A Theory of Architectural Design Patterns

Diego Marmsoler

March 1, 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

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

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

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

lemma *the-unique*:
  fixes *k*::*cnf* **and** *id*::*'id*
  assumes ‖*id*‖$_k$
  shows *id* = *the-singleton*
proof −
  have (*THE id*. ∀ *k*. ∀ *id'*. ‖*id'*‖$_k$ ⟶ *id'* = *id*) = *id*
  proof (*rule the-equality*)
    show ∀ *k id'*. ‖*id'*‖$_k$ ⟶ *id'* = *id*
    proof
      fix *k* **show** ∀ *id'*. ‖*id'*‖$_k$ ⟶ *id'* = *id*
      proof
        fix *id'* **show** ‖*id'*‖$_k$ ⟶ *id'* = *id*
        proof
          assume ‖*id'*‖$_k$
          from *unique* **have** ∃ *id*. ∀ *k*. ∀ *id'*. (‖*id'*‖$_k$ ⟶ *id* = *id'*) **.**
          then obtain *i''* **where** ∀ *k*. ∀ *id'*. (‖*id'*‖$_k$ ⟶ *i''* = *id'*) **by** *auto*
          with ‹‖*id'*‖$_k$› **have** *id*=*i''* **and** *id'*=*i''* **using** *assms* **by** *auto*
          thus *id'* = *id* **by** *simp*
        qed
      qed
    qed
  next
    fix *i''* **show** ∀ *k id'*. ‖*id'*‖$_k$ ⟶ *id'* = *i''* ⟹ *i''* = *id* **using** *assms* **by** *auto*
  qed
  thus *?thesis* **by** (*simp add*: *the-singleton-def*)

qed

lemma *the-active*[*simp*]:
fixes k
shows $\|\mathit{the\text{-}singleton}\|_k$
proof −
from *alwaysActive* **obtain** *id* **where** $\|\mathit{id}\|_k$ **by** *blast*
with *the-unique* **have** *id* = *the-singleton* **by** *simp*
with ‹$\|\mathit{id}\|_k$› **show** *?thesis* **by** *simp*
qed

lemma *lNact-active*[*simp*]:
fixes *cid t n*
shows $\langle \mathit{the\text{-}singleton} \leftarrow t\rangle_n = n$
using *lNact-active the-active* **by** *auto*

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

lemma *assI*[*intro*]:
fixes *t n a*
assumes φ ($\sigma_{\mathit{the\text{-}singleton}}(t\ n)$)
shows *eval the-singleton t t′ n* (*ass* φ) **using** *assms* **by** (*simp add*: *assIANow*)

lemma *assE*[*elim*]:
fixes *t n a*
assumes *eval the-singleton t t′ n* (*ass* φ)
shows φ ($\sigma_{\mathit{the\text{-}singleton}}(t\ n)$) **using** *assms* **by** (*simp add*: *assEANow*)

lemma *evtE*[*elim*]:
fixes *t id n a*
assumes *eval the-singleton t t′ n* (*evt* γ)
shows $\exists\, n'{\geq}n.$ *eval the-singleton t t′ n′* γ
proof −
have $\|\mathit{the\text{-}singleton}\|_{t\ n}$ **by** *simp*
with *assms* **obtain** n' **where** $n'{\geq}\langle \mathit{the\text{-}singleton} \rightarrow t\rangle_n$ **and** ($\exists\, i{\geq}n'.\ \|\mathit{the\text{-}singleton}\|_{t\ i} \wedge$
($\forall\, n''{\geq}\langle \mathit{the\text{-}singleton} \leftarrow t\rangle_{n'}.\ n'' \leq \langle \mathit{the\text{-}singleton} \rightarrow t\rangle_{n'} \longrightarrow$ *eval the-singleton t t′ n″* γ)) ∨
¬ ($\exists\, i{\geq}n'.\ \|\mathit{the\text{-}singleton}\|_{t\ i}$) ∧ *eval the-singleton t t′ n′* γ **using** *evtEA*[*of n the-singleton t*] **by** *blast*
moreover have $\|\mathit{the\text{-}singleton}\|_{t\ n'}$ **by** *simp*
ultimately have
$\forall\, n''{\geq}\langle \mathit{the\text{-}singleton} \leftarrow t\rangle_{n'}.\ n'' \leq \langle \mathit{the\text{-}singleton} \rightarrow t\rangle_{n'} \longrightarrow$ *eval the-singleton t t′ n″* γ **by** *auto*
hence *eval the-singleton t t′ n′* γ **by** *simp*
moreover from ‹$n'{\geq}\langle \mathit{the\text{-}singleton} \rightarrow t\rangle_n$› **have** $n'{\geq}n$ **by** (*simp add*: *nxtAct-active*)
ultimately show *?thesis* **by** *auto*
qed

lemma *globE*[*elim*]:

fixes t id n a
assumes $eval\ the\text{-}singleton\ t\ t'\ n\ (glob\ \gamma)$
shows $\forall\, n'{\geq}n.\ eval\ the\text{-}singleton\ t\ t'\ n'\ \gamma$
proof
fix n' **show** $n \leq n' \longrightarrow eval\ the\text{-}singleton\ t\ t'\ n'\ \gamma$
proof
assume $n{\leq}n'$
hence $\langle the\text{-}singleton \leftarrow t\rangle_n \leq n'$ **by** *simp*
moreover have $\|the\text{-}singleton\|_{t\ n}$ **by** *simp*
ultimately show $eval\ the\text{-}singleton\ t\ t'\ n'\ \gamma$
using ‹$eval\ the\text{-}singleton\ t\ t'\ n\ (glob\ \gamma)$› $globEA$ **by** *blast*
qed
qed

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

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

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

end

2 A Theory of Publisher-Subscriber Architectures

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

theory *Publisher-Subscriber*
imports *Singleton*
begin

2.1 Subscriptions

datatype $('id, 'evt)$ *subscription* $=$ *sub* $'id\ 'evt \mid$ *unsub* $'id\ 'evt$

2.2 Publisher-Subscriber Architectures

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

$\Longrightarrow$ *sbnt* (*sbcmp sid* (*t n''*)) = *pbnt* (*pbcmp pid* (*t n''*))
begin

notation *pb.imp* (**infixl** $\longrightarrow^p$ *10*)
notation *pb.or* (**infixl** $\vee^p$ *15*)
notation *pb.and* (**infixl** $\wedge^p$ *20*)
notation *pb.not* ($\neg^p$ - [*19*]*19*)
no-notation *pb.all* (**binder** $\forall_b$ *10*)
no-notation *pb.exists* (**binder** $\exists_b$ *10*)
notation *pb.all* (**binder** $\forall_p$ *10*)
notation *pb.exists* (**binder** $\exists_p$ *10*)

notation *sb.imp* (**infixl** $\longrightarrow^s$ *10*)
notation *sb.or* (**infixl** $\vee^s$ *15*)
notation *sb.and* (**infixl** $\wedge^s$ *20*)
notation *sb.not* ($\neg^s$ - [*19*]*19*)
no-notation *sb.all* (**binder** $\forall_b$ *10*)
no-notation *sb.exists* (**binder** $\exists_b$ *10*)
notation *sb.all* (**binder** $\forall_s$ *10*)
notation *sb.exists* (**binder** $\exists_s$ *10*)

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

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'' sid E e m*
assumes *t* $\in$ *arch*
and *sbactive sid* (*t n*)
and *sub sid E* = *sbsb* (*sbcmp sid* (*t n*))
and *n''* $\geq$ *n*
and $\nexists$ *n' E'*. *n'* $\geq$ *n* $\wedge$ *n'* $\leq$ *n''* $\wedge$ *sbactive sid* (*t n'*) $\wedge$ *unsub sid E'* = *sbsb*(*sbcmp sid* (*t n'*))
$\wedge$ *e* $\in$ *E'*
and *e* $\in$ *E*
and (*e*,*m*) $\in$ *pbnt* (*pbcmp the-publisher* (*t n''*))
and *sbactive sid* (*t n''*)
shows (*e*,*m*) $\in$ *sbnt* (*sbcmp sid* (*t n''*)) **using** *assms conn2* **by** *simp*

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

lemma *conn1A*:
fixes *k*
shows *pbsb* (*pbcmp the-publisher k*) = ($\bigcup$ *sid*$\in${*sid*. *sbactive sid k*}. {*sbsb* (*sbcmp sid k*)})
using *conn1*[*OF pb.the-active*] **.**
end

end

3 A Theory of Blackboard Architectures

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

theory *Blackboard*
imports *Publisher-Subscriber*
begin

3.1 Problems and Solutions

Blackboards work with problems and solutions for them.

typedecl *PROB*
consts *sb* :: (*PROB* × *PROB*) *set*
axiomatization where *sbWF*: *wf sb*
typedecl *SOL*
consts *solve*:: *PROB* ⇒ *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* ⇒ (*'kid*, *PROB set*) *subscription set*
    and *bbcs* :: *'BB* ⇒ (*PROB* × *SOL*) *set*
    and *kscs* :: *'KS* ⇒ (*PROB* × *SOL*) *set*
    and *ksrp* :: *'KS* ⇒ (*'kid*, *PROB set*) *subscription* +
  fixes *bbns* :: *'BB* ⇒ (*PROB* × *SOL*) *set*
    and *ksns* :: *'KS* ⇒ (*PROB* × *SOL*) *set*
    and *bbop* :: *'BB* ⇒ *PROB set*
    and *ksop* :: *'KS* ⇒ *PROB set*
    and *prob* :: *'kid* ⇒ *PROB*
  assumes
    ks1: ∀ *p*. ∃ *ks*. *p*=*prob ks* — Component Parameter
    — Assertions about component behavior.
    and *bhvbb1*: ⋀*t t' bId p s*. ⟦*t* ∈ *arch*⟧ ⟹ *pb.eval bId t t' 0*
      (*pb.glob* (*pb.ass* (λ*bb*. (*p*,*s*)∈*bbns bb*)
      ⟶^*p* (*pb.evt* (*pb.ass* (λ*bb*. (*p*,*s*) ∈ *bbcs bb*)))))
    and *bhvbb2*: ⋀*t t' bId kId P q*. ⟦*t*∈*arch*⟧ ⟹ *pb.eval bId t t' 0*
      (*pb.glob* (*pb.ass* (λ*bb*. *sub kId P* ∈ *bbrp bb* ∧ *q* ∈ *P*) ⟶^*p*
      (*pb.evt* (*pb.ass* (λ*bb*. *q* ∈ *bbop bb*)))))
    and *bhvbb3*: ⋀*t t' bId p* . ⟦*t*∈*arch*⟧ ⟹ *pb.eval bId t t' 0*
      (*pb.glob* (*pb.ass* (λ*bb*. *p* ∈ *bbop*(*bb*)) ⟶^*p*
      (*pb.wuntil* (*pb.ass* (λ*bb*. *p*∈*bbop*(*bb*))) (*pb.ass* (λ*bb*. (*p*,*solve*(*p*))∈*bbcs*(*bb*))))))
    and *bhvks1*: ⋀*t t' kId p P*. ⟦*t*∈*arch*; *p* = *prob kId*⟧ ⟹ *sb.eval kId t t' 0*
      (*sb.glob* ((*sb.ass* (λ*ks*. *sub kId P* = *ksrp ks*)) ∧^*s*

$(sb.all\ (\lambda q.\ (sb.pred\ (q{\in}P)) \longrightarrow^s (sb.evt\ (sb.ass\ (\lambda ks.\ (q,solve(q)) \in kscs\ ks)))))$
$\longrightarrow^s (sb.evt\ (sb.ass\ (\lambda ks.\ (p,\ solve\ p) \in ksns\ ks)))))$
and *bhvks2*: $\bigwedge t\ t'\ kId\ p\ P\ q.\ [\![t \in arch; p = prob\ kId]\!] \Longrightarrow sb.eval\ kId\ t\ t'\ 0$
$(sb.glob\ (sb.ass\ (\lambda ks.\ sub\ kId\ P = ksrp\ ks \wedge q \in P \longrightarrow (q,p) \in sb)))$
and *bhvks3*: $\bigwedge t\ t'\ kId\ p.\ [\![t{\in}arch; p = prob\ kId]\!] \Longrightarrow sb.eval\ kId\ t\ t'\ 0$
$(sb.glob\ ((sb.ass\ (\lambda ks.\ p{\in}ksop\ ks)) \longrightarrow^s (sb.evt\ (sb.ass\ (\lambda ks.\ (\exists P.\ sub\ kId\ P = ksrp\ ks))))))$
and *bhvks4*: $\bigwedge t\ t'\ kId\ p\ P.\ [\![t{\in}arch;\ p{\in}P]\!] \Longrightarrow sb.eval\ kId\ t\ t'\ 0$
$(sb.glob\ ((sb.ass\ (\lambda ks.\ sub\ kId\ P = ksrp\ ks)) \longrightarrow^s$
$(sb.wuntil\ (\neg^s\ (\exists_s\ P'.\ (sb.pred\ (p{\in}P') \wedge^s (sb.ass\ (\lambda ks.\ unsub\ kId\ P' = ksrp\ ks)))))$
$(sb.ass\ (\lambda ks.\ (p,solve\ p) \in kscs\ ks)))))$

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

— Assertions about connections.
and *conn1*: $\bigwedge k\ bid.\ bbactive\ bid\ k$
$\Longrightarrow bbns\ (bbcmp\ bid\ k) = (\bigcup kid{\in}\{kid.\ ksactive\ kid\ k\}.\ ksns\ (kscmp\ kid\ k))$
and *conn2*: $\bigwedge k\ kid.\ ksactive\ kid\ k$
$\Longrightarrow ksop\ (kscmp\ kid\ k) = (\bigcup bid{\in}\{bid.\ bbactive\ bid\ k\}.\ bbop\ (bbcmp\ bid\ k))$
begin
notation *pb.lNAct* $(\langle\text{-} \leftarrow \text{-}\rangle\_)$
notation *pb.nxtAct* $(\langle\text{-} \rightarrow \text{-}\rangle\_)$

3.2.1 The Blackboard Component

In the following we introduce an abbreviation for the unique blackboard component.

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

3.2.2 Knowledge Sources

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

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

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

3.2.3 Verifying Blackboards

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.

theorem *pSolved*:
fixes t **and** t'::*nat* ⇒'*BB* **and** p **and** t''::*nat* ⇒'*KS*
assumes t∈*arch* **and**
$\forall n.\ \forall p{\in}bbop(bbcmp\ the\text{-}bb\ (t\ n)).$
$\exists n'{\geq}n.\ ksactive\ (sKs\ p)\ (t\ n')$
shows
$\forall n.\ p{\in}bbop(bbcmp\ the\text{-}bb\ (t\ n)) \longrightarrow$
$(\exists m{\geq}n.\ (p,solve(p)) \in bbcs\ (bbcmp\ the\text{-}bb\ (t\ m)))$
— The proof is by well-founded induction over the subproblem relation *sb*
proof (*rule wf-induct*[**where** r=*sb*])
— We first show that the subproblem relation is indeed well-founded ...
show *wf sb* **by** (*simp add*: *sbWF*)
next
— ... then we show that a problem p is indeed solved
— if all its sub-problems p' are eventually solved
fix p **assume** *indH*: $\forall p'.\ (p',\ p) \in sb \longrightarrow (\forall n.\ (p' \in bbop\ (bbcmp\ the\text{-}bb\ (t\ n))$
$\longrightarrow (\exists m{\geq}n.\ (p',solve(p')) \in bbcs\ (bbcmp\ the\text{-}bb\ (t\ m)))))$
show $\forall n.\ p \in bbop\ (bbcmp\ the\text{-}bb\ (t\ n))$
$\longrightarrow (\exists m{\geq}n.\ (p,solve(p)) \in bbcs\ (bbcmp\ the\text{-}bb\ (t\ m)))$
proof
fix n **show** $p \in bbop\ (bbcmp\ the\text{-}bb\ (t\ n)) \longrightarrow$
$(\exists m{\geq}n.\ (p,solve(p)) \in bbcs\ (bbcmp\ the\text{-}bb\ (t\ m)))$
proof
assume $p \in bbop\ (bbcmp\ the\text{-}bb\ (t\ n))$

— Problem p is provided at the output of the blackboard until it is solved
— or forever...
from *pb.globE*[*OF bhvbb3*] **have**
$pb.eval\ the\text{-}bb\ t\ t'\ n\ (pb.ass\ (\lambda\ bb.\ p \in bbop(bb)) \longrightarrow^p$
$(pb.wuntil\ (pb.ass\ (\lambda\ bb.\ p{\in}bbop(bb)))$
$(pb.ass\ (\lambda bb.\ (p,solve(p)){\in}bbcs(bb)))))$
using ‹t∈*arch*› **by** *auto*
moreover from ‹$p \in bbop\ (bbcmp\ the\text{-}bb\ (t\ n))$› **have**
$pb.eval\ the\text{-}bb\ t\ t'\ n\ (pb.ass\ (\lambda\ bb.\ p{\in}bbop\ bb))$
using ‹t∈*arch*› *pb.assI* **by** *simp*
ultimately have $pb.eval\ the\text{-}bb\ t\ t'\ n$
$(pb.wuntil\ (pb.ass\ (\lambda\ bb.\ p{\in}bbop(bb)))$
$(pb.ass\ (\lambda\ bb.\ (p,solve(p)){\in}bbcs(bb))))$
using *pb.impE* **by** *blast*
hence $pb.eval\ the\text{-}bb\ t\ t'\ n\ ((pb.until\ (pb.ass\ (\lambda\ bb.\ p{\in}bbop\ bb))$
$(pb.ass\ (\lambda\ bb.\ (p,solve(p)){\in}bbcs\ bb))) \vee^p (pb.glob\ (pb.ass\ (\lambda\ bb.\ p{\in}bbop\ bb))))$
using *pb.wuntil-def* **by** *simp*
hence $pb.eval\ the\text{-}bb\ t\ t'\ n$
$(pb.until\ (pb.ass\ (\lambda bb.\ p{\in}bbop\ bb))$
$(pb.ass\ (\lambda bb.\ (p,solve(p)){\in}bbcs\ bb))) \vee$
$(pb.eval\ the\text{-}bb\ t\ t'\ n\ (pb.glob\ (pb.ass\ (\lambda\ bb.\ p{\in}bbop\ bb))))$
using *pb.orE* **by** *simp*

thus $\exists\, m{\geq}n.\ (p,solve\ p) \in bbcs(bbcmp\ the\text{-}bb\ (t\ m))$
— We need to consider both cases, the case in which the problem is eventually
— solved and the case in which the problem is always provided as an output
proof
— First we consider the case in which the problem is eventually solved:
assume *pb.eval the-bb t t′ n*
$(pb.until\ (pb.ass\ (\lambda bb.\ p{\in}bbop\ bb))$
$(pb.ass\ (\lambda bb.\ (p,solve(p)){\in}bbcs\ bb)))$
hence $\exists\, i{\geq}n.\ (pb.eval\ the\text{-}bb\ t\ t'\ i$
$(pb.ass\ (\lambda bb.\ (p,solve(p)) \in bbcs\ bb)) \wedge$
$(\forall\, k{\geq}n.\ k{<}i \longrightarrow pb.eval\ the\text{-}bb\ t\ t'\ k\ (pb.ass\ (\lambda\ bb.\ p \in bbop\ bb))))$
using ‹*t∈arch*› *pb.untilE* **by** *simp*
then obtain *i* **where** $i{\geq}n$ **and**
pb.eval the-bb t t′ i $(pb.ass\ (\lambda bb.\ (p,solve(p)) \in bbcs\ bb))$ **by** *auto*
hence $(p,solve(p)){\in}bbcs(bbcmp\ the\text{-}bb\ (t\ i))$
using ‹*t∈arch*› *pb.assEA* **by** *auto*
thus *?thesis* **using** ‹$i{\geq}n$› **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*
$(pb.glob\ (pb.ass\ (\lambda bb.\ p{\in}bbop\ bb)))$
hence $\forall\, n'{\geq}n.\ (pb.eval\ the\text{-}bb\ t\ t'\ n'\ (pb.ass\ (\lambda bb.\ p \in bbop\ bb)))$
using ‹*t∈arch*› *pb.globE* **by** *auto*
hence *outp*: $\forall\, n'{\geq}n.\ (p \in bbop\ (bbcmp\ the\text{-}bb\ (t\ n')))$
using ‹*t∈arch*› *pb.assE* **by** *blast*

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

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

$\wedge\ (\forall\, n''{\geq}n_k.\ n'' < n' \longrightarrow \|sKs\ p\|_{t\ n''})$
— it is forwarded to the blackboard
then obtain n_s **where** $n_s{\geq}n_k$ **and** $\|sKs\ p\|_{t\ n_s}$
and $(p,\ solve\ p) \in ksns\ (\sigma_{sKs\ p} t\ n_s)$ **by** *auto*
moreover have $sb.nxtAct\ (sKs\ p)\ t\ n_s = n_s$
by (*simp add*: ‹$\|sKs\ p\|_{t\ n_s}$› *sb.nxtAct-active*)
ultimately have
$(p,solve(p)) \in bbns\ (bbcmp\ the\text{-}bb\ (t\ (sb.nxtAct\ (sKs\ p)\ t\ n_s)))$
using *conn1*[*OF pb.the-active*] ‹$\|sKs\ p\|_{t\ n_s}$› **by** *auto*

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

— first, the KS will eventually register for the subproblems P it requires to solve p...
from ‹$ksactive\ (sKs\ p)\ (t\ n_k)$› **have** $\exists\, i{\geq}0.\ ksactive\ (sKs\ p)\ (t\ i)$ **by** *auto*
moreover have $sb.lNAct\ (sKs\ p)\ t\ 0 \leq n_k$ **by** *simp*
ultimately have $sb.eval\ (sKs\ p)\ t\ t''\ n_k$
$((sb.ass\ (\lambda ks.\ p{\in}ksop\ ks)) \longrightarrow^s$
$(sb.evt\ (sb.ass\ (\lambda ks.\ \exists\, P.\ sub\ (sKs\ p)\ P = ksrp\ ks))))$
using *sb.globEA*[*OF - bhvks3*[*of t p sKs p t''*]] ‹$t{\in}arch$› *sks-prob* **by** *simp*
moreover have $sb.eval\ (sKs\ p)\ t\ t''\ n_k\ (sb.ass\ (\lambda ks.\ p \in ksop\ ks))$
proof −
from ‹$ksactive\ (sKs\ p)\ (t\ n_k)$› **have** $\exists\, n'{\geq}n_k.\ ksactive\ (sKs\ p)\ (t\ n')$ **by** *auto*
moreover have $p \in ksop\ (kscmp\ (sKs\ p)\ (t\ (sb.nxtAct\ (sKs\ p)\ t\ n_k)))$
proof −
from ‹$ksactive\ (sKs\ p)\ (t\ n_k)$› **have** $sb.nxtAct\ (sKs\ p)\ t\ n_k{=}n_k$
using *sb.nxtAct-active* **by** *blast*
with ‹$p{\in}ksop(kscmp\ (sKs\ p)\ (t\ n_k))$› **show** *?thesis* **by** *simp*
qed
ultimately show *?thesis* **using** *sb.assIA*[*of* n_k *sKs p t*] **by** *blast*
qed

ultimately have $sb.eval\ (sKs\ p)\ t\ t''\ n_k\ (sb.evt\ (sb.ass\ (\lambda ks.\ \exists P.\ sub\ (sKs\ p)\ P = ksrp\ ks)))$
using *sb.impE* **by** *blast*
then obtain n_r **where** $n_r \geq sb.nxtAct\ (sKs\ p)\ t\ n_k$ **and**
$\exists i \geq n_r.\ ksactive\ (sKs\ p)\ (t\ i) \wedge$
$(\forall n'' \geq sb.lNAct\ (sKs\ p)\ t\ n_r.\ n'' \leq sb.nxtAct\ (sKs\ p)\ t\ n_r$
$\longrightarrow sb.eval\ (sKs\ p)\ t\ t''\ n''\ (sb.ass\ (\lambda ks.\ \exists P.\ sub\ (sKs\ p)\ P = ksrp\ ks))) \vee$
$\neg\ (\exists i \geq n_r.\ ksactive\ (sKs\ p)\ (t\ i)) \wedge$
$sb.eval\ (sKs\ p)\ t\ t''\ n_r\ (sb.ass\ (\lambda ks.\ \exists P.\ sub\ (sKs\ p)\ P = ksrp\ ks))$
using ‹$ksactive\ (sKs\ p)\ (t\ n_k)$› $sb.evtEA[of\ n_k\ sKs\ p\ t]$ **by** *blast*
moreover from *case-ass* **have** $sb.nxtAct\ (sKs\ p)\ t\ n_k \geq n_k$ **using** *sb.nxtActI* **by** *blast*
with ‹$n_r \geq sb.nxtAct\ (sKs\ p)\ t\ n_k$› **have** $n_r \geq n_k$ **by** *arith*
hence $\exists i \geq n_r.\ ksactive\ (sKs\ p)\ (t\ i)$ **using** *case-ass* **by** *auto*
hence $n_r \leq sb.nxtAct\ (sKs\ p)\ t\ n_r$ **using** *sb.nxtActLe* **by** *simp*
moreover have $n_r \geq sb.lNAct\ (sKs\ p)\ t\ n_r$ **by** *simp*
ultimately have
$sb.eval\ (sKs\ p)\ t\ t''\ n_r\ (sb.ass\ (\lambda ks.\ \exists P.\ sub\ (sKs\ p)\ P = ksrp\ ks))$ **by** *blast*
with ‹$\exists i \geq n_r.\ ksactive\ (sKs\ p)\ (t\ i)$› **obtain** P **where**
$sub\ (sKs\ p)\ P = ksrp\ (kscmp\ (sKs\ p)\ (t\ (sb.nxtAct\ (sKs\ p)\ t\ n_r)))$
using *sb.assEA* **by** *blast*
hence $sb.eval\ (sKs\ p)\ t\ t''\ n_r\ (sb.ass\ (\lambda ks.\ sub\ (sKs\ p)\ P = ksrp\ ks))$
using ‹$\exists i \geq n_r.\ ksactive\ (sKs\ p)\ (t\ i)$› *sb.assIA sks-prob* **by** *blast*

— the knowledgesource will eventually get a solution for each required subproblem:
moreover have $sb.eval\ (sKs\ p)\ t\ t''\ n_r\ (sb.all\ (\lambda\ p'.\ sb.pred\ (p' \in P) \longrightarrow^s$
$(sb.evt\ (sb.ass\ (\lambda ks.\ (p', solve\ p') \in kscs\ ks)))))$
proof −
have $\forall p'.\ sb.eval\ (sKs\ p)\ t\ t''\ n_r\ (sb.pred\ (p' \in P) \longrightarrow^s$
$(sb.evt\ (sb.ass\ (\lambda ks.\ (p', solve\ p') \in kscs\ ks))))$
proof
— by induction hypothesis, the blackboard will eventually provide solutions for subproblems

fix p'
have $sb.eval\ (sKs\ p)\ t\ t''\ n_r\ (sb.pred\ (p' \in P)) \longrightarrow$
$(sb.eval\ (sKs\ p)\ t\ t''\ n_r$
$(sb.evt\ (sb.ass\ (\lambda ks.\ (p', solve\ p') \in kscs\ ks))))$
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 $(sb.eval\ (sKs\ p)\ t\ t''\ n_r\ (sb.evt\ (sb.ass\ (\lambda ks.\ (p', solve\ p') \in kscs\ ks))))$
proof −
have $sb.lNAct\ (sKs\ p)\ t\ 0 \leq n_r$ **by** *simp*
moreover from ‹$ksactive\ (sKs\ p)\ (t\ n_k)$› **have** $\exists i \geq 0.\ ksactive\ (sKs\ p)\ (t\ i)$ **by** *auto*
ultimately have $sb.eval\ (sKs\ p)\ t\ t''\ n_r\ ((sb.ass\ (\lambda ks.\ sub\ (sKs\ p)\ P = ksrp\ ks))$
$\longrightarrow^s (sb.wuntil\ (\neg^s\ (\exists_s\ P'.\ (sb.pred\ (p' \in P') \wedge^s$
$(sb.ass\ (\lambda ks.\ unsub\ (sKs\ p)\ P' = ksrp\ ks)))))$
$(sb.ass\ (\lambda ks.\ (p', solve\ p') \in kscs\ ks))))$
using $sb.globEA[OF\ -\ bhvks4[of\ t\ p'\ P\ sKs\ p\ t'']]$
‹$t \in arch$› ‹$ksactive\ (sKs\ p)\ (t\ n_k)$› ‹$p' \in P$› **by** *simp*
with ‹$sb.eval\ (sKs\ p)\ t\ t''\ n_r\ (sb.ass\ (\lambda ks.\ sub\ (sKs\ p)\ P = ksrp\ ks))$› **have**

sb.eval (*sKs p*) *t t''* n_r (*sb.wuntil* ($\neg^s$ ($\exists_s$ *P'*. (*sb.pred* (*p'*$\in$*P'*) $\wedge^s$
(*sb.ass* (λ*ks*. *unsub* (*sKs p*) *P'* = *ksrp ks*))))) (*sb.ass* (λ*ks*. (*p'*,*solve p'*) $\in$ *kscs ks*)))
using *sb.impE*[*of* (*sKs p*) *t t''* n_r *sb.ass* (λ*ks*. *sub* (*sKs p*) *P* = *ksrp ks*)] **by** *blast*
hence *sb.eval* (*sKs p*) *t t''* n_r (*sb.until* ($\neg^s$ ($\exists_s$ *P'*. (*sb.pred* (*p'*$\in$*P'*) $\wedge^s$
(*sb.ass* (λ*ks*. *unsub* (*sKs p*) *P'* = *ksrp ks*))))) (*sb.ass* (λ*ks*. (*p'*,*solve p'*) $\in$ *kscs ks*))) $\vee$
sb.eval (*sKs p*) *t t''* n_r (*sb.glob* ($\neg^s$ ($\exists_s$ *P'*. (*sb.pred* (*p'*$\in$*P'*) $\wedge^s$
sb.ass (λ*ks*. *unsub* (*sKs p*) *P'* = *ksrp ks*))))) **using** *sb.wuntil-def* **by** *auto*
thus (*sb.eval* (*sKs p*) *t t''* n_r (*sb.evt* (*sb.ass* (λ*ks*. (*p'*,*solve p'*) $\in$ *kscs ks*))))
proof
let *?*γ'=$\neg^s$ ($\exists_s$ *P'*. (*sb.pred* (*p'*$\in$*P'*) $\wedge^s$ (*sb.ass* (λ*ks*. *unsub* (*sKs p*) *P'* = *ksrp ks*))))
let *?*γ=*sb.ass* (λ*ks*. (*p'*,*solve p'*) $\in$ *kscs ks*)
assume *sb.eval* (*sKs p*) *t t''* n_r (*sb.until ?*γ' *?*γ)
with ‹$\exists\, i{\geq}n_r$. $\|$*sKs p*$\|_{t\ i}$› **obtain** *n'* **where** *n'*$\geq$*sb.nxtAct* (*sKs p*) *t* n_r **and**
lass: ($\exists\, i{\geq}$*n'*. $\|$*sKs p*$\|_{t\ i}$) $\wedge$ ($\forall$ *n''*$\geq$*sb.lNAct* (*sKs p*) *t n'*. *n''* $\leq$ *sb.nxtAct* (*sKs p*) *t n'*
$\longrightarrow$ *sb.eval* (*sKs p*) *t t''* *n''* *?*γ) $\wedge$
($\forall$ *n''*$\geq$*sb.lNAct* (*sKs p*) *t* n_r. *n''* < *sb.lNAct* (*sKs p*) *t n'*
$\longrightarrow$ *sb.eval* (*sKs p*) *t t''* *n''* *?*γ') $\vee$
$\neg$ ($\exists\, i{\geq}$*n'*. $\|$*sKs p*$\|_{t\ i}$) $\wedge$ *sb.eval* (*sKs p*) *t t''* *n'* *?*γ $\wedge$
($\forall$ *n''*$\geq$*sb.lNAct* (*sKs p*) *t* n_r. *n''* < *n'* $\longrightarrow$ *sb.eval* (*sKs p*) *t t''* *n''* *?*γ')
using *sb.untilEA*[*of* n_r *sKs p t t''*] ‹$\exists\, i{\geq}n_r$. *ksactive* (*sKs p*) (*t i*)› **by** *blast*
thus *?thesis*
proof *cases*
assume $\exists\, i{\geq}$*n'*. $\|$*sKs p*$\|_{t\ i}$
with *lass* **have** $\forall$ *n''*$\geq$*sb.lNAct* (*sKs p*) *t n'*. *n''* $\leq$ *sb.nxtAct* (*sKs p*) *t n'*
$\longrightarrow$ *sb.eval* (*sKs p*) *t t''* *n''* *?*γ **by** *auto*
moreover have *n'*$\geq$*sb.lNAct* (*sKs p*) *t n'* **by** *simp*
moreover have *n'* $\leq$ *sb.nxtAct* (*sKs p*) *t n'*
using ‹$\exists\, i{\geq}$*n'*. $\|$*sKs p*$\|_{t\ i}$› *sb.nxtActLe* **by** *simp*
ultimately have *sb.eval* (*sKs p*) *t t''* *n'* *?*γ **by** *simp*
moreover have *sb.lNAct* (*sKs p*) *t* n_r $\leq$ *n'* **using** ‹n_r $\leq$ *sb.nxtAct* (*sKs p*) *t* n_r›
‹*sb.lNAct* (*sKs p*) *t* n_r $\leq$ n_r› ‹*sb.nxtAct* (*sKs p*) *t* n_r $\leq$ *n'*› **by** *linarith*
ultimately show *?thesis* **using** ‹$\exists\, i{\geq}n_r$. $\|$*sKs p*$\|_{t\ i}$› ‹$\exists\, i{\geq}$*n'*. $\|$*sKs p*$\|_{t\ i}$›
‹*n'*$\geq$*sb.lNAct* (*sKs p*) *t n'*› ‹*n'* $\leq$ *sb.nxtAct* (*sKs p*) *t n'*›
sb.evtIA[*of* n_r *sKs p t n' t''* *?*γ] **by** *blast*
next
assume $\neg$ ($\exists\, i{\geq}$*n'*. $\|$*sKs p*$\|_{t\ i}$)
with *lass* **have** *sb.eval* (*sKs p*) *t t''* *n'* *?*γ $\wedge$
($\forall$ *n''*$\geq$*sb.lNAct* (*sKs p*) *t* n_r. *n''* < *n'* $\longrightarrow$ *sb.eval* (*sKs p*) *t t''* *n''* *?*γ') **by** *auto*
moreover have *sb.lNAct* (*sKs p*) *t* n_r $\leq$ *n'*
using ‹n_r $\leq$ *sb.nxtAct* (*sKs p*) *t* n_r› ‹*sb.lNAct* (*sKs p*) *t* n_r $\leq$ n_r›
‹*sb.nxtAct* (*sKs p*) *t* n_r $\leq$ *n'*› **by** *linarith*
ultimately show *?thesis* **using** ‹$\exists\, i{\geq}n_r$. $\|$*sKs p*$\|_{t\ i}$› ‹$\neg$ ($\exists\, i{\geq}$*n'*. $\|$*sKs p*$\|_{t\ i}$)›
sb.evtIA[*of* n_r *sKs p t n' t''* *?*γ] **by** *blast*
qed
next
assume *cass*: *sb.eval* (*sKs p*) *t t''* n_r
(*sb.glob* ($\neg^s$ ($\exists_s$ *P'*. (*sb.pred* (*p'*$\in$*P'*) $\wedge^s$ *sb.ass* (λ*ks*. *unsub* (*sKs p*) *P'* = *ksrp ks*)))))

have *sub* (*sKs p*) *P* = *ksrp* (*kscmp* (*sKs p*) (*t* (*sb.nxtAct* (*sKs p*) *t* n_r))) $\wedge$

$p' \in P \longrightarrow (p', p) \in sb$
proof −
have $\exists\, i{\geq}0.\ ksactive\ (sKs\ p)\ (t\ i)$ **using** ‹$\exists\, i{\geq}0.\ ksactive\ (sKs\ p)\ (t\ i)$› **by** *auto*
moreover have $sb.lNAct\ (sKs\ p)\ t\ 0 \leq (sb.nxtAct\ (sKs\ p)\ t\ n_r)$ **by** *simp*
ultimately have $sb.eval\ (sKs\ p)\ t\ t''\ (sb.nxtAct\ (sKs\ p)\ t\ n_r)$
$(sb.ass\ (\lambda ks.\ sub\ (sKs\ p)\ P = ksrp\ ks \wedge p' \in P \longrightarrow (p', p) \in sb))$
using *sb.globEA*[*OF - bhvks2*[*of t p sKs p t'' P*]] ‹$t \in arch$› *sks-prob* **by** *blast*
moreover from ‹$\exists\, i{\geq}n_r.\ ksactive\ (sKs\ p)\ (t\ i)$› **have**
$ksactive\ (sKs\ p)\ (t\ (sb.nxtAct\ (sKs\ p)\ t\ n_r))$ **using** *sb.nxtActI* **by** *blast*
ultimately show *?thesis*
using $sb.assEANow[of\ sKs\ p\ t\ t''\ sb.nxtAct\ (sKs\ p)\ t\ n_r]$ **by** *simp*
qed
with ‹$p' \in P$› **have** $(p', p) \in sb$
using ‹$sub\ (sKs\ p)\ P = ksrp\ (kscmp\ (sKs\ p)\ (t\ (sb.nxtAct\ (sKs\ p)\ t\ n_r)))$›
sks-prob **by** *simp*
moreover have $\exists\, n_p{\geq}(sb.nxtAct\ (sKs\ p)\ t\ n_r).$
$pb.eval\ the\text{-}bb\ t\ t'\ n_p\ (pb.ass\ (\lambda bb.\ p' \in bbop\ bb))$
proof −
from *pb.globE*[*OF bhvbb2*[*of t the-bb t'*]]
have $pb.eval\ the\text{-}bb\ t\ t'\ (sb.nxtAct\ (sKs\ p)\ t\ n_r)$
$(pb.ass\ (\lambda bb.\ sub\ (sKs\ p)\ P \in bbrp\ bb \wedge p' \in P) \longrightarrow^p$
$(pb.evt\ (pb.ass\ (\lambda bb.\ p' \in bbop\ bb))))$ **using** ‹$t \in arch$› **by** *auto*
moreover from ‹$\exists\, i{\geq}n_r.\ ksactive\ (sKs\ p)\ (t\ i)$› **have**
$ksactive\ (sKs\ p)\ (t\ (sb.nxtAct\ (sKs\ p)\ t\ n_r))$ **using** *sb.nxtActI* **by** *blast*
with ‹$sub\ (sKs\ p)\ P = ksrp\ (kscmp\ (sKs\ p)\ (t\ (sb.nxtAct\ (sKs\ p)\ t\ n_r)))$›
have $sub\ (sKs\ p)\ P \in bbrp\ (bbcmp\ the\text{-}bb\ (t\ (sb.nxtAct\ (sKs\ p)\ t\ n_r)))$
using *conn1A* **by** *auto*
with ‹$p' \in P$› **have** $pb.eval\ the\text{-}bb\ t\ t'\ (sb.nxtAct\ (sKs\ p)\ t\ n_r)$
$(pb.ass\ (\lambda bb.\ sub\ (sKs\ p)\ P \in bbrp\ bb \wedge p' \in P))$ **using** ‹$t \in arch$›
pb.assIA[**where** $\varphi{=}\lambda bb.\ sub\ (sKs\ p)\ P \in bbrp\ bb \wedge p' \in P$] **by** *blast*
ultimately have $pb.eval\ the\text{-}bb\ t\ t'\ (sb.nxtAct\ (sKs\ p)\ t\ n_r)$
$(pb.evt\ (pb.ass\ (\lambda bb.\ p' \in bbop\ bb)))$ **using** *pb.impE* ‹$p' \in P$›
by *blast*
with ‹$p' \in P$› **have** $pb.eval\ the\text{-}bb\ t\ t'\ (sb.nxtAct\ (sKs\ p)\ t\ n_r)$
$(pb.evt\ (pb.ass\ (\lambda bb.\ p' \in bbop\ bb)))$ **by** *simp*
thus *?thesis* **using** ‹$t \in arch$› $pb.evtE[of\ t\ t'\ sb.nxtAct\ (sKs\ p)\ t\ n_r]$
by *simp*
qed
then obtain n_p **where** $n_p \geq sb.nxtAct\ (sKs\ p)\ t\ n_r$ **and**
$pb.eval\ the\text{-}bb\ t\ t'\ n_p\ (pb.ass\ (\lambda bb.\ p' \in bbop\ bb))$ **by** *auto*
hence $p' \in bbop\ (bbcmp\ the\text{-}bb\ (t\ n_p))$ **using** ‹$t \in arch$› *pb.assEA* **by** *auto*
ultimately obtain m **where** $m{\geq}n_p$ **and** $(p', solve\ p') \in bbcs\ (bbcmp\ the\text{-}bb\ (t\ m))$
using *indH* **by** *auto*

— and due to the publisher subscriber property,
— the knowledge source will receive them
moreover have
$\nexists\, n\ P.\ sb.nxtAct\ (sKs\ p)\ t\ n_r \leq n \wedge n \leq m \wedge ksactive\ (sKs\ p)\ (t\ n) \wedge$
$unsub\ (sKs\ p)\ P = ksrp\ (kscmp\ (sKs\ p)\ (t\ n)) \wedge p' \in P$

proof
assume $\exists\, n\ P'.\ sb.nxtAct\ (sKs\ p)\ t\ n_r \leq n \wedge n \leq m \wedge ksactive\ (sKs\ p)\ (t\ n) \wedge$
$unsub\ (sKs\ p)\ P' = ksrp\ (kscmp\ (sKs\ p)\ (t\ n)) \wedge p' \in P'$
then obtain $n\ P'$ **where**
$ksactive\ (sKs\ p)\ (t\ n)$ **and** $sb.nxtAct\ (sKs\ p)\ t\ n_r \leq n$ **and** $n \leq m$ **and**
$unsub\ (sKs\ p)\ P' = ksrp\ (kscmp\ (sKs\ p)\ (t\ n))$ **and** $p' \in P'$ **by** *auto*
hence $sb.eval\ (sKs\ p)\ t\ t''\ n\ (\exists_s\ P'.\ sb.pred\ (p' \in P') \wedge^s$
$sb.ass\ (\lambda ks.\ unsub\ (sKs\ p)\ P' = ksrp\ ks))$ **by** *blast*
moreover have $sb.lNAct\ (sKs\ p)\ t\ n_r \leq n$
using $\langle n_r \leq sb.nxtAct\ (sKs\ p)\ t\ n_r\rangle\ \langle sb.lNAct\ (sKs\ p)\ t\ n_r \leq n_r\rangle$
$\langle sb.nxtAct\ (sKs\ p)\ t\ n_r \leq n\rangle$ **by** *linarith*
with *cass* **have** $sb.eval\ (sKs\ p)\ t\ t''\ n\ (\neg^s\ (\exists_s\ P'.\ (sb.pred\ (p' \in P')$
$\wedge^s\ sb.ass\ (\lambda ks.\ unsub\ (sKs\ p)\ P' = ksrp\ ks))))$
using $sb.globEA[of\ n_r\ sKs\ p\ t\ t''$
$\neg^s\ (\exists_s P'.\ sb.pred\ (p' \in P') \wedge^s\ sb.ass\ (\lambda ks.\ unsub\ (sKs\ p)\ P' = ksrp\ ks))\ n]$
$\langle \exists\, i \geq n_r.\ ksactive\ (sKs\ p)\ (t\ i)\rangle$ **by** *auto*
ultimately show *False* **using** *sb.notE* **by** *auto*
qed
moreover from $\langle \exists\, i \geq n_r.\ ksactive\ (sKs\ p)\ (t\ i)\rangle$ **have**
$ksactive\ (sKs\ p)\ (t\ (sb.nxtAct\ (sKs\ p)\ t\ n_r))$ **using** *sb.nxtActI* **by** *blast*
moreover have $sub\ (sKs\ p)\ P = ksrp\ (kscmp\ (sKs\ p)\ (t\ (sb.nxtAct\ (sKs\ p)\ t\ n_r)))$
using $\langle sub\ (sKs\ p)\ P = ksrp\ (kscmp\ (sKs\ p)\ (t\ (sb.nxtAct\ (sKs\ p)\ t\ n_r)))\rangle$ **.**
moreover from $\langle m \geq n_p\rangle\ \langle n_p \geq sb.nxtAct\ (sKs\ p)\ t\ n_r\rangle$
have $sb.nxtAct\ (sKs\ p)\ t\ n_r \leq m$ **by** *simp*
moreover from $\langle \exists\, i \geq n_r.\ ksactive\ (sKs\ p)\ (t\ i)\rangle$
have $sb.nxtAct\ (sKs\ p)\ t\ n_r \geq n_r$ **using** *sb.nxtActI* **by** *blast*
hence $m \geq n_k$ **using** $\langle sb.nxtAct\ (sKs\ p)\ t\ n_r \leq m\rangle\ \langle sb.nxtAct\ (sKs\ p)\ t\ n_k \leq n_r\rangle$
$\langle sb.nxtAct\ (sKs\ p)\ t\ n_k \geq n_k\rangle$ **by** *simp*
with *case-ass* **have** $ksactive\ (sKs\ p)\ (t\ m)$ **by** *simp*
ultimately have $(p',\ solve\ p') \in kscs\ (kscmp\ (sKs\ p)\ (t\ m))$
and $ksactive\ (sKs\ p)\ (t\ m)$
using $\langle t \in arch\rangle\ msgDelivery[of\ t\ sKs\ p\ sb.nxtAct\ (sKs\ p)\ t\ n_r\ P\ m\ p'\ solve\ p']$
$\langle p' \in P\rangle$ **by** *auto*
hence $sb.eval\ (sKs\ p)\ t\ t''\ m\ (sb.ass\ (\lambda ks.\ (p',solve\ p') \in kscs\ ks))$
using *sb.assIANow* **by** *simp*
moreover have $m \geq sb.lNAct\ (sKs\ p)\ t\ m$ **by** *simp*
moreover from $\langle ksactive\ (sKs\ p)\ (t\ m)\rangle$ **have** $m \leq sb.nxtAct\ (sKs\ p)\ t\ m$
using *sb.nxtActLe* **by** *auto*
moreover from $\langle \exists\, i \geq n_r.\ ksactive\ (sKs\ p)\ (t\ i)\rangle$ **have**
$sb.lNAct\ (sKs\ p)\ t\ n_r \leq sb.nxtAct\ (sKs\ p)\ t\ n_r$ **by** *simp*
with $\langle sb.nxtAct\ (sKs\ p)\ t\ n_r \leq m\rangle$ **have** $sb.lNAct\ (sKs\ p)\ t\ n_r \leq m$ **by** *arith*
ultimately show $sb.eval\ (sKs\ p)\ t\ t''\ n_r$
$(sb.evt\ (sb.ass\ (\lambda ks.\ (p',solve\ p') \in kscs\ ks)))$
using $\langle \exists\, i \geq n_r.\ ksactive\ (sKs\ p)\ (t\ i)\rangle\ sb.evtIA$ **by** *blast*
qed
qed
qed
thus $sb.eval\ (sKs\ p)\ t\ t''\ n_r\ (sb.pred\ (p' \in P) \longrightarrow^s$
$(sb.evt\ (sb.ass\ (\lambda ks.\ (p',solve\ p') \in kscs\ ks))))$

using *sb.impI* **by** *auto*
qed
thus *?thesis* **using** *sb.allI* **by** *blast*
qed

— Thus, the knowlege source will eventually solve the problem at hand...
ultimately have *sb.eval* (*sKs p*) *t t''* n_r
(*sb.ass* (λ*ks*. *sub* (*sKs p*) *P* = *ksrp ks*) ∧^{s}
(∀$_s$*q*. (*sb.pred* (*q* ∈ *P*) ⟶^{s} *sb.evt* (*sb.ass* (λ*ks*. (*q*, *solve q*) ∈ *kscs ks*)))))
using *sb.andI* **by** *simp*
moreover from ‹∃ *i*≥n_r. *ksactive* (*sKs p*) (*t i*)› **have** ∃ *i*≥*0*. *ksactive* (*sKs p*) (*t i*) **by** *blast*
hence *sb.eval* (*sKs p*) *t t''* n_r
((*sb.ass* (λ*ks*. *sub* (*sKs p*) *P* = *ksrp ks*) ∧^{s}
(∀$_s$*q*. (*sb.pred* (*q* ∈ *P*) ⟶^{s}
sb.evt (*sb.ass* (λ*ks*. (*q*, *solve q*) ∈ *kscs ks*))))) ⟶^{s}
(*sb.evt* (*sb.ass* (λ*ks*. (*p*, *solve p*) ∈ *ksns ks*)))) **using** ‹*t* ∈ *arch*›
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
(*sb.evt* (*sb.ass* (λ*ks*. (*p*,*solve*(*p*))∈*ksns*(*ks*))))
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≥*sb.nxtAct* (*sKs p*) *t* n_r **and**
(∃ *i*≥n_s. *ksactive* (*sKs p*) (*t i*) ∧
(∀ *n''*≥*sb.lNAct* (*sKs p*) *t* n_s. *n''* ≤ *sb.nxtAct* (*sKs p*) *t* n_s ⟶
sb.eval (*sKs p*) *t t''* *n''* (*sb.ass* (λ*ks*. (*p*,*solve*(*p*))∈*ksns*(*ks*))))) ∨
¬ (∃ *i*≥n_s. *ksactive* (*sKs p*) (*t i*)) ∧
sb.eval (*sKs p*) *t t''* n_s (*sb.ass* (λ*ks*. (*p*,*solve*(*p*))∈*ksns*(*ks*)))
using *sb.evtEA*[*of* n_r *sKs p t*] ‹∃ *i*≥n_r. *ksactive* (*sKs p*) (*t i*)› **by** *blast*
moreover from ‹*sb.nxtAct* (*sKs p*) *t* n_r ≥ n_r› ‹n_r≥n_k› ‹n_s≥*sb.nxtAct* (*sKs p*) *t* n_r›
have n_s≥n_k **by** *arith*
with *case-ass* **have** ∃ *i*≥n_s. *ksactive* (*sKs p*) (*t i*) **by** *auto*
moreover have n_s≥*sb.lNAct* (*sKs p*) *t* n_s **by** *simp*
moreover from ‹∃ *i*≥n_s. *ksactive* (*sKs p*) (*t i*)› **have** n_s ≤ *sb.nxtAct* (*sKs p*) *t* n_s
using *sb.nxtActLe* **by** *simp*
ultimately have *sb.eval* (*sKs p*) *t t''* n_s (*sb.ass* (λ*ks*. (*p*,*solve*(*p*))∈*ksns*(*ks*)))
using *sb.evtEA*[*of* n_r *sKs p t*] ‹∃ *i*≥n_r. *ksactive* (*sKs p*) (*t i*)› **by** *blast*
with ‹∃ *i*≥n_s. *ksactive* (*sKs p*) (*t i*)› **have**
(*p*,*solve*(*p*)) ∈ *ksns* (*kscmp* (*sKs p*) (*t* (*sb.nxtAct* (*sKs p*) *t* n_s)))
using *sb.assEA*[*of* n_s *sKs p t t'' λks*. (*p*, *solve p*) ∈ *ksns ks*] **by** *auto*
moreover from ‹∃ *i*≥n_s. *ksactive* (*sKs p*) (*t i*)›
have *ksactive* (*sKs p*) (*t* (*sb.nxtAct* (*sKs p*) *t* n_s)) **using** *sb.nxtActI* **by** *simp*
ultimately have (*p*,*solve*(*p*)) ∈ *bbns* (*bbcmp the-bb* (*t* (*sb.nxtAct* (*sKs p*) *t* n_s)))
using *conn1*[*OF pb.the-active*[*of t* (*sb.nxtAct* (*sKs p*) *t* n_s)]] **by** *auto*
hence *pb.eval the-bb t t'*
(*sb.nxtAct* (*sKs p*) *t* n_s) (*pb.ass* (λ*bb*. (*p*,*solve*(*p*)) ∈ *bbns bb*))
using ‹*t*∈*arch*› *pb.assI* **by** *simp*

— finally, the blackboard will forward the solution which finishes the proof.

```
        with bhvbb1 have pb.eval the-bb t t' (sb.nxtAct (sKs p) t n_s)
          (pb.evt (pb.ass (λbb. (p, solve p) ∈ bbcs bb)))
          using ⟨t∈arch⟩ pb.globE pb.impE[of the-bb t t'] by blast
        then obtain n_f where n_f≥sb.nxtAct (sKs p) t n_s and
          pb.eval the-bb t t' n_f (pb.ass (λbb. (p, solve p) ∈ bbcs bb))
          using ⟨t∈arch⟩ pb.evtE[of t t' sb.nxtAct (sKs p) t n_s] by auto
        hence (p, solve p) ∈ bbcs (bbcmp the-bb (t n_f))
          using ⟨t ∈ arch⟩ pb.assEA by auto
        moreover have n_f≥n
        proof −
          from ⟨∃ n'''≥n_s. ksactive (sKs p) (t n''')⟩ have sb.nxtAct (sKs p) t n_s≥n_s
            using sb.nxtActLe by simp
          moreover from ⟨n_k≥n⟩ and ⟨ksactive (sKs p) (t n_k)⟩ have sb.nxtAct (sKs p) t n_k≥n_k
            using sb.nxtActI by blast
          ultimately show ?thesis
            using ⟨n_f≥sb.nxtAct (sKs p) t n_s⟩ ⟨n_s≥sb.nxtAct (sKs p) t n_r⟩
            ⟨sb.nxtAct (sKs p) t n_r≥n_r⟩ ⟨n_r≥sb.nxtAct (sKs p) t n_k⟩ ⟨n_k≥n⟩ by arith
        qed
        ultimately show ?thesis by auto
      qed
    qed
  qed
 qed
end

end
```

4 A Theory of Blockchain Architectures

```
theory Blockchain imports DynamicArchitectures.Dynamic-Architecture-Calculus
begin
```

4.1 Additions for Dynamic Components

These additions should go to theory Configuration_Traces for the next version of the AFP.

```
context dynamic-component
begin

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

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

```
    fix n
    assume a2: ∀ m<n. i ≤ m ⟶ P m
    show i ≤ n ⟹ P n
    proof −
      assume i ≤ n
      with a1 have (∀ m ≥ i.  m<n ⟶ P m) ⟹ P n by simp
      moreover from a2 have ∀ m ≥ i.  m<n ⟶ P m by simp
      ultimately show P n by simp
    qed
  qed
  with nat-less-induct[of λj. i ≤ j ⟶ P j j] have i ≤ j ⟶ P j .
  with a0 show ?thesis by simp
qed

lemma nxtAct-eq:
  assumes n'≥n
    and ‖c‖t n'
    and ∀ n''≥n. n''<n' ⟶ ¬ ‖c‖t n''
  shows n' = ⟨c → t⟩n
  by (metis assms(1) assms(2) assms(3) dynamic-component.nxtActI linorder-neqE-nat nxtActLe)

lemma globEANow:
  fixes c t t' n i γ
  assumes n ≤ i
    and ‖c‖t i
    and eval c t t' n (□γ)
  shows eval c t t' i γ
proof −
  from ‹‖c‖t i› ‹n ≤ i› have ∃ i≥n. ‖c‖t i by auto
  moreover from ‹n ≤ i› have ⟨c ← t⟩n ≤ i using dual-order.trans lNactLe by blast
  ultimately show ?thesis using globEA[of n c t t' γ i] ‹eval c t t' n (□γ)› by simp
qed

abbreviation lastAct-cond:: 'id ⇒ trace ⇒ nat ⇒ nat ⇒ bool
  where lastAct-cond c t n n' ≡ n'<n ∧ ‖c‖t n'

definition lastAct:: 'id ⇒ trace ⇒ nat ⇒ nat (⟨- ⇐ -⟩-)
  where lastAct c t n = (GREATEST n'. lastAct-cond c t n n')

lemma lastActEx:
  assumes ∃ n'<n. ‖nid‖t n'
  shows ∃ n'. lastAct-cond nid t n n' ∧ (∀ n''. lastAct-cond nid t n n'' ⟶ n'' ≤ n')
proof −
  from assms obtain n' where lastAct-cond nid t n n' by auto
  moreover have ∀ n''>n. ¬ lastAct-cond nid t n n'' by simp
  ultimately obtain n' where lastAct-cond nid t n n' ∧ (∀ n''. lastAct-cond nid t n n'' ⟶ n'' ≤ n')
using boundedGreatest[of lastAct-cond nid t n n'] by blast
  thus ?thesis ..
qed
```

lemma *lastAct-prop*:
assumes $\exists\, n'{<}n.\ \|nid\|_{t\ n'}$
shows $\|nid\|_{t\ (lastAct\ nid\ t\ n)}$ **and** *lastAct nid t* $n{<}n$
proof −
from *assms lastActEx* **have** *lastAct-cond nid t n* (*GREATEST x. lastAct-cond nid t n x*) **using** *GreatestI-ex-nat*[*of lastAct-cond nid t n*] **by** *blast*
thus $\|nid\|_{t\ \langle nid \Leftarrow t\rangle n}$ **and** *lastAct nid t* $n{<}n$ **using** *lastAct-def* **by** *auto*
qed

lemma *lastAct-less*:
assumes *lastAct-cond nid t n n'*
shows $n' \leq \langle nid \Leftarrow t\rangle n$
proof −
from *assms lastActEx* **have** $n' \leq$ (*GREATEST x. lastAct-cond nid t n x*) **using** *Greatest-le-nat*[*of lastAct-cond nid t n*] **by** *blast*
thus *?thesis* **using** *lastAct-def* **by** *auto*
qed

lemma *lastActNxt*:
assumes $\exists\, n'{<}n.\ \|nid\|_{t\ n'}$
shows $\langle nid \rightarrow t\rangle_{\langle nid \Leftarrow t\rangle n}{=}\langle nid \Leftarrow t\rangle n$
using *assms lastAct-prop*(*1*) *nxtAct-active* **by** *auto*

lemma *lastActNxtAct*:
assumes $\exists\, n'{\geq}n.\ \|tid\|_{t\ n'}$
and $\exists\, n'{<}n.\ \|tid\|_{t\ n'}$
shows $\langle tid \rightarrow t\rangle n > \langle tid \Leftarrow t\rangle n$
by (*meson assms lastAct-prop*(*2*) *less-le-trans nxtActI zero-le*)

lemma *lastActless*:
assumes $\exists\, n'{\geq}n_S.\ n'{<}n \wedge \|nid\|_{t\ n'}$
shows $\langle nid \Leftarrow t\rangle n{\geq}n_S$
by (*meson assms dual-order.trans lastAct-less*)

end

4.2 Blockchains

A blockchain itself is modeled as a simple list.

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

abbreviation *max-cond*:: (*'a BC*) *set* ⇒ *'a BC* ⇒ *bool*
where *max-cond B b* ≡ $b \in B \wedge (\forall\, b'{\in}B.\ length\ b' \leq length\ b)$

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

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

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

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

with ‹$b \in B$› **have** *length* $b \leq$ *length* (*MAX* B) **by** *simp*
with ‹*length* $b >$ *length* b'› **show** *?thesis* **by** *simp*
qed

4.3 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* :: *trace* $\Rightarrow$ *nat* $\Rightarrow$ $'nid$ *set*
and *actUt* :: *trace* $\Rightarrow$ *nat* $\Rightarrow$ $'nid$ *set*
and *PoW*:: *trace* $\Rightarrow$ *nat* $\Rightarrow$ *nat*
and *trNxt*:: *trace* $\Rightarrow$ *nat* $\Rightarrow$ *bool*
defines *actTr* $t\ n \equiv \{nid.\ \|nid\|_{t\ n} \wedge trusted\ nid\}$
and *actUt* $t\ n \equiv \{nid.\ \|nid\|_{t\ n} \wedge \neg\ trusted\ nid\}$
and *PoW* $t\ n \equiv$ (*LEAST* $x.\ \forall nid \in actTr\ t\ n.\ length\ (bc\ (\sigma_{nid}(t\ n))) \leq x$)
and *trNxt* $t\ n \equiv (\exists n' \geq n.\ PoW\ t\ n' > PoW\ t\ n \wedge (\forall n'' > n.\ n'' \leq n' \longrightarrow \neg(\exists nid \in actUt\ t\ n''.\ \|nid\|_{t\ n''} \wedge mining\ (\sigma_{nid}(t\ n'')))))\ \vee\ (\forall n' > n.\ \neg(\exists nid \in actUt\ t\ n'.\ \|nid\|_{t\ n'} \wedge mining\ (\sigma_{nid}(t\ n'))))$
assumes *consensus*: $\bigwedge kid\ t\ t'\ bc' :: ('nid\ BC).\ [\![trusted\ kid]\!] \Longrightarrow eval\ kid\ t\ t'\ 0\ (\Box(ass\ (\lambda kt.\ bc' = (if\ (\exists b \in pin\ kt.\ length\ b > length\ (bc\ kt))\ then\ (MAX\ (pin\ kt))\ else\ (bc\ kt))) \longrightarrow^b \bigcirc(ass\ (\lambda kt.(\neg\ mining\ kt \wedge bc\ kt = bc' \vee mining\ kt \wedge bc\ kt = bc' @ [kid])))))$
and *attacker*: $\bigwedge kid\ t\ t'\ bc'.\ [\![\neg\ trusted\ kid]\!] \Longrightarrow eval\ kid\ t\ t'\ 0\ (\Box(ass\ (\lambda kt.\ bc' = (SOME\ b.\ b \in (pin\ kt \cup \{bc\ kt\}))) \longrightarrow^b \bigcirc(ass\ (\lambda kt.(\neg\ mining\ kt \wedge prefix\ (bc\ kt)\ bc' \vee mining\ kt \wedge bc\ kt = bc' @ [kid])))))$
and *forward*: $\bigwedge kid\ t\ t'.\ eval\ kid\ t\ t'\ 0\ (\Box(ass\ (\lambda kt.\ pout\ kt = bc\ kt)))$
— At each time point a node will forward its blockchain to the network
and *conn*: $\bigwedge k\ kid.\ active\ kid\ k \Longrightarrow pin\ (cmp\ kid\ k) = (\bigcup kid' \in \{kid'.\ active\ kid'\ k\}.\ \{pout\ (cmp\ kid'\ k)\})$
and *act*: $\bigwedge t\ n :: nat.\ finite\ \{kid :: 'nid.\ \|kid\|_{t\ n}\}$
and *actTr*: $\bigwedge t\ n :: nat.\ \exists nid.\ trusted\ nid \wedge \|nid\|_{t\ n} \wedge \|nid\|_{t\ (Suc\ n)}$
and *fair*: $\bigwedge t\ kid\ n :: nat.\ [\![\neg\ trusted\ kid;\ mining\ (\sigma_{kid}(t\ n))]\!] \Longrightarrow trNxt\ t\ n$
and *closed*: $\bigwedge t\ kid\ b\ n :: nat.\ [\![b \in pin\ (\sigma_{kid}(t\ n))]\!] \Longrightarrow \exists kid'.\ \|kid'\|_{t\ n} \wedge pout\ (\sigma_{kid'}(t\ n)) = b$
begin

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 assEANow*[*of nid t t' n*]] **by** *blast*

lemma *finite-input*:
fixes *t n kid*
assumes $\|kid\|_{t\ n}$

shows *finite* (*pin* (*cmp kid* (*t n*))) **using** *conn*[*of kid t n*] *act assms* **by** *auto*

lemma *nempty-input*:
fixes *t n kid*
assumes $\|kid\|_{t\ n}$
shows *pin* (*cmp kid* (*t n*))≠{} **using** *conn*[*of kid t n*] *act assms* **by** *auto*

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

4.3.1 Component Behavior

lemma *bhv-tr-ex*:
fixes *t* **and** *t'*::*nat* ⇒ *'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 ¬ *mining* $(\sigma_{tid} t\ \langle tid \rightarrow t\rangle_n) \wedge bc\ (\sigma_{tid} t\ \langle tid \rightarrow t\rangle_n) = Blockchain.MAX\ (pin\ (\sigma_{tid} t\ \langle tid \Leftarrow t\rangle_n)) \vee mining\ (\sigma_{tid} t\ \langle tid \rightarrow t\rangle_n) \wedge bc\ (\sigma_{tid} t\ \langle tid \rightarrow t\rangle_n) = Blockchain.MAX\ (pin\ (\sigma_{tid} t\ \langle tid \Leftarrow t\rangle_n))$ @ [*tid*]
proof −
let *?cond* = $\lambda kt.\ MAX\ (pin\ (\sigma_{tid} t\ \langle tid \Leftarrow t\rangle_n)) = (if\ (\exists\, b{\in}pin\ kt.\ length\ b > length\ (bc\ kt))\ then\ (MAX\ (pin\ kt))\ else\ (bc\ kt))$
let *?check* = $\lambda kt.$ ¬ *mining kt* ∧ *bc kt* = *MAX* (*pin* $(\sigma_{tid} t\ \langle tid \Leftarrow t\rangle_n)) \vee mining\ kt \wedge bc\ kt = MAX\ (pin\ (\sigma_{tid} t\ \langle tid \Leftarrow t\rangle_n))$ @ [*tid*]
from ‹*trusted tid*› **have** *eval tid t t' 0* $((\Box((ass\ ?cond) \longrightarrow^b \bigcirc ass\ ?check)))$ **using** *consensus*[*of tid - - MAX* (*pin* $(\sigma_{tid} t\ \langle tid \Leftarrow t\rangle_n))$] **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'* $\langle tid \Leftarrow t\rangle_n$ *(ass (?cond)* $\longrightarrow^b \bigcirc ass\ ?check)$ **using** *globEA*[*of 0 tid t t'* $((ass\ ?cond) \longrightarrow^b \bigcirc ass\ ?check)\ \langle tid \Leftarrow t\rangle_n$] **by** *fastforce*
moreover have *eval tid t t'* $\langle tid \Leftarrow t\rangle_n$ *(ass (?cond))*
proof (*rule assIA*)
from ‹$\exists\, n'{<}n.\ \|tid\|_{t\ n'}$› **show** $\exists\, i{\geq}\langle tid \Leftarrow t\rangle_n.\ \|tid\|_{t\ i}$ **using** *lastAct-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** *lastActNxt* **by** *simp*
qed
ultimately have *eval tid t t'* $\langle tid \Leftarrow t\rangle_n$ $(\bigcirc ass\ ?check)$ **using** *impE*[*of tid t t' - ass* (*?cond*) $\bigcirc ass\ ?check$] **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** *lastActNxtAct* **by** *simp*
with *assms(3)* **have** $\langle tid \rightarrow t\rangle_n{>}\langle tid \rightarrow t\rangle_{\langle tid \Leftarrow t\rangle_n}$ **using** *lastActNxt* **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** *lastActNxtAct* **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'* $\langle tid \rightarrow t\rangle_n$ (*ass ?check*) **using** *nxtEA1*[*of tid t* $\langle tid \Leftarrow t\rangle_n$ *t' ass ?check* $\langle tid \rightarrow t\rangle_n$] **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** *assEANow*[*of tid t t'* $\langle tid \rightarrow t\rangle_n$ *?check*] **by** *simp*
qed

lemma *bhv-tr-in*:
fixes *t* **and** *t'*::*nat* ⇒ *'ND* **and** *tid*
assumes *trusted tid*
and $\exists\, n'{\geq}n.\ \|tid\|_{t\ n'}$
and $\exists\, n'{<}n.\ \|tid\|_{t\ n'}$
and $\neg\ (\exists\, b{\in}pin\ (\sigma_{tid} t\ \langle tid \Leftarrow t\rangle_n).\ length\ b > length\ (bc\ (\sigma_{tid} t\ \langle tid \Leftarrow t\rangle_n)))$
shows $\neg\ mining\ (\sigma_{tid} t\ \langle tid \rightarrow t\rangle_n) \wedge bc\ (\sigma_{tid} t\ \langle tid \rightarrow t\rangle_n) = bc\ (\sigma_{tid} t\ \langle tid \Leftarrow t\rangle_n) \vee mining\ (\sigma_{tid} t\ \langle tid \rightarrow t\rangle_n) \wedge bc\ (\sigma_{tid} t\ \langle tid \rightarrow t\rangle_n) = bc\ (\sigma_{tid} t\ \langle tid \Leftarrow t\rangle_n)\ @\ [tid]$
proof −
let *?cond* = $\lambda kt.\ bc\ (\sigma_{tid} t\ \langle tid \Leftarrow t\rangle_n) = (if\ (\exists\, b{\in}pin\ kt.\ length\ b > length\ (bc\ kt))\ then\ (MAX\ (pin\ kt))\ else\ (bc\ kt))$
let *?check* = $\lambda kt.\ \neg\ mining\ kt \wedge bc\ kt = bc\ (\sigma_{tid} t\ \langle tid \Leftarrow t\rangle_n) \vee mining\ kt \wedge bc\ kt = bc\ (\sigma_{tid} t\ \langle tid \Leftarrow t\rangle_n)\ @\ [tid]$
from ⟨*trusted tid*⟩ **have** *eval tid t t' 0* ((□((*ass ?cond*) $\longrightarrow^b$ ○*ass ?check*))) **using** *consensus*[*of tid - - bc* ($\sigma_{tid} t\ \langle tid \Leftarrow t\rangle_n$)] **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'* $\langle tid \Leftarrow t\rangle_n$ (*ass* (*?cond*) $\longrightarrow^b$ ○*ass ?check*) **using** *globEA*[*of 0 tid t t'* ((*ass ?cond*) $\longrightarrow^b$ ○*ass ?check*) $\langle tid \Leftarrow t\rangle_n$] **by** *fastforce*
moreover have *eval tid t t'* $\langle tid \Leftarrow t\rangle_n$ (*ass* (*?cond*))
proof (*rule assIA*)
from $\langle\exists\, n'{<}n.\ \|tid\|_{t\ n'}\rangle$ **show** $\exists\, i{\geq}\langle tid \Leftarrow t\rangle_n.\ \|tid\|_{t\ i}$ **using** *lastAct-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** *lastActNxt* **by** *simp*
qed
ultimately have *eval tid t t'* $\langle tid \Leftarrow t\rangle_n$ (○*ass ?check*) **using** *impE*[*of tid t t' - ass* (*?cond*) ○*ass ?check*] **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** *lastActNxtAct* **by** *simp*
with *assms(3)* **have** $\langle tid \rightarrow t\rangle_n{>}\langle tid \rightarrow t\rangle_{\langle tid \Leftarrow t\rangle_n}$ **using** *lastActNxt* **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** *lastActNxtAct* **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'* $\langle tid \rightarrow t\rangle_n$ (*ass ?check*) **using** *nxtEA1*[*of tid t* $\langle tid \Leftarrow t\rangle_n$ *t' ass ?check* $\langle tid \rightarrow t\rangle_n$] **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** *assEANow*[*of tid t t'* $\langle tid \rightarrow t\rangle_n$ *?check*] **by** *simp*
qed

lemma *bhv-ut*:
fixes *t* **and** *t'*::*nat* ⇒ *'ND* **and** *uid*
assumes ¬ *trusted uid*
and $\exists\, n'{\geq}n.\ \|uid\|_{t\ n'}$
and $\exists\, n'{<}n.\ \|uid\|_{t\ n'}$
shows ¬ *mining* $(\sigma_{uid} t\ \langle uid \rightarrow t\rangle_n) \wedge$ *prefix* (*bc* $(\sigma_{uid} t\ \langle uid \rightarrow t\rangle_n)$) (*SOME b. b* ∈ *pin* $(\sigma_{uid} t\ \langle uid \Leftarrow t\rangle_n) \cup \{bc\ (\sigma_{uid} t\ \langle uid \Leftarrow t\rangle_n)\}) \vee$ *mining* $(\sigma_{uid} t\ \langle uid \rightarrow t\rangle_n) \wedge$ *bc* $(\sigma_{uid} t\ \langle uid \rightarrow t\rangle_n) =$ (*SOME b. b* ∈ *pin* $(\sigma_{uid} t\ \langle uid \Leftarrow t\rangle_n) \cup \{bc\ (\sigma_{uid} t\ \langle uid \Leftarrow t\rangle_n)\})$ @ [*uid*]
proof −
let *?cond* = λ*kt*. (*SOME b. b* ∈ (*pin* $(\sigma_{uid} t\ \langle uid \Leftarrow t\rangle_n) \cup \{bc\ (\sigma_{uid} t\ \langle uid \Leftarrow t\rangle_n)\})) = (*SOME b. b* ∈ *pin kt* ∪ {*bc kt*})
let *?check* = λ*kt*. ¬ *mining kt* ∧ *prefix* (*bc kt*) (*SOME b. b* ∈ *pin* $(\sigma_{uid} t\ \langle uid \Leftarrow t\rangle_n) \cup \{bc\ (\sigma_{uid} t\ \langle uid \Leftarrow t\rangle_n)\}) \vee$ *mining kt* ∧ *bc kt* = (*SOME b. b* ∈ *pin* $(\sigma_{uid} t\ \langle uid \Leftarrow t\rangle_n) \cup \{bc\ (\sigma_{uid} t\ \langle uid \Leftarrow t\rangle_n)\})$ @ [*uid*]
from ⟨¬ *trusted uid*⟩ **have** *eval uid t t' 0* ((□((*ass ?cond*) $\longrightarrow^b$ ○*ass ?check*))) **using** *attacker*[*of uid - -* (*SOME b. b* ∈ *pin* $(\sigma_{uid} t\ \langle uid \Leftarrow t\rangle_n) \cup \{bc\ (\sigma_{uid} t\ \langle uid \Leftarrow t\rangle_n)\})$] **by** *simp*
moreover from *assms* **have** $\exists\, i{\geq}0.\ \|uid\|_{t\ i}$ **by** *auto*
moreover have $\langle uid \leftarrow t\rangle_0 \leq \langle uid \Leftarrow t\rangle_n$ **by** *simp*
ultimately have *eval uid t t'* $\langle uid \Leftarrow t\rangle_n$ (*ass* (*?cond*) $\longrightarrow^b$ ○*ass ?check*) **using** *globEA*[*of 0 uid t t'* ((*ass ?cond*) $\longrightarrow^b$ ○*ass ?check*) $\langle uid \Leftarrow t\rangle_n$] **by** *fastforce*
moreover have *eval uid t t'* $\langle uid \Leftarrow t\rangle_n$ (*ass* (*?cond*))
proof (*rule assIA*)
from $\langle \exists\, n'{<}n.\ \|uid\|_{t\ n'}\rangle$ **show** $\exists\, i{\geq}\langle uid \Leftarrow t\rangle_n.\ \|uid\|_{t\ i}$ **using** *lastAct-prop*(*1*) **by** *blast*
with *assms*(*3*) **show** *?cond* $(\sigma_{uid} t\ \langle uid \rightarrow t\rangle_{\langle uid \Leftarrow t\rangle_n})$ **using** *lastActNxt* **by** *simp*
qed
ultimately have *eval uid t t'* $\langle uid \Leftarrow t\rangle_n$ (○*ass ?check*) **using** *impE*[*of uid t t' - ass* (*?cond*) ○*ass ?check*] **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** *lastActNxtAct* **by** *simp*
with *assms*(*3*) **have** $\langle uid \rightarrow t\rangle_n{>}\langle uid \rightarrow t\rangle_{\langle uid \Leftarrow t\rangle_n}$ **using** *lastActNxt* **by** *simp*
moreover from $\langle \exists\, n'{\geq}n.\ \|uid\|_{t\ n'}\rangle$ **have** $\|uid\|_{t\ \langle uid \rightarrow t\rangle_n}$ **using** *nxtActI* **by** *simp*
ultimately show *?thesis* **by** *auto*
qed
moreover from *assms* **have** $\langle uid \Leftarrow t\rangle_n \leq \langle uid \rightarrow t\rangle_n$ **using** *lastActNxtAct* **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* (*ass ?check*) **using** *nxtEA1*[*of uid t* ⟨*uid* ⇐ *t*⟩*n t′ ass ?check* ⟨*uid* → *t*⟩*n*] **by** *simp*
moreover from ‹∃ *n′*≥*n*. ‖*uid*‖*t n′*› **have** ‖*uid*‖*t* ⟨*uid* → *t*⟩*n* **using** *nxtActI* **by** *simp*
ultimately show *?thesis* **using** *assEANow*[*of uid t t′* ⟨*uid* → *t*⟩*n ?check*] **by** *simp*
qed

lemma *bhv-tr-context*:
assumes *trusted tid*
and ‖*tid*‖*t n*
and ∃ *n′*≥*nS*. *n′*<*n* ∧ ‖*tid*‖*t n′*
shows *prefix* (*bc* (σ*tid t* ⟨*tid* ⇐ *t*⟩*n*)) (*bc* (σ*tid t n*)) ∨
(∃ *nid′*. ‖*nid′*‖*t* ⟨*tid* ⇐ *t*⟩*n* ∧ *length* (*bc* (σ*nid′ t* ⟨*tid* ⇐ *t*⟩*n*)) ≥ *length* (*MAX* (*pin* (σ*tid t* ⟨*tid* ⇐ *t*⟩*n*))) ∧ *prefix* (*bc* (σ*nid′ t* ⟨*tid* ⇐ *t*⟩*n*)) (*bc* (σ*tid t n*)))
proof *cases*
assume *casmp*: ∃ *b*∈*pin* (σ*tid t* ⟨*tid* ⇐ *t*⟩*n*). *length b* > *length* (*bc* (σ*tid t* ⟨*tid* ⇐ *t*⟩*n*))
moreover from *assms*(*2*) **have** ∃ *n′*≥*n*. ‖*tid*‖*t n′* **by** *auto*
moreover from *assms*(*3*) **have** ∃ *n′*<*n*. ‖*tid*‖*t n′* **by** *auto*
ultimately have *bc* (σ*tid t* ⟨*tid* → *t*⟩*n*) = *Blockchain.MAX* (*pin* (σ*tid t* ⟨*tid* ⇐ *t*⟩*n*)) ∨ *bc* (σ*tid t* ⟨*tid* → *t*⟩*n*) = *Blockchain.MAX* (*pin* (σ*tid t* ⟨*tid* ⇐ *t*⟩*n*)) @ [*tid*] **using** *assms*(*1*) *bhv-tr-ex* **by** *auto*
moreover from *assms*(*2*) **have** ⟨*tid* → *t*⟩*n* = *n* **using** *nxtAct-active* **by** *simp*
ultimately have *bc* (σ*tid t n*) = *Blockchain.MAX* (*pin* (σ*tid t* ⟨*tid* ⇐ *t*⟩*n*)) ∨ *bc* (σ*tid t n*) = *Blockchain.MAX* (*pin* (σ*tid t* ⟨*tid* ⇐ *t*⟩*n*)) @ [*tid*] **by** *simp*
hence *prefix* (*Blockchain.MAX* (*pin* (σ*tid t* ⟨*tid* ⇐ *t*⟩*n*))) (*bc* (σ*tid t n*)) **by** *auto*
moreover have *length* (*MAX* (*pin* (σ*tid t* ⟨*tid* ⇐ *t*⟩*n*))) ≥ *length* (*MAX* (*pin* (σ*tid t* ⟨*tid* ⇐ *t*⟩*n*))) **..**
moreover have *Blockchain.MAX* (*pin* (σ*tid t* ⟨*tid* ⇐ *t*⟩*n*)) ∈ *pin* (σ*tid t* ⟨*tid* ⇐ *t*⟩*n*)
proof −
from ‹∃ *n′*<*n*. ‖*tid*‖*t n′*› **have** ‖*tid*‖*t* ⟨*tid* ⇐ *t*⟩*n* **using** *lastAct-prop*(*1*) **by** *simp*
hence *finite* (*pin* (σ*tid*(*t* ⟨*tid* ⇐ *t*⟩*n*))) **using** *finite-input*[*of tid t* ⟨*tid* ⇐ *t*⟩*n*] **by** *simp*
moreover from *casmp* **obtain** *b* **where** *b* ∈ *pin* (σ*tid t* ⟨*tid* ⇐ *t*⟩*n*) **and** *length b* > *length* (*bc* (σ*tid t* ⟨*tid* ⇐ *t*⟩*n*)) **by** *auto*
ultimately show *?thesis* **using** *max-prop*(*1*) **by** *auto*
qed
with *closed* **obtain** *nid* **where** ‖*nid*‖*t* ⟨*tid* ⇐ *t*⟩*n* **and** *pout* (σ*nid t* ⟨*tid* ⇐ *t*⟩*n*) = *Blockchain.MAX* (*pin* (σ*tid t* ⟨*tid* ⇐ *t*⟩*n*)) **by** *blast*
with *fwd-bc*[*of nid t* ⟨*tid* ⇐ *t*⟩*n*] **have** ‖*nid*‖*t* ⟨*tid* ⇐ *t*⟩*n* **and** *bc* (σ*nid t* ⟨*tid* ⇐ *t*⟩*n*) = *Blockchain.MAX* (*pin* (σ*tid t* ⟨*tid* ⇐ *t*⟩*n*)) **by** *auto*
ultimately show *?thesis* **by** *auto*
next
assume ¬ (∃ *b*∈*pin* (σ*tid t* ⟨*tid* ⇐ *t*⟩*n*). *length b* > *length* (*bc* (σ*tid t* ⟨*tid* ⇐ *t*⟩*n*)))
moreover from *assms*(*2*) **have** ∃ *n′*≥*n*. ‖*tid*‖*t n′* **by** *auto*
moreover from *assms*(*3*) **have** ∃ *n′*<*n*. ‖*tid*‖*t n′* **by** *auto*
ultimately have *bc* (σ*tid t* ⟨*tid* → *t*⟩*n*) = *bc* (σ*tid t* ⟨*tid* ⇐ *t*⟩*n*) ∨ *bc* (σ*tid t* ⟨*tid* → *t*⟩*n*) = *bc* (σ*tid t* ⟨*tid* ⇐ *t*⟩*n*) @ [*tid*] **using** *assms*(*1*) *bhv-tr-in*[*of tid n t*] **by** *auto*
moreover from *assms*(*2*) **have** ⟨*tid* → *t*⟩*n* = *n* **using** *nxtAct-active* **by** *simp*
ultimately have *bc* (σ*tid t n*) = *bc* (σ*tid t* ⟨*tid* ⇐ *t*⟩*n*) ∨ *bc* (σ*tid t n*) = *bc* (σ*tid t* ⟨*tid* ⇐ *t*⟩*n*) @ [*tid*] **by** *simp*
thus *?thesis* **by** *auto*
qed

lemma *bhv-ut-context*:
assumes ¬ *trusted uid*
and $\|uid\|_{t\ n}$
and $\exists\, n'{\geq}n_S.\ n'{<}n \wedge \|uid\|_{t\ n'}$
shows (*mining* ($\sigma_{uid}t\ n$) ∧ *prefix* (*bc* ($\sigma_{uid}t\ n$)) (*bc* ($\sigma_{uid}t\ \langle uid \Leftarrow t\rangle_n$) @ [*uid*])) ∨ ¬ *mining* ($\sigma_{uid}t$ *n*) ∧ *prefix* (*bc* ($\sigma_{uid}t\ n$)) (*bc* ($\sigma_{uid}t\ \langle uid \Leftarrow t\rangle_n$)) ∨
($\exists\, nid'.\ \|nid'\|_{t\ \langle uid \Leftarrow t\rangle_n} \wedge$ (*mining* ($\sigma_{uid}t\ n$) ∧ *prefix* (*bc* ($\sigma_{uid}t\ n$)) (*bc* ($\sigma_{nid'}t\ \langle uid \Leftarrow t\rangle_n$) @ [*uid*]) ∨ ¬ *mining* ($\sigma_{uid}t\ n$) ∧ *prefix* (*bc* ($\sigma_{uid}t\ n$)) (*bc* ($\sigma_{nid'}t\ \langle uid \Leftarrow t\rangle_n$))))
proof −
let *?bc*=*SOME b*. *b* ∈ *pin* ($\sigma_{uid}t\ \langle uid \Leftarrow t\rangle_n$) ∪ {*bc* ($\sigma_{uid}t\ \langle uid \Leftarrow t\rangle_n$)}
have *bc-ex*: *?bc* ∈ *pin* ($\sigma_{uid}t\ \langle uid \Leftarrow t\rangle_n$) ∨ *?bc* ∈ {*bc* ($\sigma_{uid}t\ \langle uid \Leftarrow t\rangle_n$)}
proof −
have ∃ *b*. *b*∈ *pin* ($\sigma_{uid}t\ \langle uid \Leftarrow t\rangle_n$) ∪ {*bc* ($\sigma_{uid}t\ \langle uid \Leftarrow t\rangle_n$)} **by** *auto*
hence *?bc* ∈ *pin* ($\sigma_{uid}t\ \langle uid \Leftarrow t\rangle_n$) ∪ {*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 ¬ *mining* ($\sigma_{uid}t\ \langle uid \rightarrow t\rangle_n$) ∧ *prefix* (*bc* ($\sigma_{uid}t\ \langle uid \rightarrow t\rangle_n$)) *?bc* ∨ *mining* ($\sigma_{uid}t$ $\langle uid \rightarrow t\rangle_n$) ∧ *bc* ($\sigma_{uid}t\ \langle uid \rightarrow t\rangle_n$) = *?bc* @ [*uid*] **using** *bhv-ut*[*of uid n t*] *assms(1)* **by** *simp*
moreover from *assms(2)* **have** $\langle uid \rightarrow t\rangle_n = n$ **using** *nxtAct-active* **by** *simp*
ultimately have *casmp*: ¬ *mining* ($\sigma_{uid}t\ n$) ∧ *prefix* (*bc* ($\sigma_{uid}t\ n$)) *?bc* ∨ *mining* ($\sigma_{uid}t\ n$) ∧ *bc* ($\sigma_{uid}t\ n$) = *?bc* @ [*uid*] **by** *simp*

from *bc-ex* **have** *?bc* ∈ *pin* ($\sigma_{uid}t\ \langle uid \Leftarrow t\rangle_n$) ∨ *?bc* ∈ {*bc* ($\sigma_{uid}t\ \langle uid \Leftarrow t\rangle_n$)} .
thus *?thesis*
proof
assume *?bc* ∈ *pin* ($\sigma_{uid}t\ \langle uid \Leftarrow t\rangle_n$)
with *closed* **obtain** *nid* **where** $\|nid\|_{t\ \langle uid \Leftarrow t\rangle_n}$ **and** *pout* ($\sigma_{nid}t\ \langle uid \Leftarrow t\rangle_n$) = *?bc* **by** *blast*
with *fwd-bc*[*of nid t* $\langle uid \Leftarrow t\rangle_n$] **have** *bc* ($\sigma_{nid}t\ \langle uid \Leftarrow t\rangle_n$) = *?bc* **by** *simp*
with *casmp* **have** ¬ *mining* ($\sigma_{uid}t\ n$) ∧ *prefix* (*bc* ($\sigma_{uid}t\ n$)) (*bc* ($\sigma_{nid}t\ \langle uid \Leftarrow t\rangle_n$)) ∨ *mining* ($\sigma_{uid}t\ n$) ∧ *bc* ($\sigma_{uid}t\ n$) = (*bc* ($\sigma_{nid}t\ \langle uid \Leftarrow t\rangle_n$)) @ [*uid*] **by** *simp*
with ‹$\|nid\|_{t\ \langle uid \Leftarrow t\rangle_n}$› **show** *?thesis* **by** *auto*
next
assume *?bc* ∈ {*bc* ($\sigma_{uid}t\ \langle uid \Leftarrow t\rangle_n$)}
hence *?bc* = *bc* ($\sigma_{uid}t\ \langle uid \Leftarrow t\rangle_n$) **by** *simp*
with *casmp* **show** *?thesis* **by** *auto*
qed
qed

4.3.2 Maximal Trusted Blockchains

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

lemma *mbc-ex*:

fixes t n
shows $\exists x.$ *mbc-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-def* **by** *blast*
thus *?ALL*≠{} **by** *auto*
from *act* **have** *finite* (*actTr* t n) **using** *actTr-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 ‹*bc* $(\sigma_{nid}(t\ n)) \in$ *?ALL*› **show** *?ALL*≠{} **by** *auto*
from *act* **have** *finite* (*actTr* t n) **using** *actTr-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-cond* t n b)

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

4.3.3 Trusted Proof of Work

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

abbreviation *pow-cond*:: *trace* $\Rightarrow$ *nat* $\Rightarrow$ *nat* $\Rightarrow$ *bool*
where *pow-cond* t n $n' \equiv \forall$ *nid*$\in$*actTr* t $n.$ *length* (*bc* $(\sigma_{nid}(t\ n))) \leq n'$

lemma *pow-ex*:
fixes t n
shows *pow-cond* t n (*length* (*bc* $(\sigma_{MBC\ t\ n}(t\ n))))$
and $\forall x'.$ *pow-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 ∃ *tid*∈*actTr t n. length* (*bc* (σ*tid*(*t n*))) = *x*
and ∀ *tid*∈*actTr t n. length* (*bc* (σ*tid*(*t n*))) ≤ *x*
shows *PoW t n* = *x*
proof −
have (*LEAST x. pow-cond t n x*) = *x*
proof (*rule Least-equality*)
from *assms*(*2*) **show** ∀ *nid*∈*actTr t n. length* (*bc* (σ*nid t n*)) ≤ *x* **by** *simp*
next
fix *y*
assume ∀ *nid*∈*actTr t n. length* (*bc* (σ*nid t n*)) ≤ *y*
thus *x* ≤ *y* **using** *assms*(*1*) **by** *auto*
qed
with *PoW-def* **show** *?thesis* **by** *simp*
qed

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

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

lemma *pow-le-max*:
assumes *trusted tid*
and ‖*tid*‖*t n*
shows *PoW t n* ≤ *length* (*MAX* (*pin* (σ*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* (σ*MBC t n t n*) = *bc* (σ*MBC t n t n*) **using** *forward globEANow*[*THEN assEANow*[*of MBC t n t t′ n* λ*kt. pout kt* = *bc kt*]] **by** *auto*
with *assms* ‹‖*MBC t n*‖*t n*› ‹*trusted* (*MBC t n*)› **have** *bc* (σ*MBC t n t n*) ∈ *pin* (σ*tid t n*) **using** *conn* **by** *auto*

moreover from *assms* (*2*) **have** *finite* (*pin* (σ_{tid}*t n*)) **using** *finite-input*[*of tid t n*] **by** *simp*
ultimately have *length* (*bc* ($\sigma_{MBC\ t\ n}$*t n*)) ≤ *length* (*MAX* (*pin* (σ_{tid}*t n*))) **using** *max-prop*(*2*) **by** *auto*
with *pow-mbc* **show** *?thesis* **by** *simp*
qed

lemma *pow-ge-lgth*:
assumes *trusted tid*
and $\|tid\|_{t\ n}$
shows *length* (*bc* (σ_{tid}*t n*)) ≤ *PoW t n*
proof −
from *assms* **have** *tid* ∈ *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 ¬(∃ *b*∈*pin* (σ_{tid}*t n*). *length b* > *length* (*bc* (σ_{tid}*t n*)))
shows *length* (*bc* (σ_{tid}*t n*)) ≥ *PoW t n*
proof −
from *assms* (*3*) **have** ∀ *b*∈*pin* (σ_{tid}*t n*). *length b* ≤ *length* (*bc* (σ_{tid}*t n*)) **by** *auto*
moreover from *assms*(*2*) *nempty-input*[*of tid t n*] *finite-input*[*of tid t n*] **have** *MAX* (*pin* (σ_{tid}*t n*)) ∈ *pin* (σ_{tid}*t n*) **using** *max-prop*(*1*)[*of pin* (σ_{tid}*t n*)] **by** *simp*
ultimately have *length* (*MAX* (*pin* (σ_{tid}*t n*))) ≤ *length* (*bc* (σ_{tid}*t n*)) **by** *simp*
moreover from *assms* **have** *PoW t n* ≤ *length* (*MAX* (*pin* (σ_{tid}*t n*))) **using** *pow-le-max* **by** *simp*
ultimately show *?thesis* **by** *simp*
qed

lemma *pow-mono*:
shows *n'*≥*n* ⟹ *PoW t n'* ≥ *PoW t n*
proof (*induction n' rule*: *dec-induct*)
case *base*
then show *?case* **by** *simp*
next
case (*step n'*)
hence *PoW t n* ≤ *PoW t n'* **by** *simp*
moreover have *PoW t* (*Suc n'*) ≥ *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 ∃ *b*∈*pin* (σ_{tid}*t n'*). *length b* > *length* (*bc* (σ_{tid}*t n'*))
moreover from ‹$\|tid\|_{t\ (Suc\ n')}$› **have** $\langle tid \rightarrow t\rangle_{Suc\ n'} = Suc\ n'$ **using** *nxtAct-active* **by** *simp*
moreover from ‹$\|tid\|_{t\ n'}$› **have** $\langle tid \Leftarrow t\rangle_{Suc\ n'} = n'$ **using** *lastAct-prop*(*2*) *lastActless le-less-Suc-eq* **by** *blast*
moreover from ‹$\|tid\|_{t\ n'}$› **have** ∃ *n''*<*Suc n'*. $\|tid\|_{t\ n''}$ **by** *blast*
moreover from ‹$\|tid\|_{t\ (Suc\ n')}$› **have** ∃ *n''*≥*Suc n'*. $\|tid\|_{t\ n''}$ **by** *auto*
ultimately have *bc* (σ_{tid}*t* (*Suc n'*)) = *Blockchain.MAX* (*pin* (σ_{tid}*t n'*)) ∨ *bc* (σ_{tid}*t* (*Suc n'*)) =

Blockchain.MAX (*pin* (σ_{tid}*t* *n'*)) @ [*tid*] **using** ‹*trusted tid*› *bhv-tr-ex*[*of tid Suc n' t*] **by** *auto*
hence *length* (*bc* (σ_{tid}*t* (*Suc n'*))) ≥ *length* (*Blockchain.MAX* (*pin* (σ_{tid}*t* *n'*))) **by** *auto*
moreover from ‹*trusted tid*› ‹$\|tid\|_{t\ n'}$› **have** *length* (*Blockchain.MAX* (*pin* (σ_{tid}*t* *n'*))) ≥ *PoW t n'* **using** *pow-le-max* **by** *simp*
ultimately have *PoW t n'* ≤ *length* (*bc* (σ_{tid}*t* (*Suc n'*))) **by** *simp*
moreover from ‹*trusted tid*› ‹$\|tid\|_{t\ (Suc\ n')}$› **have** *length* (*bc* (σ_{tid}*t* (*Suc n'*))) ≤ *PoW t* (*Suc n'*) **using** *pow-ge-lgth* **by** *simp*
ultimately show *?thesis* **by** *simp*
next
assume *asmp*: ¬(∃ *b*∈*pin* (σ_{tid}*t* *n'*). *length b* > *length* (*bc* (σ_{tid}*t* *n'*)))
moreover from ‹$\|tid\|_{t\ (Suc\ n')}$› **have** $\langle tid \rightarrow t\rangle_{Suc\ n'}$ = *Suc n'* **using** *nxtAct-active* **by** *simp*
moreover from ‹$\|tid\|_{t\ n'}$› **have** $\langle tid \Leftarrow t\rangle_{Suc\ n'} = n'$ **using** *lastAct-prop*(*2*) *lastActless le-less-Suc-eq* **by** *blast*
moreover from ‹$\|tid\|_{t\ n'}$› **have** ∃ *n''*<*Suc n'*. $\|tid\|_{t\ n''}$ **by** *blast*
moreover from ‹$\|tid\|_{t\ (Suc\ n')}$› **have** ∃ *n''*≥*Suc n'*. $\|tid\|_{t\ n''}$ **by** *auto*
ultimately have *bc* (σ_{tid}*t* (*Suc n'*)) = *bc* (σ_{tid}*t* *n'*) ∨ *bc* (σ_{tid}*t* (*Suc n'*)) = *bc* (σ_{tid}*t* *n'*) @ [*tid*] **using** ‹*trusted tid*› *bhv-tr-in*[*of tid Suc n' t*] **by** *auto*
hence *length* (*bc* (σ_{tid}*t* (*Suc n'*))) ≥ *length* (*bc* (σ_{tid}*t* *n'*)) **by** *auto*
moreover from ‹*trusted tid*› ‹$\|tid\|_{t\ n'}$› *asmp* **have** *length* (*bc* (σ_{tid}*t* *n'*)) ≥ *PoW t n'* **using** *pow-le-lgth* **by** *simp*
moreover from ‹*trusted tid*› ‹$\|tid\|_{t\ (Suc\ n')}$› **have** *length* (*bc* (σ_{tid}*t* (*Suc n'*))) ≤ *PoW t* (*Suc n'*) **using** *pow-ge-lgth* **by** *simp*
ultimately show *?thesis* **by** *simp*
qed
qed
ultimately show *?case* **by** *auto*
qed

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

4.3.4 Trusted Next

lemma *pow-eq-trnxt*:
assumes *PoW t n* = *PoW t n'*
and *trNxt t n*
and *n'*≥*n*
shows *trNxt t n'*
proof −
from *assms* (*2*) **have** (∃ *n'*≥*n*. *PoW t n'* > *PoW t n* ∧ (∀ *n''*>*n*. *n''* ≤ *n'* ⟶ ¬(∃ *nid*∈*actUt t n''*. $\|nid\|_{t\ n''}$ ∧ *mining* (σ_{nid}(*t n''*))))) ∨ (∀ *n'*>*n*. ¬(∃ *nid*∈*actUt t n'*. $\|nid\|_{t\ n'}$ ∧ *mining* (σ_{nid}(*t n'*))))
using *trNxt-def* **by** *simp*
thus *?thesis*
proof
assume ∃ *n'*≥*n*. *PoW t n'* > *PoW t n* ∧ (∀ *n''*>*n*. *n''* ≤ *n'* ⟶ ¬(∃ *nid*∈*actUt t n''*. $\|nid\|_{t\ n''}$ ∧

mining $(\sigma_{nid}(t\ n'')))$
then obtain n'' **where** $n''\geq n$ **and** $PoW\ t\ n'' > PoW\ t\ n$ **and** $\forall\, n'''{>}n.\ n''' \leq n'' \longrightarrow \neg(\exists\, nid{\in}actUt\ t\ n'''.\ \|nid\|_{t\ n'''} \wedge mining\ (\sigma_{nid}(t\ n''')))$ **by** *auto*
moreover have $n'' \geq n'$
proof (*rule ccontr*)
assume $\neg n''{\geq}n'$
hence $n''{<}n'$ **by** *simp*
with *assms(1)* ‹$n''\geq n$› ‹$PoW\ t\ n'' > PoW\ t\ n$› **show** *False* **using** *pow-equals*[*of t n n' n''*] **by** *simp*
qed
moreover from ‹$PoW\ t\ n'' > PoW\ t\ n$› *assms(1)* **have** $PoW\ t\ n'' > PoW\ t\ n'$ **by** *simp*
moreover from *assms(3)* ‹$\forall\, n'''{>}n.\ n''' \leq n'' \longrightarrow \neg(\exists\, nid{\in}actUt\ t\ n'''.\ \|nid\|_{t\ n'''} \wedge mining\ (\sigma_{nid}(t\ n''')))$› **have** $\forall\, n'''{>}n'.\ n''' \leq n'' \longrightarrow \neg(\exists\, nid{\in}actUt\ t\ n'''.\ \|nid\|_{t\ n'''} \wedge mining\ (\sigma_{nid}(t\ n''')))$ **using** *le-less-trans* **by** *blast*
ultimately show *?thesis* **using** *trNxt-def*[*of t n'*] **by** *auto*
next
assume $\forall\, n'{>}n.\ \neg(\exists\, nid{\in}actUt\ t\ n'.\ \|nid\|_{t\ n'} \wedge mining\ (\sigma_{nid}(t\ n')))$
with *assms(3)* **have** $\forall\, n''{>}n'.\ \neg(\exists\, nid{\in}actUt\ t\ n''.\ \|nid\|_{t\ n''} \wedge mining\ (\sigma_{nid}(t\ n'')))$ **by** *simp*
thus *?thesis* **using** *trNxt-def*[*of t n'*] **by** *simp*
qed
qed

lemma *trnxt-pow-gr*:
assumes *trNxt t n*
and ¬ *trusted nid*
and *mining* $(\sigma_{nid}t\ n')$
and $\|nid\|_{t\ n'}$
and $n' > n$
shows $PoW\ t\ n' > PoW\ t\ n$
proof −
from *assms(1)* **have** $(\exists\, n'{\geq}n.\ PoW\ t\ n' > PoW\ t\ n \wedge (\forall\, n''{>}n.\ n'' \leq n' \longrightarrow \neg(\exists\, nid{\in}actUt\ t\ n''.\ \|nid\|_{t\ n''} \wedge mining\ (\sigma_{nid}(t\ n'')))))\ \vee\ (\forall\, n'{>}n.\ \neg(\exists\, nid{\in}actUt\ t\ n'.\ \|nid\|_{t\ n'} \wedge mining\ (\sigma_{nid}(t\ n'))))$ **using** *trNxt-def* **by** *simp*
moreover have $\neg(\forall\, n'{>}n.\ \neg(\exists\, nid{\in}actUt\ t\ n'.\ \|nid\|_{t\ n'} \wedge mining\ (\sigma_{nid}(t\ n'))))$
proof (*rule ccontr*)
assume ¬ ¬ $(\forall\, n'{>}n.\ \neg\ (\exists\, nid{\in}actUt\ t\ n'.\ \|nid\|_{t\ n'} \wedge mining\ (\sigma_{nid}t\ n')))$
hence $\forall\, n'{>}n.\ \neg\ (\exists\, nid{\in}actUt\ t\ n'.\ \|nid\|_{t\ n'} \wedge mining\ (\sigma_{nid}t\ n'))$ **by** *simp*
moreover from *assms* **have** $nid \in actUt\ t\ n'$ **using** *actUt-def* **by** *simp*
ultimately show *False* **using** *assms* **by** *simp*
qed
ultimately have $\exists\, n'{\geq}n.\ PoW\ t\ n' > PoW\ t\ n \wedge (\forall\, n''{>}n.\ n'' \leq n' \longrightarrow \neg(\exists\, nid{\in}actUt\ t\ n''.\ \|nid\|_{t\ n''} \wedge mining\ (\sigma_{nid}(t\ n''))))$ **by** *auto*
then obtain n'' **where** $n''\geq n$ **and** $PoW\ t\ n'' > PoW\ t\ n$ **and** *asmp*: $\forall\, n'''{>}n.\ n''' \leq n'' \longrightarrow \neg(\exists\, nid{\in}actUt\ t\ n'''.\ \|nid\|_{t\ n'''} \wedge mining\ (\sigma_{nid}(t\ n''')))$ **by** *auto*
moreover have $n'' < n'$
proof (*rule ccontr*)
assume $\neg n''{<}n'$
hence $n''{\geq}n'$ **by** *simp*
moreover from *assms* **have** $nid \in actUt\ t\ n'$ **using** *actUt-def* **by** *simp*

ultimately show *False* **using** *assms(3) assms(4) assms(5) asmp* **by** *simp*
qed
hence $PoW\ t\ n' \geq PoW\ t\ n''$ **using** *pow-mono* **by** *simp*
ultimately show *?thesis* **by** *simp*
qed

4.3.5 Secure Blockchains

lemma *ut-src-tr*:
assumes *prefix sbc* (*bc* ($\sigma_{nid'}t$ $\langle nid \Leftarrow t\rangle_n$))
and *build*: *mining* ($\sigma_{nid}t$ *n*) $\wedge$ *prefix* (*bc* ($\sigma_{nid}t$ *n*)) (*bc* ($\sigma_{nid'}t$ $\langle nid \Leftarrow t\rangle_n$) @ [*nid*]) $\vee$ $\neg$ *mining* ($\sigma_{nid}t$ *n*) $\wedge$ *prefix* (*bc* ($\sigma_{nid}t$ *n*)) (*bc* ($\sigma_{nid'}t$ $\langle nid \Leftarrow t\rangle_n$))
and $PoW\ t\ n >$ *length sbc* $\vee$ $PoW\ t\ n =$ *length sbc* $\wedge$ *trNxt t n*
shows *Suc* (*length* (*bc* ($\sigma_{nid}t$ *n*))) < $PoW\ t\ n$ $\vee$ *Suc* (*length* (*bc* ($\sigma_{nid}t$ *n*))) = $PoW\ t\ n$ $\wedge$ *trNxt t n* $\vee$ *prefix sbc* (*bc* ($\sigma_{nid}t$ *n*))
proof *cases*
assume *length* (*bc* ($\sigma_{nid}t$ *n*)) $\geq$ *length sbc*
with *build assms(1)* **show** *?thesis* **using** *prefix-length-prefix*[*of sbc bc* ($\sigma_{nid'}t$ $\langle nid \Leftarrow t\rangle_n$)] **by** *auto*
next
assume $\neg$ *length* (*bc* ($\sigma_{nid}t$ *n*)) $\geq$ *length sbc*
hence *length* (*bc* ($\sigma_{nid}t$ *n*)) < *length sbc* **by** *simp*
hence *Suc* (*length* (*bc* ($\sigma_{nid}t$ *n*))) < *length sbc* $\vee$ *Suc* (*length* (*bc* ($\sigma_{nid}t$ *n*))) = *length sbc* **by** *auto*
thus *?thesis*
proof
assume *Suc* (*length* (*bc* ($\sigma_{nid}t$ *n*))) < *length sbc*
with *assms(3)* **show** *?thesis* **by** *auto*
next
assume *asmp*: *Suc* (*length* (*bc* ($\sigma_{nid}t$ *n*))) = *length sbc*
from *assms(3)* **show** *?thesis*
proof
assume $PoW\ t\ n >$ *length sbc*
with *asmp* **show** *?thesis* **by** *simp*
next
assume $PoW\ t\ n =$ *length sbc* $\wedge$ *trNxt t n*
with *asmp* **show** *?thesis* **by** *simp*
qed
qed
qed

lemma *ut-src-ut-less*:
assumes $\neg$ *trusted nid*
and *Suc* (*length* (*bc* ($\sigma_{nid'}t$ $\langle nid \Leftarrow t\rangle_n$))) < $PoW\ t\ \langle nid \Leftarrow t\rangle_n$
and $\neg$*mining* ($\sigma_{nid}t$ *n*) $\wedge$ *prefix* (*bc* ($\sigma_{nid}t$ *n*)) (*bc* ($\sigma_{nid'}t$ $\langle nid \Leftarrow t\rangle_n$)) $\vee$ *mining* ($\sigma_{nid}t$ *n*) $\wedge$ *prefix* (*bc* ($\sigma_{nid}t$ *n*)) (*bc* ($\sigma_{nid'}t$ $\langle nid \Leftarrow t\rangle_n$) @ [*nid*])
and $\exists\, n' \geq n_S.\ n' < n \wedge \|nid\|_{t\ n'}$
and $\|nid\|_{t\ n}$
shows *Suc* (*length* (*bc* ($\sigma_{nid}t$ *n*))) < $PoW\ t\ n$ $\vee$ *Suc* (*length* (*bc* ($\sigma_{nid}t$ *n*))) = $PoW\ t\ n$ $\wedge$ *trNxt t n*
proof −
from *assms(2)* **have** *Suc* (*Suc* (*length* (*bc* ($\sigma_{nid'}t$ $\langle nid \Leftarrow t\rangle_n$)))) < $PoW\ t\ \langle nid \Leftarrow t\rangle_n$ $\vee$ *Suc* (*Suc*

(*length* (*bc* ($\sigma_{nid'}t$ $\langle nid \Leftarrow t\rangle_n$)))) = *PoW t* $\langle nid \Leftarrow t\rangle_n$ **by** *auto*
thus *?thesis*
proof
assume *Suc* (*Suc* (*length* (*bc* ($\sigma_{nid'}t$ $\langle nid \Leftarrow t\rangle_n$)))) < *PoW t* $\langle nid \Leftarrow t\rangle_n$
with *assms*(*3*) **have** *Suc* (*length* (*bc* ($\sigma_{nid}t$ *n*))) < *PoW t* $\langle nid \Leftarrow t\rangle_n$ **using** *prefix-length-le* **by** *fastforce*
moreover from *assms*(*4*) **have** $\langle nid \Leftarrow t\rangle_n \leq n$ **using** *lastAct-prop*(*2*) *order.strict-implies-order* **by** *blast*
ultimately show *?thesis* **using** *pow-mono*[*of* $\langle nid \Leftarrow t\rangle_n$ *n t*] **by** *simp*
next
assume *asmp*: *Suc* (*Suc* (*length* (*bc* ($\sigma_{nid'}t$ $\langle nid \Leftarrow t\rangle_n$)))) = *PoW t* $\langle nid \Leftarrow t\rangle_n$
from *assms* (*3*) **have** *prefix* (*bc* ($\sigma_{nid}t$ *n*)) (*bc* ($\sigma_{nid'}t$ $\langle nid \Leftarrow t\rangle_n$)) ∨ *bc* ($\sigma_{nid}t$ *n*) = (*bc* ($\sigma_{nid'}t$ $\langle nid \Leftarrow t\rangle_n$) @ [*nid*]) **by** *auto*
thus *?thesis*
proof
assume *prefix* (*bc* ($\sigma_{nid}t$ *n*)) (*bc* ($\sigma_{nid'}t$ $\langle nid \Leftarrow t\rangle_n$))
with *asmp* **have** *Suc* (*length* (*bc* ($\sigma_{nid}t$ *n*))) < *PoW t* $\langle nid \Leftarrow t\rangle_n$ **using** *prefix-length-le* **by** *fastforce*
moreover from *assms*(*4*) **have** $\langle nid \Leftarrow t\rangle_n \leq n$ **using** *lastAct-prop*(*2*) *order.strict-implies-order* **by** *blast*
ultimately show *?thesis* **using** *pow-mono*[*of* $\langle nid \Leftarrow t\rangle_n$ *n t*] **by** *simp*
next
assume *asmp2*: *bc* ($\sigma_{nid}t$ *n*) = *bc* ($\sigma_{nid'}t$ $\langle nid \Leftarrow t\rangle_n$) @ [*nid*]
show *?thesis*
proof *cases*
assume *PoW t n* > *PoW t* $\langle nid \Leftarrow t\rangle_n$
with *asmp asmp2* **show** *?thesis* **by** *simp*
next
assume ¬(*PoW t n* > *PoW t* $\langle nid \Leftarrow t\rangle_n$)
hence *PoW t n* ≤ *PoW t* $\langle nid \Leftarrow t\rangle_n$ **by** *simp*
moreover have $\langle nid \Leftarrow t\rangle_n < n$ **using** *assms*(*4*) *lastAct-prop*(*2*) **by** *auto*
hence *PoW t n* ≥ *PoW t* $\langle nid \Leftarrow t\rangle_n$ **using** *pow-mono* **by** *simp*
ultimately have *PoW t n* = *PoW t* $\langle nid \Leftarrow t\rangle_n$ **using** *pow-mono* **by** *simp*
with *asmp asmp2* **have** *Suc* (*length* (*bc* ($\sigma_{nid}t$ *n*))) = *PoW t n* **by** *simp*
moreover from *asmp2 assms*(*3*) **have** *mining* ($\sigma_{nid}t$ *n*) **by** *simp*
with ‹¬ *trusted nid*› **have** *trNxt t n* **using** *fair* **by** *simp*
ultimately show *?thesis* **by** *simp*
qed
qed
qed
qed

lemma *ut-src-ut-eq*:
assumes ¬ *trusted nid*
and *Suc* (*length* (*bc* ($\sigma_{nid'}t$ $\langle nid \Leftarrow t\rangle_n$))) = *PoW t* $\langle nid \Leftarrow t\rangle_n$
and *trNxt t* $\langle nid \Leftarrow t\rangle_n$
and ¬*mining* ($\sigma_{nid}t$ *n*) ∧ *prefix* (*bc* ($\sigma_{nid}t$ *n*)) (*bc* ($\sigma_{nid'}t$ $\langle nid \Leftarrow t\rangle_n$)) ∨ *mining* ($\sigma_{nid}t$ *n*) ∧ *prefix* (*bc* ($\sigma_{nid}t$ *n*)) (*bc* ($\sigma_{nid'}t$ $\langle nid \Leftarrow t\rangle_n$) @ [*nid*])
and $\exists n' \geq n_S.\ n' < n \wedge \|nid\|_{t\ n'}$

and $\|nid\|_{t\ n}$
shows $Suc\ (length\ (bc\ (\sigma_{nid} t\ n))) < PoW\ t\ n \vee Suc\ (length\ (bc\ (\sigma_{nid} t\ n))) = PoW\ t\ n \wedge trNxt\ t\ n$
proof *cases*
assume *mining* $(\sigma_{nid} t\ n)$
with *assms(1)* **have** *trNxt t n* **using** *fair* **by** *simp*
moreover from *assms(5)* **have** $\langle nid \Leftarrow t\rangle_n < n$ **using** *lastAct-prop(2)* **by** *blast*
with ‹*mining* $(\sigma_{nid} t\ n)$› ‹*trNxt t* $\langle nid \Leftarrow t\rangle_n$› *assms(1)* **have** $PoW\ t\ n > PoW\ t\ \langle nid \Leftarrow t\rangle_n$ **using** *trnxt-pow-gr*[*of t* $\langle nid \Leftarrow t\rangle_n$ *nid n*] *assms(6)* **by** *simp*
moreover from *assms (4)* **have** *prefix* $(bc\ (\sigma_{nid} t\ n))\ (bc\ (\sigma_{nid}{}'t\ \langle nid \Leftarrow t\rangle_n) @ [nid])$ **by** *auto*
hence $length\ (bc\ (\sigma_{nid} t\ n)) \leq length\ (bc\ (\sigma_{nid}{}'t\ \langle nid \Leftarrow t\rangle_n) @ [nid])$ **using** *prefix-length-le* **by** *metis*
hence $length\ (bc\ (\sigma_{nid} t\ n)) \leq Suc\ (length\ (bc\ (\sigma_{nid}{}'t\ \langle nid \Leftarrow t\rangle_n)))$ **by** *simp*
with *assms(2)* **have** $length\ (bc\ (\sigma_{nid} t\ n)) \leq PoW\ t\ \langle nid \Leftarrow t\rangle_n$ **by** *simp*
ultimately show *?thesis* **by** *auto*
next
assume $\neg mining\ (\sigma_{nid} t\ n)$
with *assms(4)* **have** $prefix\ (bc\ (\sigma_{nid} t\ n))\ (bc\ (\sigma_{nid}{}'t\ \langle nid \Leftarrow t\rangle_n))$ **by** *simp*
hence $length\ (bc\ (\sigma_{nid} t\ n)) < length\ (bc\ (\sigma_{nid}{}'t\ \langle nid \Leftarrow t\rangle_n)) \vee length\ (bc\ (\sigma_{nid} t\ n)) = length\ (bc\ (\sigma_{nid}{}'t\ \langle nid \Leftarrow t\rangle_n))$ **using** *le-neq-implies-less prefix-length-le* **by** *blast*
thus *?thesis*
proof
assume $length\ (bc\ (\sigma_{nid} t\ n)) < length\ (bc\ (\sigma_{nid}{}'t\ \langle nid \Leftarrow t\rangle_n))$
with *assms(2)* **have** $Suc\ (length\ (bc\ (\sigma_{nid} t\ n))) < PoW\ t\ \langle nid \Leftarrow t\rangle_n$ **by** *simp*
moreover from *assms(5)* **have** $\langle nid \Leftarrow t\rangle_n \leq n$ **using** *lastAct-prop(2) order.strict-implies-order* **by** *blast*
ultimately show *?thesis* **using** *pow-mono*[*of* $\langle nid \Leftarrow t\rangle_n$ *n t*] **by** *simp*
next
assume *asmp*: $length\ (bc\ (\sigma_{nid} t\ n)) = length\ (bc\ (\sigma_{nid}{}'t\ \langle nid \Leftarrow t\rangle_n))$
show *?thesis*
proof *cases*
assume $PoW\ t\ n > PoW\ t\ \langle nid \Leftarrow t\rangle_n$
with *asmp assms(2)* **show** *?thesis* **by** *simp*
next
assume $\neg(PoW\ t\ n > PoW\ t\ \langle nid \Leftarrow t\rangle_n)$
moreover have $\langle nid \Leftarrow t\rangle_n < n$ **using** *assms(5) lastAct-prop(2)* **by** *auto*
hence $PoW\ t\ n \geq PoW\ t\ \langle nid \Leftarrow t\rangle_n$ **using** *pow-mono* **by** *simp*
ultimately have $PoW\ t\ n = PoW\ t\ \langle nid \Leftarrow t\rangle_n$ **using** *pow-mono* **by** *simp*
with *asmp assms(2)* **have** $Suc\ (length\ (bc\ (\sigma_{nid} t\ n))) = PoW\ t\ n$ **by** *simp*
moreover from *assms(5)* **have** $\langle nid \Leftarrow t\rangle_n \leq n$ **using** ‹$\langle nid \Leftarrow t\rangle_n < n$› *nat-less-le* **by** *blast*
with ‹$PoW\ t\ n = PoW\ t\ \langle nid \Leftarrow t\rangle_n$› *assms(3)* **have** *trNxt t n* **using** *pow-eq-trnxt*[*of t* $\langle nid \Leftarrow t\rangle_n$ *n*] **by** *simp*
ultimately show *?thesis* **by** *simp*
qed
qed
qed

lemma *sbc-pow*:
fixes *t*::*nat*⇒*cnf* **and** n_S **and** *sbc* **and** *n*
assumes $\forall\, nid.\ bc\ (\sigma_{nid}(t\ (\langle nid \rightarrow t\rangle_{n_S}))) = sbc$
and *trNxt t* n_S

shows $n{\geq}n_S \Longrightarrow PoW\ t\ n > length\ sbc \vee PoW\ t\ n = length\ sbc \wedge trNxt\ t\ n$
proof (*induction n rule*: *dec-induct*)
case *base*
have $PoW\ t\ n_S = length\ sbc$
proof (*rule pow-eq*)
show $\exists\, tid{\in}actTr\ t\ n_S.\ length\ (bc\ (\sigma_{tid} t\ n_S)) = length\ sbc$
proof −
from *actTr* **have** $actTr\ t\ n_S \neq \{\}$ **using** *actTr-def* **by** *blast*
then obtain *tid* **where** $tid{\in}actTr\ t\ n_S$ **by** *auto*
moreover from ‹$tid{\in}actTr\ t\ n_S$› **have** $\|tid\|_t\ n_S$ **using** *actTr-def* **by** *simp*
hence $\langle tid \rightarrow t\rangle_{n_S}{=}n_S$ **by** (*simp add*: *nxtAct-active*)
with *assms(1)* **have** $length\ (bc\ (\sigma_{tid} t\ n_S)) = length\ sbc$ **by** *metis*
ultimately show *?thesis* **by** *auto*
qed
next
show $\forall\, tid{\in}actTr\ t\ n_S.\ length\ (bc\ (\sigma_{tid} t\ n_S)) \leq length\ sbc$
proof
fix *tid*
assume $tid \in actTr\ t\ n_S$
hence $\|tid\|_t\ n_S$ **using** *actTr-def* **by** *simp*
hence $\langle tid \rightarrow t\rangle_{n_S}{=}n_S$ **by** (*simp add*: *nxtAct-active*)
with *assms(1)* **show** $length\ (bc\ (\sigma_{tid} t\ n_S)) \leq length\ sbc$ **by** (*metis order-refl*)
qed
qed
with *assms(2)* **show** *?case* **by** *simp*
next
case (*step n*)
from *step.IH* **show** *?case*
proof
assume $length\ sbc < PoW\ t\ n$
with *pow-mono*[*of n Suc n t*] **show** *?thesis* **by** *simp*
next
assume $PoW\ t\ n = length\ sbc \wedge trNxt\ t\ n$
hence $PoW\ t\ n = length\ sbc$ **and** $trNxt\ t\ n$ **by** *auto*
show *?thesis*
proof *cases*
assume $PoW\ t\ n = PoW\ t\ (Suc\ n)$
with ‹$PoW\ t\ n = length\ sbc$› **have** $PoW\ t\ (Suc\ n) = length\ sbc$ **by** *simp*
moreover from ‹$PoW\ t\ n = PoW\ t\ (Suc\ n)$› ‹$trNxt\ t\ n$› **have***trNxt t* (*Suc n*) **using** *pow-eq-trnxt*[*of t n Suc n*] **by** *simp*
ultimately show *?thesis* **by** *simp*
next
assume $\neg\ PoW\ t\ n = PoW\ t\ (Suc\ n)$
moreover have $PoW\ t\ (Suc\ n) \geq PoW\ t\ n$ **using** *pow-mono* **by** *simp*
ultimately have $PoW\ t\ (Suc\ n) > PoW\ t\ n$ **by** *simp*
with ‹$PoW\ t\ n = length\ sbc$› **show** *?thesis* **by** *simp*
qed
qed
qed

theorem *blockchain-save*:
fixes t::$nat \Rightarrow cnf$ **and** n_S **and** sbc **and** n
assumes $\forall\, nid.\ bc\ (\sigma_{nid}(t\ (\langle nid \rightarrow t\rangle_{n_S}))) = sbc$
and $trNxt\ t\ n_S$
and $prems$:$n \geq n_S$
shows $n \geq n_S \Longrightarrow \forall\, nid.\ (trusted\ nid \wedge \|nid\|_{t\ n} \longrightarrow prefix\ sbc\ (bc\ (\sigma_{nid}(t\ n)))) \wedge (\neg trusted\ nid \wedge \|nid\|_{t\ n} \longrightarrow Suc\ (length\ (bc\ (\sigma_{nid}(t\ n)))) < PoW\ t\ n \vee Suc\ (length\ (bc\ (\sigma_{nid}(t\ n)))) = PoW\ t\ n \wedge trNxt\ t\ n \vee prefix\ sbc\ (bc\ (\sigma_{nid}(t\ n))))$
proof (*induction n rule*: *ge-induct*)
case (*step n*)
show *?case*
proof
fix *nid*
show $(trusted\ nid \wedge \|nid\|_{t\ n} \longrightarrow prefix\ sbc\ (bc\ (\sigma_{nid}(t\ n)))) \wedge$
$(\neg trusted\ nid \wedge \|nid\|_{t\ n} \longrightarrow Suc\ (length\ (bc\ (\sigma_{nid}(t\ n)))) < PoW\ t\ n \vee Suc\ (length\ (bc\ (\sigma_{nid}(t\ n)))) = PoW\ t\ n \wedge trNxt\ t\ n \vee prefix\ sbc\ (bc\ (\sigma_{nid}(t\ n))))$
proof (*rule conjI*)
show $trusted\ nid \wedge \|nid\|_{t\ n} \longrightarrow prefix\ sbc\ (bc\ (\sigma_{nid}t\ n))$
proof
assume $trusted\ nid \wedge \|nid\|_{t\ n}$
hence $trusted\ nid$ **and** $\|nid\|_{t\ n}$ **by** *auto*
show $prefix\ sbc\ (bc\ (\sigma_{nid}t\ n))$
proof *cases*
assume $\forall\, n'{\geq}n_S.\ n'{<}n \longrightarrow \neg\ \|nid\|_{t\ n'}$
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*
thus *?thesis* **using** *assms* **by** *auto*
next
assume $\neg\ (\forall\, n'{\geq}n_S.\ n'{<}n \longrightarrow \neg\ \|nid\|_{t\ n'})$
hence *act*: $\exists\, n'{\geq}n_S.\ n'{<}n \wedge \|nid\|_{t\ n'}$ **by** *simp*
hence $\langle nid \Leftarrow t\rangle_n \geq n_S$ **using** *lastActless* **by** *simp*
moreover from *act* **have** $\langle nid \Leftarrow t\rangle_n < n$ **using** *lastAct-prop*(*2*) **by** *auto*
moreover from *act* **have** $\|nid\|_{t\ \langle nid \Leftarrow t\rangle_n}$ **using** *lastAct-prop*(*1*) **by** *auto*
ultimately have $prefix\ sbc\ (bc\ (\sigma_{nid}t\ \langle nid \Leftarrow t\rangle_n))$ **using** *step.IH* $\langle trusted\ nid\rangle$ **by** *simp*
show *?thesis*
proof −
from $\langle trusted\ nid\rangle$ $\langle\|nid\|_{t\ n}\rangle$ *act* **have** $prefix\ (bc\ (\sigma_{nid}t\ \langle nid \Leftarrow t\rangle_n))\ (bc\ (\sigma_{nid}t\ n)) \vee (\exists\, nid'.\ \|nid'\|_{t\ \langle nid \Leftarrow t\rangle_n} \wedge length\ (bc\ (\sigma_{nid'}t\ \langle nid \Leftarrow t\rangle_n)) \geq length\ (MAX\ (pin\ (\sigma_{nid}t\ \langle nid \Leftarrow t\rangle_n))) \wedge prefix\ (bc\ (\sigma_{nid'}t\ \langle nid \Leftarrow t\rangle_n))\ (bc\ (\sigma_{nid}t\ n)))$ **using** *bhv-tr-context* **by** *simp*
thus *?thesis*
proof
assume $prefix\ (bc\ (\sigma_{nid}t\ \langle nid \Leftarrow t\rangle_n))\ (bc\ (\sigma_{nid}t\ n))$
with $\langle prefix\ sbc\ (bc\ (\sigma_{nid}t\ \langle nid \Leftarrow t\rangle_n))\rangle$ **show** *?thesis* **by** *simp*
next
assume $\exists\, nid'.\ \|nid'\|_{t\ \langle nid \Leftarrow t\rangle_n} \wedge length\ (bc\ (\sigma_{nid'}t\ \langle nid \Leftarrow t\rangle_n)) \geq length\ (MAX\ (pin\ (\sigma_{nid}t\ \langle nid \Leftarrow t\rangle_n))) \wedge prefix\ (bc\ (\sigma_{nid'}t\ \langle nid \Leftarrow t\rangle_n))\ (bc\ (\sigma_{nid}t\ n))$
then obtain nid' **where** $\|nid'\|_{t\ \langle nid \Leftarrow t\rangle_n}$ **and** *maxbc*: $length\ (bc\ (\sigma_{nid'}t\ \langle nid \Leftarrow t\rangle_n)) \geq length\ (MAX\ (pin\ (\sigma_{nid}t\ \langle nid \Leftarrow t\rangle_n)))$ **and** *pref*: $prefix\ (bc\ (\sigma_{nid'}t\ \langle nid \Leftarrow t\rangle_n))\ (bc\ (\sigma_{nid}t\ n))$ **by**

auto
show *?thesis*
proof *cases*
assume *trusted nid'*
with ‹$\|nid'\|_{t\ \langle nid \Leftarrow t\rangle_n}$› ‹$\langle nid \Leftarrow t\rangle_n \geq n_S$› ‹$\langle nid \Leftarrow t\rangle_n < n$› **have** *prefix sbc* (*bc* ($\sigma_{nid'} t\ \langle nid \Leftarrow t\rangle_n$)) **using** *step.IH* **by** *blast*
with *pref* **show** *?thesis* **by** *simp*
next
assume ¬ *trusted nid'*
with ‹$\|nid'\|_{t\ \langle nid \Leftarrow t\rangle_n}$› **have** *Suc* (*length* (*bc* ($\sigma_{nid'} t\ \langle nid \Leftarrow t\rangle_n$))) < *PoW t* $\langle nid \Leftarrow t\rangle_n$ ∨ *Suc* (*length* (*bc* ($\sigma_{nid'} t\ \langle nid \Leftarrow t\rangle_n$))) = *PoW t* $\langle nid \Leftarrow t\rangle_n$ ∧ *trNxt t* $\langle nid \Leftarrow t\rangle_n$ ∨ *prefix sbc* (*bc* ($\sigma_{nid'} t\ \langle nid \Leftarrow t\rangle_n$)) **using** *step.IH* ‹$\langle nid \Leftarrow t\rangle_n \geq n_S$› ‹$\langle nid \Leftarrow t\rangle_n < n$› **by** *simp*
hence *length* (*bc* ($\sigma_{nid'} t\ \langle nid \Leftarrow t\rangle_n$)) < *PoW t* $\langle nid \Leftarrow t\rangle_n$ ∨ *prefix sbc* (*bc* ($\sigma_{nid'} t\ \langle nid \Leftarrow t\rangle_n$)) **by** *auto*
thus *?thesis*
proof
assume *length* (*bc* ($\sigma_{nid'} t\ \langle nid \Leftarrow t\rangle_n$)) < *PoW t* $\langle nid \Leftarrow t\rangle_n$
moreover from *maxbc* **have** *length* (*bc* ($\sigma_{nid'} t\ \langle nid \Leftarrow t\rangle_n$)) ≥ *length* (*MAX* (*pin* ($\sigma_{nid} t\ \langle nid \Leftarrow t\rangle_n$))) **by** *simp*
with ‹*trusted nid*› ‹$\|nid\|_{t\ \langle nid \Leftarrow t\rangle_n}$› **have** *length* (*bc* ($\sigma_{nid'} t\ \langle nid \Leftarrow t\rangle_n$)) ≥ *PoW t* $\langle nid \Leftarrow t\rangle_n$ **using** *pow-le-max*[*of nid t* $\langle nid \Leftarrow t\rangle_n$] **by** *simp*
ultimately show *?thesis* **by** *simp*
next
assume *prefix sbc* (*bc* ($\sigma_{nid'} t\ \langle nid \Leftarrow t\rangle_n$))
with *pref* **show** *?thesis* **by** *simp*
qed
qed
qed
qed
qed
next
show ¬*trusted nid* ∧ $\|nid\|_{t\ n}$ ⟶ *Suc* (*length* (*bc* ($\sigma_{nid}(t\ n)$))) < *PoW t n* ∨ *Suc* (*length* (*bc* ($\sigma_{nid}(t\ n)$))) = *PoW t n* ∧ *trNxt t n* ∨ *prefix sbc* (*bc* ($\sigma_{nid}(t\ n)$))
proof
assume ¬ *trusted nid* ∧ $\|nid\|_{t\ n}$
hence ¬ *trusted nid* **and** $\|nid\|_{t\ n}$ **by** *auto*
show *Suc* (*length* (*bc* ($\sigma_{nid} t\ n$))) < *PoW t n* ∨ *Suc* (*length* (*bc* ($\sigma_{nid} t\ n$))) = *PoW t n* ∧ *trNxt t n* ∨ *prefix sbc* (*bc* ($\sigma_{nid} t\ n$))
proof *cases*
assume ∀ *n'*≥n_S. *n'*<*n* ⟶ ¬ $\|nid\|_{t\ n'}$
moreover from *step.hyps* **have** $n_S \leq n$ **by** *simp*
ultimately have $\langle nid \rightarrow t\rangle_{n_S} = n$ **using** ‹$\|nid\|_{t\ n}$› *nxtAct-eq*[*of* n_S *n nid t*] **by** *simp*
thus *?thesis* **using** *assms* **by** *auto*
next
assume ¬ (∀ *n'*≥n_S. *n'*<*n* ⟶ ¬ $\|nid\|_{t\ n'}$)
hence *act*: ∃ *n'*≥n_S. *n'*<*n* ∧ $\|nid\|_{t\ n'}$ **by** *simp*
hence $\langle nid \Leftarrow t\rangle_n \geq n_S$ **using** *lastActless* **by** *simp*
moreover from *act* **have** $\langle nid \Leftarrow t\rangle_n < n$ **using** *lastAct-prop*(*2*) **by** *auto*

moreover from *act* **have** $\|nid\|_{t\ \langle nid \Leftarrow t\rangle n}$ **using** *lastAct-prop(1)* **by** *auto*

ultimately have *Suc* (*length* (*bc* ($\sigma_{nid}t\ \langle nid \Leftarrow t\rangle n$))) < *PoW t* $\langle nid \Leftarrow t\rangle n$ ∨ *Suc* (*length* (*bc* ($\sigma_{nid}t\ \langle nid \Leftarrow t\rangle n$))) = *PoW t* $\langle nid \Leftarrow t\rangle n$ ∧ *trNxt t* $\langle nid \Leftarrow t\rangle n$ ∨ *prefix sbc* (*bc* ($\sigma_{nid}t\ \langle nid \Leftarrow t\rangle n$)) **using** *step.IH* ‹¬ *trusted nid*› **by** *simp*

thus *?thesis*

proof (*rule disjE3*)

assume *cass1*: *Suc* (*length* (*bc* ($\sigma_{nid}t\ \langle nid \Leftarrow t\rangle n$))) < *PoW t* $\langle nid \Leftarrow t\rangle n$

show *?thesis*

proof −

from ‹¬ *trusted nid*› ‹$\|nid\|_{t\ n}$› *act*

have (*mining* ($\sigma_{nid}t\ n$) ∧ *prefix* (*bc* ($\sigma_{nid}t\ n$)) (*bc* ($\sigma_{nid}t\ \langle nid \Leftarrow t\rangle n$) @ [*nid*]) ∨ ¬ *mining* ($\sigma_{nid}t\ n$) ∧ *prefix* (*bc* ($\sigma_{nid}t\ n$)) (*bc* ($\sigma_{nid}t\ \langle nid \Leftarrow t\rangle n$))) ∨ ((∃ *nid′*. $\|nid'\|_{t\ \langle nid \Leftarrow t\rangle n}$ ∧ (*mining* ($\sigma_{nid}t\ n$) ∧ *prefix* (*bc* ($\sigma_{nid}t\ n$)) (*bc* ($\sigma_{nid'}t\ \langle nid \Leftarrow t\rangle n$) @ [*nid*]) ∨ ¬ *mining* ($\sigma_{nid}t\ n$) ∧ *prefix* (*bc* ($\sigma_{nid}t\ n$)) (*bc* ($\sigma_{nid'}t\ \langle nid \Leftarrow t\rangle n$))))) **using** *bhv-ut-context*[*of nid t n* n_S] **by** *simp*

thus *?thesis*

proof

assume (*mining* ($\sigma_{nid}t\ n$) ∧ *prefix* (*bc* ($\sigma_{nid}t\ n$)) (*bc* ($\sigma_{nid}t\ \langle nid \Leftarrow t\rangle n$) @ [*nid*]) ∨ ¬ *mining* ($\sigma_{nid}t\ n$) ∧ *prefix* (*bc* ($\sigma_{nid}t\ n$)) (*bc* ($\sigma_{nid}t\ \langle nid \Leftarrow t\rangle n$)))

with ‹¬ *trusted nid*› *act cass1* ‹$\|nid\|_{t\ n}$› **show** *?thesis* **using** *ut-src-ut-less*[*of nid nid t n* n_S] **by** *auto*

next

assume ∃ *nid′*. $\|nid'\|_{t\ \langle nid \Leftarrow t\rangle n}$ ∧ (*mining* ($\sigma_{nid}t\ n$) ∧ *prefix* (*bc* ($\sigma_{nid}t\ n$)) (*bc* ($\sigma_{nid'}t\ \langle nid \Leftarrow t\rangle n$) @ [*nid*]) ∨ ¬ *mining* ($\sigma_{nid}t\ n$) ∧ *prefix* (*bc* ($\sigma_{nid}t\ n$)) (*bc* ($\sigma_{nid'}t\ \langle nid \Leftarrow t\rangle n$)))

then obtain *nid′* **where** $\|nid'\|_{t\ \langle nid \Leftarrow t\rangle n}$ **and** *build*: *mining* ($\sigma_{nid}t\ n$) ∧ *prefix* (*bc* ($\sigma_{nid}t\ n$)) (*bc* ($\sigma_{nid'}t\ \langle nid \Leftarrow t\rangle n$) @ [*nid*]) ∨ ¬ *mining* ($\sigma_{nid}t\ n$) ∧ *prefix* (*bc* ($\sigma_{nid}t\ n$)) (*bc* ($\sigma_{nid'}t\ \langle nid \Leftarrow t\rangle n$)) **by** *auto*

show *?thesis*

proof *cases*

assume *trusted nid′*

with ‹$\langle nid \Leftarrow t\rangle n \geq n_S$› ‹$\langle nid \Leftarrow t\rangle n < n$› ‹$\|nid'\|_{t\ \langle nid \Leftarrow t\rangle n}$› **have** *prefix sbc* (*bc* ($\sigma_{nid'}t\ \langle nid \Leftarrow t\rangle n$)) **using** *step.IH* **by** *simp*

moreover from *assms(1) assms(2) step.hyps* **have** *PoW t n* > *length sbc* ∨ *PoW t n* = *length sbc* ∧ *trNxt t n* **using** *sbc-pow*[*of t* n_S *sbc n*] **by** *simp*

ultimately show *?thesis* **using** *build ut-src-tr* **by** *simp*

next

assume ¬ *trusted nid′*

with ‹$\langle nid \Leftarrow t\rangle n \geq n_S$› ‹$\langle nid \Leftarrow t\rangle n < n$› ‹$\|nid'\|_{t\ \langle nid \Leftarrow t\rangle n}$› **have** *Suc* (*length* (*bc* ($\sigma_{nid'}t\ \langle nid \Leftarrow t\rangle n$))) < *PoW t* $\langle nid \Leftarrow t\rangle n$ ∨ *Suc* (*length* (*bc* ($\sigma_{nid'}t\ \langle nid \Leftarrow t\rangle n$))) = *PoW t* $\langle nid \Leftarrow t\rangle n$ ∧ *trNxt t* $\langle nid \Leftarrow t\rangle n$ ∨ *prefix sbc* (*bc* ($\sigma_{nid'}t\ \langle nid \Leftarrow t\rangle n$)) **using** *step.IH* **by** *simp*

thus *?thesis*

proof (*rule disjE3*)

assume *Suc* (*length* (*bc* ($\sigma_{nid'}t\ \langle nid \Leftarrow t\rangle n$))) < *PoW t* $\langle nid \Leftarrow t\rangle n$

with ‹¬ *trusted nid*› *act build* ‹$\|nid\|_{t\ n}$› **show** *?thesis* **using** *ut-src-ut-less*[*of nid nid′ t n* n_S] **by** *auto*

next

assume *Suc* (*length* (*bc* ($\sigma_{nid'}t\ \langle nid \Leftarrow t\rangle n$))) = *PoW t* $\langle nid \Leftarrow t\rangle n$ ∧ *trNxt t* $\langle nid \Leftarrow t\rangle n$

with ‹¬ *trusted nid*› *act build* ‹$\|nid\|_{t\ n}$› **show** *?thesis* **using** *ut-src-ut-eq*[*of nid nid' t n* n_S] **by** *auto*

next

assume *prefix sbc* (*bc* ($\sigma_{nid'}t$ ⟨*nid* ⇐ *t*⟩*n*))

moreover from *assms*(*1*) *assms*(*2*) *step.hyps* **have** *PoW t n* > *length sbc* ∨ *PoW t n* = *length sbc* ∧ *trNxt t n* **using** *sbc-pow*[*of t* n_S *sbc n*] **by** *simp*

ultimately show *?thesis* **using** *build ut-src-tr* **by** *simp*

qed

qed

qed

qed

next

assume *cass2*: *Suc* (*length* (*bc* ($\sigma_{nid}t$ ⟨*nid* ⇐ *t*⟩*n*))) = *PoW t* ⟨*nid* ⇐ *t*⟩*n* ∧ *trNxt t* ⟨*nid* ⇐ *t*⟩*n*

show *?thesis*

proof −

from ‹¬ *trusted nid*› ‹$\|nid\|_{t\ n}$› *act*

have (*mining* ($\sigma_{nid}t$ *n*) ∧ *prefix* (*bc* ($\sigma_{nid}t$ *n*)) (*bc* ($\sigma_{nid}t$ ⟨*nid* ⇐ *t*⟩*n*) @ [*nid*]) ∨ ¬ *mining* ($\sigma_{nid}t$ *n*) ∧ *prefix* (*bc* ($\sigma_{nid}t$ *n*)) (*bc* ($\sigma_{nid}t$ ⟨*nid* ⇐ *t*⟩*n*))) ∨ ((∃ *nid'*. $\|nid'\|_{t\ \langle nid \Leftarrow t\rangle n}$ ∧ (*mining* ($\sigma_{nid}t$ *n*) ∧ *prefix* (*bc* ($\sigma_{nid}t$ *n*)) (*bc* ($\sigma_{nid'}t$ ⟨*nid* ⇐ *t*⟩*n*) @ [*nid*]) ∨ ¬ *mining* ($\sigma_{nid}t$ *n*) ∧ *prefix* (*bc* ($\sigma_{nid}t$ *n*)) (*bc* ($\sigma_{nid'}t$ ⟨*nid* ⇐ *t*⟩*n*))))) **using** *bhv-ut-context*[*of nid t n* n_S] **by** *simp*

thus *?thesis*

proof

assume (*mining* ($\sigma_{nid}t$ *n*) ∧ *prefix* (*bc* ($\sigma_{nid}t$ *n*)) (*bc* ($\sigma_{nid}t$ ⟨*nid* ⇐ *t*⟩*n*) @ [*nid*]) ∨ ¬ *mining* ($\sigma_{nid}t$ *n*) ∧ *prefix* (*bc* ($\sigma_{nid}t$ *n*)) (*bc* ($\sigma_{nid}t$ ⟨*nid* ⇐ *t*⟩*n*)))

with ‹¬ *trusted nid*› *act cass2* ‹$\|nid\|_{t\ n}$› **show** *?thesis* **using** *ut-src-ut-eq*[*of nid nid t n* n_S] **by** *auto*

next

assume ∃ *nid'*. $\|nid'\|_{t\ \langle nid \Leftarrow t\rangle n}$ ∧ (*mining* ($\sigma_{nid}t$ *n*) ∧ *prefix* (*bc* ($\sigma_{nid}t$ *n*)) (*bc* ($\sigma_{nid'}t$ ⟨*nid* ⇐ *t*⟩*n*) @ [*nid*]) ∨ ¬ *mining* ($\sigma_{nid}t$ *n*) ∧ *prefix* (*bc* ($\sigma_{nid}t$ *n*)) (*bc* ($\sigma_{nid'}t$ ⟨*nid* ⇐ *t*⟩*n*)))

then obtain *nid'* **where** $\|nid'\|_{t\ \langle nid \Leftarrow t\rangle n}$ **and** *build*: *mining* ($\sigma_{nid}t$ *n*) ∧ *prefix* (*bc* ($\sigma_{nid}t$ *n*)) (*bc* ($\sigma_{nid'}t$ ⟨*nid* ⇐ *t*⟩*n*) @ [*nid*]) ∨ ¬ *mining* ($\sigma_{nid}t$ *n*) ∧ *prefix* (*bc* ($\sigma_{nid}t$ *n*)) (*bc* ($\sigma_{nid'}t$ ⟨*nid* ⇐ *t*⟩*n*)) **by** *auto*

show *?thesis*

proof *cases*

assume *trusted nid'*

with ‹⟨*nid* ⇐ *t*⟩*n* ≥ n_S› ‹⟨*nid* ⇐ *t*⟩*n* < *n*› ‹$\|nid'\|_{t\ \langle nid \Leftarrow t\rangle n}$› **have** *prefix sbc* (*bc* ($\sigma_{nid'}t$ ⟨*nid* ⇐ *t*⟩*n*)) **using** *step.IH* **by** *simp*

moreover from *assms*(*1*) *assms*(*2*) *step.hyps* **have** *PoW t n* > *length sbc* ∨ *PoW t n* = *length sbc* ∧ *trNxt t n* **using** *sbc-pow*[*of t* n_S *sbc n*] **by** *simp*

ultimately show *?thesis* **using** *build ut-src-tr* **by** *simp*

next

assume ¬ *trusted nid'*

with ‹⟨*nid* ⇐ *t*⟩*n* ≥ n_S› ‹⟨*nid* ⇐ *t*⟩*n* < *n*› ‹$\|nid'\|_{t\ \langle nid \Leftarrow t\rangle n}$› **have** *Suc* (*length* (*bc* ($\sigma_{nid'}t$ ⟨*nid* ⇐ *t*⟩*n*))) < *PoW t* ⟨*nid* ⇐ *t*⟩*n* ∨ *Suc* (*length* (*bc* ($\sigma_{nid'}t$ ⟨*nid* ⇐ *t*⟩*n*))) = *PoW t* ⟨*nid* ⇐ *t*⟩*n* ∧ *trNxt t* ⟨*nid* ⇐ *t*⟩*n* ∨ *prefix sbc* (*bc* ($\sigma_{nid'}t$ ⟨*nid* ⇐ *t*⟩*n*)) **using** *step.IH* **by** *simp*

thus *?thesis*

proof (*rule disjE3*)
assume *Suc* (*length* (*bc* ($\sigma_{nid'}t$ $\langle nid \Leftarrow t\rangle_n$))) < *PoW t* $\langle nid \Leftarrow t\rangle_n$
with ‹¬ *trusted nid*› *act build* ‹$\|nid\|_{t\ n}$› **show** *?thesis* **using** *ut-src-ut-less*[*of nid nid' t n* n_S] **by** *auto*
next
assume *Suc* (*length* (*bc* ($\sigma_{nid'}t$ $\langle nid \Leftarrow t\rangle_n$))) = *PoW t* $\langle nid \Leftarrow t\rangle_n$ ∧ *trNxt t* $\langle nid \Leftarrow t\rangle_n$
with ‹¬ *trusted nid*› *act build* ‹$\|nid\|_{t\ n}$› **show** *?thesis* **using** *ut-src-ut-eq*[*of nid nid' t n* n_S] **by** *auto*
next
assume *prefix sbc* (*bc* ($\sigma_{nid'}t$ $\langle nid \Leftarrow t\rangle_n$))
moreover from *assms(1) assms(2) step.hyps* **have** *PoW t n* > *length sbc* ∨ *PoW t n* = *length sbc* ∧ *trNxt t n* **using** *sbc-pow*[*of t* n_S *sbc n*] **by** *simp*
ultimately show *?thesis* **using** *build ut-src-tr* **by** *simp*
qed
qed
qed
qed
next
assume *cass3*: *prefix sbc* (*bc* ($\sigma_{nid}t$ $\langle nid \Leftarrow t\rangle_n$))
show *?thesis*
proof −
from ‹¬ *trusted nid*› ‹$\|nid\|_{t\ n}$› *act*
have (*mining* ($\sigma_{nid}t$ *n*) ∧ *prefix* (*bc* ($\sigma_{nid}t$ *n*)) (*bc* ($\sigma_{nid}t$ $\langle nid \Leftarrow t\rangle_n$) @ [*nid*]) ∨
¬ *mining* ($\sigma_{nid}t$ *n*) ∧ *prefix* (*bc* ($\sigma_{nid}t$ *n*)) (*bc* ($\sigma_{nid}t$ $\langle nid \Leftarrow t\rangle_n$))) ∨ ((∃ *nid'*. $\|nid'\|_{t\ \langle nid \Leftarrow t\rangle_n}$ ∧ (*mining* ($\sigma_{nid}t$ *n*) ∧ *prefix* (*bc* ($\sigma_{nid}t$ *n*)) (*bc* ($\sigma_{nid'}t$ $\langle nid \Leftarrow t\rangle_n$) @ [*nid*]) ∨ ¬ *mining* ($\sigma_{nid}t$ *n*) ∧ *prefix* (*bc* ($\sigma_{nid}t$ *n*)) (*bc* ($\sigma_{nid'}t$ $\langle nid \Leftarrow t\rangle_n$))))) **using** *bhv-ut-context*[*of nid t n* n_S] **by** *simp*
thus *?thesis*
proof
assume *mining* ($\sigma_{nid}t$ *n*) ∧ *prefix* (*bc* ($\sigma_{nid}t$ *n*)) (*bc* ($\sigma_{nid}t$ $\langle nid \Leftarrow t\rangle_n$) @ [*nid*]) ∨
¬ *mining* ($\sigma_{nid}t$ *n*) ∧ *prefix* (*bc* ($\sigma_{nid}t$ *n*)) (*bc* ($\sigma_{nid}t$ $\langle nid \Leftarrow t\rangle_n$))
moreover from *assms(1) assms(2) step.hyps* **have** *PoW t n* > *length sbc* ∨ *PoW t n* = *length sbc* ∧ *trNxt t n* **using** *sbc-pow*[*of t* n_S *sbc n*] **by** *simp*
ultimately show *?thesis* **using** *cass3 ut-src-tr* **by** *simp*
next
assume ∃ *nid'*. $\|nid'\|_{t\ \langle nid \Leftarrow t\rangle_n}$ ∧ (*mining* ($\sigma_{nid}t$ *n*) ∧ *prefix* (*bc* ($\sigma_{nid}t$ *n*)) (*bc* ($\sigma_{nid'}t$ $\langle nid \Leftarrow t\rangle_n$) @ [*nid*]) ∨ ¬ *mining* ($\sigma_{nid}t$ *n*) ∧ *prefix* (*bc* ($\sigma_{nid}t$ *n*)) (*bc* ($\sigma_{nid'}t$ $\langle nid \Leftarrow t\rangle_n$)))
then obtain *nid'* **where** $\|nid'\|_{t\ \langle nid \Leftarrow t\rangle_n}$ **and** *build*: *mining* ($\sigma_{nid}t$ *n*) ∧ *prefix* (*bc* ($\sigma_{nid}t$ *n*)) (*bc* ($\sigma_{nid'}t$ $\langle nid \Leftarrow t\rangle_n$) @ [*nid*]) ∨ ¬ *mining* ($\sigma_{nid}t$ *n*) ∧ *prefix* (*bc* ($\sigma_{nid}t$ *n*)) (*bc* ($\sigma_{nid'}t$ $\langle nid \Leftarrow t\rangle_n$)) **by** *auto*
show *?thesis*
proof *cases*
assume *trusted nid'*
with ‹$\langle nid \Leftarrow t\rangle_n \geq n_S$› ‹$\langle nid \Leftarrow t\rangle_n < n$› ‹$\|nid'\|_{t\ \langle nid \Leftarrow t\rangle_n}$› **have** *prefix sbc* (*bc* ($\sigma_{nid'}t$ $\langle nid \Leftarrow t\rangle_n$)) **using** *step.IH* **by** *simp*
moreover from *assms(1) assms(2) step.hyps* **have** *PoW t n* > *length sbc* ∨ *PoW t n* = *length sbc* ∧ *trNxt t n* **using** *sbc-pow*[*of t* n_S *sbc n*] **by** *simp*
ultimately show *?thesis* **using** *build ut-src-tr* **by** *simp*

```
            next
              assume ¬ trusted nid'
                with ‹⟨nid ⇐ t⟩n ≥ nS› ‹⟨nid ⇐ t⟩n < n› ‹‖nid'‖t ⟨nid ⇐ t⟩n› have Suc (length (bc
(σnid't ⟨nid ⇐ t⟩n))) < PoW t ⟨nid ⇐ t⟩n ∨ Suc (length (bc (σnid't ⟨nid ⇐ t⟩n))) = PoW t ⟨nid ⇐
t⟩n ∧ trNxt t ⟨nid ⇐ t⟩n ∨ prefix sbc (bc (σnid't ⟨nid ⇐ t⟩n)) using step.IH by simp
                thus ?thesis
                proof (rule disjE3)
                  assume Suc (length (bc (σnid't ⟨nid ⇐ t⟩n))) < PoW t ⟨nid ⇐ t⟩n
                  with ‹¬ trusted nid› act build ‹‖nid‖t n› show ?thesis using ut-src-ut-less[of nid nid'
t n nS] by auto
                next
                  assume Suc (length (bc (σnid't ⟨nid ⇐ t⟩n))) = PoW t ⟨nid ⇐ t⟩n ∧ trNxt t ⟨nid ⇐
t⟩n
                  with ‹¬ trusted nid› act build ‹‖nid‖t n› show ?thesis using ut-src-ut-eq[of nid nid' t
n nS] by auto
                next
                  assume prefix sbc (bc (σnid't ⟨nid ⇐ t⟩n))
                  moreover from assms(1) assms(2) step.hyps have PoW t n > length sbc ∨ PoW t n
= length sbc ∧ trNxt t n using sbc-pow[of t nS sbc n] by simp
                  ultimately show ?thesis using build ut-src-tr by simp
                qed
              qed
            qed
          qed
        qed
      qed
    qed
  qed
qed

end

end
```

References

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

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

[3] Diego Marmsoler. Dynamic architectures. *Archive of Formal Proofs*, pages 1–65, July 2017. Formal proof development.

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