Original

Nicole

August 24, 2025

Contents

theory *FormalLanguages*
  imports *Main*
begin

1 Formal Languages

type-synonym $'a\ word\ \ = 'a\ list$
type-synonym $'a\ language = 'a\ word\ set$

1.1 Words

abbreviation $emptyWord :: 'a\ word\ \ (\varepsilon)$ **where**
  $\varepsilon \equiv []$

abbreviation *concat* :: $'a$ *word* $\Rightarrow$ $'a$ *word* $\Rightarrow$ $'a$ *word* (**infixl** $\cdot$ *60*) **where**
$v{\cdot}w \equiv v@w$

abbreviation *length-of-word* :: $'a$ *word* $\Rightarrow$ *nat* ($|$-$|$ [*90*] *60*) **where**
$|w| \equiv$ *length* w

1.2 Alphabets

locale *Alphabet* =
fixes *Letters* :: $'a$ *set* (Σ)
assumes *not-empty*: $\Sigma \neq \{\}$
and *finite-letters*: *finite* Σ
begin

inductive-set *WordsOverAlphabet* :: $'a$ *word set* (Σ^* *100*) **where**
EmptyWord: $\varepsilon \in \Sigma^*$ |
Composed: $[\![a \in \Sigma;\ w \in \Sigma^*]\!] \Longrightarrow (a\#w) \in \Sigma^*$

lemma *word-over-alphabet-rev*:
fixes a :: $'a$
and w :: $'a$ *word*
assumes $([a] \cdot w) \in \Sigma^*$
shows $a \in \Sigma$ **and** $w \in \Sigma^*$
using *assms WordsOverAlphabet.cases*[*of* $a\#w$]
by *auto*

lemma *concat-words-over-an-alphabet*:
fixes v w :: $'a$ *word*
assumes $v \in \Sigma^*$
and $w \in \Sigma^*$
shows $(v \cdot w) \in \Sigma^*$
using *assms*
proof (*induct* v)
case *EmptyWord*
assume $w \in \Sigma^*$
thus $(\varepsilon \cdot w) \in \Sigma^*$
by *simp*
next
case (*Composed* a v)
assume $a \in \Sigma$
moreover assume $w \in \Sigma^* \Longrightarrow (v \cdot w) \in \Sigma^*$ **and** $w \in \Sigma^*$
hence $(v \cdot w) \in \Sigma^*$ **.**
ultimately show $((a\#v) \cdot w) \in \Sigma^*$
using *WordsOverAlphabet.Composed*[*of* a $v \cdot w$]
by *simp*
qed

lemma *split-a-word-over-an-alphabet*:

```
  fixes v w :: 'a word
  assumes (v · w) ∈ Σ*
  shows v ∈ Σ* and w ∈ Σ*
  using assms
proof (induct v)
  case Nil
  {
    case 1
    show ε ∈ Σ*
      using EmptyWord
      by simp
  next
    case 2
    assume ε · w ∈ Σ*
    thus w ∈ Σ*
      by simp
  }
next
  case (Cons a v)
  assume a#v · w ∈ Σ*
  hence A1: a ∈ Σ and A2: v · w ∈ Σ*
    using word-over-alphabet-rev[of a v · w]
    by simp-all
  assume IH1: v · w ∈ Σ* ⟹ v ∈ Σ* and IH2: v · w ∈ Σ* ⟹ w ∈ Σ*
  {
    case 1
    from A1 A2 IH1 show a#v ∈ Σ*
      using Composed[of a v]
      by simp
  next
    case 2
    from A2 IH2 show w ∈ Σ*
      by simp
  }
qed

end

end


theory CommunicatingAutomata
  imports FormalLanguages
begin

declare [[quick-and-dirty=true]]
```

2 Communicating Automata

2.1 Messages and Actions

datatype ('*information*, '*peer*) *message* =
  Message '*information* '*peer* '*peer* ($\text{-}^{\text{-}\rightarrow\text{-}}$ [*120*, *120*, *120*] *100*)

primrec *get-information* :: ('*information*, '*peer*) *message* ⇒ '*information* **where**
  get-information ($i^{p \rightarrow q}$) = *i*

primrec *get-sender* :: ('*information*, '*peer*) *message* ⇒ '*peer* **where**
  get-sender ($i^{p \rightarrow q}$) = *p*

primrec *get-receiver* :: ('*information*, '*peer*) *message* ⇒ '*peer* **where**
  get-receiver ($i^{p \rightarrow q}$) = *q*

value *get-information* ($i^{p \rightarrow q}$)
value *get-sender* ($i^{p \rightarrow q}$)
value *get-receiver* ($i^{p \rightarrow q}$)

datatype ('*information*, '*peer*) *action* =
  Output ('*information*, '*peer*) *message* (!⟨-⟩ [*120*] *100*) |
  Input ('*information*, '*peer*) *message* (*?*⟨-⟩ [*120*] *100*)

primrec *is-output* :: ('*information*, '*peer*) *action* ⇒ *bool* **where**
  is-output (!⟨*m*⟩) = *True* |
  is-output (*?*⟨*m*⟩) = *False*

abbreviation *is-input* :: ('*information*, '*peer*) *action* ⇒ *bool* **where**
  is-input a ≡ ¬(*is-output a*)

primrec *get-message* :: ('*information*, '*peer*) *action* ⇒ ('*information*, '*peer*) *message* **where**
  get-message (!⟨*m*⟩) = *m* |
  get-message (*?*⟨*m*⟩) = *m*

primrec *get-actor* :: ('*information*, '*peer*) *action* ⇒ '*peer* **where**
  get-actor (!⟨*m*⟩) = *get-sender m* |
  get-actor (*?*⟨*m*⟩) = *get-receiver m*

primrec *get-object* :: ('*information*, '*peer*) *action* ⇒ '*peer* **where**
  get-object (!⟨*m*⟩) = *get-receiver m* |
  get-object (*?*⟨*m*⟩) = *get-sender m*

abbreviation *get-info* :: ('*information*, '*peer*) *action* ⇒ '*information* **where**
  get-info a ≡ *get-information* (*get-message a*)

abbreviation *projection-on-outputs*
  :: ('*information*, '*peer*) *action word* ⇒ ('*information*, '*peer*) *action word* ($\text{-}\downarrow_{!}$ [*90*] *110*)

where
$w{\downarrow_!} \equiv$ *filter is-output w*

abbreviation *projection-on-outputs-language*
:: $('information, 'peer)$ *action language* $\Rightarrow$ $('information, 'peer)$ *action language* $(\text{-}{\downharpoonright_!}\ [120]\ 100)$
where
$L{\downharpoonright_!} \equiv \{w{\downarrow_!} \mid w.\ w \in L\}$

abbreviation *projection-on-inputs*
:: $('information, 'peer)$ *action word* $\Rightarrow$ $('information, 'peer)$ *action word* $(\text{-}{\downarrow_?}\ [90]\ 110)$
where
$w{\downarrow_?} \equiv$ *filter is-input w*

abbreviation *projection-on-inputs-language*
:: $('information, 'peer)$ *action language* $\Rightarrow$ $('information, 'peer)$ *action language* $(\text{-}{\downharpoonright_?}\ [120]\ 100)$
where
$L{\downharpoonright_?} \equiv \{w{\downarrow_?} \mid w.\ w \in L\}$

abbreviation *ignore-signs*
:: $('information, 'peer)$ *action word* $\Rightarrow$ $('information, 'peer)$ *message word* $(\text{-}{\downarrow_{!?}}\ [90]\ 110)$
where
$w{\downarrow_{!?}} \equiv$ *map get-message w*

abbreviation *ignore-signs-in-language*
:: $('information, 'peer)$ *action language* $\Rightarrow$ $('information, 'peer)$ *message language* $(\text{-}{\downharpoonright_{!?}}\ [90]\ 110)$ **where**
$L{\downharpoonright_{!?}} \equiv \{w{\downarrow_{!?}} \mid w.\ w \in L\}$

2.2 A Communicating Automaton

locale *CommunicatingAutomaton* =
fixes *peer* :: *'peer*
and *States* :: *'state set*
and *initial* :: *'state*
and *Messages* :: $('information, 'peer)$ *message set*
and *Transitions* :: $('state \times ('information, 'peer)\ action \times 'state)$ *set*
assumes *finite-states*: *finite States*
and *initial-state*: $initial \in States$
and *message-alphabet*: *Alphabet Messages*
and *well-formed-transition*: $\bigwedge s1\ a\ s2.\ (s1, a, s2) \in Transitions \Longrightarrow$
$s1 \in States \wedge get\text{-}message\ a \in Messages \wedge get\text{-}actor\ a = peer \wedge$
$get\text{-}object\ a \neq peer \wedge s2 \in States$
begin

inductive-set *ActionsOverMessages* :: ('*information*, '*peer*) *action set* **where**
AOMOutput: $m \in$ *Messages* $\Longrightarrow$!$\langle m \rangle \in$ *ActionsOverMessages* |
AOMInput: $m \in$ *Messages* $\Longrightarrow$?$\langle m \rangle \in$ *ActionsOverMessages*

lemma *ActionsOverMessages-is-finite*:
  shows *finite ActionsOverMessages*
  using *message-alphabet Alphabet.finite-letters*[*of Messages*]
  by (*simp add*: *ActionsOverMessages-def ActionsOverMessagesp.simps*)

lemma *action-is-action-over-message*:
  fixes *s1 s2* :: '*state*
    and *a* :: ('*information*, '*peer*) *action*
  assumes (*s1*, *a*, *s2*) $\in$ *Transitions*
  shows *a* $\in$ *ActionsOverMessages*
  using *assms*
proof (*induct a*)
  case (*Output m*)
  assume (*s1*, !$\langle m \rangle$, *s2*) $\in$ *Transitions*
  thus !$\langle m \rangle \in$ *ActionsOverMessages*
    using *well-formed-transition*[*of s1* !$\langle m \rangle$ *s2*] *AOMOutput*[*of m*]
    by *simp*
next
  case (*Input m*)
  assume (*s1*, ?$\langle m \rangle$, *s2*) $\in$ *Transitions*
  thus ?$\langle m \rangle \in$ *ActionsOverMessages*
    using *well-formed-transition*[*of s1* ?$\langle m \rangle$ *s2*] *AOMInput*[*of m*]
    by *simp*
qed

lemma *transition-set-is-finite*:
  shows *finite Transitions*
proof –
  have *Transitions* $\subseteq$ {(*s1*, *a*, *s2*). *s1* $\in$ *States* $\wedge$ *a* $\in$ *ActionsOverMessages* $\wedge$ *s2* $\in$ *States*}
    using *well-formed-transition action-is-action-over-message*
    by *blast*
  moreover have *finite* {(*s1*, *a*, *s2*). *s1* $\in$ *States* $\wedge$ *a* $\in$ *ActionsOverMessages* $\wedge$ *s2* $\in$ *States*}
    using *finite-states ActionsOverMessages-is-finite*
    by *simp*
  ultimately show *finite Transitions*
    using *finite-subset*[*of Transitions*
                {(*s1*, *a*, *s2*). *s1* $\in$ *States* $\wedge$ *a* $\in$ *ActionsOverMessages* $\wedge$ *s2* $\in$ *States*}]
    by *simp*
qed

inductive-set *Actions* :: ('*information*, '*peer*) *action set* (*Act*) **where**
ActOfTrans: (*s1*, *a*, *s2*) $\in$ *Transitions* $\Longrightarrow$ *a* $\in$ *Act*

lemma *Act-is-subset-of-ActionsOverMessages*:
shows $Act \subseteq ActionsOverMessages$
proof
fix a :: ($'information$, $'peer$) *action*
assume $a \in Act$
then obtain *s1 s2* **where** $(s1, a, s2) \in Transitions$
by (*auto simp add*: *Actions-def Actionsp.simps*)
hence *get-message* $a \in Messages$
using *well-formed-transition*[*of s1 a s2*]
by *simp*
thus $a \in ActionsOverMessages$
proof (*induct a*)
case (*Output m*)
assume *get-message* $(!\langle m\rangle) \in Messages$
thus $!\langle m\rangle \in ActionsOverMessages$
using *AOMOutput*[*of m*]
by *simp*
next
case (*Input m*)
assume *get-message* $(?\langle m\rangle) \in Messages$
thus $?\langle m\rangle \in ActionsOverMessages$
using *AOMInput*[*of m*]
by *simp*
qed
qed

lemma *Act-is-finite*:
shows *finite Act*
using *ActionsOverMessages-is-finite Act-is-subset-of-ActionsOverMessages*
finite-subset[*of Act ActionsOverMessages*]
by *simp*

inductive-set *CommunicationPartners* :: $'peer$ *set* **where**
CPAction: $(s1, a, s2) \in Transitions \Longrightarrow$ *get-object* $a \in CommunicationPartners$

lemma *ComunicationPartners-is-finite*:
shows *finite CommunicationPartners*
proof −
have $CommunicationPartners \subseteq \{p.\ \exists a.\ a \in ActionsOverMessages \wedge p =$ *get-object* $a\}$
using *action-is-action-over-message*
by (*auto simp add*: *CommunicationPartners-def CommunicationPartnersp.simps*)
moreover have *finite* $\{p.\ \exists a.\ a \in ActionsOverMessages \wedge p =$ *get-object* $a\}$
using *ActionsOverMessages-is-finite*
by *simp*
ultimately show *finite CommunicationPartners*
using *finite-subset*[*of CommunicationPartners*
$\{p.\ \exists a.\ a \in ActionsOverMessages \wedge p =$ *get-object* $a\}$]

by *simp*
qed

inductive-set *SendingToPeers* :: *'peer set* **where**
SPSend: ⟦(*s1*, *a*, *s2*) ∈ *Transitions*; *is-output a*⟧ ⟹ *get-object a* ∈ *SendingToPeers*

lemma *SendingToPeers-rev*:
fixes *p* :: *'peer*
assumes *p* ∈ *SendingToPeers*
shows ∃ *s1 a s2*. (*s1*, *a*, *s2*) ∈ *Transitions* ∧ *is-output a* ∧ *get-object a* = *p*
using *assms*
by (*induct*, *blast*)

lemma *SendingToPeers-is-subset-of-CommunicationPartners*:
shows *SendingToPeers* ⊆ *CommunicationPartners*
using *CommunicationPartners.intros SendingToPeersp.simps SendingToPeersp-SendingToPeers-eq*
by *auto*

inductive-set *ReceivingFromPeers* :: *'peer set* **where**
RPRecv: ⟦(*s1*, *a*, *s2*) ∈ *Transitions*; *is-input a*⟧ ⟹ *get-object a* ∈ *ReceivingFromPeers*

lemma *ReceivingFromPeers-rev*:
fixes *p* :: *'peer*
assumes *p* ∈ *ReceivingFromPeers*
shows ∃ *s1 a s2*. (*s1*, *a*, *s2*) ∈ *Transitions* ∧ *is-input a* ∧ *get-object a* = *p*
using *assms*
by (*induct*, *blast*)

lemma *ReceivingFromPeers-is-subset-of-CommunicationPartners*:
shows *ReceivingFromPeers* ⊆ *CommunicationPartners*
using *CommunicationPartners.intros ReceivingFromPeersp.simps*
ReceivingFromPeersp-ReceivingFromPeers-eq
by *auto*

abbreviation *step*
:: *'state* ⇒ (*'information*, *'peer*) *action* ⇒ *'state* ⇒ *bool* (- −-→ - [*90*, *90*, *90*]
110)
where
s1 −*a*→ *s2* ≡ (*s1*, *a*, *s2*) ∈ *Transitions*

inductive *run* :: *'state* ⇒ (*'information*, *'peer*) *action word* ⇒ *'state list* ⇒ *bool*
where
REmpty: *run s ε* ([]) |
RComposed: ⟦*run s0 w xs*; *last* (*s0*#*xs*) −*a*→ *s*⟧ ⟹ *run s0* (*w*·[*a*]) (*xs*@[*s*])

inductive-set *Traces* :: (*'information*, *'peer*) *action word set* **where**
STRun: *run initial w xs* ⟹ *w* ∈ *Traces*

abbreviation *Lang* :: (*'information*, *'peer*) *action language* **where**

$Lang \equiv Traces$

abbreviation *LangSend* :: ('*information*, '*peer*) *action language* **where**
$LangSend \equiv Lang\downharpoonright_!$

abbreviation *LangRecv* :: ('*information*, '*peer*) *action language* **where**
$LangRecv \equiv Lang\downharpoonright_?$

end

2.3 Network of Communicating Automata

locale *NetworkOfCA* =
fixes *automata* :: '*peer* ⇒ ('*state set* × '*state* ×
('*state* × ('*information*, '*peer*) *action* × '*state*) *set*) ($\mathcal{A}$ *1000*)
and *messages* :: ('*information*, '*peer*) *message set* ($\mathcal{M}$ *1000*)
assumes *finite-peers*: *finite* (*UNIV* :: '*peer set*)
and *automaton-of-peer*: $\bigwedge p.$ *CommunicatingAutomaton p* (*fst* ($\mathcal{A}$ *p*)) (*fst* (*snd* ($\mathcal{A}$ *p*))) $\mathcal{M}$
(*snd* (*snd* ($\mathcal{A}$ *p*)))
and *message-alphabet*: *Alphabet* $\mathcal{M}$
and *peers-of-message*: $\bigwedge m.\ m \in \mathcal{M} \Longrightarrow$ *get-sender m* ≠ *get-receiver m*
and *messages-used*: $\forall m \in \mathcal{M}.\ \exists s1\ a\ s2\ p.\ (s1,\ a,\ s2) \in snd\ (snd\ (\mathcal{A}\ p))$ ∧
m = *get-message a*
begin

abbreviation *get-states* :: '*peer* ⇒ '*state set* ($\mathcal{S}$ - [*90*] *110*) **where**
$\mathcal{S}(p) \equiv fst\ (\mathcal{A}\ p)$

abbreviation *get-initial-state* :: '*peer* ⇒ '*state* ($\mathcal{I}$ - [*90*] *110*) **where**
$\mathcal{I}(p) \equiv fst\ (snd\ (\mathcal{A}\ p))$

abbreviation *get-transitions*
:: '*peer* ⇒ ('*state* × ('*information*, '*peer*) *action* × '*state*) *set* ($\mathcal{R}$ - [*90*] *110*)
where
$\mathcal{R}(p) \equiv snd\ (snd\ (\mathcal{A}\ p))$

abbreviation *WordsOverMessages* :: ('*information*, '*peer*) *message word set* ($\mathcal{M}^*$ *100*) **where**
$\mathcal{M}^* \equiv$ *Alphabet.WordsOverAlphabet* $\mathcal{M}$

abbreviation *sendingToPeers-of-peer* :: '*peer* ⇒ '*peer set* ($\mathcal{P}_!$ - [*90*] *110*) **where**
$\mathcal{P}_!(p) \equiv$ *CommunicatingAutomaton.SendingToPeers* (*snd* (*snd* ($\mathcal{A}$ *p*)))

abbreviation *receivingFromPeers-of-peer* :: '*peer* ⇒ '*peer set* ($\mathcal{P}_?$ - [*90*] *110*)
where
$\mathcal{P}_?(p) \equiv$ *CommunicatingAutomaton.ReceivingFromPeers* (*snd* (*snd* ($\mathcal{A}$ *p*)))

abbreviation *step-of-peer*
:: *'state* ⇒ (*'information*, *'peer*) *action* ⇒ *'peer* ⇒ *'state* ⇒ *bool*
(- −-→- - [*90*, *90*, *90*, *90*] *110*) **where**
$s1\ -a\rightarrow p\ s2 \equiv (s1,\ a,\ s2) \in snd\ (snd\ (\mathcal{A}\ p))$

abbreviation *language-of-peer*
:: *'peer* ⇒ (*'information*, *'peer*) *action language* ($\mathcal{L}$ - [*90*] *110*) **where**
$\mathcal{L}(p) \equiv$ *CommunicatingAutomaton.Lang* (*fst* (*snd* ($\mathcal{A}$ *p*))) (*snd* (*snd* ($\mathcal{A}$ *p*)))

abbreviation *output-language-of-peer*
:: *'peer* ⇒ (*'information*, *'peer*) *action language* ($\mathcal{L}_!$ - [*90*] *110*) **where**
$\mathcal{L}_!(p) \equiv$ *CommunicatingAutomaton.LangSend* (*fst* (*snd* ($\mathcal{A}$ *p*))) (*snd* (*snd* ($\mathcal{A}$ *p*)))

abbreviation *input-language-of-peer*
:: *'peer* ⇒ (*'information*, *'peer*) *action language* ($\mathcal{L}_?$ - [*90*] *110*) **where**
$\mathcal{L}_?(p) \equiv$ *CommunicatingAutomaton.LangRecv* (*fst* (*snd* ($\mathcal{A}$ *p*))) (*snd* (*snd* ($\mathcal{A}$ *p*)))

2.4 Synchronous System

definition *is-sync-config* :: (*'peer* ⇒ *'state*) ⇒ *bool* **where**
is-sync-config $C \equiv (\forall p.\ C\ p \in \mathcal{S}(p))$

abbreviation *initial-sync-config* :: *'peer* ⇒ *'state* ($\mathcal{C}_{\mathcal{I}\mathbf{0}}$) **where**
$\mathcal{C}_{\mathcal{I}\mathbf{0}} \equiv \lambda p.\ \mathcal{I}(p)$

lemma *initial-configuration-is-synchronous-configuration*:
shows *is-sync-config* $\mathcal{C}_{\mathcal{I}\mathbf{0}}$
unfolding *is-sync-config-def*
proof *clarify*
fix *p* :: *'peer*
show $\mathcal{C}_{\mathcal{I}\mathbf{0}}(p) \in \mathcal{S}(p)$
using *automaton-of-peer*[*of p*]
CommunicatingAutomaton.initial-state[*of p* $\mathcal{S}$ *p* $\mathcal{C}_{\mathcal{I}\mathbf{0}}$ *p* $\mathcal{M}$ $\mathcal{R}$ *p*]
by *simp*
qed

inductive *sync-step*
:: (*'peer* ⇒ *'state*) ⇒ (*'information*, *'peer*) *action* ⇒ (*'peer* ⇒ *'state*) ⇒ *bool*
(- −⟨-, **0**⟩→ - [*90*, *90*, *90*] *110*) **where**
SynchStep: ⟦*is-sync-config C1*; $a = !\langle(i^{p \rightarrow q})\rangle$; $C1\ p\ -!\langle(i^{p \rightarrow q})\rangle\rightarrow p\ (C2\ p)$;
$C1\ q\ -?\langle(i^{p \rightarrow q})\rangle\rightarrow q\ (C2\ q)$; $\forall x.\ x \notin \{p,\ q\} \longrightarrow C1(x) = C2(x)$⟧ ⟹
$C1\ -\langle a,\ \mathbf{0}\rangle\rightarrow\ C2$

lemma *sync-step-rev*:
fixes *C1 C2* :: *'peer* ⇒ *'state*
and *a* :: (*'information*, *'peer*) *action*
assumes $C1\ -\langle a,\ \mathbf{0}\rangle\rightarrow\ C2$

shows *is-sync-config C1* **and** *is-sync-config C2* **and** $\exists\, i\ p\ q.\ a = !\langle(i^{p\rightarrow q})\rangle$
and *get-actor* $a \neq$ *get-object* a **and** *C1* (*get-actor* a) $-a\rightarrow$(*get-actor* a) (*C2* (*get-actor* a))
and $\exists\, m.\ a = !\langle m\rangle \wedge$ *C1* (*get-object* a) $-?\langle m\rangle\rightarrow$(*get-object* a) (*C2* (*get-object* a))
and $\forall\, x.\ x \notin \{$*get-actor* a, *get-object* $a\} \longrightarrow C1(x) = C2(x)$
using *assms*
proof *induct*
case (*SynchStep C1 a i p q C2*)
assume *A1*: *is-sync-config C1*
thus *is-sync-config C1* **.**
assume *A2*: $a = !\langle(i^{p\rightarrow q})\rangle$
thus $\exists\, i\ p\ q.\ a = !\langle(i^{p\rightarrow q})\rangle$
by *blast*
assume *A3*: *C1* p $-!\langle(i^{p\rightarrow q})\rangle\rightarrow p$ (*C2* p)
with *A2* **show** *C1* (*get-actor* a) $-a\rightarrow$(*get-actor* a) (*C2* (*get-actor* a))
by *simp*
have *A4*: *CommunicatingAutomaton* p ($\mathcal{S}$ p) ($\mathcal{I}$ p) $\mathcal{M}$ ($\mathcal{R}$ p)
using *automaton-of-peer*[*of* p]
by *simp*
with *A2 A3* **show** *get-actor* $a \neq$ *get-object* a
using *CommunicatingAutomaton.well-formed-transition*[*of* p $\mathcal{S}$ p $\mathcal{I}$ p $\mathcal{M}$ $\mathcal{R}$ p *C1* p a *C2* p]
by *auto*
assume *A5*: *C1* q $-?\langle(i^{p\rightarrow q})\rangle\rightarrow q$ (*C2* q)
with *A2* **show** $\exists\, m.\ a = !\langle m\rangle \wedge$ *C1* (*get-object* a) $-?\langle m\rangle\rightarrow$(*get-object* a) (*C2* (*get-object* a))
by *auto*
assume *A6*: $\forall\, x.\ x \notin \{p,\ q\} \longrightarrow$ *C1* x = *C2* x
with *A2* **show** $\forall\, x.\ x \notin \{$*get-actor* a, *get-object* $a\} \longrightarrow C1(x) = C2(x)$
by *simp*
show *is-sync-config C2*
unfolding *is-sync-config-def*
proof *clarify*
fix x :: *'peer*
show $C2(x) \in \mathcal{S}(x)$
proof (*cases* $x = p$)
assume $x = p$
with *A3 A4* **show** $C2(x) \in \mathcal{S}(x)$
using *CommunicatingAutomaton.well-formed-transition*[*of* p $\mathcal{S}$ p $\mathcal{I}$ p $\mathcal{M}$ $\mathcal{R}$ p *C1* p
$!\langle(i^{p\rightarrow q})\rangle$ *C2* p]
by *simp*
next
assume B: $x \neq p$
show $C2(x) \in \mathcal{S}(x)$
proof (*cases* $x = q$)
assume $x = q$
with *A5* **show** $C2(x) \in \mathcal{S}(x)$

using *automaton-of-peer*[*of q*]
CommunicatingAutomaton.well-formed-transition[*of q* $\mathcal{S}$ *q* $\mathcal{I}$ *q* $\mathcal{M}$ $\mathcal{R}$ *q C1 q*
$?\langle(i^{p \rightarrow q})\rangle$ *C2 q*]
by *simp*
next
assume $x \neq q$
with *A1 A6 B* **show** $C2(x) \in \mathcal{S}(x)$
unfolding *is-sync-config-def*
by (*metis empty-iff insertE*)
qed
qed
qed
qed

lemma *sync-step-output-rev*:
fixes *C1 C2* :: *'peer* ⇒ *'state*
and *i* :: *'information*
and *p q* :: *'peer*
assumes $C1 \, -\langle !\langle(i^{p \rightarrow q})\rangle, \mathbf{0}\rangle\rightarrow \, C2$
shows *is-sync-config C1* **and** *is-sync-config C2* **and** $p \neq q$ **and** $C1\ p \, -!\langle(i^{p \rightarrow q})\rangle\rightarrow p \, (C2\ p)$
and $C1\ q \, -?\langle(i^{p \rightarrow q})\rangle\rightarrow q \, (C2\ q)$ **and** $\forall x.\ x \notin \{p, q\} \longrightarrow C1(x) = C2(x)$
using *assms sync-step-rev*[*of C1* $!\langle(i^{p \rightarrow q})\rangle$ *C2*]
by *simp-all*

inductive *sync-run*
:: (*'peer* ⇒ *'state*) ⇒ (*'information*, *'peer*) *action word* ⇒ (*'peer* ⇒ *'state*) *list* ⇒ *bool*
where
SREmpty: *sync-run C* ε ([]) |
SRComposed: ⟦*sync-run C0 w xc*; *last* (*C0#xc*) $-\langle a, \mathbf{0}\rangle\rightarrow$ *C*⟧ ⟹ *sync-run C0* ($w \cdot [a]$) (*xc*@[*C*])

lemma *run-produces-synchronous-configurations*:
fixes *C C'* :: *'peer* ⇒ *'state*
and *w* :: (*'information*, *'peer*) *action word*
and *xc* :: (*'peer* ⇒ *'state*) *list*
assumes *sync-run C w xc*
and $C' \in$ *set xc*
shows *is-sync-config C'*
using *assms*
proof *induct*
case (*SREmpty C*)
assume $C' \in$ *set* []
hence *False*
by *simp*
thus *is-sync-config C'*
by *simp*

```
next
  case (SRComposed C0 w xc a C)
  assume A1: C' ∈ set xc ⟹ is-sync-config C' and A2: last (C0#xc) −⟨a, 0⟩→
C
     and A3: C' ∈ set (xc·[C])
  show is-sync-config C'
  proof (cases C = C')
    assume C = C'
    with A2 show is-sync-config C'
      using sync-step-rev(2)[of last (C0#xc) a C]
      by simp
  next
    assume C ≠ C'
    with A1 A3 show is-sync-config C'
      by simp
  qed
qed

lemma run-produces-no-inputs:
  fixes C C' :: 'peer ⇒ 'state
    and w    :: ('information, 'peer) action word
    and xc   :: ('peer ⇒ 'state) list
  assumes sync-run C w xc
  shows w↓! = w and w↓? = ε
  using assms
proof induct
  case (SREmpty C)
  show ε↓! = ε and ε↓? = ε
    by simp-all
next
  case (SRComposed C0 w xc a C)
  assume w↓! = w
  moreover assume last (C0#xc) −⟨a, 0⟩→ C
  hence A: is-output a
    using sync-step-rev(3)[of last (C0#xc) a C]
    by auto
  ultimately show (w · [a])↓! = w · [a]
    by simp
  assume w↓? = ε
  with A show (w · [a])↓? = ε
    by simp
qed

inductive-set SyncTraces :: ('information, 'peer) action language (𝒯₀ 120) where
STRun: sync-run 𝒞ℐ₀ w xc ⟹ w ∈ 𝒯₀

abbreviation SyncLang :: ('information, 'peer) action language (ℒ₀ 120) where
  ℒ₀ ≡ 𝒯₀
```

lemma *no-inputs-in-synchronous-communication*:
  shows $\mathcal{L}_\mathbf{0}\lfloor_! = \mathcal{L}_\mathbf{0}$ **and** $\mathcal{L}_\mathbf{0}\lfloor_? \subseteq \{\varepsilon\}$
proof $-$
  have $\forall\, w \in \mathcal{L}_\mathbf{0}.\ w{\downarrow_!} = w$
    using *SyncTraces.simps run-produces-no-inputs*(*1*)
    by *blast*
  thus $\mathcal{L}_\mathbf{0}\lfloor_! = \mathcal{L}_\mathbf{0}$
    by *force*
  have $\forall\, w \in \mathcal{L}_\mathbf{0}.\ w{\downarrow_?} = \varepsilon$
    using *SyncTraces.simps run-produces-no-inputs*(*2*)
    by *blast*
  thus $\mathcal{L}_\mathbf{0}\lfloor_? \subseteq \{\varepsilon\}$
    by *auto*
qed

2.5 Mailbox System

2.5.1 Semantics and Language

definition *is-mbox-config*
  :: ($'peer \Rightarrow ('state \times ('information, 'peer)\ message\ word)) \Rightarrow bool$ **where**
  $is\text{-}mbox\text{-}config\ C \equiv (\forall\, p.\ fst\ (C\ p) \in \mathcal{S}(p) \wedge snd\ (C\ p) \in \mathcal{M}^*)$

abbreviation *initial-mbox-config*
  :: $'peer \Rightarrow ('state \times ('information, 'peer)\ message\ word)$ $(\mathcal{C}_{\mathcal{I}\mathfrak{m}})$ **where**
  $\mathcal{C}_{\mathcal{I}\mathfrak{m}} \equiv \lambda p.\ (\mathcal{I}\ p, \varepsilon)$

lemma *initial-configuration-is-mailbox-configuration*:
  shows *is-mbox-config* $\mathcal{C}_{\mathcal{I}\mathfrak{m}}$
  unfolding *is-mbox-config-def*
proof *clarify*
  fix $p :: 'peer$
  show $fst\ (\mathcal{C}_{\mathcal{I}\mathbf{0}}\ p, \varepsilon) \in \mathcal{S}\ p \wedge snd\ (\mathcal{C}_{\mathcal{I}\mathbf{0}}\ p, \varepsilon) \in \mathcal{M}^*$
    using *automaton-of-peer*[*of p*] *message-alphabet Alphabet.EmptyWord*[*of* $\mathcal{M}$]
        CommunicatingAutomaton.initial-state[*of p* $\mathcal{S}$ *p* $\mathcal{I}$ *p* $\mathcal{M}$ $\mathcal{R}$ *p*]
    by *simp*
qed

definition *is-stable*
  :: ($'peer \Rightarrow ('state \times ('information, 'peer)\ message\ word)) \Rightarrow bool$ **where**
  $is\text{-}stable\ C \equiv is\text{-}mbox\text{-}config\ C \wedge (\forall\, p.\ snd\ (C\ p) = \varepsilon)$

lemma *initial-configuration-is-stable*:
  shows *is-stable* $\mathcal{C}_{\mathcal{I}\mathfrak{m}}$
  unfolding *is-stable-def* **using** *initial-configuration-is-mailbox-configuration*
  by *simp*

type-synonym *bound* = *nat option*

abbreviation *nat-bound* :: $nat \Rightarrow bound$ ($\mathcal{B}$ - [*90*] *110*) **where**

$\mathcal{B}\ k \equiv Some\ k$

abbreviation *unbounded* :: *bound* (∞ *100*) **where**
$\infty \equiv None$

primrec *is-bounded* :: *nat* $\Rightarrow$ *bound* $\Rightarrow$ *bool* (- $<_{\mathcal{B}}$ - [*90*, *90*] *110*) **where**
$n <_{\mathcal{B}} \infty = True$ |
$n <_{\mathcal{B}} \mathcal{B}\ k = (n < k)$

inductive *mbox-step*
:: ($'peer \Rightarrow ('state \times ('information, 'peer)\ message\ word)) \Rightarrow ('information, 'peer)$ *action* $\Rightarrow$
bound $\Rightarrow ('peer \Rightarrow ('state \times ('information, 'peer)\ message\ word)) \Rightarrow bool$
where
MboxSend: $[\![is\text{-}mbox\text{-}config\ C1;\ a = !\langle(i^{p \rightarrow q})\rangle;\ fst\ (C1\ p)\ -!\langle(i^{p \rightarrow q})\rangle\rightarrow p\ (fst\ (C2\ p));$
$snd\ (C1\ p) = snd\ (C2\ p);\ (\ |\ (snd\ (C1\ q))\ |\ ) <_{\mathcal{B}} k;$
$C2\ q = (fst\ (C1\ q),\ (snd\ (C1\ q)) \cdot [(i^{p \rightarrow q})]);\ \forall x.\ x \notin \{p,\ q\} \longrightarrow C1(x) = C2(x)]\!] \Longrightarrow$
$mbox\text{-}step\ C1\ a\ k\ C2$ |
MboxRecv: $[\![is\text{-}mbox\text{-}config\ C1;\ a = ?\langle(i^{p \rightarrow q})\rangle;\ fst\ (C1\ q)\ -?\langle(i^{p \rightarrow q})\rangle\rightarrow q\ (fst\ (C2\ q));$
$(snd\ (C1\ q)) = [(i^{p \rightarrow q})] \cdot snd\ (C2\ q);\ \forall x.\ x \neq q \longrightarrow C1(x) = C2(x)]\!] \Longrightarrow$
$mbox\text{-}step\ C1\ a\ k\ C2$

lemma *mbox-step-rev*:
fixes *C1 C2* :: $'peer \Rightarrow ('state \times ('information, 'peer)\ message\ word)$
and *a* :: $('information, 'peer)\ action$
and *k* :: *bound*
assumes *mbox-step C1 a k C2*
shows *is-mbox-config C1* **and** *is-mbox-config C2*
and $\exists\, i\ p\ q.\ a = !\langle(i^{p \rightarrow q})\rangle \vee a = ?\langle(i^{p \rightarrow q})\rangle$ **and** $get\text{-}actor\ a \neq get\text{-}object\ a$
and $fst\ (C1\ (get\text{-}actor\ a))\ -a\rightarrow(get\text{-}actor\ a)\ (fst\ (C2\ (get\text{-}actor\ a)))$
and $is\text{-}output\ a \Longrightarrow snd\ (C1\ (get\text{-}actor\ a)) = snd\ (C2\ (get\text{-}actor\ a))$
and $is\text{-}output\ a \Longrightarrow (\ |\ (snd\ (C1\ (get\text{-}object\ a)))\ |\ ) <_{\mathcal{B}} k$
and $is\text{-}output\ a \Longrightarrow C2\ (get\text{-}object\ a) =$
$(fst\ (C1\ (get\text{-}object\ a)),\ (snd\ (C1\ (get\text{-}object\ a))) \cdot [get\text{-}message\ a])$
and $is\text{-}input\ a \Longrightarrow (snd\ (C1\ (get\text{-}actor\ a))) = [get\text{-}message\ a] \cdot snd\ (C2\ (get\text{-}actor\ a))$
and $is\text{-}output\ a \Longrightarrow \forall x.\ x \notin \{get\text{-}actor\ a,\ get\text{-}object\ a\} \longrightarrow C1(x) = C2(x)$
and $is\text{-}input\ a \Longrightarrow \forall x.\ x \neq get\text{-}actor\ a \longrightarrow C1(x) = C2(x)$
using *assms*
proof *induct*
case (*MboxSend C1 a i p q C2 k*)
assume *A1*: *is-mbox-config C1*
thus *is-mbox-config C1* **.**
assume *A2*: $a = !\langle(i^{p \rightarrow q})\rangle$

thus $\exists\, i\ p\ q.\ a = !\langle(i^{p \rightarrow q})\rangle \vee a = ?\langle(i^{p \rightarrow q})\rangle$
  by *blast*
assume *A3*: $\mathit{fst}\ (\mathit{C1}\ p)\ {-!\langle(i^{p \rightarrow q})\rangle{\rightarrow}_p}\ (\mathit{fst}\ (\mathit{C2}\ p))$
with *A2* **show** $\mathit{fst}\ (\mathit{C1}\ (\mathit{get\text{-}actor}\ a))\ {-a{\rightarrow}}(\mathit{get\text{-}actor}\ a)\ (\mathit{fst}\ (\mathit{C2}\ (\mathit{get\text{-}actor}\ a)))$
  by *simp*
have *A4*: $\mathit{CommunicatingAutomaton}\ p\ (\mathcal{S}\ p)\ (\mathcal{I}\ p)\ \mathcal{M}\ (\mathcal{R}\ p)$
  using *automaton-of-peer*[*of p*]
  by *simp*
with *A2 A3* **show** $\mathit{get\text{-}actor}\ a \neq \mathit{get\text{-}object}\ a$
  using *CommunicatingAutomaton.well-formed-transition*[*of* $p\ \mathcal{S}\ p\ \mathcal{I}\ p\ \mathcal{M}\ \mathcal{R}\ p$ $\mathit{fst}\ (\mathit{C1}\ p)\ a$
    $\mathit{fst}\ (\mathit{C2}\ p)$]
  by *auto*
assume *A5*: $\mathit{snd}\ (\mathit{C1}\ p) = \mathit{snd}\ (\mathit{C2}\ p)$
with *A2* **show** $\mathit{is\text{-}output}\ a \Longrightarrow \mathit{snd}\ (\mathit{C1}\ (\mathit{get\text{-}actor}\ a)) = \mathit{snd}\ (\mathit{C2}\ (\mathit{get\text{-}actor}\ a))$
  by *simp*
assume $(\ |\mathit{snd}\ (\mathit{C1}\ q)|\ ) <_{\mathcal{B}} k$
with *A2* **show** $\mathit{is\text{-}output}\ a \Longrightarrow (\ |\ (\mathit{snd}\ (\mathit{C1}\ (\mathit{get\text{-}object}\ a)))\ |\ ) <_{\mathcal{B}} k$
  by *simp*
assume *A6*: $\mathit{C2}\ q = (\mathit{fst}\ (\mathit{C1}\ q), \mathit{snd}\ (\mathit{C1}\ q) \cdot [i^{p \rightarrow q}])$
with *A2* **show** $\mathit{is\text{-}output}\ a \Longrightarrow \mathit{C2}\ (\mathit{get\text{-}object}\ a) =$
    $(\mathit{fst}\ (\mathit{C1}\ (\mathit{get\text{-}object}\ a)), (\mathit{snd}\ (\mathit{C1}\ (\mathit{get\text{-}object}\ a))) \cdot [\mathit{get\text{-}message}\ a])$
  by *simp*
from *A2* **show** $\mathit{is\text{-}input}\ a \Longrightarrow (\mathit{snd}\ (\mathit{C1}\ (\mathit{get\text{-}actor}\ a))) = [\mathit{get\text{-}message}\ a] \cdot \mathit{snd}\ (\mathit{C2}\ (\mathit{get\text{-}actor}\ a))$
  by *simp*
assume *A7*: $\forall\, x.\ x \notin \{p,\ q\} \longrightarrow \mathit{C1}\ x = \mathit{C2}\ x$
with *A2* **show** $\mathit{is\text{-}output}\ a \Longrightarrow \forall\, x.\ x \notin \{\mathit{get\text{-}actor}\ a,\ \mathit{get\text{-}object}\ a\} \longrightarrow \mathit{C1}(x) = \mathit{C2}(x)$
  by *simp*
from *A2* **show** $\mathit{is\text{-}input}\ a \Longrightarrow \forall\, x.\ x \neq \mathit{get\text{-}actor}\ a \longrightarrow \mathit{C1}(x) = \mathit{C2}(x)$
  by *simp*
show *is-mbox-config C2*
  unfolding *is-mbox-config-def*
proof *clarify*
  fix $x :: {'}\mathit{peer}$
  show $\mathit{fst}\ (\mathit{C2}\ x) \in \mathcal{S}(x) \wedge \mathit{snd}\ (\mathit{C2}\ x) \in \mathcal{M}^*$
  proof (*cases* $x = p$)
    assume *B*: $x = p$
    with *A3 A4* **have** $\mathit{fst}\ (\mathit{C2}\ x) \in \mathcal{S}(x)$
      using *CommunicatingAutomaton.well-formed-transition*[*of* $p\ \mathcal{S}\ p\ \mathcal{I}\ p\ \mathcal{M}\ \mathcal{R}\ p\ \mathit{fst}\ (\mathit{C1}\ p)$
        $!\langle(i^{p \rightarrow q})\rangle\ \mathit{fst}\ (\mathit{C2}\ p)$]
      by *simp*
    moreover from *A1 A5 B* **have** $\mathit{snd}\ (\mathit{C2}\ x) \in \mathcal{M}^*$
      unfolding *is-mbox-config-def*
      by *metis*
    ultimately show $\mathit{fst}\ (\mathit{C2}\ x) \in \mathcal{S}(x) \wedge \mathit{snd}\ (\mathit{C2}\ x) \in \mathcal{M}^*$
      by *simp*

next
assume B: $x \neq p$
show $fst\ (C2\ x) \in \mathcal{S}(x) \wedge snd\ (C2\ x) \in \mathcal{M}^*$
proof $(cases\ x = q)$
assume $x = q$
moreover from $A1\ A6$ **have** $fst\ (C2\ q) \in \mathcal{S}(q)$
unfolding *is-mbox-config-def*
by *simp*
moreover from $A3\ A4$ **have** $i^{p \to q} \in \mathcal{M}$
using $CommunicatingAutomaton.well\text{-}formed\text{-}transition[of\ p\ \mathcal{S}\ p\ \mathcal{I}\ p\ \mathcal{M}$
$\mathcal{R}\ p$
$fst\ (C1\ p)\ !\langle(i^{p \to q})\rangle\ fst\ (C2\ p)]$
by *simp*
with $A1\ A6$ **have** $snd\ (C2\ q) \in \mathcal{M}^*$
unfolding *is-mbox-config-def*
using $message\text{-}alphabet\ Alphabet.EmptyWord[of\ \mathcal{M}]\ Alphabet.Composed[of$
$\mathcal{M}\ i^{p \to q}\ \varepsilon]$
$Alphabet.concat\text{-}words\text{-}over\text{-}an\text{-}alphabet[of\ \mathcal{M}\ snd\ (C1\ q)\ [i^{p \to q}]]$
by *simp*
ultimately show $fst\ (C2\ x) \in \mathcal{S}(x) \wedge snd\ (C2\ x) \in \mathcal{M}^*$
by *simp*
next
assume $x \neq q$
with $A1\ A7\ B$ **show** $fst\ (C2\ x) \in \mathcal{S}(x) \wedge snd\ (C2\ x) \in \mathcal{M}^*$
unfolding *is-mbox-config-def*
by (*metis insertE singletonD*)
qed
qed
qed
next
case (*MboxRecv C1 a i p q C2 k*)
assume $A1$: *is-mbox-config C1*
thus *is-mbox-config C1* **.**
assume $A2$: $a = ?\langle(i^{p \to q})\rangle$
thus $\exists\, i\ p\ q.\ a = !\langle(i^{p \to q})\rangle \vee a = ?\langle(i^{p \to q})\rangle$
by *blast*
assume $A3$: $fst\ (C1\ q)\ -?\langle(i^{p \to q})\rangle{\to}q\ (fst\ (C2\ q))$
with $A2$ **show** $fst\ (C1\ (get\text{-}actor\ a))\ -a{\to}(get\text{-}actor\ a)\ (fst\ (C2\ (get\text{-}actor\ a)))$
by *simp*
have $A4$: $CommunicatingAutomaton\ q\ (\mathcal{S}\ q)\ (\mathcal{I}\ q)\ \mathcal{M}\ (\mathcal{R}\ q)$
using $automaton\text{-}of\text{-}peer[of\ q]$
by *simp*
with $A2\ A3$ **show** $get\text{-}actor\ a \neq get\text{-}object\ a$
using $CommunicatingAutomaton.well\text{-}formed\text{-}transition[of\ q\ \mathcal{S}\ q\ \mathcal{I}\ q\ \mathcal{M}\ \mathcal{R}\ q$
$fst\ (C1\ q)\ a$
$fst\ (C2\ q)]$
by *auto*
from $A2$ **show** $is\text{-}output\ a \Longrightarrow snd\ (C1\ (get\text{-}actor\ a)) = snd\ (C2\ (get\text{-}actor\ a))$
by *simp*

from $A2$ **show** $is\text{-}output\ a \Longrightarrow (\ |\ (snd\ (C1\ (get\text{-}object\ a)))\ |\ ) <_{\mathcal{B}} k$
  by $simp$
from $A2$ **show** $is\text{-}output\ a \Longrightarrow C2\ (get\text{-}object\ a) =$
      $(fst\ (C1\ (get\text{-}object\ a)), (snd\ (C1\ (get\text{-}object\ a))) \cdot [get\text{-}message\ a])$
  by $simp$
assume $A5$: $snd\ (C1\ q) = [i^{p \rightarrow q}] \cdot snd\ (C2\ q)$
with $A2$ **show** $is\text{-}input\ a \Longrightarrow (snd\ (C1\ (get\text{-}actor\ a))) = [get\text{-}message\ a] \cdot snd$ $(C2\ (get\text{-}actor\ a))$
  by $simp$
from $A2$ **show** $is\text{-}output\ a \Longrightarrow \forall x.\ x \notin \{get\text{-}actor\ a,\ get\text{-}object\ a\} \longrightarrow C1(x)$ $= C2(x)$
  by $simp$
assume $A6$: $\forall x.\ x \neq q \longrightarrow C1\ x = C2\ x$
with $A2$ **show** $is\text{-}input\ a \Longrightarrow \forall x.\ x \neq get\text{-}actor\ a \longrightarrow C1(x) = C2(x)$
  by $simp$
show *is-mbox-config C2*
  unfolding *is-mbox-config-def*
proof *clarify*
  fix x :: *'peer*
  show $fst\ (C2\ x) \in \mathcal{S}(x) \wedge snd\ (C2\ x) \in \mathcal{M}^*$
  proof $(cases\ x = q)$
    assume B: $x = q$
    with *A3 A4* **have** $fst\ (C2\ x) \in \mathcal{S}(x)$
      using *CommunicatingAutomaton.well-formed-transition*[*of* $q\ \mathcal{S}\ q\ \mathcal{I}\ q\ \mathcal{M}\ \mathcal{R}$ q *fst* $(C1\ q)$
        $?\langle(i^{p \rightarrow q})\rangle$ *fst* $(C2\ q)$]
      by $simp$
    moreover from *A3 A4* **have** $i^{p \rightarrow q} \in \mathcal{M}$
      using *CommunicatingAutomaton.well-formed-transition*[*of* $q\ \mathcal{S}\ q\ \mathcal{I}\ q\ \mathcal{M}\ \mathcal{R}$ q *fst* $(C1\ q)$
        $?\langle(i^{p \rightarrow q})\rangle$ *fst* $(C2\ q)$]
      by $simp$
    with *A1 A5 B* **have** $snd\ (C2\ x) \in \mathcal{M}^*$
      unfolding *is-mbox-config-def*
      using *message-alphabet*
        Alphabet.split-a-word-over-an-alphabet(2)[*of* $\mathcal{M}$ $[i^{p \rightarrow q}]$ *snd* $(C2\ q)$]
      by *metis*
    ultimately show $fst\ (C2\ x) \in \mathcal{S}(x) \wedge snd\ (C2\ x) \in \mathcal{M}^*$
      by $simp$
  next
    assume $x \neq q$
    with *A1 A6* **show** $fst\ (C2\ x) \in \mathcal{S}(x) \wedge snd\ (C2\ x) \in \mathcal{M}^*$
      unfolding *is-mbox-config-def*
      by *metis*
  qed
qed
qed

lemma *mbox-step-output-rev*:

fixes *C1 C2* :: *'peer* ⇒ (*'state* × (*'information*, *'peer*) *message word*)
  and *i* :: *'information*
  and *p q* :: *'peer*
  and *k* :: *bound*
assumes *mbox-step C1* $(!\langle(i^{p\rightarrow q})\rangle)$ *k C2*
shows *is-mbox-config C1* **and** *is-mbox-config C2* **and** $p \neq q$
  and *fst* (*C1 p*) $-(!\langle(i^{p\rightarrow q})\rangle)\rightarrow p$ (*fst* (*C2 p*)) **and** *snd* (*C1 p*) = *snd* (*C2 p*)
  and (| (*snd* (*C1 q*)) |) $<_{\mathcal{B}}$ *k*
  and *C2 q* = (*fst* (*C1 q*), (*snd* (*C1 q*)) · [*get-message* $(!\langle(i^{p\rightarrow q})\rangle)$])
  and $\forall x.\ x \notin \{p,\ q\} \longrightarrow$ *C1*(*x*) = *C2*(*x*)
proof −
show *is-mbox-config C1*
  using *assms mbox-step-rev*(*1*)[*of C1* $!\langle(i^{p\rightarrow q})\rangle$ *k C2*]
  by *simp*
show *is-mbox-config C2*
  using *assms mbox-step-rev*(*2*)[*of C1* $!\langle(i^{p\rightarrow q})\rangle$ *k C2*]
  by *simp*
show $p \neq q$
  using *assms mbox-step-rev*(*4*)[*of C1* $!\langle(i^{p\rightarrow q})\rangle$ *k C2*]
  by *simp*
show *fst* (*C1 p*) $-!\langle(i^{p\rightarrow q})\rangle\rightarrow p$ (*fst* (*C2 p*))
  using *assms mbox-step-rev*(*5*)[*of C1* $!\langle(i^{p\rightarrow q})\rangle$ *k C2*]
  by *simp*
show *snd* (*C1 p*) = *snd* (*C2 p*)
  using *assms mbox-step-rev*(*6*)[*of C1* $!\langle(i^{p\rightarrow q})\rangle$ *k C2*]
  by *simp*
show (| (*snd* (*C1 q*)) |) $<_{\mathcal{B}}$ *k*
  using *assms mbox-step-rev*(*7*)[*of C1* $!\langle(i^{p\rightarrow q})\rangle$ *k C2*]
  by *fastforce*
show *C2 q* = (*fst* (*C1 q*), (*snd* (*C1 q*)) · [*get-message* $(!\langle(i^{p\rightarrow q})\rangle)$])
  using *assms mbox-step-rev*(*8*)[*of C1* $!\langle(i^{p\rightarrow q})\rangle$ *k C2*]
  by *simp*
show $\forall x.\ x \notin \{p,\ q\} \longrightarrow$ *C1*(*x*) = *C2*(*x*)
  using *assms mbox-step-rev*(*10*)[*of C1* $!\langle(i^{p\rightarrow q})\rangle$ *k C2*]
  by *simp*
qed

lemma *mbox-step-input-rev*:
fixes *C1 C2* :: *'peer* ⇒ (*'state* × (*'information*, *'peer*) *message word*)
  and *i* :: *'information*
  and *p q* :: *'peer*
  and *k* :: *bound*
assumes *mbox-step C1* $(?\langle(i^{p\rightarrow q})\rangle)$ *k C2*
shows *is-mbox-config C1* **and** *is-mbox-config C2* **and** $p \neq q$
  and *fst* (*C1 q*) $-?\langle(i^{p\rightarrow q})\rangle\rightarrow q$ (*fst* (*C2 q*)) **and** (*snd* (*C1 q*)) = $[i^{p\rightarrow q}]$ · *snd* (*C2 q*)
  and $\forall x.\ x \neq q \longrightarrow$ *C1*(*x*) = *C2*(*x*)
using *assms mbox-step-rev*[*of C1* $?\langle(i^{p\rightarrow q})\rangle$ *k C2*]
by *simp-all*

```
abbreviation mbox-step-bounded
  :: ('peer ⇒ ('state × ('information, 'peer) message word)) ⇒ ('information, 'peer)
action ⇒
      nat ⇒ ('peer ⇒ ('state × ('information, 'peer) message word)) ⇒ bool
     (- −⟨-, -⟩→ - [90, 90, 90, 90] 110) where
  C1 −⟨a, n⟩→ C2 ≡ mbox-step C1 a (Some n) C2

abbreviation mbox-step-unbounded
  :: ('peer ⇒ ('state × ('information, 'peer) message word)) ⇒ ('information, 'peer)
action ⇒
      ('peer ⇒ ('state × ('information, 'peer) message word)) ⇒ bool
     (- −⟨-, ∞⟩→ - [90, 90, 90] 110) where
  C1 −⟨a, ∞⟩→ C2 ≡ mbox-step C1 a None C2

inductive mbox-run
  :: ('peer ⇒ ('state × ('information, 'peer) message word)) ⇒ bound ⇒
      ('information, 'peer) action word ⇒
      ('peer ⇒ ('state × ('information, 'peer) message word)) list ⇒ bool where
MREmpty:        mbox-run C k ε ([]) |
MRComposedNat: ⟦mbox-run C0 (Some k) w xc; last (C0#xc) −⟨a, k⟩→ C⟧ ⟹
                mbox-run C0 (Some k) (w·[a]) (xc@[C]) |
MRComposedInf: ⟦mbox-run C0 None w xc; last (C0#xc) −⟨a, ∞⟩→ C⟧ ⟹
                mbox-run C0 None (w·[a]) (xc@[C])

lemma run-produces-mailbox-configurations:
  fixes C C' :: 'peer ⇒ ('state × ('information, 'peer) message word)
    and k    :: bound
    and w    :: ('information, 'peer) action word
    and xc   :: ('peer ⇒ ('state × ('information, 'peer) message word)) list
  assumes mbox-run C k w xc
      and C' ∈ set xc
    shows is-mbox-config C'
  using assms
proof induct
  case (MREmpty C k)
  assume C' ∈ set []
  hence False
    by simp
  thus is-mbox-config C'
    by simp
next
  case (MRComposedNat C0 k w xc a C)
  assume A1: C' ∈ set xc ⟹ is-mbox-config C' and A2: last (C0#xc) −⟨a, k⟩→
C
     and A3: C' ∈ set (xc · [C])
  show is-mbox-config C'
  proof (cases C = C')
    assume C = C'
```

with *A2* **show** *is-mbox-config* C'
using *mbox-step-rev*(*2*)[*of last* (*C0#xc*) *a Some k C*]
by *simp*
next
assume $C \neq C'$
with *A1 A3* **show** *is-mbox-config* C'
by *simp*
qed
next
case (*MRComposedInf C0 w xc a C*)
assume *A1*: $C' \in$ *set xc* $\Longrightarrow$ *is-mbox-config* C' **and** *A2*: *last* (*C0#xc*) $-\langle a, \infty\rangle\rightarrow$ C
and *A3*: $C' \in$ *set* (*xc* $\cdot$ [*C*])
show *is-mbox-config* C'
proof (*cases* $C = C'$)
assume $C = C'$
with *A2* **show** *is-mbox-config* C'
using *mbox-step-rev*(*2*)[*of last* (*C0#xc*) *a None C*]
by *simp*
next
assume $C \neq C'$
with *A1 A3* **show** *is-mbox-config* C'
by *simp*
qed
qed

inductive-set *MboxTraces*
:: *nat option* $\Rightarrow$ (*'information*, *'peer*) *action language* ($\mathcal{T}_{\_}$ [*100*] *120*)
for *k* :: *nat option* **where**
MTRun: *mbox-run* $\mathcal{C}_{\mathcal{I}\mathfrak{m}}$ *k w xc* $\Longrightarrow$ $w \in \mathcal{T}_k$

abbreviation *MboxLang* :: *bound* $\Rightarrow$ (*'information*, *'peer*) *action language* ($\mathcal{L}_{\_}$ [*100*] *120*)
where
$\mathcal{L}_k \equiv \{\ w{\downarrow_!} \mid w.\ w \in \mathcal{T}_k\ \}$

abbreviation *MboxLang-bounded-by-one* :: (*'information*, *'peer*) *action language* ($\mathcal{L}_{\mathbf{1}}$ *120*) **where**
$\mathcal{L}_{\mathbf{1}} \equiv \mathcal{L}_{\mathcal{B}\ 1}$

abbreviation *MboxLang-unbounded* :: (*'information*, *'peer*) *action language* ($\mathcal{L}_\infty$ *120*) **where**
$\mathcal{L}_\infty \equiv \mathcal{L}_\infty$

abbreviation *MboxLangSend* :: *bound* $\Rightarrow$ (*'information*, *'peer*) *action language* ($\mathcal{L}_{!\_}$ [*100*] *120*)
where
$\mathcal{L}_{!k} \equiv (\mathcal{L}_k)\rfloor_!$

abbreviation *MboxLangRecv* :: *bound* ⇒ (*'information*, *'peer*) *action language* ($\mathcal{L}_{?}$*-* [*100*] *120*)
where
$\mathcal{L}_{?k} \equiv (\mathcal{L}_k)\lfloor_?$

2.5.2 Language Hierarchy

theorem *sync-word-in-mbox-size-one*:
shows $\mathcal{L}_{\mathbf{0}} \subseteq \mathcal{L}_{\mathbf{1}}$
proof *clarify*
fix *v* :: (*'information*, *'peer*) *action word*
assume $v \in \mathcal{L}_{\mathbf{0}}$
then obtain *xcs C0* **where** *sync-run C0 v xcs* **and** $C0 = \mathcal{C}_{\mathcal{I}\mathbf{0}}$
using *SyncTracesp.simps SyncTracesp-SyncTraces-eq*
by *auto*
hence $\exists\, w\ xcm.\ mbox\text{-}run\ \mathcal{C}_{\mathcal{I}\mathfrak{m}}\ (\mathcal{B}\ 1)\ w\ xcm \wedge v = w{\downarrow_!} \wedge$
$(\forall\, p.\ last\ (\mathcal{C}_{\mathcal{I}\mathfrak{m}}\#xcm)\ p = (last\ (\mathcal{C}_{\mathcal{I}\mathbf{0}}\#xcs)\ p,\ \varepsilon))$
proof *induct*
case (*SREmpty C*)
have $mbox\text{-}run\ \mathcal{C}_{\mathcal{I}\mathfrak{m}}\ (\mathcal{B}\ 1)\ \varepsilon\ []$
using *MREmpty*[*of* $\mathcal{C}_{\mathcal{I}\mathfrak{m}}$ $\mathcal{B}$ *1*]
by *simp*
moreover have $\varepsilon = \varepsilon{\downarrow_!}$
by *simp*
moreover have $\forall\, p.\ \mathcal{C}_{\mathcal{I}\mathfrak{m}}\ p = (\mathcal{C}_{\mathcal{I}\mathbf{0}}\ p,\ \varepsilon)$
by *simp*
ultimately show $\exists\, w\ xcm.\ mbox\text{-}run\ \mathcal{C}_{\mathcal{I}\mathfrak{m}}\ (\mathcal{B}\ 1)\ w\ xcm \wedge \varepsilon = w{\downarrow_!} \wedge$
$(\forall\, p.\ last\ (\mathcal{C}_{\mathcal{I}\mathfrak{m}}\#xcm)\ p = (last\ [\mathcal{C}_{\mathcal{I}\mathbf{0}}]\ p,\ \varepsilon))$
by *fastforce*
next
case (*SRComposed C0 v xc a C*)
assume $C0 = \mathcal{C}_{\mathcal{I}\mathbf{0}} \Longrightarrow \exists\, w\ xcm.\ mbox\text{-}run\ \mathcal{C}_{\mathcal{I}\mathfrak{m}}\ (\mathcal{B}\ 1)\ w\ xcm \wedge v = w{\downarrow_!} \wedge$
$(\forall\, p.\ last\ (\mathcal{C}_{\mathcal{I}\mathfrak{m}}\#xcm)\ p = (last\ (\mathcal{C}_{\mathcal{I}\mathbf{0}}\#xc)\ p,\ \varepsilon))$
and *B1*: $C0 = \mathcal{C}_{\mathcal{I}\mathbf{0}}$
then obtain *w xcm* **where** *B2*: $mbox\text{-}run\ \mathcal{C}_{\mathcal{I}\mathfrak{m}}\ (\mathcal{B}\ 1)\ w\ xcm$ **and** *B3*: $v = w{\downarrow_!}$
and *B4*: $\forall\, p.\ last\ (\mathcal{C}_{\mathcal{I}\mathfrak{m}}\#xcm)\ p = (last\ (\mathcal{C}_{\mathcal{I}\mathbf{0}}\#xc)\ p,\ \varepsilon)$
by *blast*
assume $last\ (C0\#xc)\ -\langle a,\ \mathbf{0}\rangle\rightarrow\ C$
with *B1* **obtain** *C1* **where** *B5*: $C1 = last\ (\mathcal{C}_{\mathcal{I}\mathbf{0}}\#xc)$ **and** *B6*: $C1\ -\langle a,\ \mathbf{0}\rangle\rightarrow$
C
by *blast*
from *B6* **obtain** *i p q* **where** *B7*: $a = !\langle(i^{p\rightarrow q})\rangle$ **and** *B8*: $C1\ p\ -a\rightarrow p\ (C\ p)$
and *B9*: $C1\ q\ -(?\langle(i^{p\rightarrow q})\rangle)\rightarrow q\ (C\ q)$ **and** *B10*: $p \neq q$
and *B11*: $\forall\, x.\ x \notin \{p,\ q\} \longrightarrow C1\ x = C\ x$
using *sync-step-rev*[*of C1 a C*]
by *auto*
define *C2*::*'peer* ⇒ (*'state* × (*'information*, *'peer*) *message word*) **where**
C2-def: $C2 \equiv \lambda x.\ if\ x = p\ then\ (C\ p,\ \varepsilon)\ else\ (C1\ x,\ if\ x = q\ then\ [i^{p\rightarrow q}]$
$else\ \varepsilon)$

define $C3$::$'peer \Rightarrow ('state \times ('information,\ 'peer)\ message\ word)$ **where**
$C3\text{-}def$: $C3 \equiv \lambda x.\ (C\ x,\ \varepsilon)$
from $B2$ **have** $is\text{-}mbox\text{-}config\ (last\ (\mathcal{C}_{\mathcal{I}\mathfrak{m}}\#xcm))$
using $run\text{-}produces\text{-}mailbox\text{-}configurations[of\ \mathcal{C}_{\mathcal{I}\mathfrak{m}}\ \mathcal{B}\ 1\ w\ xcm\ last\ xcm]$
$initial\text{-}configuration\text{-}is\text{-}mailbox\text{-}configuration$
by $simp$
moreover from $B4\ B5\ B7\ B8$ **have** $fst\ (last\ (\mathcal{C}_{\mathcal{I}\mathfrak{m}}\#xcm)\ p)\ -(!\langle(i^{p\rightarrow q})\rangle)\rightarrow p\ (fst\ (C2\ p))$
unfolding $C2\text{-}def$
by $simp$
moreover from $B4$ **have** $snd\ (last\ (\mathcal{C}_{\mathcal{I}\mathfrak{m}}\#xcm)\ p) = snd\ (C2\ p)$
unfolding $C2\text{-}def$
by $simp$
moreover from $B4$ **have** $(\ |\ snd\ (last\ (\mathcal{C}_{\mathcal{I}\mathfrak{m}}\#xcm)\ q)\ |\ ) <_{\mathcal{B}} \mathcal{B}\ 1$
by $simp$
moreover from $B4\ B5\ B10$
have $C2\ q = (fst\ (last\ (\mathcal{C}_{\mathcal{I}\mathfrak{m}}\#xcm)\ q),\ snd\ (last\ (\mathcal{C}_{\mathcal{I}\mathfrak{m}}\#xcm)\ q) \cdot [i^{p\rightarrow q}])$
unfolding $C2\text{-}def$
by $simp$
moreover from $B4\ B5$ **have** $\forall x.\ x \notin \{p,\ q\} \longrightarrow last\ (\mathcal{C}_{\mathcal{I}\mathfrak{m}}\#xcm)\ x = C2\ x$
unfolding $C2\text{-}def$
by $simp$
ultimately have $B12$: $last\ (\mathcal{C}_{\mathcal{I}\mathfrak{m}}\#xcm)\ -\langle a,\ 1\rangle\rightarrow\ C2$
using $B7\ MboxSend[of\ last\ (\mathcal{C}_{\mathcal{I}\mathfrak{m}}\#xcm)\ !\langle(i^{p\rightarrow q})\rangle\ i\ p\ q\ C2\ \mathcal{B}\ 1]$
by $simp$
hence $is\text{-}mbox\text{-}config\ C2$
using $mbox\text{-}step\text{-}rev(2)[of\ last\ (\mathcal{C}_{\mathcal{I}\mathfrak{m}}\#xcm)\ a\ \mathcal{B}\ 1\ C2]$
by $simp$
moreover from $B9\ B10$ **have** $fst\ (C2\ q)\ -(?\langle(i^{p\rightarrow q})\rangle)\rightarrow q\ (fst\ (C3\ q))$
unfolding $C2\text{-}def\ C3\text{-}def$
by $simp$
moreover from $B10$ **have** $snd\ (C2\ q) = [i^{p\rightarrow q}] \cdot snd\ (C3\ q)$
unfolding $C2\text{-}def\ C3\text{-}def$
by $simp$
moreover from $B11$ **have** $\forall x.\ x \neq q \longrightarrow C2\ x = C3\ x$
unfolding $C2\text{-}def\ C3\text{-}def$
by $simp$
ultimately have $C2\ -\langle ?\langle(i^{p\rightarrow q})\rangle,\ 1\rangle\rightarrow\ C3$
using $MboxRecv[of\ C2\ ?\langle(i^{p\rightarrow q})\rangle\ i\ p\ q\ C3\ \mathcal{B}\ 1]$
by $simp$
with $B2\ B12$ **have** $mbox\text{-}run\ \mathcal{C}_{\mathcal{I}\mathfrak{m}}\ (\mathcal{B}\ 1)\ (w\cdot[a,\ ?\langle(i^{p\rightarrow q})\rangle])\ (xcm\cdot[C2,\ C3])$
using $MRComposedNat[of\ \mathcal{C}_{\mathcal{I}\mathfrak{m}}\ 1\ w\ xcm\ a\ C2]$
$MRComposedNat[of\ \mathcal{C}_{\mathcal{I}\mathfrak{m}}\ 1\ w\cdot[a]\ xcm\cdot[C2]\ ?\langle(i^{p\rightarrow q})\rangle\ C3]$
by $simp$
moreover from $B3\ B7$ **have** $v\cdot[a] = (w\cdot[a,\ ?\langle(i^{p\rightarrow q})\rangle])\downarrow_!$
using $filter\text{-}append[of\ is\text{-}output\ w\ [a,\ ?\langle(i^{p\rightarrow q})\rangle]]$
by $simp$
moreover have $\forall p.\ last\ (\mathcal{C}_{\mathcal{I}\mathfrak{m}}\#(xcm\cdot[C2,\ C3]))\ p = (last\ (\mathcal{C}_{\mathcal{I}\mathbf{0}}\#(xc\cdot[C]))\ p,\ \varepsilon)$

unfolding *C3-def*
by *simp*
ultimately show $\exists\, w\ xcm.\ mbox\text{-}run\ \mathcal{C}_{\mathcal{I}\mathfrak{m}}\ (\mathcal{B}\ 1)\ w\ xcm \wedge v{\cdot}[a] = w{\downarrow_!} \wedge$
$(\forall\, p.\ last\ (\mathcal{C}_{\mathcal{I}\mathfrak{m}}\#xcm)\ p = (last\ (\mathcal{C}_{\mathcal{I}\mathbf{0}}\#(xc{\cdot}[C]))\ p,\ \varepsilon))$
by *blast*
qed
then obtain *w xcm* **where** *A1*: $mbox\text{-}run\ \mathcal{C}_{\mathcal{I}\mathfrak{m}}\ (\mathcal{B}\ 1)\ w\ xcm$ **and** *A2*: $v = w{\downarrow_!}$
by *blast*
from *A1* **have** $w \in \mathcal{T}_{\mathcal{B}\ 1}$
by (*simp add*: *MboxTraces.intros*)
with *A2* **show** $\exists\, w.\ v = w{\downarrow_!} \wedge w \in \mathcal{T}_{\mathcal{B}\ 1}$
by *blast*
qed

3 Synchronisability

abbreviation *is-synchronisable* :: *bool* **where**
is-synchronisable $\equiv \mathcal{L}_\infty = \mathcal{L}_{\mathbf{0}}$

type-synonym *'a topology* = (*'a* × *'a*) *set*

inductive-set *Edges* :: *'peer topology* ($\mathcal{G}$ *110*) **where**
TEdge: $i^{p \to q} \in \mathcal{M} \Longrightarrow (p,\ q) \in \mathcal{G}$

lemma *Edges-rev*:
fixes *e* :: *'peer* × *'peer*
assumes $e \in \mathcal{G}$
shows $\exists\, i\ p\ q.\ i^{p \to q} \in \mathcal{M} \wedge e = (p,\ q)$
proof −
obtain *p q* **where** *A*: $e = (p,\ q)$
by *fastforce*
with *assms* **have** $(p,\ q) \in \mathcal{G}$
by *simp*
from *this A* **show** $\exists\, i\ p\ q.\ i^{p \to q} \in \mathcal{M} \wedge e = (p,\ q)$
by (*induct*, *blast*)
qed

abbreviation *Successors* :: *'peer topology* ⇒ *'peer* ⇒ *'peer set* (-⟨-→⟩ [*90*, *90*] *110*) **where**
$E\langle p \to \rangle \equiv \{q.\ (p,\ q) \in E\}$

abbreviation *Predecessors* :: *'peer topology* ⇒ *'peer* ⇒ *'peer set* (-⟨→-⟩ [*90*, *90*] *110*) **where**
$E\langle \to q \rangle \equiv \{p.\ (p,\ q) \in E\}$

3.1 Synchronisability is Deciable for Tree Topology in Mailbox Communication

3.1.1 Topology is a Tree

inductive *is-tree* :: $'peer\ set \Rightarrow 'peer\ topology \Rightarrow bool$ **where**
ITRoot: *is-tree* $\{p\}$ $\{\}$ |
ITNode: $[\![$*is-tree* P E; $p \in P$; $q \notin P]\!] \Longrightarrow$ *is-tree* (*insert* q P) (*insert* (p, q) E)

lemma *edge-on-peers-in-tree*:
  fixes P :: *'peer set*
    and E :: *'peer topology*
    and p q :: *'peer*
  assumes *is-tree* P E
      and $(p, q) \in E$
    shows $p \in P$ **and** $q \in P$
  using *assms*
proof *induct*
  case (*ITRoot* x)
  assume $(p, q) \in \{\}$
  thus $p \in \{x\}$ **and** $q \in \{x\}$
    by *simp-all*
next
  case (*ITNode* P E x y)
  assume $(p, q) \in E \Longrightarrow p \in P$ **and** $(p, q) \in E \Longrightarrow q \in P$ **and** $x \in P$
     and $(p, q) \in$ *insert* (x, y) E
  thus $p \in$ *insert* y P **and** $q \in$ *insert* y P
    by *auto*
qed

lemma *at-most-one-parent-in-tree*:
  fixes P :: *'peer set*
    and E :: *'peer topology*
    and p :: *'peer*
  assumes *is-tree* P E
  shows *card* $(E\langle\rightarrow p\rangle) \leq 1$
  using *assms*
proof *induct*
  case (*ITRoot* x)
  have $\{\}\langle\rightarrow p\rangle = \{\}$
    by *simp*
  thus *card* $(\{\}\langle\rightarrow p\rangle) \leq 1$
    by *simp*
next
  case (*ITNode* P E x y)
  assume *A1*: *is-tree* P E **and** *A2*: *card* $(E\langle\rightarrow p\rangle) \leq 1$ **and** *A3*: $y \notin P$
  show *card* (*insert* (x, y) $E\langle\rightarrow p\rangle) \leq 1$
  proof (*cases* $y = p$)
    assume B: $y = p$
    with *A1 A3* **have** $E\langle\rightarrow p\rangle = \{\}$

using *edge-on-peers-in-tree*(*2*)[*of P E - p*]
by *blast*
with *B* **have** *insert* (x, y) $E\langle\rightarrow p\rangle = \{x\}$
by *simp*
thus *card* (*insert* (x, y) $E\langle\rightarrow p\rangle$) $\leq$ *1*
by *simp*
next
assume $y \neq p$
hence *insert* (x, y) $E\langle\rightarrow p\rangle = E\langle\rightarrow p\rangle$
by *simp*
with *A2* **show** *card* (*insert* (x, y) $E\langle\rightarrow p\rangle$) $\leq$ *1*
by *simp*
qed
qed

abbreviation *tree-topology* :: *bool* **where**
tree-topology $\equiv$ *is-tree* (*UNIV* :: *'peer set*) ($\mathcal{G}$)

lemma *paranents-in-tree-is-ReceivedFromPeers*:
fixes *p* :: *'peer*
assumes *tree-topology*
shows $\mathcal{G}\langle\rightarrow p\rangle = \mathcal{P}_?(p)$
sorry

3.1.2 Topology is a Forest

inductive *is-forest* :: *'peer set* $\Rightarrow$ *'peer topology* $\Rightarrow$ *bool* **where**
IFSingle: *is-tree P E* $\Longrightarrow$ *is-forest P E* |
IFAddTree: $[\![$*is-forest P1 E1*; *is-tree P2 E2*; *P1* $\cap$ *P2* = $\{\}]\!]$ $\Longrightarrow$ *is-forest* (*P1* $\cup$ *P2*) (*E1* $\cup$ *E2*)

abbreviation *forest-topology* :: *bool* **where**
forest-topology $\equiv$ *is-forest* (*UNIV* :: *'peer set*) ($\mathcal{G}$)

end

end