Conformance Relations between Input/Output Languages

Robert Sachtleben

October 11, 2023

Abstract

This entry formalises the paper of the same name by Huang et al. [1] and presents a unifying characterisation of well-known conformance relations such as equivalence and language inclusion (reduction) on languages over input/output pairs. This characterisation simplifies comparisons between conformance relations and from it a fundamental necessary and sufficient criterion for conformance testing is developed.

Contents

theory *Input-Output-Language-Conformance*
imports *HOL−Library.Sublist*
begin

1 Preliminaries

type-synonym $('a)$ *alphabet* $=$ $'a$ *set*
type-synonym $('x,'y)$ *word* $=$ $('x \times 'y)$ *list*
type-synonym $('x,'y)$ *language* $=$ $('x,'y)$ *word set*
type-synonym $('y)$ *output-relation* $=$ $('y$ *set* $\times$ $'y$ *set*$)$ *set*

fun *is-language* :: $'x$ *alphabet* $\Rightarrow$ $'y$ *alphabet* $\Rightarrow$ $('x,'y)$ *language* $\Rightarrow$ *bool* **where**
is-language $X\ Y\ L = ($
— nonempty
$(L \neq \{\}) \wedge$
$(\forall\ \pi \in L\ .$
— over X and Y
$(\forall\ xy \in set\ \pi\ .\ fst\ xy \in X \wedge snd\ xy \in Y) \wedge$
— prefix closed
$(\forall\ \pi'\ .\ prefix\ \pi'\ \pi \longrightarrow \pi' \in L)))$

lemma *language-contains-nil* :
assumes *is-language* $X\ Y\ L$
shows $[] \in L$
⟨*proof*⟩

lemma *language-intersection-is-language* :
assumes *is-language* $X\ Y\ L1$
and *is-language* $X\ Y\ L2$
shows *is-language* $X\ Y\ (L1 \cap L2)$
⟨*proof*⟩

fun *language-for-state* :: $('x,'y)$ *language* $\Rightarrow$ $('x,'y)$ *word* $\Rightarrow$ $('x,'y)$ *language* **where**
language-for-state $L\ \pi = \{\tau\ .\ \pi@\tau \in L\}$

notation *language-for-state* $(\mathcal{L}[\text{-},\text{-}])$

lemma *language-for-state-is-language* :
assumes *is-language* $X\ Y\ L$
and $\pi \in L$
shows *is-language* $X\ Y\ \mathcal{L}[L,\pi]$
⟨*proof*⟩

lemma *language-of-state-empty-iff* :
assumes *is-language* $X\ Y\ L$
shows $(\mathcal{L}[L,\pi] = \{\}) \longleftrightarrow (\pi \notin L)$
⟨*proof*⟩

fun *are-equivalent-for-language* :: $('x,'y)$ *language* $\Rightarrow$ $('x,'y)$ *word* $\Rightarrow$ $('x,'y)$ *word* $\Rightarrow$ *bool* **where**
are-equivalent-for-language $L\ \alpha\ \beta = (\mathcal{L}[L,\alpha] = \mathcal{L}[L,\beta])$

abbreviation(*input*) *input-projection* $\pi \equiv$ *map fst* π
abbreviation(*input*) *output-projection* $\pi \equiv$ *map snd* π
notation *input-projection* $([-]_I)$
notation *output-projection* $([-]_O)$

fun *is-executable* :: $('x,'y)$ *language* $\Rightarrow$ $('x,'y)$ *word* $\Rightarrow$ $'x$ *list* $\Rightarrow$ *bool* **where**
is-executable $L\ \pi\ xs = (\exists\ \tau \in \mathcal{L}[L,\pi]\ .\ [\tau]_I = xs)$

fun *executable-sequences* :: $('x,'y)$ *language* $\Rightarrow$ $('x,'y)$ *word* $\Rightarrow$ $'x$ *list set* **where**
executable-sequences $L\ \pi = \{xs\ .\ $ *is-executable* $L\ \pi\ xs\}$

fun *executable-inputs* :: $('x,'y)$ *language* $\Rightarrow$ $('x,'y)$ *word* $\Rightarrow$ $'x$ *set* **where**
executable-inputs $L\ \pi = \{x\ .\ $ *is-executable* $L\ \pi\ [x]\}$

notation *executable-inputs* (*exec*[-,-])

lemma *executable-sequences-alt-def* : *executable-sequences* $L\ \pi = \{xs\ .\ \exists\ ys\ .$ *length* $ys =$ *length* $xs\ \wedge$ *zip* $xs\ ys \in \mathcal{L}[L,\pi]\}$
⟨*proof*⟩

lemma *executable-inputs-alt-def* : *executable-inputs* $L\ \pi = \{x\ .\ \exists\ y\ .\ [(x,y)] \in \mathcal{L}[L,\pi]\}$
⟨*proof*⟩

lemma *executable-inputs-in-alphabet* :
assumes *is-language* $X\ Y\ L$
and $x \in exec[L,\pi]$
shows $x \in X$
⟨*proof*⟩

fun *output-sequences* :: $('x,'y)$ *language* $\Rightarrow$ $('x,'y)$ *word* $\Rightarrow$ $'x$ *list* $\Rightarrow$ $'y$ *list set* **where**
output-sequences $L\ \pi\ xs =$ *output-projection* ‘ $\{\tau \in \mathcal{L}[L,\pi]\ .\ [\tau]_I = xs\}$

lemma *prefix-closure-no-member* :
assumes *is-language* $X\ Y\ L$
and $\pi \notin L$
shows $\pi @ \tau \notin L$
⟨*proof*⟩

lemma *output-sequences-empty-iff* :
  assumes *is-language X Y L*
shows $(output\text{-}sequences\ L\ \pi\ xs = \{\}) = ((\pi \notin L) \vee (\neg\ is\text{-}executable\ L\ \pi\ xs))$
  ⟨*proof*⟩

fun *outputs* :: $('x,'y)$ *language* ⇒ $('x,'y)$ *word* ⇒ $'x$ ⇒ $'y$ *set* **where**
  $outputs\ L\ \pi\ x = \{y\ .\ [(x,y)] \in \mathcal{L}[L,\pi]\}$

notation *outputs* (*out*[-,-,-])

lemma *outputs-in-alphabet* :
  assumes *is-language X Y L*
shows $out[L,\pi,x] \subseteq Y$
  ⟨*proof*⟩

lemma *outputs-executable* : $(out[L,\pi,x] = \{\}) \longleftrightarrow (x \notin exec[L,\pi])$
  ⟨*proof*⟩

fun *is-completely-specified-for* :: $'x$ *set* ⇒ $('x,'y)$ *language* ⇒ *bool* **where**
  $is\text{-}completely\text{-}specified\text{-}for\ X\ L = (\forall\ \pi \in L\ .\ \forall\ x \in X\ .\ out[L,\pi,x] \neq \{\})$

lemma *prefix-executable* :
  assumes *is-language X Y L*
  and $\pi \in L$
  and $i <$ *length* π
shows $fst\ (\pi\ !\ i) \in exec[L,take\ i\ \pi]$
⟨*proof*⟩

2 Conformance Relations

definition *language-equivalence* :: $('x,'y)$ *language* ⇒ $('x,'y)$ *language* ⇒ *bool*
where
  $language\text{-}equivalence\ L1\ L2 = (L1 = L2)$

definition *language-inclusion* :: $('x,'y)$ *language* ⇒ $('x,'y)$ *language* ⇒ *bool* **where**

  $language\text{-}inclusion\ L1\ L2 = (L1 \subseteq L2)$

abbreviation(*input*) *reduction L1 L2* ≡ *language-inclusion L1 L2*

definition *quasi-equivalence* :: $('x,'y)$ *language* ⇒ $('x,'y)$ *language* ⇒ *bool* **where**
  $quasi\text{-}equivalence\ L1\ L2 = (\forall\ \pi \in L1 \cap L2\ .\ \forall\ x \in exec[L2,\pi]\ .\ out[L1,\pi,x] =$

$out[L2,\pi,x])$

definition $quasi\text{-}reduction$ :: $('x,'y)$ $language \Rightarrow ('x,'y)$ $language \Rightarrow bool$ **where**
$quasi\text{-}reduction\ L1\ L2 = (\forall\ \pi \in L1 \cap L2\ .\ \forall\ x \in exec[L2,\pi]\ .\ (out[L1,\pi,x] \neq \{\} \wedge out[L1,\pi,x] \subseteq out[L2,\pi,x]))$

definition $strong\text{-}reduction$ :: $('x,'y)$ $language \Rightarrow ('x,'y)$ $language \Rightarrow bool$ **where**
$strong\text{-}reduction\ L1\ L2 = (quasi\text{-}reduction\ L1\ L2 \wedge (\forall\ \pi \in L1 \cap L2\ .\ \forall\ x\ .\ out[L2,\pi,x] = \{\} \longrightarrow out[L1,\pi,x] = \{\}))$

definition $semi\text{-}equivalence$ :: $('x,'y)$ $language \Rightarrow ('x,'y)$ $language \Rightarrow bool$ **where**
$semi\text{-}equivalence\ L1\ L2 = (\forall\ \pi \in L1 \cap L2\ .\ \forall\ x \in exec[L2,\pi]\ .$
$(out[L1,\pi,x] = \{\} \vee out[L1,\pi,x] = out[L2,\pi,x]) \wedge$
$(\exists\ x'\ .\ out[L1,\pi,x'] \cap out[L2,\pi,x'] \neq \{\}))$

definition $semi\text{-}reduction$ :: $('x,'y)$ $language \Rightarrow ('x,'y)$ $language \Rightarrow bool$ **where**
$semi\text{-}reduction\ L1\ L2 = (\forall\ \pi \in L1 \cap L2\ .\ \forall\ x \in exec[L2,\pi]\ .$
$(out[L1,\pi,x] \subseteq out[L2,\pi,x]) \wedge$
$(\exists\ x'\ .\ out[L1,\pi,x'] \cap out[L2,\pi,x'] \neq \{\}))$

definition $strong\text{-}semi\text{-}equivalence$ :: $('x,'y)$ $language \Rightarrow ('x,'y)$ $language \Rightarrow bool$ **where**
$strong\text{-}semi\text{-}equivalence\ L1\ L2 = (\forall\ \pi \in L1 \cap L2\ .\ \forall\ x\ .$
$(x \in exec[L2,\pi] \longrightarrow ((out[L1,\pi,x] = \{\} \vee out[L1,\pi,x] = out[L2,\pi,x]) \wedge (\exists\ x'\ .\ out[L1,\pi,x'] \cap out[L2,\pi,x'] \neq \{\}))) \wedge$
$(x \notin exec[L2,\pi] \longrightarrow out[L1,\pi,x] = \{\}))$

definition $strong\text{-}semi\text{-}reduction$ :: $('x,'y)$ $language \Rightarrow ('x,'y)$ $language \Rightarrow bool$ **where**
$strong\text{-}semi\text{-}reduction\ L1\ L2 = (\forall\ \pi \in L1 \cap L2\ .\ \forall\ x\ .$
$(x \in exec[L2,\pi] \longrightarrow (out[L1,\pi,x] \subseteq out[L2,\pi,x] \wedge (\exists\ x'\ .\ out[L1,\pi,x'] \cap out[L2,\pi,x'] \neq \{\}))) \wedge$
$(x \notin exec[L2,\pi] \longrightarrow out[L1,\pi,x] = \{\}))$

3 Unifying Characterisations

3.1 $\preceq$ Conformance

fun $type\text{-}1\text{-}conforms$ :: $('x,'y)$ $language \Rightarrow 'x$ $alphabet \Rightarrow 'y$ $output\text{-}relation \Rightarrow ('x,'y)$ $language \Rightarrow bool$ **where**
$type\text{-}1\text{-}conforms\ L1\ X\ H\ L2 = (\forall\ \pi \in L1 \cap L2\ .\ \forall\ x \in X\ .\ (out[L1,\pi,x],out[L2,\pi,x]) \in H)$

notation $type\text{-}1\text{-}conforms$ (- $\preceq$[-,-] -)

fun $equiv$ :: $'y$ $alphabet \Rightarrow 'y$ $output\text{-}relation$ **where**
$equiv\ Y = \{(A,A) \mid A\ .\ A \subseteq Y\}$

fun red :: $'y$ $alphabet \Rightarrow 'y$ $output\text{-}relation$ **where**

red $Y = \{(A,B) \mid A\ B\ .\ A \subseteq B \land B \subseteq Y\}$

fun *quasieq* :: *'y alphabet* $\Rightarrow$ *'y output-relation* **where**
quasieq $Y = \{(A,A) \mid A\ .\ A \subseteq Y\} \cup \{(A,\{\}) \mid A\ .\ A \subseteq Y\}$

fun *quasired* :: *'y alphabet* $\Rightarrow$ *'y output-relation* **where**
quasired $Y = \{(A,B) \mid A\ B\ .\ A \neq \{\} \land A \subseteq B \land B \subseteq Y\} \cup \{(C,\{\}) \mid C\ .\ C \subseteq Y\}$

fun *strongred* :: *'y alphabet* $\Rightarrow$ *'y output-relation* **where**
strongred $Y = \{(A,B) \mid A\ B\ .\ A \neq \{\} \land A \subseteq B \land B \subseteq Y\} \cup \{(\{\},\{\})\}$

lemma *red-type-1* :
assumes *is-language X Y L1*
and *is-language X Y L2*
shows *reduction L1 L2* $\longleftrightarrow$ $(L1 \preceq[X,red\ Y]\ L2)$
$\langle proof \rangle$

lemma *equiv-by-reduction* : $(L1 \preceq[X,equiv\ Y]\ L2) \longleftrightarrow ((L1 \preceq[X,red\ Y]\ L2) \land (L2 \preceq[X,red\ Y]\ L1))$
$\langle proof \rangle$

lemma *equiv-type-1* :
assumes *is-language X Y L1*
and *is-language X Y L2*
shows $(L1 = L2) \longleftrightarrow (L1 \preceq[X,equiv\ Y]\ L2)$
$\langle proof \rangle$

lemma *quasired-type-1* :
assumes *is-language X Y L1*
and *is-language X Y L2*
shows *quasi-reduction L1 L2* $\longleftrightarrow$ $(L1 \preceq[X,quasired\ Y]\ L2)$
$\langle proof \rangle$

lemma *quasieq-type-1* :
assumes *is-language X Y L1*
and *is-language X Y L2*
shows *quasi-equivalence L1 L2* $\longleftrightarrow$ $(L1 \preceq[X,quasieq\ Y]\ L2)$
$\langle proof \rangle$

lemma *strongred-type-1* :
assumes *is-language X Y L1*
and *is-language X Y L2*

shows *strong-reduction L1 L2* $\longleftrightarrow$ (*L1* $\preceq$[*X*,*strongred Y*] *L2*)
⟨*proof*⟩

3.2 ≤ Conformance

fun *type-2-conforms* :: ($'x$,$'y$) *language* ⇒ $'x$ *alphabet* ⇒ $'y$ *output-relation* ⇒ ($'x$,$'y$) *language* ⇒ *bool* **where**
type-2-conforms L1 X H L2 = (
(∀ π ∈ *L1* ∩ *L2* . ∀ *x* ∈ *X* . (*out*[*L1*,π,*x*],*out*[*L2*,π,*x*]) ∈ *H*) ∧
(∀ π ∈ *L1* ∩ *L2* . *exec*[*L2*,π] ≠ {} ⟶ (∃ *x* . *out*[*L1*,π,*x*] ∩ *out*[*L2*,π,*x*] ≠ {})))

notation *type-2-conforms* (- ≤[-,-] -)

fun *semieq* :: $'y$ *alphabet* ⇒ $'y$ *output-relation* **where**
semieq Y = {(*A*,*A*) | *A* . *A* ⊆ *Y*} ∪ {({},*A*) | *A* . *A* ⊆ *Y*} ∪ {(*A*,{}) | *A* . *A* ⊆ *Y*}

fun *semired* :: $'y$ *alphabet* ⇒ $'y$ *output-relation* **where**
semired Y = {(*A*,*B*) | *A B* . *A* ⊆ *B* ∧ *B* ⊆ *Y*} ∪ {(*C*,{}) | *C* . *C* ⊆ *Y*}

fun *strongsemieq* :: $'y$ *alphabet* ⇒ $'y$ *output-relation* **where**
strongsemieq Y = {(*A*,*A*) | *A* . *A* ⊆ *Y*} ∪ {({},*A*) | *A* . *A* ⊆ *Y*}

fun *strongsemired* :: $'y$ *alphabet* ⇒ $'y$ *output-relation* **where**
strongsemired Y = {(*A*,*B*) | *A B* . *A* ⊆ *B* ∧ *B* ⊆ *Y*}

lemma *strongsemieq-alt-def* : *strongsemieq Y* = *semieq Y* ∩ *red Y*
⟨*proof*⟩

lemma *strongsemired-alt-def* : *strongsemired Y* = *red Y*
⟨*proof*⟩

lemma *semired-type-2* :
assumes *is-language X Y L1*
and *is-language X Y L2*
shows (*semi-reduction L1 L2*) $\longleftrightarrow$ (*L1* ≤[*X*, *semired Y*] *L2*)
⟨*proof*⟩

lemma *semieq-type-2* :
assumes *is-language X Y L1*
and *is-language X Y L2*
shows (*semi-equivalence L1 L2*) $\longleftrightarrow$ (*L1* ≤[*X*, *semieq Y*] *L2*)
⟨*proof*⟩

lemma *strongsemired-type-2* :

assumes *is-language X Y L1*
and *is-language X Y L2*
shows (*strong-semi-reduction L1 L2*) $\longleftrightarrow$ (*L1* $\leq$[*X*, *strongsemired Y*] *L2*)
⟨*proof*⟩

lemma *strongsemieq-type-2* :
assumes *is-language X Y L1*
and *is-language X Y L2*
shows (*strong-semi-equivalence L1 L2*) $\longleftrightarrow$ (*L1* $\leq$[*X*, *strongsemieq Y*] *L2*)
⟨*proof*⟩

4 Comparing Conformance Relations

lemma *type-1-subset* :
assumes *L1* $\preceq$[*X*,*H1*] *L2*
and *H1* $\subseteq$ *H2*
shows *L1* $\preceq$[*X*,*H2*] *L2*
⟨*proof*⟩

lemma *type-1-subsets* :
shows *equiv Y* $\subseteq$ *strongred Y*
and *equiv Y* $\subseteq$ *quasieq Y*
and *strongred Y* $\subseteq$ *red Y*
and *strongred Y* $\subseteq$ *quasired Y*
and *quasieq Y* $\subseteq$ *quasired Y*
⟨*proof*⟩

lemma *type-1-implications* :
shows *L1* $\preceq$[*X*, *equiv Y*] *L2* $\Longrightarrow$ *L1* $\preceq$[*X*, *strongred Y*] *L2*
and *L1* $\preceq$[*X*, *equiv Y*] *L2* $\Longrightarrow$ *L1* $\preceq$[*X*, *red Y*] *L2*
and *L1* $\preceq$[*X*, *equiv Y*] *L2* $\Longrightarrow$ *L1* $\preceq$[*X*, *quasired Y*] *L2*
and *L1* $\preceq$[*X*, *equiv Y*] *L2* $\Longrightarrow$ *L1* $\preceq$[*X*, *quasieq Y*] *L2*
and *L1* $\preceq$[*X*, *strongred Y*] *L2* $\Longrightarrow$ *L1* $\preceq$[*X*, *red Y*] *L2*
and *L1* $\preceq$[*X*, *strongred Y*] *L2* $\Longrightarrow$ *L1* $\preceq$[*X*, *quasired Y*] *L2*
and *L1* $\preceq$[*X*, *quasieq Y*] *L2* $\Longrightarrow$ *L1* $\preceq$[*X*, *quasired Y*] *L2*
⟨*proof*⟩

lemma *type-2-subset* :
assumes *L1* $\leq$[*X*,*H1*] *L2*
and *H1* $\subseteq$ *H2*
shows *L1* $\leq$[*X*,*H2*] *L2*
⟨*proof*⟩

lemma *type-2-subsets* :
shows *strongsemieq Y* $\subseteq$ *strongsemired Y*
and *strongsemieq Y* $\subseteq$ *semieq Y*
and *semieq Y* $\subseteq$ *semired Y*

and *strongsemired* $Y \subseteq$ *semired* Y
and *strongsemired* $Y \subseteq$ *red* Y
⟨*proof*⟩

lemma *type-2-implications* :
shows $L1 \leq[X,$ *strongsemieq* $Y]$ $L2 \Longrightarrow L1 \leq[X,$ *strongsemired* $Y]$ $L2$
and $L1 \leq[X,$ *strongsemieq* $Y]$ $L2 \Longrightarrow L1 \leq[X,$ *semieq* $Y]$ $L2$
and $L1 \leq[X,$ *strongsemieq* $Y]$ $L2 \Longrightarrow L1 \leq[X,$ *semired* $Y]$ $L2$
and $L1 \leq[X,$ *strongsemired* $Y]$ $L2 \Longrightarrow L1 \leq[X,$ *semired* $Y]$ $L2$
and $L1 \leq[X,$ *semieq* $Y]$ $L2 \Longrightarrow L1 \leq[X,$ *semired* $Y]$ $L2$
⟨*proof*⟩

lemma *type-1-conformance-to-type-2* :
assumes *is-language* X Y $L2$
and $L1 \preceq[X,H1]$ $L2$
and $H1 \subseteq H2$
and $\bigwedge A\ B\ .\ (A,B) \in H1 \Longrightarrow B \neq \{\} \Longrightarrow A \cap B \neq \{\}$
shows $L1 \leq[X,H2]$ $L2$
⟨*proof*⟩

lemma *type-1-and-2-mixed-implications* :
assumes *is-language* X Y $L2$
shows $L1 \leq[X,$ *strongsemieq* $Y]$ $L2 \Longrightarrow L1 \preceq[X,$ *red* $Y]$ $L2$
and $L1 \leq[X,$ *strongsemired* $Y]$ $L2 \Longrightarrow L1 \preceq[X,$ *red* $Y]$ $L2$
and $L1 \preceq[X,$ *quasieq* $Y]$ $L2 \Longrightarrow L1 \leq[X,$ *semieq* $Y]$ $L2$
and $L1 \preceq[X,$ *quasired* $Y]$ $L2 \Longrightarrow L1 \leq[X,$ *semired* $Y]$ $L2$
and $L1 \preceq[X,$ *equiv* $Y]$ $L2 \Longrightarrow L1 \leq[X,$ *strongsemieq* $Y]$ $L2$
and $L1 \preceq[X,$ *strongred* $Y]$ $L2 \Longrightarrow L1 \leq[X,$ *strongsemired* $Y]$ $L2$
⟨*proof*⟩

4.1 Completely Specified Languages

definition *partiality-component* :: $'y$ *set* $\Rightarrow$ $'y$ *output-relation* **where**
partiality-component $Y = \{(A,\{\})\ |\ A\ .\ A \subseteq Y\} \cup \{(\{\},A)\ |\ A\ .\ A \subseteq Y\}$

abbreviation(*input*) $\Pi\ Y \equiv$ *partiality-component* Y

lemma *conformance-without-partiality* :
shows *strongsemieq* $Y - \Pi\ Y =$ *semieq* $Y - \Pi\ Y$
and *semieq* $Y - \Pi\ Y =$ *equiv* $Y - \Pi\ Y$
and *strongsemired* $Y - \Pi\ Y =$ *semired* $Y - \Pi\ Y$
and *semired* $Y - \Pi\ Y =$ *red* $Y - \Pi\ Y$
⟨*proof*⟩

5 Conformance Testing

type-synonym $('x,'y)$ *state-cover* = $('x,'y)$ *language*

type-synonym $('x,'y)$ *transition-cover* = $('x,'y)$ *state-cover* × *'x set*

fun *is-state-cover* :: $('x,'y)$ *language* ⇒ $('x,'y)$ *language* ⇒ $('x,'y)$ *state-cover* ⇒ *bool* **where**
is-state-cover L1 L2 V = (∀ π ∈ *L1* ∩ *L2* . ∃ α ∈ *V* . ℒ[*L1*,π] = ℒ[*L1*,α] ∧ ℒ[*L2*,π] = ℒ[*L2*,α])

lemma *state-cover-subset* :
assumes *is-language X Y L1*
and *is-language X Y L2*
and *is-state-cover L1 L2 V*
and π ∈ *L1* ∩ *L2*
obtains α **where** α ∈ *V*
and α ∈ *L1* ∩ *L2*
and ℒ[*L1*,π] = ℒ[*L1*,α]
and ℒ[*L2*,π] = ℒ[*L2*,α]
⟨*proof*⟩

theorem *sufficient-condition-for-type-1-conformance* :
assumes *is-language X Y L1*
and *is-language X Y L2*
and *is-state-cover L1 L2 V*
shows (*L1* ⪯[*X*,*H*] *L2*) ⟷ (∀ π ∈ *V* . ∀ *x* ∈ *X* . π ∈ *L1* ∩ *L2* ⟶ (*out*[*L1*,π,*x*], *out*[*L2*,π,*x*]) ∈ *H*)
⟨*proof*⟩

theorem *sufficient-condition-for-type-2-conformance* :
assumes *is-language X Y L1*
and *is-language X Y L2*
and *is-state-cover L1 L2 V*
shows (*L1* ≤[*X*,*H*] *L2*) ⟷ (∀ π ∈ *V* . ∀ *x* ∈ *X* . π ∈ *L1* ∩ *L2* ⟶ (*out*[*L1*,π,*x*], *out*[*L2*,π,*x*]) ∈ *H* ∧ (*out*[*L2*,π,*x*] ≠ {} ⟶ (∃ *x'* ∈ *X* . *out*[*L1*,π,*x'*] ∩ *out*[*L2*,π,*x'*] ≠ {})))
⟨*proof*⟩

lemma *intersections-card-helper* :
assumes *finite X*
and *finite Y*
shows *card* {*A* ∩ *B* | *A B* . *A* ∈ *X* ∧ *B* ∈ *Y*} ≤ *card X* ∗ *card Y*
⟨*proof*⟩

lemma *prefix-length-take* :
(*prefix xs ys* ∧ *length xs* ≤ *k*) ⟷ (*prefix xs* (*take k ys*))
⟨*proof*⟩

lemma *brute-force-state-cover* :
assumes *is-language X Y L1*
and *is-language X Y L2*
and *finite* $\{\mathcal{L}[L1,\pi] \mid \pi \,.\, \pi \in L1\}$
and *finite* $\{\mathcal{L}[L2,\pi] \mid \pi \,.\, \pi \in L2\}$
and *card* $\{\mathcal{L}[L1,\pi] \mid \pi \,.\, \pi \in L1\} \leq n$
and *card* $\{\mathcal{L}[L2,\pi] \mid \pi \,.\, \pi \in L2\} \leq m$
shows *is-state-cover L1 L2* $\{\alpha \,.\, length\ \alpha \leq m * n - 1 \wedge (\forall\ xy \in set\ \alpha \,.\, fst\ xy \in X \wedge snd\ xy \in Y)\}$
$\langle proof \rangle$

6 Reductions Between Relations

6.1 Quasi-Equivalence via Quasi-Reduction and Absences

fun *absence-completion* :: $'x\ alphabet \Rightarrow\ 'y\ alphabet \Rightarrow ('x,'y)\ language \Rightarrow ('x,\ 'y \times bool)\ language$ **where**
absence-completion X Y L =
$((\lambda\ \pi \,.\, map\ (\lambda(x,y) \,.\, (x,(y,True)))\ \pi)\ `\ L)$
$\cup\ \{(map\ (\lambda(x,y)\ .\ (x,(y,True)))\ \pi)@[(x,(y,False))]@\tau \mid \pi\ x\ y\ \tau\ .\ \pi \in L \wedge out[L,\pi,x] \neq \{\} \wedge y \in Y \wedge y \notin out[L,\pi,x] \wedge (\forall\ (x,(y,a)) \in set\ \tau\ .\ x \in X \wedge y \in Y)\}$

lemma *absence-completion-is-language* :
assumes *is-language X Y L*
shows *is-language X* $(Y \times UNIV)$ (*absence-completion X Y L*)
$\langle proof \rangle$

lemma *absence-completion-inclusion-R* :
assumes *is-language X Y L*
and $\pi \in$ *absence-completion X Y L*
shows $(map\ (\lambda(x,y,a)\ .\ (x,y))\ \pi \in L) \longleftrightarrow (\forall\ (x,y,a) \in set\ \pi\ .\ a = True)$
$\langle proof \rangle$

lemma *absence-completion-inclusion-L* :
$(\pi \in L) \longleftrightarrow (map\ (\lambda(x,y)\ .\ (x,y,True))\ \pi \in$ *absence-completion X Y L*$)$
$\langle proof \rangle$

fun *is-present* :: $('x,'y \times bool)\ word \Rightarrow ('x,'y)\ language \Rightarrow bool$ **where**
is-present $\pi\ L = (\pi \in map\ (\lambda(x,\ y).\ (x,\ y,\ True))\ `\ L)$

lemma *is-present-rev* :
assumes *is-present* $\pi\ L$
shows $map\ (\lambda(x,\ y,\ a).\ (x,\ y))\ \pi \in L$
$\langle proof \rangle$

lemma *absence-completion-out* :
assumes *is-language* X Y L
and $x \in X$
and $\pi \in$ *absence-completion* X Y L
shows *is-present* π $L \Longrightarrow$ *out*[L,*map* ($\lambda(x, y, a).\ (x, y)$) π,x] $\neq \{\} \Longrightarrow$ *out*[*absence-completion* X Y L, π, x] $= \{(y, True) \mid y\ .\ y \in$ *out*[L,*map* ($\lambda(x, y, a).\ (x, y)$) π,x]$\} \cup \{(y, False) \mid y\ .\ y \in Y \wedge y \notin$ *out*[L,*map* ($\lambda(x, y, a).\ (x, y)$) π,x]$\}$
and *is-present* π $L \Longrightarrow$ *out*[L,*map* ($\lambda(x, y, a).\ (x, y)$) π,x] $= \{\} \Longrightarrow$ *out*[*absence-completion* X Y L, π, x] $= \{\}$
and $\neg$ *is-present* π $L \Longrightarrow$ *out*[*absence-completion* X Y L, π, x] $= Y \times UNIV$
⟨*proof*⟩

theorem *quasieq-via-quasired* :
assumes *is-language* X Y $L1$
and *is-language* X Y $L2$
shows ($L1 \preceq$[X,*quasieq* Y] $L2$) $\longleftrightarrow$ ((*absence-completion* X Y $L1$) $\preceq$[X, *quasired* ($Y \times UNIV$)] (*absence-completion* X Y $L2$))
⟨*proof*⟩

6.2 Quasi-Reduction via Reduction and explicit Undefined Behaviour

fun *bottom-completion* :: $'x$ *alphabet* $\Rightarrow$ $'y$ *alphabet* $\Rightarrow$ ($'x$,$'y$) *language* $\Rightarrow$ ($'x$, $'y$ *option*) *language* **where**
bottom-completion X Y L =
(($\lambda\ \pi$. *map* ($\lambda(x,y)$. (x,*Some* y)) π) ‘ L)
$\cup$ {(*map* ($\lambda(x,y)$. (x,*Some* y)) π)@[(x,y)]@τ | π x y τ . $\pi \in L \wedge$ *out*[L,π,x] = {} $\wedge$ $x \in X$ $\wedge$ (y = *None* $\vee$ $y \in$ *Some* ‘ Y) $\wedge$ ($\forall$ (x,y) $\in$ *set* τ . $x \in X$ $\wedge$ (y = *None* $\vee$ $y \in$ *Some* ‘ Y))}

lemma *bottom-completion-is-language* :
assumes *is-language* X Y L
shows *is-language* X ({*None*} $\cup$ *Some* ‘ Y) (*bottom-completion* X Y L)
⟨*proof*⟩

fun *is-not-undefined* :: ($'x$,$'y$ *option*) *word* $\Rightarrow$ ($'x$,$'y$) *language* $\Rightarrow$ *bool* **where**
is-not-undefined π L = ($\pi \in$ *map* ($\lambda(x, y).\ (x,$ *Some* $y)$) ‘ L)

lemma *bottom-id* : *map* ($\lambda(x,y)$. (x, *the* y)) (*map* ($\lambda(x, y).\ (x,$ *Some* $y)$) π) = π
⟨*proof*⟩

fun *maximum-prefix-with-property* :: $('a\ list \Rightarrow bool) \Rightarrow 'a\ list \Rightarrow 'a\ list$ **where**
maximum-prefix-with-property P xs = (*last* (*filter P* (*prefixes xs*)))

lemma *maximum-prefix-with-property-props* :
assumes $\exists$ *ys* $\in$ *set* (*prefixes xs*) . *P ys*
shows *P* (*maximum-prefix-with-property P xs*)
and (*maximum-prefix-with-property P xs*) $\in$ *set* (*prefixes xs*)
and $\bigwedge$ *ys* . *prefix ys xs* $\Longrightarrow$ *P ys* $\Longrightarrow$ *length ys* $\leq$ *length* (*maximum-prefix-with-property P xs*)
⟨*proof*⟩

lemma *bottom-completion-out* :
assumes *is-language X Y L*
and $x \in X$
and $\pi \in$ *bottom-completion X Y L*
shows *is-not-undefined* π *L* $\Longrightarrow$ *out*[*L*,*map* (λ(*x*,*y*) . (*x*, *the y*)) π,*x*] $\neq$ {} $\Longrightarrow$ *out*[*bottom-completion X Y L*, π, *x*] = *Some* ‘ *out*[*L*, *map* (λ(*x*,*y*) . (*x*, *the y*)) π, *x*]
and *is-not-undefined* π *L* $\Longrightarrow$ *out*[*L*,*map* (λ(*x*,*y*) . (*x*, *the y*)) π,*x*] = {} $\Longrightarrow$ *out*[*bottom-completion X Y L*, π, *x*] = {*None*} $\cup$ *Some* ‘ *Y*
and $\neg$ *is-not-undefined* π *L* $\Longrightarrow$ *out*[*bottom-completion X Y L*, π, *x*] = {*None*} $\cup$ *Some* ‘ *Y*
⟨*proof*⟩

theorem *quasired-via-red* :
assumes *is-language X Y L1*
and *is-language X Y L2*
shows (*L1* $\preceq$[*X*,*quasired Y*] *L2*) $\longleftrightarrow$ ((*bottom-completion X Y L1*) $\preceq$[*X*, *red* ({*None*} $\cup$ *Some* ‘ *Y*)] (*bottom-completion X Y L2*))
⟨*proof*⟩

6.3 Strong Reduction via Reduction and Undefinedness Outputs

fun *non-bottom-shortening* :: $('x,'y\ option)\ word \Rightarrow ('x,'y\ option)\ word$ **where**
non-bottom-shortening π = *filter* (λ (*x*,*y*) . *y* $\neq$ *None*) π

fun *non-bottom-projection* :: $('x,'y\ option)\ word \Rightarrow ('x,'y)\ word$ **where**
non-bottom-projection π = *map* (λ(*x*,*y*) . (*x*,*the y*)) (*non-bottom-shortening* π)

lemma *non-bottom-projection-split*: *non-bottom-projection* (π'@π'') = (*non-bottom-projection* π')@(*non-bottom-projection* π'')
⟨*proof*⟩

lemma *non-bottom-projection-id* : *non-bottom-projection* (*map* (λ(*x*,*y*) . (*x*,*Some y*)) π) = π

⟨*proof*⟩

fun *undefinedness-completion* :: *'x alphabet* ⇒ (*'x*,*'y*) *language* ⇒ (*'x*, *'y option*) *language* **where**
  undefinedness-completion X L =
    {π . *non-bottom-projection* $\pi \in L \wedge (\forall\ \pi'\ x\ \pi''\ .\ \pi = \pi'$ @ [(*x*,*None*)] @ $\pi'' \longrightarrow x \in X \wedge$ *out*[*L*, *non-bottom-projection* π', *x*] = {})}

lemma *undefinedness-completion-is-language* :
  assumes *is-language X Y L*
shows *is-language X* ({*None*} ∪ *Some* ' *Y*) (*undefinedness-completion X L*)
⟨*proof*⟩

lemma *undefinedness-completion-inclusion* :
  assumes $\pi \in L$
shows *map* (λ(*x*,*y*) . (*x*,*Some y*)) $\pi \in$ *undefinedness-completion X L*
⟨*proof*⟩

lemma *undefinedness-completion-out-shortening* :
  assumes *is-language X Y L*
  and $\pi \in$ *undefinedness-completion X L*
  and $x \in X$
shows *out*[*undefinedness-completion X L*, π, *x*] = *out*[*undefinedness-completion X L*, *non-bottom-shortening* π, *x*]
⟨*proof*⟩

lemma *undefinedness-completion-out-projection-not-empty* :
  assumes *is-language X Y L*
  and $\pi \in$ *undefinedness-completion X L*
  and $x \in X$
  and *out*[*L*, *non-bottom-projection* π, *x*] ≠ {}
shows *out*[*undefinedness-completion X L*, *non-bottom-shortening* π, *x*] = *Some* ' *out*[*L*, *non-bottom-projection* π, *x*]
⟨*proof*⟩

lemma *undefinedness-completion-out-projection-empty* :
  assumes *is-language X Y L*
  and $\pi \in$ *undefinedness-completion X L*
  and $x \in X$
  and *out*[*L*, *non-bottom-projection* π, *x*] = {}
shows *out*[*undefinedness-completion X L*, *non-bottom-shortening* π, *x*] = {*None*}
⟨*proof*⟩

theorem *strongred-via-red* :
  assumes *is-language X Y L1*
  and *is-language X Y L2*
shows (*L1* ⪯[*X*,*strongred Y*] *L2*) ⟷ ((*undefinedness-completion X L1*) ⪯[*X*, *red* ({*None*} ∪ *Some* ‘ *Y*)] (*undefinedness-completion X L2*))
⟨*proof*⟩

end

References

[1] W.-l. Huang and R. Sachtleben. *Conformance Relations Between Input/Output Languages*, pages 49–67. Springer Nature Switzerland, Cham, 2023.