

A Theory of Architectural Design Patterns

Diego Marmsoler

August 16, 2018

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	30
6.2.1	Component Behavior	31
6.2.2	Maximal Trusted Blockchains	36
6.2.3	Trusted 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_ct ⟨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** $\exists id. \forall k. \forall id'. (\|id'\|_k \longrightarrow id = id')$.
then obtain i'' **where** $\forall k. \forall id'. (\|id'\|_k \longrightarrow i'' = id)$ **by** *auto*
with $\langle\|id'\|_k\rangle$ **have** $id=i''$ **and** $id'=i''$ **using** *a1* **by** *auto*
thus $id' = id$ **by** *simp*
qed
qed
qed
next
fix i'' **show** $\forall k id'. \|id'\|_k \longrightarrow id' = i'' \Longrightarrow i'' = id$ **using** *a1* **by** *auto*
qed
hence $\|id\|_k \Longrightarrow id = \text{the-singleton}$ **by** (*simp add: the-singleton-def*)
} note $g1 = \text{this}$
thus $\bigwedge id. \|id\|_k \Longrightarrow id = \text{the-singleton}$ **by** *simp*

from *alwaysActive* **obtain** id **where** $\|id\|_k$ **by** *blast*
with $g1$ **have** $id = \text{the-singleton}$ **by** *simp*
with $\langle\|id\|_k\rangle$ **show** $\| \text{the-singleton} \|_k$ **by** *simp*
qed
declare *ts-prop(2)[simp]*

lemma *lNact-active[simp]*:
fixes $cid\ t\ n$
shows $\langle \text{the-singleton} \Leftarrow t \rangle_n = n$
using *lNact-active ts-prop(2)* **by** *auto*

lemma *lNxt-active[simp]*:
fixes $cid\ t\ n$
shows $\langle \text{the-singleton} \rightarrow t \rangle_n = n$
by (*simp add: nxtAct-active*)

lemma *baI[intro]*:
fixes $t\ n\ a$
assumes $\varphi (\sigma_{\text{the-singleton}}(t\ n))$
shows $\text{eval the-singleton } t\ t'\ n\ [\varphi]_b$ **using** *assms* **by** (*simp add: baIANow*)

lemma *baE[elim]*:
fixes $t\ n\ a$
assumes $\text{eval the-singleton } t\ t'\ n\ [\varphi]_b$
shows $\varphi (\sigma_{\text{the-singleton}}(t\ n))$ **using** *assms* **by** (*simp add: baEANow*)

lemma *evtE[elim]*:
fixes $t\ id\ n\ a$
assumes $\text{eval the-singleton } t\ t'\ n\ (\Diamond_b\ \gamma)$
shows $\exists n' \geq n. \text{eval the-singleton } t\ t'\ n'\ \gamma$
proof –
have $\| \text{the-singleton} \|_t\ n$ **by** *simp*
with *assms* **obtain** n' **where** $n' \geq \langle \text{the-singleton} \rightarrow t \rangle_n$ **and** $(\exists i \geq n'. \| \text{the-singleton} \|_t\ i \wedge$
 $(\forall n'' \geq \langle \text{the-singleton} \Leftarrow t \rangle_{n'}. n'' \leq \langle \text{the-singleton} \rightarrow t \rangle_{n'} \longrightarrow \text{eval the-singleton } t\ t'\ n''\ \gamma)) \vee$
 $\neg (\exists i \geq n'. \| \text{the-singleton} \|_t\ i) \wedge \text{eval the-singleton } t\ t'\ n'\ \gamma$ **using** *evtEA[of n the-singleton t]* **by** *blast*
moreover **have** $\| \text{the-singleton} \|_t\ n'$ **by** *simp*
ultimately **have**
 $\forall n'' \geq \langle \text{the-singleton} \Leftarrow t \rangle_{n'}. n'' \leq \langle \text{the-singleton} \rightarrow t \rangle_{n'} \longrightarrow \text{eval the-singleton } t\ t'\ n''\ \gamma$ **by** *auto*
hence $\text{eval the-singleton } t\ t'\ n'\ \gamma$ **by** *simp*
moreover **from** $\langle n' \rangle \geq \langle \text{the-singleton} \rightarrow t \rangle_n$ **have** $n' \geq n$ **by** (*simp add: nxtAct-active*)

ultimately show ?thesis by auto
qed

lemma globE[elim]:

fixes $t \text{ id } n \ a$
assumes $\text{eval the-singleton } t \ t' \ n \ (\Box_b \ \gamma)$
shows $\forall n' \geq n. \text{eval the-singleton } t \ t' \ n' \ \gamma$

proof

fix n' show $n \leq n' \longrightarrow \text{eval the-singleton } t \ t' \ n' \ \gamma$

proof

assume $n \leq n'$
hence $\langle \text{the-singleton} \Leftarrow t \rangle_n \leq n'$ by simp
moreover have $\|\text{the-singleton}\|_t \ n$ by simp
ultimately show $\text{eval the-singleton } t \ t' \ n' \ \gamma$
using $\langle \text{eval the-singleton } t \ t' \ n \ (\Box_b \ \gamma) \rangle$ globEA by blast

qed

qed

lemma untilI[intro]:

fixes $t :: \text{nat} \Rightarrow \text{cnf}$
and $t' :: \text{nat} \Rightarrow 'cmp$
and $n :: \text{nat}$
and $n' :: \text{nat}$
assumes $n' \geq n$
and $\text{eval the-singleton } t \ t' \ n' \ \gamma$
and $\bigwedge n''. \llbracket n \leq n''; n'' < n \rrbracket \Longrightarrow \text{eval the-singleton } t \ t' \ n'' \ \gamma'$
shows $\text{eval the-singleton } t \ t' \ n \ (\gamma' \ \mathfrak{U}_b \ \gamma)$

proof -

have $\|\text{the-singleton}\|_t \ n$ by simp
moreover from $\langle n' \geq n \rangle$ have $\langle \text{the-singleton} \Leftarrow t \rangle_n \leq n'$ by simp
moreover have $\|\text{the-singleton}\|_t \ n'$ by simp
moreover have
 $\exists n'' \geq \langle \text{the-singleton} \Leftarrow t \rangle_{n'}. n'' \leq \langle \text{the-singleton} \rightarrow t \rangle_{n'} \wedge \text{eval the-singleton } t \ t' \ n'' \ \gamma \wedge$
 $(\forall n''' \geq \langle \text{the-singleton} \rightarrow t \rangle_n. n''' < \langle \text{the-singleton} \Leftarrow t \rangle_{n''} \longrightarrow$
 $(\exists n'''' \geq \langle \text{the-singleton} \Leftarrow t \rangle_{n''}. n'''' \leq \langle \text{the-singleton} \rightarrow t \rangle_{n''''} \wedge \text{eval the-singleton } t \ t' \ n'''' \ \gamma'))$

proof -

have $n' \geq \langle \text{the-singleton} \Leftarrow t \rangle_{n'}$ by simp
moreover have $n' \leq \langle \text{the-singleton} \rightarrow t \rangle_{n'}$ by simp
moreover from $\text{assms}(3)$ have $(\forall n'' \geq \langle \text{the-singleton} \rightarrow t \rangle_n. n'' < \langle \text{the-singleton} \Leftarrow t \rangle_{n'} \longrightarrow$
 $(\exists n''' \geq \langle \text{the-singleton} \Leftarrow t \rangle_{n''}. n''' \leq \langle \text{the-singleton} \rightarrow t \rangle_{n''} \wedge \text{eval the-singleton } t \ t' \ n''' \ \gamma'))$
by auto

ultimately show ?thesis using $\langle \text{eval the-singleton } t \ t' \ n' \ \gamma \rangle$ by auto

qed

ultimately show ?thesis using untilIA[$\text{of } n \ \text{the-singleton } t \ n' \ t' \ \gamma \ \gamma'$] by blast

qed

lemma untilE[elim]:

fixes $t \ \text{id} \ n \ \gamma' \ \gamma$
assumes $\text{eval the-singleton } t \ t' \ n \ (\gamma' \ \mathfrak{U}_b \ \gamma)$
shows $\exists n' \geq n. \text{eval the-singleton } t \ t' \ n' \ \gamma \wedge (\forall n'' \geq n. n'' < n' \longrightarrow \text{eval the-singleton } t \ t' \ n'' \ \gamma')$

proof -

have $\|\text{the-singleton}\|_t \ n$ by simp
with $\langle \text{eval the-singleton } t \ t' \ n \ (\gamma' \ \mathfrak{U}_b \ \gamma) \rangle$ obtain n' where $n' \geq \langle \text{the-singleton} \rightarrow t \rangle_n$ and
 $(\exists i \geq n'. \|\text{the-singleton}\|_t \ i) \wedge$
 $(\forall n'' \geq \langle \text{the-singleton} \Leftarrow t \rangle_{n'}. n'' \leq \langle \text{the-singleton} \rightarrow t \rangle_{n'} \longrightarrow \text{eval the-singleton } t \ t' \ n'' \ \gamma) \wedge$

$(\forall n'' \geq \langle \text{the-singleton} \Leftarrow t \rangle_n. n'' < \langle \text{the-singleton} \Leftarrow t \rangle_{n'} \longrightarrow \text{eval the-singleton } t \ t' \ n'' \ \gamma') \vee$
 $\neg (\exists i \geq n'. \|\text{the-singleton}\|_t i) \wedge$
 $\text{eval the-singleton } t \ t' \ n' \ \gamma \wedge (\forall n'' \geq \langle \text{the-singleton} \Leftarrow t \rangle_n. n'' < n' \longrightarrow \text{eval the-singleton } t \ t' \ n'' \ \gamma')$
using *untilEA*[of *n the-singleton t t' γ' γ'*] **by** *auto*
moreover have $\|\text{the-singleton}\|_t n'$ **by** *simp*
ultimately have
 $(\forall n'' \geq \langle \text{the-singleton} \Leftarrow t \rangle_{n'}. n'' \leq \langle \text{the-singleton} \rightarrow t \rangle_{n'} \longrightarrow \text{eval the-singleton } t \ t' \ n'' \ \gamma) \wedge$
 $(\forall n'' \geq \langle \text{the-singleton} \Leftarrow t \rangle_n. n'' < \langle \text{the-singleton} \Leftarrow t \rangle_{n'} \longrightarrow \text{eval the-singleton } t \ t' \ n'' \ \gamma')$ **by** *auto*
hence *eval the-singleton t t' n' γ* **and** $(\forall n'' \geq n. n'' < n' \longrightarrow \text{eval the-singleton } t \ t' \ n'' \ \gamma')$ **by** *auto*
with $\langle \text{eval the-singleton } t \ t' \ n' \ \gamma \rangle \langle n' \geq \langle \text{the-singleton} \rightarrow t \rangle_n \rangle$ **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 \Rightarrow cnf \Rightarrow bool* ($\|\cdot\|_- [0,110]60$)
and *pbcmp :: 'pid \Rightarrow cnf \Rightarrow 'PB* ($\sigma_-(-) [0,110]60$)
and *sbactive :: 'sid \Rightarrow cnf \Rightarrow bool* ($\|\cdot\|_- [0,110]60$)
and *sbcmp :: 'sid \Rightarrow cnf \Rightarrow 'SB* ($\sigma_-(-) [0,110]60$) +
fixes *pbsb :: 'PB \Rightarrow ('evt set) subscription set*
and *pbnt :: 'PB \Rightarrow ('evt \times 'msg)*
and *sbnt :: 'SB \Rightarrow ('evt \times 'msg) set*
and *sbsb :: 'SB \Rightarrow ('evt set) subscription*
assumes *conn1: $\bigwedge k \text{ pid. } \|\text{pid}\|_k$*
 $\implies \text{pbsb } (\sigma_{\text{pid}}(k)) = (\bigcup \text{sid} \in \{\text{sid. } \|\text{sid}\|_k\}. \{\text{sbsb } (\sigma_{\text{sid}}(k))\})$
and *conn2: $\bigwedge t \ n \ n'' \ \text{sid} \ \text{pid} \ E \ e \ m.$*
 $\llbracket t \in \text{arch}; \|\text{pid}\|_t n; \|\text{sid}\|_t n; \text{sub } E = \text{sbsb } (\sigma_{\text{sid}}(t \ n)); n'' \geq n; e \in E;$
 $\nexists n' \ E'. n' \geq n \wedge n' \leq n'' \wedge \|\text{sid}\|_t n' \wedge$
 $\text{unsub } E' = \text{sbsb } (\sigma_{\text{sid}}(t \ n')) \wedge e \in E';$
 $(e, m) = \text{pbnt } (\sigma_{\text{pid}}(t \ n'')); \|\text{sid}\|_t n'' \rrbracket$
 $\implies \text{pbnt } (\sigma_{\text{pid}}(t \ n'')) \in \text{sbnt } (\sigma_{\text{sid}}(t \ n''))$
begin

2.2.1 Calculus Interpretation

pb.baIA: $\llbracket \exists i \geq n. \|c\|_t i; \varphi (\sigma_{ct} (\text{pb.nextAct } c \ t \ n)) \rrbracket \implies \text{pb.eval } c \ t \ t' \ n \ [\varphi]_b$

sb.baIA: $\llbracket \exists i \geq n. \|c\|_{t\ i}; \varphi (\sigma_{ct} (sb.nextAct\ c\ t\ n)) \rrbracket \implies sb.eval\ c\ t\ t'\ n\ [\varphi]_b$

2.2.2 Results from Singleton

abbreviation *the-pb* :: 'pid where
the-pb $\equiv pb.the-singleton$

pb.ts-prop (1): $\|id\|_k \implies id = the-pb$

pb.ts-prop (2): $\|the-pb\|_k$

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 $(e, m) = pbnt (\sigma_{the-pb}(t\ n''))$

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* ($\sigma_{the-pb}(k)$) = $(\bigcup sid \in \{sid. \|sid\|_k\}. \{sbsb (\sigma_{sid}(k))\})$

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* \times *PROB*) *set*

axiomatization where *sbWF*: *wf sb*

typedecl SOL

consts solve:: PROB ⇒ 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 ⇒ cnf ⇒ bool (||-||_ [0,110]60)
and bbcmp :: 'bid ⇒ cnf ⇒ 'BB (σ_-) [0,110]60
and ksactive :: 'kid ⇒ cnf ⇒ bool (||-||_ [0,110]60)
and kscmp :: 'kid ⇒ cnf ⇒ 'KS (σ_-) [0,110]60
and bbrp :: 'BB ⇒ (PROB set) subscription set
and bbcs :: 'BB ⇒ (PROB × SOL)
and kscs :: 'KS ⇒ (PROB × SOL) set
and ksrp :: 'KS ⇒ (PROB set) subscription +
fixes bbns :: 'BB ⇒ (PROB × SOL) set
and ksns :: 'KS ⇒ (PROB × SOL)
and bbop :: 'BB ⇒ PROB
and ksop :: 'KS ⇒ PROB set
and prob :: 'kid ⇒ PROB

assumes

ks1: ∀ p. ∃ ks. p=prob ks — Component Parameter

— Assertions about component behavior.

and bhvbb1: $\bigwedge t t' bId p s. \llbracket t \in arch \rrbracket \Longrightarrow pb.eval bId t t' 0$

$(\Box_b ([\lambda bb. (p,s) \in bbns bb]_b \longrightarrow^b (\Diamond_b [\lambda bb. (p,s) = bbcs bb]_b)))$

and bhvbb2: $\bigwedge t t' bId P q. \llbracket t \in arch \rrbracket \Longrightarrow pb.eval bId t t' 0$

$(\Box_b ([\lambda bb. sub P \in bbrp bb \wedge q \in P]_b \longrightarrow^b (\Diamond_b [\lambda bb. q = bbop bb]_b)))$

and bhvbb3: $\bigwedge t t' bId p. \llbracket t \in arch \rrbracket \Longrightarrow pb.eval bId t t' 0$

$(\Box_b ([\lambda bb. p = bbop(bb)]_b \longrightarrow^b ([\lambda bb. p=bbop(bb)]_b \mathfrak{W}_b [\lambda bb. (p,solve(p)) = bbcs(bb)]_b)))$

and bhvks1: $\bigwedge t t' kId p P. \llbracket t \in arch; p = prob kId \rrbracket \Longrightarrow sb.eval kId t t' 0$

$(\Box_b ([\lambda ks. sub P = ksrp ks]_b \wedge^b (\forall_b q. ((sb.pred (q \in P)) \longrightarrow^b (\Diamond_b ([\lambda ks. (q,solve(q)) \in kscs ks]_b)))) \longrightarrow^b (\Diamond_b [\lambda ks. (p, solve p) = ksns ks]_b)))$

and bhvks2: $\bigwedge t t' kId p P q. \llbracket t \in arch; p = prob kId \rrbracket \Longrightarrow sb.eval kId t t' 0$

$(\Box_b [\lambda ks. sub P = ksrp ks \wedge q \in P \longrightarrow (q,p) \in sb]_b)$

and bhvks3: $\bigwedge t t' kId p. \llbracket t \in arch; p = prob kId \rrbracket \Longrightarrow sb.eval kId t t' 0$

$(\Box_b ([\lambda ks. p \in ksop ks]_b \longrightarrow^b (\Diamond_b [\lambda ks. (\exists P. sub P = ksrp ks)]_b)))$

and bhvks4: $\bigwedge t t' kId p P. \llbracket t \in arch; p \in P \rrbracket \Longrightarrow sb.eval kId t t' 0$

$(\Box_b ([\lambda ks. sub P = ksrp ks]_b \longrightarrow^b ((\neg^b (\exists_b P'. (sb.pred (p \in P') \wedge^b [\lambda ks. unsub P' = ksrp ks]_b))) \mathfrak{W}_b [\lambda ks. (p,solve p) \in kscs ks]_b)))$

— Assertions about component activation.

and actks:

$\bigwedge t n kid p. \llbracket t \in arch; \llbracket kid \rrbracket_t n; p=prob kid; p \in ksop (\sigma_{kid}(t n)) \rrbracket$

$\Longrightarrow (\exists n' \geq n. \llbracket kid \rrbracket_{t n'} \wedge (p, solve p) = ksns (\sigma_{kid}(t n')) \wedge$

$(\forall n'' \geq n. n'' < n' \longrightarrow \llbracket kid \rrbracket_{t n''}))$

$\vee (\forall n' \geq n. (\llbracket kid \rrbracket_{t n'} \wedge (\neg(p, solve p) = ksns (\sigma_{kid}(t n')))))$

— Assertions about connections.

and conn1: $\bigwedge k bid. \llbracket bid \rrbracket_k$

$\Longrightarrow bbns (\sigma_{bid}(k)) = (\bigcup_{kid \in \{kid. \llbracket kid \rrbracket_k\}. \{ksns (\sigma_{kid}(k))\}}$

and *conn2*: $\bigwedge k \text{ kid}. \llbracket \text{kid} \rrbracket_k$
 $\implies \text{ksop} (\sigma_{\text{kid}}(k)) = (\bigcup \text{bid} \in \{\text{bid}. \llbracket \text{bid} \rrbracket_k\}. \{\text{bbop} (\sigma_{\text{bid}}(k))\})$
begin
notation *sb.lNAct* ($\langle - \leftarrow - \rangle$ -)
notation *sb.nextAct* ($\langle - \rightarrow - \rangle$ -)
notation *pb.lNAct* ($\langle - \leftarrow - \rangle$ -)
notation *pb.nextAct* ($\langle - \rightarrow - \rangle$ -)

3.2.1 Calculus Interpretation

pb.baIA: $\llbracket \exists i \geq n. \llbracket c \rrbracket_{t \ i}; \varphi (\sigma_{ct} \langle c \rightarrow t \rangle_n) \rrbracket \implies \text{pb.eval } c \ t \ t' \ n \ [\varphi]_b$
sb.baIA: $\llbracket \exists i \geq n. \llbracket c \rrbracket_{t \ i}; \varphi (\sigma_{ct} \langle c \rightarrow t \rangle_n) \rrbracket \implies \text{sb.eval } c \ t \ t' \ n \ [\varphi]_b$

3.2.2 Results from Singleton

abbreviation *the-bb* \equiv *the-pb*
pb.ts-prop (1): $\llbracket \text{id} \rrbracket_k \implies \text{id} = \text{the-bb}$
pb.ts-prop (2): $\llbracket \text{the-bb} \rrbracket_k$

3.2.3 Results from Publisher Subscriber

msgDelivery: $\llbracket t \in \text{arch}; \llbracket \text{sid} \rrbracket_{t \ n}; \text{sub } E = \text{ksrp} (\sigma_{\text{sidt}} n); n \leq n''; \nexists n' E'. n \leq n' \wedge n' \leq n'' \wedge \llbracket \text{sid} \rrbracket_{t \ n'} \wedge \text{unsub } E' = \text{ksrp} (\sigma_{\text{sidt}} n') \wedge e \in E'; e \in E; (e, m) = \text{bbcs} (\sigma_{\text{the-bbt}} n''); \llbracket \text{sid} \rrbracket_{t \ n''} \rrbracket \implies (e, m) \in \text{kscs} (\sigma_{\text{sidt}} n'')$

lemma *conn2-bb*:

fixes *k* **and** *kid*: '*kid*

assumes $\llbracket \text{kid} \rrbracket_k$

shows $\text{bbop} (\sigma_{\text{the-bb}}(k)) \in \text{ksop} (\sigma_{\text{kid}}(k))$

proof –

from *assms* **have** $\text{ksop} (\sigma_{\text{kid}}(k)) = (\bigcup \text{bid} \in \{\text{bid}. \llbracket \text{bid} \rrbracket_k\}. \{\text{bbop} (\sigma_{\text{bid}}(k))\})$ **using** *conn2* **by** *simp*

moreover **have** $(\bigcup \text{bid}. \{\text{bid}. \llbracket \text{bid} \rrbracket_k\}) = \{\text{the-bb}\}$ **using** *pb.ts-prop(1)* **by** *auto*

hence $(\bigcup \text{bid} \in \{\text{bid}. \llbracket \text{bid} \rrbracket_k\}. \{\text{bbop} (\sigma_{\text{bid}}(k))\}) = \{\text{bbop} (\sigma_{\text{the-bb}}(k))\}$ **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* $\lambda \text{kid}. \text{prob } \text{kid}$] *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

$\forall n. (\exists n' \geq n. \|sKs (bbop(\sigma_{the-bb}(t n)))\|_{t n'})$

shows

$\forall n. (\exists P. sub P \in bbrp(\sigma_{the-bb}(t n)) \wedge p \in P) \longrightarrow$

$(\exists m \geq n. (p, solve(p)) = bbc(\sigma_{the-bb}(t m)))$

— 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*: $\forall p'. (p', p) \in sb \longrightarrow (\forall n. (\exists P. sub P \in bbrp(\sigma_{the-bb}(t n)) \wedge p' \in P)$

$\longrightarrow (\exists m \geq n. (p', solve(p')) = bbc(\sigma_{the-bb}(t m))))$

show $\forall n. (\exists P. sub P \in bbrp(\sigma_{the-bb}(t n)) \wedge p \in P)$

$\longrightarrow (\exists m \geq n. (p, solve(p)) = bbc(\sigma_{the-bb}(t m)))$

proof

fix n_0 **show** $(\exists P. sub P \in bbrp(\sigma_{the-bb}(t n_0)) \wedge p \in P) \longrightarrow$

$(\exists m \geq n_0. (p, solve(p)) = bbc(\sigma_{the-bb}(t m)))$

proof

assume $\exists P. sub P \in bbrp(\sigma_{the-bb}(t n_0)) \wedge p \in P$

moreover **have** $(\exists P. sub P \in bbrp(\sigma_{the-bb}(t n_0)) \wedge p \in P) \longrightarrow (\exists n' \geq n_0. p = bbop(\sigma_{the-bb}(t n')))$

proof

assume $\exists P. sub P \in bbrp(\sigma_{the-bb}(t n_0)) \wedge p \in P$

then obtain P **where** $sub P \in bbrp(\sigma_{the-bb}(t n_0))$ **and** $p \in P$ **by** *auto*

hence $pb.eval the-bb t t' n_0 [\lambda bb. sub P \in bbrp bb \wedge p \in P]_b$ **using** *pb.baI* **by** *simp*

moreover from $pb.globE[OF bhvbb2]$ **have**

$pb.eval the-bb t t' n_0 ([\lambda bb. sub P \in bbrp bb \wedge p \in P]_b \longrightarrow^b \diamond_b [\lambda bb. p = bbop bb]_b)$

using $\langle t \in arch \rangle$ **by** *simp*

ultimately have $pb.eval the-bb t t' n_0 (\diamond_b [\lambda bb. p = bbop bb]_b)$ **using** *pb.impE* **by** *blast*

then obtain n' **where** $n' \geq n_0$ **and** $pb.eval the-bb t t' n' [\lambda bb. p = bbop bb]_b$

using *pb.evtE* **by** *blast*

hence $p = bbop(\sigma_{the-bb}(t n'))$ **using** *pb.baE* **by** *auto*

with $\langle n' \geq n_0 \rangle$ **show** $\exists n' \geq n_0. p = bbop(\sigma_{the-bb}(t n'))$ **by** *auto*

qed

ultimately obtain n **where** $n \geq n_0$ **and** $p = bbop(\sigma_{the-bb}(t n))$ **by** *auto*

— Problem p is provided at the output of the blackboard until it is solved

— or forever...

from $pb.globE[OF bhvbb3]$ **have**

$pb.eval the-bb t t' n ([\lambda bb. p = bbop(bb)]_b \longrightarrow^b$

$([\lambda bb. p = bbop(bb)]_b \mathfrak{W}_b [\lambda bb. (p, solve(p)) = bbc(bb)]_b))$

using $\langle t \in arch \rangle$ **by** *auto*

moreover from $\langle p = bbop(\sigma_{the-bb}(t n)) \rangle$ **have**

$pb.eval the-bb t t' n [\lambda bb. p = bbop bb]_b$

using $\langle t \in arch \rangle$ *pb.baI* **by** *simp*

ultimately have $pb.eval the-bb t t' n$

$([\lambda bb. p = bbop(bb)]_b \mathfrak{W}_b [\lambda bb. (p, solve(p)) = bbc(bb)]_b)$

using *pb.impE* **by** *blast*

hence $pb.eval the-bb t t' n (([\lambda bb. p = bbop bb]_b \mathfrak{L}_b$

$(\lambda bb. (p, solve(p)) = bbcs\ bb]_b) \vee^b (\Box_b [\lambda bb. p=bbop\ bb]_b)$
using *pb.until-def* **by** *simp*
hence *pb.eval the-bb t t' n*
 $([\lambda bb. p=bbop\ bb]_b \ \mathfrak{A}_b \ [\lambda bb. (p, solve(p)) = bbcs\ bb]_b) \vee$
 $(pb.eval\ the-bb\ t\ t'\ n \ (\Box_b \ [\lambda\ bb.\ p=bbop\ bb]_b))$
using *pb.disjE* **by** *simp*
thus $\exists m \geq n_0. (p, solve\ p) = bbcs(\sigma_{the-bb}(t\ m))$
— 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*
 $([\lambda bb. p=bbop\ bb]_b \ \mathfrak{A}_b \ [\lambda bb. (p, solve(p)) = bbcs\ bb]_b)$
hence $\exists i \geq n. (pb.eval\ the-bb\ t\ t'\ i$
 $[\lambda bb. (p, solve(p)) = bbcs\ bb]_b \wedge$
 $(\forall k \geq n. k < i \longrightarrow pb.eval\ the-bb\ t\ t'\ k \ [\lambda bb. p = bbop\ bb]_b))$
using $\langle t \in arch \rangle$ *pb.untilE* **by** *simp*
then obtain *i* **where** $i \geq n$ **and**
pb.eval the-bb t t' i $[\lambda bb. (p, solve(p)) = bbcs\ bb]_b$ **by** *auto*
hence $(p, solve(p)) = bbcs(\sigma_{the-bb}(t\ i))$
using $\langle t \in arch \rangle$ *pb.baEA* **by** *auto*
moreover from $\langle i \geq n \rangle \langle n \geq n_0 \rangle$ **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 $\langle t \in arch \rangle$ *pb.globE* **by** *auto*
hence *outp*: $\forall n' \geq n. (p = bbop\ (\sigma_{the-bb}(t\ n')))$
using $\langle t \in arch \rangle$ *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 $\langle n_k \geq n \rangle$ **have** $p = bbop\ (\sigma_{the-bb}(t\ n_k))$
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 $\langle \|sKs\ p\|_{t\ n_k} \rangle$ *actks[of t sKs p]* $\langle t \in arch \rangle$ *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 $(p, \text{solve } p) = ksns\ (\sigma_{sKs}\ p\ t\ n_s)$ **by** *auto*
moreover have $\langle sKs\ p \rightarrow t \rangle_{n_s} = n_s$
by (*simp add*: $\langle \|sKs\ p\|_t\ n_s \rangle\ sb.\text{nextAct-active}$)
ultimately have
 $(p, \text{solve}(p)) \in bbns\ (\sigma_{the-bb}(t\ (\langle sKs\ p \rightarrow t \rangle_{n_s})))$
using *conn1*[*OF pb.ts-prop*(\varnothing)] $\langle \|sKs\ p\|_t\ n_s \rangle$ **by** *auto*

— finally, the blackboard will forward the solution which finishes the proof.
with *bhvbb1* **have** *pb.eval the-bb t t'* $(\langle sKs\ p \rightarrow t \rangle_{n_s})$
 $(\diamond_b\ [\lambda bb. (p, \text{solve } p) = bbcs\ bb]_b)$
using $\langle t \in arch \rangle\ pb.\text{globE } pb.\text{impE}[of\ the-bb\ t\ t']$ **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 $[\lambda bb. (p, \text{solve } p) = bbcs\ bb]_b$
using $\langle t \in arch \rangle\ pb.\text{evtE}[of\ t\ t'\ \langle sKs\ p \rightarrow t \rangle_{n_s}]$ **by** *auto*
hence $(p, \text{solve } p) = bbcs\ (\sigma_{the-bb}(t\ n_f))$
using $\langle t \in arch \rangle\ pb.\text{baEA}$ **by** *auto*
moreover have $n_f \geq n_0$
proof —
from $\langle \|sKs\ p\|_t\ n_k \rangle$ **have** $\langle sKs\ p \rightarrow t \rangle_{n_k} \geq n_k$
using *sb.nextActI* **by** *blast*
with $\langle sKs\ p \rightarrow t \rangle_{n_s} = n_s$ **show** *?thesis*
using $\langle n_f \geq \langle sKs\ p \rightarrow t \rangle_{n_s} \rangle\ \langle n_s \geq n_k \rangle\ \langle n_k \geq n \rangle\ \langle n \geq n_0 \rangle$ **by** *arith*
qed
ultimately show *?thesis* **by** *auto*

next
— otherwise, we derive a contradiction
assume *case-ass*: $\forall n' \geq n_k. \|sKs\ p\|_t\ n' \wedge \neg(p, \text{solve } p) = ksns\ (\sigma_{sKs}\ p\ t\ n')$

— first, the KS will eventually register for the subproblems P it requires to solve p...
from $\langle \|sKs\ p\|_t\ n_k \rangle$ **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*
 $([\lambda ks. p \in ksop\ ks]_b \longrightarrow^b (\diamond_b\ [\lambda ks. \exists P. \text{sub } P = ksrp\ ks]_b))$
using *sb.globEA*[*OF - bhvks3*[*of t p sKs p t''*]] $\langle t \in arch \rangle\ sks-prob$ **by** *simp*
moreover have *sb.eval (sKs p) t t'' n_k* $[\lambda ks. p \in ksop\ ks]_b$
proof —
from $\langle \|sKs\ p\|_t\ n_k \rangle$ **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 $\langle \|sKs\ p\|_t\ n_k \rangle$ **have** $\langle sKs\ p \rightarrow t \rangle_{n_k} = n_k$
using *sb.nextAct-active* **by** *blast*
with $\langle p \in ksop(\sigma_{sKs}\ p\ (t\ n_k)) \rangle$ **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* $(\diamond_b\ [\lambda ks. \exists P. \text{sub } P = ksrp\ ks]_b)$
using *sb.impE* **by** *blast*
then obtain n_r **where** $n_r \geq \langle sKs\ p \rightarrow t \rangle_{n_k}$ **and**
 $\exists i \geq n_r. \|sKs\ p\|_t\ i \wedge$
 $(\forall n'' \geq \langle sKs\ p \leftarrow t \rangle_{n_r}. n'' \leq \langle sKs\ p \rightarrow t \rangle_{n_r})$
 $\longrightarrow sb.\text{eval } (sKs\ p)\ t\ t''\ n''\ [\lambda ks. \exists P. \text{sub } P = ksrp\ ks]_b) \vee$
 $\neg (\exists i \geq n_r. \|sKs\ p\|_t\ i) \wedge$
 $sb.\text{eval } (sKs\ p)\ t\ t''\ n_r\ [\lambda ks. \exists P. \text{sub } P = ksrp\ ks]_b$

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

— the knowledgesource will eventually get a solution for each required subproblem:
moreover have *sb.eval* ($sKs p$) $t t'' n_r$ ($\forall_b p'. (sb.pred (p' \in P) \rightarrow^b$
 $(\Diamond_b [\lambda ks. (p', solve p') \in kscs ks]_b))$)

proof —

have $\forall p'. sb.eval (sKs p) t t'' n_r (sb.pred (p' \in P) \rightarrow^b$
 $(\Diamond_b [\lambda ks. (p', solve p') \in kscs ks]_b))$

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* ($p' \in P$)) \rightarrow
 $(sb.eval (sKs p) t t'' n_r$
 $(\Diamond_b [\lambda ks. (p', solve p') \in kscs ks]_b))$

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 $\langle \|sKs p\|_{t n_k} \rangle$ **have** $\exists i \geq 0. \|sKs p\|_{t i}$ **by** *auto*

ultimately have *sb.eval* ($sKs p$) $t t'' n_r$ ($[\lambda ks. sub P = ks rp ks]_b$
 $\rightarrow^b ((\neg^b (\exists_b P'. (sb.pred (p' \in P') \wedge^b [\lambda ks. unsub P' = ks rp ks]_b))) \mathfrak{W}_b$
 $[\lambda ks. (p', solve p') \in kscs ks]_b))$

using *sb.globEA*[*OF - bhvks4*[of $t p' P sKs p t''$]]

$\langle t \in arch \rangle \langle \|sKs p\|_{t n_k} \rangle \langle p' \in P \rangle$ **by** *simp*

with $\langle sb.eval (sKs p) t t'' n_r [\lambda ks. sub P = ks rp ks]_b \rangle$ **have**

sb.eval ($sKs p$) $t t'' n_r$ ($(\neg^b (\exists_b P'. (sb.pred (p' \in P') \wedge^b$
 $[\lambda ks. unsub P' = ks rp ks]_b))) \mathfrak{W}_b [\lambda ks. (p', solve p') \in kscs ks]_b$)

using *sb.impE*[of ($sKs p$) $t t'' n_r$ [$\lambda ks. sub P = ks rp ks$]_b] **by** *blast*

hence *sb.eval* ($sKs p$) $t t'' n_r$ ($(\neg^b (\exists_b P'. (sb.pred (p' \in P') \wedge^b$
 $[\lambda ks. unsub P' = ks rp ks]_b))) \mathfrak{U}_b [\lambda ks. (p', solve p') \in kscs ks]_b \vee$

sb.eval ($sKs p$) $t t'' n_r$ ($\square_b (\neg^b (\exists_b P'. (sb.pred (p' \in P') \wedge^b$
 $[\lambda ks. unsub P' = ks rp ks]_b)))$) **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 $? \gamma = \neg^b (\exists_b P'. (sb.pred (p' \in P') \wedge^b ([\lambda ks. unsub P' = ks rp ks]_b)))$

let $? \gamma = [\lambda ks. (p', solve p') \in kscs ks]_b$

assume *sb.eval* ($sKs p$) $t t'' n_r$ ($? \gamma \mathfrak{U}_b ? \gamma$)

with $\langle \exists i \geq n_r. \|sKs p\|_{t i} \rangle$ **obtain** n' **where** $n' \geq \langle sKs p \rightarrow t \rangle_{n_r}$ **and**

lass: ($\exists i \geq n'. \|sKs p\|_{t i} \wedge (\forall n'' \geq \langle sKs p \leftarrow t \rangle_{n'}. n'' \leq \langle sKs p \rightarrow t \rangle_{n'}$
 $\rightarrow sb.eval (sKs p) t t'' n'' ? \gamma) \wedge$

$(\forall n'' \geq \langle sKs p \leftarrow t \rangle_{n_r}. n'' < \langle sKs p \leftarrow t \rangle_{n'})$

$\rightarrow sb.eval (sKs p) t t'' n'' ?\gamma') \vee$
 $\neg (\exists i \geq n'. \|sKs p\|_t i) \wedge sb.eval (sKs p) t t'' n' ?\gamma \wedge$
 $(\forall n'' \geq \langle sKs p \Leftarrow t \rangle_{n_r}. n'' < n' \rightarrow sb.eval (sKs p) t t'' n'' ?\gamma')$
using *sb.untilEA*[of n_r $sKs p t t''$] $\langle \exists i \geq n_r. \|sKs p\|_t i \rangle$ **by** *blast*
thus *?thesis*
proof cases
assume $\exists i \geq n'. \|sKs p\|_t i$
with lass have $\forall n'' \geq \langle sKs p \Leftarrow t \rangle_{n'}. n'' \leq \langle sKs p \rightarrow t \rangle_{n'}$
 $\rightarrow sb.eval (sKs p) t t'' n'' ?\gamma$ **by** *auto*
moreover have $n' \geq \langle sKs p \Leftarrow t \rangle_{n'}$ **by** *simp*
moreover have $n' \leq \langle sKs p \rightarrow t \rangle_{n'}$
using $\langle \exists i \geq n'. \|sKs p\|_t i \rangle$ *sb.nextActLe* **by** *simp*
ultimately have $sb.eval (sKs p) t t'' n' ?\gamma$ **by** *simp*
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*
ultimately show *?thesis* **using** $\langle \exists i \geq n_r. \|sKs p\|_t i \rangle \langle \exists i \geq n'. \|sKs p\|_t i \rangle$
 $\langle n' \geq \langle sKs p \Leftarrow t \rangle_{n'} \rangle \langle n' \leq \langle sKs p \rightarrow t \rangle_{n'} \rangle$
sb.evtIA[of n_r $sKs p t n' t'' ?\gamma$] **by** *blast*
next
assume $\neg (\exists i \geq n'. \|sKs p\|_t i)$
with lass have $sb.eval (sKs p) t t'' n' ?\gamma \wedge$
 $(\forall n'' \geq \langle sKs p \Leftarrow t \rangle_{n_r}. n'' < n' \rightarrow sb.eval (sKs p) t t'' n'' ?\gamma')$ **by** *auto*
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*
ultimately show *?thesis* **using** $\langle \exists i \geq n_r. \|sKs p\|_t i \rangle \langle \neg (\exists i \geq n'. \|sKs p\|_t i) \rangle$
sb.evtIA[of n_r $sKs p t n' t'' ?\gamma$] **by** *blast*
qed
next
assume *case*: $sb.eval (sKs p) t t'' n_r$
 $(\square_b (\neg^b (\exists_b P'. (sb.pred (p' \in P') \wedge^b [\lambda ks. unsub P' = ksrp ks]_b))))$

have $sub P = ksrp (\sigma_{sKs p} (t (\langle sKs p \rightarrow t \rangle_{n_r}))) \wedge$
 $p' \in P \rightarrow (p', p) \in sb$
proof –
have $\exists i \geq 0. \|sKs p\|_t i$ **using** $\langle \exists i \geq 0. \|sKs p\|_t i \rangle$ **by** *auto*
moreover have $\langle sKs p \Leftarrow t \rangle_0 \leq (\langle sKs p \rightarrow t \rangle_{n_r})$ **by** *simp*
ultimately have $sb.eval (sKs p) t t'' (\langle sKs p \rightarrow t \rangle_{n_r})$
 $[\lambda ks. sub P = ksrp ks \wedge p' \in P \rightarrow (p', p) \in sb]_b$
using *sb.globEA*[OF - *bhvks2*[of $t p sKs p t'' P$]] $\langle t \in arch \rangle$ *sks-prob* **by** *blast*
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.nextActI* **by** *blast*
ultimately show *?thesis*
using *sb.baEANow*[of $sKs p t t'' \langle sKs p \rightarrow t \rangle_{n_r}$] **by** *simp*
qed
with $\langle p' \in P \rangle$ **have** $(p', p) \in sb$
using $\langle sub P = ksrp (\sigma_{sKs p} (t (\langle sKs p \rightarrow t \rangle_{n_r}))) \rangle$
sks-prob **by** *simp*
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.nextActI* **by** *blast*
with $\langle sub P = ksrp (\sigma_{sKs p} (t (\langle sKs p \rightarrow t \rangle_{n_r}))) \rangle$
have $sub P \in bbrp (\sigma_{the-bb} (t (\langle sKs p \rightarrow t \rangle_{n_r})))$
using *conn1A* **by** *auto*
with $\langle p' \in P \rangle$ **have** $sub P \in bbrp (\sigma_{the-bb} (t (\langle sKs p \rightarrow t \rangle_{n_r}))) \wedge p' \in P$ **by** *auto*
ultimately obtain m **where** $m \geq \langle sKs p \rightarrow t \rangle_{n_r}$ **and** $(p', solve p') = bbs (\sigma_{the-bb} (t$

m))

using *indH* by *auto*

- and due to the publisher subscriber property,
- the knowledge source will receive them

moreover have

$$\# n P. \langle sKs p \rightarrow t \rangle_{n_r} \leq n \wedge n \leq m \wedge \|sKs p\|_t n \wedge \\ unsub P = ksrp (\sigma_{sKs p}(t n)) \wedge p' \in P$$

proof

$$\text{assume } \exists n P'. \langle sKs p \rightarrow t \rangle_{n_r} \leq n \wedge n \leq m \wedge \|sKs p\|_t n \wedge \\ unsub P' = ksrp (\sigma_{sKs p}(t n)) \wedge p' \in P'$$

then obtain $n P'$ where

$$\|sKs p\|_t n \text{ and } \langle sKs p \rightarrow t \rangle_{n_r} \leq n \text{ and } n \leq m \text{ and} \\ unsub P' = ksrp (\sigma_{sKs p}(t n)) \text{ and } p' \in P' \text{ by } auto$$

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 $\langle sKs p \Leftarrow t \rangle_{n_r} \leq n$

$$\text{using } \langle n_r \leq \langle sKs p \rightarrow t \rangle_{n_r} \rangle \langle \langle sKs p \Leftarrow t \rangle_{n_r} \leq n_r \rangle$$

$$\langle \langle sKs p \rightarrow t \rangle_{n_r} \leq n \rangle \text{ by } linarith$$

with *case* have *sb.eval* (*sKs p*) *t t'' n* ($\neg^b (\exists_b P'. (sb.pred (p' \in P')$

$$\wedge^b [\lambda ks. unsub P' = ksrp ks]_b))$$

$$\text{using } sb.globEA[of n_r sKs p t t'']$$

$$\neg^b (\exists_b P'. sb.pred (p' \in P') \wedge^b [\lambda ks. unsub P' = ksrp ks]_b) n]$$

$$\langle \exists i \geq n_r. \|sKs p\|_t i \rangle \text{ by } auto$$

ultimately show *False* using *sb.negE* by *auto*

qed

moreover from $\langle \exists i \geq n_r. \|sKs p\|_t i \rangle$ have

$$\|sKs p\|_t (\langle sKs p \rightarrow t \rangle_{n_r}) \text{ using } sb.nextActI \text{ by } blast$$

moreover have *sub P* = *ksrp* ($\sigma_{sKs p}(t (\langle sKs p \rightarrow t \rangle_{n_r}))$)

$$\text{using } \langle sub P = ksrp (\sigma_{sKs p}(t (\langle sKs p \rightarrow t \rangle_{n_r})) \rangle .$$

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 $\langle \exists i \geq n_r. \|sKs p\|_t i \rangle$

$$\text{have } \langle sKs p \rightarrow t \rangle_{n_r} \geq n_r \text{ using } sb.nextActI \text{ by } blast$$

hence $m \geq n_k$ using $\langle \langle sKs p \rightarrow t \rangle_{n_r} \leq m \rangle \langle \langle sKs p \rightarrow t \rangle_{n_k} \leq n_r \rangle$

$$\langle \langle sKs p \rightarrow t \rangle_{n_k} \geq n_k \rangle \text{ by } simp$$

with *case-ass* have $\|sKs p\|_t m$ by *simp*

ultimately have $(p', solve p') \in kscs (\sigma_{sKs p}(t m))$

$$\text{and } \|sKs p\|_t m$$

$$\text{using } \langle t \in arch \rangle msgDelivery[of t sKs p \langle sKs p \rightarrow t \rangle_{n_r} P m p' solve p']$$

$$\langle p' \in P \rangle \text{ by } auto$$

hence *sb.eval* (*sKs p*) *t t'' m* $[\lambda ks. (p', solve p') \in kscs ks]_b$

$$\text{using } sb.baIANow \text{ by } simp$$

moreover have $m \geq \langle sKs p \Leftarrow t \rangle_m$ by *simp*

moreover from $\langle \|sKs p\|_t m \rangle$ have $m \leq \langle sKs p \rightarrow t \rangle_m$

$$\text{using } sb.nextActLe \text{ by } auto$$

moreover from $\langle \exists i \geq n_r. \|sKs p\|_t i \rangle$ have

$$\langle sKs p \Leftarrow t \rangle_{n_r} \leq \langle sKs p \rightarrow t \rangle_{n_r} \text{ by } simp$$

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*

$$(\diamond_b [\lambda ks. (p', solve p') \in kscs ks]_b)$$

$$\text{using } \langle \exists i \geq n_r. \|sKs p\|_t i \rangle sb.evtIA \text{ by } blast$$

qed

qed

qed

thus *sb.eval* (*sKs p*) *t t'' n_r* (*sb.pred* ($p' \in P$) \rightarrow^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 knowledge 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 - bhvks1*[*of t p sKs p t'' P*]] *sks-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.nextActLe* **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.nextActI* **by** *simp*
ultimately have $(p, solve(p)) \in bbns (\sigma_{the-bb} (t (\langle sKs p \rightarrow t \rangle_{n_s})))$
using *conn1*[*OF pb.ts-prop*(2)[*of t*] ($\langle sKs p \rightarrow t \rangle_{n_s}$)] **by** *auto*
hence *pb.eval the-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 *bhvvb1* **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 $\langle t \in arch \rangle$ *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 $\langle t \in arch \rangle$ *pb.evtE*[*of t t'*] $\langle sKs p \rightarrow t \rangle_{n_s}$ **by** *auto*

hence $(p, \text{solve } p) = \text{bbcs } (\sigma_{\text{the-bb}}(t \ n_f))$
 using $\langle t \in \text{arch} \rangle \text{ pb.baEA}$ by *auto*
 moreover have $n_f \geq n_0$
proof –
 from $\langle \exists n'' \geq n_s. \|sKs \ p\|_{t \ n''} \rangle$ have $\langle sKs \ p \rightarrow t \rangle_{n_s \geq n_s}$
 using *sb.nextActLe* by *simp*
 moreover from $\langle n_k \geq n \rangle$ and $\langle \|sKs \ p\|_{t \ n_k} \rangle$ have $\langle sKs \ p \rightarrow t \rangle_{n_k \geq n_k}$
 using *sb.nextActI* by *blast*
 ultimately show *?thesis*
 using $\langle n_f \geq \langle sKs \ p \rightarrow t \rangle_{n_s} \rangle \langle n_s \geq \langle sKs \ p \rightarrow t \rangle_{n_r} \rangle$
 $\langle \langle sKs \ p \rightarrow t \rangle_{n_r} \geq n_r \rangle \langle n_r \geq \langle sKs \ p \rightarrow t \rangle_{n_k} \rangle \langle n_k \geq n \rangle \langle n \geq n_0 \rangle$ by *arith*
qed
 ultimately show *?thesis* by *auto*
qed
qed
qed
qed
qed

theorem *pSolved*:
 fixes t and $t'::\text{nat} \Rightarrow \text{'BB}$ and $t''::\text{nat} \Rightarrow \text{'KS}$
 assumes $t \in \text{arch}$ and
 $\forall n. (\exists n' \geq n. \|sKs \ (\text{bbop}(\sigma_{\text{the-bb}}(t \ n)))\|_{t \ n'})$
 shows
 $\forall n. (\forall P. (\text{sub } P \in \text{bbrp}(\sigma_{\text{the-bb}}(t \ n))$
 $\longrightarrow (\forall p \in P. (\exists m \geq n. (p, \text{solve}(p)) = \text{bbcs } (\sigma_{\text{the-bb}}(t \ m))))))$
 using *assms pSolved-Ind* by *blast*
end

end

4 Some Auxiliary Results

theory *Auxiliary* imports *Main*
begin

lemma *disjE3*: $P \vee Q \vee R \Longrightarrow (P \Longrightarrow S) \Longrightarrow (Q \Longrightarrow S) \Longrightarrow (R \Longrightarrow S) \Longrightarrow S$ by *auto*

lemma *ge-induct*[*consumes 1, case-names step*]:

fixes $i::\text{nat}$ and $j::\text{nat}$ and $P::\text{nat} \Rightarrow \text{bool}$
 shows $i \leq j \Longrightarrow (\bigwedge n. i \leq n \Longrightarrow ((\forall m \geq i. m < n \longrightarrow P \ m) \Longrightarrow P \ n)) \Longrightarrow P \ j$

proof –

assume $a0: i \leq j$ and $a1: (\bigwedge n. i \leq n \Longrightarrow ((\forall m \geq i. m < n \longrightarrow P \ m) \Longrightarrow P \ n))$
 have $(\bigwedge n. \forall m < n. i \leq m \longrightarrow P \ m \Longrightarrow i \leq n \longrightarrow P \ n)$

proof

fix n

assume $a2: \forall m < n. i \leq m \longrightarrow P \ m$

show $i \leq n \Longrightarrow P \ n$

proof –

assume $i \leq n$

with $a1$ have $(\forall m \geq i. m < n \longrightarrow P \ m) \Longrightarrow P \ n$ by *simp*

moreover from $a2$ have $\forall m \geq i. m < n \longrightarrow P \ m$ by *simp*

ultimately show $P \ n$ by *simp*

qed

qed

with *nat-less-induct*[of $\lambda j. i \leq j \longrightarrow P j$] have $i \leq j \longrightarrow P j$.
 with *a0* show *?thesis* by *simp*
 qed

lemma *my-induct*[consumes 1, case-names base step]:

fixes $P::nat \Rightarrow bool$
 assumes *less*: $i \leq j$
 and *base*: $P j$
 and *step*: $\bigwedge n. i \leq n \Longrightarrow n < j \Longrightarrow (\forall n' > n. n' \leq j \longrightarrow P n') \Longrightarrow P n$
 shows $P i$

proof *cases*

assume $j=0$

thus *?thesis* using *less base* by *simp*

next

assume $\neg j=0$

have $j - (j - i) \geq i \longrightarrow P (j - (j - i))$

proof (rule *less-induct*[of $\lambda n::nat. j - n \geq i \longrightarrow P (j - n) j - i$])

fix x assume *asmp*: $\bigwedge y. y < x \Longrightarrow i \leq j - y \longrightarrow P (j - y)$

show $i \leq j - x \longrightarrow P (j - x)$

proof *cases*

assume $x=0$

with *base* show *?thesis* by *simp*

next

assume $\neg x=0$

with $\langle j \neq 0 \rangle$ have $j - x < j$ by *simp*

show *?thesis*

proof

assume $i \leq j - x$

moreover have $\forall n' > j - x. n' \leq j \longrightarrow P n'$

proof

fix n'

show $n' > j - x \longrightarrow n' \leq j \longrightarrow P n'$

proof (rule *HOL.impI*[OF *HOL.impI*])

assume $j - x < n'$ and $n' \leq j$

hence $j - n' < x$ by *simp*

moreover from $\langle i \leq j - x \rangle \langle j - x < n' \rangle$ have $i \leq n'$ using *le-less-trans less-imp-le-nat* by

blast

with $\langle n' \leq j \rangle$ have $i \leq j - (j - n')$ by *simp*

ultimately have $P (j - (j - n'))$ using *asmp* by *simp*

moreover from $\langle n' \leq j \rangle$ have $j - (j - n') = n'$ by *simp*

ultimately show $P n'$ by *simp*

qed

qed

ultimately show $P (j - x)$ using $\langle j - x < j \rangle$ *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 $\exists k. P k \wedge (\forall k'. P k' \longrightarrow k' \leq k)$ shows $\neg(\exists n' > \text{Greatest } P. P n')$

by (metis *Greatest-equality assms less-le-not-le*)

lemma cardEx: assumes *finite A* and *finite B* and $\text{card } A > \text{card } B$ shows $\exists x \in A. \neg x \in B$

proof cases

assume $A \subseteq B$

with *assms* have $\text{card } A \leq \text{card } B$ 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: $\text{card } \{i::\text{nat}. i > n \wedge i \leq n' \wedge p(n'' + i)\} = \text{card } \{i. i > (n + n'') \wedge i \leq (n' + n'') \wedge p i\}$

proof –

let $?f = \lambda i. i + n''$

have *bij-betw* $?f$ $\{i::\text{nat}. i > n \wedge i \leq n' \wedge p(n'' + i)\}$ $\{i. i > (n + n'') \wedge i \leq (n' + n'') \wedge p i\}$

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 $x::\text{nat}$ 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 $y::\text{nat}$ 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 $\langle n + n'' < y \rangle$ have $y = x + n''$ by *simp*

moreover from $\langle x = y - n'' \rangle \langle n + n'' < y \rangle$ have $x > n$ by *simp*

moreover from $\langle x = y - n'' \rangle \langle y \leq n' + n'' \rangle$ have $x \leq n'$ by *simp*

moreover from $\langle y = x + n'' \rangle$ have $y = n'' + x$ by *simp*

with $\langle p y \rangle$ 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 $'s \text{ seq} = \text{nat} \Rightarrow 's$

abbreviation $\text{ccard } n \ n' \ p \equiv \text{card } \{i. i > n \wedge i \leq n' \wedge p i\}$

lemma *ccard-same:*

assumes $\neg p(\text{Suc } n')$

shows $\text{ccard } n \ n' \ p = \text{ccard } n \ (\text{Suc } n') \ p$

proof –

have $\{i. i > n \wedge i \leq \text{Suc } n' \wedge p\ i\} = \{i. i > n \wedge i \leq n' \wedge p\ i\}$
proof
show $\{i. n < i \wedge i \leq \text{Suc } n' \wedge p\ i\} \subseteq \{i. n < i \wedge i \leq n' \wedge p\ i\}$
proof
fix x **assume** $x \in \{i. n < i \wedge i \leq \text{Suc } n' \wedge p\ i\}$
hence $n < x$ **and** $x \leq \text{Suc } n'$ **and** $p\ x$ **by** *auto*
with *assms* (1) **have** $x \neq \text{Suc } n'$ **by** *auto*
with $\langle x \leq \text{Suc } n' \rangle$ **have** $x \leq n'$ **by** *simp*
with $\langle n < x \rangle \langle p\ x \rangle$ **show** $x \in \{i. n < i \wedge i \leq n' \wedge p\ i\}$ **by** *simp*
qed
next
show $\{i. n < i \wedge i \leq n' \wedge p\ i\} \subseteq \{i. n < i \wedge i \leq \text{Suc } n' \wedge p\ i\}$ **by** *auto*
qed
thus *?thesis* **by** *simp*
qed

lemma *ccard-zero*[*simp*]:
fixes $n::\text{nat}$
shows $\text{ccard } n\ n\ p = 0$
by *auto*

lemma *ccard-inc*:
assumes $p\ (\text{Suc } n')$
and $n' \geq n$
shows $\text{ccard } n\ (\text{Suc } n')\ p = \text{Suc } (\text{ccard } n\ n'\ p)$
proof –
let $?A = \{i. i > n \wedge i \leq n' \wedge p\ i\}$
have *finite* $?A$ **by** *simp*
moreover **have** $\text{Suc } n' \notin ?A$ **by** *simp*
ultimately **have** $\text{card } (\text{insert } (\text{Suc } n')\ ?A) = \text{Suc } (\text{card } ?A)$ **using** *card-insert-disjoint*[*of* $?A$] **by**
simp
moreover **have** $\text{insert } (\text{Suc } n')\ ?A = \{i. i > n \wedge i \leq (\text{Suc } n') \wedge p\ i\}$
proof
show $\text{insert } (\text{Suc } n')\ ?A \subseteq \{i. n < i \wedge i \leq \text{Suc } n' \wedge p\ i\}$
proof
fix x **assume** $x \in \text{insert } (\text{Suc } n')\ \{i. n < i \wedge i \leq n' \wedge p\ i\}$
hence $x = \text{Suc } n' \vee n < x \wedge x \leq n' \wedge p\ x$ **by** *simp*
thus $x \in \{i. n < i \wedge i \leq \text{Suc } n' \wedge p\ i\}$
proof
assume $x = \text{Suc } n'$
with *assms* (1) *assms* (2) **show** *?thesis* **by** *simp*
next
assume $n < x \wedge x \leq n' \wedge p\ x$
thus *?thesis* **by** *simp*
qed
qed
next
show $\{i. n < i \wedge i \leq \text{Suc } n' \wedge p\ i\} \subseteq \text{insert } (\text{Suc } n')\ ?A$ **by** *auto*
qed
ultimately **show** *?thesis* **by** *simp*
qed

lemma *ccard-mono*:
assumes $n' \geq n$
shows $n'' \geq n' \implies \text{ccard } n\ (n''::\text{nat})\ p \geq \text{ccard } n\ n'\ p$

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 $n \leq n''$ by *simp*
 ultimately have $\text{ccard } n$ (*Suc* n'') $p = \text{Suc}$ ($\text{ccard } n$ n'' p) using *ccard-inc[of p n'' n]* by *simp*
 also have $\dots \geq \text{ccard } n$ n' p using *step.IH* by *simp*
 finally show *?case .*
next
 assume $\neg p$ (*Suc* n'')
 moreover from *step.hyps assms* have $n \leq n''$ by *simp*
 ultimately have $\text{ccard } n$ (*Suc* n'') $p = \text{ccard } n$ n'' p using *ccard-same[of p n'' n]* by *simp*
 also have $\dots \geq \text{ccard } n$ n' p using *step.IH* by *simp*
 finally show *?case* by *simp*
qed
qed

lemma *ccard-ub[simp]*:
 $\text{ccard } n$ n' $p \leq \text{Suc } n' - n$
proof –
 have $\{i. i > n \wedge i \leq n' \wedge p\ i\} \subseteq \{i. i \geq n \wedge i \leq n'\}$ by *auto*
 hence $\text{ccard } n$ n' $p \leq \text{card } \{i. i \geq n \wedge i \leq n'\}$ by (*simp add: card-mono*)
 moreover have $\{i. i \geq n \wedge i \leq n'\} = \{n..n'\}$ by *auto*
 hence $\text{card } \{i. i \geq n \wedge i \leq n'\} = \text{Suc } n' - n$ by *simp*
 ultimately show *?thesis* by *simp*
qed

lemma *ccard-sum*:
 fixes $n::\text{nat}$
 assumes $n' \geq n''$
 and $n'' \geq n$
 shows $\text{ccard } n$ n' $P = \text{ccard } n$ n'' $P + \text{ccard } n''$ n' P
proof –
 have $\text{ccard } n$ n' $P = \text{card } \{i. i > n \wedge i \leq n' \wedge P\ i\}$ by *simp*
 moreover have $\{i. i > n \wedge i \leq n' \wedge P\ i\} =$
 $\{i. i > n \wedge i \leq n'' \wedge P\ i\} \cup \{i. i > n'' \wedge i \leq n' \wedge P\ i\}$ (is *?LHS = ?RHS*)
proof
 show *?LHS* \subseteq *?RHS* by *auto*
next
 show *?RHS* \subseteq *?LHS*
proof
 fix x
 assume $x \in ?RHS$
 hence $x > n \wedge x \leq n'' \wedge P\ x \vee x > n'' \wedge x \leq n' \wedge P\ x$ by *auto*
 thus $x \in ?LHS$
proof
 assume $n < x \wedge x \leq n'' \wedge P\ x$
 with *assms* show *?thesis* by *simp*
next
 assume $n'' < x \wedge x \leq n' \wedge P\ x$
 with *assms* show *?thesis* by *simp*

qed
 qed
 qed
 hence $\text{card } ?LHS = \text{card } ?RHS$ by *simp*
 ultimately have $\text{ccard } n \ n' \ P = \text{card } ?RHS$ by *simp*
 moreover have $\text{card } ?RHS = \text{card } \{i. i > n \wedge i \leq n'' \wedge P \ i\} + \text{card } \{i. i > n'' \wedge i \leq n' \wedge P \ i\}$
 proof (rule *card-Un-disjoint*)
 show *finite* $\{i. n < i \wedge i \leq n'' \wedge P \ i\}$ by *simp*
 show *finite* $\{i. n'' < i \wedge i \leq n' \wedge P \ i\}$ by *simp*
 show $\{i. n < i \wedge i \leq n'' \wedge P \ i\} \cap \{i. n'' < i \wedge i \leq n' \wedge P \ i\} = \{\}$ by *auto*
 qed
 moreover have $\text{ccard } n \ n'' \ P = \text{card } \{i. i > n \wedge i \leq n'' \wedge P \ i\}$ by *simp*
 moreover have $\text{ccard } n'' \ n' \ P = \text{card } \{i. i > n'' \wedge i \leq n' \wedge P \ i\}$ by *simp*
 ultimately show *?thesis* by *simp*
 qed

lemma *ccard-ex*:

fixes $n::\text{nat}$
 shows $c \geq 1 \implies c < \text{ccard } n \ n'' \ P \implies \exists n' < n''. n' > n \wedge \text{ccard } n \ n' \ P = c$
 proof (induction *c* rule: *dec-induct*)
 let $?l = \text{LEAST } i::\text{nat}. n < i \wedge i < n'' \wedge P \ i$
 case *base*
 moreover have $\text{ccard } n \ n'' \ P \leq \text{Suc } (\text{card } \{i. n < i \wedge i < n'' \wedge P \ i\})$
 proof –
 from $\langle \text{ccard } n \ n'' \ P > 1 \rangle$ have $n'' > n$ using *less-le-trans* by *force*
 then obtain n' where $\text{Suc } n' = n''$ and $\text{Suc } n' \geq n$ by (metis *lessE less-imp-le-nat*)
 moreover have $\{i. n < i \wedge i < \text{Suc } n' \wedge P \ i\} = \{i. n < i \wedge i \leq n' \wedge P \ i\}$ by *auto*
 hence $\text{card } \{i. n < i \wedge i < \text{Suc } n' \wedge P \ i\} = \text{card } \{i. n < i \wedge i \leq n' \wedge P \ i\}$ by *simp*
 moreover have $\text{card } \{i. n < i \wedge i \leq \text{Suc } n' \wedge P \ i\} \leq \text{Suc } (\text{card } \{i. n < i \wedge i \leq n' \wedge P \ i\})$
 proof cases
 assume $P \ (\text{Suc } n')$
 moreover from $\langle n'' > n \rangle \langle \text{Suc } n' = n'' \rangle$ have $n' \geq n$ by *simp*
 ultimately show *?thesis* using *ccard-inc*[of $P \ n' \ n$] by *simp*
 next
 assume $\neg P \ (\text{Suc } n')$
 moreover from $\langle n'' > n \rangle \langle \text{Suc } n' = n'' \rangle$ have $n' \geq n$ by *simp*
 ultimately show *?thesis* using *ccard-same*[of $P \ n' \ n$] by *simp*
 qed
 ultimately show *?thesis* by *simp*
 qed
 ultimately have $\text{card } \{i. n < i \wedge i < n'' \wedge P \ i\} \geq 1$ by *simp*
 hence $\{i. n < i \wedge i < n'' \wedge P \ i\} \neq \{\}$ by *fastforce*
 hence $\exists i. n < i \wedge i < n'' \wedge P \ i$ by *auto*
 hence $?l > n$ and $?l < n''$ and $P \ ?l$ using *LeastI-ex*[of $\lambda i::\text{nat}. n < i \wedge i < n'' \wedge P \ i$] by *auto*
 moreover have $\{i. n < i \wedge i \leq ?l \wedge P \ i\} = \{?l\}$
 proof
 show $\{i. n < i \wedge i \leq ?l \wedge P \ i\} \subseteq \{?l\}$
 proof
 fix i
 assume $i \in \{i. n < i \wedge i \leq ?l \wedge P \ i\}$
 hence $n < i$ and $i \leq ?l$ and $P \ i$ by *auto*
 with $\langle \exists i. n < i \wedge i < n'' \wedge P \ i \rangle$ have $i = ?l$
 using *Least-le*[of $\lambda i. n < i \wedge i < n'' \wedge P \ i$] by (meson *antisym le-less-trans*)
 thus $i \in \{?l\}$ by *simp*
 qed
 qed

next
show $\{?l\} \subseteq \{i. n < i \wedge i \leq ?l \wedge P i\}$
proof
fix i
assume $i \in \{?l\}$
hence $i = ?l$ **by** *simp*
with $\langle ?l > n \rangle \langle ?l < n'' \rangle \langle P ?l \rangle$ **show** $i \in \{i. n < i \wedge i \leq ?l \wedge P i\}$ **by** *simp*
qed
qed
hence $\text{ccard } n \ ?l \ P = 1$ **by** *simp*
ultimately show *?case* **by** *auto*
next
case (*step c*)
moreover from *step.prem*s **have** $\text{Suc } c < \text{ccard } n \ n'' \ P$ **by** *simp*
ultimately obtain n' **where** $n' < n''$ **and** $n < n'$ **and** $\text{ccard } n \ n' \ P = c$ **by** *auto*
hence $\text{ccard } n \ n'' \ P = \text{ccard } n \ n' \ P + \text{ccard } n' \ n'' \ P$ **using** *ccard-sum*[of $n' \ n'' \ n$] **by** *simp*
with $\langle \text{Suc } c < \text{ccard } n \ n'' \ P \rangle \langle \text{ccard } n \ n' \ P = c \rangle$ **have** $\text{ccard } n' \ n'' \ P > 1$ **by** *simp*
moreover have $\text{ccard } n' \ n'' \ P \leq \text{Suc } (\text{card } \{i. n' < i \wedge i < n'' \wedge P i\})$
proof –
from $\langle \text{ccard } n' \ n'' \ P > 1 \rangle$ **have** $n'' > n'$ **using** *less-le-trans* **by** *force*
then obtain n''' **where** $\text{Suc } n''' = n''$ **and** $\text{Suc } n''' \geq n'$ **by** (*metis lessE less-imp-le-nat*)
moreover have $\{i. n' < i \wedge i < \text{Suc } n''' \wedge P i\} = \{i. n' < i \wedge i \leq n''' \wedge P i\}$ **by** *auto*
hence $\text{card } \{i. n' < i \wedge i < \text{Suc } n''' \wedge P i\} = \text{card } \{i. n' < i \wedge i \leq n''' \wedge P i\}$ **by** *simp*
moreover have $\text{card } \{i. n' < i \wedge i \leq \text{Suc } n''' \wedge P i\} \leq \text{Suc } (\text{card } \{i. n' < i \wedge i \leq n''' \wedge P i\})$
proof cases
assume $P (\text{Suc } n''')$
moreover from $\langle n'' > n' \rangle \langle \text{Suc } n''' = n'' \rangle$ **have** $n''' \geq n'$ **by** *simp*
ultimately show *?thesis* **using** *ccard-inc*[of $P \ n''' \ n'$] **by** *simp*
next
assume $\neg P (\text{Suc } n''')$
moreover from $\langle n'' > n' \rangle \langle \text{Suc } n''' = n'' \rangle$ **have** $n''' \geq n'$ **by** *simp*
ultimately show *?thesis* **using** *ccard-same*[of $P \ n''' \ n'$] **by** *simp*
qed
ultimately show *?thesis* **by** *simp*
qed
ultimately have $\text{card } \{i. n' < i \wedge i < n'' \wedge P i\} \geq 1$ **by** *simp*
hence $\{i. n' < i \wedge i < n'' \wedge P i\} \neq \{\}$ **by** *fastforce*
hence $\exists i. n' < i \wedge i < n'' \wedge P i$ **by** *auto*
let $?l = \text{LEAST } i :: \text{nat. } n' < i \wedge i < n'' \wedge P i$
from $\langle \exists i. n' < i \wedge i < n'' \wedge P i \rangle$ **have** $n' < ?l$
using *LeastI-ex*[of $\lambda i :: \text{nat. } n' < i \wedge i < n'' \wedge P i$] **by** *auto*
with $\langle n < n' \rangle$ **have** $\text{ccard } n \ ?l \ P = \text{ccard } n \ n' \ P + \text{ccard } n' \ ?l \ P$ **using** *ccard-sum*[of $n' \ ?l \ n$] **by** *simp*
moreover have $\{i. n' < i \wedge i \leq ?l \wedge P i\} = \{?l\}$
proof
show $\{i. n' < i \wedge i \leq ?l \wedge P i\} \subseteq \{?l\}$
proof
fix i
assume $i \in \{i. n' < i \wedge i \leq ?l \wedge P i\}$
hence $n' < i$ **and** $i \leq ?l$ **and** $P i$ **by** *auto*
with $\langle \exists i. n' < i \wedge i < n'' \wedge P i \rangle$ **have** $i = ?l$
using *Least-le*[of $\lambda i. n' < i \wedge i < n'' \wedge P i$] **by** (*meson antisym le-less-trans*)
thus $i \in \{?l\}$ **by** *simp*
qed
next
show $\{?l\} \subseteq \{i. n' < i \wedge i \leq ?l \wedge P i\}$

proof
fix i
assume $i \in \{?l\}$
hence $i = ?l$ **by** *simp*
moreover from $\langle \exists i. n' < i \wedge i < n'' \wedge P i \rangle$ **have** $?l < n''$ **and** $P ?l$
using *LeastI-ex*[of $\lambda i. n' < i \wedge i < n'' \wedge P i$] **by** *auto*
ultimately show $i \in \{i. n' < i \wedge i \leq ?l \wedge P i\}$ **using** $\langle ?l > n' \rangle$ **by** *simp*
qed
qed
hence $\text{ccard } n' ?l P = 1$ **by** *simp*
ultimately have $\text{card } \{i. n < i \wedge i \leq ?l \wedge P i\} = \text{Suc } c$ **using** $\langle \text{ccard } n n' P = c \rangle$ **by** *simp*
moreover from $\langle \exists i. n' < i \wedge i < n'' \wedge P i \rangle$ **have** $n' < ?l$ **and** $?l < n''$ **and** $P ?l$
using *LeastI-ex*[of $\lambda i::\text{nat}. n' < i \wedge i < n'' \wedge P i$] **by** *auto*
with $\langle n < n' \rangle$ **have** $n < ?l$ **and** $?l < n''$ **by** *auto*
ultimately show *?case* **by** *auto*
qed

lemma *ccard-freq*:
assumes $(n'::\text{nat}) \geq n$
and $\text{ccard } n n' P > \text{ccard } n n' Q + \text{cnf}$
shows $\exists n' n''. \text{ccard } n' n'' P > \text{cnf} \wedge \text{ccard } n' n'' Q \leq \text{cnf}$

proof *cases*
assume $\text{cnf} = 0$
with *assms*(2) **have** $\text{ccard } n n' P > \text{ccard } n n' Q$ **by** *simp*
hence $\text{card } \{i. n < i \wedge i \leq n' \wedge P i\} > \text{card } \{i. n < i \wedge i \leq n' \wedge Q i\}$ (**is** $\text{card } ?LHS > \text{card } ?RHS$)
by *simp*
then obtain i **where** $i \in ?LHS$ **and** $\neg i \in ?RHS$ **and** $i > 0$ **using** *cardEx*[of $?LHS ?RHS$] **by** *auto*
hence $P i$ **and** $\neg Q i$ **by** *auto*
with $\langle i > 0 \rangle$ **obtain** n'' **where** $P (\text{Suc } n'')$ **and** $\neg Q (\text{Suc } n'')$ **using** *gr0-implies-Suc* **by** *auto*
hence $\text{ccard } n'' (\text{Suc } n'') P = 1$ **using** *ccard-inc* **by** *auto*
with $\langle \text{cnf} = 0 \rangle$ **have** $\text{ccard } n'' (\text{Suc } n'') P > \text{cnf}$ **by** *simp*
moreover from $\langle \neg Q (\text{Suc } n'') \rangle$ **have** $\text{ccard } n'' (\text{Suc } n'') Q = 0$ **using** *ccard-same*[of $Q n'' n''$] **by** *auto*
with $\langle \text{cnf} = 0 \rangle$ **have** $\text{ccard } n'' (\text{Suc } n'') Q \leq \text{cnf}$ **by** *simp*
ultimately show *?thesis* **by** *auto*

next
assume $\neg \text{cnf} = 0$
show *?thesis*
proof (*rule ccontr*)
assume $\neg (\exists n' n''. \text{ccard } n' n'' P > \text{cnf} \wedge \text{ccard } n' n'' Q \leq \text{cnf})$
hence *hyp*: $\forall n' n''. \text{ccard } n' n'' Q \leq \text{cnf} \longrightarrow \text{ccard } n' n'' P \leq \text{cnf}$
using *leI less-imp-le-nat* **by** *blast*
show *False*
proof *cases*
assume $\text{ccard } n n' Q \leq \text{cnf}$
with *hyp* **have** $\text{ccard } n n' P \leq \text{cnf}$ **by** *simp*
with *assms* **show** *False* **by** *simp*

next
let $?g\text{cond} = \lambda n''. n'' \geq n \wedge n'' \leq n' \wedge (\exists x \geq 1. \text{ccard } n n'' Q = x * \text{cnf})$
let $?g = \text{GREATEST } n''. ?g\text{cond } n''$
assume $\neg \text{ccard } n n' Q \leq \text{cnf}$
hence $\text{ccard } n n' Q > \text{cnf}$ **by** *simp*
hence $\exists n''. ?g\text{cond } n''$
proof –
from $\langle \text{ccard } n n' Q > \text{cnf} \rangle$ $\langle \neg \text{cnf} = 0 \rangle$ **obtain** n'' **where** $n'' > n$ **and** $n'' \leq n'$ **and** $\text{ccard } n n'' Q$

= cnf

using *ccard-ex*[of *cnf n n' Q*] by *auto*
 moreover from $\langle \text{ccard } n \ n'' \ Q = \text{cnf} \rangle$ have $\exists x \geq 1. \text{ccard } n \ n'' \ Q = x * \text{cnf}$ 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 $\exists x \geq 1. \text{ccard } n \ ?g \ Q = x * \text{cnf}$ and $?g \geq n$ using *GreatestI-ex-nat*[of *?gcond*] by *auto*
 moreover {fix *n''*

have $n'' \geq n \implies \exists x \geq 1. \text{ccard } n \ n'' \ Q = x * \text{cnf} \implies \text{ccard } n \ n'' \ P \leq \text{ccard } n \ n'' \ Q$

proof (*induction n'' rule: ge-induct*)

case (*step n'*)

from *step.prem*s obtain *x* where $x \geq 1$ and *cas*: $\text{ccard } n \ n' \ Q = x * \text{cnf}$ by *auto*

then show *?case*

proof *cases*

assume $x=1$

with *cas* have $\text{ccard } n \ n' \ Q = \text{cnf}$ by *simp*

with *hyp* have $\text{ccard } n \ n' \ P \leq \text{cnf}$ by *simp*

with $\langle \text{ccard } n \ n' \ Q = \text{cnf} \rangle$ show *?thesis* by *simp*

next

assume $\neg x=1$

with $\langle x \geq 1 \rangle$ have $x > 1$ by *simp*

hence $x-1 \geq 1$ by *simp*

moreover from $\langle \text{cnf} \neq 0 \rangle \langle x-1 \geq 1 \rangle$ have $(x-1) * \text{cnf} < x * \text{cnf} \wedge (x-1) * \text{cnf} \neq 0$ by

auto

with $\langle x-1 \geq 1 \rangle \langle \text{cnf} \neq 0 \rangle \langle \text{ccard } n \ n' \ Q = x * \text{cnf} \rangle$ obtain *n''*

where $n'' > n$ and $n'' < n'$ and $\text{ccard } n \ n'' \ Q = (x-1) * \text{cnf}$

using *ccard-ex*[of $(x-1) * \text{cnf} \ n \ n' \ Q$] by *auto*

ultimately have $\exists x \geq 1. \text{ccard } n \ n'' \ Q = x * \text{cnf}$ and $n'' \geq n$ by *auto*

with $\langle n'' \geq n \rangle \langle n'' < n' \rangle$ have $\text{ccard } n \ n'' \ P \leq \text{ccard } n \ n'' \ Q$ using *step.IH* by *simp*

moreover have $\text{ccard } n'' \ n' \ Q = \text{cnf}$

proof -

from $\langle x-1 \geq 1 \rangle$ have $x * \text{cnf} = ((x-1) * \text{cnf}) + \text{cnf}$

using *semiring-normalization-rules*(2)[of $(x-1) \ \text{cnf}$] by *simp*

with $\langle \text{ccard } n \ n'' \ Q = (x-1) * \text{cnf} \rangle \langle \text{ccard } n \ n' \ Q = x * \text{cnf} \rangle$

have $\text{ccard } n \ n' \ Q = \text{ccard } n \ n'' \ Q + \text{cnf}$ by *simp*

moreover from $\langle n'' \geq n \rangle \langle n'' < n' \rangle$ have $\text{ccard } n \ n' \ Q = \text{ccard } n \ n'' \ Q + \text{ccard } n'' \ n' \ Q$

using *ccard-sum*[of $n'' \ n' \ n$] by *simp*

ultimately show *?thesis* by *simp*

qed

moreover from $\langle \text{ccard } n'' \ n' \ Q = \text{cnf} \rangle$ have $\text{ccard } n'' \ n' \ P \leq \text{cnf}$ using *hyp* by *simp*

ultimately show *?thesis* using $\langle n'' \geq n \rangle \langle n'' < n' \rangle$ *ccard-sum*[of $n'' \ n' \ n$] by *simp*

qed

qed } note *geq = this*

ultimately have $\text{ccard } n \ ?g \ P \leq \text{ccard } n \ ?g \ Q$ by *simp*

moreover have $\text{ccard } ?g \ n' \ P \leq \text{cnf}$

proof (*rule ccontr*)

assume $\neg \text{ccard } ?g \ n' \ P \leq \text{cnf}$

hence $\text{ccard } ?g \ n' \ P > \text{cnf}$ by *simp*

have $\text{ccard } ?g \ n' \ Q > \text{cnf}$

proof (*rule ccontr*)

assume $\neg \text{ccard } ?g \ n' \ Q > \text{cnf}$

hence $\text{ccard } ?g \ n' \ Q \leq \text{cnf}$ by *simp*

with $\langle \text{ccard } ?g \ n' \ P > \text{cnf} \rangle$ show *False*


```

    using ⟨ $\neg (\exists n' n''. \text{ccard } n' n'' P > \text{cnf} \wedge \text{ccard } n' n'' Q \leq \text{cnf})$ ⟩ by simp
  qed
  with ⟨ $\neg \text{cnf} = 0$ ⟩ obtain  $n''$  where  $n'' > ?g$  and  $n'' < n'$  and  $\text{ccard } ?g n'' Q = \text{cnf}$ 
    using  $\text{ccard-ex}[of \text{cnf } ?g n' Q]$  by auto
  moreover have  $\exists x \geq 1. \text{ccard } n n'' Q = x * \text{cnf}$ 
  proof -
    from ⟨ $\exists x \geq 1. \text{ccard } n ?g Q = x * \text{cnf}$ ⟩ obtain  $x$  where  $x \geq 1$  and  $\text{ccard } n ?g Q = x * \text{cnf}$  by
  auto
    from ⟨ $n'' > ?g$ ⟩ ⟨ $?g \geq n$ ⟩ have  $\text{ccard } n n'' Q = \text{ccard } n ?g Q + \text{ccard } ?g n'' Q$ 
      using  $\text{ccard-sum}[of ?g n'' n Q]$  by simp
    with ⟨ $\text{ccard } n ?g Q = x * \text{cnf}$ ⟩ have  $\text{ccard } n n'' Q = x * \text{cnf} + \text{ccard } ?g n'' Q$  by simp
    with ⟨ $\text{ccard } ?g n'' Q = \text{cnf}$ ⟩ have  $\text{ccard } n n'' Q = \text{Suc } x * \text{cnf}$  by simp
    thus ?thesis by auto
  qed
  moreover from ⟨ $n'' > ?g$ ⟩ ⟨ $?g \geq n$ ⟩ have  $n'' \geq n$  by simp
  ultimately have  $\exists n'' > ?g. ?g \text{cond } n''$  by auto
  moreover from  $gex$  have  $\forall n'''. ?g \text{cond } n''' \longrightarrow n''' \leq ?g$  using  $\text{Greatest-le-nat}[of ?g \text{cond}]$  by
  auto
    ultimately show False by auto
  qed
  moreover from  $gex$  have  $n' \geq ?g$  using  $\text{GreatestI-ex-nat}[of ?g \text{cond}]$  by auto
  ultimately have  $\text{ccard } n n' P \leq \text{ccard } n n' Q + \text{cnf}$  using  $\text{ccard-sum}[of ?g n' n]$  using ⟨ $?g \geq n$ ⟩
  by simp
  with assms show False by simp
  qed
  qed
  qed
  qed

  locale trusted =
    fixes  $bc:: ('a \text{ list}) \text{ seq}$ 
    and  $n:: \text{nat}$ 
    assumes  $\text{growth}: n' \neq 0 \implies n' \leq n \implies bc \ n' = bc \ (n' - 1) \vee (\exists b. bc \ n' = bc \ (n' - 1) @ b)$ 
  begin
  end

  locale untrusted =
    fixes  $bc:: ('a \text{ list}) \text{ seq}$ 
    and  $\text{mining}:: \text{bool seq}$ 
    assumes  $\text{growth}: \bigwedge n:: \text{nat}. \text{prefix } (bc \ (\text{Suc } n)) \ (bc \ n) \vee (\exists b:: 'a. bc \ (\text{Suc } n) = bc \ n @ [b]) \wedge \text{mining} \ (\text{Suc } n)$ 
  begin

  lemma prefix-save:
    assumes  $\text{prefix } sbc \ (bc \ n')$ 
    and  $\forall n''' > n'. n''' \leq n'' \longrightarrow \text{length } (bc \ n''') \geq \text{length } sbc$ 
    shows  $n'' \geq n' \implies \text{prefix } sbc \ (bc \ n'')$ 
  proof (induction  $n''$  rule: dec-induct)
  case base
  with assms(1) show ?case by simp
  next
  case (step  $n$ )
  from  $\text{growth}[of n]$  show ?case
  proof
    assume  $\text{prefix } (bc \ (\text{Suc } n)) \ (bc \ n)$ 
    moreover from step.hyps have  $\text{length } (bc \ (\text{Suc } n)) \geq \text{length } sbc$  using assms(2) by simp

```

ultimately show ?thesis using step.IH using prefix-length-prefix by auto
next
assume $(\exists b. bc (Suc n) = bc n @ [b]) \wedge mining (Suc n)$
with step.IH show ?thesis by auto
qed
qed

theorem prefix-length:
assumes prefix sbc (bc n') and \neg prefix sbc (bc n'') and $n' \leq n''$
shows $\exists n''' > n'. n''' \leq n'' \wedge length (bc n''') < length sbc$
proof (rule ccontr)
assume $\neg (\exists n''' > n'. n''' \leq n'' \wedge length (bc n''') < length sbc)$
hence $\forall n''' > n'. n''' \leq n'' \longrightarrow length (bc n''') \geq length sbc$ 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:
 $length (bc (Suc n)) \leq Suc (length (bc n))$
by (metis eq-iff growth le-SucI length-append-singleton prefix-length-le)

end

locale untrusted-growth =
fixes bc:: nat seq
and mining:: nat \Rightarrow bool
assumes as1: $\bigwedge n::nat. bc (Suc n) \leq Suc (bc n)$
and as2: $\bigwedge n::nat. bc (Suc n) > bc n \implies mining (Suc n)$
begin

end

sublocale untrusted \subseteq untrusted-growth $\lambda n. length (bc n)$ using grow-mining length-suc-length by unfold-locales auto

context untrusted-growth
begin
theorem ccard-diff-lgth:
 $n' \geq n \implies ccard n n' (\lambda n. mining n) \geq (bc n' - bc n)$
proof (induction n' rule: dec-induct)
case base
then show ?case by simp
next
case (step n')
from as1 have $bc (Suc n') < Suc (bc n') \vee bc (Suc n') = Suc (bc n')$
using le-neq-implies-less by blast
then show ?case
proof
assume $bc (Suc n') < Suc (bc n')$
hence $bc (Suc n') - bc n \leq bc n' - bc n$ by simp

```

moreover from step.hyps have  $\text{ccard } n \text{ (Suc } n') (\lambda n. \text{ mining } n) \geq \text{ccard } n \text{ } n' (\lambda n. \text{ mining } n)$ 
  using ccard-mono[of  $n \text{ } n' \text{ Suc } n'$ ] by simp
ultimately show ?thesis using step.IH by simp
next
  assume  $bc \text{ (Suc } n') = \text{Suc } (bc \text{ } n')$ 
  hence  $bc \text{ (Suc } n') - bc \text{ } n \leq \text{Suc } (bc \text{ } n' - bc \text{ } n)$  by simp
  moreover from  $\langle bc \text{ (Suc } n') = \text{Suc } (bc \text{ } n') \rangle$  have  $\text{mining } (Suc \text{ } n')$  using as2 by simp
  with step.hyps have  $\text{ccard } n \text{ (Suc } n') (\lambda n. \text{ mining } n) \geq \text{Suc } (\text{ccard } n \text{ } n' (\lambda n. \text{ mining } n))$ 
    using ccard-inc by simp
  ultimately show ?thesis using step.IH by simp
qed
qed
end

locale trusted-growth =
  fixes bc:: nat seq
    and mining:: nat  $\Rightarrow$  bool
    and init:: nat
  assumes as1:  $\bigwedge n::\text{nat}. bc \text{ (Suc } n) \geq bc \text{ } n$ 
    and as2:  $\bigwedge n::\text{nat}. \text{mining } (Suc \text{ } n) \Longrightarrow bc \text{ (Suc } n) > bc \text{ } n$ 
begin
  lemma grow-mono:  $n' \geq n \Longrightarrow bc \text{ } n' \geq bc \text{ } n$ 
  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  $n' \geq n \Longrightarrow bc \text{ } n' - bc \text{ } n \geq \text{ccard } n \text{ } n' (\lambda n. \text{ mining } n)$ 
proof (induction  $n'$  rule: dec-induct)
  case base
  then show ?case by simp
next
  case (step  $n'$ )
  then show ?case
proof cases
  assume  $\text{mining } (Suc \text{ } n')$ 
  with as2 have  $bc \text{ (Suc } n') > bc \text{ } n'$  by simp
  moreover from step.hyps have  $bc \text{ } n' \geq bc \text{ } n$  using grow-mono by simp
  ultimately have  $bc \text{ (Suc } n') - bc \text{ } n > bc \text{ } n' - bc \text{ } n$  by simp
  moreover from as1 have  $bc \text{ (Suc } n') - bc \text{ } n \geq bc \text{ } n' - bc \text{ } n$  by (simp add: diff-le-mono)
  moreover from  $\langle \text{mining } (Suc \text{ } n') \rangle$  step.hyps
    have  $\text{ccard } n \text{ (Suc } n') (\lambda n. \text{ mining } n) \leq \text{Suc } (\text{ccard } n \text{ } n' (\lambda n. \text{ mining } n))$ 
      using ccard-inc by simp
    ultimately show ?thesis using step.IH by simp
next
  assume  $\neg \text{mining } (Suc \text{ } n')$ 
  hence  $\text{ccard } n \text{ (Suc } n') (\lambda n. \text{ mining } n) \leq (\text{ccard } n \text{ } n' (\lambda n. \text{ mining } n))$  using ccard-same by simp
  moreover from as1 have  $bc \text{ (Suc } n') - bc \text{ } n \geq bc \text{ } n' - bc \text{ } n$  by (simp add: diff-le-mono)
  ultimately show ?thesis using step.IH by simp
qed
qed

```

end

locale *bounded-growth* = *tg: trusted-growth tbc tmining* + *ug: untrusted-growth ubc umining*
 for *tbc:: nat seq*
 and *ubc:: nat seq*
 and *tmining:: nat ⇒ bool*
 and *umining:: nat ⇒ bool*
 and *sbc::nat*
 and *cnf::nat* +
assumes *fair*: $\bigwedge n n'. \text{ccard } n n' (\lambda n. \text{umining } n) > \text{cnf} \implies \text{ccard } n n' (\lambda n. \text{tmining } n) > \text{cnf}$
 and *a2*: $tbc\ 0 \geq sbc + cnf$
 and *a3*: $ubc\ 0 < sbc$
begin

theorem *tr-upper-bound*: **shows** $ubc\ n < tbc\ n$

proof (*rule ccontr*)

assume $\neg ubc\ n < tbc\ n$

hence $ubc\ n \geq tbc\ n$ **by** *simp*

moreover from *a2 a3* **have** $tbc\ 0 > ubc\ 0 + cnf$ **by** *simp*

moreover have $tbc\ n \geq tbc\ 0$ **using** *tg.grow-mono* **by** *simp*

ultimately have $ubc\ n - ubc\ 0 > tbc\ n - tbc\ 0 + cnf$ **by** *simp*

moreover have $\text{ccard } 0\ n (\lambda n. \text{tmining } n) \leq tbc\ n - tbc\ 0$ **using** *tg.ccard-diff-lgth* **by** *simp*

moreover have $ubc\ n - ubc\ 0 \leq \text{ccard } 0\ n (\lambda n. \text{umining } n)$ **using** *ug.ccard-diff-lgth* **by** *simp*

ultimately have $\text{ccard } 0\ n (\lambda n. \text{umining } n) > \text{ccard } 0\ n (\lambda n. \text{tmining } n) + cnf$ **by** *simp*

hence $\exists n' n''. \text{ccard } n' n'' (\lambda n. \text{umining } n) > cnf \wedge \text{ccard } n' n'' (\lambda n. \text{tmining } n) \leq cnf$

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*:: $('a\ BC)\ set \Rightarrow 'a\ BC \Rightarrow bool$

where *max-cond* $B\ b \equiv b \in B \wedge (\forall b' \in B. \text{length } b' \leq \text{length } b)$

no-syntax

-*MAX1* :: $pttrns \Rightarrow 'b \Rightarrow 'b$ $((\exists MAX\ -./\ -)\ [0, 10]\ 10)$

-*MAX* :: $pttrn \Rightarrow 'a\ set \Rightarrow 'b \Rightarrow 'b$ $((\exists MAX\ -./\ -)\ [0, 0, 10]\ 10)$

-*MAX1* :: $pttrns \Rightarrow 'b \Rightarrow 'b$ $((\exists MAX\ -./\ -)\ [0, 10]\ 10)$

-*MAX* :: $pttrn \Rightarrow 'a\ set \Rightarrow 'b \Rightarrow 'b$ $((\exists MAX\ -\in\ -./\ -)\ [0, 0, 10]\ 10)$

definition *MAX*:: $('a\ BC)\ set \Rightarrow 'a\ BC$

where *MAX* $B = (\text{SOME } b. \text{max-cond } B\ b)$

```

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 λb. 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 λb. 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 \Rightarrow *cnf* \Rightarrow *bool* ($\|-\|$ - [0,110]60)
and *cmp* :: 'nid \Rightarrow *cnf* \Rightarrow 'ND (σ .-) [0,110]60) +
fixes *pin* :: 'ND \Rightarrow ('nid BC) *set*
and *pout* :: 'ND \Rightarrow 'nid BC
and *bc* :: 'ND \Rightarrow 'nid BC
and *mining* :: 'ND \Rightarrow *bool*
and *trusted* :: 'nid \Rightarrow *bool*
and *actTr* :: *cnf* \Rightarrow 'nid *set*
and *actUt* :: *cnf* \Rightarrow 'nid *set*
and *PoW*:: *trace* \Rightarrow *nat* \Rightarrow *nat*
and *tmining*:: *trace* \Rightarrow *nat* \Rightarrow *bool*
and *umining*:: *trace* \Rightarrow *nat* \Rightarrow *bool*
and *cb*:: *nat*
defines *actTr* *k* \equiv {*nid*. $\|nid\|_k \wedge$ *trusted* *nid*}
and *actUt* *k* \equiv {*nid*. $\|nid\|_k \wedge \neg$ *trusted* *nid*}
and *PoW* *t n* \equiv (*LEAST* *x*. \forall *nid* \in *actTr* (*t n*). *length* (*bc* (σ_{nid} (*t n*))) \leq *x*)
and *tmining* *t* \equiv (λ *n*. \exists *nid* \in *actTr* (*t n*). *mining* (σ_{nid} (*t n*)))
and *umining* *t* \equiv (λ *n*. \exists *nid* \in *actUt* (*t n*). *mining* (σ_{nid} (*t n*)))
assumes *consensus*: \bigwedge *nid t t' bc'*::('nid BC). \llbracket *trusted* *nid* $\rrbracket \Longrightarrow$ *eval* *nid t t' 0*
 $(\Box_b (\llbracket \lambda nd. bc' = (\text{if } (\exists b \in \text{pin } nd. \text{length } b > \text{length } (bc \ nd)) \text{ then } (MAX \ (pin \ nd)) \text{ else } (bc \ nd)) \rrbracket_b$
 $\longrightarrow^b \bigcirc_b [\lambda nd. (\neg \text{mining } nd \wedge bc \ nd = bc' \vee \text{mining } nd \wedge (\exists b. bc \ nd = bc' \ @ \ [b]))]_b)$
and *attacker*: \bigwedge *nid t t' bc'*. $\llbracket \neg$ *trusted* *nid* $\rrbracket \Longrightarrow$ *eval* *nid t t' 0*
 $(\Box_b (\llbracket \lambda nd. bc' = (SOME \ b. b \in (pin \ nd \cup \{bc \ nd\})) \rrbracket_b \longrightarrow^b$
 $\bigcirc_b [\lambda nd. (\neg \text{mining } nd \wedge \text{prefix } (bc \ nd) \ bc' \vee \text{mining } nd \wedge (\exists b. bc \ nd = bc' \ @ \ [b]))]_b)$
and *forward*: \bigwedge *nid t t'*. *eval* *nid t t' 0* $(\Box_b [\lambda nd. \text{pout } nd = bc \ nd]_b)$
— At each time point a node will forward its blockchain to the network
and *init*: \bigwedge *nid t t'*. *eval* *nid t t' 0* $[\lambda nd. bc \ nd = []]_b$
and *conn*: \bigwedge *k nid*. $\llbracket \|nid\|_k; \text{trusted } nid \rrbracket$
 \Longrightarrow *pin* (*cmp* *nid k*) = $(\bigcup \text{nid}' \in \text{actTr } k. \{\text{pout } (\text{cmp } \text{nid}' \ k)\})$
and *act*: \bigwedge *t n*::*nat*. *finite* {*nid*::'nid. $\|nid\|_t \ n$ }
and *actTr*: \bigwedge *t n*::*nat*. \exists *nid*. *trusted* *nid* \wedge $\|nid\|_t \ n \wedge$ $\|nid\|_t \ (\text{Suc } n)$
and *fair*: \bigwedge *n n'*. *ccard* *n n'* (*umining* *t*) $>$ *cb* \Longrightarrow *ccard* *n n'* (*tmining* *t*) $>$ *cb*
and *closed*: \bigwedge *t nid b n*::*nat*. $\llbracket \|nid\|_t \ n; b \in \text{pin } (\sigma_{nid}(t \ n)) \rrbracket \Longrightarrow \exists$ *nid'*. $\|nid'\|_t \ n \wedge bc \ (\sigma_{nid'}(t \ n))$
 $= b$
and *mine*: \bigwedge *t nid n*::*nat*. $\llbracket \text{trusted } nid; \|nid\|_t \ (\text{Suc } n); \text{mining } (\sigma_{nid}(t \ (\text{Suc } n))) \rrbracket \Longrightarrow \|nid\|_t \ n$
begin

lemma *init-model*:

assumes $\neg (\exists n'. \text{latestAct-cond } nid \ t \ n \ n')$
and $\|nid\|_t \ n$
shows $bc \ (\sigma_{nid} \ t \ n) = []$

proof —

from *assms*(2) **have** $\exists i \geq 0. \|nid\|_t \ i$ **by** *auto*
with *init* **have** $bc \ (\sigma_{nid} \ t \ \langle nid \rightarrow t \rangle_0) = []$ **using** *baEA*[of 0 *nid t*] **by** *blast*
moreover **from** *assms* **have** $n = \langle nid \rightarrow t \rangle_0$ **using** *nextAct-eq* **by** *simp*
ultimately **show** *?thesis* **by** *simp*

qed

lemma *fwd-bc*:

fixes *nid* **and** *t*::*nat* \Rightarrow *cnf* **and** *t'*::*nat* \Rightarrow 'ND
assumes $\|nid\|_t \ n$

shows $pout (\sigma_{nid} t n) = bc (\sigma_{nid} t n)$
 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 $pin (cmp\ nid (t n)) \subseteq dep \text{ ' } \{nid'. \|nid'\|_t n\}$

proof

fix x assume $x \in pin (cmp\ nid (t n))$

show $x \in dep \text{ ' } \{nid'. \|nid'\|_t n\}$

proof –

from *assms* obtain nid' where $\|nid'\|_t n$ and $bc (\sigma_{nid'}(t n)) = x$

using *closed* $\langle x \in pin (cmp\ nid (t n)) \rangle$ 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 $\langle \|nid'\|_t n \rangle$ 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 *trusted* nid

shows $pin (cmp\ nid (t n)) \neq \{\}$ using *conn*[of $nid\ t\ n$] *act* *assms* *actTr-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.nextActI latestAct-prop(1) latestAct-prop(2) less-le-trans order-refl*)

next

fix i

show $\langle tid \leftarrow t \rangle_n \leq i \wedge i < \langle tid \rightarrow t \rangle_n \wedge \|tid\|_t i \implies i = \langle tid \leftarrow t \rangle_n$

by (*metis* *latestActless(1) leI le-less-Suc-eq le-less-trans nextActI order-refl*)

qed

6.2.1 Component Behavior

lemma *bhv-tr-ex*:

fixes t and $t'::nat \Rightarrow 'ND$ and tid

assumes *trusted* 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 $\neg mining (\sigma_{tid} t \langle tid \rightarrow t \rangle_n) \wedge bc (\sigma_{tid} t \langle tid \rightarrow t \rangle_n) =$

$Blockchain.MAX (pin (\sigma_{tid} t \langle tid \leftarrow t \rangle_n)) \vee mining (\sigma_{tid} t \langle tid \rightarrow t \rangle_n) \wedge$

$(\exists b. bc (\sigma_{tid} t \langle tid \rightarrow t \rangle_n) = Blockchain.MAX (pin (\sigma_{tid} t \langle tid \leftarrow t \rangle_n)) @ [b])$

proof –

let $?cond = \lambda nd. MAX (pin (\sigma_{tidt} \langle tid \leftarrow t \rangle_n)) =$
 (if $(\exists b \in pin \ nd. \ length \ b > \ length \ (bc \ nd))$ then $(MAX (pin \ nd))$ else $(bc \ nd)$)
 let $?check = \lambda nd. \neg \ mining \ nd \wedge \ bc \ nd = MAX (pin (\sigma_{tidt} \langle tid \leftarrow t \rangle_n)) \vee \ mining \ nd \wedge$
 $(\exists b. \ bc \ nd = MAX (pin (\sigma_{tidt} \langle tid \leftarrow t \rangle_n)) \ @ \ [b])$
 from $\langle trusted \ tid \rangle$ have $eval \ tid \ t \ t' \ 0 \ (\Box_b ([?cond]_b \longrightarrow^b \circ_b [?check]_b))$
 using $consensus[of \ tid \ - \ MAX (pin (\sigma_{tidt} \langle tid \leftarrow t \rangle_n))]$ by $simp$
 moreover from $assms$ have $\exists i \geq 0. \ ||tid||_t \ i$ by $auto$
 moreover have $\langle tid \leftarrow t \rangle_0 \leq \langle tid \leftarrow t \rangle_n$ by $simp$
 ultimately have $eval \ tid \ t \ t' \ \langle tid \leftarrow t \rangle_n \ ([?cond]_b \longrightarrow^b \circ_b [?check]_b)$
 using $globEA[of \ 0 \ tid \ t \ t' \ ([?cond]_b \longrightarrow^b \circ_b [?check]_b) \ \langle tid \leftarrow t \rangle_n]$ by $fastforce$
 moreover have $eval \ tid \ t \ t' \ \langle tid \leftarrow t \rangle_n \ [?cond]_b$
proof (rule $baIA$)
 from $\langle \exists n' < n. \ ||tid||_{t \ n'} \rangle$ show $\exists i \geq \langle tid \leftarrow t \rangle_n. \ ||tid||_t \ i$ using $latestAct-prop(1)$ by $blast$
 from $assms(3) \ assms(4)$ show $?cond (\sigma_{tidt} \langle tid \rightarrow t \rangle_{\langle tid \leftarrow t \rangle_n})$ using $latestActNxt$ by $simp$

qed

ultimately have $eval \ tid \ t \ t' \ \langle tid \leftarrow t \rangle_n \ (\circ_b [?check]_b)$
 using $impE[of \ tid \ t \ t' \ - \ [?cond]_b \ \circ_b [?check]_b]$ 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 $baEANow[of \ tid \ t \ t' \ \langle tid \rightarrow t \rangle_n \ [?check]_b]$ by $simp$

qed

lemma $bhv-tr-in$:

fixes t and $t'::nat \Rightarrow 'ND$ and tid

assumes $trusted \ tid$

and $\exists n' \geq n. \ ||tid||_{t \ n'}$

and $\exists n' < n. \ ||tid||_{t \ n'}$

and $\neg (\exists b \in pin \ (\sigma_{tidt} \langle tid \leftarrow t \rangle_n. \ length \ b > \ length \ (bc \ (\sigma_{tidt} \langle tid \leftarrow t \rangle_n)))$

shows $\neg \ mining \ (\sigma_{tidt} \langle tid \rightarrow t \rangle_n) \wedge \ bc \ (\sigma_{tidt} \langle tid \rightarrow t \rangle_n) = bc \ (\sigma_{tidt} \langle tid \leftarrow t \rangle_n) \vee$
 $\ mining \ (\sigma_{tidt} \langle tid \rightarrow t \rangle_n) \wedge (\exists b. \ bc \ (\sigma_{tidt} \langle tid \rightarrow t \rangle_n) = bc \ (\sigma_{tidt} \langle tid \leftarrow t \rangle_n) \ @ \ [b])$

proof –

let $?cond = \lambda nd. \ bc \ (\sigma_{tidt} \langle tid \leftarrow t \rangle_n) = (if \ (\exists b \in pin \ nd. \ length \ b > \ length \ (bc \ nd))$ then $(MAX$
 $(pin \ nd))$ else $(bc \ nd))$

let $?check = \lambda nd. \neg \ mining \ nd \wedge \ bc \ nd = bc \ (\sigma_{tidt} \langle tid \leftarrow t \rangle_n) \vee \ mining \ nd \wedge (\exists b. \ bc \ nd = bc$
 $(\sigma_{tidt} \langle tid \leftarrow t \rangle_n) \ @ \ [b])$

from $\langle trusted \ tid \rangle$ have $eval \ tid \ t \ t' \ 0 \ ((\Box_b ([?cond]_b \longrightarrow^b \circ_b [?check]_b))$

using $consensus[of \ tid \ - \ bc \ (\sigma_{tidt} \langle tid \leftarrow t \rangle_n)]$ by $simp$

moreover from $assms$ have $\exists i \geq 0. \ ||tid||_t \ i$ by $auto$

moreover have $\langle tid \leftarrow t \rangle_0 \leq \langle tid \leftarrow t \rangle_n$ by $simp$

ultimately have $eval \ tid \ t \ t' \ \langle tid \leftarrow t \rangle_n \ ([?cond]_b \longrightarrow^b \circ_b [?check]_b)$

using $globEA[of \ 0 \ tid \ t \ t' \ [?cond]_b \longrightarrow^b \circ_b [?check]_b \ \langle tid \leftarrow t \rangle_n]$ by $fastforce$

moreover have $eval \ tid \ t \ t' \ \langle tid \leftarrow t \rangle_n \ [?cond]_b$

proof (rule baIA)

from $\langle \exists n' < n. \|tid\|_t n' \rangle$ **show** $\exists i \geq \langle tid \leftarrow t \rangle_n. \|tid\|_t i$ **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 $eval\ tid\ t\ t' \langle tid \leftarrow t \rangle_n (\circ_b [?check]_b)$

using *impE*[of $tid\ t\ t' - [?cond]_b \circ_b [?check]_b$] **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-tr-context*:

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

lemma *bhv-ut*:

fixes t **and** $t'::nat \Rightarrow 'ND$ **and** uid
assumes $\neg \text{trusted } uid$
and $\exists n' \geq n. \|uid\|_{t\ n'}$
and $\exists n' < n. \|uid\|_{t\ n'}$
shows $\neg \text{mining}(\sigma_{uid} t \langle uid \rightarrow t \rangle_n) \wedge \text{prefix}(bc(\sigma_{uid} t \langle uid \rightarrow t \rangle_n)) (\text{SOME } b. b \in \text{pin}(\sigma_{uid} t \langle uid \leftarrow t \rangle_n) \cup \{bc(\sigma_{uid} t \langle uid \leftarrow t \rangle_n)\})$
 $\vee \text{mining}(\sigma_{uid} t \langle uid \rightarrow t \rangle_n) \wedge (\exists b. bc(\sigma_{uid} t \langle uid \rightarrow t \rangle_n) = (\text{SOME } b. b \in \text{pin}(\sigma_{uid} t \langle uid \leftarrow t \rangle_n) \cup \{bc(\sigma_{uid} t \langle uid \leftarrow t \rangle_n)\}) @ [b])$

proof –

let $?cond = \lambda nd. (\text{SOME } b. b \in (\text{pin}(\sigma_{uid} t \langle uid \leftarrow t \rangle_n) \cup \{bc(\sigma_{uid} t \langle uid \leftarrow t \rangle_n)\})) = (\text{SOME } b. b \in \text{pin } nd \cup \{bc\ nd\})$

let $?check = \lambda nd. \neg \text{mining } nd \wedge \text{prefix}(bc\ nd) (\text{SOME } b. b \in \text{pin}(\sigma_{uid} t \langle uid \leftarrow t \rangle_n) \cup \{bc(\sigma_{uid} t \langle uid \leftarrow t \rangle_n)\})$

$\vee \text{mining } nd \wedge (\exists b. bc\ nd = (\text{SOME } b. b \in \text{pin}(\sigma_{uid} t \langle uid \leftarrow t \rangle_n) \cup \{bc(\sigma_{uid} t \langle uid \leftarrow t \rangle_n)\}) @ [b])$

from $(\neg \text{trusted } uid)$ **have** $eval\ uid\ t\ t'\ 0 ((\Box_b([?cond]_b \longrightarrow^b \circ_b [?check]_b)))$
using *attacker[of uid - - (SOME b. b ∈ pin(σ_{uid} t ⟨uid ← t⟩_n) ∪ {bc(σ_{uid} t ⟨uid ← t⟩_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 $eval\ uid\ t\ t' \langle uid \leftarrow t \rangle_n (([?cond]_b \longrightarrow^b \circ_b [?check]_b))$

using *globEA[of 0 uid t t' ([?cond]_b →^b ○_b[?check]_b) ⟨uid ← t⟩_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-prop(1)* **by** *blast*

with *assms(3)* **show** $?cond(\sigma_{uid} t \langle uid \rightarrow t \rangle_{\langle uid \leftarrow t \rangle_n})$ **using** *latestActNext* **by** *simp*

qed

ultimately have $eval\ uid\ t\ t' \langle uid \leftarrow t \rangle_n (\circ_b [?check]_b)$

using *impE[of uid t t' - [?cond]_b ○_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** *latestActNextAct* **by** *simp*

with *assms(3)* **have** $\langle uid \rightarrow t \rangle_n > \langle uid \rightarrow t \rangle_{\langle uid \leftarrow t \rangle_n}$ **using** *latestActNext* **by** *simp*

moreover from $\langle \exists n' \geq n. \|uid\|_{t\ n'} \rangle$ **have** $\|uid\|_t \langle uid \rightarrow t \rangle_n$ **using** *nextActI* **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 *latestActNextAct* **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' <uid → t>_n [?check]_b*
using *nextEA1[of uid t <uid ← t>_n t' [?check]_b <uid → t>_n]* **by** *simp*
moreover from $\langle \exists n' \geq n. \|uid\|_t n' \rangle$ **have** $\|uid\|_t \langle uid \rightarrow t \rangle_n$ **using** *nextActI* **by** *simp*
ultimately show *?thesis* **using** *baEANow[of uid t t' <uid → t>_n ?check]* **by** *simp*
qed

lemma *bhv-ut-context*:

assumes \neg *trusted uid*
and $\|uid\|_t n$
and $\exists n' < n. \|uid\|_t n'$
shows $\exists nid'. \|nid'\|_t \langle uid \leftarrow t \rangle_n \wedge (\text{mining } (\sigma_{uid} t n) \wedge (\exists b. \text{prefix } (bc (\sigma_{uid} t n)) (bc (\sigma_{nid'} t \langle uid \leftarrow t \rangle_n)) @ [b])))$
 $\vee \neg \text{mining } (\sigma_{uid} t n) \wedge \text{prefix } (bc (\sigma_{uid} t n)) (bc (\sigma_{nid'} t \langle uid \leftarrow t \rangle_n))$

proof –

let *?bc = SOME b. b ∈ 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 ∈ 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 $\exists b. b \in \text{pin } (\sigma_{uid} t \langle uid \leftarrow t \rangle_n) \cup \{bc (\sigma_{uid} t \langle uid \leftarrow t \rangle_n)\}$ **by** *auto*
hence *?bc ∈ 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 $\neg \text{mining } (\sigma_{uid} t \langle uid \rightarrow t \rangle_n) \wedge \text{prefix } (bc (\sigma_{uid} t \langle uid \rightarrow t \rangle_n)) ?bc \vee$
 $\text{mining } (\sigma_{uid} t \langle uid \rightarrow t \rangle_n) \wedge (\exists b. bc (\sigma_{uid} t \langle uid \rightarrow t \rangle_n) = ?bc @ [b])$
using *bhv-ut[of uid n t]* *assms(1)* **by** *simp*
moreover from *assms(2)* **have** $\langle uid \rightarrow t \rangle_n = n$ **using** *nextAct-active* **by** *simp*
ultimately have *casmp: ¬ mining* $(\sigma_{uid} t n) \wedge \text{prefix } (bc (\sigma_{uid} t n)) ?bc \vee$
 $\text{mining } (\sigma_{uid} t n) \wedge (\exists b. bc (\sigma_{uid} t n) = ?bc @ [b])$ **by** *simp*

from *bc-ex* **have** $?bc \in \text{pin } (\sigma_{uid} t \langle uid \leftarrow t \rangle_n) \vee ?bc \in \{bc (\sigma_{uid} t \langle uid \leftarrow t \rangle_n)\}$.

thus *?thesis*

proof

assume $?bc \in \text{pin } (\sigma_{uid} t \langle uid \leftarrow t \rangle_n)$
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** $\neg \text{mining } (\sigma_{uid} t n) \wedge \text{prefix } (bc (\sigma_{uid} t n)) (bc (\sigma_{nid} t \langle uid \leftarrow t \rangle_n)) \vee$
 $\text{mining } (\sigma_{uid} t n) \wedge (\exists b. bc (\sigma_{uid} t n) = (bc (\sigma_{nid} t \langle uid \leftarrow t \rangle_n)) @ [b])$ **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 $\langle \exists n'. \text{latestAct-cond } uid t n n' \rangle$ **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 Trusted Blockchains

abbreviation $mbc\text{-}cond:: trace \Rightarrow nat \Rightarrow 'nid \Rightarrow bool$

where $mbc\text{-}cond\ t\ n\ nid \equiv nid \in actTr\ (t\ n) \wedge (\forall nid' \in actTr\ (t\ n). length\ (bc\ (\sigma_{nid'}(t\ n))) \leq length\ (bc\ (\sigma_{nid}(t\ n))))$

lemma $mbc\text{-}ex$:

fixes $t\ n$

shows $\exists x. mbc\text{-}cond\ t\ n\ x$

proof –

let $?ALL = \{b. \exists nid \in actTr\ (t\ n). b = bc\ (\sigma_{nid}(t\ n))\}$

have $MAX\ ?ALL \in ?ALL$

proof (*rule max-prop*)

from $actTr$ **have** $actTr\ (t\ n) \neq \{\}$ **using** $actTr\text{-}def$ **by** $blast$

thus $?ALL \neq \{\}$ **by** $auto$

from act **have** $finite\ (actTr\ (t\ n))$ **using** $actTr\text{-}def$ **by** $simp$

thus $finite\ ?ALL$ **by** $simp$

qed

then obtain nid **where** $nid \in actTr\ (t\ n) \wedge bc\ (\sigma_{nid}(t\ n)) = MAX\ ?ALL$ **by** $auto$

moreover have $\forall nid' \in actTr\ (t\ n). length\ (bc\ (\sigma_{nid'}(t\ n))) \leq length\ (MAX\ ?ALL)$

proof

fix nid

assume $nid \in actTr\ (t\ n)$

hence $bc\ (\sigma_{nid}(t\ n)) \in ?ALL$ **by** $auto$

moreover have $\forall b' \in ?ALL. length\ b' \leq length\ (MAX\ ?ALL)$

proof (*rule max-prop*)

from $\langle bc\ (\sigma_{nid}(t\ n)) \in ?ALL \rangle$ **show** $?ALL \neq \{\}$ **by** $auto$

from act **have** $finite\ (actTr\ (t\ n))$ **using** $actTr\text{-}def$ **by** $simp$

thus $finite\ ?ALL$ **by** $simp$

qed

ultimately show $length\ (bc\ (\sigma_{nid}(t\ n))) \leq length\ (Blockchain.MAX\ \{b. \exists nid \in actTr\ (t\ n). b = bc\ (\sigma_{nid}(t\ n))\})$ **by** $simp$

qed

ultimately show $?thesis$ **by** $auto$

qed

definition $MBC:: trace \Rightarrow nat \Rightarrow 'nid$

where $MBC\ t\ n = (SOME\ b. mbc\text{-}cond\ t\ n\ b)$

lemma $mbc\text{-}prop[simp]$:

shows $mbc\text{-}cond\ t\ n\ (MBC\ t\ n)$

using $someI\text{-}ex[OF\ mbc\text{-}ex]$ $MBC\text{-}def$ **by** $simp$

6.2.3 Trusted Proof of Work

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

abbreviation $pow\text{-}cond:: trace \Rightarrow nat \Rightarrow nat \Rightarrow bool$

where $pow\text{-}cond\ t\ n\ n' \equiv \forall nid \in actTr\ (t\ n). length\ (bc\ (\sigma_{nid}(t\ n))) \leq n'$

lemma $pow\text{-}ex$:

fixes $t\ n$

shows $pow\text{-}cond\ t\ n\ (length\ (bc\ (\sigma_{MBC\ t\ n}(t\ n))))$

and $\forall x'. pow\text{-}cond\ t\ n\ x' \longrightarrow x' \geq length\ (bc\ (\sigma_{MBC\ t\ n}(t\ n)))$

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 $\exists tid \in actTr (t n). length (bc (\sigma_{tid}(t n))) = x$
and $\forall tid \in actTr (t n). length (bc (\sigma_{tid}(t n))) \leq x$
shows *PoW t n = x*

proof –
have *(LEAST x. pow-cond t n x) = x*
proof (*rule Least-equality*)
from *assms(2)* **show** $\forall nid \in actTr (t n). length (bc (\sigma_{nid} t n)) \leq x$ **by** *simp*
next
fix *y*
assume $\forall nid \in actTr (t n). length (bc (\sigma_{nid} t n)) \leq y$
thus $x \leq y$ **using** *assms(1)* **by** *auto*
qed
with *PoW-def* **show** *?thesis* **by** *simp*
qed

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

lemma *pow-less*:
fixes *t n nid*
assumes *pow-cond t n x*
shows $PoW t n \leq x$

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 *trusted tid*
and $\|tid\|_t n$
shows $PoW t n \leq length (MAX (pin (\sigma_{tid} t n)))$

proof –
from *mbc-prop* **have** *trusted (MBC t n)* **and** $\|MBC t n\|_t n$ **using** *actTr-def* **by** *auto*
hence $pout (\sigma_{MBC} t n t n) = bc (\sigma_{MBC} t n t n)$
using *forward globEANow*[*THEN baEANow*[of *MBC t n t' n* $\lambda nd. pout nd = bc nd$]] **by** *auto*
with *assms* $\langle \|MBC t n\|_t n \rangle \langle trusted (MBC t n) \rangle$ **have** $bc (\sigma_{MBC} t n t n) \in pin (\sigma_{tid} t n)$
using *conn actTr-def* **by** *auto*
moreover from *assms (2)* **have** *finite (pin (\sigma_{tid} t n))* **using** *finite-input*[of *tid t n*] **by** *simp*
ultimately have $length (bc (\sigma_{MBC} t n t n)) \leq length (MAX (pin (\sigma_{tid} t n)))$
using *max-prop(2)* **by** *auto*
with *pow-mbc* **show** *?thesis* **by** *simp*
qed

lemma *pow-ge-lgth*:

assumes *trusted tid*

and $\|tid\|_t n$

shows $length (bc (\sigma_{tid} t n)) \leq PoW t n$

proof –

from *assms* **have** $tid \in actTr (t n)$ **using** *actTr-def* **by** *simp*

thus *?thesis* **using** *pow-prop* **by** *simp*

qed

lemma *pow-le-lgth*:

assumes *trusted 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 $n' \geq n \implies PoW t n' \geq PoW t n$

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 *actTr* **obtain** *tid* **where** *trusted 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 *nextAct-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 $\langle trusted tid \rangle$ *bhv-tr-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 $\langle trusted tid \rangle$ $\langle \|tid\|_t n' \rangle$

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 $\langle trusted tid \rangle$ $\langle \|tid\|_t (Suc n') \rangle$

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*: $\neg(\exists b \in \text{pin}(\sigma_{tid} t n'). \text{length } b > \text{length}(bc(\sigma_{tid} t n')))$
moreover from $\langle \|\text{tid}\|_t (Suc\ n') \rangle$ **have** $\langle \text{tid} \rightarrow t \rangle_{Suc\ n'} = Suc\ n'$
using *nextAct-active* **by** *simp*
moreover from $\langle \|\text{tid}\|_t n' \rangle$ **have** $\langle \text{tid} \leftarrow t \rangle_{Suc\ n'} = n'$
using *latestAct-prop(2)* *latestActless* *le-less-Suc-eq* **by** *blast*
moreover from $\langle \|\text{tid}\|_t n' \rangle$ **have** $\exists n'' < Suc\ n'. \|\text{tid}\|_t n''$ **by** *blast*
moreover from $\langle \|\text{tid}\|_t (Suc\ n') \rangle$ **have** $\exists n'' \geq Suc\ n'. \|\text{tid}\|_t n''$ **by** *auto*
ultimately have $bc(\sigma_{tid} t (Suc\ n')) = bc(\sigma_{tid} t n') \vee$
 $(\exists b. bc(\sigma_{tid} t (Suc\ n')) = bc(\sigma_{tid} t n') @ b)$
using *(trusted tid)* *bhv-tr-in[of tid Suc n' t]* **by** *auto*
hence $\text{length}(bc(\sigma_{tid} t (Suc\ n'))) \geq \text{length}(bc(\sigma_{tid} t n'))$ **by** *auto*
moreover from $\langle \text{trusted tid} \rangle \langle \|\text{tid}\|_t n' \rangle$ *asmp* **have** $\text{length}(bc(\sigma_{tid} t n')) \geq PoW\ t\ n'$
using *pow-le-lgth* **by** *simp*
moreover from $\langle \text{trusted tid} \rangle \langle \|\text{tid}\|_t (Suc\ n') \rangle$
have $\text{length}(bc(\sigma_{tid} t (Suc\ n'))) \leq PoW\ t\ (Suc\ n')$ **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 $n' \geq n$
and $n'' \geq n$
and $n'' \leq n'$
shows $PoW\ t\ n = PoW\ t\ n''$ **by** *(metis pow-mono assms(1) assms(3) assms(4) eq-iff)*

lemma *pow-mining-suc*:

assumes *tmining t (Suc n)*
shows $PoW\ t\ n < PoW\ t\ (Suc\ n)$

proof –

from *assms* **obtain** *nid* **where** $nid \in \text{actTr}(t(Suc\ n))$ **and** *mining* $(\sigma_{nid} t(Suc\ n))$
using *tmining-def* **by** *auto*
show *?thesis*

proof *cases*

assume *asmp*: $(\exists b \in \text{pin}(\sigma_{nid} t \langle \text{nid} \leftarrow t \rangle_{Suc\ n}). \text{length } b > \text{length}(bc(\sigma_{nid} t \langle \text{nid} \leftarrow t \rangle_{Suc\ n})))$
moreover from $\langle \text{nid} \in \text{actTr}(t(Suc\ n)) \rangle$ **have** *trusted nid* **and** $\|\text{nid}\|_t (Suc\ n)$
using *actTr-def* **by** *auto*
moreover from $\langle \text{trusted nid} \rangle \langle \text{mining}(\sigma_{nid} t(Suc\ n)) \rangle \langle \|\text{nid}\|_t (Suc\ n) \rangle$ **have** $\|\text{nid}\|_t n$
using *mine* **by** *simp*
hence $\exists n'. \text{latestAct-cond } nid\ t\ (Suc\ n)\ n'$ **by** *auto*
ultimately have $\neg \text{mining}(\sigma_{nid} t \langle \text{nid} \rightarrow t \rangle_{Suc\ n}) \wedge bc(\sigma_{nid} t \langle \text{nid} \rightarrow t \rangle_{Suc\ n}) = MAX(\text{pin}(\sigma_{nid} t \langle \text{nid} \leftarrow t \rangle_{Suc\ n})) \vee$
 $\text{mining}(\sigma_{nid} t \langle \text{nid} \rightarrow t \rangle_{Suc\ n}) \wedge (\exists b. bc(\sigma_{nid} t \langle \text{nid} \rightarrow t \rangle_{Suc\ n}) = MAX(\text{pin}(\sigma_{nid} t \langle \text{nid} \leftarrow t \rangle_{Suc\ n})) @ [b])$ **using** *bhv-tr-ex[of nid Suc n]* **by** *auto*
moreover from $\langle \|\text{nid}\|_t (Suc\ n) \rangle$ **have** $\langle \text{nid} \rightarrow t \rangle_{Suc\ n} = Suc\ n$ **using** *nextAct-active* **by** *simp*
moreover have $\langle \text{nid} \leftarrow t \rangle_{Suc\ n} = n$
proof *(rule latestActEq)*
from $\langle \|\text{nid}\|_t n \rangle$ **show** $\|\text{nid}\|_t n$ **by** *simp*
show $\neg(\exists n'' > n. n'' < Suc\ n \wedge \|\text{nid}\|_t n)$ **by** *simp*
show $n < Suc\ n$ **by** *simp*
qed
hence $\langle \text{nid} \leftarrow t \rangle_{Suc\ n} = n$ **using** *latestAct-def* **by** *simp*

ultimately have $\neg \text{mining}(\sigma_{nidt}(Suc\ n)) \wedge bc(\sigma_{nidt}(Suc\ n)) = MAX(\text{pin}(\sigma_{nidt}\ n)) \vee$
 $\text{mining}(\sigma_{nidt}(Suc\ n)) \wedge (\exists b. bc(\sigma_{nidt}(Suc\ n)) = MAX(\text{pin}(\sigma_{nidt}\ n)) @ [b])$ **by simp**
with $\langle \text{mining}(\sigma_{nid}(t(Suc\ n))) \rangle$
have $\exists b. bc(\sigma_{nidt}(Suc\ n)) = MAX(\text{pin}(\sigma_{nidt}\ n)) @ [b]$ **by auto**
moreover from $\langle \text{trusted}\ nid \rangle \langle \|\text{nid}\|_t(Suc\ n) \rangle$ **have** $\text{length}(bc(\sigma_{nidt}(Suc\ n))) \leq PoW\ t(Suc\ n)$
using *pow-ge-lgth*[of $\text{nid}\ t\ Suc\ n$] **by simp**
ultimately have $\text{length}(MAX(\text{pin}(\sigma_{nidt}\ n))) < PoW\ t(Suc\ n)$ **by auto**
moreover from $\langle \text{trusted}\ nid \rangle \langle \|\text{nid}\|_t\ n \rangle$ **have** $\text{length}(MAX(\text{pin}(\sigma_{nidt}\ n))) \geq PoW\ t\ n$
using *pow-le-max* **by simp**
ultimately show *?thesis* **by simp**

next

assume *asmp*: $\neg (\exists b \in \text{pin}(\sigma_{nidt}(\text{nid} \leftarrow t)_{Suc\ n}). \text{length}\ b > \text{length}(bc(\sigma_{nidt}(\text{nid} \leftarrow t)_{Suc\ n})))$
moreover from $\langle \text{nid} \in \text{actTr}(t(Suc\ n)) \rangle$ **have** *trusted nid* **and** $\|\text{nid}\|_t(Suc\ n)$
using *actTr-def* **by auto**
moreover from $\langle \text{trusted}\ nid \rangle \langle \text{mining}(\sigma_{nid}(t(Suc\ n))) \rangle \langle \|\text{nid}\|_t(Suc\ n) \rangle$ **have** $\|\text{nid}\|_t\ n$
using *mine* **by simp**
hence $\exists n'. \text{latestAct-cond}\ \text{nid}\ t(Suc\ n)\ n'$ **by auto**
ultimately have $\neg \text{mining}(\sigma_{nidt}(\text{nid} \rightarrow t)_{Suc\ n}) \wedge bc(\sigma_{nidt}(\text{nid} \rightarrow t)_{Suc\ n}) = bc(\sigma_{nidt}(\text{nid} \leftarrow t)_{Suc\ n}) \vee$
 $\text{mining}(\sigma_{nidt}(\text{nid} \rightarrow t)_{Suc\ n}) \wedge (\exists b. bc(\sigma_{nidt}(\text{nid} \rightarrow t)_{Suc\ n}) = bc(\sigma_{nidt}(\text{nid} \leftarrow t)_{Suc\ n}) @ [b])$
using *bhv-tr-in*[of $\text{nid}\ Suc\ n$] **by auto**
moreover from $\langle \|\text{nid}\|_t(Suc\ n) \rangle$ **have** $\langle \text{nid} \rightarrow t \rangle_{Suc\ n} = Suc\ n$ **using** *nxtAct-active* **by simp**
moreover have $\langle \text{nid} \leftarrow t \rangle_{Suc\ n} = n$
proof (*rule latestActEq*)
from $\langle \|\text{nid}\|_t\ n \rangle$ **show** $\|\text{nid}\|_t\ n$ **by simp**
show $\neg (\exists n'' > n. n'' < Suc\ n \wedge \|\text{nid}\|_t\ n)$ **by simp**
show $n < Suc\ n$ **by simp**

qed

hence $\langle \text{nid} \leftarrow t \rangle_{Suc\ n} = n$ **using** *latestAct-def* **by simp**

ultimately have $\neg \text{mining}(\sigma_{nidt}(Suc\ n)) \wedge bc(\sigma_{nidt}(Suc\ n)) = bc(\sigma_{nidt}\ n) \vee$
 $\text{mining}(\sigma_{nidt}(Suc\ n)) \wedge (\exists b. bc(\sigma_{nidt}(Suc\ n)) = bc(\sigma_{nidt}\ n) @ [b])$ **by simp**
with $\langle \text{mining}(\sigma_{nid}(t(Suc\ n))) \rangle$ **have** $\exists b. bc(\sigma_{nidt}(Suc\ n)) = bc(\sigma_{nidt}\ n) @ [b]$ **by simp**
moreover from $\langle \langle \text{nid} \leftarrow t \rangle_{Suc\ n} = n \rangle$
have $\neg (\exists b \in \text{pin}(\sigma_{nidt}\ n). \text{length}(bc(\sigma_{nidt}\ n)) < \text{length}\ b)$
using *asmp* **by simp**
with $\langle \text{trusted}\ nid \rangle \langle \|\text{nid}\|_t\ n \rangle$ **have** $\text{length}(bc(\sigma_{nidt}\ n)) \geq PoW\ t\ n$
using *pow-le-lgth*[of $\text{nid}\ t\ n$] **by simp**
moreover from $\langle \text{trusted}\ nid \rangle \langle \|\text{nid}\|_t(Suc\ n) \rangle$ **have** $\text{length}(bc(\sigma_{nidt}(Suc\ n))) \leq PoW\ t(Suc\ n)$
using *pow-ge-lgth*[of $\text{nid}\ t\ Suc\ n$] **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* $t\ n\ \text{nid}\ n'\ \text{nid}'\ x \equiv$

$$\begin{aligned}
& (\exists n. \text{latestAct-cond}\ \text{nid}'\ t\ n'\ n) \wedge \|\text{snd}\ x\|_t(\text{fst}\ x) \wedge \text{fst}\ x = \langle \text{nid}' \leftarrow t \rangle_{n'} \wedge \\
& (\text{prefix}(bc(\sigma_{nid'}(t\ n')))(bc(\sigma_{\text{snd}\ x}(t(\text{fst}\ x)))) \vee \\
& (\exists b. bc(\sigma_{\text{nid}'}(t\ n')) = (bc(\sigma_{\text{snd}\ x}(t(\text{fst}\ x)))) @ [b] \wedge \text{mining}(\sigma_{\text{nid}'}(t\ n')))
\end{aligned}$$

inductive-set

$his:: trace \Rightarrow nat \Rightarrow 'nid \Rightarrow (nat \times 'nid) set$
for $t::trace$ **and** $n::nat$ **and** $nid::'nid$
where $\llbracket \llbracket nid \rrbracket_t n \rrbracket \Longrightarrow (n, nid) \in his\ t\ n\ nid$
 $\mid \llbracket (n', nid') \in his\ t\ n\ nid; \exists x. his-prop\ t\ n\ nid\ n'\ nid'\ x \rrbracket \Longrightarrow (SOME\ x. his-prop\ t\ n\ nid\ n'\ nid'\ x) \in his\ t\ n\ nid$

lemma *his-act*:

assumes $(n', nid') \in his\ t\ n\ nid$
shows $\llbracket nid' \rrbracket_t n'$
using *assms*

proof (*rule his.cases*)

assume $(n', nid') = (n, nid)$ **and** $\llbracket nid \rrbracket_t n$
thus $\llbracket nid' \rrbracket_t n'$ **by** *simp*

next

fix $n''\ nid''$ **assume** *asmp*: $(n', nid') = (SOME\ x. his-prop\ t\ n\ nid\ n''\ nid''\ x)$

and $(n'', nid'') \in his\ t\ n\ nid$ **and** $\exists x. his-prop\ t\ n\ nid\ n''\ nid''\ x$

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 $\llbracket snd\ (SOME\ x. his-prop\ t\ n\ nid\ n''\ nid''\ x) \rrbracket_t (fst\ (SOME\ x. his-prop\ t\ n\ nid\ n''\ nid''\ x))$

by *blast*

moreover from *asmp* **have** $fst\ (SOME\ x. his-prop\ t\ n\ nid\ n''\ nid''\ x) = fst\ (n', nid')$ **by** *simp*

moreover from *asmp* **have** $snd\ (SOME\ x. his-prop\ t\ n\ nid\ n''\ nid''\ x) = snd\ (n', nid')$ **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 $\neg(\exists x \in his\ t\ n\ nid. fst\ x < n' \wedge fst\ x > hisPred\ t\ n\ nid\ n')$

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.*hyp*s **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 $\langle his\ prop\ t\ n\ nid\ n'\ nid'\ (SOME\ x.\ his\ prop\ t\ n\ nid\ n'\ nid'\ x) \rangle$

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 $(n, nid') \in his\ t\ n\ nid \implies nid' = nid$

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 $\langle his\ prop\ t\ n\ nid\ n'\ nid'a\ (SOME\ x.\ his\ prop\ t\ n\ nid\ n'\ nid'a\ x) \rangle$

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 $\langle (n, nid') = (SOME\ x.\ his\ prop\ t\ n\ nid\ n'\ nid'a\ x) \rangle$ **have** $fst\ (n, nid') < n'$ **by** *simp*

hence $n < n'$ **by** *simp*

moreover from $\langle (n', nid'a) \in his\ t\ n\ nid \rangle$ **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 $\neg n' = n''$
hence $n' > n'' \vee n' < n''$ **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 $n' \leq n \implies (\exists nid'. (n',nid') \in his\ t\ n\ nid) \implies (\exists! nid'. (n',nid') \in his\ t\ n\ nid) \wedge$
 $((\exists n'' < n'. \exists nid'. (n'',nid'') \in his\ t\ n\ nid) \longrightarrow (\exists x. his-prop\ t\ n\ nid\ n' (THE\ nid'. (n',nid') \in his\ t\ n\ nid)\ x) \wedge$
 $(hisPred\ t\ n\ nid\ n', (SOME\ nid'. (hisPred\ t\ n\ nid\ n', nid') \in his\ t\ n\ nid)) = (SOME\ x. his-prop\ t\ n\ nid\ n' (THE\ nid'. (n',nid') \in his\ t\ n\ nid)\ x))$

proof (*induction n' rule: my-induct*)

case *base*

then obtain nid' **where** $(n, nid') \in his\ t\ n\ nid$ **by** *auto*

hence $\exists! nid'. (n, nid') \in his\ t\ n\ nid$

proof

fix nid'' **assume** $(n, nid'') \in his\ t\ n\ nid$

with *his-determ-base* **have** $nid'' = nid$ **by** *simp*

moreover from $(n, nid') \in his\ t\ n\ nid$ **have** $nid'' = nid$ **using** *his-determ-base* **by** *simp*

ultimately show $nid'' = nid'$ **by** *simp*

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*

with $\langle \exists n'' < n. \exists nid'. (n'', nid') \in his\ t\ n\ nid \rangle$ **show** *?thesis* **using** *hisPrev-prop(1)*[of *n t n nid*]
by force
next
fix $n''\ nid''$ **assume** *asmp*: $(hisPred\ t\ n\ nid\ n, SOME\ nid'. (hisPred\ t\ n\ nid\ n, nid') \in his\ t\ n\ nid)$
 $= (SOME\ x. his-prop\ t\ n\ nid\ n''\ nid''\ x)$
and $\langle n'', nid'' \rangle \in his\ t\ n\ nid$ **and** $\exists x. his-prop\ t\ n\ nid\ n''\ nid''\ x$
moreover have $n''=n$
proof (*rule antisym*)
show $n'' \geq n$
proof (*rule ccontr*)
assume $(\neg n'' \geq n)$
hence $n'' < n$ **by simp**
moreover have $n'' > hisPred\ t\ n\ nid\ n$
proof –
let $?x = \lambda x. his-prop\ t\ n\ nid\ n''\ nid''\ x$
from $\langle \exists x. his-prop\ t\ n\ nid\ n''\ nid''\ x \rangle$ **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** $fst\ (hisPred\ t\ n\ nid\ n, SOME\ nid'. (hisPred\ t\ n\ nid\ n, nid') \in his\ t\ n\ nid) = fst\ (SOME\ x. ?x\ x)$ **by simp**
ultimately show *?thesis* **by simp**
qed
moreover from $\langle \exists n'' < n. \exists nid'. (n'', nid') \in his\ t\ n\ nid \rangle$
have $\neg(\exists x \in his\ t\ n\ nid. fst\ x < n \wedge fst\ x > hisPred\ t\ n\ nid\ n)$
using *hisPrev-nex-less* **by simp**
ultimately show *False* **using** $\langle (n'', nid'') \in his\ t\ n\ nid \rangle$ **by auto**
qed
next
from $\langle (n'', nid'') \in his\ t\ n\ nid \rangle$ **show** $n'' \leq n$ **using** *his-le* **by auto**
qed
ultimately have $(hisPred\ t\ n\ nid\ n, SOME\ nid'. (hisPred\ t\ n\ nid\ n, nid') \in his\ t\ n\ nid) = (SOME\ x. his-prop\ t\ n\ nid\ n\ nid''\ x)$ **by simp**
moreover from $\langle n''=n \rangle \langle (n'', nid'') \in his\ t\ n\ nid \rangle$ **have** $\langle n, nid'' \rangle \in his\ t\ n\ nid$ **by simp**
with $\langle \exists! nid'. (n, nid') \in his\ t\ n\ nid \rangle$ **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**
moreover from $\langle \exists x. his-prop\ t\ n\ nid\ n''\ nid''\ x \rangle \langle n''=n \rangle \langle nid'' = (THE\ nid'. (n, nid') \in his\ t\ n\ nid) \rangle$
have $\exists x. his-prop\ t\ n\ nid\ n\ (THE\ nid'. (n, nid') \in his\ t\ n\ nid)\ x$ **by simp**
ultimately show *?thesis* **by simp**
qed
qed
ultimately show *?case* **by simp**
next
case (*step n*)
then obtain nid' **where** $\langle n', nid' \rangle \in his\ t\ n\ nid$ **by auto**
hence $\exists! nid'. (n', nid') \in his\ t\ n\ nid$
proof (*rule his.cases*)
assume $\langle n', nid' \rangle = \langle n, nid \rangle$
hence $n'=n$ **by simp**
with *step.hyps* **show** *?thesis* **by simp**
next
fix $n''''\ nid''''$
assume $\langle n''''\ nid'''' \rangle \in his\ t\ n\ nid$
and $n'nid'$: $\langle n', nid' \rangle = (SOME\ x. his-prop\ t\ n\ nid\ n''''\ nid''''\ x)$
and $\langle n''''\ nid'''' \rangle \in his\ t\ n\ nid$ **and** $\exists x. his-prop\ t\ n\ nid\ n''''\ nid''''\ x$
from $\langle (n', nid') \in his\ t\ n\ nid \rangle$ **show** *?thesis*

proof

fix nid'' **assume** $(n', nid'') \in his\ t\ n\ nid$

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 $n'nid''': (n', nid'') = (SOME\ x.\ his\ prop\ t\ n\ nid\ n'''\ nid'''\ x)$

and $(n''', nid''') \in his\ t\ n\ nid$ **and** $\exists x.\ his\ prop\ t\ n\ nid\ n'''\ nid'''\ x$

moreover **have** $n'''=n''''$

proof –

have $hisPred\ t\ n\ nid\ n'''' = n'$

proof –

from $n'nid'' \langle \exists x.\ his\ prop\ t\ n\ nid\ n'''\ nid'''\ x \rangle$

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** $\langle (n''', nid''') \in his\ t\ n\ nid \rangle$ **have** $n''' \leq n$ **using** *his-le* **by** *auto*

moreover **from** $\langle (n''', nid''') \in his\ t\ n\ nid \rangle$

have $\exists nid'. (n''', nid') \in his\ t\ n\ nid$ **by** *auto*

ultimately **have** $(\exists n' < n'''. \exists nid'. (n',nid') \in his\ t\ n\ nid) \longrightarrow (\exists !nid'. (n''',nid') \in his\ t\ n\ nid) \wedge (hisPred\ t\ n\ nid\ n''', (SOME\ nid'. (hisPred\ t\ n\ nid\ n''', nid') \in his\ t\ n\ nid)) = (SOME\ x.\ his\ prop\ t\ n\ nid\ n'''\ (THE\ nid'. (n''',nid') \in his\ t\ n\ nid)\ x)$ **using** *step.IH* **by** *auto*

with $\langle n''' > n' \rangle \langle (n', nid') \in his\ t\ n\ nid \rangle$ **have** $\exists !nid'. (n''',nid') \in his\ t\ n\ nid$ **and** $(hisPred\ t\ n\ nid\ n''', (SOME\ nid'. (hisPred\ t\ n\ nid\ n''', nid') \in his\ t\ n\ nid)) = (SOME\ x.\ his\ prop\ t\ n\ nid\ n'''\ (THE\ nid'. (n''',nid') \in his\ t\ n\ nid)\ x)$ **by** *auto*

moreover **from** $\langle \exists !nid'. (n''',nid') \in his\ t\ n\ nid \rangle \langle (n''', nid''') \in his\ t\ n\ nid \rangle$ **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** $(hisPred\ t\ n\ nid\ n''', (SOME\ nid'. (hisPred\ t\ n\ nid\ n''', nid') \in his\ t\ n\ nid)) = (SOME\ x.\ his\ prop\ t\ n\ nid\ n'''\ nid'''\ x)$ **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 $n'nid'' \langle \exists x.\ his\ prop\ t\ n\ nid\ n'''' nid'''' x \rangle$ **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** $\langle (n'''', nid''') \in his\ t\ n\ nid \rangle$ **have** $n'''' \leq n$ **using** *his-le* **by** *auto*

moreover **from** $\langle (n'''', nid''') \in his\ t\ n\ nid \rangle$

have $\exists nid'. (n'''', nid') \in his\ t\ n\ nid$ **by** *auto*

ultimately **have** $(\exists n' < n'''''. \exists nid'. (n',nid') \in his\ t\ n\ nid) \longrightarrow (\exists !nid'. (n'''',nid') \in his\ t\ n\ nid) \wedge (hisPred\ t\ n\ nid\ n''''', (SOME\ nid'. (hisPred\ t\ n\ nid\ n''''', nid') \in his\ t\ n\ nid)) = (SOME\ x.\ his\ prop\ t\ n\ nid\ n'''''\ (THE\ nid'. (n'''',nid') \in his\ t\ n\ nid)\ x)$ **using** *step.IH* **by** *auto*

with $\langle n'''' > n' \rangle \langle (n', nid') \in his\ t\ n\ nid \rangle$ **have** $\exists !nid'. (n'''',nid') \in his\ t\ n\ nid$ **and** $(hisPred\ t\ n\ nid\ n''''', (SOME\ nid'. (hisPred\ t\ n\ nid\ n''''', nid') \in his\ t\ n\ nid)) = (SOME\ x.\ his\ prop\ t\ n\ nid\ n'''''\ (THE\ nid'. (n'''',nid') \in his\ t\ n\ nid)\ x)$ **by** *auto*

moreover **from** $\langle \exists !nid'. (n'''',nid') \in his\ t\ n\ nid \rangle \langle (n'''', nid''') \in his\ t\ n\ nid \rangle$ **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** $(hisPred\ t\ n\ nid\ n''''', (SOME\ nid'. (hisPred\ t\ n\ nid\ n''''', nid') \in his\ t\ n\ nid)) = n'$

$nid)) = (SOME\ x.\ his\ prop\ t\ n\ nid\ n''''\ nid''''\ x)$ **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 $n'nid'' \langle \exists x.\ his\ prop\ t\ n\ nid\ n''''\ nid''''\ x \rangle$ **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 $\langle (n',\ nid') \in his\ t\ n\ nid \rangle$ **show** $?thesis$ **by auto**
qed
moreover have $\exists n' < n''''.\ \exists nid'.\ (n',nid') \in his\ t\ n\ nid$
proof –
from $n'nid' \langle \exists x.\ his\ prop\ t\ n\ nid\ n''''\ nid''''\ x \rangle$ **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 $\langle (n',\ nid') \in his\ t\ n\ nid \rangle$ **show** $?thesis$ **by auto**
qed
ultimately show $?thesis$
using $hisPrev-same\ \langle (n''',\ nid''') \in his\ t\ n\ nid \rangle\ \langle (n'''',\ nid''''') \in his\ t\ n\ nid \rangle$
by blast
qed
moreover have $nid''''=nid''''''$
proof –
from $n'nid'' \langle \exists x.\ his\ prop\ t\ n\ nid\ n''''\ nid''''\ x \rangle$
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 $\langle (n''',\ nid''') \in his\ t\ n\ nid \rangle$ **have** $n'''' \leq n$ **using** $his-le$ **by auto**
moreover from $\langle (n''',\ nid''') \in his\ t\ n\ nid \rangle$
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 $\langle (n''',\ nid''') \in his\ t\ n\ nid \rangle\ \langle (n''''',\ nid''''') \in his\ t\ n\ nid \rangle\ \langle n''''=n'''''' \rangle$
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 $\langle \exists n'' < n'. \exists nid'. (n'', nid') \in his\ t\ n\ nid \rangle$ **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** $asmp: (hisPred\ t\ n\ nid\ n', SOME\ nid'. (hisPred\ t\ n\ nid\ n', nid') \in his\ t\ n\ nid) = (SOME\ x. his-prop\ t\ n\ nid\ n''\ nid''\ x)$
and $(n'', nid'') \in his\ t\ n\ nid$ **and** $\exists x. his-prop\ t\ n\ nid\ n''\ nid''\ x$
moreover have $n''=n'$
proof $(rule\ antisym)$
show $n'' \geq n'$
proof $(rule\ ccontr)$
assume $(\neg n'' \geq n')$
hence $n'' < n'$ **by simp**
moreover have $n'' > hisPred\ t\ n\ nid\ n'$
proof –
let $?x = \lambda x. his-prop\ t\ n\ nid\ n''\ nid''\ x$
from $\langle \exists x. his-prop\ t\ n\ nid\ n''\ nid''\ x \rangle$ **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** $fst\ (hisPred\ t\ n\ nid\ n', SOME\ nid'. (hisPred\ t\ n\ nid\ n', nid') \in his\ t\ n\ nid) = fst\ (SOME\ x. ?x\ x)$ **by simp**
ultimately show $?thesis$ **by simp**
qed
moreover from $\langle \exists n'' < n'. \exists nid'. (n'', nid') \in his\ t\ n\ nid \rangle$
have $\neg(\exists x \in his\ t\ n\ nid. fst\ x < n' \wedge fst\ x > hisPred\ t\ n\ nid\ n')$
using $hisPrev-nex-less$ **by simp**
ultimately show $False$ **using** $\langle (n'', nid'') \in his\ t\ n\ nid \rangle$ **by auto**
qed
next
show $n' \geq n''$
proof $(rule\ ccontr)$
assume $(\neg n' \geq n'')$
hence $n' < n''$ **by simp**
moreover from $\langle (n'', nid'') \in his\ t\ n\ nid \rangle$ **have** $n'' \leq n$ **using** $his-le$ **by auto**
moreover from $\langle (n'', nid'') \in his\ t\ n\ nid \rangle$ **have** $\exists nid'. (n'', nid') \in his\ t\ n\ nid$ **by auto**
ultimately have $\langle \exists n' < n''. \exists nid'. (n', nid') \in his\ t\ n\ nid \rangle \longrightarrow \langle \exists !nid'. (n'', nid') \in his\ t\ n\ nid \rangle$
 $\wedge (hisPred\ t\ n\ nid\ n'', (SOME\ nid'. (hisPred\ t\ n\ nid\ n'', nid') \in his\ t\ n\ nid)) = (SOME\ x. his-prop\ t\ n\ nid\ n''\ (THE\ nid'. (n'', nid') \in his\ t\ n\ nid)\ x)$ **using** $step.IH$ **by auto**
with $\langle n' < n'' \rangle \langle (n', nid') \in his\ t\ n\ nid \rangle$ **have** $\exists !nid'. (n'', nid') \in his\ t\ n\ nid$ **and** $(hisPred\ t\ n\ nid\ n'', (SOME\ nid'. (hisPred\ t\ n\ nid\ n'', nid') \in his\ t\ n\ nid)) = (SOME\ x. his-prop\ t\ n\ nid\ n''\ (THE\ nid'. (n'', nid') \in his\ t\ n\ nid)\ x)$ **by auto**
moreover from $\langle \exists !nid'. (n'', nid') \in his\ t\ n\ nid \rangle \langle (n'', nid'') \in his\ t\ n\ nid \rangle$
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 $(hisPred\ t\ n\ nid\ n'', (SOME\ nid'. (hisPred\ t\ n\ nid\ n'', nid') \in his\ t\ n\ nid)) = (SOME\ x. his-prop\ t\ n\ nid\ n''\ nid''\ x)$ **by simp**
with $asmp$ **have** $(hisPred\ t\ n\ nid\ n', SOME\ nid'. (hisPred\ t\ n\ nid\ n', nid') \in his\ t\ n\ nid) = (hisPred\ t\ n\ nid\ n'', SOME\ nid'. (hisPred\ t\ n\ nid\ n'', nid') \in his\ t\ n\ nid)$ **by simp**
hence $hisPred\ t\ n\ nid\ n' = hisPred\ t\ n\ nid\ n''$ **by simp**
with $\langle \exists n'' < n'. \exists nid'. (n'', nid') \in his\ t\ n\ nid \rangle \langle n' < n'' \rangle \langle (n', nid') \in his\ t\ n\ nid \rangle \langle (n'', nid'') \in his\ t\ n\ nid \rangle \langle (n', nid') \in his\ t\ n\ nid \rangle$ **have** $n'=n''$ **using** $hisPrev-same$ **by blast**
with $\langle n' < n'' \rangle$ **show** $False$ **by simp**
qed
qed

ultimately have $(\text{hisPred } t \ n \ \text{nid} \ n', \text{SOME } \text{nid}'. (\text{hisPred } t \ n \ \text{nid} \ n', \text{nid}') \in \text{his } t \ n \ \text{nid}) =$
 $(\text{SOME } x. \text{his-prop } t \ n \ \text{nid} \ n' \ \text{nid}'' \ x)$ **by simp**
moreover from $\langle (n'', \text{nid}'') \in \text{his } t \ n \ \text{nid} \rangle \langle n''=n' \rangle$ **have** $(n', \text{nid}'') \in \text{his } t \ n \ \text{nid}$ **by simp**
with $\langle \exists ! \text{nid}'. (n', \text{nid}') \in \text{his } t \ n \ \text{nid} \rangle$ **have** $\text{nid}''=(\text{THE } \text{nid}'. (n', \text{nid}') \in \text{his } t \ n \ \text{nid})$
using *the1-equality*[of $\lambda \text{nid}'. (n', \text{nid}') \in \text{his } t \ n \ \text{nid}$] **by simp**
moreover from $\langle \exists x. \text{his-prop } t \ n \ \text{nid} \ n'' \ \text{nid}'' \ x \rangle \langle n''=n' \rangle \langle \text{nid}''=(\text{THE } \text{nid}'. (n', \text{nid}') \in \text{his } t \ n \ \text{nid}) \rangle$
 $\text{nid} \rangle$
have $\exists x. \text{his-prop } t \ n \ \text{nid} \ n' (\text{THE } \text{nid}'. (n', \text{nid}') \in \text{his } t \ n \ \text{nid}) \ x$ **by simp**
ultimately show *?thesis* **by simp**
qed
qed
ultimately show *?case* **by simp**
qed

corollary *his-determ-ex*:

assumes $(n', \text{nid}') \in \text{his } t \ n \ \text{nid}$
shows $\exists ! \text{nid}'. (n', \text{nid}') \in \text{his } t \ n \ \text{nid}$
using *assms his-le his-determ-ext*[of $n' \ n \ t \ \text{nid}$] **by force**

corollary *his-determ*:

assumes $(n', \text{nid}') \in \text{his } t \ n \ \text{nid}$
and $(n', \text{nid}'') \in \text{his } t \ n \ \text{nid}$
shows $\text{nid}' = \text{nid}''$ **using** *assms his-le his-determ-ext*[of $n' \ n \ t \ \text{nid}$] **by force**

corollary *his-determ-the*:

assumes $(n', \text{nid}') \in \text{his } t \ n \ \text{nid}$
shows $(\text{THE } \text{nid}'. (n', \text{nid}') \in \text{his } t \ n \ \text{nid}) = \text{nid}'$
using *assms his-determ theI'*[of $\lambda \text{nid}'. (n', \text{nid}') \in \text{his } t \ n \ \text{nid}$] *his-determ-ex* **by simp**

6.2.5 Blockchain Development

definition *devBC*:: $\text{trace} \Rightarrow \text{nat} \Rightarrow ' \text{nid} \Rightarrow \text{nat} \Rightarrow ' \text{nid}$ *option*

where *devBC* $t \ n \ \text{nid} \ n' \equiv$
 $(\text{if } (\exists \text{nid}'. (n', \text{nid}') \in \text{his } t \ n \ \text{nid}) \text{ then } (\text{Some } (\text{THE } \text{nid}'. (n', \text{nid}') \in \text{his } t \ n \ \text{nid}))$
 $\text{else } \text{Option.None})$

lemma *devBC-some*[*simp*]: **assumes** $\| \text{nid} \|_t \ n$ **shows** *devBC* $t \ n \ \text{nid} \ n = \text{Some } \text{nid}$

proof –

from *assms* **have** $(n, \text{nid}) \in \text{his } t \ n \ \text{nid}$ **using** *his.intros(1)* **by simp**
hence *devBC* $t \ n \ \text{nid} \ n = (\text{Some } (\text{THE } \text{nid}'. (n, \text{nid}') \in \text{his } t \ n \ \text{nid}))$ **using** *devBC-def* **by auto**
moreover have $(\text{THE } \text{nid}'. (n, \text{nid}') \in \text{his } t \ n \ \text{nid}) = \text{nid}$

proof

from $\langle (n, \text{nid}) \in \text{his } t \ n \ \text{nid} \rangle$ **show** $(n, \text{nid}) \in \text{his } t \ n \ \text{nid}$.

next

fix nid' **assume** $(n, \text{nid}') \in \text{his } t \ n \ \text{nid}$
thus $\text{nid}' = \text{nid}$ **using** *his-determ-base* **by simp**

qed

ultimately show *?thesis* **by simp**

qed

lemma *devBC-act*: **assumes** $\neg \text{Option.is-none } (\text{devBC } t \ n \ \text{nid} \ n')$ **shows** $\| \text{the } (\text{devBC } t \ n \ \text{nid} \ n') \|_t \ n'$

proof –

from *assms* **have** $\neg \text{devBC } t \ n \ \text{nid} \ n' = \text{Option.None}$ **by** (*metis is-none-simps(1)*)

then obtain nid' **where** $(n', \text{nid}') \in \text{his } t \ n \ \text{nid}$ **and** $\text{devBC } t \ n \ \text{nid} \ n' = (\text{Some } (\text{THE } \text{nid}'. (n', \text{nid}') \in \text{his } t \ n \ \text{nid}))$

using *devBC-def*[of $t \ n \ \text{nid}$] **by metis**

hence $nid' = (THE\ nid'. (n', nid') \in his\ t\ n\ nid)$ **using** *his-determ-the* **by** *simp*
with $\langle devBC\ t\ n\ nid\ n' = (Some\ (THE\ nid'. (n', nid') \in his\ t\ n\ nid)) \rangle$ **have** $(devBC\ t\ n\ nid\ n') =$
 nid' **by** *simp*
with $\langle (n', nid') \in his\ t\ n\ nid \rangle$ **show** *?thesis* **using** *his-act* **by** *simp*
qed

lemma *his-ex*:

assumes $\neg Option.is-none\ (devBC\ t\ n\ nid\ n')$

shows $\exists nid'. (n', nid') \in his\ t\ n\ nid$

proof (*rule ccontr*)

assume $\neg (\exists nid'. (n', nid') \in his\ t\ n\ nid)$

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 $\neg Option.is-none\ (devBC\ t\ n\ nid\ n')$

shows $n' \leq n$

proof –

from *assms* **have** $\exists nid'. (n', nid') \in his\ t\ n\ nid$ **using** *his-ex* **by** *simp*

then **obtain** nid' **where** $(n', nid') \in his\ t\ n\ nid$ **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* $\Rightarrow nat \Rightarrow 'nid \Rightarrow nat \Rightarrow nat \Rightarrow 'nid\ BC$

where $\llbracket \exists n' < n_s. \neg Option.is-none\ (devBC\ t\ n\ nid\ n'); Option.is-none\ (devBC\ t\ n\ nid\ n_s) \rrbracket \Longrightarrow devExt$
 $t\ n\ nid\ n_s\ 0 = bc\ (\sigma_{the}\ (devBC\ t\ n\ nid\ (GREATEST\ n'.\ n' < n_s \wedge \neg Option.is-none\ (devBC\ t\ n\ nid\ n'))))^t$
 $(GREATEST\ n'.\ n' < n_s \wedge \neg Option.is-none\ (devBC\ t\ n\ nid\ n'))$

$\mid \llbracket \neg (\exists n' < n_s. \neg Option.is-none\ (devBC\ t\ n\ nid\ n')); Option.is-none\ (devBC\ t\ n\ nid\ n_s) \rrbracket \Longrightarrow devExt$
 $t\ n\ nid\ n_s\ 0 = \llbracket$

$\neg Option.is-none\ (devBC\ t\ n\ nid\ n_s) \Longrightarrow devExt\ t\ n\ nid\ n_s\ 0 = bc\ (\sigma_{the}\ (devBC\ t\ n\ nid\ n_s))^t\ n_s$

$\mid \neg Option.is-none\ (devBC\ t\ n\ nid\ (n_s + Suc\ n')) \Longrightarrow devExt\ t\ n\ nid\ n_s\ (Suc\ n') = bc\ (\sigma_{the}\ (devBC\ t\ n\ nid\ (n_s + Suc\ n'))$
 $(n_s + Suc\ n'))$

$\mid Option.is-none\ (devBC\ t\ n\ nid\ (n_s + Suc\ n')) \Longrightarrow devExt\ t\ n\ nid\ n_s\ (Suc\ n') = devExt\ t\ n\ nid\ n_s$
 n'

proof –

show $\bigwedge n_s\ t\ n\ nid\ n_s'\ ta\ na\ nida.$

$\exists n' < n_s. \neg Option.is-none\ (devBC\ t\ n\ nid\ n') \Longrightarrow$

$Option.is-none\ (devBC\ t\ n\ nid\ n_s) \Longrightarrow$

$\exists n' < n_s'. \neg Option.is-none\ (devBC\ ta\ na\ nida\ n') \Longrightarrow$

$Option.is-none\ (devBC\ ta\ na\ nida\ n_s') \Longrightarrow$

$(t, n, nid, n_s, 0) = (ta, na, nida, n_s', 0) \Longrightarrow$

$bc\ (\sigma_{the}\ (devBC\ t\ n\ nid\ (GREATEST\ n'.\ n' < n_s \wedge \neg Option.is-none\ (devBC\ t\ n\ nid\ n'))))^t\ (GREATEST$
 $n'.\ n' < n_s \wedge \neg Option.is-none\ (devBC\ t\ n\ nid\ n')) =$

$bc\ (\sigma_{the}\ (devBC\ ta\ na\ nida\ (GREATEST\ n'.\ n' < n_s' \wedge \neg Option.is-none\ (devBC\ ta\ na\ nida\ n'))))^ta$
 $(GREATEST\ n'.\ n' < n_s' \wedge \neg Option.is-none\ (devBC\ ta\ na\ nida\ n'))$ **by** *auto*

show $\bigwedge n_s\ t\ n\ nid\ n_s'\ ta\ na\ nida.$

$\exists n' < n_s. \neg Option.is-none\ (devBC\ t\ n\ nid\ n') \Longrightarrow$

$Option.is-none\ (devBC\ t\ n\ nid\ n_s) \Longrightarrow$

$\neg (\exists n' < n_s'. \neg Option.is-none\ (devBC\ ta\ na\ nida\ n')) \Longrightarrow$

$Option.is-none\ (devBC\ ta\ na\ nida\ n_s') \Longrightarrow$

$(t, n, nid, n_s, 0) = (ta, na, nida, n_s', 0) \Longrightarrow$

$bc\ (\sigma_{the}\ (devBC\ t\ n\ nid\ (GREATEST\ n'.\ n' < n_s \wedge \neg Option.is-none\ (devBC\ t\ n\ nid\ n'))))^t\ (GREATEST$

$\neg \text{Option.is-none } (\text{devBC } ta \ na \ nida \ (n_s' + \text{Suc } n')) \implies$
 $(t, n, nid, n_s, 0) = (ta, na, nida, n_s', \text{Suc } n') \implies bc \ (\sigma_{the} (\text{devBC } t \ n \ nid \ n_s)^t \ n_s) = bc$
 $(\sigma_{the} (\text{devBC } ta \ na \ nida \ (n_s' + \text{Suc } n'))^{ta} \ (n_s' + \text{Suc } n')) \text{ by auto}$
show $\bigwedge t \ n \ nid \ n_s \ ta \ na \ nida \ n_s' \ n'.$
 $\neg \text{Option.is-none } (\text{devBC } t \ n \ nid \ n_s) \implies$
 $\text{Option.is-none } (\text{devBC } ta \ na \ nida \ (n_s' + \text{Suc } n')) \implies$
 $(t, n, nid, n_s, 0) = (ta, na, nida, n_s', \text{Suc } n') \implies bc \ (\sigma_{the} (\text{devBC } t \ n \ nid \ n_s)^t \ n_s) = \text{devExt-sumC}$
 $(ta, na, nida, n_s', n') \text{ by auto}$
show $\bigwedge t \ n \ nid \ n_s \ n' \ ta \ na \ nida \ n_s' \ n'a.$
 $\neg \text{Option.is-none } (\text{devBC } t \ n \ nid \ (n_s + \text{Suc } n')) \implies$
 $\neg \text{Option.is-none } (\text{devBC } ta \ na \ nida \ (n_s' + \text{Suc } n'a)) \implies$
 $(t, n, nid, n_s, \text{Suc } n') = (ta, na, nida, n_s', \text{Suc } n'a) \implies$
 $bc \ (\sigma_{the} (\text{devBC } t \ n \ nid \ (n_s + \text{Suc } n'))^t \ (n_s + \text{Suc } n')) = bc \ (\sigma_{the} (\text{devBC } ta \ na \ nida \ (n_s' + \text{Suc } n'a))^{ta}$
 $(n_s' + \text{Suc } n'a)) \text{ by auto}$
show $\bigwedge t \ n \ nid \ n_s \ n' \ ta \ na \ nida \ n_s' \ n'a.$
 $\neg \text{Option.is-none } (\text{devBC } t \ n \ nid \ (n_s + \text{Suc } n')) \implies$
 $\text{Option.is-none } (\text{devBC } ta \ na \ nida \ (n_s' + \text{Suc } n'a)) \implies$
 $(t, n, nid, n_s, \text{Suc } n') = (ta, na, nida, n_s', \text{Suc } n'a) \implies bc \ (\sigma_{the} (\text{devBC } t \ n \ nid \ (n_s + \text{Suc } n'))^t$
 $(n_s + \text{Suc } n')) = \text{devExt-sumC } (ta, na, nida, n_s', n'a) \text{ by auto}$
show $\bigwedge t \ n \ nid \ n_s \ n' \ ta \ na \ nida \ n_s' \ n'a.$
 $\text{Option.is-none } (\text{devBC } t \ n \ nid \ (n_s + \text{Suc } n')) \implies$
 $\text{Option.is-none } (\text{devBC } ta \ na \ nida \ (n_s' + \text{Suc } n'a)) \implies$
 $(t, n, nid, n_s, \text{Suc } n') = (ta, na, nida, n_s', \text{Suc } n'a) \implies \text{devExt-sumC } (t, n, nid, n_s, n') =$
 $\text{devExt-sumC } (ta, na, nida, n_s', n'a) \text{ by auto}$
show $\bigwedge P \ x. (\bigwedge n_s \ t \ n \ nid. \exists n' < n_s. \neg \text{Option.is-none } (\text{devBC } t \ n \ nid \ n') \implies \text{Option.is-none } (\text{devBC}$
 $t \ n \ nid \ n_s) \implies x = (t, n, nid, n_s, 0) \implies P) \implies$
 $(\bigwedge n_s \ t \ n \ nid. \neg (\exists n' < n_s. \neg \text{Option.is-none } (\text{devBC } t \ n \ nid \ n')) \implies \text{Option.is-none } (\text{devBC } t$
 $n \ nid \ n_s) \implies x = (t, n, nid, n_s, 0) \implies P) \implies$
 $(\bigwedge t \ n \ nid \ n_s. \neg \text{Option.is-none } (\text{devBC } t \ n \ nid \ n_s) \implies x = (t, n, nid, n_s, 0) \implies P) \implies$
 $(\bigwedge t \ n \ nid \ n_s \ n'. \neg \text{Option.is-none } (\text{devBC } t \ n \ nid \ (n_s + \text{Suc } n')) \implies x = (t, n, nid, n_s, \text{Suc}$
 $n') \implies P) \implies$
 $(\bigwedge t \ n \ nid \ n_s \ n'. \text{Option.is-none } (\text{devBC } t \ n \ nid \ (n_s + \text{Suc } n')) \implies x = (t, n, nid, n_s, \text{Suc}$
 $n') \implies P) \implies P$
proof –
fix $P::\text{bool}$ **and** $x::\text{trace} \times \text{nat} \times \text{nat} \times \text{nat} \times \text{nat}$
assume $a1:(\bigwedge n_s \ t \ n \ nid. \exists n' < n_s. \neg \text{Option.is-none } (\text{devBC } t \ n \ nid \ n') \implies \text{Option.is-none } (\text{devBC}$
 $t \ n \ nid \ n_s) \implies x = (t, n, nid, n_s, 0) \implies P) \text{ and}$
 $a2:(\bigwedge n_s \ t \ n \ nid. \neg (\exists n' < n_s. \neg \text{Option.is-none } (\text{devBC } t \ n \ nid \ n')) \implies \text{Option.is-none } (\text{devBC}$
 $t \ n \ nid \ n_s) \implies x = (t, n, nid, n_s, 0) \implies P) \text{ and}$
 $a3:(\bigwedge t \ n \ nid \ n_s. \neg \text{Option.is-none } (\text{devBC } t \ n \ nid \ n_s) \implies x = (t, n, nid, n_s, 0) \implies P) \text{ and}$
 $a4:(\bigwedge t \ n \ nid \ n_s \ n'. \neg \text{Option.is-none } (\text{devBC } t \ n \ nid \ (n_s + \text{Suc } n')) \implies x = (t, n, nid, n_s,$
 $\text{Suc } n') \implies P) \text{ and}$
 $a5:(\bigwedge t \ n \ nid \ n_s \ n'. \text{Option.is-none } (\text{devBC } t \ n \ nid \ (n_s + \text{Suc } n')) \implies x = (t, n, nid, n_s, \text{Suc}$
 $n') \implies P)$
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 $\text{Option.is-none } (\text{devBC } t \ n \ nid \ n_s)$
thus *?thesis*

proof cases
assume $\exists n' < n_s. \neg \text{Option.is-none } (\text{devBC } t \ n \ \text{nid } n')$
with $\langle x = (t, n, \text{nid}, n_s, n') \rangle \langle \text{Option.is-none } (\text{devBC } t \ n \ \text{nid } n_s) \rangle \langle n' = 0 \rangle$ **show** *?thesis* **using**
a1 **by simp**
next
assume $\neg (\exists n' < n_s. \neg \text{Option.is-none } (\text{devBC } t \ n \ \text{nid } n'))$
with $\langle x = (t, n, \text{nid}, n_s, n') \rangle \langle \text{Option.is-none } (\text{devBC } t \ n \ \text{nid } n_s) \rangle \langle n' = 0 \rangle$ **show** *?thesis* **using**
a2 **by simp**
qed
next
assume $\neg \text{Option.is-none } (\text{devBC } t \ n \ \text{nid } n_s)$
with $\langle x = (t, n, \text{nid}, n_s, n') \rangle \langle n' = 0 \rangle$ **show** *?thesis* **using** a3 **by simp**
qed
next
case (*Suc* n'')
then show *?thesis*
proof cases
assume $\text{Option.is-none } (\text{devBC } t \ n \ \text{nid } (n_s + \text{Suc } n''))$
with $\langle x = (t, n, \text{nid}, n_s, n') \rangle \langle n' = \text{Suc } n'' \rangle$ **show** *?thesis* **using** a5 [*of* $t \ n \ \text{nid } n_s \ n''$] **by simp**
next
assume $\neg \text{Option.is-none } (\text{devBC } t \ n \ \text{nid } (n_s + \text{Suc } n''))$
with $\langle x = (t, n, \text{nid}, n_s, n') \rangle \langle n' = \text{Suc } n'' \rangle$ **show** *?thesis* **using** a4 [*of* $t \ n \ \text{nid } n_s \ n''$] **by simp**
qed
qed
qed
qed
qed
termination by lexicographic-order

lemma devExt-same:

assumes $\forall n''' > n'. n''' \leq n'' \longrightarrow \text{Option.is-none } (\text{devBC } t \ n \ \text{nid } n''')$
and $n' \geq n_s$
and $n''' \leq n''$
shows $n''' \geq n' \implies \text{devExt } t \ n \ \text{nid } n_s \ (n''' - n_s) = \text{devExt } t \ n \ \text{nid } n_s \ (n' - n_s)$
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* $n'''' \leq n''$ **by simp**
ultimately have $\text{Option.is-none } (\text{devBC } t \ n \ \text{nid } (\text{Suc } n'''))$ **using** *assms*(1) **by simp**
moreover from *assms*(2) *step.hyps* **have** $n'''' \geq n_s$ **by simp**
hence $\text{Suc } n'''' = n_s + \text{Suc } (n'''' - n_s)$ **by simp**
ultimately have $\text{Option.is-none } (\text{devBC } t \ n \ \text{nid } (n_s + \text{Suc } (n'''' - n_s)))$ **by metis**
hence $\text{devExt } t \ n \ \text{nid } n_s \ (\text{Suc } (n'''' - n_s)) = \text{devExt } t \ n \ \text{nid } n_s \ (n'''' - n_s)$ **by simp**
moreover from $\langle n'''' \geq n_s \rangle$ **have** $\text{Suc } (n'''' - n_s) = \text{Suc } n'''' - n_s$ **by simp**
ultimately have $\text{devExt } t \ n \ \text{nid } n_s \ (\text{Suc } n'''' - n_s) = \text{devExt } t \ n \ \text{nid } n_s \ (n'''' - n_s)$ **by simp**
with *step.IH* **show** *?case* **by simp**
qed

lemma devExt-bc[simp]:

assumes $\neg \text{Option.is-none } (\text{devBC } t \ n \ \text{nid } (n' + n''))$
shows $\text{devExt } t \ n \ \text{nid } n' \ n'' = \text{bc } (\sigma_{\text{the}} (\text{devBC } t \ n \ \text{nid } (n' + n'')))(t \ (n' + n''))$
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 (σ_{the} (devBC t n nid (GREATEST n'''. $n''' < (n' + n'')$ \wedge \neg Option.is-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 $?G = \text{GREATEST } n'''$. $?P n'''$

have $\forall n''' > n' + n''$. $\neg ?P n'''$ by simp

with $\langle \exists n''' < n' + n''$. \neg Option.is-none (devBC t n nid n''') \rangle have $\exists n'''$. $?P n''' \wedge (\forall n''''$. $?P n'''' \rightarrow n'''' \leq n''')$ using boundedGreatest[of ?P] by blast

hence $?P ?G$ using GreatestI-ex-nat[of ?P] by auto

hence \neg 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 $\langle \neg$ Option.is-none (devBC t n nid ?G) \rangle have \neg Option.is-none (devBC t n nid (?G - n' + n'))

by simp

moreover from $\langle ?G > n' \rangle$ have $?G - n' \neq 0$ by auto

hence $\exists \text{nat. Suc nat} = ?G - n'$ by presburger

then obtain nat where $\text{Suc nat} = ?G - n'$ by auto

ultimately have \neg Option.is-none (devBC t n nid (n' + Suc nat)) by simp

hence devExt t n nid n' (Suc nat) = bc (σ_{the} (devBC t n nid (n' + Suc nat))^t (n' + Suc nat)) by

simp

with $\langle \text{Suc nat} = ?G - n' \rangle$ have devExt t n nid n' (?G - n') = bc (σ_{the} (devBC t n nid (?G - n' + n'))^t (?G - n' + n')) by simp

with $\langle ?G - n' + n' = ?G \rangle$ have devExt t n nid n' (?G - n') = bc (σ_{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 $\langle \exists n'''$. $?P n''' \wedge (\forall n''''$. $?P n'''' \rightarrow n'''' \leq n''')$ \rangle have $\forall n'''$. $?P n''' \rightarrow n''' \leq ?G$

using Greatest-le-nat[of ?P] by blast

hence $\forall n''' > ?G$. $n''' < n' + n'' \rightarrow$ Option.is-none (devBC t n nid n''') by auto

with \langle Option.is-none (devBC t n nid (n' + n'')) \rangle

have $\forall n''' > ?G$. $n''' \leq n' + n'' \rightarrow$ Option.is-none (devBC t n nid n''') by auto

moreover from $\langle ?P ?G \rangle$ have $?G \leq n' + n''$ by simp

moreover from $\langle ?G > n' \rangle$ have $?G \geq n'$ by simp

ultimately show ?thesis using $\langle ?G > n' \rangle$ devExt-same[of ?G n' + n'' t n nid n' n' + n''] by blast

qed

ultimately show ?thesis by simp

next

assume $\neg ?G > n'$

thus ?thesis

proof cases

assume $?G = n'$

with $\langle \neg$ Option.is-none (devBC t n nid ?G) \rangle have \neg Option.is-none (devBC t n nid n') by simp

with $\langle \neg$ Option.is-none (devBC t n nid ?G) \rangle have devExt t n nid n' 0 = bc (σ_{the} (devBC t n nid n')^t)

$n')$) by *simp*
moreover have $\text{devExt } t \ n \ \text{nid } n' \ n'' = \text{devExt } t \ n \ \text{nid } n' \ 0$
proof –
from $\langle \exists n'''. ?P \ n''' \wedge (\forall n'''' . ?P \ n'''' \longrightarrow n'''' \leq n''') \rangle$ **have** $\forall n'''' > ?G . ?P \ n'''' \longrightarrow n'''' \leq ?G$
using *Greatest-le-nat*[of $?P$] **by** *blast*
with $\langle ?G = n' \rangle$ **have** $\forall n'''' > n' . n'''' < n' + n'' \longrightarrow \text{Option.is-none } (\text{devBC } t \ n \ \text{nid } n''')$ **by** *simp*
with $\langle \text{Option.is-none } (\text{devBC } t \ n \ \text{nid } (n' + n'')) \rangle$
have $\forall n'''' > n' . n'''' \leq n' + n'' \longrightarrow \text{Option.is-none } (\text{devBC } t \ n \ \text{nid } n''')$ **by** *auto*
moreover from $\langle \neg n'' = 0 \rangle$ **have** $n' < n' + n''$ **by** *simp*
ultimately show $?thesis$ **using** *devExt-same*[of $n' \ n' + n'' \ t \ n \ \text{nid } n' \ n' + n''$] **by** *simp*
qed
ultimately show $?thesis$ **using** $\langle ?G = n' \rangle$ **by** *simp*
next
assume $\neg ?G = n'$
with $\langle \neg ?G > n' \rangle$ **have** $?G < n'$ **by** *simp*
hence $\text{devExt } t \ n \ \text{nid } n' \ n'' = \text{devExt } t \ n \ \text{nid } n' \ 0$
proof –
from $\langle \exists n'''. ?P \ n''' \wedge (\forall n'''' . ?P \ n'''' \longrightarrow n'''' \leq n''') \rangle$ **have** $\forall n'''' > ?G . ?P \ n'''' \longrightarrow n'''' \leq ?G$
using *Greatest-le-nat*[of $?P$] **by** *blast*
with $\langle \neg ?G > n' \rangle$ **have** $\forall n'''' > n' . n'''' < n' + n'' \longrightarrow \text{Option.is-none } (\text{devBC } t \ n \ \text{nid } n''')$ **by** *auto*
with $\langle \text{Option.is-none } (\text{devBC } t \ n \ \text{nid } (n' + n'')) \rangle$
have $\forall n'''' > n' . n'''' \leq n' + n'' \longrightarrow \text{Option.is-none } (\text{devBC } t \ n \ \text{nid } n''')$ **by** *auto*
moreover from $\langle ?P \ ?G \rangle$ **have** $?G < n' + n''$ **by** *simp*
moreover from $\langle \neg n'' = 0 \rangle$ **have** $n' < n' + n''$ **by** *simp*
ultimately show $?thesis$ **using** *devExt-same*[of $n' \ n' + n'' \ t \ n \ \text{nid } n' \ n' + n''$] **by** *simp*
qed
moreover have $\text{devExt } t \ n \ \text{nid } n' \ 0 = bc \ (\sigma_{the} \ (\text{devBC } t \ n \ \text{nid } (\text{GREATEST } n'''. n'''' < n' \wedge \neg \text{Option.is-none } (\text{devBC } t \ n \ \text{nid } n'''')) \ (\text{GREATEST } n'''. n'''' < n' \wedge \neg \text{Option.is-none } (\text{devBC } t \ n \ \text{nid } n''''))))$
proof –
from $\langle \neg n'' = 0 \rangle$ **have** $n' < n' + n''$ **by** *simp*
moreover from $\langle \exists n'''. ?P \ n''' \wedge (\forall n'''' . ?P \ n'''' \longrightarrow n'''' \leq n''') \rangle$ **have** $\forall n'''' > ?G . ?P \ n'''' \longrightarrow n'''' \leq ?G$ **using** *Greatest-le-nat*[of $?P$] **by** *blast*
ultimately have $\text{Option.is-none } (\text{devBC } t \ n \ \text{nid } n')$ **using** $\langle ?G < n' \rangle$ **by** *simp*
moreover from $\langle \forall n'''' > ?G . ?P \ n'''' \longrightarrow n'''' \leq ?G \rangle$ $\langle ?G < n' \rangle$ $\langle n' < n' + n'' \rangle$ **have** $\forall n'''' \geq n' . n'''' < n' + n'' \longrightarrow \text{Option.is-none } (\text{devBC } t \ n \ \text{nid } n''')$ **by** *auto*
have $\exists n'''' < n' . \neg \text{Option.is-none } (\text{devBC } t \ n \ \text{nid } n''')$
proof –
from $\langle \exists n'''' < n' + n'' . \neg \text{Option.is-none } (\text{devBC } t \ n \ \text{nid } n''') \rangle$ **obtain** n''''
where $n'''' < n' + n''$ **and** $\neg \text{Option.is-none } (\text{devBC } t \ n \ \text{nid } n''')$ **by** *auto*
moreover have $n'''' < n'$
proof (*rule ccontr*)
assume $\neg n'''' < n'$
hence $n'''' \geq n'$ **by** *simp*
with $\langle \forall n'''' \geq n' . n'''' < n' + n'' \longrightarrow \text{Option.is-none } (\text{devBC } t \ n \ \text{nid } n''') \rangle$ $\langle n'''' < n' + n'' \rangle$
 $\langle \neg \text{Option.is-none } (\text{devBC } t \ n \ \text{nid } n''') \rangle$ **show** *False* **by** *simp*
qed
ultimately show $?thesis$ **by** *auto*
qed
ultimately show $?thesis$ **by** *simp*
qed
moreover have $(\text{GREATEST } n'''. n'''' < n' \wedge \neg \text{Option.is-none } (\text{devBC } t \ n \ \text{nid } n''')) = ?G$
proof(*rule Greatest-equality*)
from $\langle ?P \ ?G \rangle$ **have** $?G < n' + n''$ **and** $\neg \text{Option.is-none } (\text{devBC } t \ n \ \text{nid } ?G)$ **by** *auto*
with $\langle ?G < n' \rangle$ **show** $?G < n' \wedge \neg \text{Option.is-none } (\text{devBC } t \ n \ \text{nid } ?G)$ **by** *simp*
next

fix y **assume** $y < n' \wedge \neg \text{Option.is-none} (\text{devBC } t \ n \ \text{nid } y)$
moreover from $\langle \exists n'''. ?P \ n''' \wedge (\forall n'''' . ?P \ n'''' \longrightarrow n'''' \leq n''') \rangle$
have $\forall n'''. ?P \ n''' \longrightarrow n''' \leq ?G$ **using** *Greatest-le-nat*[of $?P$] **by** *blast*
ultimately show $y \leq ?G$ **by** *simp*
qed
ultimately show $?thesis$ **by** *simp*
qed
qed
qed

lemma *devExt-shift*: $\text{devExt } t \ n \ \text{nid } (n'+n'') \ 0 = \text{devExt } t \ n \ \text{nid } n' \ n''$

proof (*cases*)

assume $n''=0$

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

next

assume $\neg (n''=0)$

thus $?thesis$

proof (*cases*)

assume $\text{Option.is-none} (\text{devBC } t \ n \ \text{nid } (n'+n''))$

thus $?thesis$

proof *cases*

assume $\exists n''' < n'+n'' . \neg \text{Option.is-none} (\text{devBC } t \ n \ \text{nid } n''')$

with $\langle \text{Option.is-none} (\text{devBC } t \ n \ \text{nid } (n'+n'')) \rangle$ **have** $\text{devExt } t \ n \ \text{nid } (n'+n'') \ 0 = \text{bc} (\sigma_{\text{the}} (\text{devBC } t \ n \ \text{nid } (\text{GREATEST } n'''. n''' < (n'+n''))))$ **by** *simp*

moreover from $\langle \neg (n''=0) \rangle$ $\langle \text{Option.is-none} (\text{devBC } t \ n \ \text{nid } (n'+n'')) \rangle$ $\langle \exists n''' < n'+n'' . \neg \text{Option.is-none} (\text{devBC } t \ n \ \text{nid } n''') \rangle$ **have** $\text{devExt } t \ n \ \text{nid } n' \ n'' = \text{bc} (\sigma_{\text{the}} (\text{devBC } t \ n \ \text{nid } (\text{GREATEST } n'''. n''' < (n'+n'')) \wedge \neg (\text{GREATEST } n'''. n''' < (n'+n'')) \wedge \neg \text{Option.is-none} (\text{devBC } t \ n \ \text{nid } n'''))))$ **using** *devExt-greatest* **by** *simp*

ultimately show $?thesis$ **by** *simp*

next

assume $\neg (\exists n''' < n'+n'' . \neg \text{Option.is-none} (\text{devBC } t \ n \ \text{nid } n'''))$

with $\langle \text{Option.is-none} (\text{devBC } t \ n \ \text{nid } (n'+n'')) \rangle$ **have** $\text{devExt } t \ n \ \text{nid } (n'+n'') \ 0 = []$ **by** *simp*

moreover have $\text{devExt } t \ n \ \text{nid } n' \ n'' = []$

proof $-$

from $\langle \neg (\exists n''' < n'+n'' . \neg \text{Option.is-none} (\text{devBC } t \ n \ \text{nid } n''')) \rangle$ $\langle n'' \neq 0 \rangle$

have $\text{Option.is-none} (\text{devBC } t \ n \ \text{nid } n')$ **by** *simp*

moreover from $\langle \neg (\exists n''' < n'+n'' . \neg \text{Option.is-none} (\text{devBC } t \ n \ \text{nid } n''')) \rangle$

have $\neg (\exists n''' < n' . \neg \text{Option.is-none} (\text{devBC } t \ n \ \text{nid } n'''))$ **by** *simp*

ultimately have $\text{devExt } t \ n \ \text{nid } n' \ 0 = []$ **by** *simp*

moreover have $\text{devExt } t \ n \ \text{nid } n' \ n'' = \text{devExt } t \ n \ \text{nid } n' \ 0$

proof $-$

from $\langle \neg (\exists n''' < n'+n'' . \neg \text{Option.is-none} (\text{devBC } t \ n \ \text{nid } n''')) \rangle$

have $\forall n''' > n' . n''' < n' + n'' \longrightarrow \text{Option.is-none} (\text{devBC } t \ n \ \text{nid } n''')$ **by** *simp*

with $\langle \text{Option.is-none} (\text{devBC } t \ n \ \text{nid } (n'+n'')) \rangle$

have $\forall n''' > n' . n''' \leq n'+n'' \longrightarrow \text{Option.is-none} (\text{devBC } t \ n \ \text{nid } n''')$ **by** *auto*

moreover from $\langle \neg (n''=0) \rangle$ **have** $n' < n'+n''$ **by** *simp*

ultimately show $?thesis$ **using** *devExt-same*[of $n' \ n'+n'' \ t \ n \ \text{nid } n' \ n'+n''$] **by** *simp*

qed

ultimately show $?thesis$ **by** *simp*

qed

ultimately show $?thesis$ **by** *simp*

qed

next

assume $\neg \text{Option.is-none} (\text{devBC } t \ n \ \text{nid } (n'+n''))$

hence $\text{devExt } t \ n \ \text{nid } (n'+n'') \ 0 = \text{bc} (\sigma_{\text{the}} (\text{devBC } t \ n \ \text{nid } (n'+n'')) (t \ (n'+n'')))$ **by** *simp*

moreover from $\langle \neg \text{Option.is-none } (\text{devBC } t \ n \ \text{nid } (n'+n')) \rangle$
 have $\text{devExt } t \ n \ \text{nid } n' \ n'' = \text{bc } (\sigma_{\text{the}} (\text{devBC } t \ n \ \text{nid } (n'+n')) (t \ (n'+n''))) \text{ by simp}$
 ultimately show $?thesis$ by simp

qed

qed

lemma *devExt-bc-geq*:

assumes $\neg \text{Option.is-none } (\text{devBC } t \ n \ \text{nid } n')$ and $n' \geq n_s$
 shows $\text{devExt } t \ n \ \text{nid } n_s \ (n'-n_s) = \text{bc } (\sigma_{\text{the}} (\text{devBC } t \ n \ \text{nid } n') (t \ n')) \text{ (is ?LHS = ?RHS)}$

proof –

have $\text{devExt } t \ n \ \text{nid } n_s \ (n'-n_s) = \text{devExt } t \ n \ \text{nid } (n_s+(n'-n_s)) \ 0 \text{ using devExt-shift by auto}$
 moreover from *assms(2)* have $n_s+(n'-n_s) = n' \text{ by simp}$
 ultimately have $\text{devExt } t \ n \ \text{nid } n_s \ (n'-n_s) = \text{devExt } t \ n \ \text{nid } n' \ 0 \text{ by simp}$
 with *assms(1)* show $?thesis$ by simp

qed

lemma *his-bc-empty*:

assumes $(n', \text{nid}') \in \text{his } t \ n \ \text{nid}$ and $\neg (\exists n'' < n'. \exists \text{nid}'' . (n'', \text{nid}'') \in \text{his } t \ n \ \text{nid})$
 shows $\text{bc } (\sigma_{\text{nid}'} (t \ n')) = []$

proof –

have $\neg (\exists x. \text{his-prop } t \ n \ \text{nid } n' \ \text{nid}' \ x)$

proof (rule ccontr)

assume $\neg \neg (\exists x. \text{his-prop } t \ n \ \text{nid } n' \ \text{nid}' \ x)$

hence $\exists x. \text{his-prop } t \ n \ \text{nid } n' \ \text{nid}' \ x \text{ by simp}$

with $\langle (n', \text{nid}') \in \text{his } t \ n \ \text{nid} \rangle$ have $(\text{SOME } x. \text{his-prop } t \ n \ \text{nid } n' \ \text{nid}' \ x) \in \text{his } t \ n \ \text{nid}$
 using *his.intros* by simp

moreover from $\langle \exists x. \text{his-prop } t \ n \ \text{nid } n' \ \text{nid}' \ x \rangle$ have $\text{his-prop } t \ n \ \text{nid } n' \ \text{nid}' \ (\text{SOME } x. \text{his-prop } t \ n \ \text{nid } n' \ \text{nid}' \ x)$

using *someI-ex*[of $\lambda x. \text{his-prop } t \ n \ \text{nid } n' \ \text{nid}' \ x$] by auto

hence $(\exists n. \text{latestAct-cond } \text{nid}' \ t \ n' \ n) \wedge \text{fst } (\text{SOME } x. \text{his-prop } t \ n \ \text{nid } n' \ \text{nid}' \ x) = \langle \text{nid}' \leftarrow t \rangle_{n'}$
 by force

hence $\text{fst } (\text{SOME } x. \text{his-prop } t \ n \ \text{nid } n' \ \text{nid}' \ x) < n' \text{ using latestAct-prop(2)[of } n' \ \text{nid}' \ t] \text{ by force}$
 ultimately have $\text{fst } (\text{SOME } x. \text{his-prop } t \ n \ \text{nid } n' \ \text{nid}' \ x) < n' \wedge$

$(\text{fst } (\text{SOME } x. \text{his-prop } t \ n \ \text{nid } n' \ \text{nid}' \ x), \text{snd } (\text{SOME } x. \text{his-prop } t \ n \ \text{nid } n' \ \text{nid}' \ x)) \in \text{his } t \ n \ \text{nid}$

by *simp*

thus *False* using *assms(2)* by blast

qed

hence $\forall x. \neg (\exists n. \text{latestAct-cond } \text{nid}' \ t \ n' \ n) \vee \neg \|\text{snd } x\|_t (\text{fst } x) \vee \neg \text{fst } x = \langle \text{nid}' \leftarrow t \rangle_{n'} \vee \neg$
 $(\text{prefix } (\text{bc } (\sigma_{\text{nid}'} (t \ n')))) (\text{bc } (\sigma_{\text{snd } x} (t \ (\text{fst } x)))) \vee (\exists b. \text{bc } (\sigma_{\text{nid}'} (t \ n')) = (\text{bc } (\sigma_{\text{snd } x} (t \ (\text{fst } x)))) @ [b]$
 $\wedge \text{mining } (\sigma_{\text{nid}'} (t \ n')))) \text{ by auto}$

hence $\neg (\exists n. \text{latestAct-cond } \text{nid}' \ t \ n' \ n) \vee (\exists n. \text{latestAct-cond } \text{nid}' \ t \ n' \ n) \wedge (\forall x. \neg \|\text{snd } x\|_t (\text{fst } x)$
 $\vee \neg \text{fst } x = \langle \text{nid}' \leftarrow t \rangle_{n'} \vee \neg (\text{prefix } (\text{bc } (\sigma_{\text{nid}'} (t \ n')))) (\text{bc } (\sigma_{\text{snd } x} (t \ (\text{fst } x)))) \vee (\exists b. \text{bc } (\sigma_{\text{nid}'} (t \ n'))$
 $= (\text{bc } (\sigma_{\text{snd } x} (t \ (\text{fst } x)))) @ [b] \wedge \text{mining } (\sigma_{\text{nid}'} (t \ n')))) \text{ by auto}$

thus $?thesis$

proof

assume $\neg (\exists n. \text{latestAct-cond } \text{nid}' \ t \ n' \ n)$

moreover from *assms(1)* have $\|\text{nid}'\|_t n' \text{ using his-act by simp}$

ultimately show $?thesis$ using *init-model* by simp

next

assume $(\exists n. \text{latestAct-cond } \text{nid}' \ t \ n' \ n) \wedge (\forall x. \neg \|\text{snd } x\|_t (\text{fst } x) \vee \neg \text{fst } x = \langle \text{nid}' \leftarrow t \rangle_{n'} \vee \neg$
 $(\text{prefix } (\text{bc } (\sigma_{\text{nid}'} (t \ n')))) (\text{bc } (\sigma_{\text{snd } x} (t \ (\text{fst } x)))) \vee (\exists b. \text{bc } (\sigma_{\text{nid}'} (t \ n')) = (\text{bc } (\sigma_{\text{snd } x} (t \ (\text{fst } x)))) @ [b]$
 $\wedge \text{mining } (\sigma_{\text{nid}'} (t \ n'))))$

hence $\exists n. \text{latestAct-cond } \text{nid}' \ t \ n' \ n$ and $\forall x. \neg \|\text{snd } x\|_t (\text{fst } x) \vee \neg \text{fst } x = \langle \text{nid}' \leftarrow t \rangle_{n'} \vee \neg$
 $(\text{prefix } (\text{bc } (\sigma_{\text{nid}'} (t \ n')))) (\text{bc } (\sigma_{\text{snd } x} (t \ (\text{fst } x)))) \vee (\exists b. \text{bc } (\sigma_{\text{nid}'} (t \ n')) = (\text{bc } (\sigma_{\text{snd } x} (t \ (\text{fst } x)))) @ [b]$

\wedge mining $(\sigma_{nid'}(t n'))$ **by auto**
hence *asmp*: $\forall x. \|\text{snd } x\|_t (fst x) \longrightarrow fst x = \langle nid' \leftarrow t \rangle_{n'} \longrightarrow \neg (\text{prefix } (bc (\sigma_{nid'}(t n'))) (bc (\sigma_{snd x}(t (fst x)))) \vee (\exists b. bc (\sigma_{nid'}(t n')) = (bc (\sigma_{snd x}(t (fst x)))) @ [b] \wedge \text{mining } (\sigma_{nid'}(t n')))$ **by auto**
show *?thesis*
proof cases
assume *trusted nid'*
moreover from *assms(1)* **have** $\|\text{nid}'\|_t n'$ **using** *his-act by simp*
ultimately obtain *nid''* **where** $\|\text{nid}''\|_t \langle nid' \leftarrow t \rangle_{n'}$ **and** $\text{mining } (\sigma_{nid'} t n') \wedge (\exists b. bc (\sigma_{nid'} t n') = bc (\sigma_{nid''} t \langle nid' \leftarrow t \rangle_{n'}) @ [b]) \vee \neg \text{mining } (\sigma_{nid'} t n') \wedge bc (\sigma_{nid'} t n') = bc (\sigma_{nid''} t \langle nid' \leftarrow t \rangle_{n'})$ **using** $\langle \exists n. \text{latestAct-cond } nid' t n' n \rangle \text{bhv-tr-context[of } nid' t n']$ **by auto**
moreover from $\langle \|\text{nid}''\|_t \langle nid' \leftarrow t \rangle_{n'} \rangle$ **have** $\neg (\text{prefix } (bc (\sigma_{nid'}(t n'))) (bc (\sigma_{nid''}(t (\langle nid' \leftarrow t \rangle_{n'})))) \vee (\exists b. bc (\sigma_{nid'}(t n')) = (bc (\sigma_{nid''}(t (\langle nid' \leftarrow t \rangle_{n'})))) @ [b] \wedge \text{mining } (\sigma_{nid'}(t n')))$ **using** *asmp by auto*
ultimately have *False* **by auto**
thus *?thesis ..*
next
assume $\neg \text{trusted } nid'$
moreover from *assms(1)* **have** $\|\text{nid}'\|_t n'$ **using** *his-act by simp*
ultimately obtain *nid''* **where** $\|\text{nid}''\|_t \langle nid' \leftarrow t \rangle_{n'}$ **and** $(\text{mining } (\sigma_{nid'} t n') \wedge (\exists b. \text{prefix } (bc (\sigma_{nid'} t n')) (bc (\sigma_{nid''} t \langle nid' \leftarrow t \rangle_{n'})) @ [b])) \vee \neg \text{mining } (\sigma_{nid'} t n') \wedge \text{prefix } (bc (\sigma_{nid'} t n')) (bc (\sigma_{nid''} t \langle nid' \leftarrow t \rangle_{n'}))$ **using** $\langle \exists n. \text{latestAct-cond } nid' t n' n \rangle \text{bhv-ut-context[of } nid' t n']$ **by auto**
moreover from $\langle \|\text{nid}''\|_t \langle nid' \leftarrow t \rangle_{n'} \rangle$ **have** $\neg (\text{prefix } (bc (\sigma_{nid'}(t n'))) (bc (\sigma_{nid''}(t (\langle nid' \leftarrow t \rangle_{n'})))) \vee (\exists b. bc (\sigma_{nid'}(t n')) = (bc (\sigma_{nid''}(t (\langle nid' \leftarrow t \rangle_{n'})))) @ [b] \wedge \text{mining } (\sigma_{nid'}(t n')))$ **using** *asmp by auto*
ultimately have *False* **by auto**
thus *?thesis ..*
qed
qed
qed

lemma *devExt-devop*:

$\text{prefix } (\text{devExt } t n \text{nid } n_s (\text{Suc } n')) (\text{devExt } t n \text{nid } n_s n') \vee (\exists b. \text{devExt } t n \text{nid } n_s (\text{Suc } n') = \text{devExt } t n \text{nid } n_s n' @ [b]) \wedge \neg \text{Option.is-none } (\text{devBC } t n \text{nid } (n_s + \text{Suc } n')) \wedge \|\text{the } (\text{devBC } t n \text{nid } (n_s + \text{Suc } n'))\|_t (n_s + \text{Suc } n') \wedge n_s + \text{Suc } n' \leq n \wedge \text{mining } (\sigma_{\text{the } (\text{devBC } t n \text{nid } (n_s + \text{Suc } n'))} (t (n_s + \text{Suc } n')))$

proof cases

assume $n_s + \text{Suc } n' > n$
hence $\neg (\exists \text{nid}'. (n_s + \text{Suc } n', \text{nid}') \in \text{his } t n \text{nid})$ **using** *his-le by fastforce*
hence $\text{Option.is-none } (\text{devBC } t n \text{nid } (n_s + \text{Suc } n'))$ **using** *devBC-def by simp*
hence $\text{devExt } t n \text{nid } n_s (\text{Suc } n') = \text{devExt } t n \text{nid } n_s n'$ **by simp**
thus *?thesis by simp*

next

assume $\neg n_s + \text{Suc } n' > n$
hence $n_s + \text{Suc } n' \leq n$ **by simp**
show *?thesis*

proof cases

assume $\text{Option.is-none } (\text{devBC } t n \text{nid } (n_s + \text{Suc } n'))$
hence $\text{devExt } t n \text{nid } n_s (\text{Suc } n') = \text{devExt } t n \text{nid } n_s n'$ **by simp**
thus *?thesis by simp*

next

assume $\neg \text{Option.is-none } (\text{devBC } t n \text{nid } (n_s + \text{Suc } n'))$

hence $\text{devExt } t n \text{nid } n_s (\text{Suc } n') = bc (\sigma_{\text{the } (\text{devBC } t n \text{nid } (n_s + \text{Suc } n'))} (t (n_s + \text{Suc } n')))$ **by**

simp

moreover have prefix $(bc (\sigma_{the} (devBC\ t\ n\ nid\ (n_s + Suc\ n'))(t\ (n_s + Suc\ n')))) (devExt\ t\ n\ nid\ n_s\ n') \vee (\exists b. bc (\sigma_{the} (devBC\ t\ n\ nid\ (n_s + Suc\ n'))(t\ (n_s + Suc\ n')) = devExt\ t\ n\ nid\ n_s\ n' @ [b] \wedge \neg Option.is-none (devBC\ t\ n\ nid\ (n_s + Suc\ n')) \wedge \parallel the (devBC\ t\ n\ nid\ (n_s + Suc\ n')) \parallel_t (n_s + Suc\ n')) \wedge n_s + Suc\ n' \leq n \wedge mining (\sigma_{the} (devBC\ t\ n\ nid\ (n_s + Suc\ n'))(t\ (n_s + Suc\ n'))))$

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 $\langle \neg Option.is-none (devBC\ t\ n\ nid\ (n_s + Suc\ n')) \rangle$

have $n_s + Suc\ n' \leq n$ **using** $devExt-nopt-leq$ **by** $simp$

moreover from $\langle \neg Option.is-none (devBC\ t\ n\ nid\ (n_s + Suc\ n')) \rangle$

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 $(hisPred\ t\ n\ nid\ (n_s + Suc\ n'), (SOME\ nid'. (hisPred\ t\ n\ nid\ (n_s + Suc\ n'), nid') \in his\ t\ n\ nid)) = ?x$

using $\langle \exists n'' < n_s + Suc\ n'. \exists nid'. (n'', nid') \in his\ t\ n\ nid$

$his-determ-ext[of\ n_s + Suc\ n'\ n\ t\ nid] \text{ by } auto$

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 $devExt\ t\ n\ nid\ n_s\ n' = bc (\sigma_{the} (devBC\ t\ n\ nid\ (GREATEST\ n''. n'' < n_s + n' \wedge \neg Option.is-none (devBC\ t\ n\ nid\ n'')) (GREATEST\ n''. n'' < n_s + n' \wedge \neg Option.is-none (devBC\ t\ n\ nid\ n''))))$

proof cases

assume $n' = 0$

moreover have $\exists n'' < n_s + n'. \neg Option.is-none (devBC\ t\ n\ nid\ n'')$

proof –

from $\langle \exists n'' < n_s + Suc\ n'. \exists nid'. (n'', nid') \in his\ t\ n\ nid \rangle$ **obtain** n''

where $n'' < Suc\ n_s + n'$ **and** $\exists nid'. (n'', nid') \in his\ t\ n\ nid$ **by** $auto$

hence $\neg Option.is-none (devBC\ t\ n\ nid\ n'')$ **using** $devBC-def$ **by** $simp$

moreover from $\langle \neg Option.is-none (devBC\ t\ n\ nid\ n'') \rangle$

$\langle Option.is-none (devBC\ t\ n\ nid\ (n_s + n')) \rangle$ **have** $\neg n'' = n_s + n'$ **by** $auto$

with $\langle n'' < Suc\ n_s + n' \rangle$ **have** $n'' < n_s + n'$ **by** $simp$

ultimately show $?thesis$ **by** $auto$

qed

ultimately show $?thesis$ **using** $\langle Option.is-none (devBC\ t\ n\ nid\ (n_s + n')) \rangle$ **by** $simp$

next

assume $\neg n' = 0$

moreover have $\exists n'' < n_s + n'. \neg Option.is-none (devBC\ t\ n\ nid\ n'')$

proof –

from $\langle \exists n'' < n_s + Suc\ n'. \exists nid'. (n'', nid') \in his\ t\ n\ nid \rangle$ **obtain** n''

where $n'' < Suc\ n_s + n'$ **and** $\exists nid'. (n'', nid') \in his\ t\ n\ nid$ **by** $auto$

hence $\neg Option.is-none (devBC\ t\ n\ nid\ n'')$ **using** $devBC-def$ **by** $simp$

moreover from $\langle \neg Option.is-none (devBC\ t\ n\ nid\ n'') \rangle \langle Option.is-none (devBC\ t\ n\ nid\ (n_s + n')) \rangle$

have $\neg n'' = n_s + n'$ **by** $auto$

with $\langle n'' < Suc\ n_s + n' \rangle$ **have** $n'' < n_s + n'$ **by** $simp$

ultimately show $?thesis$ **by** $auto$

qed

with $\langle \neg (n' = 0) \rangle \langle Option.is-none (devBC\ t\ n\ nid\ (n_s + n')) \rangle$ **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 $\langle \neg Option.is-none \ (devBC \ t \ n \ nid \ n'') \rangle \langle Option.is-none \ (devBC \ t \ n \ nid \ (n_s + n')) \rangle$
have $n'' \neq n_s + n'$ **by** *auto*
with $\langle n'' < n_s + Suc \ n' \rangle$ **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 $\langle \exists n'' < n_s + Suc \ n'. \ \exists nid'. \ (n'', nid') \in his \ t \ n \ nid \rangle$
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 $\langle \exists nid'. \ (hisPred \ t \ n \ nid \ (n_s + Suc \ n'), \ nid') \in his \ t \ n \ nid \rangle$
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 $\langle \neg Option.is-none \ (devBC \ t \ n \ nid \ (n_s + n')) \rangle$ **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 $\langle \neg Option.is-none (devBC\ t\ n\ nid\ (n_s + n')) \rangle$ **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 $\langle n' = Suc\ nat \rangle$ **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 $\langle \neg Option.is-none (devBC\ t\ n\ nid\ (n_s + n')) \rangle$ **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 $\langle \exists\ n'' < n_s + Suc\ n'.\ \exists\ nid'.\ (n'', nid') \in his\ t\ n\ nid \rangle$
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 $\langle \exists\ nid'.\ (hisPred\ t\ n\ nid\ (n_s + Suc\ n'),\ nid') \in his\ t\ n\ nid \rangle$
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*
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 $\langle \exists\ x.\ his-prop\ t\ n\ nid\ (n_s + Suc\ n')\ ?nid\ x \rangle$
have $his-prop\ t\ n\ nid\ (n_s + Suc\ n')\ ?nid\ ?x$
using *someI-ex[of $\lambda x.\ his-prop\ t\ n\ nid\ (n_s + Suc\ n')\ ?nid\ x]$* **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 $\langle \exists\ nid'.\ (n_s + Suc\ n',\ nid') \in his\ t\ n\ nid \rangle$
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 $\langle \exists\ nid'.\ (n_s + Suc\ n',\ nid') \in his\ t\ n\ nid \rangle$ **obtain** nid'

where $(n_s + \text{Suc } n', \text{nid}') \in \text{his } t \ n \ \text{nid}$ **by** *auto*
with *his-determ-the* **have** $\text{nid}' = (\text{THE } \text{nid}'. (n_s + \text{Suc } n', \text{nid}') \in \text{his } t \ n \ \text{nid})$ **by** *simp*
with $\langle ?\text{nid} = \text{the } (\text{devBC } t \ n \ \text{nid } (n_s + \text{Suc } n')) \rangle$
have *the* $(\text{devBC } t \ n \ \text{nid } (n_s + \text{Suc } n')) = \text{nid}'$ **by** *simp*
with $\langle (n_s + \text{Suc } n', \text{nid}') \in \text{his } t \ n \ \text{nid} \rangle$ **show** *?thesis* **using** *his-act* **by** *simp*
qed
ultimately show *?thesis*
using $\langle \neg \text{Option.is-none } (\text{devBC } t \ n \ \text{nid } (n_s + \text{Suc } n')) \rangle \langle n_s + \text{Suc } n' \leq n \rangle$ **by** *simp*
next
assume $\neg (\exists n'' < n_s + \text{Suc } n'. \exists \text{nid}'. (n'', \text{nid}') \in \text{his } t \ n \ \text{nid})$
moreover have $(n_s + \text{Suc } n', \text{the } (\text{devBC } t \ n \ \text{nid } (n_s + \text{Suc } n'))) \in \text{his } t \ n \ \text{nid}$
proof –
from $\langle \neg \text{Option.is-none } (\text{devBC } t \ n \ \text{nid } (n_s + \text{Suc } n')) \rangle$
have $\exists \text{nid}'. (n_s + \text{Suc } n', \text{nid}') \in \text{his } t \ n \ \text{nid}$ **using** *his-ex* **by** *simp*
hence *the* $(\text{devBC } t \ n \ \text{nid } (n_s + \text{Suc } n')) = (\text{THE } \text{nid}'. (n_s + \text{Suc } n', \text{nid}') \in \text{his } t \ n \ \text{nid})$
using *devBC-def* **by** *simp*
moreover from $\langle \exists \text{nid}'. (n_s + \text{Suc } n', \text{nid}') \in \text{his } t \ n \ \text{nid} \rangle$ **obtain** nid'
where $(n_s + \text{Suc } n', \text{nid}') \in \text{his } t \ n \ \text{nid}$ **by** *auto*
with *his-determ-the* **have** $\text{nid}' = (\text{THE } \text{nid}'. (n_s + \text{Suc } n', \text{nid}') \in \text{his } t \ n \ \text{nid})$ **by** *simp*
ultimately have *the* $(\text{devBC } t \ n \ \text{nid } (n_s + \text{Suc } n')) = \text{nid}'$ **by** *simp*
with $\langle (n_s + \text{Suc } n', \text{nid}') \in \text{his } t \ n \ \text{nid} \rangle$ **show** *?thesis* **by** *simp*
qed
ultimately have $\text{bc } (\sigma_{\text{the } (\text{devBC } t \ n \ \text{nid } (n_s + \text{Suc } n'))} (t \ (n_s + \text{Suc } n'))) = []$
using *his-bc-empty* **by** *simp*
thus *?thesis* **by** *simp*
qed
ultimately show *?thesis* **by** *simp*
qed
qed

abbreviation *devLgthBC* **where** $\text{devLgthBC } t \ n \ \text{nid } n_s \equiv (\lambda n'. \text{length } (\text{devExt } t \ n \ \text{nid } n_s \ n'))$

theorem *blockchain-save*:

fixes $t :: \text{nat} \Rightarrow \text{cnf}$ **and** n_s **and** sbc **and** n
assumes $\forall \text{nid}. \text{trusted } \text{nid} \longrightarrow \text{prefix } \text{sbc} \ (\text{bc } (\sigma_{\text{nid}} (t \ (\langle \text{nid} \rightarrow t \rangle_{n_s}))))$
and $\forall \text{nid} \in \text{actUt } (t \ n_s). \text{length } (\text{bc } (\sigma_{\text{nid}} (t \ n_s))) < \text{length } \text{sbc}$
and $\text{PoW } t \ n_s \geq \text{length } \text{sbc} + \text{cb}$
and $\forall n' < n_s. \forall \text{nid}. \|\text{nid}\|_{t \ n'} \longrightarrow \text{length } (\text{bc } (\sigma_{\text{nid}} t \ n')) < \text{length } \text{sbc} \vee \text{prefix } \text{sbc} \ (\text{bc } (\sigma_{\text{nid}} (t \ n')))$
and $n \geq n_s$
shows $\forall \text{nid} \in \text{actTr } (t \ n). \text{prefix } \text{sbc} \ (\text{bc } (\sigma_{\text{nid}} (t \ n)))$
proof (*cases*)
assume $\text{sbc} = []$
thus *?thesis* **by** *simp*
next
assume $\neg \text{sbc} = []$
have $n \geq n_s \implies \forall \text{nid} \in \text{actTr } (t \ n). \text{prefix } \text{sbc} \ (\text{bc } (\sigma_{\text{nid}} (t \ n)))$
proof (*induction n rule: ge-induct*)
case (*step n*)
show *?case*
proof
fix nid **assume** $\text{nid} \in \text{actTr } (t \ n)$
hence $\|\text{nid}\|_{t \ n}$ **and** *trusted nid* **using** *actTr-def* **by** *auto*
show $\text{prefix } \text{sbc} \ (\text{bc } (\sigma_{\text{nid}} t \ n))$
proof *cases*
assume $\text{lAct}: \exists n' < n. n' \geq n_s \wedge \|\text{nid}\|_{t \ n'}$

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

qed
then obtain n' **where** $n' \leq \langle \text{nid} \leftarrow t \rangle_n$ **and** $\neg \text{Option.is-none} (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n')$
 n'
and $n' \geq n_s$ **and** *trusted* (the $(\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n')$)
and $\forall n'' > n'. n'' \leq \langle \text{nid} \leftarrow t \rangle_n \wedge \neg \text{Option.is-none} (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n'') \rightarrow$
 $\neg \text{trusted} (\text{the } (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n''))$ **by** *auto*
hence $n' \geq n_s$ **and** *untrusted*: $\forall n'' > n'. n'' \leq \langle \text{nid} \leftarrow t \rangle_n \wedge \neg \text{Option.is-none} (\text{devBC } t$
 $\langle \text{nid} \leftarrow t \rangle_n \text{nid}' n'') \rightarrow \neg \text{trusted} (\text{the } (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n''))$ **by** *auto*
moreover have $\langle \text{nid} \leftarrow t \rangle_n < n$ **using** $\langle \exists n'. \text{latestAct-cond } \text{nid } t \ n \ n' \rangle \text{latestAct-prop}(2)$
by *blast*
with $\langle n' \leq \langle \text{nid} \leftarrow t \rangle_n \rangle$ **have** $n' < n$ **by** *simp*
moreover from $\langle \neg \text{Option.is-none} (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n') \rangle$
have $\| \text{the } (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n') \|_{t \ n'}$ **using** *devBC-act* **by** *simp*
with $\langle \text{trusted} (\text{the } (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n')) \rangle$
have the $(\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n') \in \text{actTr} (t \ n')$ **using** *actTr-def* **by** *simp*
ultimately have *prefix sbc* (bc $(\sigma_{\text{the } (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n')^t \ n'})$)
using *step.IH* **by** *simp*

interpret *ut*: *untrusted devExt* $t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n' \lambda n. \text{umining } t (n' + n)$
proof
fix n''
from *devExt-devop*[of $t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n'$] **have** *prefix* $(\text{devExt } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}'$
 $n' (\text{Suc } n'')) (\text{devExt } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n' n'') \vee (\exists b. \text{devExt } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n' (\text{Suc } n'') = \text{devExt}$
 $t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n' n'' @ [b]) \wedge \neg \text{Option.is-none} (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' (n' + \text{Suc } n'')) \wedge \| \text{the}$
 $(\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' (n' + \text{Suc } n'')) \|_{t (n' + \text{Suc } n'')} \wedge n' + \text{Suc } n'' \leq \langle \text{nid} \leftarrow t \rangle_n \wedge \text{mining}$
 $(\sigma_{\text{the } (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' (n' + \text{Suc } n''))^t (n' + \text{Suc } n'')})$.
thus *prefix* $(\text{devExt } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n' (\text{Suc } n'')) (\text{devExt } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n' n'') \vee$
 $(\exists b. \text{devExt } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n' (\text{Suc } n'') = \text{devExt } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n' n'' @ [b]) \wedge \text{umining } t (n'$
 $+ \text{Suc } n'')$
proof
assume *prefix* $(\text{devExt } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n' (\text{Suc } n'')) (\text{devExt } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n'$
 $n'')$
thus *?thesis* **by** *simp*
next
assume $(\exists b. \text{devExt } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n' (\text{Suc } n'') = \text{devExt } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n' n'' @ [b])$
 $\wedge \neg \text{Option.is-none} (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' (n' + \text{Suc } n'')) \wedge \| \text{the } (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' (n' + \text{Suc}$
 $n'')) \|_{t (n' + \text{Suc } n'')} \wedge n' + \text{Suc } n'' \leq \langle \text{nid} \leftarrow t \rangle_n \wedge \text{mining} (\sigma_{\text{the } (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' (n' + \text{Suc } n''))^t$
 $(n' + \text{Suc } n'')})$
hence $\exists b. \text{devExt } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n' (\text{Suc } n'') = \text{devExt } t \langle \text{nid} \leftarrow t \rangle_n$
 $\text{nid}' n' n'' @ [b]$ **and** $\neg \text{Option.is-none} (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' (n' + \text{Suc } n''))$ **and** $\| \text{the } (\text{devBC}$
 $t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' (n' + \text{Suc } n'')) \|_{t (n' + \text{Suc } n'')}$ **and** $n' + \text{Suc } n'' \leq \langle \text{nid} \leftarrow t \rangle_n$ **and** *mining*
 $(\sigma_{\text{the } (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' (n' + \text{Suc } n''))^t (n' + \text{Suc } n'')})$ **by** *auto*
moreover from $\langle n' + \text{Suc } n'' \leq \langle \text{nid} \leftarrow t \rangle_n \rangle \langle \neg \text{Option.is-none} (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n$
 $\text{nid}' (n' + \text{Suc } n'')) \rangle$ **have** $\neg \text{trusted} (\text{the } (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' (n' + \text{Suc } n'')))$ **using** *untrusted*
by *simp*
with $\langle \| \text{the } (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' (n' + \text{Suc } n'')) \|_{t (n' + \text{Suc } n'')} \rangle$ **have** the
 $(\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' (n' + \text{Suc } n'')) \in \text{actUt} (t (n' + \text{Suc } n''))$ **using** *actUt-def* **by** *simp*
ultimately show *?thesis* **using** *umining-def* **by** *auto*
qed
qed
from $\langle \neg \text{Option.is-none} (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n') \rangle$ **have** bc $(\sigma_{\text{the } (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n')^t$
 $n'}) = \text{devExt } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n' 0$
using *devExt-bc-geq*[of $t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n'$] **by** *simp*
moreover from $\langle n' \leq \langle \text{nid} \leftarrow t \rangle_n \rangle \langle \| \text{nid}' \|_{t \langle \text{nid} \leftarrow t \rangle_n} \rangle$ **have** bc $(\sigma_{\text{nid}'^t \langle \text{nid} \leftarrow t \rangle_n} =$

$devExt\ t\ \langle nid \leftarrow t \rangle_n\ nid'\ n'\ (\langle nid \leftarrow t \rangle_{n-n'})$
using $devExt\text{-}bc\text{-}geq$ **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 $(prefix\ sbc\ (bc\ (\sigma_{the\ (devBC\ t\ \langle nid \leftarrow t \rangle_n\ nid'\ n')}(t\ n'''))$
 $ut.prefix\ length[of\ sbc\ 0\ \langle nid \leftarrow t \rangle_{n-n'}]$ **by** $auto$
then obtain n_p **where** $n_p \leq \langle nid \leftarrow t \rangle_{n-n'}$
and $length\ (devExt\ t\ \langle nid \leftarrow t \rangle_n\ nid'\ n'\ n_p) < length\ sbc$ **by** $auto$
hence $length\ (devExt\ t\ \langle nid \leftarrow t \rangle_n\ nid'\ (n' + n_p)\ 0) < length\ sbc$ **using** $devExt\text{-}shift[of$
 $t\ \langle nid \leftarrow t \rangle_n\ nid'\ n'\ n_p]$ **by** $simp$
moreover from $\langle \langle nid \leftarrow t \rangle_{n \geq n'} \langle n_p \leq \langle nid \leftarrow t \rangle_{n-n'} \rangle$ **have** $(n' + n_p) \leq \langle nid \leftarrow t \rangle_n$
by $simp$
ultimately show $?thesis$ **using** $\langle n' \geq n_s \rangle$ **untrusted by** $auto$
next
assume $\neg(\exists\ n' \leq \langle nid \leftarrow t \rangle_n.\ n' \geq n_s \wedge \neg\ Option.is\ none\ (devBC\ t\ \langle nid \leftarrow t \rangle_n\ nid'\ n') \wedge$
 $trusted\ (the\ (devBC\ t\ \langle nid \leftarrow t \rangle_n\ nid'\ n')))$
hence cas: $\forall\ n' \leq \langle nid \leftarrow t \rangle_n.\ n' \geq n_s \wedge \neg\ Option.is\ none\ (devBC\ t\ \langle nid \leftarrow t \rangle_n\ nid'\ n')$
 $\longrightarrow \neg\ trusted\ (the\ (devBC\ t\ \langle nid \leftarrow t \rangle_n\ nid'\ n'))$ **by** $auto$
show $?thesis$
proof cases
assume $Option.is\ none\ (devBC\ t\ \langle nid \leftarrow t \rangle_n\ nid'\ n_s)$
thus $?thesis$
proof cases
assume $\forall\ n' < n_s.\ Option.is\ none\ (devBC\ t\ \langle nid \leftarrow t \rangle_n\ nid'\ n')$
with $\langle Option.is\ none\ (devBC\ t\ \langle nid \leftarrow t \rangle_n\ nid'\ n_s) \rangle$ **have** $devExt\ t\ \langle nid \leftarrow t \rangle_n\ nid'$
 $n_s\ 0 = []$ **by** $simp$
with $(\neg\ sbc = [])$ **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** $\forall\ n'' > n_s.\ n'' \leq \langle nid \leftarrow t \rangle_n \wedge \neg\ Option.is\ none\ (devBC\ t$
 $\langle nid \leftarrow t \rangle_n\ nid'\ n'')$ $\longrightarrow \neg\ trusted\ (the\ (devBC\ t\ \langle nid \leftarrow t \rangle_n\ nid'\ n''))$ **by** $simp$
ultimately show $?thesis$ **by** $auto$
next
let $?P = \lambda n'.\ n' < n_s \wedge \neg\ Option.is\ none\ (devBC\ t\ \langle nid \leftarrow t \rangle_n\ nid'\ n')$
let $?n' = GREATEST\ n'.\ ?P\ n'$
assume $\neg(\forall\ n' < n_s.\ Option.is\ none\ (devBC\ t\ \langle nid \leftarrow t \rangle_n\ nid'\ n'))$
moreover have $\forall\ n' > n_s.\ \neg\ ?P\ n'$ **by** $simp$
ultimately have exists: $\exists\ n'.\ ?P\ n' \wedge (\forall\ n''.\ ?P\ n'' \longrightarrow n'' \leq n')$
using $boundedGreatest[of\ ?P]$ **by** $blast$
hence $?P\ ?n'$ **using** $GreatestI\text{-}ex\text{-}nat[of\ ?P]$ **by** $auto$
moreover from $\langle ?P\ ?n' \rangle$ **have** $\|the\ (devBC\ t\ \langle nid \leftarrow t \rangle_n\ nid'\ ?n')\|_{t\ ?n'}$ **using**
 $devBC\text{-}act$ **by** $simp$
ultimately have $length\ (bc\ (\sigma_{the\ (devBC\ t\ \langle nid \leftarrow t \rangle_n\ nid'\ ?n')}(t\ ?n')) < length\ sbc \vee$
 $prefix\ sbc\ (bc\ (\sigma_{the\ (devBC\ t\ \langle nid \leftarrow t \rangle_n\ nid'\ ?n')}(t\ ?n')))$ **using** $assms(4)$ **by** $simp$
thus $?thesis$
proof
assume $length\ (bc\ (\sigma_{the\ (devBC\ t\ \langle nid \leftarrow t \rangle_n\ nid'\ ?n')}(t\ ?n')) < length\ sbc$
moreover from exists **have** $\neg(\exists\ n' > ?n'.\ ?P\ n')$ **using** $Greatest\text{-}ex\text{-}le\text{-}nat[of\ ?P]$ **by**
 $simp$
moreover from $\langle ?P\ ?n' \rangle$ **have** $\exists\ n' < n_s.\ \neg\ Option.is\ none\ (devBC\ t\ \langle nid \leftarrow t \rangle_n$
 $nid'\ n')$ **by** $blast$
with $\langle Option.is\ none\ (devBC\ t\ \langle nid \leftarrow t \rangle_n\ nid'\ n_s) \rangle$
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 $\text{length } (\text{devExt } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n_s 0) < \text{length } \text{sbc}$ by simp
 moreover from lAct have $\langle \text{nid} \leftarrow t \rangle_n \geq n_s$ using latestActless by blast

moreover from cas have $\forall n'' > n_s. n'' \leq \langle \text{nid} \leftarrow t \rangle_n \wedge \neg \text{Option.is-none } (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n'')$
 $\longrightarrow \neg \text{trusted } (\text{the } (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n''))$ by simp

ultimately show ?thesis by auto

next

interpret ut : $\text{untrusted devExt } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n_s \lambda n. \text{umining } t (n_s + n)$

proof

fix n''

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

thus $\text{prefix } (\text{devExt } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n_s (\text{Suc } n'')) (\text{devExt } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n_s n'') \vee (\exists b. \text{devExt } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n_s (\text{Suc } n'') = \text{devExt } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n_s n'' @ [b]) \wedge \text{umining } t (n_s + \text{Suc } n'')$

proof

assume $\text{prefix } (\text{devExt } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n_s (\text{Suc } n'')) (\text{devExt } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n_s n'')$ thus ?thesis by simp

next

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

hence $\exists b. \text{devExt } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n_s (\text{Suc } n'') = \text{devExt } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n_s n'' @ [b]$

and $\neg \text{Option.is-none } (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' (n_s + \text{Suc } n''))$

and $\|\text{the } (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' (n_s + \text{Suc } n''))\|_t (n_s + \text{Suc } n'')$

and $n_s + \text{Suc } n'' \leq \langle \text{nid} \leftarrow t \rangle_n$

and $\text{mining } (\sigma_{\text{the } (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' (n_s + \text{Suc } n''))} t (n_s + \text{Suc } n''))$

by auto

moreover from $\langle n_s + \text{Suc } n'' \leq \langle \text{nid} \leftarrow t \rangle_n \rangle \langle \neg \text{Option.is-none } (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' (n_s + \text{Suc } n'')) \rangle$

have $\neg \text{trusted } (\text{the } (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' (n_s + \text{Suc } n'')))$

using cas by simp

with $\langle \|\text{the } (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' (n_s + \text{Suc } n''))\|_t (n_s + \text{Suc } n'') \rangle$

have $\text{the } (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' (n_s + \text{Suc } n'')) \in \text{actUt } (t (n_s + \text{Suc } n''))$

using actUt-def by simp

ultimately show ?thesis using umining-def by auto

qed

qed

assume $\text{prefix } \text{sbc } (bc (\sigma_{\text{the } (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' ?n')} (t ?n')))$

moreover from exists have $\neg (\exists n' > ?n'. ?P n')$ using $\text{Greatest-ex-le-nat}[of } ?P]$ by simp

moreover from $\langle ?P ?n' \rangle$ have $\exists n' < n_s. \neg \text{Option.is-none } (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n')$ by blast

with $\langle \text{Option.is-none } (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n_s) \rangle$ have $\text{devExt } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n_s 0 = bc (\sigma_{\text{the } (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' ?n')} (t ?n'))$ by simp

ultimately have $\text{prefix } \text{sbc } (\text{devExt } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n_s 0)$ by simp

moreover from lAct have $\langle \text{nid} \leftarrow t \rangle_n \geq n_s$ using latestActless by blast

with $\langle \|\text{nid}'\|_t \langle \text{nid} \leftarrow t \rangle_n \rangle$ have $bc (\sigma_{\text{the } (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' \langle \text{nid} \leftarrow t \rangle_n) t \langle \text{nid} \leftarrow t \rangle_n}$

$\leftarrow t\}_n) = \text{devExt } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n_s (\langle \text{nid} \leftarrow t \rangle_{n-n_s})$ **using** *devExt-bc-geq* **by simp**
with $\neg \text{prefix } \text{sb}c (bc (\sigma_{\text{nid}'}(t \langle \text{nid} \leftarrow t \rangle_n))) \langle \|\text{nid}'\|_t \langle \text{nid} \leftarrow t \rangle_n \rangle$ **have** $\neg \text{prefix}$
 $\text{sb}c (\text{devExt } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n_s (\langle \text{nid} \leftarrow t \rangle_{n-n_s}))$ **by simp**
ultimately have $\exists n''' > 0. n''' \leq \langle \text{nid} \leftarrow t \rangle_{n-n_s} \wedge \text{length} (\text{devExt } t \langle \text{nid} \leftarrow t \rangle_n$
 $\text{nid}' n_s n''') < \text{length } \text{sb}c$ **using** *ut.prefix-length*[of $\text{sb}c 0 \langle \text{nid} \leftarrow t \rangle_{n-n_s}$] **by simp**
then obtain n_p **where** $n_p > 0$ **and** $n_p \leq \langle \text{nid} \leftarrow t \rangle_{n-n_s}$ **and** $\text{length} (\text{devExt } t \langle \text{nid}$
 $\leftarrow t \rangle_n \text{nid}' n_s n_p) < \text{length } \text{sb}c$ **by auto**
hence $\text{length} (\text{devExt } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' (n_s + n_p) 0) < \text{length } \text{sb}c$ **using** *devExt-shift*
by simp
moreover from *lAct* **have** $\langle \text{nid} \leftarrow t \rangle_{n \geq n_s}$ **using** *latestActless* **by blast**
with $\langle n_p \leq \langle \text{nid} \leftarrow t \rangle_{n-n_s} \rangle$ **have** $(n_s + n_p) \leq \langle \text{nid} \leftarrow t \rangle_n$ **by simp**
moreover from $\langle n_p \leq \langle \text{nid} \leftarrow t \rangle_{n-n_s} \rangle$ **have** $n_p \leq \langle \text{nid} \leftarrow t \rangle_n$ **by simp**
moreover have $\forall n'' > n_s + n_p. n'' \leq \langle \text{nid} \leftarrow t \rangle_n \wedge \neg \text{Option.is-none} (\text{devBC } t$
 $\langle \text{nid} \leftarrow t \rangle_n \text{nid}' n'')$ $\longrightarrow \neg \text{trusted} (\text{the} (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n''))$ **using** *cas* **by simp**
ultimately show *?thesis* **by auto**
qed
qed
next
assume *asmp*: $\neg \text{Option.is-none} (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n_s)$
moreover from *lAct* **have** $n_s \leq \langle \text{nid} \leftarrow t \rangle_n$ **using** *latestActless* **by blast**
ultimately have $\neg \text{trusted} (\text{the} (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n_s))$ **using** *cas* **by simp**
moreover from *asmp* **have** $\|\text{the} (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n_s)\|_t n_s$
using *devBC-act* **by simp**
ultimately have $\text{the} (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n_s) \in \text{actUt} (t n_s)$
using *actUt-def* **by simp**
hence $\text{length} (bc (\sigma_{\text{the} (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n_s)}(t n_s))) < \text{length } \text{sb}c$
using *assms(2)* **by simp**
moreover from *asmp* **have**
 $\text{devExt } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n_s 0 = bc (\sigma_{\text{the} (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n_s)}(t n_s))$
by simp
ultimately have $\text{length} (\text{devExt } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n_s 0) < \text{length } \text{sb}c$ **by simp**
moreover from *lAct* **have** $\langle \text{nid} \leftarrow t \rangle_{n \geq n_s}$ **using** *latestActless* **by blast**
moreover from *cas* **have** $\forall n'' > n_s. n'' \leq \langle \text{nid} \leftarrow t \rangle_n \wedge \neg \text{Option.is-none} (\text{devBC } t \langle \text{nid}$
 $\leftarrow t \rangle_n \text{nid}' n'')$ $\longrightarrow \neg \text{trusted} (\text{the} (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n''))$ **by simp**
ultimately show *?thesis* **by auto**
qed
qed
then obtain n' **where** $\langle \text{nid} \leftarrow t \rangle_{n \geq n'}$ **and** $n' \geq n_s$
and $\text{length} (\text{devExt } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n' 0) < \text{length } \text{sb}c$
and $\text{untrusted}: \forall n'' > n'. n'' \leq \langle \text{nid} \leftarrow t \rangle_n \wedge \neg \text{Option.is-none} (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}'$
 $n'')$ $\longrightarrow \neg \text{trusted} (\text{the} (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n''))$ **by auto**
interpret *ut*: $\text{untrusted } \text{devExt } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n' \lambda n. \text{umining } t (n' + n)$
proof
fix n''
from *devExt-devop*[of $t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n'$]
have $\text{prefix} (\text{devExt } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n' (\text{Suc } n'')) (\text{devExt } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n' n'') \vee$
 $(\exists b. \text{devExt } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n' (\text{Suc } n'') = \text{devExt } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n' n'' @ [b])$
 $\wedge \neg \text{Option.is-none} (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' (n' + \text{Suc } n'')) \wedge \|\text{the} (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' (n' + \text{Suc}$
 $n''))\|_t (n' + \text{Suc } n'') \wedge n' + \text{Suc } n'' \leq \langle \text{nid} \leftarrow t \rangle_n \wedge \text{mining} (\sigma_{\text{the} (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' (n' + \text{Suc } n''))}^t$
 $(n' + \text{Suc } n''))).$
thus $\text{prefix} (\text{devExt } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n' (\text{Suc } n'')) (\text{devExt } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n' n'')$
 $\vee (\exists b. \text{devExt } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n' (\text{Suc } n'') = \text{devExt } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n' n'' @ [b])$
 $\wedge \text{umining } t (n' + \text{Suc } n'')$
proof
assume $\text{prefix} (\text{devExt } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n' (\text{Suc } n'')) (\text{devExt } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n' n'')$

thus *?thesis by simp*
next
assume $(\exists b. \text{devExt } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n' (\text{Suc } n'') = \text{devExt } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n' n'' @ [b])$
 $\wedge \neg \text{Option.is-none } (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' (n' + \text{Suc } n'')) \wedge \ll \text{the } (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' (n' + \text{Suc } n'')) \gg_t (n' + \text{Suc } n'') \wedge n' + \text{Suc } n'' \leq \langle \text{nid} \leftarrow t \rangle_n \wedge \text{mining } (\sigma_{\text{the}} (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' (n' + \text{Suc } n'')))^t (n' + \text{Suc } n'')$
hence $\exists b. \text{devExt } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n' (\text{Suc } n'') = \text{devExt } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n' n'' @ [b]$
and $\neg \text{Option.is-none } (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' (n' + \text{Suc } n''))$
and $\ll \text{the } (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' (n' + \text{Suc } n'')) \gg_t (n' + \text{Suc } n'')$
and $n' + \text{Suc } n'' \leq \langle \text{nid} \leftarrow t \rangle_n$
and $\text{mining } (\sigma_{\text{the}} (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' (n' + \text{Suc } n'')))^t (n' + \text{Suc } n'')$
by auto
moreover from $\langle n' + \text{Suc } n'' \leq \langle \text{nid} \leftarrow t \rangle_n \rangle \neg \text{Option.is-none } (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' (n' + \text{Suc } n''))$
have $\neg \text{trusted } (\text{the } (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' (n' + \text{Suc } n'')))$ **using** *untrusted by simp*
with $\langle \ll \text{the } (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' (n' + \text{Suc } n'')) \gg_t (n' + \text{Suc } n'') \rangle$
have $\text{the } (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' (n' + \text{Suc } n'')) \in \text{actUt } (t (n' + \text{Suc } n''))$
using *actUt-def by simp*
ultimately show *?thesis using umining-def by auto*
qed
qed
interpret *untrusted-growth devLgthBC* $t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n' \lambda n. \text{umining } t (n' + n)$
by *unfold-locales*
interpret *trusted-growth* $\lambda n. \text{PoW } t (n' + n) \lambda n. \text{tmining } t (n' + n)$
proof
show $\bigwedge n. \text{PoW } t (n' + n) \leq \text{PoW } t (n' + \text{Suc } n)$ **using** *pow-mono by simp*
show $\bigwedge n. \text{tmining } t (n' + \text{Suc } n) \implies \text{PoW } t (n' + n) < \text{PoW } t (n' + \text{Suc } n)$
using *pow-mining-suc by simp*
qed
interpret *bg: bounded-growth length sbc* $\lambda n. \text{PoW } t (n' + n) \text{devLgthBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n' \lambda n. \text{tmining } t (n' + n) \lambda n. \text{umining } t (n' + n) \text{length sbc } cb$
proof
from *assms(3)* $\langle n' \geq n_s \rangle$ **show** $\text{length sbc} + cb \leq \text{PoW } t (n' + 0)$ **using** *pow-mono[of n_s n' t]* **by simp**
next
from $\langle \text{length } (\text{devExt } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n' 0) < \text{length sbc} \rangle$ **show** $\text{length } (\text{devExt } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n' 0) < \text{length sbc}$.
next
fix $n'' n'''$
assume $cb < \text{card } \{i. n'' < i \wedge i \leq n''' \wedge \text{umining } t (n' + i)\}$
hence $cb < \text{card } \{i. n'' + n' < i \wedge i \leq n''' + n' \wedge \text{umining } t i\}$
using *cardshift[of n'' n''' umining t n'] by simp*
with *fair[of n'' + n' n''' + n' t]*
have $cb < \text{card } \{i. n'' + n' < i \wedge i \leq n''' + n' \wedge \text{tmining } t i\}$ **by simp**
thus $cb < \text{card } \{i. n'' < i \wedge i \leq n''' \wedge \text{tmining } t (n' + i)\}$
using *cardshift[of n'' n''' tmining t n'] by simp*
qed
from $\langle \langle \text{nid} \leftarrow t \rangle_n \geq n' \rangle$ **have** $\text{length } (\text{devExt } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' n' (\langle \text{nid} \leftarrow t \rangle_{n-n'})) < \text{PoW } t \langle \text{nid} \leftarrow t \rangle_n$
using *bg.tr-upper-bound[of < nid ← t >_n - n'] by simp*
moreover from $\langle \ll \text{nid} \gg_t \langle \text{nid} \leftarrow t \rangle_n \rangle \langle \langle \text{nid} \leftarrow t \rangle_n \geq n' \rangle$
have $bc (\sigma_{\text{the}} (\text{devBC } t \langle \text{nid} \leftarrow t \rangle_n \text{nid}' \langle \text{nid} \leftarrow t \rangle_n)^t \langle \text{nid} \leftarrow t \rangle_n) = \text{devExt } t \langle \text{nid} \leftarrow t \rangle_n$

$nid' n' (\langle nid \leftarrow t \rangle_{n-n'})$
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 $\langle trusted\ nid \rangle \langle \|nid\|_t \langle nid \leftarrow t \rangle_n \rangle$
have $?lhs \leq length (MAX (pin (\sigma_{nid'} \langle nid \leftarrow t \rangle_n)))$ **using** *pow-le-max* **by** *simp*
also from $\langle bc (\sigma_{nid'}(t \langle nid \leftarrow t \rangle_n)) = MAX (pin (\sigma_{nid'} \langle nid \leftarrow t \rangle_n)) \rangle$
have $\dots = length (bc (\sigma_{nid'}(t \langle nid \leftarrow t \rangle_n)))$ **by** *simp*
finally show *?thesis* .
qed
ultimately show *False* **by** *simp*
qed
moreover from $\langle \|nid\|_t n \rangle$ **have** $\langle nid \rightarrow t \rangle_{n=n}$ **using** *nxtAct-active* **by** *simp*
ultimately show *?thesis* **by** *auto*
qed
moreover from $\langle \|nid\|_t n \rangle$ **have** $\langle nid \rightarrow t \rangle_{n=n}$ **using** *nxtAct-active* **by** *simp*
ultimately show *?thesis* **by** *auto*
next
assume $\neg (\exists b \in pin (\sigma_{nid'} \langle nid \leftarrow t \rangle_n). length\ b > length (bc (\sigma_{nid'} \langle nid \leftarrow t \rangle_n)))$
moreover from $\langle \|nid\|_t n \rangle$ **have** $\exists n' \geq n. \|nid\|_t n'$ **by** *auto*
moreover from *lAct* **have** $\exists n'. latestAct-cond\ nid\ t\ n\ n'$ **by** *auto*
ultimately have $\neg mining (\sigma_{nid'} \langle nid \rightarrow t \rangle_n) \wedge bc (\sigma_{nid'} \langle nid \rightarrow t \rangle_n) = bc (\sigma_{nid'} \langle nid \leftarrow t \rangle_n) \vee$
 $mining (\sigma_{nid'} \langle nid \rightarrow t \rangle_n) \wedge (\exists b. bc (\sigma_{nid'} \langle nid \rightarrow t \rangle_n) = bc (\sigma_{nid'} \langle nid \leftarrow t \rangle_n) @ [b])$
using $\langle trusted\ nid \rangle bhv-tr-in$ [of $nid\ n\ t$] **by** *simp*
moreover have *prefix sbc* $(bc (\sigma_{nid'} \langle nid \leftarrow t \rangle_n))$
proof –
from $\langle \exists n'. latestAct-cond\ nid\ t\ n\ n' \rangle$ **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*
moreover from $\langle \exists n'. latestAct-cond\ nid\ t\ n\ n' \rangle$ **have** $\|nid\|_t \langle nid \leftarrow t \rangle_n$
using *latestAct-prop(1)* **by** *simp*
with $\langle trusted\ nid \rangle$ **have** $nid \in actTr (t \langle nid \leftarrow t \rangle_n)$ **using** *actTr-def* **by** *simp*
ultimately show *?thesis* **using** *step.IH* **by** *auto*
qed
moreover from $\langle \|nid\|_t n \rangle$ **have** $\langle nid \rightarrow t \rangle_{n=n}$ **using** *nxtAct-active* **by** *simp*
ultimately show *?thesis* **by** *auto*
qed
next
assume $nAct: \neg (\exists n' < n. n' \geq n_s \wedge \|nid\|_t n')$
moreover from *step.hyps* **have** $n_s \leq n$ **by** *simp*
ultimately have $\langle nid \rightarrow t \rangle_{n_s} = n$ **using** $\langle \|nid\|_t n \rangle$ *nxtAct-eq*[of $n_s\ n\ nid\ t$] **by** *simp*
with $\langle trusted\ nid \rangle$ **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 Marmosler. 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 Marmosler. Dynamic architectures. *Archive of Formal Proofs*, July 2017. <http://isa-afp.org/entries/DynamicArchitectures.html>, Formal proof development.
- [4] Diego Marmosler. Hierarchical specification and verification 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.