

# Dyck Language

Tobias Nipkow and Moritz Roos

February 6, 2026

## Abstract

The Dyck language over a pair of brackets, e.g. ( and ), is the set of balanced strings/words/lists of brackets. That is, the set of words with the same number of ( and ), where every prefix of the word contains no more ) than (. In general, a Dyck language is defined over a whole set of matching pairs of brackets.

## Contents

<b>1 Dyck Languages</b>	<b>1</b>
1.1 Balanced, Inductive and Recursive	1
1.2 Equivalence of <i>bal</i> and <i>bal_stk</i>	2
1.3 More properties of <i>bal</i> , using <i>bal_stk</i>	3
1.4 Dyck Language over an Alphabet	5

## 1 Dyck Languages

```
theory Dyck_Language
imports Main
begin
```

Dyck languages are sets of words/lists of balanced brackets. A bracket is a pair of type  $bool \times 'a$  where *True* is an opening and *False* a closing bracket. That is, brackets are tagged with elements of type *'a*.

```
type_synonym 'a bracket = bool  $\times$  'a
```

```
abbreviation Open a  $\equiv$  (True,a)
abbreviation Close a  $\equiv$  (False,a)
```

### 1.1 Balanced, Inductive and Recursive

Definition of what it means to be a *balanced* list of brackets:

```
inductive bal :: 'a bracket list  $\Rightarrow$  bool where
  bal [] |
  bal xs  $\Longrightarrow$  bal ys  $\Longrightarrow$  bal (xs @ ys) |
```

$bal\ xs \implies bal\ (Open\ a\ \# \ xs\ @\ [Close\ a])$

**declare**  $bal.intros(1)[iff]$   $bal.intros(2)[intro,simp]$   $bal.intros(3)[intro,simp]$

**lemma**  $bal2[iff]:\ bal\ [Open\ a,\ Close\ a]$   
**using**  $bal.intros(3)[of\ []]$  **by**  $simp$

The inductive definition of balanced is complemented with a functional version that uses a stack to remember which opening brackets need to be closed:

**fun**  $bal\_stk :: 'a\ list \Rightarrow 'a\ bracket\ list \Rightarrow 'a\ list * 'a\ bracket\ list$  **where**  
 $bal\_stk\ s\ [] = (s, []) \mid$   
 $bal\_stk\ s\ (Open\ a\ \# \ bs) = bal\_stk\ (a\ \# \ s)\ bs \mid$   
 $bal\_stk\ (a'\ \# \ s)\ (Close\ a\ \# \ bs) =$   
 $(if\ a = a'\ then\ bal\_stk\ s\ bs\ else\ (a'\ \# \ s,\ Close\ a\ \# \ bs)) \mid$   
 $bal\_stk\ bs\ stk = (bs, stk)$

**lemma**  $bal\_stk\_more\_stk: bal\_stk\ s1\ xs = (s1', []) \implies bal\_stk\ (s1@s2)\ xs =$   
 $(s1'@s2, [])$   
**by**( $induction\ s1\ xs\ arbitrary: s2\ rule: bal\_stk.induct$ ) ( $auto\ split: if\_splits$ )

**lemma**  $bal\_stk\_if\_Nils[simp]: ASSUMPTION(bal\_stk\ []\ bs = ([], [])) \implies bal\_stk$   
 $s\ bs = (s, [])$   
**unfolding**  $ASSUMPTION\_def$  **using**  $bal\_stk\_more\_stk[of\ []\ \_ \ []]$  **by**  $simp$

**lemma**  $bal\_stk\_append:$   
 $bal\_stk\ s\ (xs\ @\ ys)$   
 $= (let\ (s', xs') = bal\_stk\ s\ xs\ in\ if\ xs' = []\ then\ bal\_stk\ s'\ ys\ else\ (s', xs'\ @\ ys))$   
**by**( $induction\ s\ xs\ rule: bal\_stk.induct$ ) ( $auto\ split: if\_splits$ )

**lemma**  $bal\_stk\_append\_if:$   
 $bal\_stk\ s1\ xs = (s2, []) \implies bal\_stk\ s1\ (xs\ @\ ys) = bal\_stk\ s2\ ys$   
**by**( $simp\ add: bal\_stk\_append[of\ \_ \ xs]$ )

**lemma**  $bal\_stk\_split:$   
 $bal\_stk\ s\ xs = (s', xs') \implies \exists\ us.\ xs = us@xs' \wedge bal\_stk\ s\ us = (s', [])$   
**by**( $induction\ s\ xs\ rule: bal\_stk.induct$ ) ( $auto\ split: if\_splits$ )

## 1.2 Equivalence of $bal$ and $bal\_stk$

**lemma**  $bal\_stk\_if\_bal: bal\ xs \implies bal\_stk\ s\ xs = (s, [])$   
**by**( $induction\ arbitrary: s\ rule: bal.induct$ )( $auto\ simp: bal\_stk\_append\_if\ split: if\_splits$ )

**lemma**  $bal\_insert\_AB:$   
 $bal\ (v\ @\ w) \implies bal\ (v\ @\ (Open\ a\ \# \ Close\ a\ \# \ w))$   
**proof**( $induction\ v\ @\ w\ arbitrary: v\ w\ rule: bal.induct$ )  
**case** 1 **thus**  $?case$  **by**  $blast$   
**next**  
**case** ( $\exists\ u\ b$ )

```

then show ?case
proof (cases v)
  case Nil
    hence  $w = \text{Open } b \# u @ [\text{Close } b]$ 
      using 3.hyps(3) by fastforce
    hence bal w using 3.hyps
      by blast
    hence bal ([Open a, Close a] @ w)
      by blast
    thus ?thesis using Nil by simp
  next
    case [simp]: (Cons x v')
    show ?thesis
    proof (cases w rule:rev_cases)
      case Nil
        from 3.hyps have bal ((Open a # u @ [Close a]) @ [Open a, Close a])
          using bal.intros(2) by blast
        thus ?thesis using Nil Cons 3
          by (metis append_Nil append_Nil2 bal.simps)
      next
        case (snoc w' y)
        thus ?thesis
          using 3.hyps(2,3) bal.intros(3) by force
    qed
  qed
next
  case (2 v' w')
  then obtain r where  $v' = v @ r \wedge r @ w' = w \vee v' @ r = v \wedge w' = r @ w$ 
    by (meson append_eq_append_conv2)
  thus ?case
    using 2.hyps bal.intros(2) by force
qed

lemma bal_if_bal_stk: bal_stk s w = ([],[])  $\implies$  bal (rev(map ( $\lambda x.$  Open x) s) @
w)
proof(induction s w rule: bal_stk.induct)
  case 2
    then show ?case by simp
  next
    case 3
    then show ?case by (auto simp add: bal_insert_AB split: if_splits)
qed (auto)

corollary bal_iff_bal_stk: bal w  $\longleftrightarrow$  bal_stk [] w = ([],[])
using bal_if_bal_stk[of []] bal_stk_if_bal by auto

```

### 1.3 More properties of bal, using bal\_stk

**theorem** bal\_append\_inv: bal (u @ v)  $\implies$  bal u  $\implies$  bal v

```

using bal_stk_append_if bal_iff_bal_stk by metis

lemma bal_insert_bal_iff[simp]:
  bal b  $\implies$  bal (v @ b @ w) = bal (v@w)
unfolding bal_iff_bal_stk by(auto simp add: bal_stk_append split: prod.splits
if_splits)

lemma bal_start_Open:  $\langle$ bal (x#xs)  $\implies$   $\exists$  a. x = Open a $\rangle$ 
  using bal_stk.elims bal_iff_bal_stk by blast

lemma bal_Open_split: assumes  $\langle$ bal (x # xs) $\rangle$ 
  shows  $\langle$  $\exists$  y r a. bal y  $\wedge$  bal r  $\wedge$  x = Open a  $\wedge$  xs = y @ Close a # r $\rangle$ 
proof -
  from assms obtain a where  $\langle$ x = Open a $\rangle$ 
  using bal_start_Open by blast
  have  $\langle$ bal (Open a # xs)  $\implies$   $\exists$  y r. bal y  $\wedge$  bal r  $\wedge$  xs = y @ Close a # r $\rangle$ 
  proof(induction  $\langle$ length xs $\rangle$  arbitrary: xs rule: less_induct)
    case less
    have IH:  $\langle$  $\wedge$  w.  $\llbracket$ length w < length xs; bal (Open a # w) $\rrbracket \implies \exists$  y r. bal y  $\wedge$  bal
r  $\wedge$  w = y @ Close a # r $\rangle$ 
    using less by blast
    have  $\langle$ bal (Open a # xs) $\rangle$ 
    using less by blast
    from less(2) show ?case
    proof(induction  $\langle$ Open a # xs $\rangle$  rule: bal.induct)
      case (2 as bs)
      consider (as_empty)  $\langle$ as = [] $\rangle$  | (bs_empty)  $\langle$ bs = [] $\rangle$  | (both_not_empty)
 $\langle$ as  $\neq$  []  $\wedge$  bs  $\neq$  [] $\rangle$  by blast
      then show ?case
      proof(cases)
        case as_empty
        then show ?thesis using 2 by (metis append_Nil)
      next
        case bs_empty
        then show ?thesis using 2 by (metis append_self_conv)
      next
        case both_not_empty
        then obtain as' where as'_def:  $\langle$ Open a # as' = as $\rangle$ 
          using 2 by (metis append_eq_Cons_conv)
        then have  $\langle$ length as' < length xs $\rangle$ 
          using 2.hyps(5) both_not_empty by fastforce
        with IH  $\langle$ bal as $\rangle$  obtain y r where yr:  $\langle$ bal y  $\wedge$  bal r  $\wedge$  as' = y @ Close a
# r $\rangle$ 
          using as'_def by meson
        then have  $\langle$ xs = y @ Close a # r @ bs $\rangle$ 
          using 2.hyps(5) as'_def by fastforce
        moreover have  $\langle$ bal y $\rangle$ 
          using yr by blast
        moreover have  $\langle$ bal (r@bs) $\rangle$ 

```

```

    using yr by (simp add: 2.hyps(3))
    ultimately show ?thesis by blast
  qed
next
  case (3 xs)
  then show ?case by blast
  qed
qed
then show ?thesis using assms  $\langle x = \_ \rangle$  by blast
qed

```

## 1.4 Dyck Language over an Alphabet

The Dyck/bracket language over a set  $\Gamma$  is the set of balanced words over  $\Gamma$ :

**definition** *Dyck\_lang* :: 'a set  $\Rightarrow$  'a bracket list set **where**  
*Dyck\_lang*  $\Gamma = \{w. \text{bal } w \wedge \text{snd } \langle \text{set } w \rangle \subseteq \Gamma\}$

**lemma** *Dyck\_langI*[*intro*]:  
**assumes**  $\langle \text{bal } w \rangle$   
**and**  $\langle \text{snd } \langle \text{set } w \rangle \subseteq \Gamma \rangle$   
**shows**  $\langle w \in \text{Dyck\_lang } \Gamma \rangle$   
**using** *assms* **unfolding** *Dyck\_lang\_def* **by** *blast*

**lemma** *Dyck\_langD*[*dest*]:  
**assumes**  $\langle w \in \text{Dyck\_lang } \Gamma \rangle$   
**shows**  $\langle \text{bal } w \rangle$   
**and**  $\langle \text{snd } \langle \text{set } w \rangle \subseteq \Gamma \rangle$   
**using** *assms* **unfolding** *Dyck\_lang\_def* **by** *auto*

Balanced subwords are again in the Dyck Language.

**lemma** *Dyck\_lang\_substring*:  
 $\langle \text{bal } w \Rightarrow u @ w @ v \in \text{Dyck\_lang } \Gamma \Rightarrow w \in \text{Dyck\_lang } \Gamma \rangle$   
**unfolding** *Dyck\_lang\_def* **by** *auto*

**end**