Probabilistic Hierarchy

Johannes Hölzl   Andreas Lochbihler   Dmitriy Traytel

March 17, 2025

Contents

1 Bisimilarity

definition *bisimilar* **where**
  $bisimilar\ Q\ s1\ s2\ x\ y \equiv (\exists R.\ R\ x\ y \wedge (\forall x\ y.\ R\ x\ y \longrightarrow Q\ R\ (s1\ x)\ (s2\ y)))$

abbreviation $bisimilar\text{-}mc \equiv bisimilar\ (\lambda R.\ rel\text{-}pmf\ R)$
abbreviation $bisimilar\text{-}dlts \equiv bisimilar\ (\lambda R.\ rel\text{-}fun\ (=)\ (rel\text{-}option\ R))$
abbreviation $bisimilar\text{-}lts \equiv bisimilar\ (\lambda R.\ rel\text{-}bset\ (rel\text{-}prod\ (=)\ R))$
abbreviation $bisimilar\text{-}react \equiv bisimilar\ (\lambda R.\ rel\text{-}fun\ (=)\ (rel\text{-}option\ (rel\text{-}pmf\ R)))$
abbreviation $bisimilar\text{-}lmc \equiv bisimilar\ (\lambda R.\ rel\text{-}prod\ (=)\ (rel\text{-}pmf\ R))$
abbreviation $bisimilar\text{-}lmdp \equiv bisimilar\ (\lambda R.\ rel\text{-}prod\ (=)\ (rel\text{-}nebset\ (rel\text{-}pmf\ R)))$
abbreviation $bisimilar\text{-}gen \equiv bisimilar\ (\lambda R.\ rel\text{-}option\ (rel\text{-}pmf\ (rel\text{-}prod\ (=)\ R)))$

abbreviation $\mathit{bisimilar\text{-}str} \equiv \mathit{bisimilar}\ (\lambda R.\ \mathit{rel\text{-}sum}\ (\mathit{rel\text{-}pmf}\ R)\ (\mathit{rel\text{-}option}\ (\mathit{rel\text{-}prod}\ (=)\ R)))$
abbreviation $\mathit{bisimilar\text{-}alt} \equiv \mathit{bisimilar}\ (\lambda R.\ \mathit{rel\text{-}sum}\ (\mathit{rel\text{-}pmf}\ R)\ (\mathit{rel\text{-}bset}\ (\mathit{rel\text{-}prod}\ (=)\ R)))$
abbreviation $\mathit{bisimilar\text{-}sseg} \equiv \mathit{bisimilar}\ (\lambda R.\ \mathit{rel\text{-}bset}\ (\mathit{rel\text{-}prod}\ (=)\ (\mathit{rel\text{-}pmf}\ R)))$
abbreviation $\mathit{bisimilar\text{-}seg} \equiv \mathit{bisimilar}\ (\lambda R.\ \mathit{rel\text{-}bset}\ (\mathit{rel\text{-}pmf}\ (\mathit{rel\text{-}prod}\ (=)\ R)))$
abbreviation $\mathit{bisimilar\text{-}bun} \equiv \mathit{bisimilar}\ (\lambda R.\ \mathit{rel\text{-}pmf}\ (\mathit{rel\text{-}bset}\ (\mathit{rel\text{-}prod}\ (=)\ R)))$
abbreviation $\mathit{bisimilar\text{-}pz} \equiv \mathit{bisimilar}\ (\lambda R.\ \mathit{rel\text{-}bset}\ (\mathit{rel\text{-}pmf}\ (\mathit{rel\text{-}bset}\ (\mathit{rel\text{-}prod}\ (=)\ R))))$
abbreviation $\mathit{bisimilar\text{-}mg} \equiv \mathit{bisimilar}\ (\lambda R.\ \mathit{rel\text{-}bset}\ (\mathit{rel\text{-}pmf}\ (\mathit{rel\text{-}bset}\ (\mathit{rel\text{-}sum}\ (\mathit{rel\text{-}prod}\ (=)\ R)\ R))))$

2 Systems

codatatype $\mathit{mc} = \mathit{MC}\ \mathit{mc}\ \mathit{pmf}$
codatatype $'a\ \mathit{dlts} = \mathit{DLTS}\ 'a \Rightarrow 'a\ \mathit{dlts}\ \mathit{option}$
codatatype $('a,\ 'k)\ \mathit{lts} = \mathit{LTS}\ ('a \times ('a,\ 'k)\ \mathit{lts})\ \mathit{set}['k]$
codatatype $'a\ \mathit{react} = \mathit{React}\ 'a \Rightarrow 'a\ \mathit{react}\ \mathit{pmf}\ \mathit{option}$
codatatype $'a\ \mathit{lmc} = \mathit{LMC}\ 'a \times 'a\ \mathit{lmc}\ \mathit{pmf}$
codatatype $('a,\ 'k)\ \mathit{lmdp} = \mathit{LMDP}\ 'a \times ('a,\ 'k)\ \mathit{lmdp}\ \mathit{pmf}\ \mathit{set}!['k]$
codatatype $'a\ \mathit{gen} = \mathit{Gen}\ ('a \times 'a\ \mathit{gen})\ \mathit{pmf}\ \mathit{option}$
codatatype $'a\ \mathit{str} = \mathit{Str}\ 'a\ \mathit{str}\ \mathit{pmf} + ('a \times 'a\ \mathit{str})\ \mathit{option}$
codatatype $('a,\ 'k)\ \mathit{alt} = \mathit{Alt}\ ('a,\ 'k)\ \mathit{alt}\ \mathit{pmf} + ('a \times ('a,\ 'k)\ \mathit{alt})\ \mathit{set}['k]$
codatatype $('a,\ 'k)\ \mathit{sseg} = \mathit{SSeg}\ ('a \times ('a,\ 'k)\ \mathit{sseg}\ \mathit{pmf})\ \mathit{set}['k]$
codatatype $('a,\ 'k)\ \mathit{seg} = \mathit{Seg}\ ('a \times ('a,\ 'k)\ \mathit{seg})\ \mathit{pmf}\ \mathit{set}['k]$
codatatype $('a,\ 'k)\ \mathit{bun} = \mathit{Bun}\ (('a \times ('a,\ 'k)\ \mathit{bun})\ \mathit{set}['k])\ \mathit{pmf}$
codatatype $('a,\ 'k1,\ 'k2)\ \mathit{pz} = \mathit{PZ}\ (('a \times ('a,\ 'k1,\ 'k2)\ \mathit{pz})\ \mathit{set}['k1])\ \mathit{pmf}\ \mathit{set}['k2]$
codatatype $('a,\ 'k1,\ 'k2)\ \mathit{mg} = \mathit{MG}\ (('a \times ('a,\ 'k1,\ 'k2)\ \mathit{mg} + ('a,\ 'k1,\ 'k2)\ \mathit{mg})\ \mathit{set}['k1])\ \mathit{pmf}\ \mathit{set}['k2]$

3 Unfolds

primcorec $\mathit{unfold\text{-}mc} :: ('a \Rightarrow 'a\ \mathit{pmf}) \Rightarrow 'a \Rightarrow \mathit{mc}$ **where**
$\mathit{unfold\text{-}mc}\ s\ x = \mathit{MC}\ (\mathit{map\text{-}pmf}\ (\mathit{unfold\text{-}mc}\ s)\ (s\ x))$

primcorec $\mathit{unfold\text{-}dlts} :: ('a \Rightarrow 'b \Rightarrow 'a\ \mathit{option}) \Rightarrow 'a \Rightarrow 'b\ \mathit{dlts}$ **where**
$\mathit{unfold\text{-}dlts}\ s\ x = \mathit{DLTS}\ (\mathit{map\text{-}option}\ (\mathit{unfold\text{-}dlts}\ s)\ o\ s\ x)$

primcorec $\mathit{unfold\text{-}lts} :: ('a \Rightarrow ('b \times 'a)\ \mathit{set}['k]) \Rightarrow 'a \Rightarrow ('b,\ 'k)\ \mathit{lts}$ **where**
$\mathit{unfold\text{-}lts}\ s\ x = \mathit{LTS}\ (\mathit{map\text{-}bset}\ (\mathit{map\text{-}prod}\ \mathit{id}\ (\mathit{unfold\text{-}lts}\ s))\ (s\ x))$

primcorec $\mathit{unfold\text{-}react} :: ('a \Rightarrow 'b \Rightarrow 'a\ \mathit{pmf}\ \mathit{option}) \Rightarrow 'a \Rightarrow 'b\ \mathit{react}$ **where**
$\mathit{unfold\text{-}react}\ s\ x = \mathit{React}\ (\mathit{map\text{-}option}\ (\mathit{map\text{-}pmf}\ (\mathit{unfold\text{-}react}\ s))\ o\ s\ x)$

primcorec $\mathit{unfold\text{-}lmc} :: ('a \Rightarrow 'b \times 'a\ \mathit{pmf}) \Rightarrow 'a \Rightarrow 'b\ \mathit{lmc}$ **where**
$\mathit{unfold\text{-}lmc}\ s\ x = \mathit{LMC}\ (\mathit{map\text{-}prod}\ \mathit{id}\ (\mathit{map\text{-}pmf}\ (\mathit{unfold\text{-}lmc}\ s))\ (s\ x))$

primcorec $\mathit{unfold\text{-}lmdp} :: ('a \Rightarrow 'b \times 'a\ \mathit{pmf}\ \mathit{set}!['k]) \Rightarrow 'a \Rightarrow ('b,\ 'k)\ \mathit{lmdp}$ **where**

unfold-lmdp s x = LMDP (map-prod id (map-nebset (map-pmf (unfold-lmdp s))) (s x))

primcorec *unfold-gen* :: $('a \Rightarrow (('b \times 'a)\ pmf)\ option) \Rightarrow 'a \Rightarrow 'b\ gen$ **where**
unfold-gen s x = Gen (map-option (map-pmf (map-prod id (unfold-gen s))) (s x))

primcorec *unfold-str* :: $('a \Rightarrow 'a\ pmf + ('b \times 'a)\ option) \Rightarrow 'a \Rightarrow 'b\ str$ **where**
unfold-str s x = Str (map-sum (map-pmf (unfold-str s)) (map-option (map-prod id (unfold-str s))) (s x))

primcorec *unfold-alt* :: $('a \Rightarrow 'a\ pmf + ('b \times 'a)\ set['k]) \Rightarrow 'a \Rightarrow ('b, 'k)\ alt$ **where**
unfold-alt s x = Alt (map-sum (map-pmf (unfold-alt s)) (map-bset (map-prod id (unfold-alt s))) (s x))

primcorec *unfold-sseg* :: $('a \Rightarrow ('b \times 'a\ pmf)\ set['k]) \Rightarrow 'a \Rightarrow ('b, 'k)\ sseg$ **where**
unfold-sseg s x = SSeg (map-bset (map-prod id (map-pmf (unfold-sseg s))) (s x))

primcorec *unfold-seg* :: $('a \Rightarrow (('b \times 'a)\ pmf)\ set['k]) \Rightarrow 'a \Rightarrow ('b, 'k)\ seg$ **where**
unfold-seg s x = Seg (map-bset (map-pmf (map-prod id (unfold-seg s))) (s x))

primcorec *unfold-bun* :: $('a \Rightarrow (('b \times 'a)\ set['k])\ pmf) \Rightarrow 'a \Rightarrow ('b, 'k)\ bun$ **where**
unfold-bun s x = Bun (map-pmf (map-bset (map-prod id (unfold-bun s))) (s x))

primcorec *unfold-pz* :: $('a \Rightarrow (('b \times 'a)\ set['k1])\ pmf\ set['k2]) \Rightarrow 'a \Rightarrow ('b, 'k1, 'k2)\ pz$ **where**
unfold-pz s x = PZ (map-bset (map-pmf (map-bset (map-prod id (unfold-pz s)))) (s x))

primcorec *unfold-mg* :: $('a \Rightarrow (('b \times 'a + 'a)\ set['k1])\ pmf\ set['k2]) \Rightarrow 'a \Rightarrow ('b, 'k1, 'k2)\ mg$ **where**
unfold-mg s x = MG (map-bset (map-pmf (map-bset (map-sum (map-prod id (unfold-mg s)) (unfold-mg s)))) (s x))

4 Embeddings

abbreviation (*input*) *react-of-dlts-emb dlts* ≡ *map-option return-pmf o dlts*
abbreviation (*input*) *lts-of-dlts-emb* ≡ *bgraph*
abbreviation (*input*) *sseg-of-react-emb* ≡ *bgraph*
abbreviation (*input*) *gen-of-lmc-emb* ≡ *Some o case-prod (map-pmf o Pair)*
abbreviation (*input*) *lmdp-of-lmc-emb* ≡ *map-prod id nebsingleton*
abbreviation (*input*) *sseg-of-lmdp-emb* ≡ ($\lambda(a, X)$. *map-bset (Pair a) (bset-of-nebset X)*)
abbreviation (*input*) *sseg-of-lts-emb* ≡ *map-bset (map-prod id return-pmf)*
abbreviation (*input*) *ssegopt-of-alt-emb* ≡ *case-sum*
(*map-bset (Pair None) o bsingleton*)
(*map-bset (map-prod Some return-pmf)*)
abbreviation (*input*) *bunopt-of-alt-emb* ≡ *case-sum*

(*map-pmf* (*bsingleton o Pair None*))
(*map-pmf* (*map-bset* (*map-prod Some id*)) *o return-pmf*)
abbreviation (*input*) *segopt-of-seg-emb* ≡ *map-bset* (*map-pmf* (*map-prod Some id*))
abbreviation (*input*) *ssegopt-of-sseg-emb* ≡ *map-bset* (*map-prod Some id*)
abbreviation (*input*) *bunopt-of-bun-emb* ≡ *map-pmf* (*map-bset* (*map-prod Some id*))
abbreviation (*input*) *pzopt-of-pz-emb* ≡ *map-bset* (*map-pmf* (*map-bset* (*map-prod Some id*)))
abbreviation (*input*) *seg-of-sseg-emb* ≡ *map-bset* (*case-prod* (*map-pmf o Pair*))
abbreviation (*input*) *pz-of-seg-emb* ≡ *map-bset* (*map-pmf bsingleton*)
abbreviation (*input*) *pz-of-bun-emb* ≡ *bsingleton*
abbreviation (*input*) *seg-of-gen-emb* ≡ *bset-of-option*
abbreviation (*input*) *bun-of-lts-emb* ≡ *return-pmf*
abbreviation (*input*) *bun-of-gen-emb* ≡ *case-option* (*return-pmf bempty*) (*map-pmf bsingleton*)
abbreviation (*input*) *str-of-mc-emb* ≡ *Inl*
abbreviation (*input*) *alt-of-str-emb* ≡ *map-sum id bset-of-option*
abbreviation (*input*) *pzopt-of-mg-emb* ≡ *map-bset* (*map-pmf* (*map-bset* (*case-sum* (*map-prod Some id*) (*Pair None*))))
abbreviation (*input*) *mg-of-pzopt-emb* ≡ *map-bset* (*map-pmf* (*map-bset* (λ(*a*, *s*). *case-option* (*Inr s*) (λ*a*. (*Inl* (*a*, *s*))) *a*)))

Obsolete edges (susumed by transitive ones)

abbreviation (*input*) *mg-of-pz-emb* ≡ *map-bset* (*map-pmf* (*map-bset Inl*))
abbreviation (*input*) *mg-of-alt1-emb* ≡ *case-sum*
(*map-bset* (*map-pmf* (*map-bset Inr o bsingleton*)) *o bsingleton*)
(*map-bset* (*map-pmf* (*map-bset Inl o bsingleton*) *o return-pmf*))
abbreviation (*input*) *mg-of-alt2-emb* ≡ *case-sum*
(*map-bset* (*map-pmf* (*map-bset Inr o bsingleton*)) *o bsingleton*)
(*map-bset* (*map-pmf* (*map-bset Inl*) *o return-pmf*) *o bsingleton*)
abbreviation (*input*) *pz-of-alt1-emb* ≡ *case-sum*
(*map-bset* (*map-pmf* (*map-bset* (*Pair None*) *o bsingleton*)) *o bsingleton*)
(*map-bset* (*map-pmf* (*map-bset* (*map-prod Some id*) *o bsingleton*) *o return-pmf*))
abbreviation (*input*) *pz-of-alt2-emb* ≡ *case-sum*
(*map-bset* (*map-pmf* (*map-bset* (*Pair None*) *o bsingleton*)) *o bsingleton*)
(*map-bset* (*map-pmf* (*map-bset* (*map-prod Some id*)) *o return-pmf*) *o bsingleton*)

definition *react-of-dlts* :: *'a dlts* ⇒ *'a react* **where**
[*simp*]: *react-of-dlts* = *unfold-react* (*react-of-dlts-emb o un-DLTS*)

definition *lts-of-dlts* :: *'a dlts* ⇒ (*'a*, *'a set*) *lts* **where**
[*simp*]: *lts-of-dlts* = *unfold-lts* (*lts-of-dlts-emb o un-DLTS*)

definition *sseg-of-react* :: *'a react* ⇒ (*'a*, *'a set*) *sseg* **where**
[*simp*]: *sseg-of-react* = *unfold-sseg* (*sseg-of-react-emb o un-React*)

definition *lmdp-of-lmc* :: *'a lmc* ⇒ (*'a*, *'k*) *lmdp* **where**

[*simp*]: *lmdp-of-lmc* = *unfold-lmdp* (*lmdp-of-lmc-emb o un-LMC*)

definition *gen-of-lmc* :: *'a lmc* ⇒ *'a gen* **where**
[*simp*]: *gen-of-lmc* = *unfold-gen* (*gen-of-lmc-emb o un-LMC*)

definition *sseg-of-lmdp* :: (*'a*, *'k*) *lmdp* ⇒ (*'a*, *'k*) *sseg* **where**
[*simp*]: *sseg-of-lmdp* = *unfold-sseg* (*sseg-of-lmdp-emb o un-LMDP*)

definition *sseg-of-lts* :: (*'a*, *'k*) *lts* ⇒ (*'a*, *'k*) *sseg* **where**
[*simp*]: *sseg-of-lts* = *unfold-sseg* (*sseg-of-lts-emb o un-LTS*)

definition *ssegopt-of-alt* :: (*'a*, *'k*) *alt* ⇒ (*'a option*, *'k*) *sseg* **where**
[*simp*]: *ssegopt-of-alt* = *unfold-sseg* (*ssegopt-of-alt-emb o un-Alt*)

definition *bunopt-of-alt* :: (*'a*, *'k*) *alt* ⇒ (*'a option*, *'k*) *bun* **where**
[*simp*]: *bunopt-of-alt* = *unfold-bun* (*bunopt-of-alt-emb o un-Alt*)

definition *seg-of-sseg* :: (*'a*, *'k*) *sseg* ⇒ (*'a*, *'k*) *seg* **where**
[*simp*]: *seg-of-sseg* = *unfold-seg* (*seg-of-sseg-emb o un-SSeg*)

definition *seg-of-gen* :: *'a gen* ⇒ (*'a*, *'k*) *seg* **where**
[*simp*]: *seg-of-gen* = *unfold-seg* (*seg-of-gen-emb o un-Gen*)

definition *bun-of-lts* :: (*'a*, *'k*) *lts* ⇒ (*'a*, *'k*) *bun* **where**
[*simp*]: *bun-of-lts* = *unfold-bun* (*bun-of-lts-emb o un-LTS*)

definition *bun-of-gen* :: *'a gen* ⇒ (*'a*, *'k*) *bun* **where**
[*simp*]: *bun-of-gen* = *unfold-bun* (*bun-of-gen-emb o un-Gen*)

definition *pz-of-seg* :: (*'a*, *'k*) *seg* ⇒ (*'a*, *'k1*, *'k*) *pz* **where**
[*simp*]: *pz-of-seg* = *unfold-pz* (*pz-of-seg-emb o un-Seg*)

definition *pz-of-bun* :: (*'a*, *'k*) *bun* ⇒ (*'a*, *'k*, *'k1*) *pz* **where**
[*simp*]: *pz-of-bun* = *unfold-pz* (*pz-of-bun-emb o un-Bun*)

definition *mg-of-pz* :: (*'a*, *'k1*, *'k2*) *pz* ⇒ (*'a*, *'k1*, *'k2*) *mg* **where**
[*simp*]: *mg-of-pz* = *unfold-mg* (*mg-of-pz-emb o un-PZ*)

definition *str-of-mc* :: *mc* ⇒ *'a str* **where**
[*simp*]: *str-of-mc* = *unfold-str* (*str-of-mc-emb o un-MC*)

definition *alt-of-str* :: *'a str* ⇒ (*'a*, *'k*) *alt* **where**
[*simp*]: *alt-of-str* = *unfold-alt* (*alt-of-str-emb o un-Str*)

definition *ssegopt-of-sseg* :: (*'a*, *'k*) *sseg* ⇒ (*'a option*, *'k*) *sseg* **where**
[*simp*]: *ssegopt-of-sseg* = *unfold-sseg* (*ssegopt-of-sseg-emb o un-SSeg*)

definition *segopt-of-seg* :: (*'a*, *'k*) *seg* ⇒ (*'a option*, *'k*) *seg* **where**
[*simp*]: *segopt-of-seg* = *unfold-seg* (*segopt-of-seg-emb o un-Seg*)

definition *bunopt-of-bun* :: $('a, 'k)$ *bun* $\Rightarrow$ $('a\ option, 'k)$ *bun* **where**
[*simp*]: *bunopt-of-bun* = *unfold-bun* (*bunopt-of-bun-emb o un-Bun*)

definition *pzopt-of-pz* :: $('a, 'k1, 'k2)$ *pz* $\Rightarrow$ $('a\ option, 'k1, 'k2)$ *pz* **where**
[*simp*]: *pzopt-of-pz* = *unfold-pz* (*pzopt-of-pz-emb o un-PZ*)

definition *pzopt-of-mg* :: $('a, 'k1, 'k2)$ *mg* $\Rightarrow$ $('a\ option, 'k1, 'k2)$ *pz* **where**
[*simp*]: *pzopt-of-mg* = *unfold-pz* (*pzopt-of-mg-emb o un-MG*)

definition *mg-of-pzopt* :: $('a\ option, 'k1, 'k2)$ *pz* $\Rightarrow$ $('a, 'k1, 'k2)$ *mg* **where**
[*simp*]: *mg-of-pzopt* = *unfold-mg* (*mg-of-pzopt-emb o un-PZ*)

definition *mg-of-alt1* :: $('a, 'k)$ *alt* $\Rightarrow$ $('a, 'k1, 'k)$ *mg* **where**
[*simp*]: *mg-of-alt1* = *unfold-mg* (*mg-of-alt1-emb o un-Alt*)

definition *mg-of-alt2* :: $('a, 'k)$ *alt* $\Rightarrow$ $('a, 'k, 'k1)$ *mg* **where**
[*simp*]: *mg-of-alt2* = *unfold-mg* (*mg-of-alt2-emb o un-Alt*)

definition *pz-of-alt1* :: $('a, 'k)$ *alt* $\Rightarrow$ $('a\ option, 'k1, 'k)$ *pz* **where**
[*simp*]: *pz-of-alt1* = *unfold-pz* (*pz-of-alt1-emb o un-Alt*)

definition *pz-of-alt2* :: $('a, 'k)$ *alt* $\Rightarrow$ $('a\ option, 'k, 'k2)$ *pz* **where**
[*simp*]: *pz-of-alt2* = *unfold-pz* (*pz-of-alt2-emb o un-Alt*)

5 Automation Setup

lemma *mc-rel-eq*[*unfolded vimage2p-def*]:
BNF-Def.vimage2p un-MC un-MC (*rel-pmf* (=)) = (=)
⟨*proof*⟩

lemma *dlts-rel-eq*[*unfolded vimage2p-def*]:
BNF-Def.vimage2p un-DLTS un-DLTS (*rel-fun* (=) (*rel-option* (=))) = (=)
⟨*proof*⟩

lemma *react-rel-eq*[*unfolded vimage2p-def*]:
BNF-Def.vimage2p un-React un-React (*rel-fun* (=) (*rel-option* (*rel-pmf* (=)))) = (=)
⟨*proof*⟩

lemma *all-neq-Inl-ex-eq-Inr*[*dest*]: $(\forall l.\ x \neq Inl\ l) \Longrightarrow (\exists r.\ x = Inr\ r)$ ⟨*proof*⟩
lemma *all-neq-Inr-ex-eq-Inl*[*dest*]: $(\forall r.\ x \neq Inr\ r) \Longrightarrow (\exists l.\ x = Inl\ l)$ ⟨*proof*⟩
lemma *all2-neq-Inl-ex-eq-Inr*[*dest*]: $(\forall a\ b.\ x \neq Inl\ (a, b)) \Longrightarrow (\exists r.\ x = Inr\ r)$ ⟨*proof*⟩
lemma *all2-neq-Inr-ex-eq-Inl*[*dest*]: $(\forall a\ b.\ x \neq Inr\ (a, b)) \Longrightarrow (\exists l.\ x = Inl\ l)$ ⟨*proof*⟩

lemma *rel-prod-simp-asym*[*simp*]:
$\bigwedge x\ y.\ rel\text{-}prod\ R\ S\ (x, y) = (\lambda z.\ case\ z\ of\ (x', y') \Rightarrow R\ x\ x' \wedge S\ y\ y')$

$\bigwedge x\ y\ z.\ \mathit{rel\text{-}prod}\ R\ S\ x\ (y,\ z) = (\mathit{case}\ x\ \mathit{of}\ (y',\ z') \Rightarrow R\ y'\ y \wedge S\ z'\ z)$
⟨*proof*⟩

lemma *map-prod-eq-Pair-iff* [*simp*]:
$\mathit{map\text{-}prod}\ f\ g\ x = (y,\ z) \longleftrightarrow (f\ (\mathit{fst}\ x) = y \wedge g\ (\mathit{snd}\ x) = z)$
⟨*proof*⟩

lemmas [*abs-def*, *simp*] =
sum.rel-map prod.rel-map option.rel-map pmf.rel-map bset.rel-map fun.rel-map nebset.rel-map

lemmas [*simp*] =
lts.rel-eq lmc.rel-eq lmdp.rel-eq gen.rel-eq str.rel-eq alt.rel-eq sseg.rel-eq seg.rel-eq bun.rel-eq pz.rel-eq mg.rel-eq
rel-pmf-return-pmf1 rel-pmf-return-pmf2 set-pmf-not-empty rel-pmf-rel-prod
bset.set-map nebset.set-map

lemmas [*simp del*] =
split-paired-Ex

lemma *bisimilar-eqI*:
assumes $\bigwedge R.\ [\![R\ x\ y;\ \bigwedge x\ y.\ R\ x\ y \Longrightarrow Q\ R\ (s1\ x)\ (s2\ y)]\!] \Longrightarrow P\ x\ y$
and $P\ x\ y \Longrightarrow \forall x\ y.\ P\ x\ y \longrightarrow Q\ P\ (s1\ x)\ (s2\ y)$
shows $\mathit{bisimilar}\ Q\ s1\ s2\ x\ y = P\ x\ y$
⟨*proof*⟩

bundle *probabilistic-hierarchy* =
rel-fun-def[*simp*]
sum.splits[*split*]
prod.splits[*split*]
option.splits[*split*]

predicate2-eqD[*THEN iffD2*, *OF mc-rel-eq*, *dest*]
predicate2-eqD[*THEN iffD2*, *OF dlts-rel-eq*, *dest*]
predicate2-eqD[*THEN iffD2*, *OF lts.rel-eq*, *dest*]
predicate2-eqD[*THEN iffD2*, *OF react-rel-eq*, *dest*]
predicate2-eqD[*THEN iffD2*, *OF lmc.rel-eq*, *dest*]
predicate2-eqD[*THEN iffD2*, *OF lmdp.rel-eq*, *dest*]
predicate2-eqD[*THEN iffD2*, *OF gen.rel-eq*, *dest*]
predicate2-eqD[*THEN iffD2*, *OF str.rel-eq*, *dest*]
predicate2-eqD[*THEN iffD2*, *OF alt.rel-eq*, *dest*]
predicate2-eqD[*THEN iffD2*, *OF sseg.rel-eq*, *dest*]
predicate2-eqD[*THEN iffD2*, *OF seg.rel-eq*, *dest*]
predicate2-eqD[*THEN iffD2*, *OF bun.rel-eq*, *dest*]
predicate2-eqD[*THEN iffD2*, *OF pz.rel-eq*, *dest*]
predicate2-eqD[*THEN iffD2*, *OF mg.rel-eq*, *dest*]

iffD1 [*OF lts.rel-sel*, *dest*!]
iffD1 [*OF lmc.rel-sel*, *dest*!]

iffD1 [*OF lmdp.rel-sel*, *dest*!]
iffD1 [*OF gen.rel-sel*, *dest*!]
iffD1 [*OF str.rel-sel*, *dest*!]
iffD1 [*OF alt.rel-sel*, *dest*!]
iffD1 [*OF sseg.rel-sel*, *dest*!]
iffD1 [*OF seg.rel-sel*, *dest*!]
iffD1 [*OF bun.rel-sel*, *dest*!]
iffD1 [*OF pz.rel-sel*, *dest*!]
iffD1 [*OF mg.rel-sel*, *dest*!]

pmf.rel-refl[*intro*]
bset.rel-refl[*intro*]
nebset.rel-refl[*intro*]
prod.rel-refl[*intro*]
sum.rel-refl[*intro*]
option.rel-refl[*intro*]

pmf.rel-mono-strong[*intro*]
bset.rel-mono-strong[*intro*]
nebset.rel-mono-strong[*intro*]
prod.rel-mono-strong[*intro*]
sum.rel-mono-strong[*intro*]
option.rel-mono-strong[*intro*]

6 Proofs

context
includes *probabilistic-hierarchy*
begin

method *bisimilar-alt* =
rule bisimilar-eqI,
match **conclusion in** *u1 s1 x* = *u2 s2 y* **for** *u1 u2 s1 s2 x y* ⇒
‹*coinduction arbitrary*: *x y*, *fastforce*›,
fastforce

lemma *bisimilar-alt*:
$\bigwedge$*s1 s2*. *bisimilar-mc s1 s2 x y* = (*unfold-mc s1 x* = *unfold-mc s2 y*)
$\bigwedge$*s1 s2*. *bisimilar-dlts s1 s2 x y* = (*unfold-dlts s1 x* = *unfold-dlts s2 y*)
$\bigwedge$*s1 s2*. *bisimilar-lts s1 s2 x y* = (*unfold-lts s1 x* = *unfold-lts s2 y*)
$\bigwedge$*s1 s2*. *bisimilar-react s1 s2 x y* = (*unfold-react s1 x* = *unfold-react s2 y*)
$\bigwedge$*s1 s2*. *bisimilar-lmc s1 s2 x y* = (*unfold-lmc s1 x* = *unfold-lmc s2 y*)
$\bigwedge$*s1 s2*. *bisimilar-lmdp s1 s2 x y* = (*unfold-lmdp s1 x* = *unfold-lmdp s2 y*)
$\bigwedge$*s1 s2*. *bisimilar-gen s1 s2 x y* = (*unfold-gen s1 x* = *unfold-gen s2 y*)
$\bigwedge$*s1 s2*. *bisimilar-str s1 s2 x y* = (*unfold-str s1 x* = *unfold-str s2 y*)
$\bigwedge$*s1 s2*. *bisimilar-alt s1 s2 x y* = (*unfold-alt s1 x* = *unfold-alt s2 y*)
$\bigwedge$*s1 s2*. *bisimilar-sseg s1 s2 x y* = (*unfold-sseg s1 x* = *unfold-sseg s2 y*)
$\bigwedge$*s1 s2*. *bisimilar-seg s1 s2 x y* = (*unfold-seg s1 x* = *unfold-seg s2 y*)
$\bigwedge$*s1 s2*. *bisimilar-bun s1 s2 x y* = (*unfold-bun s1 x* = *unfold-bun s2 y*)

$\bigwedge s1\ s2.\ \textit{bisimilar-pz}\ s1\ s2\ x\ y = (\textit{unfold-pz}\ s1\ x = \textit{unfold-pz}\ s2\ y)$
$\bigwedge s1\ s2.\ \textit{bisimilar-mg}\ s1\ s2\ x\ y = (\textit{unfold-mg}\ s1\ x = \textit{unfold-mg}\ s2\ y)$
⟨*proof*⟩

method *commute-prover* =
intro ext,
match **conclusion in** *u1 s1 x* = (*emb o u2 s2*) *x* **for** *emb u1 u2 s1 s2 x* ⇒
‹*coinduction arbitrary*: *x*, *fastforce*›

lemma *emb-commute*:
⋀*s*. *unfold-lts* (*lts-of-dlts-emb o s*) = *lts-of-dlts o unfold-dlts s*
⋀*s*. *unfold-gen* (*gen-of-lmc-emb o s*) = *gen-of-lmc o unfold-lmc s*
⋀*s*. *unfold-lmdp* (*lmdp-of-lmc-emb o s*) = *lmdp-of-lmc o unfold-lmc s*
⋀*s*. *unfold-react* (*react-of-dlts-emb o s*) = *react-of-dlts o unfold-dlts s*
⋀*s*. *unfold-sseg* (*sseg-of-lmdp-emb o s*) = *sseg-of-lmdp o unfold-lmdp s*
⋀*s*. *unfold-sseg* (*sseg-of-lts-emb o s*) = *sseg-of-lts o unfold-lts s*
⋀*s*. *unfold-sseg* (*ssegopt-of-alt-emb o s*) = *ssegopt-of-alt o unfold-alt s*
⋀*s*. *unfold-sseg* (*sseg-of-react-emb o s*) = *sseg-of-react o unfold-react s*
⋀*s*. *unfold-seg* (*seg-of-sseg-emb o s*) = *seg-of-sseg o unfold-sseg s*
⋀*s*. *unfold-seg* (*seg-of-gen-emb o s*) = *seg-of-gen o unfold-gen s*
⋀*s*. *unfold-bun* (*bun-of-lts-emb o s*) = *bun-of-lts o unfold-lts s*
⋀*s*. *unfold-bun* (*bunopt-of-alt-emb o s*) = *bunopt-of-alt o unfold-alt s*
⋀*s*. *unfold-bun* (*bun-of-gen-emb o s*) = *bun-of-gen o unfold-gen s*
⋀*s*. *unfold-pz* (*pz-of-seg-emb o s*) = *pz-of-seg o unfold-seg s*
⋀*s*. *unfold-pz* (*pz-of-bun-emb o s*) = *pz-of-bun o unfold-bun s*
⋀*s*. *unfold-str* (*str-of-mc-emb o s*) = *str-of-mc o unfold-mc s*
⋀*s*. *unfold-alt* (*alt-of-str-emb o s*) = *alt-of-str o unfold-str s*
⋀*s*. *unfold-sseg* (*ssegopt-of-sseg-emb o s*) = *ssegopt-of-sseg o unfold-sseg s*
⋀*s*. *unfold-seg* (*segopt-of-seg-emb o s*) = *segopt-of-seg o unfold-seg s*
⋀*s*. *unfold-bun* (*bunopt-of-bun-emb o s*) = *bunopt-of-bun o unfold-bun s*
⋀*s*. *unfold-pz* (*pzopt-of-pz-emb o s*) = *pzopt-of-pz o unfold-pz s*
⋀*s*. *unfold-pz* (*pzopt-of-mg-emb o s*) = *pzopt-of-mg o unfold-mg s*
⋀*s*. *unfold-mg* (*mg-of-pzopt-emb o s*) = *mg-of-pzopt o unfold-pz s*

⋀*s*. *unfold-mg* (*mg-of-pz-emb o s*) = *mg-of-pz o unfold-pz s*
⋀*s*. *unfold-mg* (*mg-of-alt1-emb o s*) = *mg-of-alt1 o unfold-alt s*
⋀*s*. *unfold-mg* (*mg-of-alt2-emb o s*) = *mg-of-alt2 o unfold-alt s*
⋀*s*. *unfold-pz* (*pz-of-alt1-emb o s*) = *pz-of-alt1 o unfold-alt s*
⋀*s*. *unfold-pz* (*pz-of-alt2-emb o s*) = *pz-of-alt2 o unfold-alt s*
⟨*proof*⟩

method *inj-prover* =
intro injI,
match **conclusion in** *x* = *y* **for** *x y* ⇒ ‹*coinduction arbitrary*: *x y*, *fastforce*›

lemma *inj*:
inj lts-of-dlts
inj react-of-dlts
inj gen-of-lmc

inj lmdp-of-lmc
inj sseg-of-lmdp
inj sseg-of-react
inj sseg-of-lts
inj ssegopt-of-alt
inj seg-of-gen
inj seg-of-sseg
inj bun-of-lts
inj bunopt-of-alt
inj bun-of-gen
inj pz-of-seg
inj pz-of-bun
inj str-of-mc
inj alt-of-str
inj ssegopt-of-sseg
inj segopt-of-seg
inj bunopt-of-bun
inj pzopt-of-pz
inj pzopt-of-mg
inj mg-of-pzopt

inj mg-of-pz
inj mg-of-alt1
inj mg-of-alt2
inj pz-of-alt1
inj pz-of-alt2
⟨*proof*⟩

end

lemma *hierarchy*:
⋀*s1 s2*. *bisimilar-dlts s1 s2 x y* ⟷ *bisimilar-lts* (*lts-of-dlts-emb o s1*) (*lts-of-dlts-emb o s2*) *x y*
⋀*s1 s2*. *bisimilar-lmc s1 s2 x y* ⟷ *bisimilar-gen* (*gen-of-lmc-emb o s1*) (*gen-of-lmc-emb o s2*) *x y*
⋀*s1 s2*. *bisimilar-lmc s1 s2 x y* ⟷ *bisimilar-lmdp* (*lmdp-of-lmc-emb o s1*) (*lmdp-of-lmc-emb o s2*) *x y*
⋀*s1 s2*. *bisimilar-dlts s1 s2 x y* ⟷ *bisimilar-react* (*react-of-dlts-emb o s1*) (*react-of-dlts-emb o s2*) *x y*
⋀*s1 s2*. *bisimilar-lmdp s1 s2 x y* ⟷ *bisimilar-sseg* (*sseg-of-lmdp-emb o s1*) (*sseg-of-lmdp-emb o s2*) *x y*
⋀*s1 s2*. *bisimilar-lts s1 s2 x y* ⟷ *bisimilar-sseg* (*sseg-of-lts-emb o s1*) (*sseg-of-lts-emb o s2*) *x y*
⋀*s1 s2*. *bisimilar-alt s1 s2 x y* ⟷ *bisimilar-sseg* (*ssegopt-of-alt-emb o s1*) (*ssegopt-of-alt-emb o s2*) *x y*
⋀*s1 s2*. *bisimilar-react s1 s2 x y* ⟷ *bisimilar-sseg* (*sseg-of-react-emb o s1*) (*sseg-of-react-emb o s2*) *x y*
⋀*s1 s2*. *bisimilar-sseg s1 s2 x y* ⟷ *bisimilar-seg* (*seg-of-sseg-emb o s1*) (*seg-of-sseg-emb o s2*) *x y*

$\bigwedge$(*s1* :: - ⇒ (*'a option* × - *pmf*) *set*[-]) *s2*.
bisimilar-sseg s1 s2 x y ⟷ *bisimilar-seg* (*seg-of-sseg-emb o s1*) (*seg-of-sseg-emb o s2*) *x y*
$\bigwedge$*s1 s2*. *bisimilar-gen s1 s2 x y* ⟷ *bisimilar-seg* (*seg-of-gen-emb o s1*) (*seg-of-gen-emb o s2*) *x y*
$\bigwedge$*s1 s2*. *bisimilar-lts s1 s2 x y* ⟷ *bisimilar-bun* (*bun-of-lts-emb o s1*) (*bun-of-lts-emb o s2*) *x y*
$\bigwedge$*s1 s2*. *bisimilar-alt s1 s2 x y* ⟷ *bisimilar-bun* (*bunopt-of-alt-emb o s1*) (*bunopt-of-alt-emb o s2*) *x y*
$\bigwedge$*s1 s2*. *bisimilar-gen s1 s2 x y* ⟷ *bisimilar-bun* (*bun-of-gen-emb o s1*) (*bun-of-gen-emb o s2*) *x y*
$\bigwedge$*s1 s2*. *bisimilar-seg s1 s2 x y* ⟷ *bisimilar-pz* (*pz-of-seg-emb o s1*) (*pz-of-seg-emb o s2*) *x y*
$\bigwedge$*s1 s2*. *bisimilar-bun s1 s2 x y* ⟷ *bisimilar-pz* (*pz-of-bun-emb o s1*) (*pz-of-bun-emb o s2*) *x y*
$\bigwedge$(*s1* :: - ⇒ (*'a option* × -) *pmf set*[-]) *s2*.
bisimilar-seg s1 s2 x y ⟷ *bisimilar-pz* (*pz-of-seg-emb o s1*) (*pz-of-seg-emb o s2*) *x y*
$\bigwedge$(*s1* :: - ⇒ ((*'a option* × -) *set*[-]) *pmf*) *s2*.
bisimilar-bun s1 s2 x y ⟷ *bisimilar-pz* (*pz-of-bun-emb o s1*) (*pz-of-bun-emb o s2*) *x y*
$\bigwedge$*s1 s2*. *bisimilar-mc s1 s2 x y* ⟷ *bisimilar-str* (*str-of-mc-emb o s1*) (*str-of-mc-emb o s2*) *x y*
$\bigwedge$*s1 s2*. *bisimilar-sseg s1 s2 x y* ⟷ *bisimilar-sseg* (*ssegopt-of-sseg-emb o s1*) (*ssegopt-of-sseg-emb o s2*) *x y*
$\bigwedge$*s1 s2*. *bisimilar-seg s1 s2 x y* ⟷ *bisimilar-seg* (*segopt-of-seg-emb o s1*) (*segopt-of-seg-emb o s2*) *x y*
$\bigwedge$*s1 s2*. *bisimilar-bun s1 s2 x y* ⟷ *bisimilar-bun* (*bunopt-of-bun-emb o s1*) (*bunopt-of-bun-emb o s2*) *x y*
$\bigwedge$*s1 s2*. *bisimilar-pz s1 s2 x y* ⟷ *bisimilar-pz* (*pzopt-of-pz-emb o s1*) (*pzopt-of-pz-emb o s2*) *x y*
$\bigwedge$*s1 s2*. *bisimilar-str s1 s2 x y* ⟷ *bisimilar-alt* (*alt-of-str-emb o s1*) (*alt-of-str-emb o s2*) *x y*
$\bigwedge$*s1 s2*. *bisimilar-mg s1 s2 x y* ⟷ *bisimilar-pz* (*pzopt-of-mg-emb o s1*) (*pzopt-of-mg-emb o s2*) *x y*
⟨*proof*⟩

An edge that would make the graph cyclic

lemma
$\bigwedge$*s1 s2*. *bisimilar-pz s1 s2 x y* ⟷ *bisimilar-mg* (*mg-of-pz-emb o s1*) (*mg-of-pz-emb o s2*) *x y*
⟨*proof*⟩

Some redundant (historic) transitive edges

lemma
$\bigwedge$*s1 s2*. *bisimilar-pz s1 s2 x y* ⟷ *bisimilar-mg* (*mg-of-pzopt-emb o s1*) (*mg-of-pzopt-emb o s2*) *x y*
$\bigwedge$*s1 s2*. *bisimilar-alt s1 s2 x y* ⟷ *bisimilar-mg* (*mg-of-alt1-emb o s1*) (*mg-of-alt1-emb o s2*) *x y*

$\bigwedge s1\ s2.\ bisimilar\text{-}alt\ s1\ s2\ x\ y \longleftrightarrow bisimilar\text{-}mg\ (mg\text{-}of\text{-}alt2\text{-}emb\ o\ s1)\ (mg\text{-}of\text{-}alt2\text{-}emb\ o\ s2)\ x\ y$
$\bigwedge s1\ s2.\ bisimilar\text{-}alt\ s1\ s2\ x\ y \longleftrightarrow bisimilar\text{-}pz\ (pz\text{-}of\text{-}alt1\text{-}emb\ o\ s1)\ (pz\text{-}of\text{-}alt1\text{-}emb\ o\ s2)\ x\ y$
$\bigwedge s1\ s2.\ bisimilar\text{-}alt\ s1\ s2\ x\ y \longleftrightarrow bisimilar\text{-}pz\ (pz\text{-}of\text{-}alt2\text{-}emb\ o\ s1)\ (pz\text{-}of\text{-}alt2\text{-}emb\ o\ s2)\ x\ y$
⟨*proof*⟩

7 Some special proofs

Two views on LTS

lemma $\exists f::(('a \times 's)\ set \Rightarrow 'a \Rightarrow 's\ set).\ bij\ f$
⟨*proof*⟩

lemma $\exists f::(('a \times 's)\ set[('a \times 's)\ set] \Rightarrow 'a \Rightarrow 's\ set['s\ set]).\ bij\ f$
⟨*proof*⟩

mc is trivial

lemma *mc-unit*:
fixes $x\ y :: mc$
shows $x = y$
⟨*proof*⟩

lemma *bisimilar-mc s1 s2 x y*
⟨*proof*⟩

8 Printing the Hierarchy Graph

⟨*ML*⟩

9 Vardi Systems

context notes [[*bnf-internals*]]
begin
datatype $('a, 'b, 'k)\ var0 = PMF\ ('a \times 'b)\ pmf \mid BPS\ ('a \times 'b)\ set['k]$
end

inductive $var\text{-}eq :: ('a, 'b, 'k)\ var0 \Rightarrow ('a, 'b, 'k)\ var0 \Rightarrow bool$ (**infixl** ‹∼› *65*)
where
var-eq-reflp[*intro*]: $x \sim x$
| [*intro*]: $PMF\ (return\text{-}pmf\ (a, x)) \sim BPS\ (bsingleton\ (a, x))$
| [*intro*]: $BPS\ (bsingleton\ (a, x)) \sim PMF\ (return\text{-}pmf\ (a, x))$

lemma *var-eq-symp*: $x \sim y \Longrightarrow y \sim x$
⟨*proof*⟩

lemma *var-eq-transp*: $x \sim y \Longrightarrow y \sim z \Longrightarrow x \sim z$
⟨*proof*⟩

quotient-type $('a, 'b, 'k)$ *var* = $('a, 'b, 'k)$ *var0* / $(\sim)$
⟨*proof*⟩

lift-definition *map-var* :: $('a \Rightarrow 'c) \Rightarrow ('b \Rightarrow 'd) \Rightarrow ('a, 'b, 'k)$ *var* $\Rightarrow ('c, 'd, 'k)$ *var*
is *map-var0*
⟨*proof*⟩

lift-definition *set1-var* :: $('a, 'b, 'k)$ *var* $\Rightarrow$ *'a set*
is *set1-var0*
⟨*proof*⟩

lift-definition *set2-var* :: $('a, 'b, 'k)$ *var* $\Rightarrow$ *'b set*
is *set2-var0*
⟨*proof*⟩

inductive *rel-var* :: $('a \Rightarrow 'c \Rightarrow bool) \Rightarrow ('b \Rightarrow 'd \Rightarrow bool) \Rightarrow ('a, 'b, 'k)$ *var* $\Rightarrow ('c, 'd, 'k)$ *var* $\Rightarrow$ *bool* **for** R S **where**
set1-var $x \subseteq \{(x, y).\ R\ x\ y\} \Longrightarrow$ *set2-var* $x \subseteq \{(x, y).\ S\ x\ y\} \Longrightarrow$
rel-var R S (*map-var fst fst* x) (*map-var snd snd* x)

abbreviation (*input*) *var0-of-gen-emb* $\equiv$ *case-option* (*BPS bempty*) *PMF*
abbreviation (*input*) *var0-of-lts-emb* $\equiv$ *BPS*

lift-definition *var-of-gen-emb* :: $('a \times 'b)$ *pmf option* $\Rightarrow ('a, 'b, 'k)$ *var* **is** *var0-of-gen-emb*
⟨*proof*⟩
lift-definition *var-of-lts-emb* :: $('a \times 'b)$ *set*$['k] \Rightarrow ('a, 'b, 'k)$ *var* **is** *var0-of-lts-emb*
⟨*proof*⟩

abbreviation *bisimilar-var* $\equiv$ *bisimilar* (λR. *rel-var* $(=)$ R)

lemma *map-var0-eq-BPS-iff* [*simp*]:
map-var0 f g z = *BPS* $X \longleftrightarrow (\exists\, Y.\ z =$ *BPS* $Y \wedge$ *map-bset* (*map-prod* f g) $Y = X)$
⟨*proof*⟩

lemma *map-var0-eq-PMF-iff* [*simp*]:
map-var0 f g z = *PMF* $p \longleftrightarrow (\exists\, q.\ z =$ *PMF* $q \wedge$ *map-pmf* (*map-prod* f g) $q = p)$
⟨*proof*⟩

lemma *bisimilar-lts s1 s2 x y* $\longleftrightarrow$ *bisimilar-var* (*var-of-lts-emb o s1*) (*var-of-lts-emb o s2*) *x y*
(**is** - $\longleftrightarrow$ *bisimilar-var* (*?emb1 o* -) (*?emb2 o* -) - -)
⟨*proof*⟩

lemma *bisimilar-gen s1 s2 x y* $\longleftrightarrow$ *bisimilar-var* (*var-of-gen-emb o s1*) (*var-of-gen-emb o s2*) *x y*
(**is** - $\longleftrightarrow$ *bisimilar-var* (*?emb1 o* -) (*?emb2 o* -) - -)
$\langle$*proof*$\rangle$