Conformance Relations between Input/Output Languages

Robert Sachtleben

February 6, 2026

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

1 Preliminaries **2**

2 Conformance Relations **4**

3 Unifying Characterisations **5**
3.1 $\preceq$ Conformance 5
3.2 $\leq$ Conformance 7

4 Comparing Conformance Relations **8**
4.1 Completely Specified Languages 9

5 Conformance Testing **9**

6 Reductions Between Relations **11**
6.1 Quasi-Equivalence via Quasi-Reduction and Absences 11
6.2 Quasi-Reduction via Reduction and explicit Undefined Behaviour 12
6.3 Strong Reduction via Reduction and Undefinedness Outputs 13

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* $(\langle \mathcal{L}[\text{-},\text{-}]\rangle)$

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* ⇒ ('*x*,'*y*) *word* ⇒ ('*x*,'*y*) *word* ⇒ *bool* **where**
are-equivalent-for-language L α β = (*L*[*L*,*α*] = *L*[*L*,*β*])

abbreviation(*input*) *input-projection π* ≡ *map fst π*
abbreviation(*input*) *output-projection π* ≡ *map snd π*
notation *input-projection* (‹[-]$_I$›)
notation *output-projection* (‹[-]$_O$›)

fun *is-executable* :: ('*x*,'*y*) *language* ⇒ ('*x*,'*y*) *word* ⇒ '*x list* ⇒ *bool* **where**
is-executable L π xs = (∃ *τ* ∈ *L*[*L*,*π*] . [*τ*]$_I$ = *xs*)

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

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

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

lemma *executable-sequences-alt-def* : *executable-sequences L π* = {*xs* . ∃ *ys* . *length ys* = *length xs* ∧ *zip xs ys* ∈ *L*[*L*,*π*]}
⟨*proof*⟩

lemma *executable-inputs-alt-def* : *executable-inputs L π* = {*x* . ∃ *y* . [(*x*,*y*)] ∈ *L*[*L*,*π*]}
⟨*proof*⟩

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

fun *output-sequences* :: ('*x*,'*y*) *language* ⇒ ('*x*,'*y*) *word* ⇒ '*x list* ⇒ '*y list set* **where**
output-sequences L π xs = *output-projection* ‘ {*τ* ∈ *L*[*L*,*π*] . [*τ*]$_I$ = *xs*}

lemma *prefix-closure-no-member* :
assumes *is-language X Y L*
and *π* ∉ *L*
shows *π*@*τ* ∉ *L*
⟨*proof*⟩

lemma *output-sequences-empty-iff* :
assumes *is-language X Y L*
shows (*output-sequences L π xs* = {}) = ((*π* ∉ *L*) ∨ (¬ *is-executable L π xs*))
⟨*proof*⟩

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

notation *outputs* (‹*out*[-,-,-]›)

lemma *outputs-in-alphabet* :
assumes *is-language X Y L*
shows *out*[*L*,*π*,*x*] ⊆ *Y*
⟨*proof*⟩

lemma *outputs-executable* : (*out*[*L*,*π*,*x*] = {}) ⟷ (*x* ∉ *exec*[*L*,*π*])
⟨*proof*⟩

fun *is-completely-specified-for* :: '*x set* ⇒ ('*x*,'*y*) *language* ⇒ *bool* **where**
is-completely-specified-for X L = (∀ *π* ∈ *L* . ∀ *x* ∈ *X* . *out*[*L*,*π*,*x*] ≠ {})

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

2 Conformance Relations

definition *language-equivalence* :: ('*x*,'*y*) *language* ⇒ ('*x*,'*y*) *language* ⇒ *bool* **where**
language-equivalence L1 L2 = (*L1* = *L2*)

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

language-inclusion L1 L2 = (*L1* ⊆ *L2*)

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

definition *quasi-equivalence* :: ('*x*,'*y*) *language* ⇒ ('*x*,'*y*) *language* ⇒ *bool* **where**
quasi-equivalence L1 L2 = (∀ *π* ∈ *L1* ∩ *L2* . ∀ *x* ∈ *exec*[*L2*,*π*] . *out*[*L1*,*π*,*x*] =

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

definition *quasi-reduction* :: $('x,'y)$ *language* $\Rightarrow$ $('x,'y)$ *language* $\Rightarrow$ *bool* **where**
quasi-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-reduction* :: $('x,'y)$ *language* $\Rightarrow$ $('x,'y)$ *language* $\Rightarrow$ *bool* **where**
strong-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-equivalence* :: $('x,'y)$ *language* $\Rightarrow$ $('x,'y)$ *language* $\Rightarrow$ *bool* **where**
semi-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-reduction* :: $('x,'y)$ *language* $\Rightarrow$ $('x,'y)$ *language* $\Rightarrow$ *bool* **where**
semi-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-semi-equivalence* :: $('x,'y)$ *language* $\Rightarrow$ $('x,'y)$ *language* $\Rightarrow$ *bool* **where**
strong-semi-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-semi-reduction* :: $('x,'y)$ *language* $\Rightarrow$ $('x,'y)$ *language* $\Rightarrow$ *bool* **where**
strong-semi-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-1-conforms* :: $('x,'y)$ *language* $\Rightarrow$ *'x alphabet* $\Rightarrow$ *'y output-relation* $\Rightarrow$ $('x,'y)$ *language* $\Rightarrow$ *bool* **where**
type-1-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-1-conforms* ($\langle$- $\preceq$[-,-] -$\rangle$)

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

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

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

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

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

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

lemma *red-type-1* :
assumes *is-language X Y L1*
and *is-language X Y L2*
shows *reduction L1 L2* ⟷ *(L1* ⪯*[X,red Y] L2)*
⟨*proof*⟩

lemma *equiv-by-reduction* : *(L1* ⪯*[X,equiv Y] L2)* ⟷ *((L1* ⪯*[X,red Y] L2)* ∧ *(L2* ⪯*[X,red Y] L1))*
⟨*proof*⟩

lemma *equiv-type-1* :
assumes *is-language X Y L1*
and *is-language X Y L2*
shows *(L1 = L2)* ⟷ *(L1* ⪯*[X,equiv Y] L2)*
⟨*proof*⟩

lemma *quasired-type-1* :
assumes *is-language X Y L1*
and *is-language X Y L2*
shows *quasi-reduction L1 L2* ⟷ *(L1* ⪯*[X,quasired Y] L2)*
⟨*proof*⟩

lemma *quasieq-type-1* :
assumes *is-language X Y L1*
and *is-language X Y L2*
shows *quasi-equivalence L1 L2* ⟷ *(L1* ⪯*[X,quasieq Y] L2)*
⟨*proof*⟩

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 $\leq$ Conformance

fun *type-2-conforms* :: ('*x*,'*y*) *language* $\Rightarrow$ '*x alphabet* $\Rightarrow$ '*y output-relation* $\Rightarrow$ ('*x*,'*y*) *language* $\Rightarrow$ *bool* **where**
type-2-conforms L1 X H L2 = (
($\forall$ $\pi \in$ *L1* $\cap$ *L2* . $\forall$ *x* $\in$ *X* . (*out*[*L1*,π,*x*],*out*[*L2*,π,*x*]) $\in$ *H*) $\wedge$
($\forall$ $\pi \in$ *L1* $\cap$ *L2* . *exec*[*L2*,π] $\neq$ {} $\longrightarrow$ ($\exists$ *x* . *out*[*L1*,π,*x*] $\cap$ *out*[*L2*,π,*x*] $\neq$ {})))

notation *type-2-conforms* (‹- $\leq$[-,-] -›)

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

fun *semired* :: '*y alphabet* $\Rightarrow$ '*y output-relation* **where**
semired Y = {(*A*,*B*) | *A B* . *A* $\subseteq$ *B* $\wedge$ *B* $\subseteq$ *Y*} $\cup$ {(*C*,{}) | *C* . *C* $\subseteq$ *Y*}

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

fun *strongsemired* :: '*y alphabet* $\Rightarrow$ '*y output-relation* **where**
strongsemired Y = {(*A*,*B*) | *A B* . *A* $\subseteq$ *B* $\wedge$ *B* $\subseteq$ *Y*}

lemma *strongsemieq-alt-def* : *strongsemieq Y* = *semieq Y* $\cap$ *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* $\leq$[*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* $\leq$[*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*) $\longleftrightarrow$ ($L1 \preceq[X,$ *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*) $\longleftrightarrow$ ($L1 \preceq[X,$ *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*) Π *Y* $\equiv$ *partiality-component Y*

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

5 Conformance Testing

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

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

fun *is-state-cover* :: $('x,'y)$ *language* $\Rightarrow$ $('x,'y)$ *language* $\Rightarrow$ $('x,'y)$ *state-cover* $\Rightarrow$ *bool* **where**
is-state-cover L1 L2 V $= (\forall\ \pi \in L1 \cap L2\ .\ \exists\ \alpha \in V\ .\ \mathcal{L}[L1,\pi] = \mathcal{L}[L1,\alpha] \wedge \mathcal{L}[L2,\pi] = \mathcal{L}[L2,\alpha])$

lemma *state-cover-subset* :
assumes *is-language X Y L1*
and *is-language X Y L2*
and *is-state-cover L1 L2 V*
and $\pi \in L1 \cap L2$
obtains α **where** $\alpha \in V$
and $\alpha \in L1 \cap L2$
and $\mathcal{L}[L1,\pi] = \mathcal{L}[L1,\alpha]$
and $\mathcal{L}[L2,\pi] = \mathcal{L}[L2,\alpha]$
⟨*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 $(L1 \preceq[X,H]\ L2) \longleftrightarrow (\forall\ \pi \in V\ .\ \forall\ x \in X\ .\ \pi \in L1 \cap L2 \longrightarrow (out[L1,\pi,x], out[L2,\pi,x]) \in H)$
⟨*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 $(L1 \leq[X,H]\ L2) \longleftrightarrow (\forall\ \pi \in V\ .\ \forall\ x \in X\ .\ \pi \in L1 \cap L2 \longrightarrow (out[L1,\pi,x], out[L2,\pi,x]) \in H \wedge (out[L2,\pi,x] \neq \{\} \longrightarrow (\exists\ x' \in X\ .\ out[L1,\pi,x'] \cap out[L2,\pi,x'] \neq \{\})))$
⟨*proof*⟩

lemma *intersections-card-helper* :
assumes *finite X*
and *finite Y*
shows *card* $\{A \cap B \mid A\ B\ .\ A \in X \wedge B \in Y\} \leq$ *card X* $*$ *card Y*
⟨*proof*⟩

lemma *prefix-length-take* :
$(prefix\ xs\ ys \wedge length\ xs \leq k) \longleftrightarrow (prefix\ xs\ (take\ k\ ys))$
⟨*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* $\{\alpha \,.\, length\ \alpha \leq m * n - 1 \wedge (\forall\ xy \in set\ \alpha \,.\, fst\ xy \in X \wedge snd\ xy \in Y)\}$
⟨*proof*⟩

6 Reductions Between Relations

6.1 Quasi-Equivalence via Quasi-Reduction and Absences

fun *absence-completion* :: *'x alphabet* ⇒ *'y alphabet* ⇒ (*'x*,*'y*) *language* ⇒ (*'x*, *'y* × *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*)
⟨*proof*⟩

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

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

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

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

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 =
($(\lambda\ \pi\ .\ map\ (\lambda(x,y)\ .\ (x,Some\ y))\ \pi)$ ‘ *L*)
$\cup \{(map\ (\lambda(x,y)\ .\ (x,Some\ y))\ \pi)@[(x,y)]@\tau \mid \pi\ x\ y\ \tau\ .\ \pi \in L \wedge$ *out*[*L*,π,*x*] = $\{\} \wedge x \in X \wedge (y = None \vee y \in Some$ ‘ *Y*) $\wedge$ ($\forall$ $(x,y) \in set\ \tau\ .\ x \in X \wedge (y = None \vee y \in Some$ ‘ *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* ⇒ *bool*) ⇒ *'a list* ⇒ *'a list* **where**
maximum-prefix-with-property P xs = (*last* (*filter P* (*prefixes xs*)))

lemma *maximum-prefix-with-property-props* :
assumes ∃ *ys* ∈ *set* (*prefixes xs*) . *P ys*
shows *P* (*maximum-prefix-with-property P xs*)
and (*maximum-prefix-with-property P xs*) ∈ *set* (*prefixes xs*)
and ⋀ *ys* . *prefix ys xs* ⟹ *P ys* ⟹ *length ys* ≤ *length* (*maximum-prefix-with-property P xs*)
⟨*proof*⟩

lemma *bottom-completion-out* :
assumes *is-language X Y L*
and *x* ∈ *X*
and *π* ∈ *bottom-completion X Y L*
shows *is-not-undefined π L* ⟹ *out*[*L*,*map* (λ(*x*,*y*) . (*x*, *the y*)) *π*,*x*] ≠ {} ⟹ *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* ⟹ *out*[*L*,*map* (λ(*x*,*y*) . (*x*, *the y*)) *π*,*x*] = {} ⟹ *out*[*bottom-completion X Y L*, *π*, *x*] = {*None*} ∪ *Some* ‘ *Y*
and ¬ *is-not-undefined π L* ⟹ *out*[*bottom-completion X Y L*, *π*, *x*] = {*None*} ∪ *Some* ‘ *Y*
⟨*proof*⟩

theorem *quasired-via-red* :
assumes *is-language X Y L1*
and *is-language X Y L2*
shows (*L1* ≼[*X*,*quasired Y*] *L2*) ⟷ ((*bottom-completion X Y L1*) ≼[*X*, *red* ({*None*} ∪ *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* ⇒ (*'x*,*'y option*) *word* **where**
non-bottom-shortening π = *filter* (λ (*x*,*y*) . *y* ≠ *None*) *π*

fun *non-bottom-projection* :: (*'x*,*'y option*) *word* ⇒ (*'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* π ∈ *L* ∧ (∀ π′ *x* π″ . π = π′ @ [(*x*,*None*)] @ π″ ⟶ *x* ∈ *X* ∧ *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 π ∈ *L*
shows *map* (λ(*x*,*y*) . (*x*,*Some y*)) π ∈ *undefinedness-completion X L*
⟨*proof*⟩

lemma *undefinedness-completion-out-shortening* :
assumes *is-language X Y L*
and π ∈ *undefinedness-completion X L*
and *x* ∈ *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 π ∈ *undefinedness-completion X L*
and *x* ∈ *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 π ∈ *undefinedness-completion X L*
and *x* ∈ *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* $\preceq$[*X*,*strongred Y*] *L2*) $\longleftrightarrow$ ((*undefinedness-completion X L1*) $\preceq$[*X*, *red* ({*None*} $\cup$ *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.