

# 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</b>	<b>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 . . . . .	3
1.4	Incrementality of <i>mult1</i> and <i>mult</i> . . . . .	4
1.5	Well-orders and well-quasi-orders . . . . .	4
1.6	Splitting lists into prefix, element, and suffix . . . . .	5
<b>2</b>	<b>Decreasing Diagrams</b>	<b>6</b>
2.1	Greek accents . . . . .	6
2.2	Comparing Greek strings . . . . .	6
2.3	Preservation of strict partial orders . . . . .	7

2.4	Involution . . . . .	8
2.5	Monotonicity of <i>greek-less</i> $r$ . . . . .	9
2.6	Well-founded-ness of <i>greek-less</i> $r$ . . . . .	10
2.7	Basic Comparisons . . . . .	10
2.8	Labeled abstract rewriting . . . . .	11
<b>3</b>	<b>Results</b>	<b>13</b>
3.1	Church-Rosser modulo . . . . .	13
3.2	Commutation and confluence . . . . .	14
3.3	Extended decreasing diagrams . . . . .	14

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

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

**lemma** *mono-lex2*:  $\text{mono } (\text{lex-prod } r)$   
*<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* ::  $'a \text{ set} \Rightarrow \text{bool}$  **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*:  $\text{mono } f$

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

*<proof>*

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

*<proof>*

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

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

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

*<proof>*

**lemma** *strict-order-lfp*:

**assumes** *mono*  $f$  **and**  $\bigwedge R. \text{strict-order } R \implies \text{strict-order } (f R)$

**shows**  $\text{strict-order } (\text{lfp } f)$

*<proof>*

**lemma** *trans-lfp*:

**assumes** *mono*  $f$  **and**  $\bigwedge R. \text{trans } R \implies \text{trans } (f R)$

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

*<proof>*

**end**

### 1.3 Multiset extension

**lemma** *mulex-iff-mult*:  $\text{mulex } r M N \iff (M, N) \in \text{mult } \{(M, N) . r M N\}$

*<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-on-add-mset*:  $(M,N) \in \text{mult } r \implies (\text{add-mset } k M, \text{add-mset } k N) \in \text{mult } r$   
*<proof>*

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

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

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

**lemma** *trans-mult*:  $\text{trans } (\text{mult } R)$   
*<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**  $\bigwedge x y. x \in \text{set-mset } N \implies y \in \text{set-mset } M \implies (x,y) \in R \implies (f x, f y) \in R'$   
  **and**  $(N, M) \in \text{mult } R$   
  **shows**  $(\text{image-mset } f N, \text{image-mset } f M) \in \text{mult } R'$   
*<proof>*

## 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*:  
   $wf p \iff wfp\text{-on } (\lambda a b. (a, b) \in p)$  *UNIV*  
*<proof>*

**lemma** *well-order-implies-wqo*:  
**assumes** *well-order r*  
**shows** *wqo-on* ( $\lambda a b. (a, b) \in r$ ) *UNIV*  
 $\langle$ *proof* $\rangle$

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

**fun** *list-splits* ::  $'a$  list  $\Rightarrow$  ( $'a$  list  $\times$   $'a$   $\times$   $'a$  list) list **where**  
*list-splits* [] = []  
 $|$  *list-splits* (x # xs) = ( [], x, xs ) # map ( $\lambda(xs, x', xs'). (x \# xs, x', xs')$ ) (*list-splits* xs)

**lemma** *list-splits-empty[simp]*:  
*list-splits* xs = []  $\longleftrightarrow$  xs = []  
 $\langle$ *proof* $\rangle$

**lemma** *elem-list-splits-append*:  
**assumes** (ys, y, zs)  $\in$  set (*list-splits* xs)  
**shows** ys @ [y] @ zs = xs  
 $\langle$ *proof* $\rangle$

**lemma** *elem-list-splits-length*:  
**assumes** (ys, y, zs)  $\in$  set (*list-splits* xs)  
**shows** length ys < length xs **and** length zs < length xs  
 $\langle$ *proof* $\rangle$

**lemma** *elem-list-splits-elem*:  
**assumes** (xs, y, ys)  $\in$  set (*list-splits* zs)  
**shows** y  $\in$  set zs  
 $\langle$ *proof* $\rangle$

**lemma** *list-splits-append*:  
*list-splits* (xs @ ys) = map ( $\lambda(xs', x', ys'). (xs', x', ys' @ ys)$ ) (*list-splits* xs) @  
map ( $\lambda(xs', x', ys'). (xs @ xs', x', ys')$ ) (*list-splits* ys)  
 $\langle$ *proof* $\rangle$

**lemma** *list-splits-rev*:  
*list-splits* (rev xs) = map ( $\lambda(xs, x, ys). (rev ys, x, rev xs)$ ) (rev (*list-splits* xs))  
 $\langle$ *proof* $\rangle$

**lemma** *list-splits-map*:  
*list-splits* (map f xs) = map ( $\lambda(xs, x, ys). (map f xs, f x, map f ys)$ ) (*list-splits* xs)  
 $\langle$ *proof* $\rangle$

**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 = { Acute, Grave, Macron }
<proof>

```

```

lemma finite-accent: finite (UNIV :: accent set)
<proof>

```

```

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
<proof>

```

### 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
<proof>

```

```

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

```

<proof>

**lemma** *adj-msog-single*:

*adj-msog*  $xs\ zs\ (x, []) = (x, (case\ fst\ x\ of\ Grave\ \Rightarrow\ xs\ |\ Acute\ \Rightarrow\ zs\ |\ Macron\ \Rightarrow\ []))$   
*<proof>*

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

**assumes**  $(x, xs) \in set\ mset\ (ms\ of\ greek\ ys)$   
**shows**  $x \in set\ ys$   
*<proof>*

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

**assumes**  $(x, t) \in \# ms\ of\ greek\ s$   
**shows**  $length\ s > length\ t$   
*<proof>*

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

**definition** *nest* ::  $('a \times 'a)\ set \Rightarrow ('a\ greek \times 'a\ greek)\ set \Rightarrow ('a\ greek \times 'a\ greek)\ set$  **where**

*[simp]*:  $nest\ r\ s = \{(a, b). (ms\ of\ greek\ a, ms\ of\ greek\ b) \in mult\ (letter\ less\ r\ < *lex* > s)\}$

**lemma** *mono-nest*:  $mono\ (nest\ r)$

*<proof>*

**lemma** *nest-mono*[*mono-set*]:  $x \subseteq y \Longrightarrow (a, b) \in nest\ r\ x \longrightarrow (a, b) \in nest\ r\ y$

*<proof>*

**definition** *greek-less* ::  $('a \times 'a)\ set \Rightarrow ('a\ greek \times 'a\ greek)\ set$  **where**

*greek-less*  $r = lfp\ (nest\ r)$

**lemma** *greek-less-unfold*:

*greek-less*  $r = nest\ r\ (greek\ less\ r)$   
*<proof>*

## 2.3 Preservation of strict partial orders

**lemma** *strict-order-letter-less*:

**assumes** *strict-order*  $r$   
**shows** *strict-order*  $(letter\ less\ r)$   
*<proof>*

**lemma** *strict-order-nest*:

**assumes**  $r$ : *strict-order*  $r$  **and**  $R$ : *strict-order*  $R$   
**shows** *strict-order*  $(nest\ r\ R)$   
*<proof>*

**lemma** *strict-order-greek-less*:  
**assumes** *strict-order r*  
**shows** *strict-order (greek-less r)*  
 $\langle$ *proof* $\rangle$

**lemma** *trans-letter-less*:  
**assumes** *trans r*  
**shows** *trans (letter-less r)*  
 $\langle$ *proof* $\rangle$

**lemma** *trans-order-nest*: *trans (nest r R)*  
 $\langle$ *proof* $\rangle$

**lemma** *trans-greek-less[simp]*: *trans (greek-less r)*  
 $\langle$ *proof* $\rangle$

**lemma** *mono-greek-less*: *mono greek-less*  
 $\langle$ *proof* $\rangle$

## 2.4 Involution

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

**lemma** *inv-letter-pair[simp]*:  
*inv-letter* (a, x) = (case a of Grave  $\Rightarrow$  Acute | Acute  $\Rightarrow$  Grave | Macron  $\Rightarrow$  Macron, x)  
 $\langle$ *proof* $\rangle$

**lemma** *snd-inv-letter[simp]*:  
*snd* (*inv-letter* x) = *snd* x  
 $\langle$ *proof* $\rangle$

**lemma** *inv-letter-invol[simp]*:  
*inv-letter* (*inv-letter* x) = x  
 $\langle$ *proof* $\rangle$

**lemma** *inv-letter-mono[simp]*:  
**assumes** (x, y)  $\in$  *letter-less r*  
**shows** (*inv-letter* x, *inv-letter* y)  $\in$  *letter-less r*  
 $\langle$ *proof* $\rangle$

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

**lemma** *inv-greek-invol[simp]*:



$inv\text{-greek } (inv\text{-greek } s) = s$   
 ⟨proof⟩

**lemma** *inv-greek-append*:

$inv\text{-greek } (s @ t) = inv\text{-greek } t @ inv\text{-greek } s$   
 ⟨proof⟩

**definition** *inv-msog* :: ('a letter × 'a greek) multiset ⇒ ('a letter × 'a greek) multiset **where**

$inv\text{-msog } M = image\text{-mset } (\lambda(x, t). (inv\text{-letter } x, inv\text{-greek } t)) M$

**lemma** *inv-msog-invol[simp]*:

$inv\text{-msog } (inv\text{-msog } M) = M$   
 ⟨proof⟩

**lemma** *ms-of-greek-inv-greek*:

$ms\text{-of-greek } (inv\text{-greek } M) = inv\text{-msog } (ms\text{-of-greek } M)$   
 ⟨proof⟩

**lemma** *inv-greek-mono*:

**assumes** *trans r and*  $(s, t) \in greek\text{-less } r$   
**shows**  $(inv\text{-greek } s, inv\text{-greek } t) \in greek\text{-less } r$   
 ⟨proof⟩

## 2.5 Monotonicity of *greek-less r*

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

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

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

**assumes**  $b \neq []$   
**shows**  $(a, b) \in greek\text{-less } r \longleftrightarrow (a, b) \in nest\ r (greek\text{-less } r)$   
 ⟨proof⟩

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

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

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

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

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

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

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

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

**lemma** *greek-less-app-mono2*:  
**assumes** *trans r* **and**  $(s, t) \in \text{greek-less } r$   
**shows**  $(p @ s, p @ t) \in \text{greek-less } r$   
 $\langle \text{proof} \rangle$

**lemma** *greek-less-app-mono1*:  
**assumes** *trans r* **and**  $(s, t) \in \text{greek-less } r$   
**shows**  $(s @ p, t @ p) \in \text{greek-less } r$   
 $\langle \text{proof} \rangle$

## 2.6 Well-founded-ness of *greek-less r*

**lemma** *greek-embed*:  
**assumes** *trans r*  
**shows** *list-emb*  $(\lambda a b. (a, b): \text{reflcl } (\text{letter-less } r)) a b \implies (a, b) \in \text{reflcl } (\text{greek-less } r)$   
 $\langle \text{proof} \rangle$

**lemma** *wqo-letter-less*:  
**assumes** *t: trans r* **and** *w: wqo-on*  $(\lambda a b. (a, b) \in r^=)$  *UNIV*  
**shows** *wqo-on*  $(\lambda a b. (a, b) \in (\text{letter-less } r)^=)$  *UNIV*  
 $\langle \text{proof} \rangle$

**lemma** *wf-greek-less*:  
**assumes** *wf r* **and** *trans r*  
**shows** *wf*  $(\text{greek-less } r)$   
 $\langle \text{proof} \rangle$

## 2.7 Basic Comparisons

**lemma** *pairwise-imp-mult*:  
**assumes**  $N \neq \{\#\}$  **and**  $\forall x \in \text{set-mset } M. \exists y \in \text{set-mset } N. (x, y) \in r$   
**shows**  $(M, N) \in \text{mult } r$   
 $\langle \text{proof} \rangle$

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

**lemma** *peak-greek-less*:  
**assumes** *as: snd ' set as  $\subseteq$  under r a* **and** *b': b'  $\in$   $\{[(\text{Grave},b)], []\}$*   
**and** *cs: snd ' set cs  $\subseteq$  under r a  $\cup$  under r b* **and** *a': a'  $\in$   $\{[(\text{Acute},a)], []\}$*   
**and** *bs: snd ' set bs  $\subseteq$  under r b*  
**shows**  $(as @ b' @ cs @ a' @ bs, [(\text{Acute},a), (\text{Grave},b)]) \in \text{greek-less } r$   
 $\langle \text{proof} \rangle$

**lemma** *rcliff-greek-less1*:  
**assumes** *trans r*  
**and** *as*: *snd* ‘ *set as*  $\subseteq$  *under r a*  $\cap$  *under r b* **and** *b'*: *b'*  $\in$   $\{[(\text{Grave},b)],[]\}$   
**and** *cs*: *snd* ‘ *set cs*  $\subseteq$  *under r b* **and** *a'*: *a'* =  $[(\text{Macron},a)]$   
**and** *bs*: *snd* ‘ *set bs*  $\subseteq$  *under r b*  
**shows** (*as* @ *b'* @ *cs* @ *a'* @ *bs*,  $[(\text{Macron},a),(\text{Grave},b)]$ )  $\in$  *greek-less r*  
 $\langle$ *proof* $\rangle$

**lemma** *rcliff-greek-less2*:  
**assumes** *trans r*  
**and** *as*: *snd* ‘ *set as*  $\subseteq$  *under r a* **and** *b'*: *b'*  $\in$   $\{[(\text{Grave},b)],[]\}$   
**and** *cs*: *snd* ‘ *set cs*  $\subseteq$  *under r a*  $\cup$  *under r b*  
**shows** (*as* @ *b'* @ *cs*,  $[(\text{Macron},a),(\text{Grave},b)]$ )  $\in$  *greek-less r*  
 $\langle$ *proof* $\rangle$

**lemma** *snd-inv-greek [simp]*: *snd* ‘ *set (inv-greek as)* = *snd* ‘ *set as*  
 $\langle$ *proof* $\rangle$

**lemma** *lcliff-greek-less1*:  
**assumes** *trans r*  
**and** *as*: *snd* ‘ *set as*  $\subseteq$  *under r a* **and** *b'*: *b'* =  $[(\text{Macron},b)]$   
**and** *cs*: *snd* ‘ *set cs*  $\subseteq$  *under r a* **and** *a'*: *a'*  $\in$   $\{[(\text{Acute},a)],[]\}$   
**and** *bs*: *snd* ‘ *set bs*  $\subseteq$  *under r a*  $\cap$  *under r b*  
**shows** (*as* @ *b'* @ *cs* @ *a'* @ *bs*,  $[(\text{Acute},a),(\text{Macron},b)]$ )  $\in$  *greek-less r*  
 $\langle$ *proof* $\rangle$

**lemma** *lcliff-greek-less2*:  
**assumes** *trans r*  
**and** *cs*: *snd* ‘ *set cs*  $\subseteq$  *under r a*  $\cup$  *under r b* **and** *a'*: *a'*  $\in$   $\{[(\text{Acute},a)],[]\}$   
**and** *bs*: *snd* ‘ *set bs*  $\subseteq$  *under r b*  
**shows** (*cs* @ *a'* @ *bs*,  $[(\text{Acute},a),(\text{Macron},b)]$ )  $\in$  *greek-less r*  
 $\langle$ *proof* $\rangle$

## 2.8 Labeled abstract rewriting

**context**

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

**begin**

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

[*simp*]: *lstep x* = (*case x of* (*a, i*)  $\Rightarrow$  (*case a of* *Acute*  $\Rightarrow$  (*L i*)<sup>-1</sup> | *Grave*  $\Rightarrow$  *R i*  
| *Macron*  $\Rightarrow$  *E i*))

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

*lconv* [] = *Id*  
| *lconv (x # xs)* = *lstep x* *O lconv xs*

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

$lconv (xs @ ys) = lconv xs \ O \ lconv ys$   
 ⟨proof⟩

**lemma** *conversion-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**  $ds = as @ bs @ cs$

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

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

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

⟨proof⟩

**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⟩

**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**  $(u, v) \in conversion' \ M$

**obtains**  $xs$  **where**  $snd \ ' \ set \ xs \subseteq M$  **and**  $(u, v) \in lconv \ xs$

⟨proof⟩

**definition** *lpeak* ::  $'b \ rel \Rightarrow 'b \Rightarrow 'b \Rightarrow 'b \ greek \Rightarrow bool$  **where**

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

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

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

**definition** *lcliff* ::  $'b \ rel \Rightarrow 'b \Rightarrow 'b \Rightarrow 'b \ greek \Rightarrow bool$  **where**

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

$snd \ ' \ set \ cs \subseteq under \ r \ a \wedge a' \in \{[(Acute, a)], []\} \wedge$

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

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

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

**definition** *rcliff* ::  $'b \ rel \Rightarrow 'b \Rightarrow 'b \Rightarrow 'b \ greek \Rightarrow bool$  **where**

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

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

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

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

$snd \text{ ' set } cs \subseteq \text{ under } r \ a \cup \text{ under } r \ b \wedge xs = as \ @ \ b' \ @ \ cs)$

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

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

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

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

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

**shows** *conversion' UNIV*  $\subseteq$  *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 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*:  $\bigwedge a \ b \ s \ t \ u. (s, t) \in L \ a \implies (s, u) \in E \ b \implies$

$(t, u) \in \text{conversion}'(\text{under } r \ a) \ O \ E \ b \ O \ \text{conversion}'(\text{under } r \ a) \ O$

$((L \ a)^{-1})^{=} \ O \ \text{conversion}'(\text{under } r \ a \cap \text{under } r \ b) \ \vee$

$(t, u) \in \text{conversion}'(\text{under } r \ a \cup \text{under } r \ b) \ O \ ((L \ a)^{-1})^{=} \ O \ \text{conversion}'(\text{under } r \ b)$

**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>*

**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*:  $\bigwedge i. \text{sym } (E \ i)$

**and** *pk*:  $\bigwedge a \ b \ s \ t \ u. (s, t) \in L \ a \implies (s, u) \in L \ b \implies$

$(t, u) \in \text{conversion}' \ L \ L \ E \ (\text{under } r \ a) \ O \ (L \ b)^{=} \ O \ \text{conversion}' \ L \ L \ E \ (\text{under } r \ a \cup \text{under } r \ b) \ O$

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

**and** *cl*:  $\bigwedge a \ b \ s \ t \ u. (s, t) \in L \ a \implies (s, u) \in E \ b \implies$

$(t, u) \in \text{conversion}' L L E (\text{under } r a) O E b O \text{conversion}' L L E (\text{under } r a)$   
 $O$   
 $((L a)^{-1})^= O \text{conversion}' L L E (\text{under } r a \cap \text{under } r b) \vee$   
 $(t, u) \in \text{conversion}' L L E (\text{under } r a \cup \text{under } r b) O ((L a)^{-1})^= O \text{conversion}'$   
 $L L E (\text{under } r b)$   
**shows**  $\text{conversion}' L L E UNIV \subseteq \text{valley}' L L E UNIV$   
 $\langle \text{proof} \rangle$

### 3.2 Commutation and confluence

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

Decreasing diagrams for commutation.

**lemma**  $dd\text{-commute}[\text{case-names wf trans peak}]$ :  
**assumes**  $wf r$  **and**  $\text{trans } r$   
**and**  $pk: \bigwedge a b s t u. (s, t) \in L a \implies (s, u) \in R b \implies$   
 $(t, u) \in \text{conversion}'' L R (\text{under } r a) O (R b)^= O \text{conversion}'' L R (\text{under } r a$   
 $\cup \text{under } r b) O$   
 $((L a)^{-1})^= O \text{conversion}'' L R (\text{under } r b)$   
**shows**  $\text{commute } (\bigcup i. L i) (\bigcup i. R i)$   
 $\langle \text{proof} \rangle$

Decreasing diagrams for confluence.

**lemmas**  $dd\text{-cr}[\text{case-names wf trans peak}] =$   
 $dd\text{-commute}[\text{of - L L for L, unfolded CR-iff-self-commute}[\text{symmetric}]]$

### 3.3 Extended decreasing diagrams

**context**

**fixes**  $r q :: 'b \text{ rel}$   
**assumes**  $wf r$  **and**  $\text{trans } r$  **and**  $\text{trans } q$  **and**  $\text{refl } q$  **and**  $\text{compat: } r O q \subseteq r$   
**begin**

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

**private lemma**  $\text{Union-down: } (\bigcup i. \text{down } L i) = (\bigcup i. L i)$   
 $\langle \text{proof} \rangle$

Extended decreasing diagrams for commutation.

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

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

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