrow t\rangle_n) \wedge prefix\ (bc\ (\sigma_{uid} t\ \langle uid \rightarrow t\rangle_n))\ (SOME\ b.\ b \in pin\ (\sigma_{uid} t\ \langle uid \leftarrow t\rangle_n) \cup \{bc\ (\sigma_{uid} t\ \langle uid \leftarrow t\rangle_n)\})$
$\vee\ mining\ (\sigma_{uid} t\ \langle uid \rightarrow t\rangle_n) \wedge (\exists\, b.\ bc\ (\sigma_{uid} t\ \langle uid \rightarrow t\rangle_n) = (SOME\ b.\ b \in pin\ (\sigma_{uid} t\ \langle uid \leftarrow t\rangle_n) \cup \{bc\ (\sigma_{uid} t\ \langle uid \leftarrow t\rangle_n)\})\ @\ [b])$
proof −
let $?cond = \lambda nd.\ (SOME\ b.\ b \in (pin\ (\sigma_{uid} t\ \langle uid \leftarrow t\rangle_n) \cup \{bc\ (\sigma_{uid} t\ \langle uid \leftarrow t\rangle_n)\})) = (SOME\ b.\ b \in pin\ nd \cup \{bc\ nd\})$
let $?check = \lambda nd.\ \neg\ mining\ nd \wedge prefix\ (bc\ nd)\ (SOME\ b.\ b \in pin\ (\sigma_{uid} t\ \langle uid \leftarrow t\rangle_n) \cup \{bc\ (\sigma_{uid} t\ \langle uid \leftarrow t\rangle_n)\})$
$\vee\ mining\ nd \wedge (\exists\, b.\ bc\ nd = (SOME\ b.\ b \in pin\ (\sigma_{uid} t\ \langle uid \leftarrow t\rangle_n) \cup \{bc\ (\sigma_{uid} t\ \langle uid \leftarrow t\rangle_n)\})\ @\ [b])$
from $\langle \neg\ honest\ uid \rangle$ **have** $eval\ uid\ t\ t'\ 0\ ((\Box_b([?cond]_b \longrightarrow^b \bigcirc_b\ [?check]_b)))$
using $attacker[of\ uid\ -\ -\ (SOME\ b.\ b \in pin\ (\sigma_{uid} t\ \langle uid \leftarrow t\rangle_n) \cup \{bc\ (\sigma_{uid} t\ \langle uid \leftarrow t\rangle_n)\})]$
by *simp*
moreover from *assms* **have** $\exists\, i{\geq}0.\ \|uid\|_{t\ i}$ **by** *auto*
moreover have $\langle uid \Leftarrow t\rangle_0 \leq \langle uid \leftarrow t\rangle_n$ **by** *simp*
ultimately have $eval\ uid\ t\ t'\ \langle uid \leftarrow t\rangle_n\ ([?cond]_b \longrightarrow^b \bigcirc_b [?check]_b)$
using $globEA[of\ 0\ uid\ t\ t'\ ([?cond]_b \longrightarrow^b \bigcirc_b [?check]_b)\ \langle uid \leftarrow t\rangle_n]$ **by** *fastforce*
moreover have $eval\ uid\ t\ t'\ \langle uid \leftarrow t\rangle_n\ [?cond]_b$
proof $(rule\ baIA)$
from $\langle \exists\, n'{<}n.\ \|uid\|_{t\ n'} \rangle$ **show** $\exists\, i{\geq}\langle uid \leftarrow t\rangle_n.\ \|uid\|_{t\ i}$ **using** $latestAct\text{-}prop(1)$ **by** *blast*
with $assms(3)$ **show** $?cond\ (\sigma_{uid} t\ \langle uid \rightarrow t\rangle_{\langle uid \leftarrow t\rangle_n})$ **using** $latestActNxt$ **by** *simp*
qed
ultimately have $eval\ uid\ t\ t'\ \langle uid \leftarrow t\rangle_n\ (\bigcirc_b\ [?check]_b)$
using $impE[of\ uid\ t\ t'\ -\ [?cond]_b\ \bigcirc_b\ [?check]_b]$ **by** *simp*
moreover have $\exists\, i{>}\langle uid \rightarrow t\rangle_{\langle uid \leftarrow t\rangle_n}.\ \|uid\|_{t\ i}$
proof −
from *assms* **have** $\langle uid \rightarrow t\rangle_n{>}\langle uid \leftarrow t\rangle_n$ **using** $latestActNxtAct$ **by** *simp*
with $assms(3)$ **have** $\langle uid \rightarrow t\rangle_n{>}\langle uid \rightarrow t\rangle_{\langle uid \leftarrow t\rangle_n}$ **using** $latestActNxt$ **by** *simp*
moreover from $\langle \exists\, n'{\geq}n.\ \|uid\|_{t\ n'} \rangle$ **have** $\|uid\|_{t\ \langle uid \rightarrow t\rangle_n}$ **using** $nxtActI$ **by** *simp*
ultimately show *?thesis* **by** *auto*
qed

moreover from *assms* **have** $\langle uid \leftarrow t\rangle_n \leq \langle uid \rightarrow t\rangle_n$
using *latestActNxtAct* **by** (*simp add*: *order.strict-implies-order*)
moreover from *assms* **have** $\exists!i.\ \langle uid \leftarrow t\rangle_n \leq i \wedge i < \langle uid \rightarrow t\rangle_n \wedge \|uid\|_{t\ i}$
using *onlyone* **by** *simp*
ultimately have *eval uid t t'* $\langle uid \rightarrow t\rangle_n$ $[?check]_b$
using *nxtEA1*[*of uid t* $\langle uid \leftarrow t\rangle_n$ *t'* $[?check]_b$ $\langle uid \rightarrow t\rangle_n$] **by** *simp*
moreover from $\langle\exists\, n'{\geq}n.\ \|uid\|_{t\ n'}\rangle$ **have** $\|uid\|_{t\ \langle uid \rightarrow t\rangle_n}$ **using** *nxtActI* **by** *simp*
ultimately show *?thesis* **using** *baEANow*[*of uid t t'* $\langle uid \rightarrow t\rangle_n$ *?check*] **by** *simp*
qed

lemma *bhv-dn-context*:
assumes $\neg$ *honest uid*
and $\|uid\|_{t\ n}$
and $\exists\, n'{<}n.\ \|uid\|_{t\ n'}$
shows $\exists\, nid'.\ \|nid'\|_{t\ \langle uid \leftarrow t\rangle_n} \wedge (mining\ (\sigma_{uid}t\ n) \wedge (\exists\, b.\ prefix\ (bc\ (\sigma_{uid}t\ n))\ (bc\ (\sigma_{nid'}t\ \langle uid \leftarrow t\rangle_n)\ @\ [b]))$
$\vee \neg\ mining\ (\sigma_{uid}t\ n) \wedge prefix\ (bc\ (\sigma_{uid}t\ n))\ (bc\ (\sigma_{nid'}t\ \langle uid \leftarrow t\rangle_n)))$
proof −
let *?bc=SOME b.* $b \in pin\ (\sigma_{uid}t\ \langle uid \leftarrow t\rangle_n) \cup \{bc\ (\sigma_{uid}t\ \langle uid \leftarrow t\rangle_n)\}$
have *bc-ex*: $?bc \in pin\ (\sigma_{uid}t\ \langle uid \leftarrow t\rangle_n) \vee ?bc \in \{bc\ (\sigma_{uid}t\ \langle uid \leftarrow t\rangle_n)\}$
proof −
have $\exists\, b.\ b{\in}\ pin\ (\sigma_{uid}t\ \langle uid \leftarrow t\rangle_n) \cup \{bc\ (\sigma_{uid}t\ \langle uid \leftarrow t\rangle_n)\}$ **by** *auto*
hence $?bc \in pin\ (\sigma_{uid}t\ \langle uid \leftarrow t\rangle_n) \cup \{bc\ (\sigma_{uid}t\ \langle uid \leftarrow t\rangle_n)\}$ **using** *someI-ex* **by** *simp*
thus *?thesis* **by** *auto*
qed

from *assms(2)* **have** $\exists\, n'{\geq}n.\ \|uid\|_{t\ n'}$ **by** *auto*
moreover from *assms(3)* **have** $\exists\, n'{<}n.\ \|uid\|_{t\ n'}$ **by** *auto*
ultimately have $\neg\ mining\ (\sigma_{uid}t\ \langle uid \rightarrow t\rangle_n) \wedge prefix\ (bc\ (\sigma_{uid}t\ \langle uid \rightarrow t\rangle_n))\ ?bc \vee$
mining $(\sigma_{uid}t\ \langle uid \rightarrow t\rangle_n) \wedge (\exists\, b.\ bc\ (\sigma_{uid}t\ \langle uid \rightarrow t\rangle_n) = ?bc\ @\ [b])$
using *bhv-dn*[*of uid n t*] *assms(1)* **by** *simp*
moreover from *assms(2)* **have** $\langle uid \rightarrow t\rangle_n = n$ **using** *nxtAct-active* **by** *simp*
ultimately have *casmp*: $\neg\ mining\ (\sigma_{uid}t\ n) \wedge prefix\ (bc\ (\sigma_{uid}t\ n))\ ?bc \vee$
mining $(\sigma_{uid}t\ n) \wedge (\exists\, b.\ bc\ (\sigma_{uid}t\ n) = ?bc\ @\ [b])$ **by** *simp*

from *bc-ex* **have** $?bc \in pin\ (\sigma_{uid}t\ \langle uid \leftarrow t\rangle_n) \vee ?bc \in \{bc\ (\sigma_{uid}t\ \langle uid \leftarrow t\rangle_n)\}$ **.**
thus *?thesis*
proof
assume $?bc \in pin\ (\sigma_{uid}t\ \langle uid \leftarrow t\rangle_n)$
moreover from $\langle\exists\, n'{<}n.\ \|uid\|_{t\ n'}\rangle$ **have** $\|uid\|_{t\ \langle uid \leftarrow t\rangle_n}$ **using** *latestAct-prop(1)* **by** *simp*
ultimately obtain *nid* **where** $\|nid\|_{t\ \langle uid \leftarrow t\rangle_n}$ **and** $bc\ (\sigma_{nid}t\ \langle uid \leftarrow t\rangle_n) = ?bc$
using *closed* **by** *blast*
with *casmp* **have** $\neg\ mining\ (\sigma_{uid}t\ n) \wedge prefix\ (bc\ (\sigma_{uid}t\ n))\ (bc\ (\sigma_{nid}t\ \langle uid \leftarrow t\rangle_n)) \vee$
mining $(\sigma_{uid}t\ n) \wedge (\exists\, b.\ bc\ (\sigma_{uid}t\ n) = (bc\ (\sigma_{nid}t\ \langle uid \leftarrow t\rangle_n))\ @\ [b])$ **by** *simp*
with $\langle\|nid\|_{t\ \langle uid \leftarrow t\rangle_n}\rangle$ **show** *?thesis* **by** *auto*
next
assume $?bc \in \{bc\ (\sigma_{uid}t\ \langle uid \leftarrow t\rangle_n)\}$
hence $?bc = bc\ (\sigma_{uid}t\ \langle uid \leftarrow t\rangle_n)$ **by** *simp*
moreover from $\langle\exists\, n'.\ latestAct\text{-}cond\ uid\ t\ n\ n'\rangle$ **have** $\|uid\|_{t\ \langle uid \leftarrow t\rangle_n}$
using *latestAct-prop(1)* **by** *simp*
ultimately show *?thesis* **using** *casmp* **by** *auto*
qed
qed

6.2.2 Maximal Honest Blockchains

abbreviation *mbc-cond*:: *trace* ⇒ *nat* ⇒ *'nid* ⇒ *bool*
where *mbc-cond t n nid* ≡ *nid*∈*actHn* (*t n*) ∧ (∀ *nid'*∈*actHn* (*t n*). *length* (*bc* ($\sigma_{nid'}$(*t n*))) ≤ *length* (*bc* (σ_{nid}(*t n*))))

lemma *mbc-ex*:
fixes *t n*
shows ∃ *x*. *mbc-cond t n x*
proof −
let *?ALL*={*b*. ∃ *nid*∈*actHn* (*t n*). *b* = *bc* (σ_{nid}(*t n*))}
have *MAX ?ALL* ∈ *?ALL*
proof (*rule max-prop*)
from *actHn* **have** *actHn* (*t n*) ≠ {} **using** *actHn-def* **by** *blast*
thus *?ALL*≠{} **by** *auto*
from *act* **have** *finite* (*actHn* (*t n*)) **using** *actHn-def* **by** *simp*
thus *finite ?ALL* **by** *simp*
qed
then obtain *nid* **where** *nid* ∈ *actHn* (*t n*) ∧ *bc* (σ_{nid}(*t n*)) = *MAX ?ALL* **by** *auto*
moreover have ∀ *nid'*∈*actHn* (*t n*). *length* (*bc* ($\sigma_{nid'}$(*t n*))) ≤ *length* (*MAX ?ALL*)
proof
fix *nid*
assume *nid* ∈ *actHn* (*t n*)
hence *bc* (σ_{nid}(*t n*)) ∈ *?ALL* **by** *auto*
moreover have ∀ *b'*∈*?ALL*. *length b'* ≤ *length* (*MAX ?ALL*)
proof (*rule max-prop*)
from ‹*bc* (σ_{nid}(*t n*)) ∈ *?ALL*› **show** *?ALL*≠{} **by** *auto*
from *act* **have** *finite* (*actHn* (*t n*)) **using** *actHn-def* **by** *simp*
thus *finite ?ALL* **by** *simp*
qed
ultimately show *length* (*bc* (σ_{nid}*t n*)) ≤ *length* (*Blockchain.MAX* {*b*. ∃ *nid*∈*actHn* (*t n*). *b* = *bc* (σ_{nid}*t n*)}) **by** *simp*
qed
ultimately show *?thesis* **by** *auto*
qed

definition *MBC*:: *trace* ⇒ *nat* ⇒ *'nid*
where *MBC t n* = (*SOME b*. *mbc-cond t n b*)

lemma *mbc-prop*[*simp*]:
shows *mbc-cond t n* (*MBC t n*)
using *someI-ex*[*OF mbc-ex*] *MBC-def* **by** *simp*

6.2.3 Honest Proof of Work

An important construction is the maximal proof of work available in the honest community. The construction was already introduces in the locale itself since it was used to express some of the locale assumptions.

abbreviation *pow-cond*:: *trace* ⇒ *nat* ⇒ *nat* ⇒ *bool*
where *pow-cond t n n'* ≡ ∀ *nid*∈*actHn* (*t n*). *length* (*bc* (σ_{nid}(*t n*))) ≤ *n'*

lemma *pow-ex*:
fixes *t n*
shows *pow-cond t n* (*length* (*bc* ($\sigma_{MBC\ t\ n}$(*t n*))))
and ∀ *x'*. *pow-cond t n x'* ⟶ *x'*≥*length* (*bc* ($\sigma_{MBC\ t\ n}$(*t n*)))

using *mbc-prop* **by** *auto*

lemma *pow-prop*:
pow-cond t n (*PoW t n*)
proof −
from *pow-ex* **have** *pow-cond t n* (*LEAST x. pow-cond t n x*) **using** *LeastI-ex*[*of pow-cond t n*] **by** *blast*
thus *?thesis* **using** *PoW-def* **by** *simp*
qed

lemma *pow-eq*:
fixes *n*
assumes ∃ *tid*∈*actHn* (*t n*). *length* (*bc* (σ$_{tid}$(*t n*))) = *x*
and ∀ *tid*∈*actHn* (*t n*). *length* (*bc* (σ$_{tid}$(*t n*))) ≤ *x*
shows *PoW t n* = *x*
proof −
have (*LEAST x. pow-cond t n x*) = *x*
proof (*rule Least-equality*)
from *assms*(*2*) **show** ∀ *nid*∈*actHn* (*t n*). *length* (*bc* (σ$_{nid}$*t n*)) ≤ *x* **by** *simp*
next
fix *y*
assume ∀ *nid*∈*actHn* (*t n*). *length* (*bc* (σ$_{nid}$*t n*)) ≤ *y*
thus *x* ≤ *y* **using** *assms*(*1*) **by** *auto*
qed
with *PoW-def* **show** *?thesis* **by** *simp*
qed

lemma *pow-mbc*:
shows *length* (*bc* (σ$_{MBC\ t\ n}$*t n*)) = *PoW t n*
by (*metis mbc-prop pow-eq*)

lemma *pow-less*:
fixes *t n nid*
assumes *pow-cond t n x*
shows *PoW t n* ≤ *x*
proof −
from *pow-ex assms* **have** (*LEAST x. pow-cond t n x*) ≤ *x* **using** *Least-le*[*of pow-cond t n*] **by** *blast*
thus *?thesis* **using** *PoW-def* **by** *simp*
qed

lemma *pow-le-max*:
assumes *honest tid*
and ‖*tid*‖$_{t\ n}$
shows *PoW t n* ≤ *length* (*MAX* (*pin* (σ$_{tid}$*t n*)))
proof −
from *mbc-prop* **have** *honest* (*MBC t n*) **and** ‖*MBC t n*‖$_{t\ n}$ **using** *actHn-def* **by** *auto*
hence *pout* (σ$_{MBC\ t\ n}$*t n*) = *bc* (σ$_{MBC\ t\ n}$*t n*)
using *forward globEANow*[*THEN baEANow*[*of MBC t n t t′ n* λ*nd. pout nd* = *bc nd*]] **by** *auto*
with *assms* ‹‖*MBC t n*‖$_{t\ n}$› ‹*honest* (*MBC t n*)› **have** *bc* (σ$_{MBC\ t\ n}$*t n*) ∈ *pin* (σ$_{tid}$*t n*)
using *conn actHn-def* **by** *auto*
moreover from *assms* (*2*) **have** *finite* (*pin* (σ$_{tid}$*t n*)) **using** *finite-input*[*of tid t n*] **by** *simp*
ultimately have *length* (*bc* (σ$_{MBC\ t\ n}$*t n*)) ≤ *length* (*MAX* (*pin* (σ$_{tid}$*t n*)))
using *max-prop*(*2*) **by** *auto*
with *pow-mbc* **show** *?thesis* **by** *simp*
qed

lemma *pow-ge-lgth*:
assumes *honest tid*
and $\|tid\|_{t\ n}$
shows *length* $(bc\ (\sigma_{tid} t\ n)) \leq PoW\ t\ n$
proof −
from *assms* **have** $tid \in actHn\ (t\ n)$ **using** *actHn-def* **by** *simp*
thus *?thesis* **using** *pow-prop* **by** *simp*
qed

lemma *pow-le-lgth*:
assumes *honest tid*
and $\|tid\|_{t\ n}$
and $\neg(\exists\, b{\in}pin\ (\sigma_{tid} t\ n).\ length\ b > length\ (bc\ (\sigma_{tid} t\ n)))$
shows $length\ (bc\ (\sigma_{tid} t\ n)) \geq PoW\ t\ n$
proof −
from *assms* (*3*) **have** $\forall\, b{\in}pin\ (\sigma_{tid} t\ n).\ length\ b \leq length\ (bc\ (\sigma_{tid} t\ n))$ **by** *auto*
moreover from *assms nempty-input*[*of tid t n*] *finite-input*[*of tid t n*]
have $MAX\ (pin\ (\sigma_{tid} t\ n)) \in pin\ (\sigma_{tid} t\ n)$ **using** *max-prop*(*1*)[*of pin* $(\sigma_{tid} t\ n)$] **by** *simp*
ultimately have $length\ (MAX\ (pin\ (\sigma_{tid} t\ n))) \leq length\ (bc\ (\sigma_{tid} t\ n))$ **by** *simp*
moreover from *assms* **have** $PoW\ t\ n \leq length\ (MAX\ (pin\ (\sigma_{tid} t\ n)))$ **using** *pow-le-max* **by** *simp*
ultimately show *?thesis* **by** *simp*
qed

lemma *pow-mono*:
shows $n'{\geq}n \Longrightarrow PoW\ t\ n' \geq PoW\ t\ n$
proof (*induction* n' *rule*: *dec-induct*)
case *base*
then show *?case* **by** *simp*
next
case (*step* n')
hence $PoW\ t\ n \leq PoW\ t\ n'$ **by** *simp*
moreover have $PoW\ t\ (Suc\ n') \geq PoW\ t\ n'$
proof −
from *actHn* **obtain** *tid* **where** *honest tid* **and** $\|tid\|_{t\ n'}$ **and** $\|tid\|_{t\ (Suc\ n')}$ **by** *auto*
show *?thesis*
proof *cases*
assume $\exists\, b{\in}pin\ (\sigma_{tid} t\ n').\ length\ b > length\ (bc\ (\sigma_{tid} t\ n'))$
moreover from $\langle\|tid\|_{t\ (Suc\ n')}\rangle$ **have** $\langle tid \rightarrow t\rangle_{Suc\ n'} = Suc\ n'$
using *nxtAct-active* **by** *simp*
moreover from $\langle\|tid\|_{t\ n'}\rangle$ **have** $\langle tid \leftarrow t\rangle_{Suc\ n'} = n'$
using *latestAct-prop*(*2*) *latestActless le-less-Suc-eq* **by** *blast*
moreover from $\langle\|tid\|_{t\ n'}\rangle$ **have** $\exists\, n''{<}Suc\ n'.\ \|tid\|_{t\ n''}$ **by** *blast*
moreover from $\langle\|tid\|_{t\ (Suc\ n')}\rangle$ **have** $\exists\, n''{\geq}Suc\ n'.\ \|tid\|_{t\ n''}$ **by** *auto*
ultimately have $bc\ (\sigma_{tid} t\ (Suc\ n')) = Blockchain.MAX\ (pin\ (\sigma_{tid} t\ n')) \vee$
$(\exists\, b.\ bc\ (\sigma_{tid} t\ (Suc\ n')) = Blockchain.MAX\ (pin\ (\sigma_{tid} t\ n'))\ @\ b)$
using $\langle$*honest tid*$\rangle$ *bhv-hn-ex*[*of tid Suc n' t*] **by** *auto*
hence $length\ (bc\ (\sigma_{tid} t\ (Suc\ n'))) \geq length\ (Blockchain.MAX\ (pin\ (\sigma_{tid} t\ n')))$ **by** *auto*
moreover from $\langle$*honest tid*$\rangle$ $\langle\|tid\|_{t\ n'}\rangle$
have $length\ (Blockchain.MAX\ (pin\ (\sigma_{tid} t\ n'))) \geq PoW\ t\ n'$ **using** *pow-le-max* **by** *simp*
ultimately have $PoW\ t\ n' \leq length\ (bc\ (\sigma_{tid} t\ (Suc\ n')))$ **by** *simp*
moreover from $\langle$*honest tid*$\rangle$ $\langle\|tid\|_{t\ (Suc\ n')}\rangle$
have $length\ (bc\ (\sigma_{tid} t\ (Suc\ n'))) \leq PoW\ t\ (Suc\ n')$ **using** *pow-ge-lgth* **by** *simp*
ultimately show *?thesis* **by** *simp*
next
assume *asmp*: $\neg(\exists\, b{\in}pin\ (\sigma_{tid} t\ n').\ length\ b > length\ (bc\ (\sigma_{tid} t\ n')))$

```
    moreover from ‹‖tid‖t (Suc n')› have ⟨tid → t⟩Suc n' = Suc n'
      using nxtAct-active by simp
    moreover from ‹‖tid‖t n'› have ⟨tid ← t⟩Suc n' = n'
      using latestAct-prop(2) latestActless le-less-Suc-eq by blast
    moreover from ‹‖tid‖t n'› have ∃n''<Suc n'. ‖tid‖t n'' by blast
    moreover from ‹‖tid‖t (Suc n')› have ∃n''≥Suc n'. ‖tid‖t n'' by auto
    ultimately have bc (σtid t (Suc n')) = bc (σtid t n') ∨
      (∃b. bc (σtid t (Suc n')) = bc (σtid t n') @ b)
      using ‹honest tid› bhv-hn-in[of tid Suc n' t] by auto
    hence length (bc (σtid t (Suc n'))) ≥ length (bc (σtid t n')) by auto
    moreover from ‹honest tid› ‹‖tid‖t n'› asmp have length (bc (σtid t n')) ≥ PoW t n'
      using pow-le-lgth by simp
    moreover from ‹honest tid› ‹‖tid‖t (Suc n')›
    have length (bc (σtid t (Suc n'))) ≤ PoW t (Suc n') using pow-ge-lgth by simp
    ultimately show ?thesis by simp
  qed
 qed
 ultimately show ?case by auto
qed

lemma pow-equals:
 assumes PoW t n = PoW t n'
 and n'≥n
 and n''≥n
 and n''≤n'
shows PoW t n = PoW t n'' by (metis pow-mono assms(1) assms(3) assms(4) eq-iff)

lemma pow-mining-suc:
   assumes hmining t (Suc n)
   shows PoW t n < PoW t (Suc n)
proof −
 from assms obtain nid where nid∈actHn (t (Suc n)) and mining (σnid(t (Suc n)))
   using hmining-def by auto
 show ?thesis
 proof cases
   assume asmp: (∃b∈pin (σnid t ⟨nid ← t⟩Suc n). length b > length (bc (σnid t ⟨nid ← t⟩Suc n)))
   moreover from ‹nid∈actHn (t (Suc n))› have honest nid and ‖nid‖t (Suc n)
     using actHn-def by auto
   moreover from ‹honest nid› ‹mining (σnid(t (Suc n)))› ‹‖nid‖t (Suc n)› have ‖nid‖t n
     using mine by simp
   hence ∃n'. latestAct-cond nid t (Suc n) n' by auto
   ultimately have ¬ mining (σnid t ⟨nid → t⟩Suc n) ∧ bc (σnid t ⟨nid → t⟩Suc n) = MAX (pin (σnid t
⟨nid ← t⟩Suc n)) ∨
    mining (σnid t ⟨nid → t⟩Suc n) ∧ (∃b. bc (σnid t ⟨nid → t⟩Suc n) = MAX (pin (σnid t ⟨nid ←
t⟩Suc n)) @ [b]) using bhv-hn-ex[of nid Suc n] by auto
   moreover from ‹‖nid‖t (Suc n)› have ⟨nid → t⟩Suc n = Suc n using nxtAct-active by simp
   moreover have ⟨nid ← t⟩Suc n = n
   proof (rule latestActEq)
     from ‹‖nid‖t n› show ‖nid‖t n by simp
     show ¬ (∃n''>n. n'' < Suc n ∧ ‖nid‖t n) by simp
     show n < Suc n by simp
   qed
   hence ⟨nid ← t⟩Suc n = n using latestAct-def by simp
   ultimately have ¬ mining (σnid t (Suc n)) ∧ bc (σnid t (Suc n)) = MAX (pin (σnid t n)) ∨
```

mining ($\sigma_{nid} t\ (Suc\ n)$) $\wedge$ ($\exists\, b.\ bc\ (\sigma_{nid} t\ (Suc\ n)) = MAX\ (pin\ (\sigma_{nid} t\ n))\ @\ [b]$) **by** *simp*
with ‹*mining* ($\sigma_{nid}(t\ (Suc\ n))$)›
have $\exists\, b.\ bc\ (\sigma_{nid} t\ (Suc\ n)) = MAX\ (pin\ (\sigma_{nid} t\ n))\ @\ [b]$ **by** *auto*
moreover from ‹*honest nid*› ‹$\|nid\|_{t\ (Suc\ n)}$› **have** $length\ (bc\ (\sigma_{nid} t\ (Suc\ n))) \leq PoW\ t\ (Suc\ n)$
using *pow-ge-lgth*[*of nid t Suc n*] **by** *simp*
ultimately have $length\ (MAX\ (pin\ (\sigma_{nid} t\ n))) < PoW\ t\ (Suc\ n)$ **by** *auto*
moreover from ‹*honest nid*› ‹$\|nid\|_{t\ n}$› **have** $length\ (MAX\ (pin\ (\sigma_{nid} t\ n))) \geq PoW\ t\ n$
using *pow-le-max* **by** *simp*
ultimately show *?thesis* **by** *simp*
next
assume *asmp*: $\neg\ (\exists\, b{\in}pin\ (\sigma_{nid} t\ \langle nid \leftarrow t\rangle_{Suc\ n}).\ length\ b > length\ (bc\ (\sigma_{nid} t\ \langle nid \leftarrow t\rangle_{Suc\ n})))$
moreover from ‹*nid*∈*actHn* ($t\ (Suc\ n)$)› **have** *honest nid* **and** $\|nid\|_{t\ (Suc\ n)}$
using *actHn-def* **by** *auto*
moreover from ‹*honest nid*› ‹*mining* ($\sigma_{nid}(t\ (Suc\ n))$)› ‹$\|nid\|_{t\ (Suc\ n)}$› **have** $\|nid\|_{t\ n}$
using *mine* **by** *simp*
hence $\exists\, n'.$ *latestAct-cond nid t* ($Suc\ n$) n' **by** *auto*
ultimately have $\neg$ *mining* ($\sigma_{nid} t\ \langle nid \rightarrow t\rangle_{Suc\ n}$) $\wedge$ $bc\ (\sigma_{nid} t\ \langle nid \rightarrow t\rangle_{Suc\ n}) = bc\ (\sigma_{nid} t\ \langle nid \leftarrow t\rangle_{Suc\ n})$ $\vee$
mining ($\sigma_{nid} t\ \langle nid \rightarrow t\rangle_{Suc\ n}$) $\wedge$ ($\exists\, b.\ bc\ (\sigma_{nid} t\ \langle nid \rightarrow t\rangle_{Suc\ n}) = bc\ (\sigma_{nid} t\ \langle nid \leftarrow t\rangle_{Suc\ n})\ @\ [b]$)
using *bhv-hn-in*[*of nid Suc n*] **by** *auto*
moreover from ‹$\|nid\|_{t\ (Suc\ n)}$› **have** $\langle nid \rightarrow t\rangle_{Suc\ n} = Suc\ n$ **using** *nxtAct-active* **by** *simp*
moreover have $\langle nid \leftarrow t\rangle_{Suc\ n} = n$
proof (*rule latestActEq*)
from ‹$\|nid\|_{t\ n}$› **show** $\|nid\|_{t\ n}$ **by** *simp*
show $\neg\ (\exists\, n''{>}n.\ n'' < Suc\ n \wedge \|nid\|_{t\ n})$ **by** *simp*
show $n < Suc\ n$ **by** *simp*
qed
hence $\langle nid \leftarrow t\rangle_{Suc\ n} = n$ **using** *latestAct-def* **by** *simp*
ultimately have $\neg$ *mining* ($\sigma_{nid} t\ (Suc\ n)$) $\wedge$ $bc\ (\sigma_{nid} t\ (Suc\ n)) = bc\ (\sigma_{nid} t\ n)$ $\vee$
mining ($\sigma_{nid} t\ (Suc\ n)$) $\wedge$ ($\exists\, b.\ bc\ (\sigma_{nid} t\ (Suc\ n)) = bc\ (\sigma_{nid} t\ n)\ @\ [b]$) **by** *simp*
with ‹*mining* ($\sigma_{nid}(t\ (Suc\ n))$)› **have** $\exists\, b.\ bc\ (\sigma_{nid} t\ (Suc\ n)) = bc\ (\sigma_{nid} t\ n)\ @\ [b]$ **by** *simp*
moreover from ‹$\langle nid \leftarrow t\rangle_{Suc\ n} = n$›
have $\neg\ (\exists\, b{\in}pin\ (\sigma_{nid} t\ n).\ length\ (bc\ (\sigma_{nid} t\ n)) < length\ b)$
using *asmp* **by** *simp*
with ‹*honest nid*› ‹$\|nid\|_{t\ n}$› **have** $length\ (bc\ (\sigma_{nid} t\ n)) \geq PoW\ t\ n$
using *pow-le-lgth*[*of nid t n*] **by** *simp*
moreover from ‹*honest nid*› ‹$\|nid\|_{t\ (Suc\ n)}$› **have** $length\ (bc\ (\sigma_{nid} t\ (Suc\ n))) \leq PoW\ t\ (Suc\ n)$
using *pow-ge-lgth*[*of nid t Suc n*] **by** *simp*
ultimately show *?thesis* **by** *auto*
qed
qed

6.2.4 History

In the following we introduce an operator which extracts the development of a blockchain up to a time point n.

abbreviation *his-prop t n nid n' nid' x* ≡
($\exists\, n.$ *latestAct-cond nid' t n' n*) $\wedge$ $\|snd\ x\|_{t\ (fst\ x)}$ $\wedge$ $fst\ x = \langle nid' \leftarrow t\rangle_{n'}$ $\wedge$
(*prefix* ($bc\ (\sigma_{nid'}(t\ n'))$) ($bc\ (\sigma_{snd\ x}(t\ (fst\ x)))$) $\vee$
($\exists\, b.\ bc\ (\sigma_{nid'}(t\ n')) = (bc\ (\sigma_{snd\ x}(t\ (fst\ x))))\ @\ [b] \wedge mining\ (\sigma_{nid'}(t\ n'))$))

inductive-set
his:: *trace* ⇒ *nat* ⇒ *'nid* ⇒ (*nat* × *'nid*) *set*
for *t*::*trace* **and** *n*::*nat* **and** *nid*::*'nid*

where $[\![\|nid\|_{t\ n}]\!] \Longrightarrow (n,nid) \in his\ t\ n\ nid$
| $[\![(n',nid') \in his\ t\ n\ nid; \exists\, x.\ his\text{-}prop\ t\ n\ nid\ n'\ nid'\ x]\!] \Longrightarrow (SOME\ x.\ his\text{-}prop\ t\ n\ nid\ n'\ nid'\ x) \in his\ t\ n\ nid$

lemma *his-act*:
assumes $(n',nid') \in his\ t\ n\ nid$
shows $\|nid'\|_{t\ n'}$
using *assms*
proof (*rule his.cases*)
assume $(n',\ nid') = (n,\ nid)$ **and** $\|nid\|_{t\ n}$
thus $\|nid'\|_{t\ n'}$ **by** *simp*
next
fix $n''\ nid''$ **assume** *asmp*: $(n',\ nid') = (SOME\ x.\ his\text{-}prop\ t\ n\ nid\ n''\ nid''\ x)$
and $(n'',\ nid'') \in his\ t\ n\ nid$ **and** $\exists\, x.\ his\text{-}prop\ t\ n\ nid\ n''\ nid''\ x$
hence *his-prop t n nid n'' nid''* $(SOME\ x.\ his\text{-}prop\ t\ n\ nid\ n''\ nid''\ x)$
using *someI-ex*[*of* $\lambda x.$ *his-prop t n nid n'' nid'' x*] **by** *auto*
hence $\|snd\ (SOME\ x.\ his\text{-}prop\ t\ n\ nid\ n''\ nid''\ x)\|_{t\ (fst\ (SOME\ x.\ his\text{-}prop\ t\ n\ nid\ n''\ nid''\ x))}$
by *blast*
moreover from *asmp* **have** $fst\ (SOME\ x.\ his\text{-}prop\ t\ n\ nid\ n''\ nid''\ x) = fst\ (n',\ nid')$ **by** *simp*
moreover from *asmp* **have** $snd\ (SOME\ x.\ his\text{-}prop\ t\ n\ nid\ n''\ nid''\ x) = snd\ (n',\ nid')$ **by** *simp*
ultimately show *?thesis* **by** *simp*
qed

In addition we also introduce an operator to obtain the predecessor of a blockchains development.

definition *hisPred*
where $hisPred\ t\ n\ nid\ n' \equiv (GREATEST\ n''.\ \exists\, nid'.\ (n'',nid') \in his\ t\ n\ nid \wedge n'' < n')$

lemma *hisPrev-prop*:
assumes $\exists\, n''<n'.\ \exists\, nid'.\ (n'',nid') \in his\ t\ n\ nid$
shows *hisPred t n nid n'* $< n'$ **and** $\exists\, nid'.\ (hisPred\ t\ n\ nid\ n',nid') \in his\ t\ n\ nid$
proof −
from *assms* **obtain** n'' **where** $\exists\, nid'.\ (n'',nid') \in his\ t\ n\ nid \wedge n''<n'$ **by** *auto*
moreover from $\langle \exists\, nid'.\ (n'',nid') \in his\ t\ n\ nid \wedge n''<n' \rangle$
have $\exists\, i'\leq n'.\ (\exists\, nid'.\ (i',\ nid') \in his\ t\ n\ nid \wedge i' < n') \wedge (\forall\, n'a.\ (\exists\, nid'.\ (n'a,\ nid') \in his\ t\ n\ nid \wedge n'a < n') \longrightarrow n'a \leq i')$
using *boundedGreatest*[*of* $\lambda n''.\ \exists\, nid'.\ (n'',nid') \in his\ t\ n\ nid \wedge n'' < n'\ n''\ n'$] **by** *simp*
then obtain i' **where** $\forall\, n'a.\ (\exists\, nid'.\ (n'a,\ nid') \in his\ t\ n\ nid \wedge n'a < n') \longrightarrow n'a \leq i'$ **by** *auto*
ultimately show *hisPred t n nid n'* $< n'$ **and** $\exists\, nid'.\ (hisPred\ t\ n\ nid\ n',nid') \in his\ t\ n\ nid$
using *GreatestI-nat*[*of* $\lambda n''.\ \exists\, nid'.\ (n'',nid') \in his\ t\ n\ nid \wedge n'' < n'\ n''\ i'$] *hisPred-def* **by** *auto*
qed

lemma *hisPrev-nex-less*:
assumes $\exists\, n''<n'.\ \exists\, nid'.\ (n'',nid') \in his\ t\ n\ nid$
shows $\neg(\exists\, x \in his\ t\ n\ nid.\ fst\ x < n' \wedge fst\ x > hisPred\ t\ n\ nid\ n')$
proof (*rule ccontr*)
assume $\neg\neg(\exists\, x \in his\ t\ n\ nid.\ fst\ x < n' \wedge fst\ x > hisPred\ t\ n\ nid\ n')$
then obtain $n''\ nid''$ **where** $(n'',nid'') \in his\ t\ n\ nid$ **and** $n'' < n'$ **and** $n'' > hisPred\ t\ n\ nid\ n'$ **by** *auto*
moreover have $n'' \leq hisPred\ t\ n\ nid\ n'$
proof −
from $\langle (n'',nid'') \in his\ t\ n\ nid \rangle$ $\langle n'' < n' \rangle$ **have** $\exists\, nid'.\ (n'',nid') \in his\ t\ n\ nid \wedge n''<n'$ **by** *auto*
moreover from $\langle \exists\, nid'.\ (n'',nid') \in his\ t\ n\ nid \wedge n''<n' \rangle$ **have** $\exists\, i'\leq n'.\ (\exists\, nid'.\ (i',\ nid') \in his\ t\ n\ nid \wedge i' < n') \wedge (\forall\, n'a.\ (\exists\, nid'.\ (n'a,\ nid') \in his\ t\ n\ nid \wedge n'a < n') \longrightarrow n'a \leq i')$
using *boundedGreatest*[*of* $\lambda n''.\ \exists\, nid'.\ (n'',nid') \in his\ t\ n\ nid \wedge n'' < n'\ n''\ n'$] **by** *simp*
then obtain i' **where** $\forall\, n'a.\ (\exists\, nid'.\ (n'a,\ nid') \in his\ t\ n\ nid \wedge n'a < n') \longrightarrow n'a \leq i'$ **by** *auto*

ultimately show *?thesis* **using** *Greatest-le-nat*[*of* $\lambda n''.\ \exists\, nid'.\ (n'',nid') \in his\ t\ n\ nid \wedge n'' < n'\ n''$ *i'*] *hisPred-def* **by** *simp*
qed
ultimately show *False* **by** *simp*
qed

lemma *his-le*:
assumes $x \in his\ t\ n\ nid$
shows *fst* $x \leq n$
using *assms*
proof (*induction rule*: *his.induct*)
case *1*
then show *?case* **by** *simp*
next
case (*2* n' nid')
moreover have *fst* (*SOME x. his-prop t n nid* n' nid' x) $\leq n'$
proof −
from *2.hyps* **have** $\exists\, x.$ *his-prop t n nid* n' nid' x **by** *simp*
hence *his-prop t n nid* n' nid' (*SOME x. his-prop t n nid* n' nid' x)
using *someI-ex*[*of* $\lambda x.$ *his-prop t n nid* n' nid' x] **by** *auto*
hence *fst* (*SOME x. his-prop t n nid* n' nid' x) $= \langle nid' \leftarrow t \rangle_{n'}$ **by** *force*
moreover from ‹*his-prop t n nid* n' nid' (*SOME x. his-prop t n nid* n' nid' x)›
have $\exists\, n.$ *latestAct-cond nid'* t n' n **by** *simp*
ultimately show *?thesis* **using** *latestAct-prop*(*2*)[*of* n' nid' t] **by** *simp*
qed
ultimately show *?case* **by** *simp*
qed

lemma *his-determ-base*:
shows (n, nid') $\in$ *his t n nid* $\Longrightarrow$ nid'=nid
proof (*rule his.cases*)
assume (n, nid') = (n, nid)
thus *?thesis* **by** *simp*
next
fix n' $nid'a$
assume (n, nid') $\in$ *his t n nid* **and** (n, nid') = (*SOME x. his-prop t n nid* n' $nid'a$ x)
and (n', $nid'a$) $\in$ *his t n nid* **and** $\exists\, x.$ *his-prop t n nid* n' $nid'a$ x
hence *his-prop t n nid* n' $nid'a$ (*SOME x. his-prop t n nid* n' $nid'a$ x)
using *someI-ex*[*of* $\lambda x.$ *his-prop t n nid* n' $nid'a$ x] **by** *auto*
hence *fst* (*SOME x. his-prop t n nid* n' $nid'a$ x) $= \langle nid'a \leftarrow t \rangle_{n'}$ **by** *force*
moreover from ‹*his-prop t n nid* n' $nid'a$ (*SOME x. his-prop t n nid* n' $nid'a$ x)›
have $\exists\, n.$ *latestAct-cond nid'a t* n' n **by** *simp*
ultimately have *fst* (*SOME x. his-prop t n nid* n' $nid'a$ x) $< n'$
using *latestAct-prop*(*2*)[*of* n' $nid'a$ t] **by** *simp*
with ‹(n, nid') = (*SOME x. his-prop t n nid* n' $nid'a$ x)› **have** *fst* (n, nid')$<n'$ **by** *simp*
hence $n<n'$ **by** *simp*
moreover from ‹(n', $nid'a$) $\in$ *his t n nid*› **have** $n' \leq n$ **using** *his-le* **by** *auto*
ultimately show $nid' = nid$ **by** *simp*
qed

lemma *hisPrev-same*:
assumes $\exists\, n'<n''.\ \exists\, nid'.\ (n',nid') \in his\ t\ n\ nid$
and $\exists\, n''<n'.\ \exists\, nid'.\ (n'',nid') \in his\ t\ n\ nid$
and $(n',nid') \in his\ t\ n\ nid$
and $(n'',nid'') \in his\ t\ n\ nid$

and *hisPred t n nid n'=hisPred t n nid n''*
shows *n'=n''*
proof (*rule ccontr*)
assume ¬ *n'=n''*
hence *n'>n'' ∨ n'<n''* **by** *auto*
thus *False*
proof
assume *n'<n''*
hence *fst (n',nid')<n''* **by** *simp*
moreover from *assms(2)* **have** *hisPred t n nid n'<n'* **using** *hisPrev-prop(1)* **by** *simp*
with *assms* **have** *hisPred t n nid n''<n'* **by** *simp*
hence *hisPred t n nid n''<fst (n',nid')* **by** *simp*
ultimately show *False* **using** *hisPrev-nex-less*[*of n'' t n nid*] *assms* **by** *auto*
next
assume *n'>n''*
hence *fst (n'',nid')<n'* **by** *simp*
moreover from *assms(1)* **have** *hisPred t n nid n''<n''* **using** *hisPrev-prop(1)* **by** *simp*
with *assms* **have** *hisPred t n nid n'<n''* **by** *simp*
hence *hisPred t n nid n'<fst (n'',nid')* **by** *simp*
ultimately show *False* **using** *hisPrev-nex-less*[*of n' t n nid*] *assms* **by** *auto*
qed
qed

lemma *his-determ-ext*:
shows *n'≤n ⟹ (∃ nid'. (n',nid')∈his t n nid) ⟹ (∃!nid'. (n',nid')∈his t n nid) ∧*
((∃ n''<n'. ∃ nid'. (n'',nid')∈ his t n nid) ⟶ (∃ x. his-prop t n nid n' (THE nid'. (n',nid')∈his t n nid) x) ∧
(hisPred t n nid n', (SOME nid'. (hisPred t n nid n', nid') ∈ his t n nid)) = (SOME x. his-prop t n nid n' (THE nid'. (n',nid')∈his t n nid) x))
proof (*induction n' rule*: *my-induct*)
case *base*
then obtain *nid'* **where** *(n, nid') ∈ his t n nid* **by** *auto*
hence *∃!nid'. (n, nid') ∈ his t n nid*
proof
fix *nid''* **assume** *(n, nid'') ∈ his t n nid*
with *his-determ-base* **have** *nid''=nid* **by** *simp*
moreover from *‹(n, nid') ∈ his t n nid›* **have** *nid'=nid* **using** *his-determ-base* **by** *simp*
ultimately show *nid'' = nid'* **by** *simp*
qed
moreover have *(∃ n''<n. ∃ nid'. (n'',nid')∈ his t n nid) ⟶ (∃ x. his-prop t n nid n (THE nid'. (n,nid')∈his t n nid) x) ∧ (hisPred t n nid n, (SOME nid'. (hisPred t n nid n, nid') ∈ his t n nid)) = (SOME x. his-prop t n nid n (THE nid'. (n,nid')∈his t n nid) x)*
proof
assume *∃ n''<n. ∃ nid'. (n'',nid')∈ his t n nid*
hence *∃ nid'. (hisPred t n nid n, nid')∈ his t n nid* **using** *hisPrev-prop(2)* **by** *simp*
hence *(hisPred t n nid n, (SOME nid'. (hisPred t n nid n, nid') ∈ his t n nid)) ∈ his t n nid*
using *someI-ex*[*of λnid'. (hisPred t n nid n, nid') ∈ his t n nid*] **by** *simp*
thus *(∃ x. his-prop t n nid n (THE nid'. (n,nid')∈his t n nid) x) ∧*
(hisPred t n nid n, (SOME nid'. (hisPred t n nid n, nid') ∈ his t n nid)) = (SOME x. his-prop t n nid n (THE nid'. (n,nid')∈his t n nid) x)
proof (*rule his.cases*)
assume *(hisPred t n nid n, SOME nid'. (hisPred t n nid n, nid') ∈ his t n nid) = (n, nid)*
hence *hisPred t n nid n=n* **by** *simp*
with *‹∃ n''<n. ∃ nid'. (n'',nid')∈ his t n nid›* **show** *?thesis* **using** *hisPrev-prop(1)*[*of n t n nid*] **by** *force*

```
    next
      fix n'' nid'' assume asmp: (hisPred t n nid n, SOME nid'. (hisPred t n nid n, nid') ∈ his t n nid)
= (SOME x. his-prop t n nid n'' nid'' x)
      and (n'', nid'') ∈ his t n nid and ∃x. his-prop t n nid n'' nid'' x
      moreover have n''=n
      proof (rule antisym)
        show n''≥n
        proof (rule ccontr)
          assume (¬n''≥n)
          hence n''<n by simp
          moreover have n''>hisPred t n nid n
          proof −
            let ?x=λx. his-prop t n nid n'' nid'' x
            from ‹∃x. his-prop t n nid n'' nid'' x› have his-prop t n nid n'' nid'' (SOME x. ?x x)
              using someI-ex[of ?x] by auto
            hence n''>fst (SOME x. ?x x) using latestAct-prop(2)[of n'' nid'' t] by force
             moreover from asmp have fst (hisPred t n nid n, SOME nid'. (hisPred t n nid n, nid') ∈
his t n nid) = fst (SOME x. ?x x) by simp
            ultimately show ?thesis by simp
          qed
          moreover from ‹∃n''<n. ∃nid'. (n'',nid')∈ his t n nid›
            have ¬(∃x∈his t n nid. fst x < n ∧ fst x > hisPred t n nid n)
            using hisPrev-nex-less by simp
          ultimately show False using ‹(n'', nid'') ∈ his t n nid› by auto
        qed
      next
        from ‹(n'', nid'') ∈ his t n nid› show n'' ≤ n using his-le by auto
      qed
     ultimately have (hisPred t n nid n, SOME nid'. (hisPred t n nid n, nid') ∈ his t n nid) = (SOME
x. his-prop t n nid n nid'' x) by simp
      moreover from ‹n''=n› ‹(n'', nid'') ∈ his t n nid› have (n, nid'') ∈ his t n nid by simp
      with ‹∃!nid'. (n,nid') ∈ his t n nid› have nid''=(THE nid'. (n,nid')∈his t n nid)
        using the1-equality[of λnid'. (n, nid') ∈ his t n nid] by simp
     moreover from ‹∃x. his-prop t n nid n'' nid'' x› ‹n''=n› ‹nid''=(THE nid'. (n,nid')∈his t n nid)›
        have ∃x. his-prop t n nid n (THE nid'. (n,nid')∈his t n nid) x by simp
      ultimately show ?thesis by simp
    qed
  qed
  ultimately show ?case by simp
next
  case (step n')
  then obtain nid' where (n', nid') ∈ his t n nid by auto
  hence ∃!nid'. (n', nid') ∈ his t n nid
  proof (rule his.cases)
    assume (n', nid') = (n, nid)
    hence n'=n by simp
    with step.hyps show ?thesis by simp
  next
    fix n'''' nid''''
    assume (n'''', nid'''') ∈ his t n nid
      and n'nid': (n', nid') = (SOME x. his-prop t n nid n'''' nid'''' x)
      and (n'''', nid'''') ∈ his t n nid and ∃x. his-prop t n nid n'''' nid'''' x
    from ‹(n', nid') ∈ his t n nid› show ?thesis
    proof
      fix nid'' assume (n', nid'') ∈ his t n nid
```

thus $nid'' = nid'$
proof (*rule his.cases*)
assume $(n', nid'') = (n, nid)$
hence $n'=n$ **by** *simp*
with *step.hyps* **show** *?thesis* **by** *simp*
next
fix $n''' nid'''$
assume $(n''', nid''') \in his\ t\ n\ nid$
and $n'nid''$: $(n', nid'') = (SOME\ x.\ his\text{-}prop\ t\ n\ nid\ n''' nid''' x)$
and $(n''', nid''') \in his\ t\ n\ nid$ **and** $\exists x.\ his\text{-}prop\ t\ n\ nid\ n''' nid''' x$
moreover have $n'''=n''''$
proof −
have $hisPred\ t\ n\ nid\ n''' = n'$
proof −
from $n'nid''$ ‹$\exists x.\ his\text{-}prop\ t\ n\ nid\ n''' nid''' x$›
have $his\text{-}prop\ t\ n\ nid\ n''' nid'''\ (n',nid'')$
using *someI-ex*[*of* $\lambda x.\ his\text{-}prop\ t\ n\ nid\ n''' nid'''\ x$] **by** *auto*
hence $n'''>n'$ **using** *latestAct-prop*(*2*) **by** *simp*
moreover from ‹$(n''', nid''') \in his\ t\ n\ nid$› **have** $n'''\leq n$ **using** *his-le* **by** *auto*
moreover from ‹$(n''', nid''') \in his\ t\ n\ nid$›
have $\exists nid'.\ (n''', nid') \in his\ t\ n\ nid$ **by** *auto*
ultimately have $(\exists n'<n'''.\ \exists nid'.\ (n',nid')\in his\ t\ n\ nid) \longrightarrow (\exists!nid'.\ (n''',nid') \in his\ t\ n\ nid) \wedge (hisPred\ t\ n\ nid\ n''', (SOME\ nid'.\ (hisPred\ t\ n\ nid\ n''', nid') \in his\ t\ n\ nid)) = (SOME\ x.\ his\text{-}prop\ t\ n\ nid\ n'''\ (THE\ nid'.\ (n''',nid')\in his\ t\ n\ nid)\ x)$ **using** *step.IH* **by** *auto*
with ‹$n'''>n'$› ‹$(n', nid') \in his\ t\ n\ nid$› **have** $\exists!nid'.\ (n''',nid') \in his\ t\ n\ nid$ **and** $(hisPred\ t\ n\ nid\ n''', (SOME\ nid'.\ (hisPred\ t\ n\ nid\ n''', nid') \in his\ t\ n\ nid)) = (SOME\ x.\ his\text{-}prop\ t\ n\ nid\ n'''\ (THE\ nid'.\ (n''',nid')\in his\ t\ n\ nid)\ x)$ **by** *auto*
moreover from ‹$\exists!nid'.\ (n''',nid') \in his\ t\ n\ nid$› ‹$(n''', nid''') \in his\ t\ n\ nid$› **have** $nid'''=(THE\ nid'.\ (n''',nid')\in his\ t\ n\ nid)$ **using** *the1-equality*[*of* $\lambda nid'.\ (n''', nid') \in his\ t\ n\ nid$] **by** *simp*
ultimately have $(hisPred\ t\ n\ nid\ n''', (SOME\ nid'.\ (hisPred\ t\ n\ nid\ n''', nid') \in his\ t\ n\ nid)) = (SOME\ x.\ his\text{-}prop\ t\ n\ nid\ n'''\ nid'''\ x)$ **by** *simp*
with $n'nid''$ **have** $(n', nid'') = (hisPred\ t\ n\ nid\ n''', (SOME\ nid'.\ (hisPred\ t\ n\ nid\ n''', nid') \in his\ t\ n\ nid))$ **by** *simp*
thus *?thesis* **by** *simp*
qed
moreover have $hisPred\ t\ n\ nid\ n'''' = n'$
proof −
from $n'nid'$ ‹$\exists x.\ his\text{-}prop\ t\ n\ nid\ n'''' nid'''' x$› **have** $his\text{-}prop\ t\ n\ nid\ n'''' nid''''\ (n',nid')$
using *someI-ex*[*of* $\lambda x.\ his\text{-}prop\ t\ n\ nid\ n'''' nid''''\ x$] **by** *auto*
hence $n''''>n'$ **using** *latestAct-prop*(*2*) **by** *simp*
moreover from ‹$(n'''', nid'''') \in his\ t\ n\ nid$› **have** $n''''\leq n$ **using** *his-le* **by** *auto*
moreover from ‹$(n'''', nid'''') \in his\ t\ n\ nid$›
have $\exists nid'.\ (n'''', nid') \in his\ t\ n\ nid$ **by** *auto*
ultimately have $(\exists n'<n''''.\ \exists nid'.\ (n',nid')\in his\ t\ n\ nid) \longrightarrow (\exists!nid'.\ (n'''',nid') \in his\ t\ n\ nid) \wedge (hisPred\ t\ n\ nid\ n'''', (SOME\ nid'.\ (hisPred\ t\ n\ nid\ n'''', nid') \in his\ t\ n\ nid)) = (SOME\ x.\ his\text{-}prop\ t\ n\ nid\ n''''\ (THE\ nid'.\ (n'''',nid')\in his\ t\ n\ nid)\ x)$ **using** *step.IH* **by** *auto*
with ‹$n''''>n'$› ‹$(n', nid') \in his\ t\ n\ nid$› **have** $\exists!nid'.\ (n'''',nid') \in his\ t\ n\ nid$ **and** $(hisPred\ t\ n\ nid\ n'''', (SOME\ nid'.\ (hisPred\ t\ n\ nid\ n'''', nid') \in his\ t\ n\ nid)) = (SOME\ x.\ his\text{-}prop\ t\ n\ nid\ n''''\ (THE\ nid'.\ (n'''',nid')\in his\ t\ n\ nid)\ x)$ **by** *auto*
moreover from ‹$\exists!nid'.\ (n'''',nid') \in his\ t\ n\ nid$› ‹$(n'''', nid'''') \in his\ t\ n\ nid$› **have** $nid''''=(THE\ nid'.\ (n'''',nid')\in his\ t\ n\ nid)$ **using** *the1-equality*[*of* $\lambda nid'.\ (n'''', nid') \in his\ t\ n\ nid$] **by** *simp*
ultimately have $(hisPred\ t\ n\ nid\ n'''', (SOME\ nid'.\ (hisPred\ t\ n\ nid\ n'''', nid') \in his\ t\ n\ nid)) = (SOME\ x.\ his\text{-}prop\ t\ n\ nid\ n''''\ nid''''\ x)$ **by** *simp*
with $n'nid'$ **have** $(n', nid') = (hisPred\ t\ n\ nid\ n'''', (SOME\ nid'.\ (hisPred\ t\ n\ nid\ n'''', nid')$

$\in$ *his t n nid*)) **by** *simp*
thus *?thesis* **by** *simp*
qed
ultimately have *hisPred t n nid n'''=hisPred t n nid n''''* **..**
moreover have $\exists\, n'<n'''.\ \exists\, nid'.\ (n',nid')\in$ *his t n nid*
proof −
from *n'nid''* ‹$\exists\, x.$ *his-prop t n nid n''' nid''' x*› **have** *his-prop t n nid n''' nid'''* (*n'*,*nid''*)
using *someI-ex*[*of* $\lambda x.$ *his-prop t n nid n''' nid''' x*] **by** *auto*
hence *n'''>n'* **using** *latestAct-prop*(*2*) **by** *simp*
with ‹(*n'*, *nid'*) $\in$ *his t n nid*› **show** *?thesis* **by** *auto*
qed
moreover have $\exists\, n'<n''''.\ \exists\, nid'.\ (n',nid')\in$ *his t n nid*
proof −
from *n'nid'* ‹$\exists\, x.$ *his-prop t n nid n'''' nid'''' x*› **have** *his-prop t n nid n'''' nid''''* (*n'*,*nid'*)
using *someI-ex*[*of* $\lambda x.$ *his-prop t n nid n'''' nid'''' x*] **by** *auto*
hence *n''''>n'* **using** *latestAct-prop*(*2*) **by** *simp*
with ‹(*n'*, *nid'*) $\in$ *his t n nid*› **show** *?thesis* **by** *auto*
qed
ultimately show *?thesis*
using *hisPrev-same* ‹(*n'''*, *nid'''*) $\in$ *his t n nid*› ‹(*n''''*, *nid''''*) $\in$ *his t n nid*›
by *blast*
qed
moreover have *nid'''=nid''''*
proof −
from *n'nid''* ‹$\exists\, x.$ *his-prop t n nid n''' nid''' x*›
have *his-prop t n nid n''' nid'''* (*n'*,*nid''*)
using *someI-ex*[*of* $\lambda x.$ *his-prop t n nid n''' nid''' x*] **by** *auto*
hence *n'''>n'* **using** *latestAct-prop*(*2*) **by** *simp*
moreover from ‹(*n'''*, *nid'''*) $\in$ *his t n nid*› **have** *n'''*$\leq$ *n* **using** *his-le* **by** *auto*
moreover from ‹(*n'''*, *nid'''*) $\in$ *his t n nid*›
have $\exists\, nid'.$ (*n'''*, *nid'*) $\in$ *his t n nid* **by** *auto*
ultimately have $\exists!nid'.$ (*n'''*, *nid'*) $\in$ *his t n nid* **using** *step.IH* **by** *auto*
with ‹(*n'''*, *nid'''*) $\in$ *his t n nid*› ‹(*n''''*, *nid''''*) $\in$ *his t n nid*› ‹*n'''=n''''*›
show *?thesis* **by** *auto*
qed
ultimately have (*n'*, *nid'*) = (*n'*, *nid''*) **using** *n'nid'* **by** *simp*
thus *nid''* = *nid'* **by** *simp*
qed
qed
qed
moreover have ($\exists\, n''<n'.\ \exists\, nid'.\ (n'',nid')\in$ *his t n nid*) $\longrightarrow$ ($\exists\, x.$ *his-prop t n nid n'* (*THE nid'*. (*n'*,*nid'*)$\in$*his t n nid*) *x*) $\wedge$ (*hisPred t n nid n'*, (*SOME nid'*. (*hisPred t n nid n'*, *nid'*) $\in$ *his t n nid*)) = (*SOME x. his-prop t n nid n'* (*THE nid'*. (*n'*,*nid'*)$\in$*his t n nid*) *x*)
proof
assume $\exists\, n''<n'.\ \exists\, nid'.\ (n'',nid')\in$ *his t n nid*
hence $\exists\, nid'.$ (*hisPred t n nid n'*, *nid'*)$\in$ *his t n nid* **using** *hisPrev-prop*(*2*) **by** *simp*
hence (*hisPred t n nid n'*, (*SOME nid'*. (*hisPred t n nid n'*, *nid'*) $\in$ *his t n nid*)) $\in$ *his t n nid*
using *someI-ex*[*of* $\lambda nid'.$ (*hisPred t n nid n'*, *nid'*) $\in$ *his t n nid*] **by** *simp*
thus ($\exists\, x.$ *his-prop t n nid n'* (*THE nid'*. (*n'*,*nid'*)$\in$*his t n nid*) *x*) $\wedge$ (*hisPred t n nid n'*, (*SOME nid'*. (*hisPred t n nid n'*, *nid'*) $\in$ *his t n nid*)) = (*SOME x. his-prop t n nid n'* (*THE nid'*. (*n'*,*nid'*)$\in$*his t n nid*) *x*)
proof (*rule his.cases*)
assume (*hisPred t n nid n'*, *SOME nid'*. (*hisPred t n nid n'*, *nid'*) $\in$ *his t n nid*) = (*n*, *nid*)
hence *hisPred t n nid n'=n* **by** *simp*
moreover from ‹$\exists\, n''<n'.\ \exists\, nid'.\ (n'',nid')\in$ *his t n nid*› **have** *hisPred t n nid n'<n'*

using *hisPrev-prop(1)*[*of n′*] **by** *force*
ultimately show *?thesis* **using** *step.hyps* **by** *simp*
next
fix *n″ nid″* **assume** *asmp*: (*hisPred t n nid n′*, *SOME nid′*. (*hisPred t n nid n′*, *nid′*) ∈ *his t n nid*) = (*SOME x*. *his-prop t n nid n″ nid″ x*)
and (*n″*, *nid″*) ∈ *his t n nid* **and** ∃ *x*. *his-prop t n nid n″ nid″ x*
moreover have *n″*=*n′*
proof (*rule antisym*)
show *n″*≥*n′*
proof (*rule ccontr*)
assume (¬*n″*≥*n′*)
hence *n″*<*n′* **by** *simp*
moreover have *n″*>*hisPred t n nid n′*
proof −
let *?x*=λ*x*. *his-prop t n nid n″ nid″ x*
from ‹∃ *x*. *his-prop t n nid n″ nid″ x›* **have** *his-prop t n nid n″ nid″* (*SOME x*. *?x x*)
using *someI-ex*[*of ?x*] **by** *auto*
hence *n″*>*fst* (*SOME x*. *?x x*) **using** *latestAct-prop(2)*[*of n″ nid″ t*] **by** *force*
moreover from *asmp* **have** *fst* (*hisPred t n nid n′*, *SOME nid′*. (*hisPred t n nid n′*, *nid′*) ∈ *his t n nid*) = *fst* (*SOME x*. *?x x*) **by** *simp*
ultimately show *?thesis* **by** *simp*
qed
moreover from ‹∃ *n″*<*n′*. ∃ *nid′*. (*n″*,*nid′*)∈ *his t n nid*›
have ¬(∃ *x*∈*his t n nid*. *fst x* < *n′* ∧ *fst x* > *hisPred t n nid n′*)
using *hisPrev-nex-less* **by** *simp*
ultimately show *False* **using** ‹(*n″*, *nid″*) ∈ *his t n nid*› **by** *auto*
qed
next
show *n′*≥*n″*
proof (*rule ccontr*)
assume (¬*n′*≥*n″*)
hence *n′*<*n″* **by** *simp*
moreover from ‹(*n″*, *nid″*) ∈ *his t n nid*› **have** *n″*≤ *n* **using** *his-le* **by** *auto*
moreover from ‹(*n″*, *nid″*) ∈ *his t n nid*› **have** ∃ *nid′*. (*n″*, *nid′*) ∈ *his t n nid* **by** *auto*
ultimately have (∃ *n′*<*n″*. ∃ *nid′*. (*n′*,*nid′*)∈ *his t n nid*) ⟶ (∃!*nid′*. (*n″*,*nid′*) ∈ *his t n nid*) ∧ (*hisPred t n nid n″*, (*SOME nid′*. (*hisPred t n nid n″*, *nid′*) ∈ *his t n nid*)) = (*SOME x*. *his-prop t n nid n″* (*THE nid′*. (*n″*,*nid′*)∈*his t n nid*) *x*) **using** *step.IH* **by** *auto*
with ‹*n′*<*n″*› ‹(*n′*, *nid′*) ∈ *his t n nid*› **have** ∃!*nid′*. (*n″*,*nid′*) ∈ *his t n nid* **and** (*hisPred t n nid n″*, (*SOME nid′*. (*hisPred t n nid n″*, *nid′*) ∈ *his t n nid*)) = (*SOME x*. *his-prop t n nid n″* (*THE nid′*. (*n″*,*nid′*)∈*his t n nid*) *x*) **by** *auto*
moreover from ‹∃!*nid′*. (*n″*,*nid′*) ∈ *his t n nid*› ‹(*n″*, *nid″*) ∈ *his t n nid*›
have *nid″* = (*THE nid′*. (*n″*,*nid′*)∈*his t n nid*)
using *the1-equality*[*of* λ*nid′*. (*n″*, *nid′*) ∈ *his t n nid*] **by** *simp*
ultimately have (*hisPred t n nid n″*, (*SOME nid′*. (*hisPred t n nid n″*, *nid′*) ∈ *his t n nid*)) = (*SOME x*. *his-prop t n nid n″ nid″ x*) **by** *simp*
with *asmp* **have** (*hisPred t n nid n′*, *SOME nid′*. (*hisPred t n nid n′*, *nid′*) ∈ *his t n nid*)=(*hisPred t n nid n″*, *SOME nid′*. (*hisPred t n nid n″*, *nid′*) ∈ *his t n nid*) **by** *simp*
hence *hisPred t n nid n′* = *hisPred t n nid n″* **by** *simp*
with ‹∃ *n″*<*n′*. ∃ *nid′*. (*n″*, *nid′*) ∈ *his t n nid*› ‹*n′*<*n″*› ‹(*n′*, *nid′*) ∈ *his t n nid*› ‹(*n″*, *nid″*) ∈ *his t n nid*› ‹(*n′*, *nid′*) ∈ *his t n nid*› **have** *n′*=*n″* **using** *hisPrev-same* **by** *blast*
with ‹*n′*<*n″*› **show** *False* **by** *simp*
qed
qed
ultimately have (*hisPred t n nid n′*, *SOME nid′*. (*hisPred t n nid n′*, *nid′*) ∈ *his t n nid*) = (*SOME x*. *his-prop t n nid n′ nid″ x*) **by** *simp*

moreover from ‹(*n''*, *nid''*) ∈ *his t n nid*› ‹*n''*=*n'*› **have** (*n'*, *nid''*) ∈ *his t n nid* **by** *simp*
with ‹∃!*nid'*. (*n'*,*nid'*) ∈ *his t n nid*› **have** *nid''*=(*THE nid'*. (*n'*,*nid'*)∈*his t n nid*)
using *the1-equality*[*of* λ*nid'*. (*n'*, *nid'*) ∈ *his t n nid*] **by** *simp*
moreover from ‹∃ *x*. *his-prop t n nid n'' nid'' x*› ‹*n''*=*n'*› ‹*nid''*=(*THE nid'*. (*n'*,*nid'*)∈*his t n nid*)›
have ∃ *x*. *his-prop t n nid n'* (*THE nid'*. (*n'*,*nid'*)∈*his t n nid*) *x* **by** *simp*
ultimately show *?thesis* **by** *simp*
qed
qed
ultimately show *?case* **by** *simp*
qed

corollary *his-determ-ex*:
assumes (*n'*,*nid'*)∈*his t n nid*
shows ∃!*nid'*. (*n'*,*nid'*)∈*his t n nid*
using *assms his-le his-determ-ext*[*of n' n t nid*] **by** *force*

corollary *his-determ*:
assumes (*n'*,*nid'*)∈*his t n nid*
and (*n'*,*nid''*)∈*his t n nid*
shows *nid'*=*nid''* **using** *assms his-le his-determ-ext*[*of n' n t nid*] **by** *force*

corollary *his-determ-the*:
assumes (*n'*,*nid'*)∈*his t n nid*
shows (*THE nid'*. (*n'*, *nid'*)∈*his t n nid*) = *nid'*
using *assms his-determ theI'*[*of* λ*nid'*. (*n'*, *nid'*)∈*his t n nid*] *his-determ-ex* **by** *simp*

6.2.5 Blockchain Development

definition *devBC*::*trace* ⇒ *nat* ⇒ *'nid* ⇒ *nat* ⇒ *'nid option*
where *devBC t n nid n'* ≡
(*if* (∃ *nid'*. (*n'*, *nid'*)∈ *his t n nid*) *then* (*Some* (*THE nid'*. (*n'*, *nid'*)∈*his t n nid*))
else Option.None)

lemma *devBC-some*[*simp*]: **assumes** ‖*nid*‖$_{t\ n}$ **shows** *devBC t n nid n* = *Some nid*
proof −
from *assms* **have** (*n*, *nid*)∈ *his t n nid* **using** *his.intros*(*1*) **by** *simp*
hence *devBC t n nid n* = (*Some* (*THE nid'*. (*n*, *nid'*)∈*his t n nid*)) **using** *devBC-def* **by** *auto*
moreover have (*THE nid'*. (*n*, *nid'*)∈*his t n nid*) = *nid*
proof
from ‹(*n*, *nid*)∈ *his t n nid*› **show** (*n*, *nid*)∈ *his t n nid* **.**
next
fix *nid'* **assume** (*n*, *nid'*) ∈ *his t n nid*
thus *nid'* = *nid* **using** *his-determ-base* **by** *simp*
qed
ultimately show *?thesis* **by** *simp*
qed

lemma *devBC-act*: **assumes** ¬ *Option.is-none* (*devBC t n nid n'*) **shows** ‖*the* (*devBC t n nid n'*)‖$_{t\ n'}$
proof −
from *assms* **have** ¬ *devBC t n nid n'*=*Option.None* **by** (*metis is-none-simps*(*1*))
then obtain *nid'* **where** (*n'*, *nid'*)∈ *his t n nid* **and** *devBC t n nid n'* = (*Some* (*THE nid'*. (*n'*, *nid'*)∈*his t n nid*))
using *devBC-def*[*of t n nid*] **by** *metis*
hence *nid'*= (*THE nid'*. (*n'*, *nid'*)∈*his t n nid*) **using** *his-determ-the* **by** *simp*
with ‹*devBC t n nid n'* = (*Some* (*THE nid'*. (*n'*, *nid'*)∈*his t n nid*))› **have** *the* (*devBC t n nid n'*) =

nid′ **by** *simp*
with ‹(*n′*, *nid′*)∈ *his t n nid*› **show** *?thesis* **using** *his-act* **by** *simp*
qed

lemma *his-ex*:
assumes ¬*Option.is-none* (*devBC t n nid n′*)
shows ∃ *nid′*. (*n′*,*nid′*)∈*his t n nid*
proof (*rule ccontr*)
assume ¬(∃ *nid′*. (*n′*,*nid′*)∈*his t n nid*)
with *devBC-def* **have** *Option.is-none* (*devBC t n nid n′*) **by** *simp*
with *assms* **show** *False* **by** *simp*
qed

lemma *devExt-nopt-leq*:
assumes ¬*Option.is-none* (*devBC t n nid n′*)
shows *n′*≤*n*
proof −
from *assms* **have** ∃ *nid′*. (*n′*,*nid′*)∈*his t n nid* **using** *his-ex* **by** *simp*
then obtain *nid′* **where** (*n′*,*nid′*)∈*his t n nid* **by** *auto*
with *his-le*[*of* (*n′*,*nid′*)] **show** *?thesis* **by** *simp*
qed

An extended version of the development in which deactivations are filled with the last value.

function *devExt*::*trace* ⇒ *nat* ⇒ *'nid* ⇒ *nat* ⇒ *nat* ⇒ *'nid BC*
where ⟦∃ *n′*<n_s. ¬*Option.is-none* (*devBC t n nid n′*); *Option.is-none* (*devBC t n nid* n_s)⟧ ⟹ *devExt t n nid* n_s *0* = *bc* ($\sigma_{\textit{the (devBC t n nid (GREATEST n'. n'<n}_s \textit{ ∧ ¬Option.is-none (devBC t n nid n')))}}$(*t* (*GREATEST n′*. *n′*<n_s ∧ ¬*Option.is-none* (*devBC t n nid n′*))))
| ⟦¬ (∃ *n′*<n_s. ¬*Option.is-none* (*devBC t n nid n′*)); *Option.is-none* (*devBC t n nid* n_s)⟧ ⟹ *devExt t n nid* n_s *0* = []
| ¬ *Option.is-none* (*devBC t n nid* n_s) ⟹ *devExt t n nid* n_s *0* = *bc* ($\sigma_{\textit{the (devBC t n nid n}_s\textit{)}}$(*t* n_s))
| ¬ *Option.is-none* (*devBC t n nid* (n_s + *Suc n′*)) ⟹ *devExt t n nid* n_s (*Suc n′*) = *bc* ($\sigma_{\textit{the (devBC t n nid (n}_s \textit{ + Suc n'))}}$(
(n_s + *Suc n′*)))
| *Option.is-none* (*devBC t n nid* (n_s + *Suc n′*)) ⟹ *devExt t n nid* n_s (*Suc n′*) = *devExt t n nid* n_s *n′*
proof −
show ⋀n_s *t n nid* n_s' *ta na nida*.
∃ *n′*<n_s. ¬ *Option.is-none* (*devBC t n nid n′*) ⟹
Option.is-none (*devBC t n nid* n_s) ⟹
∃ *n′*<n_s'. ¬ *Option.is-none* (*devBC ta na nida n′*) ⟹
Option.is-none (*devBC ta na nida* n_s') ⟹
(*t*, *n*, *nid*, n_s, *0*) = (*ta*, *na*, *nida*, n_s', *0*) ⟹
bc ($\sigma_{\textit{the (devBC t n nid (GREATEST n'. n' < n}_s \textit{ ∧ ¬ Option.is-none (devBC t n nid n')))}}$*t* (*GREATEST n′*. *n′* < n_s ∧ ¬ *Option.is-none* (*devBC t n nid n′*))) =
bc ($\sigma_{\textit{the (devBC ta na nida (GREATEST n'. n' < n}_s' \textit{ ∧ ¬ Option.is-none (devBC ta na nida n')))}}$*ta* (*GREATEST n′*. *n′* < n_s' ∧ ¬ *Option.is-none* (*devBC ta na nida n′*))) **by** *auto*
show ⋀n_s *t n nid* n_s' *ta na nida*.
∃ *n′*<n_s. ¬ *Option.is-none* (*devBC t n nid n′*) ⟹
Option.is-none (*devBC t n nid* n_s) ⟹
¬ (∃ *n′*<n_s'. ¬ *Option.is-none* (*devBC ta na nida n′*)) ⟹
Option.is-none (*devBC ta na nida* n_s') ⟹
(*t*, *n*, *nid*, n_s, *0*) = (*ta*, *na*, *nida*, n_s', *0*) ⟹
bc ($\sigma_{\textit{the (devBC t n nid (GREATEST n'. n' < n}_s \textit{ ∧ ¬ Option.is-none (devBC t n nid n')))}}$*t* (*GREATEST n′*. *n′* < n_s ∧ ¬ *Option.is-none* (*devBC t n nid n′*))) = [] **by** *auto*
show ⋀n_s *t n nid ta na nida* n_s'.
∃ *n′*<n_s. ¬ *Option.is-none* (*devBC t n nid n′*) ⟹

Option.is-none (*devBC t n nid* n_s) $\Longrightarrow$
$\neg$ *Option.is-none* (*devBC ta na nida* n_s') $\Longrightarrow$
(*t*, *n*, *nid*, n_s, *0*) = (*ta*, *na*, *nida*, n_s', *0*) $\Longrightarrow$
bc ($\sigma_{the\ (devBC\ t\ n\ nid\ (GREATEST\ n'.\ n' < n_s \wedge \neg\ Option.is\text{-}none\ (devBC\ t\ n\ nid\ n')))}$ *t* (*GREATEST* *n'*. $n' < n_s \wedge \neg$ *Option.is-none* (*devBC t n nid n'*))) =
bc ($\sigma_{the\ (devBC\ ta\ na\ nida\ n_s')}$ *ta* n_s') **by** *auto*

show $\bigwedge n_s$ *t n nid ta na nida* n_s' *n'*.
$\exists\, n'{<}n_s$. $\neg$ *Option.is-none* (*devBC t n nid n'*) $\Longrightarrow$
Option.is-none (*devBC t n nid* n_s) $\Longrightarrow$
$\neg$ *Option.is-none* (*devBC ta na nida* (n_s' + *Suc n'*)) $\Longrightarrow$
(*t*, *n*, *nid*, n_s, *0*) = (*ta*, *na*, *nida*, n_s', *Suc n'*) $\Longrightarrow$
bc ($\sigma_{the\ (devBC\ t\ n\ nid\ (GREATEST\ n'.\ n' < n_s \wedge \neg\ Option.is\text{-}none\ (devBC\ t\ n\ nid\ n')))}$ *t* (*GREATEST* *n'*. $n' < n_s \wedge \neg$ *Option.is-none* (*devBC t n nid n'*))) =
bc ($\sigma_{the\ (devBC\ ta\ na\ nida\ (n_s' + Suc\ n'))}$ *ta* (n_s' + *Suc n'*)) **by** *auto*

show $\bigwedge n_s$ *t n nid ta na nida* n_s' *n'*.
$\exists\, n'{<}n_s$. $\neg$ *Option.is-none* (*devBC t n nid n'*) $\Longrightarrow$
Option.is-none (*devBC t n nid* n_s) $\Longrightarrow$
Option.is-none (*devBC ta na nida* (n_s' + *Suc n'*)) $\Longrightarrow$
(*t*, *n*, *nid*, n_s, *0*) = (*ta*, *na*, *nida*, n_s', *Suc n'*) $\Longrightarrow$
bc ($\sigma_{the\ (devBC\ t\ n\ nid\ (GREATEST\ n'.\ n' < n_s \wedge \neg\ Option.is\text{-}none\ (devBC\ t\ n\ nid\ n')))}$ *t* (*GREATEST* *n'*. $n' < n_s \wedge \neg$ *Option.is-none* (*devBC t n nid n'*))) =
devExt-sumC (*ta*, *na*, *nida*, n_s', *n'*) **by** *auto*

show$\bigwedge n_s$ *t n nid* n_s' *ta na nida.*
$\neg$ ($\exists\, n'{<}n_s$. $\neg$ *Option.is-none* (*devBC t n nid n'*)) $\Longrightarrow$
Option.is-none (*devBC t n nid* n_s) $\Longrightarrow$
$\neg$ ($\exists\, n'{<}n_s'$. $\neg$ *Option.is-none* (*devBC ta na nida n'*)) $\Longrightarrow$
Option.is-none (*devBC ta na nida* n_s') $\Longrightarrow$ (*t*, *n*, *nid*, n_s, *0*) = (*ta*, *na*, *nida*, n_s', *0*) $\Longrightarrow$ [] = [] **by** *auto*

show $\bigwedge n_s$ *t n nid ta na nida* n_s'.
$\neg$ ($\exists\, n'{<}n_s$. $\neg$ *Option.is-none* (*devBC t n nid n'*)) $\Longrightarrow$
Option.is-none (*devBC t n nid* n_s) $\Longrightarrow$
$\neg$ *Option.is-none* (*devBC ta na nida* n_s') $\Longrightarrow$ (*t*, *n*, *nid*, n_s, *0*) = (*ta*, *na*, *nida*, n_s', *0*) $\Longrightarrow$ [] = *bc* ($\sigma_{the\ (devBC\ ta\ na\ nida\ n_s')}$ *ta* n_s') **by** *auto*

show $\bigwedge n_s$ *t n nid ta na nida* n_s' *n'*.
$\neg$ ($\exists\, n'{<}n_s$. $\neg$ *Option.is-none* (*devBC t n nid n'*)) $\Longrightarrow$
Option.is-none (*devBC t n nid* n_s) $\Longrightarrow$
$\neg$ *Option.is-none* (*devBC ta na nida* (n_s' + *Suc n'*)) $\Longrightarrow$
(*t*, *n*, *nid*, n_s, *0*) = (*ta*, *na*, *nida*, n_s', *Suc n'*) $\Longrightarrow$ [] = *bc* ($\sigma_{the\ (devBC\ ta\ na\ nida\ (n_s' + Suc\ n'))}$ *ta* (n_s' + *Suc n'*)) **by** *auto*

show $\bigwedge n_s$ *t n nid ta na nida* n_s' *n'*.
$\neg$ ($\exists\, n'{<}n_s$. $\neg$ *Option.is-none* (*devBC t n nid n'*)) $\Longrightarrow$
Option.is-none (*devBC t n nid* n_s) $\Longrightarrow$
Option.is-none (*devBC ta na nida* (n_s' + *Suc n'*)) $\Longrightarrow$ (*t*, *n*, *nid*, n_s, *0*) = (*ta*, *na*, *nida*, n_s', *Suc* *n'*) $\Longrightarrow$ [] = *devExt-sumC* (*ta*, *na*, *nida*, n_s', *n'*) **by** *auto*

show $\bigwedge t$ *n nid* n_s *ta na nida* n_s'.
$\neg$ *Option.is-none* (*devBC t n nid* n_s) $\Longrightarrow$
$\neg$ *Option.is-none* (*devBC ta na nida* n_s') $\Longrightarrow$
(*t*, *n*, *nid*, n_s, *0*) = (*ta*, *na*, *nida*, n_s', *0*) $\Longrightarrow$ *bc* ($\sigma_{the\ (devBC\ t\ n\ nid\ n_s)}$ *t* n_s) = *bc* ($\sigma_{the\ (devBC\ ta\ na\ nida\ n_s')}$ *ta* n_s') **by** *auto*

show $\bigwedge t$ *n nid* n_s *ta na nida* n_s' *n'*.
$\neg$ *Option.is-none* (*devBC t n nid* n_s) $\Longrightarrow$
$\neg$ *Option.is-none* (*devBC ta na nida* (n_s' + *Suc n'*)) $\Longrightarrow$
(*t*, *n*, *nid*, n_s, *0*) = (*ta*, *na*, *nida*, n_s', *Suc n'*) $\Longrightarrow$ *bc* ($\sigma_{the\ (devBC\ t\ n\ nid\ n_s)}$ *t* n_s) = *bc* ($\sigma_{the\ (devBC\ ta\ na\ nida\ (n_s' + Suc\ n'))}$ *ta* (n_s' + *Suc n'*)) **by** *auto*

show $\bigwedge t\ n\ nid\ n_s\ ta\ na\ nida\ n_s'\ n'.$
$\neg$ *Option.is-none* (*devBC t n nid* n_s) $\Longrightarrow$
Option.is-none (*devBC ta na nida* (n_s' + *Suc* n')) $\Longrightarrow$
(t, n, *nid*, n_s, 0) = (*ta*, *na*, *nida*, n_s', *Suc* n') $\Longrightarrow$ *bc* ($\sigma_{the\ (devBC\ t\ n\ nid\ n_s)}$ t n_s) = *devExt-sumC* (*ta*, *na*, *nida*, n_s', n') **by** *auto*
show $\bigwedge t\ n\ nid\ n_s\ n'\ ta\ na\ nida\ n_s'\ n'a.$
$\neg$ *Option.is-none* (*devBC t n nid* (n_s + *Suc* n')) $\Longrightarrow$
$\neg$ *Option.is-none* (*devBC ta na nida* (n_s' + *Suc* $n'a$)) $\Longrightarrow$
(t, n, *nid*, n_s, *Suc* n') = (*ta*, *na*, *nida*, n_s', *Suc* $n'a$) $\Longrightarrow$
bc ($\sigma_{the\ (devBC\ t\ n\ nid\ (n_s + Suc\ n'))}$ t (n_s + *Suc* n')) = *bc* ($\sigma_{the\ (devBC\ ta\ na\ nida\ (n_s' + Suc\ n'a))}$ *ta* (n_s' + *Suc* $n'a$)) **by** *auto*
show $\bigwedge t\ n\ nid\ n_s\ n'\ ta\ na\ nida\ n_s'\ n'a.$
$\neg$ *Option.is-none* (*devBC t n nid* (n_s + *Suc* n')) $\Longrightarrow$
Option.is-none (*devBC ta na nida* (n_s' + *Suc* $n'a$)) $\Longrightarrow$
(t, n, *nid*, n_s, *Suc* n') = (*ta*, *na*, *nida*, n_s', *Suc* $n'a$) $\Longrightarrow$ *bc* ($\sigma_{the\ (devBC\ t\ n\ nid\ (n_s + Suc\ n'))}$ t (n_s + *Suc* n')) = *devExt-sumC* (t, n, *nid*, n_s', $n'a$) **by** *auto*
show $\bigwedge t\ n\ nid\ n_s\ n'\ ta\ na\ nida\ n_s'\ n'a.$
Option.is-none (*devBC t n nid* (n_s + *Suc* n')) $\Longrightarrow$
Option.is-none (*devBC ta na nida* (n_s' + *Suc* $n'a$)) $\Longrightarrow$
(t, n, *nid*, n_s, *Suc* n') = (*ta*, *na*, *nida*, n_s', *Suc* $n'a$) $\Longrightarrow$ *devExt-sumC* (t, n, *nid*, n_s, n') = *devExt-sumC* (*ta*, *na*, *nida*, n_s', $n'a$) **by** *auto*
show $\bigwedge P\ x.$ ($\bigwedge n_s\ t\ n\ nid.\ \exists n'<n_s.\ \neg$ *Option.is-none* (*devBC t n nid* n') $\Longrightarrow$ *Option.is-none* (*devBC t n nid* n_s) $\Longrightarrow$ x = (t, n, *nid*, n_s, 0) $\Longrightarrow$ P) $\Longrightarrow$
($\bigwedge n_s\ t\ n\ nid.\ \neg$ ($\exists n'<n_s.\ \neg$ *Option.is-none* (*devBC t n nid* n')) $\Longrightarrow$ *Option.is-none* (*devBC t n nid* n_s) $\Longrightarrow$ x = (t, n, *nid*, n_s, 0) $\Longrightarrow$ P) $\Longrightarrow$
($\bigwedge t\ n\ nid\ n_s.\ \neg$ *Option.is-none* (*devBC t n nid* n_s) $\Longrightarrow$ x = (t, n, *nid*, n_s, 0) $\Longrightarrow$ P) $\Longrightarrow$
($\bigwedge t\ n\ nid\ n_s\ n'.\ \neg$ *Option.is-none* (*devBC t n nid* (n_s + *Suc* n')) $\Longrightarrow$ x = (t, n, *nid*, n_s, *Suc* n') $\Longrightarrow$ P) $\Longrightarrow$
($\bigwedge t\ n\ nid\ n_s\ n'.$ *Option.is-none* (*devBC t n nid* (n_s + *Suc* n')) $\Longrightarrow$ x = (t, n, *nid*, n_s, *Suc* n') $\Longrightarrow$ P) $\Longrightarrow$ P
proof −
fix P::*bool* **and** x::*trace* ×*nat*×′*nid*×*nat*×*nat*
assume *a1*:($\bigwedge n_s\ t\ n\ nid.\ \exists n'<n_s.\ \neg$ *Option.is-none* (*devBC t n nid* n') $\Longrightarrow$ *Option.is-none* (*devBC t n nid* n_s) $\Longrightarrow$ x = (t, n, *nid*, n_s, 0) $\Longrightarrow$ P) **and**
a2:($\bigwedge n_s\ t\ n\ nid.\ \neg$ ($\exists n'<n_s.\ \neg$ *Option.is-none* (*devBC t n nid* n')) $\Longrightarrow$ *Option.is-none* (*devBC t n nid* n_s) $\Longrightarrow$ x = (t, n, *nid*, n_s, 0) $\Longrightarrow$ P) **and**
a3:($\bigwedge t\ n\ nid\ n_s.\ \neg$ *Option.is-none* (*devBC t n nid* n_s) $\Longrightarrow$ x = (t, n, *nid*, n_s, 0) $\Longrightarrow$ P) **and**
a4:($\bigwedge t\ n\ nid\ n_s\ n'.\ \neg$ *Option.is-none* (*devBC t n nid* (n_s + *Suc* n')) $\Longrightarrow$ x = (t, n, *nid*, n_s, *Suc* n') $\Longrightarrow$ P) **and**
a5:($\bigwedge t\ n\ nid\ n_s\ n'.$ *Option.is-none* (*devBC t n nid* (n_s + *Suc* n')) $\Longrightarrow$ x = (t, n, *nid*, n_s, *Suc* n') $\Longrightarrow$ P)
show P
proof (*cases* x)
case (*fields t n nid* n_s n')
then show *?thesis*
proof (*cases* n')
case *0*
then show *?thesis*
proof *cases*
assume *Option.is-none* (*devBC t n nid* n_s)
thus *?thesis*
proof *cases*
assume $\exists n'<n_s.\ \neg$ *Option.is-none* (*devBC t n nid* n')
with ‹x = (t, n , *nid*, n_s, n')› ‹*Option.is-none* (*devBC t n nid* n_s)› ‹n'=*0*› **show** *?thesis* **using** *a1* **by** *simp*

```
        next
          assume ¬ (∃ n'<n_s. ¬ Option.is-none (devBC t n nid n'))
            with ‹x = (t, n , nid, n_s, n')› ‹Option.is-none (devBC t n nid n_s)› ‹n'=0› show ?thesis
using a2 by simp
        qed
       next
        assume ¬ Option.is-none (devBC t n nid n_s)
        with ‹x = (t, n , nid, n_s, n')› ‹n'=0› show ?thesis using a3 by simp
       qed
      next
       case (Suc n'')
       then show ?thesis
       proof cases
        assume Option.is-none (devBC t n nid (n_s + Suc n''))
        with ‹x = (t, n , nid, n_s, n')› ‹n'=Suc n''› show ?thesis using a5[of t n nid n_s n''] by simp
       next
        assume ¬ Option.is-none (devBC t n nid (n_s + Suc n''))
        with ‹x = (t, n , nid, n_s, n')› ‹n'=Suc n''› show ?thesis using a4[of t n nid n_s n''] by simp
       qed
      qed
    qed
  qed
qed
termination by lexicographic-order

lemma devExt-same:
  assumes ∀ n'''>n'. n'''≤n'' ⟶ Option.is-none (devBC t n nid n''')
    and n'≥n_s
    and n'''≤n''
  shows n'''≥n'⟹devExt t n nid n_s (n'''−n_s) = devExt t n nid n_s (n'−n_s)
proof (induction n''' rule: dec-induct)
  case base
  then show ?case by simp
next
  case (step n'''')
  hence Suc n''''>n' by simp
  moreover from step.hyps assms(3) have Suc n''''≤n'' by simp
  ultimately have Option.is-none (devBC t n nid (Suc n'''')) using assms(1) by simp
  moreover from assms(2) step.hyps have n''''≥n_s by simp
  hence Suc n'''' = n_s + Suc (n''''−n_s) by simp
  ultimately have Option.is-none (devBC t n nid (n_s + Suc (n''''−n_s))) by metis
  hence devExt t n nid n_s (Suc (n''''−n_s)) = devExt t n nid n_s (n''''−n_s) by simp
  moreover from ‹n''''≥n_s› have Suc (n''''−n_s) = Suc n''''−n_s by simp
  ultimately have devExt t n nid n_s (Suc n''''−n_s) = devExt t n nid n_s (n''''−n_s) by simp
  with step.IH show ?case by simp
qed

lemma devExt-bc[simp]:
  assumes ¬ Option.is-none (devBC t n nid (n'+n''))
  shows devExt t n nid n' n'' = bc (σ_the (devBC t n nid (n'+n''))(t (n'+n'')))
proof (cases n'')
  case 0
  with assms show ?thesis by simp
next
  case (Suc nat)
```

with *assms* **show** *?thesis* **by** *simp*
qed

lemma *devExt-greatest*:
assumes $\exists n'''<n'+n''.\ \neg$ *Option.is-none* (*devBC t n nid* n''')
and *Option.is-none* (*devBC t n nid* $(n'+n'')$) **and** $\neg\ n''=0$
shows *devExt t n nid* $n'\ n'' =$ *bc* ($\sigma_{the\ (devBC\ t\ n\ nid\ (GREATEST\ n'''.\ n'''<(n'+n'')\ \wedge\ \neg Option.is\text{-}none\ (devBC\ t\ n\ nid\ n'}$
(*GREATEST* $n'''.\ n'''<(n'+n'')\ \wedge\ \neg$*Option.is-none* (*devBC t n nid* n'''))))
proof −
let *?P*=$\lambda n'''.\ n'''<(n'+n'')\ \wedge\ \neg$*Option.is-none* (*devBC t n nid* n''')
let *?G*=*GREATEST* $n'''.$ *?P* n'''
have $\forall\, n'''>n'+n''.\ \neg$ *?P* n''' **by** *simp*
with ‹$\exists\, n'''<n'+n''.\ \neg$ *Option.is-none* (*devBC t n nid* n''')› **have** $\exists\, n'''.$ *?P* $n''' \wedge (\forall\, n''''.$ *?P* n''''
$\longrightarrow n''''\leq n'''$) **using** *boundedGreatest*[*of ?P*] **by** *blast*
hence *?P ?G* **using** *GreatestI-ex-nat*[*of ?P*] **by** *auto*
hence ¬*Option.is-none* (*devBC t n nid ?G*) **by** *simp*
show *?thesis*
proof *cases*
assume *?G*>n'
hence *?G*−n'+n' = *?G* **by** *simp*
with ‹¬*Option.is-none* (*devBC t n nid ?G*)› **have** ¬*Option.is-none* (*devBC t n nid* (*?G*−n'+n'))
by *simp*
moreover from ‹*?G*>n'› **have** *?G*−$n'\neq 0$ **by** *auto*
hence $\exists$ *nat. Suc nat* = *?G* − n' **by** *presburger*
then obtain *nat* **where** *Suc nat* = *?G*−n' **by** *auto*
ultimately have ¬*Option.is-none* (*devBC t n nid* (n'+*Suc nat*)) **by** *simp*
hence *devExt t n nid* n' (*Suc nat*) = *bc* ($\sigma_{the\ (devBC\ t\ n\ nid\ (n'\ +\ Suc\ nat))}$*t* ($n'$ + *Suc nat*)) **by**
simp
with ‹*Suc nat* = *?G* − n'› **have** *devExt t n nid* n' (*?G* − n') = *bc* ($\sigma_{the\ (devBC\ t\ n\ nid\ (?G-n'+n'))}$(*t*
(*?G*−n'+n'))) **by** *simp*
with ‹*?G*−n'+n' = *?G*› **have** *devExt t n nid* n' (*?G* − n') = *bc* ($\sigma_{the\ (devBC\ t\ n\ nid\ ?G)}$(*t ?G*)) **by**
simp
moreover have *devExt t n nid* n' (n' + n'' − n') = *devExt t n nid* n' (*?G* − n')
proof −
from ‹$\exists\, n'''.$ *?P* $n''' \wedge (\forall\, n''''.$ *?P* $n'''' \longrightarrow n''''\leq n'''$)› **have** $\forall\, n'''.$ *?P* $n''' \longrightarrow n'''\leq$*?G*
using *Greatest-le-nat*[*of ?P*] **by** *blast*
hence $\forall\, n'''>$*?G*. $n'''<n'+n'' \longrightarrow$ *Option.is-none* (*devBC t n nid* n''') **by** *auto*
with ‹*Option.is-none* (*devBC t n nid* (n'+n''))›
have $\forall\, n'''>$*?G*. $n'''\leq n'+n'' \longrightarrow$ *Option.is-none* (*devBC t n nid* n''') **by** *auto*
moreover from ‹*?P ?G*› **have** *?G*≤n'+n'' **by** *simp*
moreover from ‹*?G*>n'› **have** *?G*≥n' **by** *simp*
ultimately show *?thesis* **using** ‹*?G*>n'› *devExt-same*[*of ?G* n'+n'' *t n nid* n' n'+n''] **by** *blast*
qed
ultimately show *?thesis* **by** *simp*
next
assume ¬*?G*>n'
thus *?thesis*
proof *cases*
assume *?G*=n'
with ‹¬*Option.is-none* (*devBC t n nid ?G*)› **have** ¬ *Option.is-none* (*devBC t n nid* n') **by** *simp*
with ‹¬*Option.is-none* (*devBC t n nid ?G*)› **have** *devExt t n nid* n' *0* = *bc* ($\sigma_{the\ (devBC\ t\ n\ nid\ n')}$(*t*
n')) **by** *simp*
moreover have *devExt t n nid* n' n'' = *devExt t n nid* n' *0*
proof −
from ‹$\exists\, n'''.$ *?P* $n''' \wedge (\forall\, n''''.$ *?P* $n'''' \longrightarrow n''''\leq n'''$)› **have** $\forall\, n'''>$*?G*. *?P* $n''' \longrightarrow n'''\leq$*?G*

```
      using Greatest-le-nat[of ?P] by blast
    with ‹?G=n'› have ∀n'''>n'. n''' < n' + n'' ⟶ Option.is-none (devBC t n nid n''') by simp
    with ‹Option.is-none (devBC t n nid (n'+n''))›
     have ∀n'''>n'. n'''≤n'+n'' ⟶ Option.is-none (devBC t n nid n''') by auto
    moreover from ‹¬ n''=0› have n'<n'+n'' by simp
    ultimately show ?thesis using devExt-same[of n' n'+n'' t n nid n' n'+n''] by simp
   qed
   ultimately show ?thesis using ‹?G=n'› by simp
  next
   assume ¬?G=n'
   with ‹¬?G>n'› have ?G<n' by simp
   hence devExt t n nid n' n'' = devExt t n nid n' 0
   proof −
    from ‹∃n'''. ?P n''' ∧ (∀n''''. ?P n'''' ⟶ n''''≤n''')› have ∀n'''>?G. ?P n''' ⟶ n'''≤?G
     using Greatest-le-nat[of ?P] by blast
    with ‹¬?G>n'› have ∀n'''>n'. n'''<n'+n'' ⟶ Option.is-none (devBC t n nid n''') by auto
    with ‹Option.is-none (devBC t n nid (n'+n''))›
     have ∀n'''>n'. n'''≤n'+n'' ⟶ Option.is-none (devBC t n nid n''') by auto
    moreover from ‹?P ?G› have ?G<n'+n'' by simp
    moreover from ‹¬ n''=0› have n'<n'+n'' by simp
    ultimately show ?thesis using devExt-same[of n' n'+n'' t n nid n' n'+n''] by simp
   qed
  moreover have devExt t n nid n' 0 = bc (σthe (devBC t n nid (GREATEST n'''. n'''<n' ∧ ¬Option.is-none (devBC t n n
(GREATEST n'''. n'''<n' ∧ ¬Option.is-none (devBC t n nid n'''))))
   proof −
    from ‹¬ n''=0› have n'<n'+n'' by simp
    moreover from ‹∃n'''. ?P n''' ∧ (∀n''''. ?P n'''' ⟶ n''''≤n''')› have ∀n'''>?G. ?P n''' ⟶
n'''≤?G using Greatest-le-nat[of ?P] by blast
    ultimately have Option.is-none (devBC t n nid n') using ‹?G<n'› by simp
      moreover from ‹∀n'''>?G. ?P n''' ⟶ n'''≤?G› ‹?G<n'› ‹n'<n'+n''› have ∀n'''≥n'.
n'''<n'+n'' ⟶ Option.is-none (devBC t n nid n''') by auto
    have ∃n'''<n'. ¬ Option.is-none (devBC t n nid n''')
    proof −
     from ‹∃n'''<n'+n''. ¬ Option.is-none (devBC t n nid n''')› obtain n'''
      where n'''<n'+n'' and ¬ Option.is-none (devBC t n nid n''') by auto
     moreover have n'''<n'
     proof (rule ccontr)
      assume ¬n'''<n'
      hence n'''≥n' by simp
      with ‹∀n'''≥n'. n'''<n'+n'' ⟶ Option.is-none (devBC t n nid n''')› ‹n'''<n'+n''›
       ‹¬ Option.is-none (devBC t n nid n''')› show False by simp
     qed
     ultimately show ?thesis by auto
    qed
    ultimately show ?thesis by simp
   qed
   moreover have (GREATEST n'''. n'''<n' ∧ ¬Option.is-none (devBC t n nid n''')) = ?G
   proof(rule Greatest-equality)
    from ‹?P ?G› have ?G < n'+n'' and ¬Option.is-none (devBC t n nid ?G) by auto
    with ‹?G<n'› show ?G < n' ∧ ¬ Option.is-none (devBC t n nid ?G) by simp
   next
    fix y assume y < n' ∧ ¬ Option.is-none (devBC t n nid y)
    moreover from ‹∃n'''. ?P n''' ∧ (∀n''''. ?P n'''' ⟶ n''''≤n''')›
     have ∀n'''. ?P n''' ⟶ n'''≤?G using Greatest-le-nat[of ?P] by blast
    ultimately show y ≤ ?G by simp
```

```
    qed
    ultimately show ?thesis by simp
   qed
  qed
qed

lemma devExt-shift: devExt t n nid (n'+n'') 0 = devExt t n nid n' n''
proof (cases)
  assume n''=0
  thus ?thesis by simp
next
  assume ¬ (n''=0)
  thus ?thesis
  proof (cases)
    assume Option.is-none (devBC t n nid (n'+n''))
    thus ?thesis
    proof cases
      assume ∃ n'''<n'+n''. ¬ Option.is-none (devBC t n nid n''')
    with ‹Option.is-none (devBC t n nid (n'+n''))› have devExt t n nid (n'+n'') 0 = bc (σthe (devBC t n nid (GREATEST
(GREATEST n'''. n'''<(n'+n'') ∧ ¬Option.is-none (devBC t n nid n''')))) by simp
      moreover from ‹¬ (n''=0)› ‹Option.is-none (devBC t n nid (n'+n''))› ‹∃ n'''<n'+n''. ¬ Op-
tion.is-none (devBC t n nid n''')› have devExt t n nid n' n'' = bc (σthe (devBC t n nid (GREATEST n'''. n'''<(n'+n'') ∧ ¬
(GREATEST n'''. n'''<(n'+n'') ∧ ¬Option.is-none (devBC t n nid n')))) using devExt-greatest by
simp
      ultimately show ?thesis by simp
    next
      assume ¬ (∃ n'''<n'+n''. ¬ Option.is-none (devBC t n nid n'''))
      with ‹Option.is-none (devBC t n nid (n'+n''))› have devExt t n nid (n'+n'') 0=[] by simp
      moreover have devExt t n nid n' n''=[]
      proof −
        from ‹¬ (∃ n'''<n'+n''. ¬ Option.is-none (devBC t n nid n'''))› ‹n''≠0›
          have Option.is-none (devBC t n nid n') by simp
        moreover from ‹¬ (∃ n'''<n'+n''. ¬ Option.is-none (devBC t n nid n'''))›
          have ¬ (∃ n'''<n'. ¬ Option.is-none (devBC t n nid n''')) by simp
        ultimately have devExt t n nid n' 0=[] by simp
        moreover have devExt t n nid n' n''=devExt t n nid n' 0
        proof −
          from ‹¬ (∃ n'''<n'+n''. ¬ Option.is-none (devBC t n nid n'''))›
            have ∀ n'''>n'. n''' < n' + n'' ⟶ Option.is-none (devBC t n nid n''') by simp
          with ‹Option.is-none (devBC t n nid (n'+n''))›
            have ∀ n'''>n'. n'''≤n'+n'' ⟶ Option.is-none (devBC t n nid n''') by auto
          moreover from ‹¬ n''=0› have n'<n'+n'' by simp
          ultimately show ?thesis using devExt-same[of n' n'+n'' t n nid n' n'+n''] by simp
        qed
        ultimately show ?thesis by simp
      qed
      ultimately show ?thesis by simp
    qed
  next
    assume ¬ Option.is-none (devBC t n nid (n'+n''))
    hence devExt t n nid (n'+n'') 0 = bc (σthe (devBC t n nid (n'+n''))(t (n'+n''))) by simp
    moreover from ‹¬ Option.is-none (devBC t n nid (n'+n''))›
      have devExt t n nid n' n'' = bc (σthe (devBC t n nid (n'+n''))(t (n'+n''))) by simp
    ultimately show ?thesis by simp
  qed
```

qed

lemma *devExt-bc-geq*:
assumes ¬ *Option.is-none* (*devBC t n nid n′*) **and** $n' \geq n_s$
shows *devExt t n nid* n_s $(n'-n_s)$ = *bc* $(\sigma_{the\ (devBC\ t\ n\ nid\ n')}(t\ n'))$ (**is** *?LHS = ?RHS*)
proof −
have *devExt t n nid* n_s $(n'-n_s)$ = *devExt t n nid* $(n_s+(n'-n_s))$ *0* **using** *devExt-shift* **by** *auto*
moreover from *assms(2)* **have** $n_s+(n'-n_s) = n'$ **by** *simp*
ultimately have *devExt t n nid* n_s $(n'-n_s)$ = *devExt t n nid n′ 0* **by** *simp*
with *assms(1)* **show** *?thesis* **by** *simp*
qed

lemma *his-bc-empty*:
assumes $(n',nid') \in$ *his t n nid* **and** $\neg(\exists\, n''<n'.\ \exists\, nid''.\ (n'',nid'') \in$ *his t n nid*$)$
shows *bc* $(\sigma_{nid'}(t\ n'))$ = []
proof −
have ¬ $(\exists\, x.$ *his-prop t n nid n′ nid′ x*$)$
proof (*rule ccontr*)
assume ¬ ¬ $(\exists\, x.$ *his-prop t n nid n′ nid′ x*$)$
hence $\exists\, x.$ *his-prop t n nid n′ nid′ x* **by** *simp*
with ‹$(n',nid') \in$ *his t n nid*› **have** (*SOME x. his-prop t n nid n′ nid′ x*) ∈ *his t n nid*
using *his.intros* **by** *simp*
moreover from ‹$\exists\, x.$ *his-prop t n nid n′ nid′ x*› **have** *his-prop t n nid n′ nid′* (*SOME x. his-prop t n nid n′ nid′ x*)
using *someI-ex*[*of* $\lambda x.$ *his-prop t n nid n′ nid′ x*] **by** *auto*
hence $(\exists\, n.$ *latestAct-cond nid′ t n′ n*$) \wedge$ *fst* (*SOME x. his-prop t n nid n′ nid′ x*) = $\langle nid' \leftarrow t \rangle_{n'}$
by *force*
hence *fst* (*SOME x. his-prop t n nid n′ nid′ x*) < *n′* **using** *latestAct-prop*(*2*)[*of n′ nid′ t*] **by** *force*
ultimately have *fst* (*SOME x. his-prop t n nid n′ nid′ x*)<*n′* ∧
(*fst* (*SOME x. his-prop t n nid n′ nid′ x*),*snd* (*SOME x. his-prop t n nid n′ nid′ x*))∈ *his t n nid*
by *simp*
thus *False* **using** *assms(2)* **by** *blast*
qed
hence $\forall\, x.$ ¬ $(\exists\, n.$ *latestAct-cond nid′ t n′ n*$)$ ∨ ¬ $\|$*snd x*$\|_{t\ (fst\ x)}$ ∨ ¬ *fst x* = $\langle nid' \leftarrow t \rangle_{n'}$ ∨ ¬ (*prefix* (*bc* $(\sigma_{nid'}(t\ n'))$) (*bc* $(\sigma_{snd\ x}(t\ (fst\ x))))$ ∨ $(\exists\, b.$ *bc* $(\sigma_{nid'}(t\ n'))$ = (*bc* $(\sigma_{snd\ x}(t\ (fst\ x))))$ @ [*b*] ∧ *mining* $(\sigma_{nid'}(t\ n'))))$ **by** *auto*
hence ¬ $(\exists\, n.$ *latestAct-cond nid′ t n′ n*$)$ ∨ $(\exists\, n.$ *latestAct-cond nid′ t n′ n*$)$ ∧ $(\forall\, x.$ ¬ $\|$*snd x*$\|_{t\ (fst\ x)}$ ∨ ¬ *fst x* = $\langle nid' \leftarrow t \rangle_{n'}$ ∨ ¬ (*prefix* (*bc* $(\sigma_{nid'}(t\ n'))$) (*bc* $(\sigma_{snd\ x}(t\ (fst\ x))))$ ∨ $(\exists\, b.$ *bc* $(\sigma_{nid'}(t\ n'))$ = (*bc* $(\sigma_{snd\ x}(t\ (fst\ x))))$ @ [*b*] ∧ *mining* $(\sigma_{nid'}(t\ n')))))$ **by** *auto*
thus *?thesis*
proof
assume ¬ $(\exists\, n.$ *latestAct-cond nid′ t n′ n*$)$
moreover from *assms(1)* **have** $\|nid'\|_{t\ n'}$ **using** *his-act* **by** *simp*
ultimately show *?thesis* **using** *init-model* **by** *simp*
next
assume $(\exists\, n.$ *latestAct-cond nid′ t n′ n*$) \wedge (\forall\, x.$ ¬ $\|$*snd x*$\|_{t\ (fst\ x)}$ ∨ ¬ *fst x* = $\langle nid' \leftarrow t \rangle_{n'}$ ∨ ¬ (*prefix* (*bc* $(\sigma_{nid'}(t\ n'))$) (*bc* $(\sigma_{snd\ x}(t\ (fst\ x))))$ ∨ $(\exists\, b.$ *bc* $(\sigma_{nid'}(t\ n'))$ = (*bc* $(\sigma_{snd\ x}(t\ (fst\ x))))$ @ [*b*] ∧ *mining* $(\sigma_{nid'}(t\ n')))))$
hence $\exists\, n.$ *latestAct-cond nid′ t n′ n* **and** $\forall\, x.$ ¬ $\|$*snd x*$\|_{t\ (fst\ x)}$ ∨ ¬ *fst x* = $\langle nid' \leftarrow t \rangle_{n'}$ ∨ ¬ (*prefix* (*bc* $(\sigma_{nid'}(t\ n'))$) (*bc* $(\sigma_{snd\ x}(t\ (fst\ x))))$ ∨ $(\exists\, b.$ *bc* $(\sigma_{nid'}(t\ n'))$ = (*bc* $(\sigma_{snd\ x}(t\ (fst\ x))))$ @ [*b*] ∧ *mining* $(\sigma_{nid'}(t\ n'))))$ **by** *auto*
hence *asmp*: $\forall\, x.$ $\|$*snd x*$\|_{t\ (fst\ x)} \longrightarrow$ *fst x* = $\langle nid' \leftarrow t \rangle_{n'} \longrightarrow$ ¬ (*prefix* (*bc* $(\sigma_{nid'}(t\ n'))$) (*bc* $(\sigma_{snd\ x}(t\ (fst\ x))))$ ∨ $(\exists\, b.$ *bc* $(\sigma_{nid'}(t\ n'))$ = (*bc* $(\sigma_{snd\ x}(t\ (fst\ x))))$ @ [*b*] ∧ *mining* $(\sigma_{nid'}(t\ n')))$ **by** *auto*

show *?thesis*
proof *cases*
assume *honest nid′*
moreover from *assms(1)* **have** $\|nid'\|_{t\ n'}$ **using** *his-act* **by** *simp*
ultimately obtain *nid″* **where** $\|nid''\|_{t\ \langle nid' \leftarrow t\rangle_{n'}}$ **and** *mining* $(\sigma_{nid'} t\ n') \wedge (\exists\, b.\ bc\ (\sigma_{nid'} t\ n') = bc\ (\sigma_{nid''} t\ \langle nid' \leftarrow t\rangle_{n'})\ @\ [b]) \vee \neg\ mining\ (\sigma_{nid'} t\ n') \wedge bc\ (\sigma_{nid'} t\ n') = bc\ (\sigma_{nid''} t\ \langle nid' \leftarrow t\rangle_{n'})$ **using** $\langle \exists\, n.\ latestAct\text{-}cond\ nid'\ t\ n'\ n \rangle\ bhv\text{-}hn\text{-}context[of\ nid'\ t\ n']$ **by** *auto*
moreover from $\langle \|nid''\|_{t\ \langle nid' \leftarrow t\rangle_{n'}} \rangle$ **have** $\neg\ (prefix\ (bc\ (\sigma_{nid'}(t\ n')))\ (bc\ (\sigma_{nid''}(t\ (\langle nid' \leftarrow t\rangle_{n'})))) \vee (\exists\, b.\ bc\ (\sigma_{nid'}(t\ n')) = (bc\ (\sigma_{nid''}(t\ (\langle nid' \leftarrow t\rangle_{n'})))) @ [b] \wedge mining\ (\sigma_{nid'}(t\ n'))))$ **using** *asmp* **by** *auto*
ultimately have *False* **by** *auto*
thus *?thesis* **..**
next
assume $\neg$ *honest nid′*
moreover from *assms(1)* **have** $\|nid'\|_{t\ n'}$ **using** *his-act* **by** *simp*
ultimately obtain *nid″* **where** $\|nid''\|_{t\ \langle nid' \leftarrow t\rangle_{n'}}$ **and** $(mining\ (\sigma_{nid'} t\ n') \wedge (\exists\, b.\ prefix\ (bc\ (\sigma_{nid'} t\ n'))\ (bc\ (\sigma_{nid''} t\ \langle nid' \leftarrow t\rangle_{n'})\ @\ [b])) \vee \neg\ mining\ (\sigma_{nid'} t\ n') \wedge prefix\ (bc\ (\sigma_{nid'} t\ n'))\ (bc\ (\sigma_{nid''} t\ \langle nid' \leftarrow t\rangle_{n'}))$ **using** $\langle \exists\, n.\ latestAct\text{-}cond\ nid'\ t\ n'\ n \rangle\ bhv\text{-}dn\text{-}context[of\ nid'\ t\ n']$ **by** *auto*
moreover from $\langle \|nid''\|_{t\ \langle nid' \leftarrow t\rangle_{n'}} \rangle$ **have** $\neg\ (prefix\ (bc\ (\sigma_{nid'}(t\ n')))\ (bc\ (\sigma_{nid''}(t\ (\langle nid' \leftarrow t\rangle_{n'})))) \vee (\exists\, b.\ bc\ (\sigma_{nid'}(t\ n')) = (bc\ (\sigma_{nid''}(t\ (\langle nid' \leftarrow t\rangle_{n'})))) @ [b] \wedge mining\ (\sigma_{nid'}(t\ n'))))$ **using** *asmp* **by** *auto*
ultimately have *False* **by** *auto*
thus *?thesis* **..**
qed
qed
qed

lemma *devExt-devop*:
$prefix\ (devExt\ t\ n\ nid\ n_s\ (Suc\ n'))\ (devExt\ t\ n\ nid\ n_s\ n') \vee (\exists\, b.\ devExt\ t\ n\ nid\ n_s\ (Suc\ n') = devExt\ t\ n\ nid\ n_s\ n'\ @\ [b]) \wedge \neg\ Option.is\text{-}none\ (devBC\ t\ n\ nid\ (n_s + Suc\ n')) \wedge \|the\ (devBC\ t\ n\ nid\ (n_s + Suc\ n'))\|_{t\ (n_s + Suc\ n')} \wedge n_s + Suc\ n' \leq n \wedge mining\ (\sigma_{the\ (devBC\ t\ n\ nid\ (n_s + Suc\ n'))}(t\ (n_s + Suc\ n')))$
proof *cases*
assume $n_s + Suc\ n' > n$
hence $\neg(\exists\, nid'.\ (n_s + Suc\ n', nid') \in his\ t\ n\ nid)$ **using** *his-le* **by** *fastforce*
hence *Option.is-none* $(devBC\ t\ n\ nid\ (n_s + Suc\ n'))$ **using** *devBC-def* **by** *simp*
hence $devExt\ t\ n\ nid\ n_s\ (Suc\ n') = devExt\ t\ n\ nid\ n_s\ n'$ **by** *simp*
thus *?thesis* **by** *simp*
next
assume $\neg n_s + Suc\ n' > n$
hence $n_s + Suc\ n' \leq n$ **by** *simp*
show *?thesis*
proof *cases*
assume *Option.is-none* $(devBC\ t\ n\ nid\ (n_s + Suc\ n'))$
hence $devExt\ t\ n\ nid\ n_s\ (Suc\ n') = devExt\ t\ n\ nid\ n_s\ n'$ **by** *simp*
thus *?thesis* **by** *simp*
next
assume $\neg$ *Option.is-none* $(devBC\ t\ n\ nid\ (n_s + Suc\ n'))$
hence $devExt\ t\ n\ nid\ n_s\ (Suc\ n') = bc\ (\sigma_{the\ (devBC\ t\ n\ nid\ (n_s + Suc\ n'))}(t\ (n_s + Suc\ n')))$ **by** *simp*
moreover have $prefix\ (bc\ (\sigma_{the\ (devBC\ t\ n\ nid\ (n_s + Suc\ n'))}(t\ (n_s + Suc\ n'))))\ (devExt\ t\ n\ nid\ n_s\ n') \vee (\exists\, b.\ bc\ (\sigma_{the\ (devBC\ t\ n\ nid\ (n_s + Suc\ n'))}(t\ (n_s + Suc\ n'))) = devExt\ t\ n\ nid\ n_s\ n'\ @\ [b] \wedge \neg\ Option.is\text{-}none\ (devBC\ t\ n\ nid\ (n_s + Suc\ n')) \wedge \|the\ (devBC\ t\ n\ nid\ (n_s + Suc\ n'))\|_{t\ (n_s + Suc\ n')} \wedge n_s + Suc\ n' \leq n \wedge mining\ (\sigma_{the\ (devBC\ t\ n\ nid\ (n_s + Suc\ n'))}(t\ (n_s + Suc\ n'))))$
proof *cases*

assume $\exists n''<n_s + Suc\ n'.\ \exists nid'.\ (n'',nid')\in his\ t\ n\ nid$
let *?nid*=(*THE nid'*. $(n_s + Suc\ n',nid')\in his\ t\ n\ nid$)
let *?x*=*SOME x. his-prop t n nid* $(n_s + Suc\ n')$ *?nid x*
from ‹¬ *Option.is-none* (*devBC t n nid* $(n_s + Suc\ n')$)›
have $n_s + Suc\ n'\leq n$ **using** *devExt-nopt-leq* **by** *simp*
moreover from ‹¬ *Option.is-none* (*devBC t n nid* $(n_s + Suc\ n')$)›
have $\exists nid'.\ (n_s + Suc\ n',nid')\in his\ t\ n\ nid$ **using** *his-ex* **by** *simp*
ultimately have $\exists x.$ *his-prop t n nid* $(n_s + Suc\ n')$ (*THE nid'*. $((n_s + Suc\ n'),nid')\in his\ t\ n\ nid)$ *x*
and (*hisPred t n nid* $(n_s + Suc\ n')$, (*SOME nid'*. (*hisPred t n nid* $(n_s + Suc\ n')$, *nid'*) ∈ *his t n nid*)) = *?x*
using ‹$\exists n''<n_s + Suc\ n'.\ \exists nid'.\ (n'',nid')\in his\ t\ n\ nid$›
his-determ-ext[*of* $n_s + Suc\ n'$ *n t nid*] **by** *auto*
moreover have *bc* ($\sigma_{(SOME\ nid'.\ (hisPred\ t\ n\ nid\ (n_s + Suc\ n'),\ nid')\ \in\ his\ t\ n\ nid)}$(*t* (*hisPred t n nid* $(n_s + Suc\ n')$))) = *devExt t n nid* $n_s\ n'$
proof *cases*
assume *Option.is-none* (*devBC t n nid* (n_s+n'))
have *devExt t n nid* $n_s\ n'$ = *bc* ($\sigma_{the\ (devBC\ t\ n\ nid\ (GREATEST\ n''.\ n''<n_s+n'\ \wedge\ \neg Option.is\text{-}none\ (devBC\ t\ n\ nid\ n'')}$ (*GREATEST n''*. $n''<n_s+n' \wedge \neg$*Option.is-none* (*devBC t n nid n''*))))
proof *cases*
assume $n'=0$
moreover have $\exists n''<n_s+n'.\ \neg$*Option.is-none* (*devBC t n nid n''*)
proof −
from ‹$\exists n''<n_s + Suc\ n'.\ \exists nid'.\ (n'',nid')\in his\ t\ n\ nid$› **obtain** *n''*
where $n''<Suc\ n_s + n'$ **and** $\exists nid'.\ (n'',nid')\in his\ t\ n\ nid$ **by** *auto*
hence ¬ *Option.is-none* (*devBC t n nid n''*) **using** *devBC-def* **by** *simp*
moreover from ‹¬ *Option.is-none* (*devBC t n nid n''*)›
‹*Option.is-none* (*devBC t n nid* (n_s+n'))› **have** ¬ $n''=n_s+n'$ **by** *auto*
with ‹$n''<Suc\ n_s+n'$› **have** $n''<n_s+n'$ **by** *simp*
ultimately show *?thesis* **by** *auto*
qed
ultimately show *?thesis* **using** ‹*Option.is-none* (*devBC t n nid* (n_s+n'))› **by** *simp*
next
assume ¬ $n'=0$
moreover have $\exists n''<n_s + n'.$ ¬ *Option.is-none* (*devBC t n nid n''*)
proof −
from ‹$\exists n''<n_s + Suc\ n'.\ \exists nid'.\ (n'',nid')\in his\ t\ n\ nid$› **obtain** *n''*
where $n''<Suc\ n_s + n'$ **and** $\exists nid'.\ (n'',nid')\in his\ t\ n\ nid$ **by** *auto*
hence ¬ *Option.is-none* (*devBC t n nid n''*) **using** *devBC-def* **by** *simp*
moreover from ‹¬ *Option.is-none* (*devBC t n nid n''*)› ‹*Option.is-none* (*devBC t n nid* (n_s+n'))›
have ¬ $n''=n_s+n'$ **by** *auto*
with ‹$n''<Suc\ n_s+n'$› **have** $n''<n_s+n'$ **by** *simp*
ultimately show *?thesis* **by** *auto*
qed
with ‹¬ $(n'=0)$› ‹*Option.is-none* (*devBC t n nid* (n_s+n'))› **show** *?thesis*
using *devExt-greatest*[*of* n_s n' *t n nid*] **by** *simp*
qed
moreover have (*GREATEST n''*. $n''<n_s+n' \wedge \neg$*Option.is-none* (*devBC t n nid n''*))=*hisPred t n nid* $(n_s + Suc\ n')$
proof −
have $(\lambda n''.\ n''<n_s+n' \wedge \neg$*Option.is-none* (*devBC t n nid n''*)) = $(\lambda n''.\ \exists nid'.\ (n'',nid')\in his\ t\ n\ nid \wedge n'' < n_s + Suc\ n')$
proof
fix *n''*

show $(n'' < n_s + n' \wedge \neg$ *Option.is-none* $(devBC\ t\ n\ nid\ n'')) = (\exists\, nid'.\ (n'',\ nid') \in his\ t\ n\ nid \wedge n'' < n_s + Suc\ n')$
proof
assume $n'' < n_s + n' \wedge \neg$ *Option.is-none* $(devBC\ t\ n\ nid\ n'')$
thus $(\exists\, nid'.\ (n'',\ nid') \in his\ t\ n\ nid \wedge n'' < n_s + Suc\ n')$ **using** *his-ex* **by** *simp*
next
assume $(\exists\, nid'.\ (n'',\ nid') \in his\ t\ n\ nid \wedge n'' < n_s + Suc\ n')$
hence $\exists\, nid'.\ (n'',\ nid') \in his\ t\ n\ nid$ **and** $n'' < n_s + Suc\ n'$ **by** *auto*
hence $\neg$ *Option.is-none* $(devBC\ t\ n\ nid\ n'')$ **using** *devBC-def* **by** *simp*
moreover from ‹$\neg$ *Option.is-none* $(devBC\ t\ n\ nid\ n'')$› ‹*Option.is-none* $(devBC\ t\ n\ nid\ (n_s+n'))$›
have $n''\neq n_s+n'$ **by** *auto*
with ‹$n'' < n_s + Suc\ n'$› **have** $n'' < n_s + n'$ **by** *simp*
ultimately show $n'' < n_s + n' \wedge \neg$ *Option.is-none* $(devBC\ t\ n\ nid\ n'')$ **by** *simp*
qed
qed
hence $(GREATEST\ n''.\ n''<n_s+n' \wedge \neg$*Option.is-none* $(devBC\ t\ n\ nid\ n'')) = (GREATEST\ n''.\ \exists\, nid'.\ (n'',nid')\in his\ t\ n\ nid \wedge n'' < n_s + Suc\ n')$ **using** *arg-cong*[*of* $\lambda n''.\ n''<n_s+n' \wedge \neg$*Option.is-none* $(devBC\ t\ n\ nid\ n'')\ (\lambda n''.\ \exists\, nid'.\ (n'',nid')\in his\ t\ n\ nid \wedge n'' < n_s + Suc\ n')$] **by** *simp*
with *hisPred-def* **show** *?thesis* **by** *simp*
qed
moreover have *the* $(devBC\ t\ n\ nid\ (hisPred\ t\ n\ nid\ (n_s + Suc\ n'))) = (SOME\ nid'.\ (hisPred\ t\ n\ nid\ (n_s + Suc\ n'),\ nid') \in his\ t\ n\ nid)$
proof −
from ‹$\exists\, n''<n_s + Suc\ n'.\ \exists\, nid'.\ (n'',nid')\in his\ t\ n\ nid$›
have $\exists\, nid'.\ (hisPred\ t\ n\ nid\ (n_s + Suc\ n'),\ nid')\in his\ t\ n\ nid$
using *hisPrev-prop*(*2*) **by** *simp*
hence *the* $(devBC\ t\ n\ nid\ (hisPred\ t\ n\ nid\ (n_s + Suc\ n'))) = (THE\ nid'.\ (hisPred\ t\ n\ nid\ (n_s + Suc\ n'),\ nid')\in his\ t\ n\ nid)$
using *devBC-def* **by** *simp*
moreover from ‹$\exists\, nid'.\ (hisPred\ t\ n\ nid\ (n_s + Suc\ n'),\ nid')\in his\ t\ n\ nid$›
have $(hisPred\ t\ n\ nid\ (n_s + Suc\ n'),\ SOME\ nid'.\ (hisPred\ t\ n\ nid\ (n_s + Suc\ n'),\ nid') \in his\ t\ n\ nid) \in his\ t\ n\ nid$
using *someI-ex*[*of* $\lambda nid'.\ (hisPred\ t\ n\ nid\ (n_s + Suc\ n'),\ nid')\in his\ t\ n\ nid$] **by** *simp*
hence $(THE\ nid'.\ (hisPred\ t\ n\ nid\ (n_s + Suc\ n'),\ nid')\in his\ t\ n\ nid) = (SOME\ nid'.\ (hisPred\ t\ n\ nid\ (n_s + Suc\ n'),\ nid') \in his\ t\ n\ nid)$
using *his-determ-the* **by** *simp*
ultimately show *?thesis* **by** *simp*
qed
ultimately show *?thesis* **by** *simp*
next
assume $\neg$ *Option.is-none* $(devBC\ t\ n\ nid\ (n_s+n'))$
hence $devExt\ t\ n\ nid\ n_s\ n' = bc\ (\sigma_{the\ (devBC\ t\ n\ nid\ (n_s+n'))}(t\ (n_s+n')))$
proof *cases*
assume $n'=0$
with ‹$\neg$ *Option.is-none* $(devBC\ t\ n\ nid\ (n_s+n'))$› **show** *?thesis* **by** *simp*
next
assume $\neg\ n'=0$
hence $\exists\, nat.\ n' = Suc\ nat$ **by** *presburger*
then obtain *nat* **where** $n' = Suc\ nat$ **by** *auto*
with ‹$\neg$ *Option.is-none* $(devBC\ t\ n\ nid\ (n_s+n'))$› **have** $devExt\ t\ n\ nid\ n_s\ (Suc\ nat) = bc\ (\sigma_{the\ (devBC\ t\ n\ nid\ (n_s + Suc\ nat))}(t\ (n_s + Suc\ nat)))$ **by** *simp*
with ‹$n' = Suc\ nat$› **show** *?thesis* **by** *simp*
qed
moreover have $hisPred\ t\ n\ nid\ (n_s + Suc\ n') = n_s+n'$

proof −
have (*GREATEST* n''. $\exists$ *nid'*. (n'',*nid'*)$\in$ *his t n nid* $\wedge$ $n'' < (n_s + Suc\ n')) = n_s{+}n'$
proof (*rule Greatest-equality*)
from ‹¬ *Option.is-none* (*devBC t n nid* ($n_s{+}n'$))› **have** $\exists$ *nid'*. ($n_s + n'$, *nid'*) $\in$ *his t n nid* **using** *his-ex* **by** *simp*
thus $\exists$ *nid'*. ($n_s + n'$, *nid'*) $\in$ *his t n nid* $\wedge$ $n_s + n' < n_s + Suc\ n'$ **by** *simp*
next
fix *y* **assume** $\exists$ *nid'*. (*y*, *nid'*) $\in$ *his t n nid* $\wedge$ $y < n_s + Suc\ n'$
thus $y \leq n_s + n'$ **by** *simp*
qed
thus *?thesis* **using** *hisPred-def* **by** *simp*
qed
moreover have *the* (*devBC t n nid* (*hisPred t n nid* ($n_s + Suc\ n'$)))=(*SOME nid'*. (*hisPred t n nid* ($n_s + Suc\ n'$), *nid'*) $\in$ *his t n nid*)
proof −
from ‹$\exists\, n''{<}n_s + Suc\ n'$. $\exists$ *nid'*. (n'',*nid'*)$\in$ *his t n nid*›
have $\exists$ *nid'*. (*hisPred t n nid* ($n_s + Suc\ n'$), *nid'*)$\in$ *his t n nid*
using *hisPrev-prop*(*2*) **by** *simp*
hence *the* (*devBC t n nid* (*hisPred t n nid* ($n_s + Suc\ n'$))) = (*THE nid'*. (*hisPred t n nid* (n_s + *Suc n'*), *nid'*)$\in$*his t n nid*)
using *devBC-def* **by** *simp*
moreover from ‹$\exists$ *nid'*. (*hisPred t n nid* ($n_s + Suc\ n'$), *nid'*)$\in$ *his t n nid*›
have (*hisPred t n nid* ($n_s + Suc\ n'$), *SOME nid'*. (*hisPred t n nid* ($n_s + Suc\ n'$), *nid'*) $\in$ *his t n nid*) $\in$ *his t n nid*
using *someI-ex*[*of* λ*nid'*. (*hisPred t n nid* ($n_s + Suc\ n'$), *nid'*)$\in$*his t n nid*] **by** *simp*
hence (*THE nid'*. (*hisPred t n nid* ($n_s + Suc\ n'$), *nid'*)$\in$*his t n nid*) = (*SOME nid'*. (*hisPred t n nid* ($n_s + Suc\ n'$), *nid'*) $\in$ *his t n nid*)
using *his-determ-the* **by** *simp*
ultimately show *?thesis* **by** *simp*
qed
ultimately show *?thesis* **by** *simp*
qed
ultimately have *bc* ($\sigma_{snd\ ?x}$(*t* (*fst ?x*))) = *devExt t n nid* n_s n'
using *fst-conv*[*of hisPred t n nid* ($n_s + Suc\ n'$)
(*SOME nid'*. (*hisPred t n nid* ($n_s + Suc\ n'$), *nid'*) $\in$ *his t n nid*)]
snd-conv[*of hisPred t n nid* ($n_s + Suc\ n'$)
(*SOME nid'*. (*hisPred t n nid* ($n_s + Suc\ n'$), *nid'*) $\in$ *his t n nid*)] **by** *simp*
moreover from ‹$\exists\, x$. *his-prop t n nid* ($n_s + Suc\ n'$) *?nid x*›
have *his-prop t n nid* ($n_s + Suc\ n'$) *?nid ?x*
using *someI-ex*[*of* λx. *his-prop t n nid* ($n_s + Suc\ n'$) *?nid x*] **by** *blast*
hence *prefix* (*bc* ($\sigma_{?nid}$(*t* ($n_s + Suc\ n'$)))) (*bc* ($\sigma_{snd\ ?x}$(*t* (*fst ?x*)))) $\vee$ ($\exists\, b$. *bc* ($\sigma_{?nid}$(*t* (n_s + *Suc n'*))) = (*bc* ($\sigma_{snd\ ?x}$(*t* (*fst ?x*)))) @ [*b*] $\wedge$ *mining* ($\sigma_{?nid}$(*t* ($n_s + Suc\ n'$)))) **by** *blast*
ultimately have *prefix* (*bc* ($\sigma_{?nid}$(*t* ($n_s + Suc\ n'$)))) (*devExt t n nid* n_s n') $\vee$ ($\exists\, b$. *bc* ($\sigma_{?nid}$(*t* (n_s + *Suc n'*))) = (*devExt t n nid* n_s n') @ [*b*] $\wedge$ *mining* ($\sigma_{?nid}$(*t* ($n_s + Suc\ n'$)))) **by** *simp*
moreover from ‹$\exists$ *nid'*. ($n_s + Suc\ n'$,*nid'*)$\in$ *his t n nid*›
have *?nid*=*the* (*devBC t n nid* ($n_s + Suc\ n'$)) **using** *devBC-def* **by** *simp*
moreover have $\|$*the* (*devBC t n nid* ($n_s + Suc\ n'$))$\|_{t\ (n_s + Suc\ n')}$
proof −
from ‹$\exists$ *nid'*. ($n_s + Suc\ n'$,*nid'*)$\in$*his t n nid*› **obtain** *nid'*
where ($n_s + Suc\ n'$,*nid'*)$\in$*his t n nid* **by** *auto*
with *his-determ-the* **have** *nid'* = (*THE nid'*. ($n_s + Suc\ n'$, *nid'*) $\in$ *his t n nid*) **by** *simp*
with ‹*?nid*=*the* (*devBC t n nid* ($n_s + Suc\ n'$))›
have *the* (*devBC t n nid* ($n_s + Suc\ n'$)) = *nid'* **by** *simp*
with ‹($n_s + Suc\ n'$,*nid'*)$\in$*his t n nid*› **show** *?thesis* **using** *his-act* **by** *simp*
qed

```
      ultimately show ?thesis
        using ‹¬ Option.is-none (devBC t n nid (n_s+Suc n'))› ‹n_s + Suc n' ≤ n› by simp
    next
      assume ¬ (∃ n''<n_s + Suc n'. ∃ nid'. (n'',nid')∈ his t n nid)
      moreover have (n_s + Suc n', the (devBC t n nid (n_s + Suc n'))) ∈ his t n nid
      proof −
        from ‹¬ Option.is-none (devBC t n nid (n_s + Suc n'))›
          have ∃ nid'. (n_s + Suc n',nid')∈his t n nid using his-ex by simp
        hence the (devBC t n nid (n_s + Suc n')) = (THE nid'. (n_s + Suc n', nid') ∈ his t n nid)
          using devBC-def by simp
        moreover from ‹∃ nid'. (n_s + Suc n',nid')∈his t n nid› obtain nid'
          where (n_s + Suc n',nid')∈his t n nid by auto
        with his-determ-the have nid' = (THE nid'. (n_s + Suc n', nid') ∈ his t n nid) by simp
        ultimately have the (devBC t n nid (n_s + Suc n')) = nid' by simp
        with ‹(n_s + Suc n',nid')∈his t n nid› show ?thesis by simp
      qed
      ultimately have bc (σ_the (devBC t n nid (n_s + Suc n'))(t (n_s + Suc n'))) = []
        using his-bc-empty by simp
      thus ?thesis by simp
    qed
    ultimately show ?thesis by simp
  qed
qed

abbreviation devLgthBC where devLgthBC t n nid n_s ≡ (λn'. length (devExt t n nid n_s n'))

theorem blockchain-save:
  fixes t::nat⇒cnf and n_s and sbc and n
  assumes ∀ nid. honest nid ⟶ prefix sbc (bc (σ_nid(t (⟨nid → t⟩_ns))))
    and ∀ nid∈actDn (t n_s). length (bc (σ_nid(t n_s))) < length sbc
    and PoW t n_s≥length sbc + cb
    and ∀ n'<n_s. ∀ nid. ‖nid‖_t n' ⟶ length (bc (σ_nid t n')) < length sbc ∨ prefix sbc (bc (σ_nid(t n')))
    and n≥n_s
  shows ∀ nid ∈ actHn (t n). prefix sbc (bc (σ_nid(t n)))
proof (cases)
  assume sbc=[]
  thus ?thesis by simp
next
  assume ¬ sbc=[]
  have n≥n_s ⟹ ∀ nid ∈ actHn (t n). prefix sbc (bc (σ_nid(t n)))
  proof (induction n rule: ge-induct)
    case (step n)
    show ?case
    proof
      fix nid assume nid ∈ actHn (t n)
      hence ‖nid‖_t n and honest nid using actHn-def by auto
      show prefix sbc (bc (σ_nid t n))
      proof cases
        assume lAct: ∃ n' < n. n' ≥ n_s ∧ ‖nid‖_t n'
        show ?thesis
        proof cases
          assume ∃ b∈pin (σ_nid t ⟨nid ← t⟩n). length b > length (bc (σ_nid t ⟨nid ← t⟩n))
          moreover from ‹‖nid‖_t n› have ∃ n'≥n. ‖nid‖_t n' by auto
          moreover from lAct have ∃ n'. latestAct-cond nid t n n' by auto
           ultimately have ¬ mining (σ_nid t ⟨nid → t⟩n) ∧ bc (σ_nid t ⟨nid → t⟩n) = MAX (pin (σ_nid t
```

$\langle nid \leftarrow t\rangle_n)) \vee$
mining $(\sigma_{nid} t\ \langle nid \rightarrow t\rangle_n) \wedge (\exists\, b.\ bc\ (\sigma_{nid} t\ \langle nid \rightarrow t\rangle_n) = MAX\ (pin\ (\sigma_{nid} t\ \langle nid \leftarrow t\rangle_n))\ @\ [b])$
using ‹*honest nid*› *bhv-hn-ex*[*of nid n t*] **by** *simp*
moreover have *prefix sbc* (*MAX* (*pin* $(\sigma_{nid} t\ \langle nid \leftarrow t\rangle_n)$))
proof −
from ‹$\exists\, n'$. *latestAct-cond nid t n n'*› **have** $\|nid\|_{t\ \langle nid \leftarrow t\rangle_n}$
using *latestAct-prop*(*1*) **by** *simp*
hence *pin* $(\sigma_{nid} t\ \langle nid \leftarrow t\rangle_n) \neq \{\}$ **and** *finite* (*pin* $(\sigma_{nid} t\ \langle nid \leftarrow t\rangle_n)$)
using *nempty-input*[*of nid t* $\langle nid \leftarrow t\rangle_n$] *finite-input*[*of nid t* $\langle nid \leftarrow t\rangle_n$] ‹*honest nid*› **by** *auto*
hence *MAX* (*pin* $(\sigma_{nid} t\ \langle nid \leftarrow t\rangle_n)) \in$ *pin* $(\sigma_{nid} t\ \langle nid \leftarrow t\rangle_n)$ **using** *max-prop*(*1*) **by** *auto*
with ‹$\|nid\|_{t\ \langle nid \leftarrow t\rangle_n}$› **obtain** nid' **where** $\|nid'\|_{t\ \langle nid \leftarrow t\rangle_n}$
and *bc* $(\sigma_{nid'}(t\ \langle nid \leftarrow t\rangle_n)) = MAX\ (pin\ (\sigma_{nid} t\ \langle nid \leftarrow t\rangle_n))$
using *closed*[**where** *b*=*MAX* (*pin* $(\sigma_{nid} t\ \langle nid \leftarrow t\rangle_n))$] **by** *blast*
moreover have *prefix sbc* (*bc* $(\sigma_{nid'}(t\ \langle nid \leftarrow t\rangle_n))$)
proof *cases*
assume *honest nid'*
with ‹$\|nid'\|_{t\ \langle nid \leftarrow t\rangle_n}$› **have** $nid' \in$ *actHn* $(t\ \langle nid \leftarrow t\rangle_n)$
using *actHn-def* **by** *simp*
moreover from ‹$\exists\, n'$. *latestAct-cond nid t n n'*› **have** $\langle nid \leftarrow t\rangle_n < n$
using *latestAct-prop*(*2*) **by** *simp*
moreover from *lAct* **have** $\langle nid \leftarrow t\rangle_n \geq n_s$ **using** *latestActless* **by** *blast*
ultimately show *?thesis* **using** ‹$\|nid'\|_{t\ \langle nid \leftarrow t\rangle_n}$› *step.IH* **by** *simp*
next
assume ¬ *honest nid'*
show *?thesis*
proof (*rule ccontr*)
assume ¬ *prefix sbc* (*bc* $(\sigma_{nid'}(t\ \langle nid \leftarrow t\rangle_n))$)
moreover have $\exists\, n' \leq \langle nid \leftarrow t\rangle_n.\ n' \geq n_s \wedge$ *length* (*devExt t* $\langle nid \leftarrow t\rangle_n$ *nid' n' 0*) < *length sbc* $\wedge$ $(\forall\, n'' > n'.\ n'' \leq \langle nid \leftarrow t\rangle_n \wedge$ ¬ *Option.is-none* (*devBC t* $\langle nid \leftarrow t\rangle_n$ *nid' n''*) $\longrightarrow$ ¬ *honest* (*the* (*devBC t* $\langle nid \leftarrow t\rangle_n$ *nid' n''*)))
proof *cases*
assume $\exists\, n' \leq \langle nid \leftarrow t\rangle_n.\ n' \geq n_s \wedge$ ¬ *Option.is-none* (*devBC t* $\langle nid \leftarrow t\rangle_n$ *nid' n'*) $\wedge$ *honest* (*the* (*devBC t* $\langle nid \leftarrow t\rangle_n$ *nid' n'*))
hence $\exists\, n' \leq \langle nid \leftarrow t\rangle_n.\ n' \geq n_s \wedge$ ¬ *Option.is-none* (*devBC t* $\langle nid \leftarrow t\rangle_n$ *nid' n'*) $\wedge$ *honest* (*the* (*devBC t* $\langle nid \leftarrow t\rangle_n$ *nid' n'*)) $\wedge$ $(\forall\, n'' > n'.\ n'' \leq \langle nid \leftarrow t\rangle_n \wedge$ ¬ *Option.is-none* (*devBC t* $\langle nid \leftarrow t\rangle_n$ *nid' n''*) $\longrightarrow$ ¬ *honest* (*the* (*devBC t* $\langle nid \leftarrow t\rangle_n$ *nid' n''*)))
proof −
let *?P*=$\lambda n'.\ n' \leq \langle nid \leftarrow t\rangle_n \wedge n' \geq n_s\ \wedge$ ¬ *Option.is-none* (*devBC t* $\langle nid \leftarrow t\rangle_n$ *nid' n'*) $\wedge$ *honest* (*the* (*devBC t* $\langle nid \leftarrow t\rangle_n$ *nid' n'*))
from ‹$\exists\, n' \leq \langle nid \leftarrow t\rangle_n.\ n' \geq n_s\ \wedge$ ¬ *Option.is-none* (*devBC t* $\langle nid \leftarrow t\rangle_n$ *nid' n'*) $\wedge$ *honest* (*the* (*devBC t* $\langle nid \leftarrow t\rangle_n$ *nid' n'*))› **have** $\exists\, n'$. *?P n'* **by** *simp*
moreover have $\forall\, n' > \langle nid \leftarrow t\rangle_n.$ ¬ *?P n'* **by** *simp*
ultimately obtain n' **where** *?P n'* **and** $\forall\, n''$. *?P n''* $\longrightarrow n'' \leq n'$ **using** *boundedGreatest*[*of ?P* - $\langle nid \leftarrow t\rangle_n$] **by** *auto*
hence $\forall\, n'' > n'.\ n'' \leq \langle nid \leftarrow t\rangle_n \wedge$ ¬ *Option.is-none* (*devBC t* $\langle nid \leftarrow t\rangle_n$ *nid' n''*) $\longrightarrow$ ¬ *honest* (*the* (*devBC t* $\langle nid \leftarrow t\rangle_n$ *nid' n''*)) **by** *auto*
thus *?thesis* **using** ‹*?P n'*› **by** *auto*
qed
then obtain n' **where** $n' \leq \langle nid \leftarrow t\rangle_n$ **and** ¬ *Option.is-none* (*devBC t* $\langle nid \leftarrow t\rangle_n$ *nid' n'*)
and $n' \geq n_s$ **and** *honest* (*the* (*devBC t* $\langle nid \leftarrow t\rangle_n$ *nid' n'*))
and $\forall\, n'' > n'.\ n'' \leq \langle nid \leftarrow t\rangle_n \wedge$ ¬ *Option.is-none* (*devBC t* $\langle nid \leftarrow t\rangle_n$ *nid' n''*) $\longrightarrow$ ¬ *honest* (*the* (*devBC t* $\langle nid \leftarrow t\rangle_n$ *nid' n''*)) **by** *auto*

hence $n' \geq n_s$ **and** *dishonest*: $\forall\, n'' > n'.\ n'' \leq \langle nid \leftarrow t \rangle_n \wedge \neg$ *Option.is-none* (*devBC t* $\langle nid \leftarrow t \rangle_n$ *nid' n''*) $\longrightarrow \neg$ *honest* (*the* (*devBC t* $\langle nid \leftarrow t \rangle_n$ *nid' n''*)) **by** *auto*

moreover have $\langle nid \leftarrow t \rangle_n < n$ **using** ‹$\exists\, n'.$ *latestAct-cond nid t n n'*› *latestAct-prop*(*2*) **by** *blast*

with ‹$n' \leq \langle nid \leftarrow t \rangle_n$› **have** $n' < n$ **by** *simp*

moreover from ‹$\neg$ *Option.is-none* (*devBC t* $\langle nid \leftarrow t \rangle_n$ *nid' n'*)›

have $\|$*the* (*devBC t* $\langle nid \leftarrow t \rangle_n$ *nid' n'*)$\|_{t\ n'}$ **using** *devBC-act* **by** *simp*

with ‹*honest* (*the* (*devBC t* $\langle nid \leftarrow t \rangle_n$ *nid' n'*))›

have *the* (*devBC t* $\langle nid \leftarrow t \rangle_n$ *nid' n'*) $\in$*actHn* (*t n'*) **using** *actHn-def* **by** *simp*

ultimately have *prefix sbc* (*bc* ($\sigma_{the\ (devBC\ t\ \langle nid \leftarrow t \rangle_n\ nid'\ n')}\, t\ n'$))

using *step.IH* **by** *simp*

interpret *ut*: *dishonest devExt t* $\langle nid \leftarrow t \rangle_n$ *nid' n'* $\lambda n.$ *dmining t* $(n' + n)$

proof

fix n''

from *devExt-devop*[*of t* $\langle nid \leftarrow t \rangle_n$ *nid' n'*] **have** *prefix* (*devExt t* $\langle nid \leftarrow t \rangle_n$ *nid' n'* (*Suc n''*)) (*devExt t* $\langle nid \leftarrow t \rangle_n$ *nid' n' n''*) $\vee$ ($\exists\, b.$ *devExt t* $\langle nid \leftarrow t \rangle_n$ *nid' n'* (*Suc n''*) = *devExt t* $\langle nid \leftarrow t \rangle_n$ *nid' n' n''* @ [*b*]) $\wedge \neg$ *Option.is-none* (*devBC t* $\langle nid \leftarrow t \rangle_n$ *nid'* $(n' +$ *Suc n''*)) $\wedge \|$*the* (*devBC t* $\langle nid \leftarrow t \rangle_n$ *nid'* $(n' +$ *Suc n''*))$\|_{t\ (n' +\ Suc\ n'')}$ $\wedge\ n' +$ *Suc n''* $\leq \langle nid \leftarrow t \rangle_n \wedge$ *mining* ($\sigma_{the\ (devBC\ t\ \langle nid \leftarrow t \rangle_n\ nid'\ (n' +\ Suc\ n''))}\, t\ (n' +$ *Suc n''*)) **.**

thus *prefix* (*devExt t* $\langle nid \leftarrow t \rangle_n$ *nid' n'* (*Suc n''*)) (*devExt t* $\langle nid \leftarrow t \rangle_n$ *nid' n' n''*) $\vee$ ($\exists\, b.$ *devExt t* $\langle nid \leftarrow t \rangle_n$ *nid' n'* (*Suc n''*) = *devExt t* $\langle nid \leftarrow t \rangle_n$ *nid' n' n''* @ [*b*]) $\wedge$ *dmining t* $(n' +$ *Suc n''*)

proof

assume *prefix* (*devExt t* $\langle nid \leftarrow t \rangle_n$ *nid' n'* (*Suc n''*)) (*devExt t* $\langle nid \leftarrow t \rangle_n$ *nid' n' n''*)

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

next

assume ($\exists\, b.$ *devExt t* $\langle nid \leftarrow t \rangle_n$ *nid' n'* (*Suc n''*) = *devExt t* $\langle nid \leftarrow t \rangle_n$ *nid' n' n''* @ [*b*]) $\wedge \neg$ *Option.is-none* (*devBC t* $\langle nid \leftarrow t \rangle_n$ *nid'* $(n' +$ *Suc n''*)) $\wedge \|$*the* (*devBC t* $\langle nid \leftarrow t \rangle_n$ *nid'* $(n' +$ *Suc n''*))$\|_{t\ (n' +\ Suc\ n'')}$ $\wedge\ n' +$ *Suc n''* $\leq \langle nid \leftarrow t \rangle_n \wedge$ *mining* ($\sigma_{the\ (devBC\ t\ \langle nid \leftarrow t \rangle_n\ nid'\ (n' +\ Suc\ n''))}\, t$ $(n' +$ *Suc n''*))

hence $\exists\, b.$ *devExt t* $\langle nid \leftarrow t \rangle_n$ *nid' n'* (*Suc n''*) = *devExt t* $\langle nid \leftarrow t \rangle_n$ *nid' n' n''* @ [*b*] **and** $\neg$ *Option.is-none* (*devBC t* $\langle nid \leftarrow t \rangle_n$ *nid'* $(n' +$ *Suc n''*)) **and** $\|$*the* (*devBC t* $\langle nid \leftarrow t \rangle_n$ *nid'* $(n' +$ *Suc n''*))$\|_{t\ (n' +\ Suc\ n'')}$ **and** $n' +$ *Suc n''* $\leq \langle nid \leftarrow t \rangle_n$ **and** *mining* ($\sigma_{the\ (devBC\ t\ \langle nid \leftarrow t \rangle_n\ nid'\ (n' +\ Suc\ n''))}\, t\ (n' +$ *Suc n''*)) **by** *auto*

moreover from ‹$n' +$ *Suc n''* $\leq \langle nid \leftarrow t \rangle_n$› ‹$\neg$ *Option.is-none* (*devBC t* $\langle nid \leftarrow t \rangle_n$ *nid'* $(n' +$ *Suc n''*))› **have** $\neg$ *honest* (*the* (*devBC t* $\langle nid \leftarrow t \rangle_n$ *nid'* $(n' +$ *Suc n''*))) **using** *dishonest* **by** *simp*

with ‹$\|$*the* (*devBC t* $\langle nid \leftarrow t \rangle_n$ *nid'* $(n' +$ *Suc n''*))$\|_{t\ (n' +\ Suc\ n'')}$› **have** *the* (*devBC t* $\langle nid \leftarrow t \rangle_n$ *nid'* $(n' +$ *Suc n''*))$\in$*actDn* (*t* $(n' +$ *Suc n''*)) **using** *actDn-def* **by** *simp*

ultimately show *?thesis* **using** *dmining-def* **by** *auto*

qed

qed

from ‹$\neg$ *Option.is-none* (*devBC t* $\langle nid \leftarrow t \rangle_n$ *nid' n'*)› **have** *bc* ($\sigma_{the\ (devBC\ t\ \langle nid \leftarrow t \rangle_n\ nid'\ n')}\, t$ n') = *devExt t* $\langle nid \leftarrow t \rangle_n$ *nid' n' 0*

using *devExt-bc-geq*[*of t* $\langle nid \leftarrow t \rangle_n$ *nid' n'*] **by** *simp*

moreover from ‹$n' \leq \langle nid \leftarrow t \rangle_n$› ‹$\|$*nid'*$\|_{t\ \langle nid \leftarrow t \rangle_n}$› **have** *bc* ($\sigma_{nid'}\, t\ \langle nid \leftarrow t \rangle_n$) = *devExt t* $\langle nid \leftarrow t \rangle_n$ *nid' n'* ($\langle nid \leftarrow t \rangle_n - n'$)

using *devExt-bc-geq* **by** *simp*

with ‹$\neg$ *prefix sbc* (*bc* ($\sigma_{nid'}$(*t* $\langle nid \leftarrow t \rangle_n$)))› **have** $\neg$ *prefix sbc* (*devExt t* $\langle nid \leftarrow t \rangle_n$ *nid' n'* ($\langle nid \leftarrow t \rangle_n - n'$)) **by** *simp*

ultimately have $\exists\, n'''.\ n''' \leq \langle nid \leftarrow t \rangle_n - n' \wedge$ *length* (*devExt t* $\langle nid \leftarrow t \rangle_n$ *nid' n' n'''*) < *length sbc*

using ‹*prefix sbc* (*bc* ($\sigma_{\textit{the}\ (\textit{devBC}\ t\ \langle \textit{nid} \leftarrow t\rangle_n\ \textit{nid}'\ n')}(t\ n')$))›
ut.prefix-length[*of sbc* 0 $\langle \textit{nid} \leftarrow t\rangle_n - n'$] **by** *auto*
then obtain n_p **where** $n_p \leq \langle \textit{nid} \leftarrow t\rangle_n - n'$
and *length* (*devExt t* $\langle \textit{nid} \leftarrow t\rangle_n$ *nid*′ *n*′ n_p) < *length sbc* **by** *auto*
hence *length* (*devExt t* $\langle \textit{nid} \leftarrow t\rangle_n$ *nid*′ (n' + n_p) 0) < *length sbc* **using** *devExt-shift*[*of t* $\langle \textit{nid} \leftarrow t\rangle_n$ *nid*′ *n*′ n_p] **by** *simp*
moreover from ‹$\langle \textit{nid} \leftarrow t\rangle_n \geq n'$› ‹$n_p \leq \langle \textit{nid} \leftarrow t\rangle_n - n'$› **have** ($n'$ + n_p) ≤ $\langle \textit{nid} \leftarrow t\rangle_n$ **by** *simp*
ultimately show *?thesis* **using** ‹$n' \geq n_s$› *dishonest* **by** *auto*
next
assume ¬(∃ $n' \leq \langle \textit{nid} \leftarrow t\rangle_n$. $n' \geq n_s$ ∧ ¬ *Option.is-none* (*devBC t* $\langle \textit{nid} \leftarrow t\rangle_n$ *nid*′ n') ∧ *honest* (*the* (*devBC t* $\langle \textit{nid} \leftarrow t\rangle_n$ *nid*′ n')))
hence *cas*: ∀ $n' \leq \langle \textit{nid} \leftarrow t\rangle_n$. $n' \geq n_s$ ∧ ¬ *Option.is-none* (*devBC t* $\langle \textit{nid} \leftarrow t\rangle_n$ *nid*′ n') ⟶ ¬ *honest* (*the* (*devBC t* $\langle \textit{nid} \leftarrow t\rangle_n$ *nid*′ n')) **by** *auto*
show *?thesis*
proof *cases*
assume *Option.is-none* (*devBC t* $\langle \textit{nid} \leftarrow t\rangle_n$ *nid*′ n_s)
thus *?thesis*
proof *cases*
assume ∀ $n' < n_s$. *Option.is-none* (*devBC t* $\langle \textit{nid} \leftarrow t\rangle_n$ *nid*′ n')
with ‹*Option.is-none* (*devBC t* $\langle \textit{nid} \leftarrow t\rangle_n$ *nid*′ n_s)› **have** *devExt t* $\langle \textit{nid} \leftarrow t\rangle_n$ *nid*′ n_s 0 = [] **by** *simp*
with ‹¬ *sbc*=[]› **have** *length* (*devExt t* $\langle \textit{nid} \leftarrow t\rangle_n$ *nid*′ n_s 0) < *length sbc* **by** *simp*
moreover from *lAct* **have** $\langle \textit{nid} \leftarrow t\rangle_n \geq n_s$ **using** *latestActless* **by** *blast*
moreover from *cas* **have** ∀ $n'' > n_s$. $n'' \leq \langle \textit{nid} \leftarrow t\rangle_n$ ∧ ¬ *Option.is-none* (*devBC t* $\langle \textit{nid} \leftarrow t\rangle_n$ *nid*′ n'') ⟶ ¬ *honest* (*the* (*devBC t* $\langle \textit{nid} \leftarrow t\rangle_n$ *nid*′ n'')) **by** *simp*
ultimately show *?thesis* **by** *auto*
next
let *?P*=λn'. $n' < n_s$ ∧ ¬*Option.is-none* (*devBC t* $\langle \textit{nid} \leftarrow t\rangle_n$ *nid*′ n')
let *?n*′=*GREATEST n*′. *?P n*′
assume ¬ (∀ $n' < n_s$. *Option.is-none* (*devBC t* $\langle \textit{nid} \leftarrow t\rangle_n$ *nid*′ n'))
moreover have ∀ $n' > n_s$. ¬ *?P n*′ **by** *simp*
ultimately have *exists*: ∃ n'. *?P n*′ ∧ (∀ n''. *?P n*″ ⟶ $n'' \leq n'$)
using *boundedGreatest*[*of ?P*] **by** *blast*
hence *?P ?n*′ **using** *GreatestI-ex-nat*[*of ?P*] **by** *auto*
moreover from ‹*?P ?n*′› **have** $\|\textit{the}\ (\textit{devBC}\ t\ \langle \textit{nid} \leftarrow t\rangle_n\ \textit{nid}'\ ?n')\|_{t\ ?n'}$ **using** *devBC-act* **by** *simp*
ultimately have *length* (*bc* ($\sigma_{\textit{the}\ (\textit{devBC}\ t\ \langle \textit{nid} \leftarrow t\rangle_n\ \textit{nid}'\ ?n')}\ t\ ?n'$)) < *length sbc* ∨ *prefix sbc* (*bc* ($\sigma_{\textit{the}\ (\textit{devBC}\ t\ \langle \textit{nid} \leftarrow t\rangle_n\ \textit{nid}'\ ?n')}(t\ ?n')$)) **using** *assms*(4) **by** *simp*
thus *?thesis*
proof
assume *length* (*bc* ($\sigma_{\textit{the}\ (\textit{devBC}\ t\ \langle \textit{nid} \leftarrow t\rangle_n\ \textit{nid}'\ ?n')}\ t\ ?n'$)) < *length sbc*
moreover from *exists* **have** ¬(∃ $n' > ?n'$. *?P n*′) **using** *Greatest-ex-le-nat*[*of ?P*] **by** *simp*
moreover from ‹*?P ?n*′› **have** ∃ $n' < n_s$. ¬ *Option.is-none* (*devBC t* $\langle \textit{nid} \leftarrow t\rangle_n$ *nid*′ n') **by** *blast*
with ‹*Option.is-none* (*devBC t* $\langle \textit{nid} \leftarrow t\rangle_n$ *nid*′ n_s)›
have *devExt t* $\langle \textit{nid} \leftarrow t\rangle_n$ *nid*′ n_s 0 = *bc* ($\sigma_{\textit{the}\ (\textit{devBC}\ t\ \langle \textit{nid} \leftarrow t\rangle_n\ \textit{nid}'\ ?n')}(t\ ?n')$) **by** *simp*
ultimately have *length* (*devExt t* $\langle \textit{nid} \leftarrow t\rangle_n$ *nid*′ n_s 0) < *length sbc* **by** *simp*
moreover from *lAct* **have** $\langle \textit{nid} \leftarrow t\rangle_n \geq n_s$ **using** *latestActless* **by** *blast*
moreover from *cas* **have** ∀ $n'' > n_s$. $n'' \leq \langle \textit{nid} \leftarrow t\rangle_n$ ∧ ¬ *Option.is-none* (*devBC t* $\langle \textit{nid} \leftarrow t\rangle_n$ *nid*′ n'') ⟶ ¬ *honest* (*the* (*devBC t* $\langle \textit{nid} \leftarrow t\rangle_n$ *nid*′ n'')) **by** *simp*
ultimately show *?thesis* **by** *auto*

next
interpret *ut*: *dishonest devExt t* $\langle nid \leftarrow t\rangle_n$ *nid' n_s* λn. *dmining t* $(n_s + n)$
proof
fix n''
from *devExt-devop*[*of t* $\langle nid \leftarrow t\rangle_n$ *nid' n_s*] **have** *prefix* (*devExt t* $\langle nid \leftarrow t\rangle_n$ *nid' n_s* (*Suc n''*)) (*devExt t* $\langle nid \leftarrow t\rangle_n$ *nid' n_s n''*) ∨ (∃ *b*. *devExt t* $\langle nid \leftarrow t\rangle_n$ *nid' n_s* (*Suc n''*) = *devExt t* $\langle nid \leftarrow t\rangle_n$ *nid' n_s n''* @ [*b*]) ∧ ¬ *Option.is-none* (*devBC t* $\langle nid \leftarrow t\rangle_n$ *nid'* $(n_s + Suc\ n'')$) ∧ $\|the\ (devBC\ t\ \langle nid \leftarrow t\rangle_n\ nid'\ (n_s + Suc\ n''))\|_{t\ (n_s + Suc\ n'')}$ ∧ $n_s + Suc\ n'' \leq \langle nid \leftarrow t\rangle_n$ ∧ *mining* $(\sigma_{the\ (devBC\ t\ \langle nid \leftarrow t\rangle_n\ nid'\ (n_s + Suc\ n''))}\ t\ (n_s + Suc\ n''))$.
thus *prefix* (*devExt t* $\langle nid \leftarrow t\rangle_n$ *nid' n_s* (*Suc n''*)) (*devExt t* $\langle nid \leftarrow t\rangle_n$ *nid' n_s n''*) ∨ (∃ *b*. *devExt t* $\langle nid \leftarrow t\rangle_n$ *nid' n_s* (*Suc n''*) = *devExt t* $\langle nid \leftarrow t\rangle_n$ *nid' n_s n''* @ [*b*]) ∧ *dmining t* $(n_s + Suc\ n'')$
proof
assume *prefix* (*devExt t* $\langle nid \leftarrow t\rangle_n$ *nid' n_s* (*Suc n''*)) (*devExt t* $\langle nid \leftarrow t\rangle_n$ *nid' n_s n''*) **thus** *?thesis* **by** *simp*
next
assume (∃ *b*. *devExt t* $\langle nid \leftarrow t\rangle_n$ *nid' n_s* (*Suc n''*) = *devExt t* $\langle nid \leftarrow t\rangle_n$ *nid' n_s n''* @ [*b*]) ∧ ¬ *Option.is-none* (*devBC t* $\langle nid \leftarrow t\rangle_n$ *nid'* $(n_s + Suc\ n'')$) ∧ $\|the\ (devBC\ t\ \langle nid \leftarrow t\rangle_n\ nid'\ (n_s + Suc\ n''))\|_{t\ (n_s + Suc\ n'')}$ ∧ $n_s + Suc\ n'' \leq \langle nid \leftarrow t\rangle_n$ ∧ *mining* $(\sigma_{the\ (devBC\ t\ \langle nid \leftarrow t\rangle_n\ nid'\ (n_s + Suc\ n''))}\ t\ (n_s + Suc\ n''))$
hence ∃ *b*. *devExt t* $\langle nid \leftarrow t\rangle_n$ *nid' n_s* (*Suc n''*) = *devExt t* $\langle nid \leftarrow t\rangle_n$ *nid' n_s n''* @ [*b*]
and ¬ *Option.is-none* (*devBC t* $\langle nid \leftarrow t\rangle_n$ *nid'* $(n_s + Suc\ n'')$)
and $\|the\ (devBC\ t\ \langle nid \leftarrow t\rangle_n\ nid'\ (n_s + Suc\ n''))\|_{t\ (n_s + Suc\ n'')}$
and $n_s + Suc\ n'' \leq \langle nid \leftarrow t\rangle_n$
and *mining* $(\sigma_{the\ (devBC\ t\ \langle nid \leftarrow t\rangle_n\ nid'\ (n_s + Suc\ n''))}\ t\ (n_s + Suc\ n''))$
by *auto*
moreover from ‹$n_s + Suc\ n'' \leq \langle nid \leftarrow t\rangle_n$› ‹¬ *Option.is-none* (*devBC t* $\langle nid \leftarrow t\rangle_n$ *nid'* $(n_s + Suc\ n'')$)›
have ¬ *honest* (*the* (*devBC t* $\langle nid \leftarrow t\rangle_n$ *nid'* $(n_s + Suc\ n'')$))
using *cas* **by** *simp*
with ‹$\|the\ (devBC\ t\ \langle nid \leftarrow t\rangle_n\ nid'\ (n_s + Suc\ n''))\|_{t\ (n_s + Suc\ n'')}$›
have *the* (*devBC t* $\langle nid \leftarrow t\rangle_n$ *nid'* $(n_s + Suc\ n'')$)∈*actDn* $(t\ (n_s + Suc\ n''))$ **using** *actDn-def* **by** *simp*
ultimately show *?thesis* **using** *dmining-def* **by** *auto*
qed
qed

assume *prefix sbc* (*bc* $(\sigma_{the\ (devBC\ t\ \langle nid \leftarrow t\rangle_n\ nid'\ ?n')}(t\ ?n'))$)
moreover from *exists* **have** ¬(∃ *n'*>*?n'*. *?P n'*) **using** *Greatest-ex-le-nat*[*of ?P*] **by** *simp*
moreover from ‹*?P ?n'*› **have** ∃ *n'*<n_s. ¬ *Option.is-none* (*devBC t* $\langle nid \leftarrow t\rangle_n$ *nid' n'*) **by** *blast*
with ‹*Option.is-none* (*devBC t* $\langle nid \leftarrow t\rangle_n$ *nid' n_s*)› **have** *devExt t* $\langle nid \leftarrow t\rangle_n$ *nid' n_s 0* = *bc* $(\sigma_{the\ (devBC\ t\ \langle nid \leftarrow t\rangle_n\ nid'\ ?n')}(t\ ?n'))$ **by** *simp*
ultimately have *prefix sbc* (*devExt t* $\langle nid \leftarrow t\rangle_n$ *nid' n_s 0*) **by** *simp*
moreover from *lAct* **have** $\langle nid \leftarrow t\rangle_n \geq n_s$ **using** *latestActless* **by** *blast*
with ‹$\|nid'\|_{t\ \langle nid \leftarrow t\rangle_n}$› **have** *bc* $(\sigma_{the\ (devBC\ t\ \langle nid \leftarrow t\rangle_n\ nid'\ \langle nid \leftarrow t\rangle_n)}\ t\ \langle nid \leftarrow t\rangle_n)$ = *devExt t* $\langle nid \leftarrow t\rangle_n$ *nid' n_s* $(\langle nid \leftarrow t\rangle_n - n_s)$ **using** *devExt-bc-geq* **by** *simp*
with ‹¬ *prefix sbc* (*bc* $(\sigma_{nid'}(t\ \langle nid \leftarrow t\rangle_n))$)› ‹$\|nid'\|_{t\ \langle nid \leftarrow t\rangle_n}$› **have** ¬ *prefix sbc* (*devExt t* $\langle nid \leftarrow t\rangle_n$ *nid' n_s* $(\langle nid \leftarrow t\rangle_n - n_s)$) **by** *simp*
ultimately have ∃ *n'''*>*0*. *n'''* ≤ $\langle nid \leftarrow t\rangle_n - n_s$ ∧ *length* (*devExt t* $\langle nid \leftarrow t\rangle_n$ *nid' n_s n'''*) < *length sbc* **using** *ut.prefix-length*[*of sbc 0* $\langle nid \leftarrow t\rangle_n - n_s$] **by** *simp*

then obtain n_p **where** $n_p{>}0$ **and** $n_p \leq \langle nid \leftarrow t\rangle_n{-}n_s$ **and** *length* (*devExt t* $\langle nid \leftarrow t\rangle_n$ *nid′* n_s n_p) < *length sbc* **by** *auto*
hence *length* (*devExt t* $\langle nid \leftarrow t\rangle_n$ *nid′* $(n_s + n_p)$ *0*) < *length sbc* **using** *devExt-shift* **by** *simp*
moreover from *lAct* **have** $\langle nid \leftarrow t\rangle_n{\geq}n_s$ **using** *latestActless* **by** *blast*
with ‹$n_p \leq \langle nid \leftarrow t\rangle_n{-}n_s$› **have** $(n_s + n_p) \leq \langle nid \leftarrow t\rangle_n$ **by** *simp*
moreover from ‹$n_p \leq \langle nid \leftarrow t\rangle_n{-}n_s$› **have** $n_p \leq \langle nid \leftarrow t\rangle_n$ **by** *simp*
moreover have $\forall\, n''{>}n_s + n_p.\ n'' \leq \langle nid \leftarrow t\rangle_n \wedge \neg$ *Option.is-none* (*devBC t* $\langle nid \leftarrow t\rangle_n$ *nid′* *n″*) $\longrightarrow \neg$ *honest* (*the* (*devBC t* $\langle nid \leftarrow t\rangle_n$ *nid′* *n″*)) **using** *cas* **by** *simp*
ultimately show *?thesis* **by** *auto*
qed
qed
next
assume *asmp*: ¬ *Option.is-none* (*devBC t* $\langle nid \leftarrow t\rangle_n$ *nid′* n_s)
moreover from *lAct* **have** $n_s{\leq}\langle nid \leftarrow t\rangle_n$ **using** *latestActless* **by** *blast*
ultimately have ¬ *honest* (*the* (*devBC t* $\langle nid \leftarrow t\rangle_n$ *nid′* n_s)) **using** *cas* **by** *simp*
moreover from *asmp* **have** $\|$*the* (*devBC t* $\langle nid \leftarrow t\rangle_n$ *nid′* n_s)$\|_{t\ n_s}$
using *devBC-act* **by** *simp*
ultimately have *the* (*devBC t* $\langle nid \leftarrow t\rangle_n$ *nid′* n_s)$\in$*actDn* (*t* n_s)
using *actDn-def* **by** *simp*
hence *length* (*bc* ($\sigma_{the\ (devBC\ t\ \langle nid \leftarrow t\rangle_n\ nid'\ n_s)}(t\ n_s)$)) < *length sbc*
using *assms(2)* **by** *simp*
moreover from *asmp* **have**
devExt t $\langle nid \leftarrow t\rangle_n$ *nid′* n_s *0* = *bc* ($\sigma_{the\ (devBC\ t\ \langle nid \leftarrow t\rangle_n\ nid'\ n_s)}(t\ n_s)$)
by *simp*
ultimately have *length* (*devExt t* $\langle nid \leftarrow t\rangle_n$ *nid′* n_s *0*) < *length sbc* **by** *simp*
moreover from *lAct* **have** $\langle nid \leftarrow t\rangle_n{\geq}n_s$ **using** *latestActless* **by** *blast*
moreover from *cas* **have** $\forall\, n''{>}n_s.\ n''{\leq}\langle nid \leftarrow t\rangle_n \wedge \neg$ *Option.is-none* (*devBC t* $\langle nid \leftarrow t\rangle_n$ *nid′* *n″*) $\longrightarrow \neg$ *honest* (*the* (*devBC t* $\langle nid \leftarrow t\rangle_n$ *nid′* *n″*)) **by** *simp*
ultimately show *?thesis* **by** *auto*
qed
qed
then obtain n' **where** $\langle nid \leftarrow t\rangle_n{\geq}n'$ **and** $n'{\geq}n_s$
and *length* (*devExt t* $\langle nid \leftarrow t\rangle_n$ *nid′* *n′* *0*) < *length sbc*
and *dishonest*: $\forall\, n''{>}n'.\ n''{\leq}\langle nid \leftarrow t\rangle_n \wedge \neg$ *Option.is-none* (*devBC t* $\langle nid \leftarrow t\rangle_n$ *nid′* *n″*) $\longrightarrow \neg$ *honest* (*the* (*devBC t* $\langle nid \leftarrow t\rangle_n$ *nid′* *n″*)) **by** *auto*
interpret *ut*: *dishonest devExt t* $\langle nid \leftarrow t\rangle_n$ *nid′* *n′* $\lambda n.$ *dmining t* $(n' + n)$
proof
fix *n″*
from *devExt-devop*[*of t* $\langle nid \leftarrow t\rangle_n$ *nid′* *n′*]
have *prefix* (*devExt t* $\langle nid \leftarrow t\rangle_n$ *nid′* *n′* (*Suc n″*)) (*devExt t* $\langle nid \leftarrow t\rangle_n$ *nid′* *n′* *n″*) ∨
(∃ *b*. *devExt t* $\langle nid \leftarrow t\rangle_n$ *nid′* *n′* (*Suc n″*) = *devExt t* $\langle nid \leftarrow t\rangle_n$ *nid′* *n′* *n″* @ [*b*]) ∧ ¬ *Option.is-none* (*devBC t* $\langle nid \leftarrow t\rangle_n$ *nid′* $(n' + Suc\ n'')$) ∧ $\|$*the* (*devBC t* $\langle nid \leftarrow t\rangle_n$ *nid′* $(n' + Suc\ n'')$)$\|_{t\ (n' + Suc\ n'')}$ ∧ $n' + Suc\ n'' \leq \langle nid \leftarrow t\rangle_n$ ∧ *mining* ($\sigma_{the\ (devBC\ t\ \langle nid \leftarrow t\rangle_n\ nid'\ (n' + Suc\ n''))}{}^{t}$ $(n' + Suc\ n'')$) **.**
thus *prefix* (*devExt t* $\langle nid \leftarrow t\rangle_n$ *nid′* *n′* (*Suc n″*)) (*devExt t* $\langle nid \leftarrow t\rangle_n$ *nid′* *n′* *n″*)
∨ (∃ *b*. *devExt t* $\langle nid \leftarrow t\rangle_n$ *nid′* *n′* (*Suc n″*) = *devExt t* $\langle nid \leftarrow t\rangle_n$ *nid′* *n′* *n″* @ [*b*]) ∧ *dmining t* $(n' + Suc\ n'')$
proof
assume *prefix* (*devExt t* $\langle nid \leftarrow t\rangle_n$ *nid′* *n′* (*Suc n″*)) (*devExt t* $\langle nid \leftarrow t\rangle_n$ *nid′* *n′* *n″*)
thus *?thesis* **by** *simp*
next
assume (∃ *b*. *devExt t* $\langle nid \leftarrow t\rangle_n$ *nid′* *n′* (*Suc n″*) = *devExt t* $\langle nid \leftarrow t\rangle_n$ *nid′* *n′* *n″* @ [*b*]) ∧ ¬ *Option.is-none* (*devBC t* $\langle nid \leftarrow t\rangle_n$ *nid′* $(n' + Suc\ n'')$) ∧ $\|$*the* (*devBC t* $\langle nid \leftarrow t\rangle_n$ *nid′* $(n' + Suc\ n'')$)$\|_{t\ (n' + Suc\ n'')}$ ∧ $n' + Suc\ n'' \leq \langle nid \leftarrow t\rangle_n$ ∧ *mining* ($\sigma_{the\ (devBC\ t\ \langle nid \leftarrow t\rangle_n\ nid'\ (n' + Suc\ n''))}{}^{t}$

$(n' + Suc\ n''))$
hence $\exists b.\ devExt\ t\ \langle nid \leftarrow t\rangle_n\ nid'\ n'\ (Suc\ n'') = devExt\ t\ \langle nid \leftarrow t\rangle_n\ nid'\ n'\ n'' @ [b]$
and $\neg\ Option.is\text{-}none\ (devBC\ t\ \langle nid \leftarrow t\rangle_n\ nid'\ (n' + Suc\ n''))$
and $\|the\ (devBC\ t\ \langle nid \leftarrow t\rangle_n\ nid'\ (n' + Suc\ n''))\|_{t\ (n' + Suc\ n'')}$
and $n' + Suc\ n'' \leq \langle nid \leftarrow t\rangle_n$
and $mining\ (\sigma_{the\ (devBC\ t\ \langle nid \leftarrow t\rangle_n\ nid'\ (n' + Suc\ n''))}t\ (n' + Suc\ n''))$
by *auto*
moreover from $\langle n' + Suc\ n'' \leq \langle nid \leftarrow t\rangle_n\rangle\ \langle\neg\ Option.is\text{-}none\ (devBC\ t\ \langle nid \leftarrow t\rangle_n\ nid'\ (n' + Suc\ n''))\rangle$
have $\neg\ honest\ (the\ (devBC\ t\ \langle nid \leftarrow t\rangle_n\ nid'\ (n' + Suc\ n')))$ **using** *dishonest* **by** *simp*
with $\langle\|the\ (devBC\ t\ \langle nid \leftarrow t\rangle_n\ nid'\ (n' + Suc\ n''))\|_{t\ (n' + Suc\ n'')}\rangle$
have $the\ (devBC\ t\ \langle nid \leftarrow t\rangle_n\ nid'\ (n' + Suc\ n''))\in actDn\ (t\ (n' + Suc\ n''))$
using *actDn-def* **by** *simp*
ultimately show *?thesis* **using** *dmining-def* **by** *auto*
qed
qed
interpret *dishonest-growth devLgthBC t* $\langle nid \leftarrow t\rangle_n$ *nid' n'* $\lambda n.\ dmining\ t\ (n' + n)$
by *unfold-locales*
interpret *honest-growth* $\lambda n.\ PoW\ t\ (n' + n)\ \lambda n.\ hmining\ t\ (n' + n)$
proof
show $\bigwedge n.\ PoW\ t\ (n' + n) \leq PoW\ t\ (n' + Suc\ n)$ **using** *pow-mono* **by** *simp*
show $\bigwedge n.\ hmining\ t\ (n' + Suc\ n) \Longrightarrow PoW\ t\ (n' + n) < PoW\ t\ (n' + Suc\ n)$
using *pow-mining-suc* **by** *simp*
qed
interpret *bg*: *bounded-growth length sbc* $\lambda n.\ PoW\ t\ (n' + n)$ *devLgthBC t* $\langle nid \leftarrow t\rangle_n$ *nid' n'* $\lambda n.\ hmining\ t\ (n' + n)\ \lambda n.\ dmining\ t\ (n' + n)$ *length sbc cb*
proof
from *assms*(3) $\langle n' \geq n_s\rangle$ **show** $length\ sbc + cb \leq PoW\ t\ (n' + 0)$ **using** *pow-mono*[*of* $n_s\ n'\ t$] **by** *simp*
next
from $\langle length\ (devExt\ t\ \langle nid \leftarrow t\rangle_n\ nid'\ n'\ 0) < length\ sbc\rangle$ **show** $length\ (devExt\ t\ \langle nid \leftarrow t\rangle_n\ nid'\ n'\ 0) < length\ sbc$ **.**
next
fix $n''\ n'''$
assume $cb < card\ \{i.\ n'' < i \wedge i \leq n''' \wedge dmining\ t\ (n' + i)\}$
hence $cb < card\ \{i.\ n'' + n' < i \wedge i \leq n''' + n' \wedge dmining\ t\ i\}$
using *cardshift*[*of* $n''\ n'''\ dmining\ t\ n'$] **by** *simp*
with *fair*[*of* $n'' + n'\ n''' + n'\ t$]
have $cb < card\ \{i.\ n'' + n' < i \wedge i \leq n''' + n' \wedge hmining\ t\ i\}$ **by** *simp*
thus $cb < card\ \{i.\ n'' < i \wedge i \leq n''' \wedge hmining\ t\ (n' + i)\}$
using *cardshift*[*of* $n''\ n'''\ hmining\ t\ n'$] **by** *simp*
qed
from $\langle\langle nid \leftarrow t\rangle_n \geq n'\rangle$ **have** $length\ (devExt\ t\ \langle nid \leftarrow t\rangle_n\ nid'\ n'\ (\langle nid \leftarrow t\rangle_n - n')) < PoW\ t\ \langle nid \leftarrow t\rangle_n$
using *bg.hn-upper-bound*[*of* $\langle nid \leftarrow t\rangle_n - n'$] **by** *simp*
moreover from $\langle\|nid'\|_{t\ \langle nid \leftarrow t\rangle_n}\rangle\ \langle\langle nid \leftarrow t\rangle_n \geq n'\rangle$
have $bc\ (\sigma_{the\ (devBC\ t\ \langle nid \leftarrow t\rangle_n\ nid'\ \langle nid \leftarrow t\rangle_n)}t\ \langle nid \leftarrow t\rangle_n) = devExt\ t\ \langle nid \leftarrow t\rangle_n\ nid'\ n'\ (\langle nid \leftarrow t\rangle_n - n')$
using *devExt-bc-geq*[*of t* $\langle nid \leftarrow t\rangle_n$ *nid'* $\langle nid \leftarrow t\rangle_n\ n'$] **by** *simp*
ultimately have $length\ (bc\ (\sigma_{nid'}(t\ \langle nid \leftarrow t\rangle_n))) < PoW\ t\ \langle nid \leftarrow t\rangle_n$
using $\langle\|nid'\|_{t\ \langle nid \leftarrow t\rangle_n}\rangle$ **by** *simp*
moreover have $PoW\ t\ \langle nid \leftarrow t\rangle_n \leq length\ (bc\ (\sigma_{nid'}(t\ \langle nid \leftarrow t\rangle_n)))$ (**is** *?lhs* $\leq$ *?rhs*)
proof −

```
            from ‹honest nid› ‹‖nid‖t ⟨nid ← t⟩n›
              have ?lhs ≤ length (MAX (pin (σnid t ⟨nid ← t⟩n))) using pow-le-max by simp
            also from ‹bc (σnid' (t ⟨nid ← t⟩n)) = MAX (pin (σnid t ⟨nid ← t⟩n))›
              have ... = length (bc (σnid' (t ⟨nid ← t⟩n))) by simp
            finally show ?thesis .
          qed
          ultimately show False by simp
        qed
      qed
      moreover from ‹‖nid‖t n› have ⟨nid → t⟩n=n using nxtAct-active by simp
      ultimately show ?thesis by auto
    qed
    moreover from ‹‖nid‖t n› have ⟨nid → t⟩n=n using nxtAct-active by simp
    ultimately show ?thesis by auto
  next
    assume ¬ (∃ b∈pin (σnid t ⟨nid ← t⟩n). length b > length (bc (σnid t ⟨nid ← t⟩n)))
    moreover from ‹‖nid‖t n› have ∃ n'≥n. ‖nid‖t n' by auto
    moreover from lAct have ∃ n'. latestAct-cond nid t n n' by auto
    ultimately have ¬ mining (σnid t ⟨nid → t⟩n) ∧ bc (σnid t ⟨nid → t⟩n) = bc (σnid t ⟨nid ←
t⟩n) ∨
      mining (σnid t ⟨nid → t⟩n) ∧ (∃ b. bc (σnid t ⟨nid → t⟩n) = bc (σnid t ⟨nid ← t⟩n) @ [b])
      using ‹honest nid› bhv-hn-in[of nid n t] by simp
    moreover have prefix sbc (bc (σnid t ⟨nid ← t⟩n))
    proof −
      from ‹∃ n'. latestAct-cond nid t n n'› have ⟨nid ← t⟩n < n using latestAct-prop(2) by simp
      moreover from lAct have ⟨nid ← t⟩n ≥ ns using latestActless by blast
      moreover from ‹∃ n'. latestAct-cond nid t n n'› have ‖nid‖t ⟨nid ← t⟩n
        using latestAct-prop(1) by simp
      with ‹honest nid› have nid ∈ actHn (t ⟨nid ← t⟩n) using actHn-def by simp
      ultimately show ?thesis using step.IH by auto
    qed
    moreover from ‹‖nid‖t n› have ⟨nid → t⟩n=n using nxtAct-active by simp
    ultimately show ?thesis by auto
  qed
 next
  assume nAct: ¬ (∃ n' < n. n' ≥ ns ∧ ‖nid‖t n')
  moreover from step.hyps have ns ≤ n by simp
  ultimately have ⟨nid → t⟩ns = n using ‹‖nid‖t n› nxtAct-eq[of ns n nid t] by simp
  with ‹honest nid› show ?thesis using assms(1) by auto
 qed
 qed
 qed
 with assms(5) show ?thesis by simp
qed

end

end
```

References

[1] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-Oriented Software Architecture: A System of Patterns*. Wiley West Sussex, England, 1996.

[2] Diego Marmsoler. Towards a theory of architectural styles. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering - FSE 2014*, pages 823–825. ACM, ACM Press, 2014.

[3] Diego Marmsoler. Dynamic architectures. *Archive of Formal Proofs*, July 2017. http://isa-afp.org/entries/DynamicArchitectures.html, Formal proof development.

[4] Diego Marmsoler. Hierarchical specication and verication of architecture design patterns. In *Fundamental Approaches to Software Engineering - 21th International Conference, FASE 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings*, 2018.