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

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**
$bal\ [] \mid$
$bal\ xs \Longrightarrow bal\ ys \Longrightarrow bal\ (xs\ @\ ys) \mid$

bal xs $\Longrightarrow$ *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*]
⟨*proof*⟩

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* ⇒ *'a bracket list* ⇒ *'a list* ∗ *'a bracket list* **where**
bal_stk s [] = (*s*,[]) |
bal_stk s (*Open a* # *bs*) = *bal_stk* (*a* # *s*) *bs* |
bal_stk (*a'* # *s*) (*Close a* # *bs*) =
(*if a* = *a'* *then bal_stk s bs else* (*a'* # *s*, *Close a* # *bs*)) |
bal_stk bs stk = (*bs*,*stk*)

lemma *bal_stk_more_stk*: *bal_stk s1 xs* = (*s1'*,[]) $\Longrightarrow$ *bal_stk* (*s1*@*s2*) *xs* = (*s1'*@*s2*,[])
⟨*proof*⟩

lemma *bal_stk_if_Nils*[*simp*]: *ASSUMPTION*(*bal_stk* [] *bs* = ([], [])) $\Longrightarrow$ *bal_stk s bs* = (*s*, [])
⟨*proof*⟩

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*))
⟨*proof*⟩

lemma *bal_stk_append_if*:
bal_stk s1 xs = (*s2*,[]) $\Longrightarrow$ *bal_stk s1* (*xs* @ *ys*) = *bal_stk s2 ys*
⟨*proof*⟩

lemma *bal_stk_split*:
bal_stk s xs = (*s'*,*xs'*) $\Longrightarrow$ ∃ *us*. *xs* = *us*@*xs'* ∧ *bal_stk s us* = (*s'*,[])
⟨*proof*⟩

1.2 Equivalence of *bal* and *bal_stk*

lemma *bal_stk_if_bal*: *bal xs* $\Longrightarrow$ *bal_stk s xs* = (*s*,[])
⟨*proof*⟩

lemma *bal_insert_AB*:
bal (*v* @ *w*) $\Longrightarrow$ *bal* (*v* @ (*Open a* # *Close a* # *w*))
⟨*proof*⟩

lemma *bal_if_bal_stk*: *bal_stk s w* = ([],[]) $\Longrightarrow$ *bal* (*rev*(*map* (λ*x*. *Open x*) *s*) @ *w*)

⟨*proof*⟩

corollary *bal_iff_bal_stk*: *bal w* ⟷ *bal_stk* [] *w* = ([],[])
⟨*proof*⟩

1.3 More properties of *bal*, using *bal_stk*

theorem *bal_append_inv*: *bal* (*u* @ *v*) ⟹ *bal u* ⟹ *bal v*
⟨*proof*⟩

lemma *bal_insert_bal_iff*[*simp*]:
bal b ⟹ *bal* (*v* @ *b* @ *w*) = *bal* (*v*@*w*)
⟨*proof*⟩

lemma *bal_start_Open*: ‹*bal* (*x*#*xs*) ⟹∃ *a*. *x* = *Open a*›
⟨*proof*⟩

lemma *bal_Open_split*: **assumes** ‹*bal* (*x* # *xs*)›
shows ‹∃ *y r a*. *bal y* ∧ *bal r* ∧ *x* = *Open a* ∧ *xs* = *y* @ *Close a* # *r*›
⟨*proof*⟩

1.4 Dyck Language over an Alphabet

The Dyck/bracket language over a set Γ is the set of balanced words over Γ:

definition *Dyck_lang* :: *'a set* ⇒ *'a bracket list set* **where**
Dyck_lang Γ = {*w*. *bal w* ∧ *snd* ' (*set w*) ⊆ Γ}

lemma *Dyck_langI*[*intro*]:
assumes ‹*bal w*›
and ‹*snd* ' (*set w*) ⊆ Γ›
shows ‹*w* ∈ *Dyck_lang* Γ›
⟨*proof*⟩

lemma *Dyck_langD*[*dest*]:
assumes ‹*w* ∈ *Dyck_lang* Γ›
shows ‹*bal w*›
and ‹*snd* ' (*set w*) ⊆ Γ›
⟨*proof*⟩

Balanced subwords are again in the Dyck Language.

lemma *Dyck_lang_substring*:
‹*bal w* ⟹ *u* @ *w* @ *v* ∈ *Dyck_lang* Γ ⟹ *w* ∈ *Dyck_lang* Γ›
⟨*proof*⟩

end