Dyck Language

Tobias Nipkow and Moritz Roos

June 19, 2025

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* × *'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* × *'a*

abbreviation *Open a* ≡ (*True*,*a*)
abbreviation *Close a* ≡ (*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* ⇒ *bool* **where**
  bal [] |
  bal xs ⟹ *bal ys* ⟹ *bal* (*xs* @ *ys*) |

bal xs ⟹ *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'*,[]) ⟹ *bal_stk* (*s1*@*s2*) *xs* = (*s1'*@*s2*,[])
⟨*proof*⟩

lemma *bal_stk_if_Nils*[*simp*]: *ASSUMPTION*(*bal_stk* [] *bs* = ([], [])) ⟹ *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*,[]) ⟹ *bal_stk s1* (*xs* @ *ys*) = *bal_stk s2 ys*
⟨*proof*⟩

lemma *bal_stk_split*:
bal_stk s xs = (*s'*,*xs'*) ⟹ ∃ *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* ⟹ *bal_stk s xs* = (*s*,[])
⟨*proof*⟩

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

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

⟨*proof*⟩

corollary *bal_iff_bal_stk*: $bal\ w \longleftrightarrow bal\_stk\ []\ w = ([],[])$
⟨*proof*⟩

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

theorem *bal_append_inv*: $bal\ (u\ @\ v) \Longrightarrow bal\ u \Longrightarrow bal\ v$
⟨*proof*⟩

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

lemma *bal_start_Open*: ‹$bal\ (x\#xs) \Longrightarrow \exists\, a.\ x = Open\ a$›
⟨*proof*⟩

lemma *bal_Open_split*: **assumes** ‹$bal\ (x\ \#\ xs)$›
shows ‹$\exists\, y\ r\ a.\ bal\ y \wedge bal\ r \wedge x = Open\ a \wedge xs = y\ @\ Close\ a\ \#\ r$›
⟨*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* :: $'a\ set \Rightarrow 'a\ bracket\ list\ set$ **where**
$Dyck\_lang\ \Gamma = \{w.\ bal\ w \wedge snd\ `\ (set\ w) \subseteq \Gamma\}$

lemma *Dyck_langI*[*intro*]:
assumes ‹$bal\ w$›
and ‹$snd\ `\ (set\ w) \subseteq \Gamma$›
shows ‹$w \in Dyck\_lang\ \Gamma$›
⟨*proof*⟩

lemma *Dyck_langD*[*dest*]:
assumes ‹$w \in Dyck\_lang\ \Gamma$›
shows ‹$bal\ w$›
and ‹$snd\ `\ (set\ w) \subseteq \Gamma$›
⟨*proof*⟩

Balanced subwords are again in the Dyck Language.

lemma *Dyck_lang_substring*:
‹$bal\ w \Longrightarrow u\ @\ w\ @\ v \in Dyck\_lang\ \Gamma \Longrightarrow w \in Dyck\_lang\ \Gamma$›
⟨*proof*⟩

end