Combinatorics on Words formalized Graph Lemma

Štěpán Holub
Štěpán Starosta

March 17, 2025

Funded by the Czech Science Foundation grant GAČR 20-20621S.

Contents

theory *Glued-Codes*
imports *Combinatorics-Words.Submonoids*
begin

Chapter 1

Glued codes

1.1 Lists that do not end with a fixed letter

lemma *append-last-neq*:
$us = \varepsilon \vee last\ us \neq w \Longrightarrow vs = \varepsilon \vee last\ vs \neq w \Longrightarrow us \cdot vs = \varepsilon \vee last\ (us \cdot vs) \neq w$
⟨*proof*⟩

lemma *last-neq-induct* [*consumes 1*, *case-names emp hd-eq hd-neq*]:
assumes *invariant*: $us = \varepsilon \vee last\ us \neq w$
and *emp*: $P\ \varepsilon$
and *hd-eq*: $\bigwedge us.\ us \neq \varepsilon \Longrightarrow last\ us \neq w \Longrightarrow P\ us \Longrightarrow P\ (w \#\ us)$
and *hd-neq*: $\bigwedge u\ us.\ u \neq w \Longrightarrow us = \varepsilon \vee last\ us \neq w \Longrightarrow P\ us \Longrightarrow P\ (u \#\ us)$
shows $P\ us$
⟨*proof*⟩

lemma *last-neq-blockE*:
assumes *last-neq*: $us \neq \varepsilon$ **and** $last\ us \neq w$
obtains $k\ u\ us'$ **where** $u \neq w$ **and** $us' = \varepsilon \vee last\ us' \neq w$ **and** $[w]\ ^{@}\ k \cdot u \#\ us' = us$
⟨*proof*⟩

lemma *last-neq-block-induct* [*consumes 1*, *case-names emp block*]:
assumes *last-neq*: $us = \varepsilon \vee last\ us \neq w$
and *emp*: $P\ \varepsilon$
and *block*: $\bigwedge k\ u\ us.\ u \neq w \Longrightarrow us = \varepsilon \vee last\ us \neq w \Longrightarrow P\ us \Longrightarrow P\ ([w]\ ^{@}\ k \cdot (u \#\ us))$
shows $P\ us$
⟨*proof*⟩

1.2 Glue a list element with its successors/predecessors

function *glue* :: *'a list* ⇒ *'a list list* ⇒ *'a list list* **where**
glue-emp: *glue w ε* = *ε* |
glue-Cons: *glue w* (*u* # *us*) =
(*let glue-tl* = *glue w us in*
if u = *w then* (*u* · *hd glue-tl*) # *tl glue-tl*
else u # *glue-tl*)
⟨*proof*⟩
termination ⟨*proof*⟩

lemma *no-gluing*: *w* ∉ *set us* ⟹ *glue w us* = *us*
⟨*proof*⟩

lemma *glue-nemp* [*simp*, *intro*!]: *us* ≠ *ε* ⟹ *glue w us* ≠ *ε*
⟨*proof*⟩

lemma *glue-is-emp-iff* [*simp*]: *glue w us* = *ε* ⟷ *us* = *ε*
⟨*proof*⟩

lemma *len-glue*: *us* = *ε* ∨ *last us* ≠ *w* ⟹ |*glue w us*| + *count-list us w* = |*us*|
⟨*proof*⟩

lemma *len-glue-le*: **assumes** *us* = *ε* ∨ *last us* ≠ *w* **shows** |*glue w us*| ≤ |*us*|
⟨*proof*⟩

lemma *len-glue-less* []: *us* = *ε* ∨ *last us* ≠ *w* ⟹ *w* ∈ *set us* ⟹ |*glue w us*| < |*us*|
⟨*proof*⟩

lemma assumes *us* = *ε* ∨ *last us* ≠ *w* **and** *ε* ∉ *set us*
shows *emp-not-in-glue*: *ε* ∉ *set* (*glue w us*)
and *glued-not-in-glue*: *w* ∉ *set* (*glue w us*)
⟨*proof*⟩

lemma *glue-glue*: *us* = *ε* ∨ *last us* ≠ *w* ⟹ *ε* ∉ *set us* ⟹ *glue w* (*glue w us*) = *glue w us*
⟨*proof*⟩

lemma *glue-block-append*: **assumes** *u* ≠ *w*
shows *glue w* ([*w*] ^@ *k* · (*u* # *us*)) = (*w* ^@ *k* · *u*) # *glue w us*
⟨*proof*⟩

lemma *concat-glue* [*simp*]: *us* = *ε* ∨ *last us* ≠ *w* ⟹ *concat* (*glue w us*) = *concat us*
⟨*proof*⟩

lemma *glue-append*:
us = *ε* ∨ *last us* ≠ *w* ⟹ *glue w* (*us* · *vs*) = *glue w us* · *glue w vs*

⟨*proof*⟩

lemma *glue-pow*:
assumes $us = \varepsilon \vee last\ us \neq w$
shows $glue\ w\ (us\ ^{@}\ k) = (glue\ w\ us)\ ^{@}\ k$
⟨*proof*⟩

lemma *glue-in-lists-hull* [*intro*]:
$us = \varepsilon \vee last\ us \neq w \Longrightarrow us \in lists\ G \Longrightarrow glue\ w\ us \in lists\ \langle G \rangle$
⟨*proof*⟩
function *gluer* :: *'a list* ⇒ *'a list list* ⇒ *'a list list* **where**
gluer-emp: $gluer\ w\ \varepsilon = \varepsilon$ |
gluer-Cons: $gluer\ w\ (u\ \#\ us) =$
$(let\ gluer\text{-}butlast = gluer\ w\ (butlast\ (u\ \#\ us))\ in$
$if\ last\ (u\ \#\ us) = w\ then\ (butlast\ gluer\text{-}butlast) \cdot [last\ gluer\text{-}butlast \cdot last\ (u\ \#\ us)]$
$else\ gluer\text{-}butlast \cdot [last\ (u\ \#\ us)])$
⟨*proof*⟩
termination ⟨*proof*⟩

lemma *gluer-nemp-def*: **assumes** $us \neq \varepsilon$
shows $gluer\ w\ us =$
$(let\ gluer\text{-}butlast = gluer\ w\ (butlast\ us)\ in$
$if\ last\ us = w\ then\ (butlast\ gluer\text{-}butlast) \cdot [last\ gluer\text{-}butlast \cdot last\ us]$
$else\ gluer\text{-}butlast \cdot [last\ us])$
⟨*proof*⟩

lemma *gluer-nemp*: **assumes** $us \neq \varepsilon$ **shows** $gluer\ w\ us \neq \varepsilon$
⟨*proof*⟩

lemma *hd-neq-induct* [*consumes 1*, *case-names emp snoc-eq snoc-neq*]:
assumes *invariant*: $us = \varepsilon \vee hd\ us \neq w$
and *emp*: $P\ \varepsilon$
and *snoc-eq*: $\bigwedge us.\ us \neq \varepsilon \Longrightarrow hd\ us \neq w \Longrightarrow P\ us \Longrightarrow P\ (us \cdot [w])$
and *snoc-neq*: $\bigwedge u\ us.\ u \neq w \Longrightarrow us = \varepsilon \vee hd\ us \neq w \Longrightarrow P\ us \Longrightarrow P\ (us \cdot [u])$
shows $P\ us$
⟨*proof*⟩

lemma *gluer-rev* [*reversal-rule*]: **assumes** $us = \varepsilon \vee last\ us \neq w$
shows $gluer\ (rev\ w)\ (rev\ (map\ rev\ us)) = rev\ (map\ rev\ (glue\ w\ us))$
⟨*proof*⟩

lemma *glue-rev* [*reversal-rule*]: **assumes** $us = \varepsilon \vee hd\ us \neq w$
shows $glue\ (rev\ w)\ (rev\ (map\ rev\ us)) = rev\ (map\ rev\ (gluer\ w\ us))$
⟨*proof*⟩

1.3 Generators with glued element

The following set will turn out to be the generating set of all words whose decomposition into a generating code does not end with w

inductive-set *glued-gens* :: *'a list* ⇒ *'a list set* ⇒ *'a list set*
  for *w G* **where**
    other-gen: $g \in G \Longrightarrow g \neq w \Longrightarrow g \in$ *glued-gens w G*
  | *glued* [*intro*!]: $u \in$ *glued-gens w G* $\Longrightarrow$ $w \cdot u \in$ *glued-gens w G*

lemma *in-glued-gensI*: **assumes** $g \in G$ $g \neq w$
  shows $w \,^{@}\, k \cdot g = u \Longrightarrow u \in$ *glued-gens w G*
  ⟨*proof*⟩

lemma *in-glued-gensE*:
  assumes $u \in$ *glued-gens w G*
  obtains *k g* **where** $g \in G$ **and** $g \neq w$ **and** $w \,^{@}\, k \cdot g = u$
⟨*proof*⟩

lemma *glued-gens-alt-def*: *glued-gens w C* = $\{w \,^{@}\, k \cdot g \mid k\ g.\ g \in C \wedge g \neq w\}$
  ⟨*proof*⟩

lemma *glued-hull-sub-hull* [*simp*, *intro*!]: $w \in G \Longrightarrow \langle$*glued-gens w G*$\rangle \subseteq \langle G \rangle$
  ⟨*proof*⟩

lemma *glued-hull-sub-hull'*: $w \in G \Longrightarrow u \in \langle$*glued-gens w G*$\rangle \Longrightarrow u \in \langle G \rangle$
  ⟨*proof*⟩

lemma *in-glued-hullE*:
  assumes $w \in G$ **and** $u \in \langle$*glued-gens w G*$\rangle$
  obtains *us* **where** *concat us* = *u* **and** *us* ∈ *lists G* **and** $us = \varepsilon \vee$ *last us* $\neq w$
⟨*proof*⟩

lemma *glue-in-lists* [*simp*, *intro*!]:
  assumes $us = \varepsilon \vee$ *last us* $\neq w$
  shows *us* ∈ *lists G* $\Longrightarrow$ *glue w us* ∈ *lists* (*glued-gens w G*)
  ⟨*proof*⟩

lemma *concat-in-glued-hull*[*intro*]:
  us ∈ *lists G* $\Longrightarrow$ $us = \varepsilon \vee$ *last us* $\neq w \Longrightarrow$ *concat us* ∈ ⟨*glued-gens w G*⟩
  ⟨*proof*⟩

lemma *glued-hull-conv*: **assumes** $w \in G$
  shows ⟨*glued-gens w G*⟩ = {*concat us* | *us*. *us* ∈ *lists G* ∧ ($us = \varepsilon \vee$ *last us* $\neq w$)}
  ⟨*proof*⟩

1.4 Bounded gluing

lemma *bounded-glue-in-lists*:
assumes $us = \varepsilon \vee last\ us \neq w$ **and** $\neg\ [w]\ ^{@}\ n \leq_f us$
shows $us \in lists\ G \Longrightarrow glue\ w\ us \in lists\ \{w\ ^{@}\ k \cdot g \mid k\ g.\ g \in G \wedge g \neq w \wedge k < n\}$
⟨*proof*⟩

1.4.1 Gluing on binary alphabet

lemma *bounded-bin-glue-in-lists*: — meaning: a binary code
assumes $us = \varepsilon \vee last\ us \neq x$
and $\neg\ [x]\ ^{@}\ n \leq_f us$
and $us \in lists\ \{x, y\}$
shows $glue\ x\ us \in lists\ \{x\ ^{@}\ k \cdot y \mid k.\ k < n\}$
⟨*proof*⟩

lemma *single-bin-glue-in-lists*: — meaning: a single occurrence
assumes $us = \varepsilon \vee last\ us \neq x$
and $\neg\ [x,x] \leq_f us$
and $us \in lists\ \{x, y\}$
shows $glue\ x\ us \in lists\ \{x \cdot y, y\}$
⟨*proof*⟩

lemma *count-list-single-bin-glue*:
assumes $x \neq \varepsilon$ **and** $x \neq y$
and $us = \varepsilon \vee last\ us \neq x$
and $us \in lists\ \{x,y\}$
and $\neg\ [x,x] \leq_f us$
shows $count\text{-}list\ (glue\ x\ us)\ (x \cdot y) = count\text{-}list\ us\ x$
and $count\text{-}list\ (glue\ x\ us)\ y + count\text{-}list\ us\ x = count\text{-}list\ us\ y$
⟨*proof*⟩

1.5 Code with glued element

context *code*
begin

If the original generating set is a code, then also the glued generators form a code

lemma *glued-hull-last-dec*: **assumes** $w \in \mathcal{C}$ **and** $u \in \langle glued\text{-}gens\ w\ \mathcal{C}\rangle$ **and** $u \neq \varepsilon$
shows $last\ (Dec\ \mathcal{C}\ u) \neq w$
⟨*proof*⟩

lemma *in-glued-hullI* [*intro*]:
assumes $u \in \langle\mathcal{C}\rangle$ **and** $(u = \varepsilon \vee last\ (Dec\ \mathcal{C}\ u) \neq w)$
shows $u \in \langle glued\text{-}gens\ w\ \mathcal{C}\rangle$
⟨*proof*⟩

lemma *code-glued-hull-conv*: **assumes** $w \in \mathcal{C}$
shows $\langle$*glued-gens* w $\mathcal{C}\rangle = \{u \in \langle\mathcal{C}\rangle.\ u = \varepsilon \vee$ *last* (*Dec* $\mathcal{C}$ $u) \neq w\}$
⟨*proof*⟩

lemma *in-glued-hull-iff*:
assumes $w \in \mathcal{C}$ **and** $u \in \langle\mathcal{C}\rangle$
shows $u \in \langle$*glued-gens* w $\mathcal{C}\rangle \longleftrightarrow u = \varepsilon \vee$ *last* (*Dec* $\mathcal{C}$ $u) \neq w$
⟨*proof*⟩

lemma *glued-not-in-glued-hull*: $w \in \mathcal{C} \Longrightarrow w \notin \langle$*glued-gens* w $\mathcal{C}\rangle$
⟨*proof*⟩

lemma *glued-gens-nemp*: **assumes** $u \in$ *glued-gens* w $\mathcal{C}$ **shows** $u \neq \varepsilon$
⟨*proof*⟩

lemma *glued-gens-code*: **assumes** $w \in \mathcal{C}$ **shows** *code* (*glued-gens* w $\mathcal{C}$)
⟨*proof*⟩

A crucial lemma showing the relation between gluing and the decomposition into generators

lemma *dec-glued-gens*: **assumes** $w \in \mathcal{C}$ **and** $u \in \langle$*glued-gens* w $\mathcal{C}\rangle$
shows *Dec* (*glued-gens* w $\mathcal{C}$) u = *glue* w (*Dec* $\mathcal{C}$ u)
⟨*proof*⟩

lemma *ref-glue*: $us = \varepsilon \vee$ *last* $us \neq w \Longrightarrow us \in$ *lists* $\mathcal{C} \Longrightarrow$ *Ref* $\mathcal{C}$ (*glue* w us) = us
⟨*proof*⟩

end
theorem *glued-code*:
assumes *code* C **and** $w \in C$
shows *code* $\{w \ ^{@}\ k \cdot u \mid k\ u.\ u \in C \wedge u \neq w\}$
⟨*proof*⟩

1.6 Gluing is primitivity preserving

It is easy to obtain that gluing lists of code elements preserves primitivity. We provide the result under weaker condition where glue blocks of the list have unique concatenation.

lemma (**in** *code*) *code-prim-glue*:
assumes *last-neq*: $us = \varepsilon \vee$ *last* $us \neq w$
and $us \in$ *lists* $\mathcal{C}$
shows *primitive* $us \Longrightarrow$ *primitive* (*glue* w us)
⟨*proof*⟩

definition *glue-block* :: *'a list* ⇒*'a list list* ⇒ *'a list list* ⇒ *bool*
where *glue-block* w us bs =
($\exists$ *ps k u ss*. (*ps* = $\varepsilon \vee$ *last ps* ≠ w) ∧ $u \neq w \wedge ps \cdot [w]\ ^{@}\ k \cdot u \#\ ss = us \wedge [w]\ ^{@}\ k \cdot [u] = bs$)

lemma *glue-blockI* [*intro*]:
ps = *ε* ∨ *last ps* ≠ *w* ⟹ *u* ≠ *w* ⟹ *ps* · [*w*] @ *k* · *u* # *ss* = *us* ⟹ [*w*] @ *k* · [*u*] = *bs*
⟹ *glue-block w us bs*
⟨*proof*⟩

lemma *glue-blockE*:
assumes *glue-block w us bs*
obtains *ps k u ss* **where** *ps* = *ε* ∨ *last ps* ≠ *w* **and** *u* ≠ *w ps* · [*w*] @ *k* · *u* # *ss* = *us*
and [*w*] @ *k* · [*u*] = *bs*
⟨*proof*⟩

lemma assumes *glue-block w us bs*
shows *glue-block-of-appendL*: *glue-block w* (*us* · *vs*) *bs*
and *glue-block-of-appendR*: *vs* = *ε* ∨ *last vs* ≠ *w* ⟹ *glue-block w* (*vs* · *us*) *bs*
⟨*proof*⟩

lemma *glue-block-of-block-append*:
u ≠ *w* ⟹ *glue-block w us bs* ⟹ *glue-block w* ([*w*] @ *k* · *u* # *us*) *bs*
⟨*proof*⟩

lemma *in-set-glueE*:
assumes *last-neq*: *us* = *ε* ∨ *last us* ≠ *w*
and *b* ∈ *set* (*glue w us*)
obtains *bs* **where** *glue-block w us bs* **and** *concat bs* = *b*
⟨*proof*⟩

definition *unglue* :: *'a list* ⇒ *'a list list* ⇒ *'a list* ⇒ *'a list list*
where *unglue w us b* = (*THE bs*. *glue-block w us bs* ∧ *concat bs* = *b*)

lemma *unglueI*:
assumes *unique-blocks*: ⋀*bs*$_1$ *bs*$_2$. *glue-block w us bs*$_1$ ⟹ *glue-block w us bs*$_2$
⟹ *concat bs*$_1$ = *concat bs*$_2$ ⟹ *bs*$_1$ = *bs*$_2$
shows *glue-block w us bs* ⟹ *concat bs* = *b* ⟹ *unglue w us b* = *bs*
⟨*proof*⟩

lemma *concat-map-unglue-glue*:
assumes *last-neq*: *us* = *ε* ∨ *last us* ≠ *w*
and *unique-blocks*: ⋀*vs*$_1$ *vs*$_2$. *glue-block w us vs*$_1$ ⟹ *glue-block w us vs*$_2$
⟹ *concat vs*$_1$ = *concat vs*$_2$ ⟹ *vs*$_1$ = *vs*$_2$
shows *concat* (*map* (*unglue w us*) (*glue w us*)) = *us*
⟨*proof*⟩

lemma *prim-glue*:
assumes *last-neq*: *us* = *ε* ∨ *last us* ≠ *w*
and *unique-blocks*: ⋀*bs*$_1$ *bs*$_2$. *glue-block w us bs*$_1$ ⟹ *glue-block w us bs*$_2$
⟹ *concat bs*$_1$ = *concat bs*$_2$ ⟹ *bs*$_1$ = *bs*$_2$

shows *primitive us* $\Longrightarrow$ *primitive* (*glue w us*)
⟨*proof*⟩

1.6.1 Gluing on binary alphabet

lemma *bin-glue-blockE*:
assumes *us* $\in$ *lists* $\{x, y\}$
and *glue-block x us bs*
obtains *k* **where** $[x]$ @ $k \cdot [y] = bs$
⟨*proof*⟩

lemma *unique-bin-glue-blocks*:
assumes *us* $\in$ *lists* $\{x, y\}$ **and** $x \neq \varepsilon$
shows *glue-block x us* $bs_1 \Longrightarrow$ *glue-block x us* $bs_2 \Longrightarrow$ *concat* bs_1 = *concat* bs_2 $\Longrightarrow$ $bs_1 = bs_2$
⟨*proof*⟩

lemma *prim-bin-glue*:
assumes *us* $\in$ *lists* $\{x, y\}$ **and** $x \neq \varepsilon$
and $us = \varepsilon \vee$ *last us* $\neq x$
shows *primitive us* $\Longrightarrow$ *primitive* (*glue x us*)
⟨*proof*⟩

end

theory *Graph-Lemma*
imports *Combinatorics-Words.Submonoids Glued-Codes*

begin

Chapter 2

Graph Lemma

The Graph Lemma is an important tool for gaining information about systems of word equations. It yields an upper bound on the rank of the solution, that is, on the number of factors into all images of unknowns can be factorized. The most straightforward application is showing that a system of equations admits periodic solutions only, which in particular holds for any nontrivial equation over two words.

The name refers to a graph whose vertices are the unknowns of the system, and edges connect front letters of the left- and right- hand sides of equations. The bound mentioned above is then the number of connected components of the graph.

We formalize the algebraic proof from [1]. Key ingredients of the proof are in the *Combinatorics-Words-Graph-Lemma.Glued-Codes*.

2.1 Graph lemma

theorem *graph-lemma-last*: $\mathfrak{B}_F\ G = \{last\ (Dec\ (\mathfrak{B}_F\ G)\ g) \mid g.\ g \in G \wedge g \neq \varepsilon\}$
⟨*proof*⟩

theorem *graph-lemma*: $\mathfrak{B}_F\ G = \{hd\ (Dec\ (\mathfrak{B}_F\ G)\ g) \mid g.\ g \in G \wedge g \neq \varepsilon\}$
⟨*proof*⟩

2.2 Binary code

We illustrate the use of the Graph Lemma in an alternative proof of the fact that two non-commuting words form a code. See also $[\![u_0 \cdot u_1 \neq u_1 \cdot u_0;\ us \in lists\ \{u_0,\ u_1\};\ vs \in lists\ \{u_0,\ u_1\};\ concat\ us = concat\ vs]\!] \Longrightarrow us = vs$ in *Combinatorics-Words.CoWBasic*.

First, we prove a lemma which is the core of the alternative proof.

lemma *non-comm-hds-neq*: **assumes** $u \cdot v \neq v \cdot u$ **shows** $hd\ (Dec\ \mathfrak{B}_F\ \{u,v\}\ u) \neq hd\ (Dec\ \mathfrak{B}_F\ \{u,v\}\ v)$

⟨*proof*⟩

theorem assumes $u \cdot v \neq v \cdot u$ **shows** *code* $\{u, v\}$
⟨*proof*⟩

end

References

[1] J. Berstel, D. Perrin, J. Perrot, and A. Restivo. Sur le théorème du défaut. *Journal of Algebra*, 60(1):169–180, 1979.