# Decreasing-Diagrams-II

By Bertram Felgenhauer

March 17, 2025

**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

# 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* $R \equiv$ *irrefl* $R \wedge$ *trans* $R$

**lemma** *strict-order-strict*: *strict-order* $q \Longrightarrow$ *strict* $(\lambda a\ b.\ (a,\ b) \in q^=) = (\lambda a\ b.\ (a,\ b) \in q)$
⟨*proof*⟩

**lemma** *mono-lex1*: *mono* $(\lambda r.$ *lex-prod* $r\ s)$
⟨*proof*⟩

**lemma** *mono-lex2*: *mono* (*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* :: $'a$ *set* $\Rightarrow$ *bool* **where**

$set\text{-}chain\ C \equiv \forall x \in C.\ \forall y \in C.\ x \leq y \lor y \leq x$

**lemma** *lfp-chain-induct*:
**assumes** *mono*: *mono f*
**and** *step*: $\bigwedge x.\ P\ x \Longrightarrow P\ (f\ x)$
**and** *chain*: $\bigwedge C.\ set\text{-}chain\ C \Longrightarrow \forall\ x \in C.\ P\ x \Longrightarrow P\ (Sup\ C)$
**shows** $P\ (lfp\ f)$
⟨*proof*⟩

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

**lemma** *trans-Sup-of-chain*:
**assumes** *set-chain C* **and** *trans*: $\bigwedge R.\ R \in C \Longrightarrow trans\ R$
**shows** *trans* (*Sup C*)
⟨*proof*⟩

**lemma** *asym-Sup-of-chain*:
**assumes** *set-chain C* **and** *asym*: $\bigwedge\ R.\ R \in C \Longrightarrow asym\ R$
**shows** *asym* (*Sup C*)
⟨*proof*⟩

**lemma** *strict-order-lfp*:
**assumes** *mono f* **and** $\bigwedge R.\ strict\text{-}order\ R \Longrightarrow strict\text{-}order\ (f\ R)$
**shows** *strict-order* (*lfp f*)
⟨*proof*⟩

**lemma** *trans-lfp*:
**assumes** *mono f* **and** $\bigwedge R.\ trans\ R \Longrightarrow trans\ (f\ R)$
**shows** *trans* (*lfp f*)
⟨*proof*⟩

**end**

## 1.3 Multiset extension

**lemma** *mulex-iff-mult*: $mulex\ r\ M\ N \longleftrightarrow (M,N) \in mult\ \{(M,N)\ .\ r\ M\ N\}$
⟨*proof*⟩

**lemma** *multI*:
**assumes** $trans\ r\ M = I + K\ N = I + J\ J \neq \{\#\}\ \forall k \in set\text{-}mset\ K.\ \exists j \in set\text{-}mset\ J.\ (k,j) \in r$
**shows** $(M,N) \in mult\ r$
⟨*proof*⟩

**lemma** *multE*:
**assumes** *trans r* **and** $(M,N) \in mult\ r$
**obtains** *I J K* **where** $M = I + K\ N = I + J\ J \neq \{\#\}\ \forall k \in set\text{-}mset\ K.\ \exists j \in set\text{-}mset\ J.\ (k,j) \in r$
⟨*proof*⟩

**lemma** *mult-on-union*: $(M,N) \in mult\ r \Longrightarrow (K + M,\ K + N) \in mult\ r$
⟨*proof*⟩

**lemma** *mult-on-union′*: $(M,N) \in mult\ r \Longrightarrow (M + K,\ N + K) \in mult\ r$
⟨*proof*⟩

**lemma** *mult-on-add-mset*: $(M,N) \in mult\ r \Longrightarrow (add\text{-}mset\ k\ M,\ add\text{-}mset\ k\ N) \in mult\ r$
⟨*proof*⟩

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

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

**lemma** *empty-mult*[*simp*]: $(\{\#\},N) \in mult\ R \longleftrightarrow N \neq \{\#\}$
⟨*proof*⟩

**lemma** *trans-mult*: *trans* (*mult R*)
⟨*proof*⟩

**lemma** *strict-order-mult*:
  **assumes** *irrefl R* **and** *trans R*
  **shows** *irrefl* (*mult R*) **and** *trans* (*mult R*)
⟨*proof*⟩

**lemma** *mult-of-image-mset*:
  **assumes** *trans R* **and** *trans R′*
  **and** $\bigwedge x\ y.\ x \in set\text{-}mset\ N \Longrightarrow y \in set\text{-}mset\ M \Longrightarrow (x,y) \in R \Longrightarrow (f\ x,\ f\ y) \in R'$
  **and** $(N,\ M) \in mult\ R$
  **shows** $(image\text{-}mset\ f\ N,\ image\text{-}mset\ f\ M) \in mult\ R'$
⟨*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*:
  $wf\ p \longleftrightarrow wfp\text{-}on\ (\lambda a\ b.\ (a,\ b) \in p)\ UNIV$
⟨*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* $(x\ \#\ xs) = ([],\ x,\ xs)\ \#\ map\ (\lambda(xs,\ x',\ xs').\ (x\ \#\ xs,\ x',\ xs'))\ (list\text{-}splits\ xs)$

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

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

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

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

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

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

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

**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*:
  $adj\text{-}msog\ xs\ zs\ (x,[]) = (x,\ (case\ fst\ x\ of\ Grave \Rightarrow xs \mid Acute \Rightarrow zs \mid Macron \Rightarrow [] ))$
⟨*proof*⟩

**lemma** *ms-of-greek-elem*:
  **assumes** $(x,xs) \in set\text{-}mset\ (ms\text{-}of\text{-}greek\ ys)$
  **shows** $x \in set\ ys$
⟨*proof*⟩

**lemma** *ms-of-greek-shorter*:
  **assumes** $(x,\ t) \in\#\ ms\text{-}of\text{-}greek\ s$
  **shows** $length\ s > length\ t$
⟨*proof*⟩

**lemma** *msog-append*: $ms\text{-}of\text{-}greek\ (xs\ @\ ys) = image\text{-}mset\ (adj\text{-}msog\ []\ ys)\ (ms\text{-}of\text{-}greek\ xs) +$
  $image\text{-}mset\ (adj\text{-}msog\ xs\ [])\ (ms\text{-}of\text{-}greek\ ys)$
⟨*proof*⟩

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

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

**lemma** *greek-less-unfold*:
  $greek\text{-}less\ r = nest\ r\ (greek\text{-}less\ r)$
⟨*proof*⟩

## 2.3 Preservation of strict partial orders

**lemma** *strict-order-letter-less*:
  **assumes** $strict\text{-}order\ r$
  **shows** $strict\text{-}order\ (letter\text{-}less\ r)$
⟨*proof*⟩

**lemma** *strict-order-nest*:
  **assumes** $r$: $strict\text{-}order\ r$ **and** $R$: $strict\text{-}order\ R$
  **shows** $strict\text{-}order\ (nest\ r\ R)$
⟨*proof*⟩

**lemma** *strict-order-greek-less*:
  **assumes** *strict-order r*
  **shows** *strict-order* (*greek-less r*)
⟨*proof*⟩

**lemma** *trans-letter-less*:
  **assumes** *trans r*
  **shows** *trans* (*letter-less r*)
⟨*proof*⟩

**lemma** *trans-order-nest*: *trans* (*nest r R*)
⟨*proof*⟩

**lemma** *trans-greek-less*[*simp*]: *trans* (*greek-less r*)
⟨*proof*⟩

**lemma** *mono-greek-less*: *mono greek-less*
⟨*proof*⟩

## 2.4 Involution

**definition** *inv-letter* :: *'a letter* ⇒ *'a letter* **where**
  *inv-letter l* ≡
    *case l of* (*a*, *x*) ⇒ (*case a of Grave* ⇒ *Acute* | *Acute* ⇒ *Grave* | *Macron* ⇒ *Macron*, *x*)

**lemma** *inv-letter-pair*[*simp*]:
  *inv-letter* (*a*, *x*) = (*case a of Grave* ⇒ *Acute* | *Acute* ⇒ *Grave* | *Macron* ⇒ *Macron*, *x*)
⟨*proof*⟩

**lemma** *snd-inv-letter*[*simp*]:
  *snd* (*inv-letter x*) = *snd x*
⟨*proof*⟩

**lemma** *inv-letter-invol*[*simp*]:
  *inv-letter* (*inv-letter x*) = *x*
⟨*proof*⟩

**lemma** *inv-letter-mono*[*simp*]:
  **assumes** (*x*, *y*) ∈ *letter-less r*
  **shows** (*inv-letter x*, *inv-letter y*) ∈ *letter-less r*
⟨*proof*⟩

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

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

*inv-greek* (*inv-greek* $s$) = $s$
⟨*proof*⟩

**lemma** *inv-greek-append*:
*inv-greek* ($s$ @ $t$) = *inv-greek* $t$ @ *inv-greek* $s$
⟨*proof*⟩

**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* $M$) = $M$
⟨*proof*⟩

**lemma** *ms-of-greek-inv-greek*:
*ms-of-greek* (*inv-greek* $M$) = *inv-msog* (*ms-of-greek* $M$)
⟨*proof*⟩

**lemma** *inv-greek-mono*:
**assumes** *trans* $r$ **and** ($s$, $t$) $\in$ *greek-less* $r$
**shows** (*inv-greek* $s$, *inv-greek* $t$) $\in$ *greek-less* $r$
⟨*proof*⟩

## 2.5 Monotonicity of *greek-less r*

**lemma** *greek-less-rempty*[*simp*]:
($a$,[]) $\in$ *greek-less* $r$ $\longleftrightarrow$ *False*
⟨*proof*⟩

**lemma** *greek-less-nonempty*:
**assumes** $b \neq$ []
**shows** ($a$,$b$) $\in$ *greek-less* $r$ $\longleftrightarrow$ ($a$,$b$) $\in$ *nest* $r$ (*greek-less* $r$)
⟨*proof*⟩

**lemma** *greek-less-lempty*[*simp*]:
([],$b$) $\in$ *greek-less* $r$ $\longleftrightarrow$ $b \neq$ []
⟨*proof*⟩

**lemma** *greek-less-singleton*:
($a$, $b$) $\in$ *letter-less* $r$ $\Longrightarrow$ ([$a$], [$b$]) $\in$ *greek-less* $r$
⟨*proof*⟩

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

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

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

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

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

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

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

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

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

## 2.7 Basic Comparisons

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

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

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

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

**lemma** *rcliff-greek-less2*:
  **assumes** *trans r*
  **and** *as*: *snd ‘ set as* $\subseteq$ *under r a* **and** $b'$: $b' \in \{[(Grave,b)],[]\}$
  **and** *cs*: *snd ‘ set cs* $\subseteq$ *under r a* $\cup$ *under r b*
  **shows** (*as* @ $b'$ @ *cs*, $[(Macron,a),(Grave,b)]$) $\in$ *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** $b'$: $b' = [(Macron,b)]$
  **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*, $[(Acute,a),(Macron,b)]$) $\in$ *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** $a'$: $a' \in \{[(Acute,a)],[]\}$
  **and** *bs*: *snd ‘ set bs* $\subseteq$ *under r b*
  **shows** (*cs* @ $a'$ @ *bs*, $[(Acute,a),(Macron,b)]$) $\in$ *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* $\Rightarrow$ $(L\ i)^{-1}$ | *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 ‘ set cs* $\subseteq$ *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*: $\bigwedge a\ b\ s\ t\ u.\ (s,\ t) \in L\ a \Longrightarrow (s,\ u) \in R\ b \Longrightarrow \exists xs.\ lpeak\ r\ a\ b\ xs \wedge (t,\ u) \in lconv\ xs$
**and** *lc*: $\bigwedge a\ b\ s\ t\ u.\ (s,\ t) \in L\ a \Longrightarrow (s,\ u) \in E\ b \Longrightarrow \exists xs.\ lcliff\ r\ a\ b\ xs \wedge (t,\ u) \in lconv\ xs$
**and** *rc*: $\bigwedge a\ b\ s\ t\ u.\ (s,\ t) \in (E\ a)^{-1} \Longrightarrow (s,\ u) \in R\ b \Longrightarrow \exists xs.\ rcliff\ r\ a\ b\ xs \wedge (t,\ u) \in lconv\ xs$
**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*: $\bigwedge a\ b\ s\ t\ u.\ (s,\ t) \in L\ a \Longrightarrow (s,\ u) \in R\ b \Longrightarrow$
$(t,\ u) \in conversion'\ (under\ r\ a)\ O\ (R\ b)^{=}\ O\ conversion'\ (under\ r\ a \cup under\ r\ b)\ O$
$((L\ a)^{-1})^{=}\ O\ conversion'\ (under\ r\ b)$
**and** *lc*: $\bigwedge a\ b\ s\ t\ u.\ (s,\ t) \in L\ a \Longrightarrow (s,\ u) \in E\ b \Longrightarrow$
$(t,\ u) \in conversion'\ (under\ r\ a)\ O\ E\ b\ O\ conversion'\ (under\ r\ a)\ O$
$((L\ a)^{-1})^{=}\ O\ conversion'\ (under\ r\ a \cap under\ r\ b) \vee$
$(t,\ u) \in conversion'\ (under\ r\ a \cup under\ r\ b)\ O\ ((L\ a\ )^{-1})^{=}\ O\ conversion'\ (under\ r\ b)$
**and** *rc*: $\bigwedge a\ b\ s\ t\ u.\ (s,\ t) \in (E\ a)^{-1} \Longrightarrow (s,\ u) \in R\ b \Longrightarrow$
$(t,\ u) \in conversion'\ (under\ r\ a \cap under\ r\ b)\ O\ (R\ b)^{=}\ O\ conversion'\ (under\ r\ b)\ O$
$E\ a\ O\ conversion'\ (under\ r\ b) \vee$
$(t,\ u) \in conversion'\ (under\ r\ a)\ O\ (R\ b)^{=}\ O\ conversion'\ (under\ r\ a \cup under\ r\ b)$
**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*: $\bigwedge i.\ sym\ (E\ i)$
**and** *pk*: $\bigwedge a\ b\ s\ t\ u.\ (s,\ t) \in L\ a \Longrightarrow (s,\ u) \in L\ b \Longrightarrow$
$(t,\ u) \in conversion'\ L\ L\ E\ (under\ r\ a)\ O\ (L\ b)^{=}\ O\ conversion'\ L\ L\ E\ (under\ r\ a \cup under\ r\ b)\ O$
$((L\ a)^{-1})^{=}\ O\ conversion'\ L\ L\ E\ (under\ r\ b)$
**and** *cl*: $\bigwedge a\ b\ s\ t\ u.\ (s,\ t) \in L\ a \Longrightarrow (s,\ u) \in E\ b \Longrightarrow$

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

## 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*: $\bigwedge a \ b \ s \ t \ u. \ (s, t) \in L \ a \Longrightarrow (s, u) \in R \ b \Longrightarrow$
$(t, u) \in conversion'' \ L \ R \ (under \ r \ a) \ O \ (R \ b)^{=} \ O \ conversion'' \ L \ R \ (under \ r \ a \cup under \ r \ b) \ O$
$((L \ a)^{-1})^{=} \ O \ conversion'' \ L \ R \ (under \ r \ b)$
**shows** $commute \ (\bigcup i. \ L \ i) \ (\bigcup i. \ R \ i)$
⟨*proof*⟩

Decreasing diagrams for confluence.

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

## 3.3 Extended decreasing diagrams

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

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

**private lemma** *Union-down*: $(\bigcup i. \ down \ L \ i) = (\bigcup i. \ L \ i)$
⟨*proof*⟩

Extended decreasing diagrams for commutation.

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

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.