

Chomsky-Schützenberger Representation Theorem

Moritz Roos and Tobias Nipkow

May 18, 2026

Abstract

The Chomsky-Schützenberger Representation Theorem says that any context-free language is the homomorphic image of the intersection of a regular language and a Dyck language.

Contents

1	Overview of the Proof	2
2	Production Transformation and Homomorphisms	4
2.1	Brackets	4
2.2	Transformation	5
2.3	Homomorphisms	6
3	The Regular Language	7
3.1	$P1$	8
3.2	$P2$	9
3.3	$P3$	9
3.4	$P4$	10
3.5	$P5$	11
3.6	$P6$ and $P7$	11
3.7	Reg and Reg_sym	12
4	Showing Regularity	13
4.1	An automaton for $\{xs. \text{ successively } Q \text{ } xs \wedge xs \in \text{brackets } P\}$.	14
4.2	Regularity of $P2, P3$ and $P4$	16
4.3	An automaton for $P1$	16
4.4	An automaton for $P5$	18
5	Definitions of L, Γ, P', L'	19
6	Lemmas for $P' \vdash A \Rightarrow^* x \longleftrightarrow x \in R_A \cap Dyck_lang \Gamma$	20
7	Showing $h(L') = L$	21

```

theory Chomsky_Schuetzenberger
imports
  Context_Free_Grammar.Parse_Tree
  Context_Free_Grammar.Chomsky_Normal_Form
  Finite_Automata_HF.Finite_Automata_HF
  Dyck_Language_Syms
begin

```

This theory proves the Chomsky-Schützenberger representation theorem [1]. We closely follow Kozen [2] for the proof. The theorem states that every context-free language L can be written as $h(R \cap \text{Dyck_lang } \Gamma)$, for a suitable alphabet Γ , a regular language R and a word-homomorphism h .

The Dyck language over a set Γ (also called it's bracket language) is defined as follows: The symbols of Γ are paired with $[$ and $]$, as in $[_g$ and $]_g$ for $g \in \Gamma$. The Dyck language over Γ is the language of correctly bracketed words. The construction of the Dyck language is found in theory *Chomsky_Schuetzenberger.Dyck_Language_Syms*.

1 Overview of the Proof

A rough proof of Chomsky-Schützenberger is as follows: Take some context-free grammar for L with productions P . Wlog assume it is in Chomsky Normal Form. Now define a new language L' with productions P' in the following way from P :

If $\pi = A \rightarrow BC$ let $\pi' = A \rightarrow [^1_\pi B]^1_p [^2_\pi C]^2_p$, if $\pi = A \rightarrow a$ let $\pi' = A \rightarrow [^1_\pi]^1_p [^2_\pi]^2_p$, where the brackets are viewed as terminals and the old variables A, B, C are again viewed as nonterminals. This transformation is implemented by the function *transform_prod* below. Note brackets are now adorned with superscripts 1 and 2 to distinguish the first and second occurrences easily. That is, we work with symbols that are triples of type $\{[,]\} \times \text{old_prod_type} \times \{1,2\}$.

This bracketing encodes the parse tree of any old word. The old word is easily recovered by the homomorphism which sends $[^1_\pi$ to a if $\pi = A \rightarrow a$, and sends every other bracket to ε . Thus we have $h(L') = L$ by essentially exchanging π for π' and the other way round in the derivation. The direction \supseteq is done in *transfer_parse_tree*, the direction \subseteq is done directly in the proof of the main theorem.

Then all that remains to show is, that L' is of the form $R \cap \text{Dyck_lang } \Gamma$ (for $\Gamma := P \times \{1, 2\}$) and the regularity of R .

For this, $R := R_S$ is defined via an intersection of 5 following regular languages. Each of these is defined via a property on words x :

P1 x : after a $]^1_p$ there always immediately follows a $[^2_p$ in x . This especially means, that $]^1_p$ cannot be the end of the string.

successively P2 x : a $]^2_\pi$ is never directly followed by some $[$ in x .

successively P3 x : each $[^1_{A \rightarrow BC}$ is directly followed by $[^1_{B \rightarrow _}$ in x (last letter isn't checked).

successively P4 x : each $[^1_{A \rightarrow a}$ is directly followed by $]^1_{A \rightarrow a}$ in x and each $]^2_{A \rightarrow a}$ is directly followed by $]^2_{A \rightarrow a}$ in x (last letter isn't checked).

P5 A x : there exists some y such that the word begins with $[^1_{A \rightarrow y}$.

One then shows the key theorem $P' \vdash A \rightarrow^* w \iff w \in R_A \cap Dyck_lang \Gamma$:

The \rightarrow -direction (see lemma *P'_imp_Reg*) is easily checked, by checking that every condition holds during all derivation steps already. For this one needs a version of R (and all the conditions) which ignores any Terminals that might still exist in such a derivation step. Since this version operates on symbols (a different type) it needs a fully new definition. Since these new versions allow more flexibility on the words, it turns out that the original 5 conditions aren't enough anymore to fully constrain to the target language. Thus we add two additional constraints *successively P6* and *successively P7* on the symbol-version of R_A that vanish when we ultimately restricts back to words consisting only of terminal symbols. With these the induction goes through:

(*successively P6_sym*) x : each $Nt Y$ is directly preceded by some Tm $[^1_{A \rightarrow YC}$ or some Tm $]^2_{A \rightarrow BY}$ in x ;

(*successively P7_sym*) x : each $Nt Y$ is directly followed by some $]^1_{A \rightarrow YC}$ or some $]^2_{A \rightarrow BY}$ in x .

The \leftarrow -direction (see lemma *Reg_and_dyck_imp_P'*) is more work. This time we stick with fully terminal words, so we work with the standard version of R_A : Proceed by induction on the length of w generalized over A . For this, let $x \in R_A \cap Dyck_lang \Gamma$, thus we have the properties *P1* x , *successively Pi* x for $i \in \{2,3,4,7,8\}$ and *P5* A x available. From *P5* A x we have that there exists $\pi \in P$ s.t. $fst \pi = A$ and x begins with $[^1_\pi$. Since $x \in Dyck_lang \Gamma$ it is balanced, so it must be of the form $x = [^1_\pi y]^1_\pi r1$ for some balanced y . From *P1* x it must then be of the form $x = [^1_\pi y]^1_\pi [^2_\pi r1'$. Since x is balanced it must then be of the form $x = [^1_\pi y]^1_\pi [^2_\pi z]^2_\pi r2$ for some balanced z . Then $r2$ must also be balanced. If $r2$ was not empty it would begin with an opening bracket, but *P2* x makes this impossible - so $r2 = []$ and as such $x = [^1_\pi y]^1_\pi [^2_\pi z]^2_\pi$. Since our grammar is in CNF, we can consider the following case distinction on π :

Case 1: $\pi = A \rightarrow BC$. Since y, z are balanced substrings of x one easily checks $Pi y$ and $Pi z$ for $i \in \{1, 2, 3, 4\}$. From $P3 x$ (and $\pi = A \rightarrow BC$) we further obtain $P5 B y$ and $P5 C z$. So $y \in R_B \cap Dyck_lang \Gamma$ and $z \in R_C \cap Dyck_lang \Gamma$. From the induction hypothesis we thus obtain $P' \vdash B \rightarrow^* y$ and $P' \vdash C \rightarrow^* z$. Since $\pi = A \rightarrow BC$ we then have $A \rightarrow^1_{\pi'} [^1_{\pi} B]^1_{\pi} [^2_{\pi} C]^2_{\pi} \rightarrow^* [^1_{\pi} y]^1_{\pi} [^2_{\pi} z]^2_{\pi} = x$ as required.

Case 2: $\pi = A \rightarrow a$. Suppose we didn't have $y = []$. Then from $P4 x$ (and $\pi = A \rightarrow a$) we would have $y =]^1_{\pi}$. But since y is balanced it needs to begin with an opening bracket, contradiction. So it must be that $y = []$. By the same argument we also have that $z = []$. So really $x = [^1_{\pi}]^1_{\pi} [^2_{\pi}]^2_{\pi}$ and of course from $\pi = A \rightarrow a$ it holds $A \rightarrow^1_{\pi'} [^1_{\pi}]^1_{\pi} [^2_{\pi}]^2_{\pi} = x$ as required.

From the key theorem we obtain (by setting $A := S$) that $L' = R_S \cap Dyck_lang \Gamma$ as wanted.

Only regularity remains to be shown. For this we use that $R_S \cap Dyck_lang \Gamma = (R_S \cap brackets \Gamma) \cap Dyck_lang \Gamma$, where $brackets \Gamma (\supseteq Dyck_lang \Gamma)$ is the set of words which only consist of brackets over Γ . Actually, what we defined as R_S , isn't regular, only $(R_S \cap brackets \Gamma)$ is. The intersection restricts to a finite amount of possible brackets, that are used in states for finite automaton for the 5 languages that R_S is the intersection of.

Throughout most of the proof below, we implicitly or explicitly assume that the grammar is in CNF. This is lifted only at the very end.

2 Production Transformation and Homomorphisms

A fixed finite set of productions P , used later on:

```

locale locale_P =
fixes P :: ('n,'t) Prods
assumes finiteP: ‹finite P›

```

2.1 Brackets

A type with 2 elements, for creating 2 copies as needed in the proof:

```

datatype version = One | Two

```

```

type_synonym ('n,'t) bracket3 = (('n, 't) prod × version) bracket

```

```

abbreviation open_bracket1 :: ('n, 't) prod ⇒ ('n,'t) bracket3 ([^1_ [1000])
where

```

```

  [^1_p ≡ (Open (p, One))

```

```

abbreviation close_bracket1 :: ('n,'t) prod ⇒ ('n,'t) bracket3 ([^1_ [1000]) where

```

$]^1_p \equiv (\text{Close } (p, \text{One}))$

abbreviation $\text{open_bracket2} :: ('n, 't) \text{prod} \Rightarrow ('n, 't) \text{bracket3 } ([^2_ [1000])$ **where**
 $[^2_p \equiv (\text{Open } (p, \text{Two}))$

abbreviation $\text{close_bracket2} :: ('n, 't) \text{prod} \Rightarrow ('n, 't) \text{bracket3 } (]^2_ [1000])$ **where**
 $]^2_p \equiv (\text{Close } (p, \text{Two}))$

Version for $p = (A, w)$ (multiple letters) with `bsub` and `esub`:

abbreviation $\text{open_bracket1}' :: ('n, 't) \text{prod} \Rightarrow ('n, 't) \text{bracket3 } ([^1_)$ **where**
 $[^1_p \equiv (\text{Open } (p, \text{One}))$

abbreviation $\text{close_bracket1}' :: ('n, 't) \text{prod} \Rightarrow ('n, 't) \text{bracket3 } (]^1_)$ **where**
 $]^1_p \equiv (\text{Close } (p, \text{One}))$

abbreviation $\text{open_bracket2}' :: ('n, 't) \text{prod} \Rightarrow ('n, 't) \text{bracket3 } ([^2_)$ **where**
 $[^2_p \equiv (\text{Open } (p, \text{Two}))$

abbreviation $\text{close_bracket2}' :: ('n, 't) \text{prod} \Rightarrow ('n, 't) \text{bracket3 } (]^2_)$ **where**
 $]^2_p \equiv (\text{Close } (p, \text{Two}))$

Nice LaTeX rendering:

notation (*latex output*) $\text{open_bracket1 } ([^1_)$

notation (*latex output*) $\text{open_bracket1}' ([^1_)$

notation (*latex output*) $\text{open_bracket2 } ([^2_)$

notation (*latex output*) $\text{open_bracket2}' ([^2_)$

notation (*latex output*) $\text{close_bracket1 } (]^1_)$

notation (*latex output*) $\text{close_bracket1}' (]^1_)$

notation (*latex output*) $\text{close_bracket2 } (]^2_)$

notation (*latex output*) $\text{close_bracket2}' (]^2_)$

2.2 Transformation

abbreviation $\text{wrap1} :: \langle 'n \Rightarrow 't \Rightarrow ('n, ('n, 't) \text{bracket3}) \text{syms} \rangle$ **where**

$\langle \text{wrap1 } A \ a \equiv$
 $[\ Tm \ [^1(A, [Tm \ a]),$
 $\quad Tm \]^1(A, [Tm \ a]),$
 $\quad Tm \ [^2(A, [Tm \ a]),$
 $\quad Tm \]^2(A, [Tm \ a]) \] \rangle$

abbreviation $\text{wrap2} :: \langle 'n \Rightarrow 'n \Rightarrow 'n \Rightarrow ('n, ('n, 't) \text{bracket3}) \text{syms} \rangle$ **where**

$\langle \text{wrap2 } A \ B \ C \equiv$
 $[\ Tm \ [^1(A, [Nt \ B, Nt \ C]),$
 $\quad Nt \ B,$
 $\quad Tm \]^1(A, [Nt \ B, Nt \ C]),$
 $\quad Tm \ [^2(A, [Nt \ B, Nt \ C]),$
 $\quad Nt \ C,$

$Tm]^2(A, [Nt B, Nt C])]\rangle$

The transformation of old productions to new productions used in the proof:

fun *transform_rhs* :: $\langle 'n \Rightarrow ('n, 't) \text{ syms} \Rightarrow ('n, ('n, 't) \text{ bracket3}) \text{ syms} \rangle$ **where**
 $\langle \text{transform_rhs } A [Tm a] = \text{wrap1 } A a \rangle$ |
 $\langle \text{transform_rhs } A [Nt B, Nt C] = \text{wrap2 } A B C \rangle$

The last equation is only added to permit us to state lemmas about

fun *transform_prod* :: $\langle ('n, 't) \text{ prod} \Rightarrow ('n, ('n, 't) \text{ bracket3}) \text{ prod} \rangle$ **where**
 $\langle \text{transform_prod } (A, \alpha) = (A, \text{transform_rhs } A \alpha) \rangle$

2.3 Homomorphisms

Definition of a monoid-homomorphism where multiplication is (@):

definition *hom_list* :: $\langle ('a \text{ list} \Rightarrow 'b \text{ list}) \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{hom_list } h = (\forall a b. h (a @ b) = h a @ h b) \rangle$

lemma *hom_list_Nil*: $\text{hom_list } h \implies h [] = []$
 $\langle \text{proof} \rangle$

The homomorphism on single brackets:

fun *the_hom1* :: $\langle ('n, 't) \text{ bracket3} \Rightarrow 't \text{ list} \rangle$ **where**
 $\langle \text{the_hom1 } [^1(A, [Tm a])] = [a] \rangle$ |
 $\langle \text{the_hom1 } _ = [] \rangle$

The homomorphism on single bracket symbols:

fun *the_hom_sym* :: $\langle ('n, ('n, 't) \text{ bracket3}) \text{ sym} \Rightarrow ('n, 't) \text{ sym list} \rangle$ **where**
 $\langle \text{the_hom_sym } (Tm [^1(A, [Tm a]))) = [Tm a] \rangle$ |
 $\langle \text{the_hom_sym } (Nt A) = [Nt A] \rangle$ |
 $\langle \text{the_hom_sym } _ = [] \rangle$

The homomorphism on bracket words:

fun *the_hom* :: $\langle ('n, 't) \text{ bracket3 list} \Rightarrow 't \text{ list} \rangle$ (h) **where**
 $\langle \text{the_hom } l = \text{concat } (\text{map } \text{the_hom1 } l) \rangle$

The homomorphism extended to symbols:

fun *the_hom_syms* :: $\langle ('n, ('n, 't) \text{ bracket3}) \text{ syms} \Rightarrow ('n, 't) \text{ syms} \rangle$ **where**
 $\langle \text{the_hom_syms } l = \text{concat } (\text{map } \text{the_hom_sym } l) \rangle$

notation *the_hom* (h)

notation *the_hom_syms* (hs)

lemma *the_hom_syms_hom*: $\langle \text{hs } (l1 @ l2) = \text{hs } l1 @ \text{hs } l2 \rangle$
 $\langle \text{proof} \rangle$

lemma *the_hom_syms_keep_var*: $\langle \text{hs } [(Nt A)] = [Nt A] \rangle$
 $\langle \text{proof} \rangle$

lemma *the_hom_syms_tms_inj*: $\langle \text{hs } w = \text{map } Tm \ m \implies \exists w'. w = \text{map } Tm \ w' \rangle$

<proof>

Helper for showing the upcoming lemma:

lemma *helper*: $\langle \text{the_hom_sym } (Tm \ x) = \text{map } Tm \ (\text{the_hom1 } x) \rangle$

<proof>

Show that the extension really is an extension in some sense:

lemma *h_eq_h_ext*: $\langle \text{hs } (\text{map } Tm \ x) = \text{map } Tm \ (\text{h } x) \rangle$

<proof>

lemma *the_hom1_strip*: $\langle (\text{the_hom_sym } x') = \text{map } Tm \ w \implies \text{the_hom1 } (\text{destTm } x') = w \rangle$

<proof>

lemma *the_hom1_strip2*: $\langle \text{concat } (\text{map } \text{the_hom_sym } w) = \text{map } Tm \ w \implies \text{concat } (\text{map } (\text{the_hom1 } \circ \text{destTm}) \ w) = w \rangle$

<proof>

lemma *h_eq_h_ext2*:

assumes $\langle \text{hs } w' = (\text{map } Tm \ w) \rangle$

shows $\langle \text{h } (\text{map } \text{destTm } w') = w \rangle$

<proof>

3 The Regular Language

The regular Language *Reg* will be an intersection of 5 Languages. The languages 2, 3, 4 are defined each via a relation *P2*, *P3*, *P4* on neighbouring letters and lifted to a language via *successively*. Language 1 is an intersection of another such lifted relation *P1'* and a condition on the last letter (if existent). Language 5 is a condition on the first letter (and requires it to exist). It takes a term of type *'n* (the original variable type) as parameter.

Additionally a version of each language (taking symbols as input) is defined which allows arbitrary interspersions of nonterminals.

As this interspersions weakens the description, the symbol version of the regular language (*Reg_sym*) is defined using two additional languages lifted from *P6* and *P7*. These vanish when restricted to words only containing terminals.

As stated in the introductory text, these languages will only be regular, when constrained to a finite bracket set. The theorems about this, are in the later section *Showing Regularity*.

3.1 $P1$

$P1$ will define a predicate on string elements. It will be true iff each $]_p^1$ is directly followed by $]_p^2$. That also means $]_p^1$ cannot be the end of the string.

But first we define a helper function, that only captures the neighbouring condition for two strings:

```
fun  $P1'$  ::  $\langle ('n, 't) \text{ bracket3} \Rightarrow ('n, 't) \text{ bracket3} \Rightarrow \text{bool} \rangle$  where
   $\langle P1' ]_p^1 b' = (b' = ]_p^2) \rangle$  |
   $\langle P1' x y = \text{True} \rangle$ 
```

A version of $P1'$ for symbols, i.e. strings that may still contain Nt's:

```
fun  $P1\_sym$  ::  $\langle ('n, ('n, 't) \text{ bracket3}) \text{ sym} \Rightarrow ('n, ('n, 't) \text{ bracket3}) \text{ sym} \Rightarrow \text{bool} \rangle$ 
where
   $\langle P1\_sym (Tm ]_p^1) s' = (s' = (Tm ]_p^2)) \rangle$  |
   $\langle P1\_sym x y = \text{True} \rangle$ 
```

Asserts that $P1'$ holds for every pair in xs , and that xs doesn't end in $]_p^1$:

```
fun  $P1$  ::  $\langle ('n, 't) \text{ bracket3 list} \Rightarrow \text{bool} \rangle$  where
   $\langle P1 u = ((\text{successively } P1' u) \wedge (u \neq [] \longrightarrow (\nexists \pi. \text{last } u = ]_p^1))) \rangle$ 
```

Asserts that $P1'$ holds for every pair in xs , and that xs doesn't end in $Tm]_p^1$:

```
fun  $P1\_sym$  where
   $\langle P1\_sym xs = ((\text{successively } P1\_sym xs) \wedge (xs \neq [] \longrightarrow (\nexists p. \text{last } xs = Tm ]_p^1))) \rangle$ 
```

```
lemma  $P1\_for\_tm\_if\_P1\_sym[dest!]$ :  $\langle P1\_sym (\text{map } Tm x) \Longrightarrow P1 x \rangle$ 
 $\langle \text{proof} \rangle$ 
```

```
lemma  $P1I[intro]$ :
  assumes  $\langle \text{successively } P1' xs \rangle$ 
  and  $\langle \nexists p. \text{last } xs = ]_p^1 \rangle$ 
  shows  $\langle P1 xs \rangle$ 
 $\langle \text{proof} \rangle$ 
```

```
lemma  $P1\_symI[intro]$ :
  assumes  $\langle \text{successively } P1\_sym xs \rangle$ 
  and  $\langle \nexists p. \text{last } xs = Tm ]_p^1 \rangle$ 
  shows  $\langle P1\_sym xs \rangle$ 
 $\langle \text{proof} \rangle$ 
```

```
lemma  $P1\_symD[dest]$ :  $\langle P1\_sym xs \Longrightarrow \text{successively } P1\_sym xs \rangle$   $\langle \text{proof} \rangle$ 
```

```
lemma  $P1D\_not\_empty[intro]$ :
  assumes  $\langle xs \neq [] \rangle$ 
  and  $\langle P1 xs \rangle$ 
  shows  $\langle \text{last } xs \neq ]_p^1 \rangle$ 
```

⟨proof⟩

lemma *P1_symD_not_empty*[intro]:
 assumes ⟨ $xs \neq []$ ⟩
 and ⟨*P1_sym* xs ⟩
 shows ⟨ $last\ xs \neq Tm\]^1_p$ ⟩
⟨proof⟩

lemma *P1_symD_not_empty*:
 assumes ⟨ $xs \neq []$ ⟩
 and ⟨*P1_sym* xs ⟩
 shows ⟨ $\nexists p. last\ xs = Tm\]^1_p$ ⟩
⟨proof⟩

3.2 *P2*

$A\]^2_\pi$ is never directly followed by some $[\cdot$:

fun *P2* :: ⟨ $('n, 't)\ bracket3 \Rightarrow ('n, 't)\ bracket3 \Rightarrow bool$ ⟩ **where**
 ⟨*P2* (*Close* (p, Two)) (*Open* (p', v)) = *False*⟩ |
 ⟨*P2* (*Close* (p, Two)) $y = True$ ⟩ |
 ⟨*P2* $x\ y = True$ ⟩

fun *P2_sym* :: ⟨ $('n, ('n, 't)\ bracket3)\ sym \Rightarrow ('n, ('n, 't)\ bracket3)\ sym \Rightarrow bool$ ⟩
where
 ⟨*P2_sym* (*Tm* (*Close* (p, Two))) (*Tm* (*Open* (p', v))) = *False*⟩ |
 ⟨*P2_sym* (*Tm* (*Close* (p, Two))) $y = True$ ⟩ |
 ⟨*P2_sym* $x\ y = True$ ⟩

lemma *P2_for_tm_if_P2_sym*[dest]: ⟨*successively* *P2_sym* (*map* *Tm* x) \implies *successively* *P2* x ⟩
⟨proof⟩

3.3 *P3*

Each $[^1_{A \rightarrow BC}$ is directly followed by $[^1_{B \rightarrow _}$, and each $[^2_{A \rightarrow BC}$ is directly followed by $[^1_{C \rightarrow _}$:

fun *P3* :: ⟨ $('n, 't)\ bracket3 \Rightarrow ('n, 't)\ bracket3 \Rightarrow bool$ ⟩ **where**
 ⟨*P3* $[^1(A, [Nt\ B, Nt\ C])\ (p, ((X, y), t)) = (p = True \wedge t = One \wedge X = B)$ ⟩ |
 ⟨*P3* $[^2(A, [Nt\ B, Nt\ C])\ (p, ((X, y), t)) = (p = True \wedge t = One \wedge X = C)$ ⟩ |
 ⟨*P3* $x\ y = True$ ⟩

Each $[^1_{A \rightarrow BC}$ is directly followed $[^1_{B \rightarrow _}$ or *Nt* B , and each $[^2_{A \rightarrow BC}$ is directly followed by $[^1_{C \rightarrow _}$ or *Nt* C :

fun *P3_sym* :: ⟨ $('n, ('n, 't)\ bracket3)\ sym \Rightarrow ('n, ('n, 't)\ bracket3)\ sym \Rightarrow bool$ ⟩
where
 ⟨*P3_sym* (*Tm* $[^1(A, [Nt\ B, Nt\ C])\ (Tm\ (p, ((X, y), t))) = (p = True \wedge t = One \wedge X = B)$ ⟩ |

— Not obvious: the case $(Tm \ [^1(A, [Nt B, Nt C]) \ Nt X)$ is set to `True` with the catch all

$\langle P3_sym \ (Tm \ [^1(A, [Nt B, Nt C]) \ (Nt X) = (X = B)] \ |$

$\langle P3_sym \ (Tm \ [^2(A, [Nt B, Nt C]) \ (Tm \ (p, ((X,y), t))) = (p = True \wedge t = One \wedge X = C)] \ |$

$\langle P3_sym \ (Tm \ [^2(A, [Nt B, Nt C]) \ (Nt X) = (X = C)] \ |$

$\langle P3_sym \ x \ y = True \rangle$

lemma $P3D1[dest]$:

fixes $r::\langle('n,'t) \ bracket3\rangle$

assumes $\langle P3 \ [^1(A, [Nt B, Nt C]) \ r \rangle$

shows $\langle \exists l. r = [^1(B, l) \rangle$

$\langle proof \rangle$

lemma $P3D2[dest]$:

fixes $r::\langle('n,'t) \ bracket3\rangle$

assumes $\langle P3 \ [^2(A, [Nt B, Nt C]) \ r \rangle$

shows $\langle \exists l. r = [^1(C, l) \rangle$

$\langle proof \rangle$

lemma $P3_for_tm_if_P3_sym[dest]$: $\langle successively \ P3_sym \ (map \ Tm \ x) \implies \ successively \ P3 \ x \rangle$

$\langle proof \rangle$

3.4 $P4$

Each $[^1_{A \rightarrow a}$ is directly followed by $]^1_{A \rightarrow a}$ and each $[^2_{A \rightarrow a}$ is directly followed by $]^2_{A \rightarrow a}$:

fun $P4 :: \langle('n,'t) \ bracket3 \Rightarrow ('n,'t) \ bracket3 \Rightarrow bool\rangle$ **where**

$\langle P4 \ (Open \ ((A, [Tm \ a]), s)) \ (p, ((X, y), t)) = (p = False \wedge X = A \wedge y = [Tm \ a] \wedge s = t) \rangle \ |$

$\langle P4 \ x \ y = True \rangle$

Each $[^1_{A \rightarrow a}$ is directly followed by $]^1_{A \rightarrow a}$ and each $[^2_{A \rightarrow a}$ is directly followed by $]^2_{A \rightarrow a}$:

fun $P4_sym :: \langle('n, ('n,'t) \ bracket3) \ sym \Rightarrow ('n, ('n,'t) \ bracket3) \ sym \Rightarrow bool\rangle$ **where**

$\langle P4_sym \ (Tm \ (Open \ ((A, [Tm \ a]), s))) \ (Tm \ (p, ((X, y), t))) = (p = False \wedge X = A \wedge y = [Tm \ a] \wedge s = t) \rangle \ |$

$\langle P4_sym \ (Tm \ (Open \ ((A, [Tm \ a]), s))) \ (Nt \ X) = False \rangle \ |$

$\langle P4_sym \ x \ y = True \rangle$

lemma $P4D[dest]$:

fixes $r::\langle('n,'t) \ bracket3\rangle$

assumes $\langle P4 \ (Open \ ((A, [Tm \ a]), v)) \ r \rangle$

shows $\langle r = Close \ ((A, [Tm \ a]), v) \rangle$

$\langle proof \rangle$

lemma $P4_for_tm_if_P4_sym[dest]$: $\langle successively\ P4_sym\ (map\ Tm\ x) \implies successively\ P4\ x \rangle$
 $\langle proof \rangle$

3.5 $P5$

$P5\ A\ x$ holds, iff there exists some y such that x begins with $[^1_{A \rightarrow y}$:

fun $P5$:: $\langle 'n \Rightarrow ('n, 't)\ bracket3\ list \Rightarrow bool \rangle$ **where**
 $\langle P5\ A\ [] = False \rangle$ |
 $\langle P5\ A\ ([^1_{(X,x)} \# xs) = (X = A) \rangle$ |
 $\langle P5\ A\ (x \# xs) = False \rangle$

$P5_sym\ A\ x$ holds, iff either there exists some y such that x begins with $[^1_{A \rightarrow y}$, or if it begins with $Nt\ A$:

fun $P5_sym$:: $\langle 'n \Rightarrow ('n, ('n, 't)\ bracket3)\ syms \Rightarrow bool \rangle$ **where**
 $\langle P5_sym\ A\ [] = False \rangle$ |
 $\langle P5_sym\ A\ (Tm\ [^1_{(X,x)} \# xs) = (X = A) \rangle$ |
 $\langle P5_sym\ A\ ((Nt\ X) \# xs) = (X = A) \rangle$ |
 $\langle P5_sym\ A\ (x \# xs) = False \rangle$

lemma $P5D[dest]$:
assumes $\langle P5\ A\ x \rangle$
shows $\langle \exists y. hd\ x = [^1_{(A,y)} \rangle$
 $\langle proof \rangle$

lemma $P5_symD[dest]$:
assumes $\langle P5_sym\ A\ x \rangle$
shows $\langle (\exists y. hd\ x = Tm\ [^1_{(A,y)}) \vee hd\ x = Nt\ A \rangle$
 $\langle proof \rangle$

lemma $P5_for_tm_if_P5_sym[dest]$: $\langle P5_sym\ A\ (map\ Tm\ x) \implies P5\ A\ x \rangle$
 $\langle proof \rangle$

3.6 $P6$ and $P7$

(*successively* $P6_sym$) w iff $Nt\ Y$ is directly preceded by some $Tm\ [^1_{A \rightarrow YC}$ or $Tm\ [^2_{A \rightarrow BY}$ in w :

fun $P6_sym$:: $\langle ('n, ('n, 't)\ bracket3)\ sym \Rightarrow ('n, ('n, 't)\ bracket3)\ sym \Rightarrow bool \rangle$
where
 $\langle P6_sym\ (Tm\ (b, (A, [Nt\ B, Nt\ C]), v))\ (Nt\ Y) = (b = True \wedge ((Y = B \wedge v = One) \vee (Y = C \wedge v = Two))) \rangle$ |
 $\langle P6_sym\ x\ (Nt\ Y) = False \rangle$ |
 $\langle P6_sym\ x\ y = True \rangle$

lemma $P6_symD[dest]$:
fixes x :: $\langle ('n, ('n, 't)\ bracket3)\ sym \rangle$

assumes $\langle P6_sym\ x\ (Nt\ Y) \rangle$
shows $\langle (\exists A\ C.\ x = Tm\ [^1(A,[Nt\ Y,\ Nt\ C])]) \vee (\exists A\ B.\ x = Tm\ [^2(A,[Nt\ B,\ Nt\ Y])]) \rangle$
 $\langle proof \rangle$

$(successively\ P7_sym)$ w iff $Nt\ Y$ is directly followed by some $]^1_{A \rightarrow YC}$ or $]^2_{A \rightarrow BY}$ in w :

fun $P7_sym :: \langle ('n,\ ('n,\ 't)\ bracket3)\ sym \Rightarrow ('n,\ ('n,\ 't)\ bracket3)\ sym \Rightarrow bool \rangle$
where
 $\langle P7_sym\ (Nt\ Y)\ (Tm\ (b,(A,\ [Nt\ B,\ Nt\ C]),\ v)) = (b = False \wedge ((Y = B \wedge v = One) \vee (Y = C \wedge v = Two))) \rangle |$
 $\langle P7_sym\ (Nt\ Y)\ x = False \rangle |$
 $\langle P7_sym\ x\ y = True \rangle$

lemma $P7_symD[dest]$:

fixes $x :: \langle ('n,\ ('n,\ 't)\ bracket3)\ sym \rangle$

assumes $\langle P7_sym\ (Nt\ Y)\ x \rangle$

shows $\langle (\exists A\ C.\ x = Tm\ [^1(A,[Nt\ Y,\ Nt\ C])]) \vee (\exists A\ B.\ x = Tm\ [^2(A,[Nt\ B,\ Nt\ Y])]) \rangle$
 $\langle proof \rangle$

3.7 *Reg* and *Reg_sym*

This is the regular language, where one takes the Start symbol as a parameter, and then has the searched for $R := R_A$:

definition $Reg :: \langle 'n \Rightarrow ('n,\ 't)\ bracket3\ list\ set \rangle$ **where**

$\langle Reg\ A = \{u.\ P1\ u \wedge$
 $successively\ P2\ u \wedge$
 $successively\ P3\ u \wedge$
 $successively\ P4\ u \wedge$
 $P5\ A\ u\} \rangle$

lemma $RegI[intro]$:

assumes $\langle P1\ u \rangle$

and $\langle successively\ P2\ u \rangle$

and $\langle successively\ P3\ u \rangle$

and $\langle successively\ P4\ u \rangle$

and $\langle P5\ A\ u \rangle$

shows $\langle u \in Reg\ A \rangle$

$\langle proof \rangle$

lemma $RegD[dest]$:

assumes $\langle u \in Reg\ A \rangle$

shows $\langle P1\ u \rangle$

and $\langle successively\ P2\ u \rangle$

and $\langle successively\ P3\ u \rangle$

and $\langle successively\ P4\ u \rangle$

and $\langle P5\ A\ u \rangle$

$\langle proof \rangle$

A version of *Reg* for symbols, i.e. strings that may still contain Nt's. It

has 2 more Properties $P6$ and $P7$ that vanish for pure terminal strings:

definition $Reg_sym :: \langle 'n \Rightarrow ('n, ('n, 't) \text{ bracket3}) \text{ syms set} \rangle$ **where**

```

  \Reg_sym A = {u. P1_sym u \
    successively P2_sym u \
    successively P3_sym u \
    successively P4_sym u \
    P5_sym A u \
    successively P6_sym u \
    successively P7_sym u}

```

lemma $Reg_symI[\text{intro}]$:

```

assumes \P1_sym u
  and \successively P2_sym u
  and \successively P3_sym u
  and \successively P4_sym u
  and \P5_sym A u
  and \successively P6_sym u
  and \successively P7_sym u
shows \u \in Reg_sym A
\proof

```

lemma $Reg_symD[\text{dest}]$:

```

assumes \u \in Reg_sym A
shows \P1_sym u
  and \successively P2_sym u
  and \successively P3_sym u
  and \successively P4_sym u
  and \P5_sym A u
  and \successively P6_sym u
  and \successively P7_sym u
\proof

```

lemma $Reg_for_tm_if_Reg_sym[\text{dest}]$: $\langle \text{map } Tm \ u \in Reg_sym \ A \implies u \in Reg \ A \rangle$

\proof

4 Showing Regularity

context $locale_P$

begin

abbreviation $brackets :: \langle ('n, 't) \text{ bracket3 list set} \rangle$ **where**

```

  \brackets \equiv {bs. \forall (\_, p, \_) \in set bs. p \in P}

```

This is needed for the construction that shows P2,P3,P4 regular.

datatype $'a \text{ state} = \text{start} \mid \text{garbage} \mid \text{letter } 'a$

definition $allStates :: \langle ('n, 't) \text{ bracket3 state set} \rangle$ **where**

$\langle allStates = \{ letter (br,(p,v)) \mid br\ p\ v.\ p \in P \} \cup \{ start, garbage \} \rangle$

lemma *allStatesI*: $\langle p \in P \implies letter (br,(p,v)) \in allStates \rangle$
 $\langle proof \rangle$

lemma *start_in_allStates[simp]*: $\langle start \in allStates \rangle$
 $\langle proof \rangle$

lemma *garbage_in_allStates[simp]*: $\langle garbage \in allStates \rangle$
 $\langle proof \rangle$

lemma *finite_allStates_if*:
shows $\langle finite(allStates) \rangle$
 $\langle proof \rangle$

end

4.1 An automaton for $\{xs.\ successively\ Q\ xs \wedge xs \in brackets\ P\}$

locale *successivelyConstruction* = *locale_P* **where** $P = P$ **for** $P :: ('n,'t)\ Prods$
 $+$
fixes $Q :: ('n,'t)\ bracket3 \Rightarrow ('n,'t)\ bracket3 \Rightarrow bool$ — e.g. P2
begin

fun *succNext* :: $\langle ('n,'t)\ bracket3\ state \Rightarrow ('n,'t)\ bracket3 \Rightarrow ('n,'t)\ bracket3\ state \rangle$
where
 $\langle succNext\ garbage\ _ = garbage \rangle \mid$
 $\langle succNext\ start\ (br',\ p',\ v') = (if\ p' \in P\ then\ letter\ (br',\ p',\ v')\ else\ garbage) \rangle \mid$
 $\langle succNext\ (letter\ (br,\ p,\ v))\ (br',\ p',\ v') = (if\ Q\ (br,\ p,\ v)\ (br',\ p',\ v') \wedge p \in P \wedge p' \in P\ then\ letter\ (br',\ p',\ v')\ else\ garbage) \rangle$

theorem *succNext_induct[case_names garbage startp startnp letterQ letternQ]*:
fixes $R :: ('n,'t)\ bracket3\ state \Rightarrow ('n,'t)\ bracket3 \Rightarrow bool$
fixes $a0 :: ('n,'t)\ bracket3\ state$
and $a1 :: ('n,'t)\ bracket3$
assumes $\bigwedge u.\ R\ garbage\ u$
and $\bigwedge br'\ p'\ v'.\ p' \in P \implies R\ state.start\ (br',\ p',\ v')$
and $\bigwedge br'\ p'\ v'.\ p' \notin P \implies R\ state.start\ (br',\ p',\ v')$
and $\bigwedge br\ p\ v\ br'\ p'\ v'.\ Q\ (br,\ p,\ v)\ (br',\ p',\ v') \wedge p \in P \wedge p' \in P \implies R\ (letter\ (br,\ p,\ v))\ (br',\ p',\ v')$
and $\bigwedge br\ p\ v\ br'\ p'\ v'.\ \neg(Q\ (br,\ p,\ v)\ (br',\ p',\ v') \wedge p \in P \wedge p' \in P) \implies R\ (letter\ (br,\ p,\ v))\ (br',\ p',\ v')$
shows $R\ a0\ a1$
 $\langle proof \rangle$

abbreviation *aut* **where** $\langle aut \equiv (\{dfa.states = allStates,$
 $init = start,$
 $final = (allStates - \{garbage\}),$
 $nxt = succNext\ \}) \rangle$

interpretation *aut* : *dfa aut*

<proof>

lemma *nextl_in_allStates*[*intro,simp*]: $\langle q \in \text{allStates} \implies \text{aut.nextl } q \text{ ys} \in \text{allStates} \rangle$

<proof>

lemma *nextl_garbage*[*simp*]: $\langle \text{aut.nextl garbage } xs = \text{garbage} \rangle$

<proof>

lemma *drop_right*: $\langle xs@ys \in \text{aut.language} \implies xs \in \text{aut.language} \rangle$

<proof>

lemma *state_after1*[*iff*]: $\langle (\text{succNext } q \ a \neq \text{garbage}) = (\text{succNext } q \ a = \text{letter } a) \rangle$

<proof>

lemma *state_after_in_P*[*intro*]: $\langle \text{succNext } q \ (br, p, v) \neq \text{garbage} \implies p \in P \rangle$

<proof>

lemma *drop_left_general*: $\langle \text{aut.nextl start } ys = \text{garbage} \implies \text{aut.nextl } q \ ys = \text{garbage} \rangle$

<proof>

lemma *drop_left*: $\langle xs@ys \in \text{aut.language} \implies ys \in \text{aut.language} \rangle$

<proof>

lemma *empty_in_aut*: $\langle [] \in \text{aut.language} \rangle$

<proof>

lemma *singleton_in_aut_iff*: $\langle [(br, p, v)] \in \text{aut.language} \iff p \in P \rangle$

<proof>

lemma *duo_in_aut_iff*: $\langle [(br, p, v), (br', p', v')] \in \text{aut.language} \iff Q \ (br,p,v) \ (br',p',v') \wedge p \in P \wedge p' \in P \rangle$

<proof>

lemma *trio_in_aut_iff*: $\langle (br, p, v) \# (br', p', v') \# zs \in \text{aut.language} \iff Q \ (br,p,v) \ (br',p',v') \wedge p \in P \wedge p' \in P \wedge (br',p',v') \# zs \in \text{aut.language} \rangle$

<proof>

lemma *aut_lang_iff_succ_Q*: $\langle (\text{successively } Q \ xs \wedge xs \in \text{brackets}) \iff (xs \in \text{aut.language}) \rangle$

<proof>

corollary *regular_successively_inter_brackets*: $\langle \text{regular } \{xs. \text{successively } Q \ xs \wedge xs \in \text{brackets}\} \rangle$

<proof>

end

4.2 Regularity of $P2$, $P3$ and $P4$

context *locale_P*

begin

lemma *P2_regular*:

shows $\langle \text{regular } \{xs. \text{ successively } P2 \text{ } xs \wedge xs \in \text{brackets}\} \rangle$
 $\langle \text{proof} \rangle$

lemma *P3_regular*:

$\langle \text{regular } \{xs. \text{ successively } P3 \text{ } xs \wedge xs \in \text{brackets}\} \rangle$
 $\langle \text{proof} \rangle$

lemma *P4_regular*:

$\langle \text{regular } \{xs. \text{ successively } P4 \text{ } xs \wedge xs \in \text{brackets}\} \rangle$
 $\langle \text{proof} \rangle$

4.3 An automaton for $P1$

More Precisely, for the *if not empty, then doesnt end in (Close_,1)* part.
Then intersect with the other construction for $P1'$ to get $P1$ regular.

datatype *P1_State* = *last_ok* | *last_bad* | *garbage*

The good ending letters, are those that are not of the form (*Close _* ,
1).

fun *good* where

$\langle \text{good }]^1_p = \text{False} \rangle$ |
 $\langle \text{good } (br, p, v) = \text{True} \rangle$

fun *nxt1* :: $\langle P1_State \Rightarrow ('n,'t) \text{ bracket3} \Rightarrow P1_State \rangle$ where

$\langle \text{nxt1 } \text{garbage_} = \text{garbage} \rangle$ |
 $\langle \text{nxt1 } \text{last_ok } (br, p, v) = (\text{if } p \notin P \text{ then garbage else (if good } (br, p, v) \text{ then } \text{last_ok} \text{ else } \text{last_bad})) \rangle$ |
 $\langle \text{nxt1 } \text{last_bad } (br, p, v) = (\text{if } p \notin P \text{ then garbage else (if good } (br, p, v) \text{ then } \text{last_ok} \text{ else } \text{last_bad})) \rangle$

theorem *nxt1_induct*[*case_names garbage startp startnp letterQ letternQ*]:

fixes *R* :: $P1_State \Rightarrow ('n,'t) \text{ bracket3} \Rightarrow \text{bool}$

fixes *a0* :: $P1_State$

and *a1* :: $('n,'t) \text{ bracket3}$

assumes $\bigwedge u. R \text{ garbage } u$

and $\bigwedge br \ p \ v. p \notin P \implies R \text{ last_ok } (br, p, v)$

and $\bigwedge br \ p \ v. p \in P \wedge \text{good } (br, p, v) \implies R \text{ last_ok } (br, p, v)$

and $\bigwedge br \ p \ v. p \in P \wedge \neg(\text{good } (br, p, v)) \implies R \text{ last_ok } (br, p, v)$

and $\bigwedge br \ p \ v. p \notin P \implies R \text{ last_bad } (br, p, v)$

and $\bigwedge br \ p \ v. p \in P \wedge \text{good } (br, p, v) \implies R \text{ last_bad } (br, p, v)$

and $\bigwedge br \ p \ v. p \in P \wedge \neg(\text{good } (br, p, v)) \implies R \text{ last_bad } (br, p, v)$

shows $R\ a0\ a1$
 $\langle proof \rangle$

abbreviation $p1_aut$ **where** $\langle p1_aut \equiv (dfa.states = \{last_ok, last_bad, garbage\},$
 $init = last_ok,$
 $final = \{last_ok\},$
 $next = next1) \rangle$

interpretation $p1_aut : dfa\ p1_aut$
 $\langle proof \rangle$

lemma $nextl_garbage_iff[simp]$: $\langle p1_aut.nextl\ last_ok\ xs = garbage \longleftrightarrow xs \notin$
 $brackets \rangle$
 $\langle proof \rangle$

lemma $lang_descr_full$:
 $\langle (p1_aut.nextl\ last_ok\ xs = last_ok \longleftrightarrow (xs = [] \vee (xs \neq [] \wedge good\ (last\ xs) \wedge$
 $xs \in brackets))) \wedge$
 $(p1_aut.nextl\ last_ok\ xs = last_bad \longleftrightarrow ((xs \neq [] \wedge \neg good\ (last\ xs) \wedge xs \in$
 $brackets))) \rangle$
 $\langle proof \rangle$

lemma $lang_descr$: $\langle xs \in p1_aut.language \longleftrightarrow (xs = [] \vee (xs \neq [] \wedge good\ (last$
 $xs) \wedge xs \in brackets) \rangle$
 $\langle proof \rangle$

lemma $good_iff[simp]$: $\langle (\forall a\ b. last\ xs \neq]^1(a, b) = good\ (last\ xs) \rangle$
 $\langle proof \rangle$

lemma in_P1_iff : $\langle (P1\ xs \wedge xs \in brackets) \longleftrightarrow (xs = [] \vee (xs \neq [] \wedge good\ (last$
 $xs) \wedge xs \in brackets)) \wedge successively\ P1'\ xs \wedge xs \in brackets \rangle$
 $\langle proof \rangle$

corollary $P1_eq$: $\langle \{xs. P1\ xs \wedge xs \in brackets\} =$
 $\{xs. successively\ P1'\ xs \wedge xs \in brackets\} \cap \{xs. xs = [] \vee (xs \neq [] \wedge good$
 $(last\ xs) \wedge xs \in brackets)\} \rangle$
 $\langle proof \rangle$

lemma $P1'_regular$:
shows $\langle regular\ \{xs. successively\ P1'\ xs \wedge xs \in brackets\} \rangle$
 $\langle proof \rangle$

corollary $aux_regular$: $\langle regular\ \{xs. xs = [] \vee (xs \neq [] \wedge good\ (last\ xs) \wedge xs \in$
 $brackets)\} \rangle$
 $\langle proof \rangle$

corollary $regular_P1$: $\langle regular\ \{xs. P1\ xs \wedge xs \in brackets\} \rangle$
 $\langle proof \rangle$

end

4.4 An automaton for $P5$

locale $P5Construction = locale_P$ **where** $P=P$ **for** $P :: ('n,'t)Prods +$
fixes $A :: 'n$
begin

datatype $P5_State = start | first_ok | garbage$

The good/ok ending letters, are those that are not of the form ($Close _$, 1).

fun ok **where**

$\langle ok (Open ((X, _), One)) = (X = A) \rangle |$
 $\langle ok _ = False \rangle$

fun $next2 :: \langle P5_State \Rightarrow ('n,'t) bracket3 \Rightarrow P5_State \rangle$ **where**

$\langle next2 garbage _ = garbage \rangle |$
 $\langle next2 start (br, (X, r), v) = (if (X,r) \notin P then garbage else (if ok (br, (X, r), v) then first_ok else garbage)) \rangle |$
 $\langle next2 first_ok (br, p, v) = (if p \notin P then garbage else first_ok) \rangle$

theorem $next2_induct[case_names garbage startnp start_p_ok start_p_nok first_ok_np first_ok_p]:$

fixes $R :: P5_State \Rightarrow ('n,'t) bracket3 \Rightarrow bool$

fixes $a0 :: P5_State$

and $a1 :: ('n,'t) bracket3$

assumes $\bigwedge u. R garbage u$

and $\bigwedge br p v. p \notin P \implies R start (br, p, v)$

and $\bigwedge br X r v. (X, r) \in P \wedge ok (br, (X, r), v) \implies R start (br, (X, r), v)$

and $\bigwedge br X r v. (X, r) \in P \wedge \neg ok (br, (X, r), v) \implies R start (br, (X, r), v)$

and $\bigwedge br X r v. (X, r) \notin P \implies R first_ok (br, (X, r), v)$

and $\bigwedge br X r v. (X, r) \in P \implies R first_ok (br, (X, r), v)$

shows $R a0 a1$

$\langle proof \rangle$

abbreviation $p5_aut$ **where** $\langle p5_aut \equiv (\langle dfa.states = \{start, first_ok, garbage\},$

$init = start,$

$final = \{first_ok\},$

$next = next2 \rangle \rangle$

interpretation $p5_aut : dfa p5_aut$

$\langle proof \rangle$

corollary $next2_start_ok_iff: \langle ok x \wedge fst(snd x) \in P \longleftrightarrow next2 start x = first_ok \rangle$

$\langle proof \rangle$

lemma $empty_not_in_lang[simp]: \langle [] \notin p5_aut.language \rangle$

$\langle proof \rangle$

lemma *singleton_in_lang_iff*: $\langle [x] \in p5_aut.language \longleftrightarrow ok (hd [x]) \wedge [x] \in brackets \rangle$
 $\langle proof \rangle$

lemma *singleton_first_ok_iff*: $\langle p5_aut.nextl start ([x]) = first_ok \vee p5_aut.nextl start ([x]) = garbage \rangle$
 $\langle proof \rangle$

lemma *first_ok_iff*: $\langle xs \neq [] \implies p5_aut.nextl start xs = first_ok \vee p5_aut.nextl start xs = garbage \rangle$
 $\langle proof \rangle$

lemma *lang_descr*: $\langle xs \in p5_aut.language \longleftrightarrow (xs \neq [] \wedge ok (hd xs) \wedge xs \in brackets) \rangle$
 $\langle proof \rangle$

lemma *in_P5_iff*: $\langle P5 A xs \wedge xs \in brackets \longleftrightarrow (xs \neq [] \wedge ok (hd xs) \wedge xs \in brackets) \rangle$
 $\langle proof \rangle$

corollary *aux_regular*: $\langle regular \{xs. xs \neq [] \wedge ok (hd xs) \wedge xs \in brackets\} \rangle$
 $\langle proof \rangle$

lemma *regular_P5*: $\langle regular \{xs. P5 A xs \wedge xs \in brackets\} \rangle$
 $\langle proof \rangle$

end

context *locale_P*
begin

corollary *regular_Reg_inter*: $\langle regular (brackets \cap Reg A) \rangle$
 $\langle proof \rangle$

A lemma saying that all *Dyck_lang* words really only consist of brackets (trivial definition wrangling):

lemma *Dyck_lang_subset_brackets*: $\langle Dyck_lang (P \times \{One, Two\}) \subseteq brackets \rangle$
 $\langle proof \rangle$

end

5 Definitions of L, Γ, P', L'

locale *Chomsky_Schuetzenberger_locale* = *locale_P* **where** $P = P$ **for** $P :: ('n, 't) Prods$
 +
fixes $S :: 'n$

assumes CNF_P : $\langle CNF\ P \rangle$

begin

lemma $P_CNFE[dest]$:

assumes $\langle \pi \in P \rangle$

shows $\langle \exists A\ a\ B\ C. \pi = (A, [Nt\ B, Nt\ C]) \vee \pi = (A, [Tm\ a]) \rangle$

$\langle proof \rangle$

definition L **where**

$\langle L = Lang\ P\ S \rangle$

definition Γ **where**

$\langle \Gamma = P \times \{One, Two\} \rangle$

definition P' **where**

$\langle P' = transform_prod\ 'P \rangle$

definition L' **where**

$\langle L' = Lang\ P'\ S \rangle$

6 Lemmas for $P' \vdash A \Rightarrow^* x \longleftrightarrow x \in R_A \cap Dyck_lang\ \Gamma$

lemma $prod1_snds_in_tm$ $[intro, simp]$: $\langle (A, [Nt\ B, Nt\ C]) \in P \Longrightarrow snds_in_tm$

$\Gamma\ (wrap2\ A\ B\ C) \rangle$

$\langle proof \rangle$

lemma $prod2_snds_in_tm$ $[intro, simp]$: $\langle (A, [Tm\ a]) \in P \Longrightarrow snds_in_tm\ \Gamma$

$(wrap1\ A\ a) \rangle$

$\langle proof \rangle$

lemma bal_tm_wrap1 $[iff]$: $\langle bal_tm\ (wrap1\ A\ a) \rangle$

$\langle proof \rangle$

lemma bal_tm_wrap2 $[iff]$: $\langle bal_tm\ (wrap2\ A\ B\ C) \rangle$

$\langle proof \rangle$

This essentially says, that the right sides of productions are in the Dyck language of Γ , if one ignores any occurring nonterminals. This will be needed for \rightarrow .

lemma $bal_tm_transform_rhs$ $[intro!]$:

$\langle (A, \alpha) \in P \Longrightarrow bal_tm\ (transform_rhs\ A\ \alpha) \rangle$

$\langle proof \rangle$

lemma $snds_in_tm_transform_rhs$ $[intro!]$:

$\langle (A, \alpha) \in P \Longrightarrow snds_in_tm\ \Gamma\ (transform_rhs\ A\ \alpha) \rangle$

$\langle proof \rangle$

The lemma for \rightarrow

lemma *P'_imp_bal*:
assumes $\langle P' \vdash [Nt A] \Rightarrow^* x \rangle$
shows $\langle bal_tm\ x \wedge snds_in_tm\ \Gamma\ x \rangle$
 $\langle proof \rangle$

Another lemma for \rightarrow

lemma *P'_imp_Reg*:
assumes $\langle P' \vdash [Nt T] \Rightarrow^* x \rangle$
shows $\langle x \in Reg_sym\ T \rangle$
 $\langle proof \rangle$

This will be needed for the direction \leftarrow .

lemma *transform_prod_one_step*:
assumes $\langle \pi \in P \rangle$
shows $\langle P' \vdash [Nt (fst\ \pi)] \Rightarrow snd\ (transform_prod\ \pi) \rangle$
 $\langle proof \rangle$

The lemma for \leftarrow

lemma *Reg_and_dyck_imp_P'*:
assumes $\langle x \in (Reg\ A \cap Dyck_lang\ \Gamma) \rangle$
shows $\langle P' \vdash [Nt A] \Rightarrow^* map\ Tm\ x \rangle$ $\langle proof \rangle$

7 Showing $h(L') = L$

Particularly \supseteq is formally hard. To create the witness in L' we need to use the corresponding production in P' in each step. We do this by defining the transformation on the parse tree, instead of only the word. Simple induction on the derivation wouldn't (in the induction step) get us enough information on where the corresponding production needs to be applied in the transformed version.

abbreviation $\langle roots\ ts \equiv map\ root\ ts \rangle$

fun *wrap1_Sym* :: $\langle 'n \Rightarrow ('n, 't)\ sym \Rightarrow version \Rightarrow ('n, ('n, 't)\ bracket3)\ ptree\ list \rangle$
where
wrap1_Sym $A\ (Tm\ a)\ v = [Sym\ (Tm\ (Open\ ((A, [Tm\ a]), v))),\ Sym\ (Tm\ (Close\ ((A, [Tm\ a]), v)))]\ |\$
 $\langle wrap1_Sym\ ___ = [] \rangle$

fun *wrap2_Sym* :: $\langle 'n \Rightarrow ('n, 't)\ sym \Rightarrow ('n, 't)\ sym \Rightarrow version \Rightarrow ('n, ('n, 't)\ bracket3)\ ptree \Rightarrow ('n, ('n, 't)\ bracket3)\ ptree\ list \rangle$ **where**
wrap2_Sym $A\ (Nt\ B)\ (Nt\ C)\ v\ t = [Sym\ (Tm\ (Open\ ((A, [Nt\ B, Nt\ C]), v))),\ t,\ Sym\ (Tm\ (Close\ ((A, [Nt\ B, Nt\ C]), v)))]\ |\$
 $\langle wrap2_Sym\ ______ = [] \rangle$

fun *transform_tree* :: $\langle ('n, 't)\ ptree \Rightarrow ('n, ('n, 't)\ bracket3)\ ptree \rangle$ **where**
 $\langle transform_tree\ (Sym\ (Nt\ A)) = (Sym\ (Nt\ A)) \rangle\ |\$

$\langle \text{transform_tree } (\text{Sym } (Tm \ a)) = (\text{Sym } (Tm \ [^1(\text{SOME } A. \ \text{True}, [Tm \ a]))) \rangle \mid$
 $\langle \text{transform_tree } (\text{Prod } A \ [\text{Sym } (Tm \ a)]) = \text{Prod } A \ ((\text{wrap1_Sym } A \ (Tm \ a) \ \text{One}) @ (\text{wrap1_Sym } A \ (Tm \ a) \ \text{Two})) \rangle \mid$
 $\langle \text{transform_tree } (\text{Prod } A \ [t1, \ t2]) = \text{Prod } A \ ((\text{wrap2_Sym } A \ (\text{root } t1) \ (\text{root } t2) \ \text{One} \ (\text{transform_tree } t1)) @ (\text{wrap2_Sym } A \ (\text{root } t1) \ (\text{root } t2) \ \text{Two} \ (\text{transform_tree } t2))) \rangle \mid$
 $\langle \text{transform_tree } (\text{Prod } A \ y) = (\text{Prod } A \ []) \rangle$

lemma *root_of_transform_tree*[*intro, simp*]: $\langle \text{root } t = Nt \ X \implies \text{root } (\text{transform_tree } t) = Nt \ X \rangle$
 $\langle \text{proof} \rangle$

lemma *transform_tree_correct*:

assumes $\langle \text{parse_tree } P \ t \wedge \text{fringe } t = w \rangle$
shows $\langle \text{parse_tree } P' \ (\text{transform_tree } t) \wedge \text{hs } (\text{fringe } (\text{transform_tree } t)) = w \rangle$
 $\langle \text{proof} \rangle$

lemma

transfer_parse_tree:
assumes $\langle w \in \text{Ders } P \ S \rangle$
shows $\langle \exists w' \in \text{Ders } P' \ S. \ w = \text{hs } w' \rangle$
 $\langle \text{proof} \rangle$

This is essentially $h(L') \supseteq L$:

lemma *P_imp_h_L'*:

assumes $\langle w \in \text{Lang } P \ S \rangle$
shows $\langle \exists w' \in L'. \ w = h \ w' \rangle$
 $\langle \text{proof} \rangle$

This lemma is used in the proof of the other direction ($h(L') \subseteq L$):

lemma *hom_ext_inv*[*simp*]:

assumes $\langle \pi \in P \rangle$
shows $\langle \text{hs } (\text{snd } (\text{transform_prod } \pi)) = \text{snd } \pi \rangle$
 $\langle \text{proof} \rangle$

This lemma is essentially the other direction ($h(L') \subseteq L$):

lemma *L'_imp_h_P*:

assumes $\langle w' \in L' \rangle$
shows $\langle h \ w' \in \text{Lang } P \ S \rangle$
 $\langle \text{proof} \rangle$

8 The Theorem

The constructive version of the Theorem, for a grammar already in CNF:

lemma *Chomsky_Schuetzenberger_CNF*:

$\langle \text{regular } (\text{brackets} \cap \text{Reg } S) \wedge L = h \ ' \ ((\text{brackets} \cap \text{Reg } S) \cap \text{Dyck_lang } \Gamma) \wedge \text{hom_list } (h \ :: \ ('n, 't) \ \text{bracket3 } \text{list} \Rightarrow 't \ \text{list}) \rangle$

<proof>

end

Now we want to prove the theorem without assuming that P is in CNF. Of course any grammar can be converted into CNF, but this requires an infinite type of nonterminals (because the conversion to CNF may need to invent new nonterminals). Therefore we cannot just re-enter $locale_P$. Now we make all the assumption explicit.

The theorem for any grammar, but only for languages not containing ε :

lemma *Chomsky_Schuetzenberger_not_empty*:

fixes $P :: \langle 'n :: fresh0, 't \rangle Prods$ **and** $S :: 'n$

defines $\langle L \equiv Lang\ P\ S - \{\ \} \rangle$

assumes *finiteP*: $\langle finite\ P \rangle$

shows $\langle \exists (R :: ('n, 't)\ bracket3\ list\ set)\ h\ \Gamma.\ regular\ R \wedge L = h\ ' (R \cap Dyck_lang\ \Gamma) \wedge hom_list\ h \rangle$

<proof>

The Chomsky-Schützenberger theorem that we really want to prove:

theorem *Chomsky_Schuetzenberger*:

fixes $P :: \langle 'n :: fresh0, 't \rangle Prods$ **and** $S :: 'n$

defines $\langle L \equiv Lang\ P\ S \rangle$

assumes *finite*: $\langle finite\ P \rangle$

shows $\langle \exists (R :: ('n, 't)\ bracket3\ list\ set)\ h\ \Gamma.\ regular\ R \wedge L = h\ ' (R \cap Dyck_lang\ \Gamma) \wedge hom_list\ h \rangle$

<proof>

no_notation *the_hom* (h)

no_notation *the_hom_syms* (hs)

end

References

- [1] N. Chomsky and M. Schützenberger. The algebraic theory of context-free languages. In P. Braffort and D. Hirschberg, editors, *Computer Programming and Formal Systems*, volume 26 of *Studies in Logic and the Foundations of Mathematics*, pages 118–161. Elsevier, 1959.
- [2] D. Kozen. *Automata and computability*. Undergraduate texts in computer science. Springer, 1997.