

# Decreasing-Diagrams-II

By Bertram Felgenhauer

May 26, 2024

## 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].

## Contents

<b>1 Preliminaries</b>	<b>2</b>
1.1 Trivialities . . . . .	2
1.2 Complete lattices and least fixed points . . . . .	2
1.2.1 A chain-based induction principle . . . . .	2
1.2.2 Preservation of transitivity, asymmetry, irreflexivity by suprema . . . . .	3
1.3 Multiset extension . . . . .	4
1.4 Incrementality of <i>mult1</i> and <i>mult</i> . . . . .	5
1.5 Well-orders and well-quasi-orders . . . . .	6
1.6 Splitting lists into prefix, element, and suffix . . . . .	6
<b>2 Decreasing Diagrams</b>	<b>7</b>
2.1 Greek accents . . . . .	7
2.2 Comparing Greek strings . . . . .	8
2.3 Preservation of strict partial orders . . . . .	9

2.4	Involution . . . . .	10
2.5	Monotonicity of <i>greek-less</i> $r$ . . . . .	12
2.6	Well-founded-ness of <i>greek-less</i> $r$ . . . . .	13
2.7	Basic Comparisons . . . . .	14
2.8	Labeled abstract rewriting . . . . .	19
<b>3</b>	<b>Results</b>	<b>24</b>
3.1	Church-Rosser modulo . . . . .	24
3.2	Commutation and confluence . . . . .	27
3.3	Extended decreasing diagrams . . . . .	27

## 1 Preliminaries

**theory** *Decreasing-Diagrams-II-Aux*

**imports**

*Well-Quasi-Orders.Multiset-Extension*

*Well-Quasi-Orders.Well-Quasi-Orders*

**begin**

### 1.1 Trivialities

**abbreviation** *strict-order*  $R \equiv \text{irrefl } R \wedge \text{trans } R$

**lemma** *strict-order-strict*:  $\text{strict-order } q \implies \text{strict } (\lambda a b. (a, b) \in q^{\bar{=}}) = (\lambda a b. (a, b) \in q)$

**unfolding** *trans-def irrefl-def* **by** *fast*

**lemma** *mono-lex1*:  $\text{mono } (\lambda r. \text{lex-prod } r \text{ } s)$

**by** (*auto simp add: mono-def*)

**lemma** *mono-lex2*:  $\text{mono } (\text{lex-prod } r)$

**by** (*auto simp add: mono-def*)

**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*  $:: 'a \text{ set} \Rightarrow \text{bool}$  **where**

*set-chain*  $C \equiv \forall x \in C. \forall y \in C. x \leq y \vee y \leq x$

**lemma** *lfp-chain-induct*:

**assumes** *mono*: *mono*  $f$

**and** *step*:  $\bigwedge x. P\ x \implies P\ (f\ x)$

**and** *chain*:  $\bigwedge C. \text{set-chain } C \implies \forall x \in C. P\ x \implies P\ (\text{Sup } C)$

**shows**  $P\ (\text{lfp } f)$

**unfolding** *lfp-eq-fixp*[*OF mono*]

**proof** (*rule fixp-induct*)

**show** *monotone*  $(\leq)$   $(\leq)$   $f$  **using** *mono* **unfolding** *order-class.mono-def monotone-def* .

**next**

**show**  $P\ (\text{Sup } \{\})$  **using** *chain*[*of {}*] **by** *simp*

**next**

**show** *ccpo.admissible*  $\text{Sup } (\leq)$   $P$

**by** (*auto simp add: chain ccpo.admissible-def Complete-Partial-Order.chain-def*)

**qed** *fact*

### 1.2.2 Preservation of transitivity, asymmetry, irreflexivity by suprema

**lemma** *trans-Sup-of-chain*:

**assumes** *set-chain*  $C$  **and** *trans*:  $\bigwedge R. R \in C \implies \text{trans } R$

**shows** *trans*  $(\text{Sup } C)$

**proof** (*intro transI*)

**fix**  $x\ y\ z$

**assume**  $(x,y) \in \text{Sup } C$  **and**  $(y,z) \in \text{Sup } C$

**from**  $\langle (x,y) \in \text{Sup } C \rangle$  **obtain**  $R \in C$  **and**  $(x,y) \in R$  **by** *blast*

**from**  $\langle (y,z) \in \text{Sup } C \rangle$  **obtain**  $S \in C$  **and**  $(y,z) \in S$  **by** *blast*

**from**  $\langle R \in C \rangle$  **and**  $\langle S \in C \rangle$  **and**  $\langle \text{set-chain } C \rangle$  **have**  $R \cup S = R \vee R \cup S = S$

**by** *blast*

**with**  $\langle R \in C \rangle$  **and**  $\langle S \in C \rangle$  **have**  $R \cup S \in C$  **by** *fastforce*

**with**  $\langle (x,y) \in R \rangle$  **and**  $\langle (y,z) \in S \rangle$  **and** *trans*[*of*  $R \cup S$ ]

**have**  $(x,z) \in R \cup S$  **unfolding** *trans-def* **by** *blast*

**with**  $\langle R \cup S \in C \rangle$  **show**  $(x,z) \in \bigcup C$  **by** *blast*

**qed**

**lemma** *asym-Sup-of-chain*:

**assumes** *set-chain*  $C$  **and** *asym*:  $\bigwedge R. R \in C \implies \text{asym } R$

**shows** *asym*  $(\text{Sup } C)$

**proof** (*intro asymI notI*)

**fix**  $a\ b$

**assume**  $(a,b) \in \text{Sup } C$  **then** **obtain**  $R$  **where**  $R \in C$  **and**  $(a,b) \in R$  **by** *blast*

**assume**  $(b,a) \in \text{Sup } C$  **then** **obtain**  $S$  **where**  $S \in C$  **and**  $(b,a) \in S$  **by** *blast*

**from**  $\langle R \in C \rangle$  **and**  $\langle S \in C \rangle$  **and**  $\langle \text{set-chain } C \rangle$  **have**  $R \cup S = R \vee R \cup S = S$

**by** *blast*

**with**  $\langle R \in C \rangle$  **and**  $\langle S \in C \rangle$  **have**  $R \cup S \in C$  **by** *fastforce*

**with**  $\langle (a,b) \in R \rangle$  **and**  $\langle (b,a) \in S \rangle$  **and** *asym*[*THEN asymD*] **show** *False* **by** *blast*

**qed**

**lemma** *strict-order-lfp*:  
**assumes** *mono f* **and**  $\bigwedge R. \text{strict-order } R \implies \text{strict-order } (f R)$   
**shows** *strict-order (lfp f)*  
**proof** (*intro lfp-chain-induct[of f strict-order]*)  
**fix**  $C :: ('b \times 'b) \text{ set set}$   
**assume** *set-chain C* **and**  $\forall R \in C. \text{strict-order } R$   
**from this show** *strict-order (Sup C)*  
**using** *asym-on-iff-irrefl-on-if-trans-on[of UNIV]*  
**by** (*metis asym-Sup-of-chain trans-Sup-of-chain*)  
**qed fact+**

**lemma** *trans-lfp*:  
**assumes** *mono f* **and**  $\bigwedge R. \text{trans } R \implies \text{trans } (f R)$   
**shows** *trans (lfp f)*  
**by** (*metis lfp-chain-induct[of f trans] assms trans-Sup-of-chain*)  
**end**

### 1.3 Multiset extension

**lemma** *mulex-iff-mult*:  $\text{mulex } r M N \iff (M, N) \in \text{mult } \{(M, N) . r M N\}$   
**by** (*auto simp add: mulex-on-def restrict-to-def mult-def mulex1-def tranclp-unfold*)

**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**  $(M, N) \in \text{mult } r$   
**using** *assms one-step-implies-mult* **by** *blast*

**lemma** *multE*:  
**assumes** *trans r* **and**  $(M, N) \in \text{mult } r$   
**obtains**  $I J K$  **where**  $M = I + K$   $N = I + J$   $J \neq \{\#\}$   $\forall k \in \text{set-mset } K. \exists j \in \text{set-mset } J. (k, j) \in r$   
**using** *mult-implies-one-step[OF assms]* **by** *blast*

**lemma** *mult-on-union*:  $(M, N) \in \text{mult } r \implies (K + M, K + N) \in \text{mult } r$   
**using** *mulex-on-union[of  $\lambda x y. (x, y) \in r$  UNIV]* **by** (*auto simp: mulex-iff-mult*)

**lemma** *mult-on-union'*:  $(M, N) \in \text{mult } r \implies (M + K, N + K) \in \text{mult } r$   
**using** *mulex-on-union'[of  $\lambda x y. (x, y) \in r$  UNIV]* **by** (*auto simp: mulex-iff-mult*)

**lemma** *mult-on-add-mset*:  $(M, N) \in \text{mult } r \implies (\text{add-mset } k M, \text{add-mset } k N) \in \text{mult } r$   
**unfolding** *add-mset-add-single[of k M]* *add-mset-add-single[of k N]* **by** (*rule mult-on-union'*)

**lemma** *mult-empty[simp]*:  $(M, \{\#\}) \notin \text{mult } R$   
**by** (*metis mult-def not-less-empty trancl.cases*)

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

*r*  
**unfolding** *add-mset-add-single*[of *x M*] *add-mset-add-single*[of *y M*]  
**apply** (*rule mult-on-union*)  
**using** *mult1-singleton*[of *x y r*] **by** (*auto simp add: mult-def mult-on-union*)

**lemma** *empty-mult*[*simp*]:  $(\{\#\}, N) \in \text{mult } R \longleftrightarrow N \neq \{\#\}$   
**using** *empty-mulex-on*[of *N UNIV*  $\lambda M N. (M, N) \in R$ ] **by** (*auto simp add: mulex-iff-mult*)

**lemma** *trans-mult*: *trans* (*mult R*)  
**unfolding** *mult-def* **by** *simp*

**lemma** *strict-order-mult*:  
**assumes** *irrefl R* **and** *trans R*  
**shows** *irrefl* (*mult R*) **and** *trans* (*mult R*)  
**proof** –  
**show** *irrefl* (*mult R*) **unfolding** *irrefl-def*  
**proof** (*intro allI notI, elim multE*[*OF*  $\langle \text{trans } R \rangle$ ])  
**fix** *M I J K*  
**assume**  $M = I + J$   $M = I + K$   $J \neq \{\#\}$  **and**  $*$ :  $\forall k \in \text{set-mset } K. \exists j \in \text{set-mset } J. (k, j) \in R$   
**from**  $\langle M = I + J \rangle$  **and**  $\langle M = I + K \rangle$  **have**  $J = K$  **by** *simp*  
**have** *finite* (*set-mset J*) **by** *simp*  
**then have** *set-mset J* =  $\{\}$  **using**  $*$  **unfolding**  $\langle J = K \rangle$   
**by** (*induct rule: finite-induct*)  
*(simp, metis assms insert-absorb insert-iff insert-not-empty irrefl-def transD)*  
**then show** *False* **using**  $\langle J \neq \{\#\} \rangle$  **by** *simp*  
**qed**  
**qed** (*simp add: trans-mult*)

**lemma** *mult-of-image-mset*:  
**assumes** *trans R* **and** *trans R'*  
**and**  $\bigwedge x y. x \in \text{set-mset } N \implies y \in \text{set-mset } M \implies (x, y) \in R \implies (f x, f y) \in R'$   
**and**  $(N, M) \in \text{mult } R$   
**shows**  $(\text{image-mset } f N, \text{image-mset } f M) \in \text{mult } R'$   
**proof** (*insert assms*(4), *elim multE*[*OF* *assms*(1)])  
**fix** *I J K*  
**assume**  $N = I + K$   $M = I + J$   $J \neq \{\#\}$   $\forall k \in \text{set-mset } K. \exists j \in \text{set-mset } J. (k, j) \in R$   
**thus**  $(\text{image-mset } f N, \text{image-mset } f M) \in \text{mult } R'$  **using** *assms*(2,3)  
**by** (*intro multI*) (*auto, blast*)  
**qed**

#### 1.4 Incrementality of *mult1* and *mult*

**lemma** *mono-mult1*: *mono mult1*  
**unfolding** *mono-def mult1-def* **by** *blast*

**lemma** *mono-mult*: *mono mult*

```

unfolding mono-def mult-def
proof (intro allI impI subsetI)
  fix  $R S :: 'a \text{ rel}$  and  $x$ 
  assume  $R \subseteq S$  and  $x \in (\text{mult1 } R)^+$ 
  then show  $x \in (\text{mult1 } S)^+$ 
  using mono-mult1 [unfolded mono-def] trancl-mono [of x mult1 R mult1 S] by
auto
qed

```

## 1.5 Well-orders and well-quasi-orders

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

$wf\ p \longleftrightarrow wfp\text{-on } (\lambda a\ b. (a, b) \in p)\ UNIV$

**unfolding** *wfp-on-iff-inductive-on wf-def inductive-on-def* **by force**

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

**assumes** *well-order r*

**shows** *wqo-on*  $(\lambda a\ b. (a, b) \in r)\ UNIV$

**proof** (*intro wqo-onI almost-full-onI*)

**show** *transp*  $(\lambda a\ b. (a, b) \in r)$  **using** *assms*

**by** (*auto simp only: well-order-on-def linear-order-on-def partial-order-on-def pre-order-on-def*

*trans-def transp-def*)

**next**

**fix**  $f :: \text{nat} \Rightarrow 'a$

**show** *good*  $(\lambda a\ b. (a, b) \in r)\ f$

**using** *assms unfolding well-order-on-def wf-iff-wfp-on wfp-on-def not-ex not-all de-Morgan-conj*

**proof** (*elim conjE allE exE*)

**fix**  $x$  **assume** *linear-order r* **and**  $f\ x \notin UNIV \vee (f\ (\text{Suc } x), f\ x) \notin r - Id$

**then have**  $(f\ x, f\ (\text{Suc } x)) \in r$  **using**  $\langle \text{linear-order } r \rangle$

**by** (*force simp: linear-order-on-def Relation.total-on-def partial-order-on-def pre-order-on-def*

*refl-on-def*)

**then show** *good*  $(\lambda a\ b. (a, b) \in r)\ f$  **by** (*auto simp: good-def*)

**qed**

**qed**

## 1.6 Splitting lists into prefix, element, and suffix

**fun** *list-splits*  $:: 'a \text{ list} \Rightarrow ('a \text{ list} \times 'a \times 'a \text{ list}) \text{ list}$  **where**

*list-splits*  $[] = []$

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

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

*list-splits*  $xs = [] \longleftrightarrow xs = []$

**by** (*cases xs*) *simp-all*

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

**assumes**  $(ys, y, zs) \in \text{set } (\text{list-splits } xs)$   
**shows**  $ys @ [y] @ zs = xs$   
**using** *assms* **by** (*induct xs arbitrary: ys*) *auto*

**lemma** *elem-list-splits-length*:  
**assumes**  $(ys, y, zs) \in \text{set } (\text{list-splits } xs)$   
**shows**  $\text{length } ys < \text{length } xs$  **and**  $\text{length } zs < \text{length } xs$   
**using** *elem-list-splits-append[OF assms]* **by** *auto*

**lemma** *elem-list-splits-elem*:  
**assumes**  $(xs, y, ys) \in \text{set } (\text{list-splits } zs)$   
**shows**  $y \in \text{set } zs$   
**using** *elem-list-splits-append[OF assms]* **by** *force*

**lemma** *list-splits-append*:  
 $\text{list-splits } (xs @ ys) = \text{map } (\lambda(xs', x', ys'). (xs', x', ys' @ ys)) (\text{list-splits } xs) @$   
 $\text{map } (\lambda(xs', x', ys'). (xs @ xs', x', ys')) (\text{list-splits } ys)$   
**by** (*induct xs*) *auto*

**lemma** *list-splits-rev*:  
 $\text{list-splits } (\text{rev } xs) = \text{map } (\lambda(xs, x, ys). (\text{rev } ys, x, \text{rev } xs)) (\text{rev } (\text{list-splits } xs))$   
**by** (*induct xs*) (*auto simp add: list-splits-append comp-def prod.case-distrib rev-map*)

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

**end**

## 2 Decreasing Diagrams

**theory** *Decreasing-Diagrams-II*

**imports**

*Decreasing-Diagrams-II-Aux*

*HOL-Cardinals.Wellorder-Extension*

*Abstract-Rewriting.Abstract-Rewriting*

**begin**

### 2.1 Greek accents

**datatype** *accent* = *Acute* | *Grave* | *Macron*

**lemma** *UNIV-accent*:  $UNIV = \{ \text{Acute}, \text{Grave}, \text{Macron} \}$   
**using** *accent.nchotomy* **by** *blast*

**lemma** *finite-accent*:  $\text{finite } (UNIV :: \text{accent set})$   
**by** (*simp add: UNIV-accent*)

**type-synonym** 'a letter = accent × 'a

**definition** letter-less :: ('a × 'a) set ⇒ ('a letter × 'a letter) set **where**  
[simp]: letter-less R = {(a,b). (snd a, snd b) ∈ R}

**lemma** mono-letter-less: mono letter-less  
**by** (auto simp add: mono-def)

## 2.2 Comparing Greek strings

**type-synonym** 'a greek = 'a letter list

**definition** adj-msog :: 'a greek ⇒ 'a greek ⇒ ('a letter × 'a greek) ⇒ ('a letter × 'a greek)

**where**

adj-msog xs zs l ≡  
case l of (y,ys) ⇒ (y, case fst y of Acute ⇒ ys @ zs | Grave ⇒ xs @ ys | Macron  
⇒ ys)

**definition** ms-of-greek :: 'a greek ⇒ ('a letter × 'a greek) multiset **where**

ms-of-greek as = mset  
(map (λ(xs, y, zs) ⇒ adj-msog xs zs (y, [])) (list-splits as))

**lemma** adj-msog-adj-msog[simp]:

adj-msog xs zs (adj-msog xs' zs' y) = adj-msog (xs @ xs') (zs' @ zs) y  
**by** (auto simp: adj-msog-def split: accent.splits prod.splits)

**lemma** compose-adj-msog[simp]: adj-msog xs zs ∘ adj-msog xs' zs' = adj-msog (xs  
@ xs') (zs' @ zs)

**by** (simp add: comp-def)

**lemma** adj-msog-single:

adj-msog xs zs (x,[]) = (x, (case fst x of Grave ⇒ xs | Acute ⇒ zs | Macron ⇒  
[]))

**by** (simp add: adj-msog-def split: accent.splits)

**lemma** ms-of-greek-elem:

**assumes** (x,xs) ∈ set-mset (ms-of-greek ys)

**shows** x ∈ set ys

**using** assms **by** (auto dest: elem-list-splits-elem simp: adj-msog-def ms-of-greek-def)

**lemma** ms-of-greek-shorter:

**assumes** (x, t) ∈# ms-of-greek s

**shows** length s > length t

**using** assms[unfolded ms-of-greek-def in-multiset-in-set]

**by** (auto simp: elem-list-splits-length adj-msog-def split: accent.splits)

**lemma** msog-append: ms-of-greek (xs @ ys) = image-mset (adj-msog [] ys) (ms-of-greek  
xs) +



*image-mset (adj-msog xs []) (ms-of-greek ys)*  
**by** (*auto simp: ms-of-greek-def list-splits-append multiset.map-comp comp-def prod.case-distrib*)

**definition** *nest* :: ('a × 'a) set ⇒ ('a greek × 'a greek) set ⇒ ('a greek × 'a greek) set **where**  
*[simp]: nest r s = {(a,b). (ms-of-greek a, ms-of-greek b) ∈ mult (letter-less r <\*>lex\* s)}*

**lemma** *mono-nest: mono (nest r)*  
**unfolding** *mono-def*  
**proof** (*intro allI impI subsetI*)  
**fix** *R S x*  
**assume** *1: R ⊆ S and 2: x ∈ nest r R*  
**from** *1* **have** *mult (letter-less r <\*>lex\* R) ⊆ mult (letter-less r <\*>lex\* S)*  
**using** *mono-mult mono-lex2[of letter-less r]* **unfolding** *mono-def* **by** *blast*  
**with** *2* **show** *x ∈ nest r S* **by** *auto*  
**qed**

**lemma** *nest-mono[mono-set]: x ⊆ y ⇒ (a,b) ∈ nest r x ⇒ (a,b) ∈ nest r y*  
**using** *mono-nest[unfolding mono-def, rule-format, of x y r]* **by** *blast*

**definition** *greek-less* :: ('a × 'a) set ⇒ ('a greek × 'a greek) set **where**  
*greek-less r = lfp (nest r)*

**lemma** *greek-less-unfold:*  
*greek-less r = nest r (greek-less r)*  
**using** *mono-nest[of r] lfp-unfold[of nest r]* **by** (*simp add: greek-less-def*)

## 2.3 Preservation of strict partial orders

**lemma** *strict-order-letter-less:*  
**assumes** *strict-order r*  
**shows** *strict-order (letter-less r)*  
**using** *assms unfolding irrefl-def trans-def letter-less-def* **by** *fast*

**lemma** *strict-order-nest:*  
**assumes** *r: strict-order r and R: strict-order R*  
**shows** *strict-order (nest r R)*  
**proof** –  
**have** *strict-order (mult (letter-less r <\*>lex\* R))*  
**using** *strict-order-letter-less[of r] irrefl-lex-prod[of letter-less r R]*  
*trans-lex-prod[of letter-less r R] strict-order-mult[of letter-less r <\*>lex\* R]*  
*assms*  
**by** *fast*  
**from** *this* **show** *strict-order (nest r R)* **unfolding** *nest-def trans-def irrefl-def*  
**by** *fast*  
**qed**

**lemma** *strict-order-greek-less:*

**assumes** *strict-order*  $r$   
**shows** *strict-order* (*greek-less*  $r$ )  
**by** (*simp add: greek-less-def strict-order-lfp[OF mono-nest strict-order-nest[OF assms]]*)

**lemma** *trans-letter-less*:  
**assumes** *trans*  $r$   
**shows** *trans* (*letter-less*  $r$ )  
**using** *assms unfolding trans-def letter-less-def by fast*

**lemma** *trans-order-nest: trans* (*nest*  $r$   $R$ )  
**using** *trans-mult unfolding nest-def trans-def by fast*

**lemma** *trans-greek-less[simp]: trans* (*greek-less*  $r$ )  
**by** (*subst greek-less-unfold*) (*rule trans-order-nest*)

**lemma** *mono-greek-less: mono greek-less*  
**unfolding** *greek-less-def mono-def*  
**proof** (*intro allI impI lfp-mono*)  
**fix**  $r\ s :: ('a \times 'a)$  *set* **and**  $R :: ('a\ \text{greek} \times 'a\ \text{greek})$  *set*  
**assume**  $r \subseteq s$   
**then have** *letter-less*  $r <*\text{lex}*> R \subseteq \text{letter-less } s <*\text{lex}*> R$   
**using** *mono-letter-less mono-lex1 unfolding mono-def by metis*  
**then show** *nest*  $r$   $R \subseteq \text{nest } s$   $R$  **using** *mono-mult unfolding nest-def mono-def*  
**by** *blast*  
**qed**

## 2.4 Involution

**definition** *inv-letter*  $:: 'a\ \text{letter} \Rightarrow 'a\ \text{letter}$  **where**  
*inv-letter*  $l \equiv$   
*case*  $l$  of  $(a, x) \Rightarrow (\text{case } a \text{ of Grave} \Rightarrow \text{Acute} \mid \text{Acute} \Rightarrow \text{Grave} \mid \text{Macron} \Rightarrow \text{Macron}, x)$

**lemma** *inv-letter-pair[simp]*:  
*inv-letter*  $(a, x) = (\text{case } a \text{ of Grave} \Rightarrow \text{Acute} \mid \text{Acute} \Rightarrow \text{Grave} \mid \text{Macron} \Rightarrow \text{Macron}, x)$   
**by** (*simp add: inv-letter-def*)

**lemma** *snd-inv-letter[simp]*:  
*snd* (*inv-letter*  $x$ ) = *snd*  $x$   
**by** (*simp add: inv-letter-def split: prod.splits*)

**lemma** *inv-letter-invol[simp]*:  
*inv-letter* (*inv-letter*  $x$ ) =  $x$   
**by** (*simp add: inv-letter-def split: prod.splits accent.splits*)

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

**using** *assms* **by** *simp*

**definition** *inv-greek* :: 'a greek  $\Rightarrow$  'a greek **where**  
*inv-greek* *s* = *rev* (*map inv-letter* *s*)

**lemma** *inv-greek-invol*[*simp*]:  
*inv-greek* (*inv-greek* *s*) = *s*  
**by** (*simp* *add: inv-greek-def rev-map comp-def*)

**lemma** *inv-greek-append*:  
*inv-greek* (*s* @ *t*) = *inv-greek* *t* @ *inv-greek* *s*  
**by** (*simp* *add: inv-greek-def*)

**definition** *inv-msog* :: ('a letter  $\times$  'a greek) multiset  $\Rightarrow$  ('a letter  $\times$  'a greek) multiset **where**  
*inv-msog* *M* = *image-mset* ( $\lambda(x, t). (inv-letter\ x, inv-greek\ t)$ ) *M*

**lemma** *inv-msog-invol*[*simp*]:  
*inv-msog* (*inv-msog* *M*) = *M*  
**by** (*simp* *add: inv-msog-def multiset.map-comp comp-def prod.case-distrib*)

**lemma** *ms-of-greek-inv-greek*:  
*ms-of-greek* (*inv-greek* *M*) = *inv-msog* (*ms-of-greek* *M*)  
**unfolding** *inv-msog-def inv-greek-def ms-of-greek-def list-splits-rev list-splits-map mset-map*  
*multiset.map-comp mset-rev inv-letter-def adj-msog-def*  
**by** (*rule cong[OF cong[OF refl[of image-mset]] refl]*) (*auto split: accent.splits*)

**lemma** *inv-greek-mono*:  
**assumes** *trans* *r* **and** (*s*, *t*)  $\in$  *greek-less* *r*  
**shows** (*inv-greek* *s*, *inv-greek* *t*)  $\in$  *greek-less* *r*  
**using** *assms*(2)  
**proof** (*induct* *length* *s* + *length* *t* *arbitrary: s t* *rule: less-induct*)  
**note** \* = *trans-lex-prod*[*OF trans-letter-less*[*OF*  $\langle$ *trans* *r* $\rangle$ ] *trans-greek-less*[*of* *r*]]  
**case** (*less* *s* *t*)  
**have** (*inv-msog* (*ms-of-greek* *s*), *inv-msog* (*ms-of-greek* *t*))  $\in$  *mult* (*letter-less* *r* <math>lex</math> *greek-less* *r*)  
**unfolding** *inv-msog-def*  
**proof** (*induct* *rule: mult-of-image-mset*[*OF* \* \*])  
**case** (*1* *x* *y*) **thus** ?*case*  
**by** (*auto* *intro: less*(1) *split: prod.splits dest!: ms-of-greek-shorter*)  
**next**  
**case** 2 **thus** ?*case* **using** *less*(2) **by** (*subst*(*asm*) *greek-less-unfold*) *simp*  
**qed**  
**thus** ?*case* **by** (*subst* *greek-less-unfold*) (*auto* *simp: ms-of-greek-inv-greek*)  
**qed**

## 2.5 Monotonicity of *greek-less r*

**lemma** *greek-less-empty[simp]*:

$(a, []) \in \text{greek-less } r \longleftrightarrow \text{False}$

**by** (*subst greek-less-unfold*) (*auto simp: ms-of-greek-def*)

**lemma** *greek-less-nonempty*:

**assumes**  $b \neq []$

**shows**  $(a, b) \in \text{greek-less } r \longleftrightarrow (a, b) \in \text{nest } r \text{ (greek-less } r)$

**by** (*subst greek-less-unfold*) *simp*

**lemma** *greek-less-empty[simp]*:

$([], b) \in \text{greek-less } r \longleftrightarrow b \neq []$

**proof**

**assume**  $([], b) \in \text{greek-less } r$

**then show**  $b \neq []$  **using** *greek-less-empty* **by fast**

**next**

**assume**  $b \neq []$

**then show**  $([], b) \in \text{greek-less } r$

**unfolding** *greek-less-nonempty*[*OF*  $\langle b \neq [] \rangle$ ] **by** (*simp add: ms-of-greek-def*)

**qed**

**lemma** *greek-less-singleton*:

$(a, b) \in \text{letter-less } r \implies ([a], [b]) \in \text{greek-less } r$

**by** (*subst greek-less-unfold*) (*auto split: accent.splits simp: adj-msog-def ms-of-greek-def*)

**lemma** *ms-of-greek-cons*:

$\text{ms-of-greek } (x \# s) = \{\# \text{ adj-msog } [] \text{ } s (x, []) \# \} + \text{image-mset } (\text{adj-msog } [x] [])$   
*(ms-of-greek s)*

**using** *msog-append*[*of*  $[x] \ s$ ]

**by** (*auto simp add: adj-msog-def ms-of-greek-def accent.splits*)

**lemma** *greek-less-cons-mono*:

**assumes** *trans r*

**shows**  $(s, t) \in \text{greek-less } r \implies (x \# s, x \# t) \in \text{greek-less } r$

**proof** (*induct length s + length t arbitrary: s t rule: less-induct*)

**note**  $*$  = *trans-lex-prod*[*OF* *trans-letter-less*[*OF*  $\langle \text{trans } r \rangle$ ] *trans-greek-less*[*of*  $r$ ]]

**case** (*less s t*)

{

**fix**  $M$  **have**  $(M + \text{image-mset } (\text{adj-msog } [x] [])) \text{ (ms-of-greek } s),$

$M + \text{image-mset } (\text{adj-msog } [x] []) \text{ (ms-of-greek } t) \in \text{mult } (\text{letter-less } r \langle * \text{lex} * \rangle$

*greek-less r*)

**proof** (*rule mult-on-union, induct rule: mult-of-image-mset*[*OF*  $* \ *$ ])

**case** ( $1 \ x \ y$ ) **thus** *?case* **unfolding** *adj-msog-def*

**by** (*auto intro: less(1) split: prod.splits accent.splits dest!: ms-of-greek-shorter*)

**next**

**case**  $2$  **thus** *?case* **using** *less(2)* **by** (*subst(asm) greek-less-unfold*) *simp*

**qed**

}

**moreover** {

**fix**  $N$  **have**  $(\{\# \text{ adj-msog } [] s (x, []) \# \} + N, \{\# \text{ adj-msog } [] t (x, []) \# \} + N) \in$   
 $(\text{mult } (\text{letter-less } r \langle *lex* \rangle \text{ greek-less } r))^=$   
**by**  $(\text{auto simp: adj-msog-def less split: accent.splits})$  }  
**ultimately show**  $?case$  **using**  $\text{transD}[OF \text{trans-mult}]$   
**by**  $(\text{subst greek-less-unfold}) (\text{fastforce simp: ms-of-greek-cons})$   
**qed**

**lemma**  $\text{greek-less-app-mono2}$ :  
**assumes**  $\text{trans } r$  **and**  $(s, t) \in \text{greek-less } r$   
**shows**  $(p @ s, p @ t) \in \text{greek-less } r$   
**using**  $\text{assms}$  **by**  $(\text{induct } p) (\text{auto simp add: greek-less-cons-mono})$

**lemma**  $\text{greek-less-app-mono1}$ :  
**assumes**  $\text{trans } r$  **and**  $(s, t) \in \text{greek-less } r$   
**shows**  $(s @ p, t @ p) \in \text{greek-less } r$   
**using**  $\text{inv-greek-mono}[of r \text{ inv-greek } p @ \text{inv-greek } s \text{ inv-greek } p @ \text{inv-greek } t]$   
**by**  $(\text{simp add: assms inv-greek-append inv-greek-mono greek-less-app-mono2})$

## 2.6 Well-founded-ness of $\text{greek-less } r$

**lemma**  $\text{greek-embed}$ :  
**assumes**  $\text{trans } r$   
**shows**  $\text{list-emb } (\lambda a b. (a, b): \text{reflcl } (\text{letter-less } r)) a b \implies (a, b) \in \text{reflcl } (\text{greek-less } r)$   
**proof**  $(\text{induct rule: list-emb.induct})$   
**case**  $(\text{list-emb-Cons } a b y)$  **thus**  $?case$   
**using**  $\text{trans-greek-less}[unfolding \text{trans-def}] \langle \text{trans } r \rangle$   
 $\text{greek-less-app-mono1}[of r [] [y] a]$   $\text{greek-less-app-mono2}[of r a b [y]]$  **by**  $\text{auto}$   
**next**  
**case**  $(\text{list-emb-Cons2 } x y a b)$  **thus**  $?case$   
**using**  $\text{trans-greek-less}[unfolding \text{trans-def}] \langle \text{trans } r \rangle$   $\text{greek-less-singleton}[of x y r]$   
 $\text{greek-less-app-mono1}[of r [x] [y] a]$   $\text{greek-less-app-mono2}[of r a b [y]]$  **by**  $\text{auto}$   
**qed**  $\text{simp}$

**lemma**  $\text{wqo-letter-less}$ :  
**assumes**  $t: \text{trans } r$  **and**  $w: \text{wqo-on } (\lambda a b. (a, b) \in r^=)$   $UNIV$   
**shows**  $\text{wqo-on } (\lambda a b. (a, b) \in (\text{letter-less } r)^=)$   $UNIV$   
**proof**  $(\text{rule wqo-on-hom}[of id - - \text{prod-le } (=) (\lambda a b. (a, b) \in r^=), \text{unfolding image-id id-apply}])$   
**show**  $\text{wqo-on } (\text{prod-le } ((=) :: \text{accent} \implies \text{accent} \implies \text{bool}) (\lambda a b. (a, b) \in r^=))$   $UNIV$   
**by**  $(\text{rule dickson}[OF \text{finite-eq-wqo-on}[OF \text{finite-accent}] w, \text{unfolding UNIV-Times-UNIV}])$   
**qed**  $(\text{insert } t, \text{auto simp: transp-on-def trans-def prod-le-def})$

**lemma**  $\text{wf-greek-less}$ :  
**assumes**  $\text{wf } r$  **and**  $\text{trans } r$   
**shows**  $\text{wf } (\text{greek-less } r)$   
**proof** –  
**obtain**  $q$  **where**  $r \subseteq q$  **and**  $\text{well-order } q$  **by**  $(\text{metis total-well-order-extension } \langle \text{wf } r \rangle)$

```

define  $q'$  where  $q' = q - Id$ 
from  $\langle well\text{-}order\ q \rangle$  have  $reflcl\ q' = q$ 
by (auto simp add: well-order-on-def linear-order-on-def partial-order-on-def pre-
order-on-def
    refl-on-def q'-def)
from  $\langle well\text{-}order\ q \rangle$  have  $trans\ q'$  and  $irrefl\ q'$ 
unfolding well-order-on-def linear-order-on-def partial-order-on-def preorder-on-def
antisym-def
    trans-def irrefl-def q'-def by blast+
from  $\langle r \subseteq q \rangle \langle wf\ r \rangle$  have  $r \subseteq q'$  by (auto simp add: q'-def)
have  $wqo\text{-}on\ (\lambda a\ b. (a, b) \in (greek\text{-}less\ q')^=)$  UNIV
proof (intro wqo-on-hom[of id UNIV (\lambda a b. (a, b) \in (greek-less q')^=)
    list-emb (\lambda a b. (a, b) \in (letter-less q')^=), unfolded surj-id)
    show  $transp\ (\lambda a\ b. (a, b) \in (greek\text{-}less\ q')^=)$ 
    using trans-greek-less[of q'] unfolding trans-def transp-on-def by blast
next
    show  $\forall x \in UNIV. \forall y \in UNIV. list\text{-}emb\ (\lambda a\ b. (a, b) \in (letter\text{-}less\ q')^=)\ x\ y \longrightarrow$ 
     $(id\ x, id\ y) \in (greek\text{-}less\ q')^=$ 
    using greek-embed[OF \langle trans q' \rangle] by auto
next
    show  $wqo\text{-}on\ (list\text{-}emb\ (\lambda a\ b. (a, b) \in (letter\text{-}less\ q')^=))\ UNIV$ 
    using higman[OF wqo-letter-less[OF \langle trans q' \rangle]] \langle well-order\ q \rangle \langle reflcl\ q' = q \rangle
    by (auto simp: well-order-implies-wqo)
qed
with  $wqo\text{-}on\text{-}imp\text{-}wfp\text{-}on[OF\ this]\ strict\text{-}order\text{-}strict[OF\ strict\text{-}order\text{-}greek\text{-}less]$ 
     $\langle irrefl\ q' \rangle \langle trans\ q' \rangle$ 
have  $wfp\text{-}on\ (\lambda a\ b. (a, b) \in greek\text{-}less\ q')\ UNIV$  by force
then show ?thesis
    using mono-greek-less \langle r \subseteq q' \rangle wf-subset unfolding wf-iff-wfp-on[symmetric]
mono-def by metis
qed

```

## 2.7 Basic Comparisons

**lemma** *pairwise-imp-mult:*

```

assumes  $N \neq \{\#\}$  and  $\forall x \in set\text{-}mset\ M. \exists y \in set\text{-}mset\ N. (x, y) \in r$ 
shows  $(M, N) \in mult\ r$ 
using assms one-step-implies-mult[of - - \{\#\}] by auto

```

**lemma** *singleton-greek-less:*

```

assumes as: snd ' set as  $\subseteq$  under r b
shows  $(as, [(a, b)]) \in greek\text{-}less\ r$ 

```

**proof** –

```

{
  fix  $e$  assume  $e \in set\text{-}mset\ (ms\text{-}of\text{-}greek\ as)$ 
  with  $as\ ms\text{-}of\text{-}greek\text{-}elem[of\ -\ -\ as]$ 
  have  $(e, ((a, b), [])) \in letter\text{-}less\ r <*\text{lex*}> greek\text{-}less\ r$ 
  by (cases e) (fastforce simp: adj-msog-def under-def)
}

```

**moreover have** *ms-of-greek* [(a,b)] = {# ((a,b),[]) #}  
**by** (*auto simp: ms-of-greek-def adj-msog-def split: accent.splits*)  
**ultimately show** *?thesis*  
**by** (*subst greek-less-unfold*) (*auto intro!: pairwise-imp-mult*)  
**qed**

**lemma** *peak-greek-less*:

**assumes** *as*: *snd* ' set  $as \subseteq$  under  $r$   $a$  **and**  $b'$ :  $b' \in \{[(Grave,b)],[]\}$   
**and** *cs*: *snd* ' set  $cs \subseteq$  under  $r$   $a \cup$  under  $r$   $b$  **and**  $a'$ :  $a' \in \{[(Acute,a)],[]\}$   
**and** *bs*: *snd* ' set  $bs \subseteq$  under  $r$   $b$   
**shows** ( $as @ b' @ cs @ a' @ bs$ ,  $[(Acute,a),(Grave,b)] \in$  *greek-less*  $r$ )  
**proof** –  
**let**  $?A = (Acute,a)$  **and**  $?B = (Grave,b)$   
**have** (*ms-of-greek* ( $as @ b' @ cs @ a' @ bs$ ), *ms-of-greek* [ $?A,?B$ ])  $\in$  *mult*  
(*letter-less*  $r$  *<\*lex\*>* *greek-less*  $r$ )  
**proof** (*intro pairwise-imp-mult*)

{  
**fix**  $e$  **assume**  $e \in$  *set-mset* (*ms-of-greek*  $as$ )  
**with**  $as$  *ms-of-greek-elem*[*of* - -  $as$ ]  
**have** (*adj-msog* [] ( $b' @ cs @ a' @ bs$ )  $e$ , ( $?A,[?B]$ ))  $\in$  *letter-less*  $r$  *<\*lex\*>*  
*greek-less*  $r$   
**by** (*cases*  $e$ ) (*fastforce simp: adj-msog-def under-def*)  
}  
**moreover** {  
**fix**  $e$  **assume**  $e \in$  *set-mset* (*ms-of-greek*  $b'$ )  
**with**  $b'$  *singleton-greek-less*[*OF*  $as$ ] *ms-of-greek-elem*[*of* - -  $b'$ ]  
**have** (*adj-msog*  $as$  ( $cs @ a' @ bs$ )  $e$ , ( $?B,[?A]$ ))  $\in$  *letter-less*  $r$  *<\*lex\*>*  
*greek-less*  $r$   
**by** (*cases*  $e$ ) (*fastforce simp: adj-msog-def ms-of-greek-def*)  
}  
**moreover** {  
**fix**  $e$  **assume**  $e \in$  *set-mset* (*ms-of-greek*  $cs$ )  
**with**  $cs$  *ms-of-greek-elem*[*of* - -  $cs$ ]  
**have** (*adj-msog* ( $as @ b'$ ) ( $a' @ bs$ )  $e$ , ( $?A,[?B]$ ))  $\in$  *letter-less*  $r$  *<\*lex\*>*  
*greek-less*  $r \vee$   
(*adj-msog* ( $as @ b'$ ) ( $a' @ bs$ )  $e$ , ( $?B,[?A]$ ))  $\in$  *letter-less*  $r$  *<\*lex\*>*  
*greek-less*  $r$   
**by** (*cases*  $e$ ) (*fastforce simp: adj-msog-def under-def*)  
}  
**moreover** {  
**fix**  $e$  **assume**  $e \in$  *set-mset* (*ms-of-greek*  $a'$ )  
**with**  $a'$  *singleton-greek-less*[*OF*  $bs$ ] *ms-of-greek-elem*[*of* - -  $a'$ ]  
**have** (*adj-msog* ( $as @ b' @ cs$ )  $bs$   $e$ , ( $?A,[?B]$ ))  $\in$  *letter-less*  $r$  *<\*lex\*>*  
*greek-less*  $r$   
**by** (*cases*  $e$ ) (*fastforce simp: adj-msog-def ms-of-greek-def*)  
}  
**moreover** {  
**fix**  $e$  **assume**  $e \in$  *set-mset* (*ms-of-greek*  $bs$ )

**with**  $bs$  *ms-of-greek-elim*[of - -  $bs$ ]  
**have** (*adj-msog* ( $as @ b' @ cs @ a'$ ) []  $e$ , ( $?B, [?A]$ ))  $\in$  *letter-less r*  $\langle *lex* \rangle$   
*greek-less r*  
**by** (*cases e*) (*fastforce simp: adj-msog-def under-def*)  
**}**  
**moreover have** *ms-of-greek* [ $?A, ?B$ ] = {# ( $?B, [?A]$ ), ( $?A, [?B]$ ) #}  
**by** (*simp add: adj-msog-def ms-of-greek-def*)  
**ultimately show**  $\forall x \in \text{set-mset}$  (*ms-of-greek* ( $as @ b' @ cs @ a' @ bs$ )).  
 $\exists y \in \text{set-mset}$  (*ms-of-greek* [ $?A, ?B$ ]). ( $x, y$ )  $\in$  *letter-less r*  $\langle *lex* \rangle$  *greek-less r*  
**by** (*auto simp: msog-append*) *blast*  
**qed** (*auto simp: ms-of-greek-def*)  
**then show** *?thesis* **by** (*subst greek-less-unfold*) *auto*  
**qed**

**lemma** *rciff-greek-less1*:

**assumes** *trans r*  
**and**  $as$ : *snd* ' *set*  $as \subseteq \text{under } r \ a \cap \text{under } r \ b$  **and**  $b'$ :  $b' \in \{[(\text{Grave}, b)], []\}$   
**and**  $cs$ : *snd* ' *set*  $cs \subseteq \text{under } r \ b$  **and**  $a'$ :  $a' = [(\text{Macron}, a)]$   
**and**  $bs$ : *snd* ' *set*  $bs \subseteq \text{under } r \ b$   
**shows** ( $as @ b' @ cs @ a' @ bs$ ,  $[(\text{Macron}, a), (\text{Grave}, b)]$ )  $\in$  *greek-less r*  
**proof** –  
**let**  $?A = (\text{Macron}, a)$  **and**  $?B = (\text{Grave}, b)$   
**have** \*: *ms-of-greek* [ $?A, ?B$ ] = {# ( $?B, [?A]$ ), ( $?A, []$ ) #} *ms-of-greek* [ $?A$ ] = {# ( $?A, []$ ) #}  
**by** (*simp-all add: ms-of-greek-def adj-msog-def*)  
**then have** \*\*: *ms-of-greek*  $[(\text{Macron}, a), (\text{Grave}, b)] - \{#((\text{Macron}, a), [])\}$   
 $\neq \{#\}$   
**by** (*auto*)  
  
**{**  
**fix**  $e$  **assume**  $e \in \text{set-mset}$  (*ms-of-greek as*)  
**with**  $as$  *ms-of-greek-elim*[of - -  $as$ ]  
**have** (*adj-msog* [] ( $b' @ cs @ a' @ bs$ )  $e$ , ( $?B, [?A]$ ))  $\in$  *letter-less r*  $\langle *lex* \rangle$   
*greek-less r*  
**by** (*cases e*) (*force simp: adj-msog-def under-def*)  
**}**  
**moreover {**  
**fix**  $e$  **assume**  $e \in \text{set-mset}$  (*ms-of-greek b'*)  
**with**  $b'$  *singleton-greek-less as ms-of-greek-elim*[of - -  $b'$ ]  
**have** (*adj-msog as* ( $cs @ a' @ bs$ )  $e$ , ( $?B, [?A]$ ))  $\in$  *letter-less r*  $\langle *lex* \rangle$  *greek-less*  
 $r$   
**by** (*cases e*) (*fastforce simp: adj-msog-def ms-of-greek-def*)  
**}**  
**moreover {**  
**fix**  $e$  **assume**  $e \in \text{set-mset}$  (*ms-of-greek cs*)  
**with**  $cs$  *ms-of-greek-elim*[of - -  $cs$ ]  
**have** (*adj-msog as* ( $b'$ ) ( $a' @ bs$ )  $e$ , ( $?B, [?A]$ ))  $\in$  *letter-less r*  $\langle *lex* \rangle$   
*greek-less r*  
**by** (*cases e*) (*fastforce simp: adj-msog-def under-def*)  
**}**



```

moreover {
  fix  $e$  assume  $e \in \text{set-mset } (ms\text{-of-greek } bs)$ 
  with  $bs$   $ms\text{-of-greek-elem}[of - - bs]$ 
  have  $(adj\text{-msog } (as @ b' @ cs @ a') \sqcup e, (?B, [?A])) \in \text{letter-less } r <*\text{lex}*>$ 
  greek-less r
  by  $(cases\ e)$   $(fastforce\ simp: adj\text{-msog-def under-def})$ 
}
moreover have  $ms\text{-of-greek } [?A, ?B] = \{\#( ?B, [?A]), (?A, []) \# \}$ 
by  $(simp\ add: adj\text{-msog-def ms-of-greek-def})$ 
ultimately have  $\forall x \in \text{set-mset } (ms\text{-of-greek } (as @ b' @ cs @ a' @ bs) - \{\#(?A, []) \# \})$ .
 $\exists y \in \text{set-mset } (ms\text{-of-greek } [?A, ?B] - \{\#(?A, []) \# \})$ .  $(x, y) \in \text{letter-less } r <*\text{lex}*>$ 
greek-less r
unfolding  $msog\text{-append}$  by  $(auto\ simp: a'\ msog\text{-append ac-simps} * adj\text{-msog-single})$ 
from  $one\text{-step-implies-mult}[OF ** this, of \{\#(?A, []) \# \}]$ 
have  $(ms\text{-of-greek } (as @ b' @ cs @ a' @ bs), ms\text{-of-greek } [?A, ?B]) \in \text{mult}$ 
(letter-less r <*\text{lex}*> greek-less r)
unfolding  $a'\ msog\text{-append}$  by  $(auto\ simp: a'\ ac-simps * adj\text{-msog-single})$ 
then show  $?thesis$ 
by  $(subst\ greek\text{-less-unfold})\ auto$ 
qed

```

**lemma** *rcliff-greek-less2*:

```

assumes  $trans\ r$ 
and  $as: snd\ 'set\ as \subseteq \text{under } r\ a$  and  $b': b' \in \{[(Grave, b)], []\}$ 
and  $cs: snd\ 'set\ cs \subseteq \text{under } r\ a \cup \text{under } r\ b$ 
shows  $(as @ b' @ cs, [(Macron, a), (Grave, b)]) \in \text{greek-less } r$ 
proof -
  let  $?A = (Macron, a)$  and  $?B = (Grave, b)$ 
  have  $(ms\text{-of-greek } (as @ b' @ cs), ms\text{-of-greek } [?A, ?B]) \in \text{mult } (\text{letter-less } r$ 
   $<*\text{lex}*> \text{greek-less } r)$ 
  proof  $(intro\ pairwise\text{-imp-mult})$ 

```

```

{
  fix  $e$  assume  $e \in \text{set-mset } (ms\text{-of-greek } as)$ 
  with  $as$   $ms\text{-of-greek-elem}[of - - as]$ 
  have  $(adj\text{-msog } \sqcup (b' @ cs)\ e, (?A, [])) \in \text{letter-less } r <*\text{lex}*>$ 
  greek-less r
  by  $(cases\ e)$   $(fastforce\ simp: adj\text{-msog-def under-def})$ 
}

```

```

moreover {
  fix  $e$  assume  $e \in \text{set-mset } (ms\text{-of-greek } b')$ 
  with  $b'$   $singleton\text{-greek-less}[OF\ as]\ ms\text{-of-greek-elem}[of - - b']$ 
  have  $(adj\text{-msog } as\ (cs)\ e, (?B, [?A])) \in \text{letter-less } r <*\text{lex}*>$ 
  greek-less r
  by  $(cases\ e)$   $(fastforce\ simp: adj\text{-msog-def ms-of-greek-def})$ 
}

```

```

moreover {
  fix  $e$  assume  $e \in \text{set-mset } (ms\text{-of-greek } cs)$ 
  with  $cs$   $ms\text{-of-greek-elem}[of - - cs]$ 
  have  $(adj\text{-msog } (as @ b') \sqcup e, (?A, [])) \in \text{letter-less } r <*\text{lex}*>$ 
  greek-less r  $\vee$ 
 $(adj\text{-msog } (as @ b') \sqcup e, (?B, [?A])) \in \text{letter-less } r <*\text{lex}*>$ 
  greek-less r
}

```

```

    by (cases e) (fastforce simp: adj-msog-def under-def)
  }
  moreover have *: ms-of-greek [ $?A, ?B$ ] = {# ( $?B, [?A]$ ), ( $?A, []$ ) #}
  by (simp add: adj-msog-def ms-of-greek-def)
  ultimately show  $\forall x \in \text{set-mset} \ (ms\text{-of-greek} \ (as \ @ \ b' \ @ \ cs)).$ 
     $\exists y \in \text{set-mset} \ (ms\text{-of-greek} \ [?A, ?B]). \ (x, y) \in \text{letter-less } r \ < *lex* > \text{ greek-less } r$ 
  by (auto simp: msog-append adj-msog-single ac-simps *) blast
  qed (auto simp: ms-of-greek-def)
  then show ?thesis by (subst greek-less-unfold) auto
qed

```

**lemma** *snd-inv-greek [simp]:*  $\text{snd} \ ' \ \text{set} \ (\text{inv-greek} \ as) = \text{snd} \ ' \ \text{set} \ as$   
**by** (*force simp: inv-greek-def*)

**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'  $\in$  {[Acute,a], []}
  and bs: snd ' set bs  $\subseteq$  under r a  $\cap$  under r b
  shows  $(as \ @ \ b' \ @ \ cs \ @ \ a' \ @ \ bs, [(Acute,a), (Macron,b)]) \in \text{greek-less } r$ 
proof –
  have *: inv-greek [(Acute,a), (Macron,b)] = [(Macron,b), (Grave,a)] by (simp add: inv-greek-def)
  have (inv-greek (inv-greek (as @ b' @ cs @ a' @ bs)),
    inv-greek (inv-greek ([Acute,a], [Macron,b])))  $\in \text{greek-less } r$ 
  apply (rule inv-greek-mono[OF <trans r>])
  apply (unfold inv-greek-append append-assoc *)
  apply (insert assms)
  apply (rule rcliff-greek-less1, auto simp: inv-greek-def)
  done
  then show ?thesis by simp
qed

```

**lemma** *lcliff-greek-less2:*

```

  assumes trans r
  and cs: snd ' set cs  $\subseteq$  under r a  $\cup$  under r b and a': a'  $\in$  {[Acute,a], []}
  and bs: snd ' set bs  $\subseteq$  under r b
  shows  $(cs \ @ \ a' \ @ \ bs, [(Acute,a), (Macron,b)]) \in \text{greek-less } r$ 
proof –
  have *: inv-greek [(Acute,a), (Macron,b)] = [(Macron,b), (Grave,a)] by (simp add: inv-greek-def)
  have (inv-greek (inv-greek (cs @ a' @ bs)),
    inv-greek (inv-greek ([Acute,a], [Macron,b])))  $\in \text{greek-less } r$ 
  apply (rule inv-greek-mono[OF <trans r>])
  apply (unfold inv-greek-append append-assoc *)
  apply (insert assms)
  apply (rule rcliff-greek-less2, auto simp: inv-greek-def)
  done
  then show ?thesis by simp

```

qed

## 2.8 Labeled abstract rewriting

**context**

**fixes**  $L R E :: 'b \Rightarrow 'a \text{ rel}$

**begin**

**definition**  $lstep :: 'b \text{ letter} \Rightarrow 'a \text{ rel}$  **where**

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

**fun**  $lconv :: 'b \text{ greek} \Rightarrow 'a \text{ rel}$  **where**

$lconv [] = Id$

$\mid lconv (x \# xs) = lstep x \circ lconv xs$

**lemma**  $lconv\text{-append}[simp]:$

$lconv (xs @ ys) = lconv xs \circ lconv ys$

**by**  $(induct xs) \text{ auto}$

**lemma**  $conversion\text{-join-or-peak-or-cliff}:$

**obtains**  $(join) as bs cs$  **where**  $set as \subseteq \{Grave\}$  **and**  $set bs \subseteq \{Macron\}$  **and**  $set cs \subseteq \{Acute\}$

**and**  $ds = as @ bs @ cs$

$\mid (peak) as bs$  **where**  $ds = as @ ([Acute] @ [Grave]) @ bs$

$\mid (lcliff) as bs$  **where**  $ds = as @ ([Acute] @ [Macron]) @ bs$

$\mid (rcliff) as bs$  **where**  $ds = as @ ([Macron] @ [Grave]) @ bs$

**proof**  $(induct ds \text{ arbitrary: thesis})$

**case**  $(Cons d ds \text{ thesis})$  **note**  $IH = \text{this show ?case}$

**proof**  $(rule IH(1))$

**fix**  $as bs$  **assume**  $ds = as @ ([Acute] @ [Grave]) @ bs$  **then show**  $?case$

**using**  $IH(3)[\text{of } d \# as \ bs]$  **by**  $simp$

**next**

**fix**  $as bs$  **assume**  $ds = as @ ([Acute] @ [Macron]) @ bs$  **then show**  $?case$

**using**  $IH(4)[\text{of } d \# as \ bs]$  **by**  $simp$

**next**

**fix**  $as bs$  **assume**  $ds = as @ ([Macron] @ [Grave]) @ bs$  **then show**  $?case$

**using**  $IH(5)[\text{of } d \# as \ bs]$  **by**  $simp$

**next**

**fix**  $as bs cs$  **assume**  $*$ :  $set as \subseteq \{Grave\}$   $set bs \subseteq \{Macron\}$   $set cs \subseteq \{Acute\}$   
 $ds = as @ bs @ cs$

**show**  $?case$

**proof**  $(cases d)$

**case**  $Grave$  **thus**  $?thesis$  **using**  $* IH(2)[\text{of } d \# as \ bs \ cs]$  **by**  $simp$

**next**

**case**  $Macron$  **show**  $?thesis$

**proof**  $(cases as)$

**case**  $Nil$  **thus**  $?thesis$  **using**  $* Macron IH(2)[\text{of } as \ d \# bs \ cs]$  **by**  $simp$

**next**

```

      case (Cons a as) thus ?thesis using * Macron IH(5)[of [] as @ bs @ cs]
by simp
  qed
next
  case Acute show ?thesis
proof (cases as)
  case Nil note as = this show ?thesis
proof (cases bs)
  case Nil thus ?thesis using * as Acute IH(2)[of [] [] d # cs] by simp
next
  case (Cons b bs) thus ?thesis using * as Acute IH(4)[of [] bs @ cs] by
simp
  qed
next
  case (Cons a as) thus ?thesis using * Acute IH(3)[of [] as @ bs @ cs] by
simp
  qed
qed
qed
qed auto

```

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

```

  assumes map f xs = ys1 @ ys2
  obtains xs1 xs2 where ys1 = map f xs1 ys2 = map f xs2 xs = xs1 @ xs2
proof (insert assms, induct ys1 arbitrary: xs thesis)
  case (Cons y ys) note IH = this show ?case
proof (cases xs)
  case (Cons x xs') show ?thesis
proof (rule IH(1))
  fix xs1 xs2 assume ys = map f xs1 ys2 = map f xs2 xs' = xs1 @ xs2 thus
?thesis
  using Cons IH(2)[of x # xs1 xs2] IH(3) by simp
next
  show map f xs' = ys @ ys2 using Cons IH(3) by simp
qed
qed (insert Cons, simp)
qed auto

```

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

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

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

**lemma** *conversion-to-lconv*:

```

  assumes (u, v) ∈ conversion' M
  obtains xs where snd ' set xs ⊆ M and (u, v) ∈ lconv xs
using assms

```

**proof** (*induct arbitrary: thesis rule: converse-rtrancl-induct*)

**case base show** *?case using base[of []] by simp*

**next**

**case** (*step u' x*)

**from** *step(1)* **obtain** *p* **where** *snd p ∈ M* **and** (*u', x*) ∈ *lstep p*

**by** (*force split: accent.splits*)

**moreover obtain** *xs* **where** *snd ' set xs ⊆ M* (*x, v*) ∈ *lconv xs* **by** (*rule step(3)*)

**ultimately show** *?case using step(4)[of p # xs] by auto*

**qed**

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

*lpeak r a b xs*  $\longleftrightarrow$  ( $\exists$  *as b' cs a' bs. snd ' set as ⊆ under r a*  $\wedge$  *b' ∈ {[(Grave,b)], []}*)

$\wedge$

*snd ' set cs ⊆ under r a*  $\cup$  *under r b*  $\wedge$  *a' ∈ {[(Acute,a)], []}*  $\wedge$

*snd ' set bs ⊆ under r b*  $\wedge$  *xs = as @ b' @ cs @ a' @ bs*)

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

*lcliff r a b xs*  $\longleftrightarrow$  ( $\exists$  *as b' cs a' bs. snd ' set as ⊆ under r a*  $\wedge$  *b' = [(Macron,b)]*)

$\wedge$

*snd ' set cs ⊆ under r a*  $\wedge$  *a' ∈ {[(Acute,a)], []}*  $\wedge$

*snd ' set bs ⊆ under r a*  $\cap$  *under r b*  $\wedge$  *xs = as @ b' @ cs @ a' @ bs*)  $\vee$

( $\exists$  *cs a' bs. snd ' set cs ⊆ under r a*  $\cup$  *under r b*  $\wedge$  *a' ∈ {[(Acute,a)], []}*  $\wedge$

*snd ' set bs ⊆ under r b*  $\wedge$  *xs = cs @ a' @ bs*)

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

*rcliff r a b xs*  $\longleftrightarrow$  ( $\exists$  *as b' cs a' bs. snd ' set as ⊆ under r a*  $\cap$  *under r b*  $\wedge$  *b' ∈ {[(Grave,b)], []}*  $\wedge$

*snd ' set cs ⊆ under r b*  $\wedge$  *a' = [(Macron,a)]*  $\wedge$

*snd ' set bs ⊆ under r b*  $\wedge$  *xs = as @ b' @ cs @ a' @ bs*)  $\vee$

( $\exists$  *as b' cs. snd ' set as ⊆ under r a*  $\wedge$  *b' ∈ {[(Grave,b)], []}*  $\wedge$

*snd ' set cs ⊆ under r a*  $\cup$  *under r b*  $\wedge$  *xs = as @ b' @ cs*)

**lemma** *dd-commute-modulo-conv[case-names wf trans peak lcliff rcliff]*:

**assumes** *wf r* **and** *trans r*

**and** *pk:  $\bigwedge a b s t u. (s, t) \in L a \implies (s, u) \in R b \implies \exists xs. lpeak r a b xs \wedge (t, u) \in lconv xs$*

**and** *lc:  $\bigwedge a b s t u. (s, t) \in L a \implies (s, u) \in E b \implies \exists xs. lcliff r a b xs \wedge (t, u) \in lconv xs$*

**and** *rc:  $\bigwedge a b s t u. (s, t) \in (E a)^{-1} \implies (s, u) \in R b \implies \exists xs. rcliff r a b xs \wedge (t, u) \in lconv xs$*

**shows** *conversion' UNIV ⊆ valley' UNIV*

**proof** (*intro subrelI*)

**fix** *u v*

**assume** (*u, v*) ∈ *conversion' UNIV*

**then obtain** *xs* **where** (*u, v*) ∈ *lconv xs* **by** (*auto intro: conversion-to-lconv[of u v]*)

**then show** (*u, v*) ∈ *valley' UNIV*

**proof** (*induct xs rule: wf-induct[of greek-less r]*)

**case 1 thus** *?case using wf-greek-less[OF <wf r> <trans r>]* .

```

next
  case (2 xs) show ?case
  proof (rule conversion-join-or-peak-or-cliff[of map fst xs])
    fix as bs cs
    assume *: set as  $\subseteq$  {Grave} set bs  $\subseteq$  {Macron} set cs  $\subseteq$  {Acute} map fst xs
    = as @ bs @ cs
    then show (u, v)  $\in$  valley' UNIV
    proof (elim map-eq-append-splits)
      fix as' bs' cs' bcs'
      assume as: set as  $\subseteq$  {Grave} as = map fst as' and
            bs: set bs  $\subseteq$  {Macron} bs = map fst bs' and
            cs: set cs  $\subseteq$  {Acute} cs = map fst cs' and
            xs: xs = as' @ bcs' bcs' = bs' @ cs'
      from as(1)[unfolded as(2)] have as':  $\bigwedge x y. (x,y) \in \text{lconv } as' \implies (x,y) \in$ 
      ( $\bigcup a. R a$ )*
      proof (induct as')
        case (Cons x' xs)
          have  $\bigwedge x y z i. (x,y) \in R i \implies (y,z) \in (\bigcup a. R a)^* \implies (x,z) \in (\bigcup a. R$ 
          a)*
          by (rule rtrancl-trans) auto
          with Cons show ?case by auto
        qed simp
      from bs(1)[unfolded bs(2)] have bs':  $\bigwedge x y. (x,y) \in \text{lconv } bs' \implies (x,y) \in$ 
      ( $\bigcup a. E a$ )*
      proof (induct bs')
        case (Cons x' xs)
          have  $\bigwedge x y z i. (x,y) \in E i \implies (y,z) \in (\bigcup a. E a)^* \implies (x,z) \in (\bigcup a. E a)^*$ 
          by (rule rtrancl-trans) auto
          with Cons show ?case by auto
        qed simp
      from cs(1)[unfolded cs(2)] have cs':  $\bigwedge x y. (x,y) \in \text{lconv } cs' \implies (x,y) \in$ 
      (( $\bigcup a. L a$ )-1)*
      proof (induct cs')
        case (Cons x' xs)
          have  $\bigwedge x y z i. (x,y) \in (L i)^{-1} \implies (y,z) \in ((\bigcup a. L a)^{-1})^* \implies (x,z) \in$ 
          (( $\bigcup a. L a$ )-1)*
          by (rule rtrancl-trans) auto
          with Cons show ?case by auto
        qed simp
      from 2(2) as' bs' cs' show (u, v)  $\in$  valley' UNIV
      unfolding xs lconv-append by auto (meson relcomp.simps)
    qed
  next
  fix as bs assume *: map fst xs = as @ ([Acute] @ [Grave]) @ bs
  {
    fix p a b q t' s' u'
    assume xs: xs = p @ [(Acute,a),(Grave,b)] @ q and p: (u,t')  $\in$  lconv p
      and a: (s',t')  $\in$  L a and b: (s',u')  $\in$  R b and q: (u',v)  $\in$  lconv q
    obtain js where lp: lpeak r a b js and js: (t',u')  $\in$  lconv js using pk[OF a

```

```

b] by auto
  from lp have (js, [(Acute,a),(Grave,b)]) ∈ greek-less r
  unfolding lpeak-def using peak-greek-less[of - r a - b] by fastforce
  then have (p @ js @ q, xs) ∈ greek-less r unfolding xs
  by (intro greek-less-app-mono1 greek-less-app-mono2 ‹trans r›) auto
  moreover have (u, v) ∈ lconv (p @ js @ q)
  using p q js by auto
  ultimately have (u, v) ∈ valley' UNIV using 2(1) by blast
}
with * show (u, v) ∈ valley' UNIV using 2(2)
  by (auto elim!: map-eq-append-splits relcompEpair simp del: append.simps)
simp
next
  fix as bs assume *: map fst xs = as @ ([Acute] @ [Macron]) @ bs
  {
    fix p a b q t' s' u'
    assume xs: xs = p @ [(Acute,a),(Macron,b)] @ q and p: (u,t') ∈ lconv p
      and a: (s',t') ∈ L a and b: (s',u') ∈ E b and q: (u',v) ∈ lconv q
    obtain js where lp: lcliff r a b js and js: (t',u') ∈ lconv js using lc[OF a
b] by auto
    from lp have (js, [(Acute,a),(Macron,b)]) ∈ greek-less r
    unfolding lcliff-def
    using lcliff-greek-less1[OF ‹trans r›, of - a - b] lcliff-greek-less2[OF ‹trans
r›, of - a b]
    by fastforce
    then have (p @ js @ q, xs) ∈ greek-less r unfolding xs
    by (intro greek-less-app-mono1 greek-less-app-mono2 ‹trans r›) auto
    moreover have (u, v) ∈ lconv (p @ js @ q)
    using p q js by auto
    ultimately have (u, v) ∈ valley' UNIV using 2(1) by blast
  }
with * show (u, v) ∈ valley' UNIV using 2(2)
  by (auto elim!: map-eq-append-splits relcompEpair simp del: append.simps)
simp
next
  fix as bs assume *: map fst xs = as @ ([Macron] @ [Grave]) @ bs
  {
    fix p a b q t' s' u'
    assume xs: xs = p @ [(Macron,a),(Grave,b)] @ q and p: (u,t') ∈ lconv p
      and a: (s',t') ∈ (E a)-1 and b: (s',u') ∈ R b and q: (u',v) ∈ lconv q
    obtain js where lp: rcliff r a b js and js: (t',u') ∈ lconv js using rc[OF a
b] by auto
    from lp have (js, [(Macron,a),(Grave,b)]) ∈ greek-less r
    unfolding rcliff-def
    using rcliff-greek-less1[OF ‹trans r›, of - a - b] rcliff-greek-less2[OF ‹trans
r›, of - a - b]
    by fastforce
    then have (p @ js @ q, xs) ∈ greek-less r unfolding xs
    by (intro greek-less-app-mono1 greek-less-app-mono2 ‹trans r›) auto

```

```

    moreover have (u, v) ∈ lconv (p @ js @ q)
    using p q js by auto
    ultimately have (u, v) ∈ valley' UNIV using 2(1) by blast
  }
  with * show (u, v) ∈ valley' UNIV using 2(2)
  by (auto elim!: map-eq-append-splits relcompEpair simp del: append.simps)
simp
qed
qed
qed

```

### 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 and trans r
  and pk:  $\bigwedge a b s t u. (s, t) \in L a \implies (s, u) \in R b \implies$ 
     $(t, u) \in \text{conversion}'(\text{under } r a) O (R b)^= O \text{conversion}'(\text{under } r a \cup \text{under } r$ 
  b) O
     $((L a)^{-1})^= O \text{conversion}'(\text{under } r b)$ 
  and lc:  $\bigwedge a b s t u. (s, t) \in L a \implies (s, u) \in E b \implies$ 
     $(t, u) \in \text{conversion}'(\text{under } r a) O E b O \text{conversion}'(\text{under } r a) O$ 
     $((L a)^{-1})^= O \text{conversion}'(\text{under } r a \cap \text{under } r b) \vee$ 
     $(t, u) \in \text{conversion}'(\text{under } r a \cup \text{under } r b) O ((L a)^{-1})^= O \text{conversion}'$ 
  (under r b)
  and rc:  $\bigwedge a b s t u. (s, t) \in (E a)^{-1} \implies (s, u) \in R b \implies$ 
     $(t, u) \in \text{conversion}'(\text{under } r a \cap \text{under } r b) O (R b)^= O \text{conversion}'(\text{under } r$ 
  b) O
     $E a O \text{conversion}'(\text{under } r b) \vee$ 
     $(t, u) \in \text{conversion}'(\text{under } r a) O (R b)^= O \text{conversion}'(\text{under } r a \cup \text{under } r$ 
  b)
  shows conversion' UNIV  $\subseteq$  valley' UNIV
proof (cases rule: dd-commute-modulo-conv[of r])
  case (peak a b s t u)
  {
    fix w x y z
    assume (t, w) ∈ conversion' (under r a)
    from conversion-to-lconv[OF this]
    obtain as where snd ' set as  $\subseteq$  under r a (t, w) ∈ lconv as by auto
    moreover assume (w, x) ∈ (R b)^=
    then obtain b' where b' ∈ {[(Grave,b)],[]} (w, x) ∈ lconv b' by fastforce
    moreover assume (x, y) ∈ conversion' (under r a  $\cup$  under r b)
    from conversion-to-lconv[OF this]
    obtain cs where snd ' set cs  $\subseteq$  under r a  $\cup$  under r b (x, y) ∈ lconv cs by
  auto
    moreover assume (y, z) ∈ ((L a)^{-1})^=
  
```



```

then obtain  $a'$  where  $a' \in \{[(Acute,a)],[]\}$   $(y, z) \in lconv\ a'$  by fastforce
moreover assume  $(z, u) \in conversion'$  (under  $r\ b$ )
from conversion-to-lconv[OF this]
obtain  $bs$  where  $snd\ 'set\ bs \subseteq under\ r\ b\ (z, u) \in lconv\ bs$  by auto
ultimately have  $\exists xs. lpeak\ r\ a\ b\ xs \wedge (t, u) \in lconv\ xs$ 
by (intro exI[of - as @ b' @ cs @ a' @ bs], unfold lconv-append lpeak-def) blast
}
then show ?case using pk[OF peak] by blast
next
case (lcliff a b s t u)
{
  fix  $w\ x\ y\ z$ 
  assume  $(t, w) \in conversion'$  (under  $r\ a$ )
  from conversion-to-lconv[OF this]
  obtain  $as$  where  $snd\ 'set\ as \subseteq under\ r\ a\ (t, w) \in lconv\ as$  by auto
  moreover assume  $(w, x) \in E\ b$ 
  then obtain  $b'$  where  $b' = [(Macron,b)]$   $(w, x) \in lconv\ b'$  by fastforce
  moreover assume  $(x, y) \in conversion'$  (under  $r\ a$ )
  from conversion-to-lconv[OF this]
  obtain  $cs$  where  $snd\ 'set\ cs \subseteq under\ r\ a\ (x, y) \in lconv\ cs$  by auto
  moreover assume  $(y, z) \in ((L\ a)^{-1})=$ 
  then obtain  $a'$  where  $a' \in \{[(Acute,a)],[]\}$   $(y, z) \in lconv\ a'$  by fastforce
  moreover assume  $(z, u) \in conversion'$  (under  $r\ a \cap under\ r\ b$ )
  from conversion-to-lconv[OF this]
  obtain  $bs$  where  $snd\ 'set\ bs \subseteq under\ r\ a \cap under\ r\ b\ (z, u) \in lconv\ bs$  by
auto
  ultimately have  $\exists xs. lcliff\ r\ a\ b\ xs \wedge (t, u) \in lconv\ xs$ 
  by (intro exI[of - as @ b' @ cs @ a' @ bs], unfold lconv-append lcliff-def) blast
}
moreover {
  fix  $w\ x$ 
  assume  $(t, w) \in conversion'$  (under  $r\ a \cup under\ r\ b$ )
  from conversion-to-lconv[OF this]
  obtain  $cs$  where  $snd\ 'set\ cs \subseteq under\ r\ a \cup under\ r\ b\ (t, w) \in lconv\ cs$  by
auto
  moreover assume  $(w, x) \in ((L\ a)^{-1})=$ 
  then obtain  $a'$  where  $a' \in \{[(Acute,a)],[]\}$   $(w, x) \in lconv\ a'$  by fastforce
  moreover assume  $(x, u) \in conversion'$  (under  $r\ b$ )
  from conversion-to-lconv[OF this]
  obtain  $bs$  where  $snd\ 'set\ bs \subseteq under\ r\ b\ (x, u) \in lconv\ bs$  by auto
  ultimately have  $\exists xs. lcliff\ r\ a\ b\ xs \wedge (t, u) \in lconv\ xs$ 
  by (intro exI[of - cs @ a' @ bs], unfold lconv-append lcliff-def) blast
}
ultimately show ?case using lc[OF lcliff] by blast
next
case (rcliff a b s t u)
{
  fix  $w\ x\ y\ z$ 
  assume  $(t, w) \in conversion'$  (under  $r\ a \cap under\ r\ b$ )

```

**from** *conversion-to-lconv*[*OF this*]  
**obtain** *as* **where** *snd* ‘ *set as*  $\subseteq$  *under r a*  $\cap$  *under r b*  $(t, w) \in$  *lconv as* **by**  
*auto*  
**moreover assume**  $(w, x) \in (R b)^=$   
**then obtain** *b'* **where**  $b' \in \{[(Grave, b), []]\}$   $(w, x) \in$  *lconv b'* **by** *fastforce*  
**moreover assume**  $(x, y) \in$  *conversion'* (*under r b*)  
**from** *conversion-to-lconv*[*OF this*]  
**obtain** *cs* **where** *snd* ‘ *set cs*  $\subseteq$  *under r b*  $(x, y) \in$  *lconv cs* **by** *auto*  
**moreover assume**  $(y, z) \in E a$   
**then obtain** *a'* **where**  $a' = [(Macron, a)]$   $(y, z) \in$  *lconv a'* **by** *fastforce*  
**moreover assume**  $(z, u) \in$  *conversion'* (*under r b*)  
**from** *conversion-to-lconv*[*OF this*]  
**obtain** *bs* **where** *snd* ‘ *set bs*  $\subseteq$  *under r b*  $(z, u) \in$  *lconv bs* **by** *auto*  
**ultimately have**  $\exists xs. rcliff r a b xs \wedge (t, u) \in$  *lconv xs*  
**by** (*intro exI*[*of - as @ b' @ cs @ a' @ bs*], *unfold lconv-append rcliff-def*) *blast*  
**}**  
**moreover {**  
**fix** *w x*  
**assume**  $(t, w) \in$  *conversion'* (*under r a*)  
**from** *conversion-to-lconv*[*OF this*]  
**obtain** *as* **where** *snd* ‘ *set as*  $\subseteq$  *under r a*  $(t, w) \in$  *lconv as* **by** *auto*  
**moreover assume**  $(w, x) \in (R b)^=$   
**then obtain** *b'* **where**  $b' \in \{[(Grave, b), []]\}$   $(w, x) \in$  *lconv b'* **by** *fastforce*  
**moreover assume**  $(x, u) \in$  *conversion'* (*under r a*  $\cup$  *under r b*)  
**from** *conversion-to-lconv*[*OF this*]  
**obtain** *cs* **where** *snd* ‘ *set cs*  $\subseteq$  *under r a*  $\cup$  *under r b*  $(x, u) \in$  *lconv cs* **by**  
*auto*  
**ultimately have**  $\exists xs. rcliff r a b xs \wedge (t, u) \in$  *lconv xs*  
**by** (*intro exI*[*of - as @ b' @ cs*], *unfold lconv-append rcliff-def*) *blast*  
**}**  
**ultimately show** *?case using rc*[*OF rcliff*] **by** *blast*  
**qed fact+**  
**end**

Decreasing diagrams for Church-Rosser modulo.

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

**assumes** *wf r* **and** *trans r* **and**  $E: \bigwedge i. sym (E i)$   
**and** *pk*:  $\bigwedge a b s t u. (s, t) \in L a \implies (s, u) \in L b \implies$   
 $(t, u) \in conversion' L L E (under r a) O (L b)^= O conversion' L L E (under$   
 $r a \cup under r b) O$   
 $((L a)^{-1})^= O conversion' L L E (under r b)$   
**and** *cl*:  $\bigwedge a b s t u. (s, t) \in L a \implies (s, u) \in E b \implies$   
 $(t, u) \in conversion' L L E (under r a) O E b O conversion' L L E (under r a)$   
 $O$   
 $((L a)^{-1})^= O conversion' L L E (under r a \cap under r b) \vee$   
 $(t, u) \in conversion' L L E (under r a \cup under r b) O ((L a)^{-1})^= O conversion'$   
 $L L E (under r b)$   
**shows**  $conversion' L L E UNIV \subseteq valley' L L E UNIV$

**proof** (*induct rule: dd-commute-modulo*[of  $r$ ])  
**note**  $E' = E$ [*unfolded sym-conv-converse-eq*]  
**case** (*rcliff a b s t u*) **show** ?*case*  
**using**  $cl$ [*OF rcliff(2) rcliff(1)*[*unfolded E'*], *unfolded converse-iff*[of  $t u$ , *symmetric*]]  
**by** (*auto simp only: E' converse-inward*) (*auto simp only: ac-simps*)  
**qed fact+**

### 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**  $pk: \bigwedge a b s t u. (s, t) \in L a \implies (s, u) \in R b \implies$   
 $(t, u) \in conversion'' L R (under\ r\ a) O (R b)^= O conversion'' L R (under\ r\ a$   
 $\cup under\ r\ b) O$   
 $((L a)^{-1})^= O conversion'' L R (under\ r\ b)$   
**shows** *commute*  $(\bigcup i. L i) (\bigcup i. R i)$   
**proof** –  
**have**  $((\bigcup i. L i)^{-1})^* O (\bigcup i. R i)^* \subseteq conversion'' L R UNIV$  **by** *regexp*  
**also have**  $\dots \subseteq valley'' L R UNIV$   
**using** *dd-commute-modulo*[*OF assms(1,2), of L R λ-. {}*] **pk by** *auto*  
**finally show** ?*thesis by* (*simp only: commute-def*)  
**qed**

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 ⊆ r*  
**begin**

**private abbreviation** (*input*)  $down :: ('b \Rightarrow 'a\ rel) \Rightarrow ('b \Rightarrow 'a\ rel)$  **where**  
 $down\ L \equiv \lambda i. \bigcup j \in under\ q\ i. L j$

**private lemma** *Union-down*:  $(\bigcup i. down\ L\ i) = (\bigcup i. L\ i)$   
**using**  $\langle refl\ q \rangle$  **by** (*auto simp: refl-on-def under-def*)

Extended decreasing diagrams for commutation.

**lemma** *edd-commute*[*case-names wf transr transq reflq compat peak*]:  
**assumes**  $pk: \bigwedge a b s t u. (s, t) \in L a \implies (s, u) \in R b \implies$   
 $(t, u) \in conversion'' L R (under\ r\ a) O (down\ R\ b)^= O conversion'' L R (under$   
 $r\ a \cup under\ r\ b) O$

$((\text{down } L \ a)^{-1})^= \text{O conversion'' } L \ R \ (\text{under } r \ b)$   
**shows** *commute*  $(\bigcup i. L \ i) \ (\bigcup i. R \ i)$   
**unfolding** *Union-down[of L, symmetric] Union-down[of R, symmetric]*  
**proof** (*induct rule: dd-commute[of r down L down R]*)  
**case** (*peak a b s t u*)  
**then obtain**  $a' \ b'$  **where**  $a': (a', a) \in q \ (s, t) \in L \ a'$  **and**  $b': (b', b) \in q \ (s, u) \in R \ b'$   
**by** (*auto simp: under-def*)  
**have**  $\bigwedge a' \ a. (a', a) \in q \implies \text{under } r \ a' \subseteq \text{under } r \ a$  **using** *compat* **by** (*auto simp: under-def*)  
**then have**  $\text{aux1}: \bigwedge a' \ a \ L. (a', a) \in q \implies (\bigcup i \in \text{under } r \ a'. L \ i) \subseteq (\bigcup i \in \text{under } r \ a. L \ i)$  **by** *auto*  
**have**  $\text{aux2}: \bigwedge a' \ a \ L. (a', a) \in q \implies \text{down } L \ a' \subseteq \text{down } L \ a$   
**using** *<trans q>* **by** (*auto simp: under-def trans-def*)  
**have**  $\text{aux3}: \bigwedge a \ L. (\bigcup i \in \text{under } r \ a. L \ i) \subseteq (\bigcup i \in \text{under } r \ a. \text{down } L \ i)$   
**using** *<refl q>* **by** (*auto simp: under-def refl-on-def*)  
**from**  $\text{aux1}[OF \ a'(1), \text{of } L] \ \text{aux1}[OF \ a'(1), \text{of } R] \ \text{aux2}[OF \ a'(1), \text{of } L]$   
 $\text{aux1}[OF \ b'(1), \text{of } L] \ \text{aux1}[OF \ b'(1), \text{of } R] \ \text{aux2}[OF \ b'(1), \text{of } R]$   
 $\text{aux3}[\text{of } L] \ \text{aux3}[\text{of } R]$   
**show** *?case*  
**by** (*intro subsetD[OF - pk[OF <(s, t) ∈ L a'> <(s, u) ∈ R b'>]], unfold UN-Un*)  
*(intro relcomp-mono rtrancl-mono Un-mono iffD2[OF converse-mono]; fast)*  
**qed fact+**

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

- [1] B. Felgenhauer and V. van Oostrom. Proof orders for decreasing diagrams. In *Proc. 24th International Conference on Rewriting Techniques and Applications*, number 21 in Leibniz International Proceedings in Informatics, pages 174–189, 2013.
- [2] N. Hirokawa and A. Middeldorp. Decreasing diagrams and relative termination. In *Proc. 5th International Joint Conference on Automated Reasoning*, volume 6173 of *Lecture Notes in Artificial Intelligence*, pages 487–501, 2010.
- [3] V. van Oostrom. Confluence by decreasing diagrams – converted. In *Proc. 19th International Conference on Rewriting Techniques and Ap-*

*plications*, volume 5117 of *Lecture Notes in Computer Science*, pages 306–320, 2008.