Abstract

This theory formalizes a commutation version of decreasing diagrams for Church-Rosser modulo. The proof follows Felgenhauer and van Oostrom (RTA 2013). The theory also provides important specializations, in particular van Oostrom’s conversion version (TCS 2008) of decreasing diagrams.

We follow the development described in [1]: Conversions are mapped to Greek strings, and we prove that whenever a local peak (or cliff) is replaced by a joining sequence from a locally decreasing diagram, then the corresponding Greek strings become smaller in a specially crafted well-founded order on Greek strings. Once there are no more local peaks or cliffs are left, the result is a valley that establishes the Church-Rosser modulo property.

As special cases we provide non-commutation versions and the conversion version of decreasing diagrams by van Oostrom [3]. We also formalize extended decreasingness [2].
1 Preliminaries

theory Decreasing-Diagrams-II-Aux
imports
  ../Well-Quasi-Orders/Multiset-Extension
  ../Well-Quasi-Orders/Well-Quasi-Orders
begin

1.1 Trivialities

lemma asymI2: (∀a b. (a,b) ∈ R ⇒ (b,a) ∉ R) ⇒ asym R
⟨proof⟩

abbreviation strict-order R ≡ irrefl R ∧ trans R

lemma order-asym: trans R ⇒ asym R = irrefl R
⟨proof⟩

lemma strict-order-strict: strict-order q ⇒ strict (λa b. (a, b) ∈ q") = (λa b. (a, b) ∈ q) 
⟨proof⟩

lemma mono-lex1: mono (λr. lex-prod r s)
⟨proof⟩

lemma mono-lex2: mono (lex-prod r)
⟨proof⟩

lemma irrefl-lex-prod: irrefl R ⇒ irrefl S ⇒ irrefl (R <*lex*> S)
⟨proof⟩
lemmas converse-inward = rtrancl-converse[symmetric] converse-Un converse-UNION
converse-relcomp
  converse-converse converse-Id

1.2 Complete lattices and least fixed points
context complete-lattice
begin

1.2.1 A chain-based induction principle
abbreviation set-chain :: 'a set ⇒ bool where
  set-chain C ≡ ∀x ∈ C. ∀y ∈ C. x ≤ y ∨ y ≤ x

lemma lfp-chain-induct:
  assumes mono: mono f
  and step: ∀x. P x ⇒ P (f x)
  and chain: ∀C. set-chain C ⇒ ∀x ∈ C. P x ⇒ P (Sup C)
  shows P (lfp f)
 ⟨proof⟩

1.2.2 Preservation of transitivity, asymmetry, irreflexivity by suprema
lemma trans-Sup-of-chain:
  assumes set-chain C and trans: ∀R. R ∈ C ⇒ trans R
  shows trans (Sup C)
 ⟨proof⟩

lemma asym-Sup-of-chain:
  assumes set-chain C and asym: ∀R. R ∈ C ⇒ asym R
  shows asym (Sup C)
 ⟨proof⟩

lemma strict-order-lfp:
  assumes mono f and ∀R. strict-order R ⇒ strict-order (f R)
  shows strict-order (lfp f)
 ⟨proof⟩

lemma trans-lfp:
  assumes mono f and ∀R. trans R ⇒ trans (f R)
  shows trans (lfp f)
 ⟨proof⟩

end

1.3 Multiset extension
lemma mulex-iff-mult: mulex r M N ←→ (M, N) ∈ mult {((M, N) . r M N}
 ⟨proof⟩
**lemma** multI:

assumes trans \( r \) \( M = I + K \) \( N = I + J \) \( J \neq \{\#\} \) \( \forall k \in \text{set-mset } K. \exists j \in \text{set-mset } J. (k,j) \in r \)

shows \((M,N) \in \text{mult } r\)

⟨proof⟩

**lemma** multE:

assumes trans \( r \) and \((M,N) \in \text{mult } r\)

obtains \( I J K \) where \( M = I + K \) \( N = I + J \) \( J \neq \{\#\} \) \( \forall k \in \text{set-mset } K. \exists j \in \text{set-mset } J. (k,j) \in r \)

⟨proof⟩

**lemma** mult-on-union:

\((M,N) \in \text{mult } r \implies (K + M, K + N) \in \text{mult } r\)

⟨proof⟩

**lemma** mult-on-union':

\((M,N) \in \text{mult } r \implies (M + K, N + K) \in \text{mult } r\)

⟨proof⟩

**lemma** mult-empty[simp]: \((M,\{\#\}) \notin \text{mult } R\)

⟨proof⟩

**lemma** mult-singleton[simp]: \((x, y) \in r \implies (\{\#x\#\}, \{\#y\#\}) \in \text{mult } r\)

⟨proof⟩

**lemma** empty-mult[simp]: \((\{\#\}, N) \in \text{mult } R \iff N \neq \{\#\}\)

⟨proof⟩

**lemma** trans-mult: trans \((\text{mult } R)\)

⟨proof⟩

**lemma** strict-order-mult:

assumes irrefl \( R \) and trans \( R \)

shows irrefl \((\text{mult } R)\) and trans \((\text{mult } R)\)

⟨proof⟩

**lemma** mult-of-image-mset:

assumes trans \( R \) and trans \( R' \)

and \( \forall x \ y. \ x \in \text{set-mset } N \implies y \in \text{set-mset } M \implies (x,y) \in R \implies (f \ x, f \ y) \in R' \)

and \((N, M) \in \text{mult } R\)

shows \((\text{image-mset } f \ N, \text{image-mset } f \ M) \in \text{mult } R'\)

⟨proof⟩

1.4 Incrementality of mult1 and mult

**lemma** mono-mult1: mono mult1

⟨proof⟩

**lemma** mono-mult: mono mult

⟨proof⟩
1.5 Well-orders and well-quasi-orders

**lemma** *wf-iff-wfp-on*:

\[ \text{wf } p \iff \text{wfp-on } (\lambda a b. (a, b) \in p) \text{ UNIV} \]

**proof**

**lemma** *well-order-implies-wqo*:

assumes well-order \( r \)

shows \( \text{wqo-on } (\lambda a b. (a, b) \in r) \text{ UNIV} \)

**proof**

1.6 Splitting lists into prefix, element, and suffix

**fun** *list-splits* :: `'a list ⇒ ('a list × 'a × list) list*

where

\[ \text{list-splits } \text{[ ]} = \text{[]} \]

\[ \text{list-splits } (x \# xs) = (\text{[]}, x, xs) \# \text{map } (\lambda (xs', x', ys'). (x \# xs, x', xs')) \text{ (list-splits } xs) \]

**lemma** *list-splits-empty*[simp]:

\[ \text{list-splits } xs = \text{[]} \iff xs = \text{[]} \]

**proof**

**lemma** *elem-list-splits-append*:

assumes \( (ys, y, zs) \in \text{set } \text{(list-splits } xs) \)

shows \( ys @ [y] @ zs = xs \)

**proof**

**lemma** *elem-list-splits-length*:

assumes \( (ys, y, zs) \in \text{set } \text{(list-splits } xs) \)

shows \( \text{length } ys < \text{length } xs \) and \( \text{length } zs < \text{length } xs \)

**proof**

**lemma** *elem-list-splits-elem*:

assumes \( (xs, y, ys) \in \text{set } \text{(list-splits } zs) \)

shows \( y \in \text{set } zs \)

**proof**

**lemma** *list-splits-append*:

\[ \text{list-splits } (xs @ ys) = \text{map } (\lambda (xs', x', ys'). (xs', x', ys' @ ys)) \text{ (list-splits } xs) @ \text{map } (\lambda (xs', x', ys'). (xs @ xs', x', ys')) \text{ (list-splits } ys) \]

**proof**

**lemma** *list-splits-rev*:

\[ \text{list-splits } (\text{rev } xs) = \text{map } (\lambda (xs, x, ys). (\text{rev } ys, x, \text{rev } xs)) \text{ (rev } \text{(list-splits } xs)) \]

**proof**

**lemma** *list-splits-map*:
list-splits \( (\text{map } f \; \text{xs}) = \text{map} \; (\lambda (\text{xs}, \; \text{x}, \; \text{ys}). \; (\text{map } f \; \text{xs}, \; f \; \text{x}, \; \text{map } f \; \text{ys})) \; \) (list-splits \text{xs})

\( \langle \text{proof} \rangle \)

end

2 Decreasing Diagrams

theory Decreasing-Diagrams-II

imports
  Decreasing-Diagrams-II-Aux
  ~~/src/HOL/Cardinals/Wellorder-Extension
  ../Abstract-Rewriting/Abstract-Rewriting

begin

2.1 Greek accents

datatype \text{accent} = \text{Acute} | \text{Grave} | \text{Macron}

lemma \text{UNIV-accent}: \text{UNIV} = \{ \text{Acute}, \text{Grave}, \text{Macron} \} 

\( \langle \text{proof} \rangle \)

lemma \text{finite-accent}: finite (\text{UNIV} :: \text{accent} set)

\( \langle \text{proof} \rangle \)

type-synonym \text{'a letter} = \text{accent} \times \text{'a}

definition \text{letter-less} :: (\text{'a} \times \text{'a}) \text{ set} \Rightarrow (\text{'a} \text{ letter} \times \text{'a} \text{ letter}) \text{ set where}

\[ \text{simp}: \text{letter-less} \; R = \{(a,b). \; (\text{snd} \; a, \; \text{snd} \; b) \in \; R\} \]

lemma \text{mono-letter-less}: \text{mono letter-less}

\( \langle \text{proof} \rangle \)

2.2 Comparing Greek strings

type-synonym \text{'a greek} = \text{'a letter list}

definition \text{adj-msog} :: \text{'a greek} \Rightarrow \text{'a greek} \Rightarrow (\text{'a letter} \times \text{'a greek}) \Rightarrow (\text{'a letter} \times \text{'a greek})

\text{where}

\[ \text{adj-msog} \; \text{xs} \; \text{zs} \; \text{l} \equiv \]

\text{case} \; \text{l} \; \text{of} \; \{(y,ys) \Rightarrow (y, \; \text{case} \; \text{fst} \; y \; \text{of} \; \text{Acute} \Rightarrow \; \text{ys} \; @ \; \text{zs} \; | \; \text{Grave} \Rightarrow \; \text{xs} \; @ \; \text{ys} \; | \; \text{Macron} \Rightarrow \; \text{ys}\} \]

definition \text{ms-of-greek} :: \text{'a greek} \Rightarrow (\text{'a letter} \times \text{'a greek}) \text{ multiset where}

\text{ms-of-greek as} = \text{mset}

\( (\text{map} \; (\lambda (\text{x}, \; \text{y}, \; \text{zs}) \Rightarrow \text{adj-msog} \; \text{xs} \; \text{zs} \; (y, \; [])) \; \text{(list-splits as)}) \)

lemma \text{adj-msog-adj-msog[simp]}:
\[
\text{adj-msog } xs \ zs (\text{adj-msog } xs' \ zs' \ y) = \text{adj-msog} (xs @ xs') (zs' @ zs) \ y
\]

\textit{lemma} \texttt{compose-adj-msog}\[\text{simp}]: \text{adj-msog} \ zs \ zs' \ adj-msog \ xs \ xs' = \text{adj-msog} \ (xs @ xs') \ (zs' @ zs)

\textit{lemma} \texttt{adj-msog-single}: 
\text{adj-msog} \ zs \ xs \ (x,[]) = (x, (\text{case } \text{fst} \ x \ of \ \text{Grave} \Rightarrow xs | \text{Acute} \Rightarrow zs | \text{Macron} \Rightarrow []))

\textit{lemma} \texttt{ms-of-greek-elem}: 
\texttt{assumes} (x,xs) \in \text{set-mset} \ (\text{ms-of-greek} \ ys) 
\texttt{shows} x \in \text{set} \ ys

\textit{lemma} \texttt{ms-of-greek-shorter}: 
\texttt{assumes} (x, t) \notin \# \ (\text{ms-of-greek} \ s) 
\texttt{shows} \text{length} \ s \succ \text{length} \ t

\textit{lemma} \texttt{msog-append}: \text{ms-of-greek} \ (xs @ ys) = \text{image-mset} \ (\text{adj-msog} \ [ ] \ ys) \ (\text{ms-of-greek} \ xs) + 
\text{image-mset} \ (\text{adj-msog} \ xs \ []) \ (\text{ms-of-greek} \ ys)

\textit{definition} \texttt{nest} :: ('a \times 'a) set \Rightarrow ('a \text{ greek} \times 'a \text{ greek}) set 
\texttt{where} 
\texttt{simp}: \text{nest} \ r \ s = \{(a,b). \ (\text{ms-of-greek} \ a, \text{ms-of-greek} \ b) \in \text{mult} \ (\text{letter-less} \ r \ \text{<*lex*>} \ s)\}

\textit{lemma} \texttt{mono-nest}: \text{mono} \ (\text{nest} \ r)

\textit{lemma} \texttt{nest-mono}[mono-set]: x \subseteq y \Rightarrow (a,b) \in \text{nest} \ r \ x \Rightarrow (a,b) \in \text{nest} \ r \ y

\textit{definition} \texttt{greek-less} :: ('a \times 'a) set \Rightarrow ('a \text{ greek} \times 'a \text{ greek}) set 
\texttt{where} 
\texttt{greek-less} \ r = \text{lfp} \ (\text{nest} \ r)

\textit{lemma} \texttt{greek-less-unfold}: 
\texttt{greek-less} \ r = \text{nest} \ r \ (\text{greek-less} \ r)

\text{2.3 Preservation of strict partial orders}

\textit{lemma} \texttt{strict-order-letter-less}: 
\texttt{assumes} \text{strict-order} \ r
\textbf{shows} strict-order (letter-less \( r \))
\textit{\langle proof \rangle}

\textbf{lemma} strict-order-nest:
\textbf{assumes} \( r: \text{strict-order} \) \textbf{and} \( R: \text{strict-order} \) \( R \) \( \text{shows} \) strict-order (nest \( r \) \( R \))
\textit{\langle proof \rangle}

\textbf{lemma} strict-order-greek-less:
\textbf{assumes} strict-order \( r \) \( \text{shows} \) strict-order (greek-less \( r \))
\textit{\langle proof \rangle}

\textbf{lemma} trans-letter-less:
\textbf{assumes} trans \( r \) \( \text{shows} \) trans (letter-less \( r \))
\textit{\langle proof \rangle}

\textbf{lemma} trans-order-nest: trans (nest \( r \) \( R \))
\textit{\langle proof \rangle}

\textbf{lemma} trans-greek-less[simp]: trans (greek-less \( r \))
\textit{\langle proof \rangle}

\textbf{lemma} mono-greek-less: mono greek-less
\textit{\langle proof \rangle}

\textbf{2.4 Involution}

\textbf{definition} inv-letter :: \"a letter\ \Rightarrow\ \text{\"a letter}\ where
\begin{align*}
\text{inv-letter } l & \equiv \\
\text{case } l \text{ of } (a, x) & \Rightarrow (\text{case } a \text{ of Grave } \Rightarrow \text{Acute} \text{ | Acute } \Rightarrow \text{Grave} \text{ | Macron } \Rightarrow \text{Macron}, x)
\end{align*}

\textbf{lemma} inv-letter-pair[simp]:
\begin{align*}
\text{inv-letter } (a, x) & = (\text{case } a \text{ of Grave } \Rightarrow \text{Acute} \text{ | Acute } \Rightarrow \text{Grave} \text{ | Macron } \Rightarrow \text{Macron}, x)
\end{align*}
\textit{\langle proof \rangle}

\textbf{lemma} snd-inv-letter[simp]:
\begin{align*}
\text{snd } (\text{inv-letter } x) & = \text{snd } x
\end{align*}
\textit{\langle proof \rangle}

\textbf{lemma} inv-letter-inv[1: simp]:
\begin{align*}
\text{inv-letter } (\text{inv-letter } x) & = x
\end{align*}
\textit{\langle proof \rangle}

\textbf{lemma} inv-letter-mono[1: simp]:
\textbf{assumes} \((x, y) \in \text{letter-less } r \)
shows \((\text{inv-letter } x, \text{inv-letter } y) \in \text{letter-less } r\)

\[\begin{align*}
\text{definition } \text{inv-greek} :: & \quad {'} \text{a greek} \Rightarrow {'} \text{a greek} \quad \text{where} \\
& \text{inv-greek } s = \text{rev } (\text{map } \text{inv-letter } s)
\end{align*}\]

\text{lemma } \text{inv-greek-invol}[\text{simp}]:
\text{inv-greek } (\text{inv-greek } s) = s

\[\begin{align*}
\text{lemma } \text{inv-greek-append}: \\
& \text{inv-greek } (s \mathbin{@} t) = \text{inv-greek } t \mathbin{@} \text{inv-greek } s
\end{align*}\]

\[\begin{align*}
\text{definition } \text{inv-msog} :: & \quad (\text{'a letter} \times \text{'a greek}) \text{ multiset} \Rightarrow (\text{'a letter} \times \text{'a greek}) \\
\text{multiset} \quad \text{where} \\
& \text{inv-msog } M = \text{image-mset } (\lambda(x, t). (\text{inv-letter } x, \text{inv-greek } t)) M
\end{align*}\]

\text{lemma } \text{inv-msog-invol}[\text{simp}]:
\text{inv-msog } (\text{inv-msog } M) = M

\[\begin{align*}
\text{lemma } \text{ms-of-greek-inv-greek}: \\
& \text{ms-of-greek } (\text{inv-greek } M) = \text{inv-msog } (\text{ms-of-greek } M)
\end{align*}\]

\[\begin{align*}
\text{lemma } \text{inv-greek-mono}: \\
& \text{assumes } \text{trans } r \text{ and } (s, t) \in \text{greek-less } r \\
& \text{shows } (\text{inv-greek } s, \text{inv-greek } t) \in \text{greek-less } r
\end{align*}\]

\[\begin{align*}
\text{2.5 Monotonicity of } \text{greek-less } r
\end{align*}\]

\[\begin{align*}
\text{lemma } \text{greek-less-rempty}[\text{simp}]: \\
& (a,\text{[]}) \in \text{greek-less } r \iff \text{False}
\end{align*}\]

\[\begin{align*}
\text{lemma } \text{greek-less-nonempty}: \\
& \text{assumes } b \neq \text{[]} \\
& \text{shows } (a, b) \in \text{greek-less } r \iff (a, b) \in \text{nest } (\text{greek-less } r)
\end{align*}\]

\[\begin{align*}
\text{lemma } \text{greek-less-lempty}[\text{simp}]: \\
& ([\text{[]}], b) \in \text{greek-less } r \iff b \neq \text{[]}
\end{align*}\]

\[\begin{align*}
\text{lemma } \text{greek-less-singleton}: \\
& (a, b) \in \text{letter-less } r \Rightarrow ([a], [b]) \in \text{greek-less } r
\end{align*}\]
lemma \textit{ms-of-greek-cons}:
\[
\text{ms-of-greek } (x \# s) = \{ \# \text{ adj-msog } [[x,[]] \# \} + \text{ image-mset } (\text{adj-msog } [x] [])(\text{ms-of-greek } s)
\]
\end{proof}

lemma \textit{greek-less-cons-mono}:
\begin{itemize}
\item \textbf{assumes} trans \( r \)
\item \textbf{shows} \((s, t) \in \text{greek-less } r \Rightarrow (x \# s, x \# t) \in \text{greek-less } r\)
\end{itemize}
\end{proof}

lemma \textit{greek-less-app-mono2}:
\begin{itemize}
\item \textbf{assumes} trans \( r \) and \((s, t) \in \text{greek-less } r\)
\item \textbf{shows} \((p @ s, p @ t) \in \text{greek-less } r\)
\end{itemize}
\end{proof}

lemma \textit{greek-less-app-mono1}:
\begin{itemize}
\item \textbf{assumes} trans \( r \) and \((s, t) \in \text{greek-less } r\)
\item \textbf{shows} \((s @ p, t @ p) \in \text{greek-less } r\)
\end{itemize}
\end{proof}

2.6 Well-founded-ness of \textit{greek-less} \( r \)

lemma \textit{greek-embed}:
\begin{itemize}
\item \textbf{assumes} trans \( r \)
\item \textbf{shows} list-emb \((\lambda a b. (a, b): \text{reficl } (\text{letter-less } r)) a b \Rightarrow (a, b) \in \text{reficl } (\text{greek-less } r)\)
\end{itemize}
\end{proof}

lemma \textit{wqo-letter-less}:
\begin{itemize}
\item \textbf{assumes} \( t : \text{trans } r \) and \( w : \text{wqo-on } (\lambda a b. (a, b) \in r^-) \text{ UNIV}\)
\item \textbf{shows} \( \text{wqo-on } (\lambda a b. (a, b) \in (\text{letter-less } r)^-) \text{ UNIV}\)
\end{itemize}
\end{proof}

lemma \textit{uf-greek-less}:
\begin{itemize}
\item \textbf{assumes} \( \text{uf } r \) and \( \text{trans } r \)
\item \textbf{shows} \( \text{uf } (\text{greek-less } r)\)
\end{itemize}
\end{proof}

2.7 Basic Comparisons

lemma \textit{pairwise-imp-mult}:
\begin{itemize}
\item \textbf{assumes} trans \( r \) and \( N \neq \{\#\} \) and \( \forall x \in \text{set-mset } M. \exists y \in \text{set-mset } N. (x, y) \in r\)
\item \textbf{shows} \( (M, N) \in \text{mult } r\)
\end{itemize}
\end{proof}

lemma \textit{singleton-greek-less}:
\begin{itemize}
\item \textbf{assumes} trans \( r \) and \( \text{as: snd } \text{ set as } \subseteq \text{ under } r b\)
\item \textbf{shows} \( (\text{as, } [(a,b)]) \in \text{greek-less } r\)
\end{itemize}
lemma peak-greek-less:
  assumes trans r
  and as: snd \set as \subseteq under r a and b': b' ∈ \{((Grave,b)),\}\n  and cs: snd \set cs \subseteq under r a ∪ under r b and a': a' ∈ \{((Acute,a)),\}\n  and bs: snd \set bs \subseteq under r b
  shows (as @ b' @ cs @ a' @ bs, [(Acute,a),(Grave,b)]) ∈ greek-less r

lemma rcliff-greek-less1:
  assumes trans r
  and as: snd \set as \subseteq under r a and b': b' ∈ \{((Grave,b)),\}\n  and cs: snd \set cs \subseteq under r b and a': a' = [((Macron,a)]
  and bs: snd \set bs \subseteq under r b
  shows (as @ b' @ cs @ a' @ bs, [(Macron,a),(Grave,b)]) ∈ greek-less r

lemma rcliff-greek-less2:
  assumes trans r
  and as: snd \set as \subseteq under r a and b': b' ∈ \{((Grave,b)),\}\n  and bs: snd \set bs \subseteq under r a and a': a' ∈ \{((Macron,a)),\}\n  and cs: snd \set cs \subseteq under r b
  shows (as @ b' @ cs @ a' @ bs, [(Macron,a),(Grave,b)]) ∈ greek-less r

lemma snd-inv-greek [simp]: snd \set (inv-greek as) = snd \set as

lemma lcliff-greek-less1:
  assumes trans r
  and as: snd \set as \subseteq under r a and b': b' = [((Macron,b)]
  and cs: snd \set cs \subseteq under r a and a': a' ∈ \{((Acute,a)),\}\n  and bs: snd \set bs \subseteq under r b
  shows (as @ b' @ cs @ a' @ bs, [(Acute,a),(Macron,b)]) ∈ greek-less r

lemma lcliff-greek-less2:
  assumes trans r
  and cs: snd \set cs \subseteq under r a and a': a' ∈ \{((Acute,a)),\}\n  and bs: snd \set bs \subseteq under r b
  shows (cs @ a' @ bs, [(Acute,a),(Macron,b)]) ∈ greek-less r

2.8 Labeled abstract rewriting

case
  fixes L R E :: 'b ⇒ 'a rel
begin
**definition lstep :: 'b letter ⇒ 'a rel where**

\[ \text{simp: lstep x} = \text{(case x of (a, i) ⇒ (case a of Acute ⇒ (L i)^-1 | Grave ⇒ R i | Macron ⇒ E i))} \]

**fun lconv :: 'b greek ⇒ 'a rel where**

\[
\begin{align*}
lconv & [] = \text{Id} \\
lconv (x # xs) &= \text{lstep x O lconv xs}
\end{align*}
\]

**lemma lconv-append[simp]:**

\[
lconv (xs @ ys) = lconv xs O lconv ys
\]

**proof**

**lemma conversion-join-or-peak-or-cliff:**

\[
\begin{align*}
\text{obtains (join) as bs cs where set as} & \subseteq \{\text{Grave}\} \text{ and set bs} \subseteq \{\text{Macron}\} \text{ and set cs} \subseteq \{\text{Acute}\} \\
\text{and ds} &= \text{as @ bs @ cs} \\
\text{| (peak) as bs where ds} &= \text{as @ ([Acute] @ [Grave]) @ bs} \\
\text{| (lcliff) as bs where ds} &= \text{as @ ([Acute] @ [Macron]) @ bs} \\
\text{| (rcliff) as bs where ds} &= \text{as @ ([Macron] @ [Grave]) @ bs}
\end{align*}
\]

**proof**

**lemma map-eq-append-split:**

\[
\begin{align*}
\text{assumes} \text{map f xs} & = \text{ys1 @ ys2} \\
\text{obtains} \text{xs1 xs2 where} \text{ys1} &= \text{map f xs1} \text{ ys2} = \text{map f xs2} \text{ xs} &= \text{xs1 @ xs2}
\end{align*}
\]

**proof**

**lemmas** map-eq-append-splits = map-eq-append-split map-eq-append-split[OF sym]

**abbreviation conversion' M ≡ (≤ i ∈ M. R i) ∪ (≤ i ∈ M. E i) ∪ (≤ i ∈ M. L i)^{-1})*

**abbreviation valley' M ≡ (∪ i ∈ M. R i)* O (∪ i ∈ M. E i)* O (∪ i ∈ M. L i)^{-1})*

**lemma conversion-to-lconv:**

\[
\begin{align*}
\text{assumes (u, v) ∈ conversion' M} \\
\text{obtains xs where} \text{snd ' set xs} & \subseteq M \text{ and (u, v) ∈ lconv xs}
\end{align*}
\]

**proof**

**definition lpeak :: 'b rel ⇒ 'b greek ⇒ 'b ⇒ 'b ⇒ bool where**

\[
\begin{align*}
lpeak & r a b xs \leftrightarrow (\exists \text{as b' cs a' bs. snd ' set as} \subseteq \text{under r a} \land b' \in \{([\text{Grave}, b],[])\} \\
\text{and sn}d ' & \text{ set cs} \subseteq \text{under r a} \cup \text{under r b} \land a' \in \{([\text{Acute}, a],[])\} \land \text{snd ' set bs} \subseteq \text{under r b} \land xs = \text{as @ b' @ cs @ a' @ bs})
\end{align*}
\]

**definition lcliff :: 'b rel ⇒ 'b ⇒ 'b ⇒ bool where**

\[
\begin{align*}
lcliff & r a b xs \leftrightarrow (\exists as b' cs a' bs. \text{snd ' set as} \subseteq \text{under r a} \land b' = ([\text{Macron}, b]) \\
\text{and sn}d ' & \text{ set cs} \subseteq \text{under r a} \land a' \in \{([\text{Acute}, a],[])\} \land \text{snd ' set bs} \subseteq \text{under r a} \land \text{under r b} \land xs = \text{as @ b' @ cs @ a' @ bs}) \lor
\end{align*}
\]

12
(∃cs a’ bs. snd ‘ set cs ⊆ under r a ∪ under r b ∧ a’ ∈ \{[(\text{Acute},a)],[]\} ∧
snd ‘ set bs ⊆ under r b ∧ xs = cs @ a’ @ bs)

**definition** rcliff ::= 'b rel ⇒ 'b ⇒ 'b greek ⇒ bool where
rcliff r a b xs ←→ (∃as b’ cs a’ bs. snd ‘ set as ⊆ under r a ∩ under r b ∧ b’ ∈
\{[(\text{Grave},b)],[]\} ∧
snd ‘ set cs ⊆ under r b ∧ a’ = [(\text{Macron},a)] ∧
snd ‘ set bs ⊆ under r b ∧ xs = as @ b’ @ cs @ a’ @ bs) ∨
(∃as b’ cs. snd ‘ set as ⊆ under r a ∧ b’ ∈ \{[(\text{Grave},b)],[]\} ∧
snd ‘ set cs ⊆ under r a ∪ under r b ∧ xs = as @ b’ @ cs)

**lemma** dd-commute-modulo-conv[case-names wf trans peak lcliff rcliff]:
assumes \(wf \ r \ \text{and} \ trans \ r\)
and pk: \(\forall a \ b \ s \ t \ u. (s, t) ∈ L a \implies (s, u) ∈ R b \implies ∃xs. \ lpeak r a b xs \land (t, u) ∈ lcone xs\)
and lc: \(\forall a \ b \ s \ t \ u. (s, t) ∈ L a \implies (s, u) ∈ E b \implies ∃xs. \ lcliff r a b xs \land (t, u) ∈ lcone xs\)
and rc: \(\forall a \ b \ s \ t \ u. (s, t) ∈ (E a)^{-1} \implies (s, u) ∈ R b \implies ∃xs. \ rcliff r a b xs \land
(t, u) ∈ lconv xs\)
shows conversion’ UNIV ⊆ valley’ UNIV

(proof)

3 Results

3.1 Church-Rosser modulo

Decreasing diagrams for Church-Rosser modulo, commutation version.

**lemma** dd-commute-modulo[case-names wf trans peak lcliff rcliff]:
assumes \(wf \ r \ \text{and} \ trans \ r\)
and pk: \(\forall a \ b \ s \ t \ u. (s, t) ∈ L a \implies (s, u) ∈ R b \implies ∃xs. \ lpeak r a b xs \land (t, u) ∈ lcone xs\)
and lc: \(\forall a \ b \ s \ t \ u. (s, t) ∈ L a \implies (s, u) ∈ E b \implies ∃xs. \ lcliff r a b xs \land (t, u) ∈ lcone xs\)
and rc: \(\forall a \ b \ s \ t \ u. (s, t) ∈ (E a)^{-1} \implies (s, u) ∈ R b \implies ∃xs. \ rcliff r a b xs \land
(t, u) ∈ lconv xs\)
shows conversion’ UNIV ⊆ valley’ UNIV

(proof)
Decreasing diagrams for Church-Rosser modulo.

**Lemma** `dd-cr-modulo`[
`case-names` `wf` `trans` `symE` `peak` `cliff`]:

**Assumes** `wf r` and `trans r` and `E`:

- `\( \wedge i. \text{sym} (E i) \)`
- `\( \text{pk: } \bigwedge a b s t u. (s, t) \in L a \implies (s, u) \in L b \implies (t, u) \in \text{conversion}' L L E \text{ (under } r a) O (L b)' O \text{ conversion}' L L E \text{ (under } r a \cup \text{ under } r b) O \)`

**And** `\( (L a)^{-1}' = O \text{ conversion}' L L E \text{ (under } r b) \)`

**And** `\( cl: \bigwedge a b s t u. (s, t) \in L a \implies (s, u) \in E b \implies (t, u) \in \text{conversion}' L L E \text{ (under } r a) O E b O \text{ conversion}' L L E \text{ (under } r a \cap \text{ under } r b) \)`

**Shows** `\( \text{conversion}' L L E \text{ UNIV } \subseteq \text{valley}' L L E \text{ UNIV} \)`

⟨**Proof**⟩

3.2 Commutation and confluence

**Abbreviation** `conversion'' L R M` `equiv` `((\bigcup i \in M. R i) \cup (\bigcup i \in M. L i)^{-1})`

**Abbreviation** `valley'' L R M` `equiv` `((\bigcup i \in M. R i)^* O ((\bigcup i \in M. L i)^{-1})^*`  

Decreasing diagrams for commutation.

**Lemma** `dd-commute`[
`case-names` `wf` `trans` `peak`]:

**Assumes** `wf r` and `trans r` and `trans q` and `refl q` and `compat`:

`\( r O q \subseteq r \)`

**And** `\( (L a)^{-1}' = O \text{ conversion}' L L E \text{ (under } r a \cup \text{ under } r b) \)`

**And** `\( (t, u) \in \text{conversion}' L L E \text{ (under } r a \cap \text{ under } r b) \)`

**Shows** `\( \text{commute} (\bigcup i . L i) (\bigcup i . R i) \)`

⟨**Proof**⟩

Decreasing diagrams for confluence.

**Lemmas** `dd-cr`[
`case-names` `wf` `trans` `peak`] =  
`dd-commute[of - L L for L, unfolded CR-iff-self-commute[symmetric]]`

3.3 Extended decreasing diagrams

**Context**

**Fixes** `r q :: 'b rel`

**Assumes** `wf r` and `trans r` and `trans q` and `refl q` and `compat: r O q \subseteq r`

**Begin**

**Private Abbreviation** `(input) down` `equiv` `(\'b \Rightarrow \'a rel) \Rightarrow (\'b \Rightarrow \'a rel)` **where**  
`\( \text{down } L \equiv \lambda i. \bigcup j \in \text{under } q i . L j \)`

**Private** **Lemma** `Union-down`:`\( \bigcup i . \text{down } L i = (\bigcup i . L i) \)`

⟨**Proof**⟩

Extended decreasing diagrams for confluence.
lemma edd-commute[case-names wf transr transq reflq compat peak]:
  assumes pk: \( a b s t u \). (s, t) \( \in L a \implies (s, u) \in R b \implies \)
  \( (t, u) \in \text{conversion}'' L R \) \( (\text{under } r a)\) \( O \) \( (\text{down } R b)'' O \) \( \text{conversion}'' L R \) \( (\text{under } r a \cup \text{under } r b)\) \( O \)
  \( (\text{down } L a)^{-1})= O \) \( \text{conversion}'' L R \) \( (\text{under } r b)\)
  shows commute \( (\bigcup i. L i) \) \( (\bigcup i. R i)\)
(\langle proof\rangle

Extended decreasing diagrams for confluence.

lemmas edd-cr[case-names wf transr transq reflq compat peak] =
edd-commute[of L L for L, unfolded CR-iff-self-commute[symmetric]]
\end
\end

References

