

Deterministic Context-Free Languages are Closed Under Complementation (Hopcroft and Ullman)

Kaan Taskin and Tobias Nipkow

October 6, 2026

Abstract

A deterministic context-free language (DCFL) is a language accepted by a deterministic pushdown automaton (DPDA). This entry proves that the deterministic context-free languages are closed under complementation. The proof follows the one in the book by Hopcroft and Ullman (1979) [1].

Contents

1	Deterministic Pushdown Automata	1
2	DCFLs are Closed Under Complementation	3
2.1	Setup and Auxiliary Lemmas	3
2.2	Scan Construction	6
2.2.1	Definition	6
2.2.2	Determinism	7
2.2.3	Equivalence	8
2.2.4	Scan Property	18
2.3	Complement Construction	25
2.3.1	Definition	26
2.3.2	Determinism	27
2.3.3	Complementation	28

1 Deterministic Pushdown Automata

```
theory Det_Pushdown_Automata
imports Pushdown_Automata.Pushdown_Automata
begin
```

A deterministic pushdown automaton, following Hopcroft and Ullman [1]:

```
locale dpda = pda M for M :: ('q :: finite, 'a :: finite, 's :: finite) pda +
```

assumes $\delta_nonempty$: $\delta \ M \ q \ a \ X \neq \{\}$ $\longrightarrow \delta\varepsilon \ M \ q \ X = \{\}$
and $\delta_singleton$: $\delta \ M \ q \ a \ X = \{\} \vee (\exists p \ \gamma. \delta \ M \ q \ a \ X = \{(p, \gamma)\})$
and $\delta\varepsilon_singleton$: $\delta\varepsilon \ M \ q \ X = \{\} \vee (\exists p \ \gamma. \delta\varepsilon \ M \ q \ X = \{(p, \gamma)\})$
begin

In every configuration,

- $\delta_nonempty$ enforces that not both an epsilon step and a true step can be enabled.
- $\delta_singleton$ allows for at most one true step.
- $\delta\varepsilon_singleton$ allows for at most one epsilon step.

The automaton can take at most one step from any configuration:

lemma $dpda_step$: $step \ (q, w, \alpha) = \{\} \vee (\exists p \ u \ \gamma. step \ (q, w, \alpha) = \{(p, u, \gamma)\})$
proof (*cases* α)
case [*simp*]: (*Cons* $X \ \alpha'$)
show *?thesis* **proof** (*cases* w)
case *Nil*
then show *?thesis*
using $\delta\varepsilon_singleton[of \ q \ X]$ **by** *auto*
next
case [*simp*]: (*Cons* $a \ w'$)
consider (*a*) $\delta \ M \ q \ a \ X = \{\}$ | (*b*) $\delta \ M \ q \ a \ X \neq \{\}$ **by** *satx*
then show *?thesis* **proof** *cases*
case *a*
then show *?thesis*
using $\delta\varepsilon_singleton[of \ q \ X]$ **by** *auto*
next
case *b*
then show *?thesis*
using $\delta_nonempty[of \ q \ a \ X]$ $\delta_singleton[of \ q \ a \ X]$ **by** *auto*
qed
qed
qed *auto*

A step of the automaton is indeed deterministic:

lemma $dpda_step1_det$:
assumes $(q, w, \alpha) \rightsquigarrow (p, u, \gamma)$
and $(q, w, \alpha) \rightsquigarrow (p', u', \gamma')$
shows $p = p' \wedge u = u' \wedge \gamma = \gamma'$
using *assms* $dpda_step$ **by** *fastforce*

lemma $dpda_stepn_det$:
assumes $(q, w, \alpha) \rightsquigarrow(n) (p, u, \gamma)$
and $(q, w, \alpha) \rightsquigarrow(n) (p', u', \gamma')$
shows $p = p' \wedge u = u' \wedge \gamma = \gamma'$
using *assms* **proof** (*induction* $(q, w, \alpha) \ (p, u, \gamma)$ *arbitrary: q w α rule: stepn_induct*)

```

    case (stepn n q w α r u β)
    from stepn(4) have *: (r, u, β)  $\rightsquigarrow$ (n) (p', u', γ')
      using stepn_split_first[of q w α n p' u' γ'] dpda_step1_det[OF stepn(1)] by
    auto
    from stepn(3)[OF *] show ?case .
  qed auto

end

end

```

2 DCFLs are Closed Under Complementation

```

theory DPDA_Complement_HU
  imports Det_Pushdown_Automata
begin

```

2.1 Setup and Auxiliary Lemmas

```

context pda begin

```

```

definition steps1 :: 'q × 'a list × 's list  $\Rightarrow$  'q × 'a list × 's list  $\Rightarrow$  bool (infix  $\rightsquigarrow$ +
55) where
  steps1  $\equiv$  step1  $\hat{+}$ 

```

```

abbreviation no_step1 (( $\_ \rightsquigarrow$  |) [1000] 999) where
  cf  $\rightsquigarrow$  |  $\equiv$  ( $\forall$  cf'.  $\neg$ cf  $\rightsquigarrow$  cf')

```

```

lemma steps1_induct[consumes 1, case_names base step]:
  assumes x1  $\rightsquigarrow$ + x2
    and  $\bigwedge q w \alpha p u \gamma. (q, w, \alpha) \rightsquigarrow (p, u, \gamma) \Longrightarrow P (q, w, \alpha) (p, u, \gamma)$ 
    and  $\bigwedge q w \alpha r v \beta p u \gamma. (q, w, \alpha) \rightsquigarrow (r, v, \beta) \Longrightarrow (r, v, \beta) \rightsquigarrow+ (p, u, \gamma)$ 
 $\Longrightarrow$ 
     $P (r, v, \beta) (p, u, \gamma) \Longrightarrow P (q, w, \alpha) (p, u, \gamma)$ 
  shows P x1 x2
using assms[unfolded steps1_def]
proof(induction rule: converse_tranclp_induct)
  case base thus ?case by (metis prod_cases3)
next
  case step thus ?case by simp (metis prod_cases3 step1.simps)
qed

```

```

lemma steps1_converse_induct[consumes 1, case_names base step]:
  assumes x1  $\rightsquigarrow$ + x2
    and  $\bigwedge q w \alpha p u \gamma. (q, w, \alpha) \rightsquigarrow (p, u, \gamma) \Longrightarrow P (q, w, \alpha) (p, u, \gamma)$ 
    and  $\bigwedge q w \alpha r v \beta p u \gamma. (q, w, \alpha) \rightsquigarrow+ (r, v, \beta) \Longrightarrow (r, v, \beta) \rightsquigarrow (p, u, \gamma)$ 
 $\Longrightarrow$ 
     $P (q, w, \alpha) (r, v, \beta) \Longrightarrow P (q, w, \alpha) (p, u, \gamma)$ 
  shows P x1 x2

```

```

    using assms[unfolded steps1_def]
  proof(induction rule: tranclp_induct)
    case base
    then show ?case by (metis prod_cases3)
  next
    case (step)
    then show ?case by simp (metis prod_cases3 step1.simps)
  qed

lemma steps1_steps:
   $(q, w, \alpha) \rightsquigarrow^+ (p, u, \gamma) \implies (q, w, \alpha) \rightsquigarrow^* (p, u, \gamma)$ 
  by (simp add: steps_def steps1_def)

lemma steps1_step:
   $(q, w, \alpha) \rightsquigarrow (p, u, \gamma) \implies (q, w, \alpha) \rightsquigarrow^+ (p, u, \gamma)$ 
  by (simp add: steps1_def tranclp.r_into_trancl)

lemma steps1_trans:
   $(q, w, \alpha) \rightsquigarrow^+ (p, u, \gamma) \implies (p, u, \gamma) \rightsquigarrow^+ (r, v, \beta) \implies (q, w, \alpha) \rightsquigarrow^+ (r, v, \beta)$ 
  using steps1_def by force

lemma steps_steps1:
  assumes  $(q, w, \alpha) \rightsquigarrow^* (p, u, \gamma)$ 
  and  $(q, w, \alpha) \neq (p, u, \gamma)$ 
  shows  $(q, w, \alpha) \rightsquigarrow^+ (p, u, \gamma)$ 
  using assms unfolding steps_def steps1_def by (meson rtranclpD)

lemma steps1_split_last:  $(\exists r v \beta. (q, w, \alpha) \rightsquigarrow^* (r, v, \beta) \wedge (r, v, \beta) \rightsquigarrow (p, u, \gamma))$ 


$$\longleftrightarrow (q, w, \alpha) \rightsquigarrow^+ (p, u, \gamma)$$

  by (smt (verit, ccfv_threshold) rtranclp_into_tranclp1 step1.elims(2) steps1_def
  steps_def steps_refl tranclp.simps
  tranclp_into_rtranclp)

lemma split_path:
  assumes  $(q, w, \alpha) \rightsquigarrow^{(n)} (p, [], \gamma)$ 
  and  $m \leq n$ 
  shows  $\exists r u \beta. (q, w, \alpha) \rightsquigarrow^{(m)} (r, u, \beta) \wedge (r, u, \beta) \rightsquigarrow^{(n-m)} (p, [], \gamma)$ 
  using assms proof (induction  $n-m$  arbitrary:  $m$ )
    case (Suc x)
    from Suc.prem(2) consider (a)  $m=n$  | (b)  $m < n$  by linarith
    then show ?case
    proof cases
      case a
      with Suc.prem(1) show ?thesis by auto
    next
      case b
      with Suc.hyps(1)[of Suc m] Suc.hyps(2) Suc.prem(1) obtain  $r u \beta$ 
      where  $p1: (q, w, \alpha) \rightsquigarrow^{(Suc\ m)} (r, u, \beta)$  and  $p2: (r, u, \beta) \rightsquigarrow^{(n - Suc\ m)}$ 

```

```


$(p, [], \gamma)$  by fastforce  

  from  $p1$  obtain  $s\ v\ \zeta$  where  $*$ :  $(q, w, \alpha) \rightsquigarrow(m) (s, v, \zeta)$  and  $s$ :  $(s, v, \zeta) \rightsquigarrow$   

 $(r, u, \beta)$   

  using stepn_split_last[of  $m\ q\ w\ \alpha\ r\ u\ \beta$ ] by auto  

from  $s\ p2\ b$  have  $*$ :  $(s, v, \zeta) \rightsquigarrow(n-m) (p, [], \gamma)$   

using stepn_split_first[of  $s\ v\ \zeta\ n - \text{Suc}\ m\ p\ []\ \gamma$ ] Suc_diff_Suc by force  

from  $*$   $*$  show ?thesis by blast  

qed  

qed auto



end



context dpda begin



lemma max_eps_steps:  

  assumes  $(q, w, \alpha) \rightsquigarrow(n) (p, u, \gamma)$   

  and  $(p, [], \gamma) \rightsquigarrow|$   

and  $(q, w, \alpha) \rightsquigarrow(m) (r, u, \beta)$   

shows  $m \leq n$   

proof (rule ccontr)  

  assume  $\neg m \leq n$   

then have  $n\_less\_m$ :  $\text{Suc}\ n \leq m$  by simp  

from assms(1) obtain  $v$  where  $w\_def$ :  $w = v@u$   

using stepn_steps[of  $q\ w\ \alpha\ p\ u\ \gamma$ ] decreasing_word[of  $q\ w\ \alpha\ p\ u\ \gamma$ ] by blast  

from assms(3)[unfolded  $w\_def$ ] have  $p$ :  $(q, v, \alpha) \rightsquigarrow(m) (r, [], \beta)$   

using stepn_word_app[of  $m\ q\ v\ \alpha\ r\ []\ \beta\ u$ ] by simp  

obtain  $s\ y\ \zeta$  where  $*$ :  $(q, v, \alpha) \rightsquigarrow(\text{Suc}\ n) (s, y, \zeta)$   

using split_path[OF  $p\ n\_less\_m$ ] by fastforce  

from assms(1)[unfolded  $w\_def$ ] have  $*$ :  $(q, v, \alpha) \rightsquigarrow(n) (p, [], \gamma)$   

using stepn_word_app[of  $n\ q\ v\ \alpha\ p\ []\ \gamma\ u$ ] by simp  

from  $*$   $*$  have  $(p, [], \gamma) \rightsquigarrow (s, y, \zeta)$   

using stepn_split_last[of  $n\ q\ v\ \alpha\ s\ y\ \zeta$ ] dpda_stepn_det[of  $n\ q\ v\ \alpha\ p\ []\ \gamma$ ] by  

metis  

with assms(2) show False by simp  

qed



end


```

In what follows, we show that the complement of a deterministic context-free language is also a deterministic context-free language. For this purpose, we construct a deterministic pushdown automaton that recognizes the complement language of a given deterministic pushdown automaton. The proof follows that of Hopcroft and Ullman [1]. The construction is divided into two parts: First, construct an equivalent deterministic pushdown automaton that scans the entire input for all given input words, and then construct the automaton that recognizes the complement language out of this automaton.

2.2 Scan Construction

To scan the entire input, we address two complications: ensuring that every configuration has a possible step and that no configuration allows infinite epsilon steps. For the first problem, we introduce a new stack symbol to prevent the stack from becoming empty and a dead state that the automaton moves to when there are no possible steps. For the second problem, we introduce a new final state that the automaton moves to if an infinite number of epsilon steps pass through a final state; otherwise, the automaton moves to the dead state.

2.2.1 Definition

```
datatype 'q st_scan = St 'q | Q0' | D | F
datatype 's sym_scan = Sym 's | X0
```

```
lemma inj_Sym: inj Sym
  by (simp add: inj_def)
```

```
instance st_scan :: (finite) finite
```

```
proof
```

```
  have *: UNIV = {t.  $\exists q. t = St\ q$ }  $\cup$  {Q0', D, F}
```

```
    by auto (metis st_scan.exhaust)
```

```
  show finite (UNIV :: 'a st_scan set)
```

```
    by (simp add: * full_SetCompr_eq)
```

```
qed
```

```
instance sym_scan :: (finite) finite
```

```
proof
```

```
  have *: UNIV = {t.  $\exists q. t = Sym\ q$ }  $\cup$  {X0}
```

```
    by auto (metis sym_scan.exhaust)
```

```
  show finite (UNIV :: 'a sym_scan set)
```

```
    by (simp add: * full_SetCompr_eq)
```

```
qed
```

```
locale dpda_scan = dpda M for M :: ('q :: finite, 'a :: finite, 's :: finite) pda
```

```
begin
```

```
definition scan_dpda_final_states :: 'q st_scan set where
```

```
  scan_dpda_final_states  $\equiv$  St ' final_states M  $\cup$  {F}
```

```
fun scan_dpda_delta :: 'q st_scan  $\Rightarrow$  'a  $\Rightarrow$  's sym_scan  $\Rightarrow$  ('q st_scan  $\times$  's sym_scan list) set where
```

```
  scan_dpda_delta (St q) a (Sym X) = (if  $\delta\ M\ q\ a\ X = \{\}$   $\wedge$   $\delta\varepsilon\ M\ q\ X = \{\}$  then
```

```
    {(D, [Sym X])}
```

```
      else ( $\lambda(q, \alpha). (St\ q, map\ Sym\ \alpha)$ ) '  $\delta\ M\ q\ a\ X$ )
```

```
  | scan_dpda_delta (St q) _ X0 = {(D, [X0])}
```

```
  | scan_dpda_delta D _ X = {(D, [X])}
```

```
  | scan_dpda_delta _ _ _ = {\}
```

definition $eps_nonfinal :: 'q \Rightarrow 's \Rightarrow bool$ **where**

$eps_nonfinal\ q\ X \equiv (\forall i. \exists p\ \alpha. (q, [], [X]) \rightsquigarrow(i) (p, [], \alpha) \wedge p \notin final_states\ M)$

definition $eps_final :: 'q \Rightarrow 's \Rightarrow bool$ **where**

$eps_final\ q\ X \equiv (\forall i. \exists p\ \alpha. (q, [], [X]) \rightsquigarrow(i) (p, [], \alpha)) \wedge (\exists i\ p\ \alpha. (q, [], [X]) \rightsquigarrow(i) (p, [], \alpha) \wedge p \in final_states\ M)$

fun $scan_dpda_delta_eps :: 'q\ st_scan \Rightarrow 's\ sym_scan \Rightarrow ('q\ st_scan \times 's\ sym_scan\ list) \Rightarrow set$ **where**

$scan_dpda_delta_eps\ Q0'\ X0 = \{(St\ (init_state\ M), [Sym\ (init_symbol\ M), X0])\}$

$| scan_dpda_delta_eps\ (St\ q)\ (Sym\ X) = (if\ eps_nonfinal\ q\ X\ then\ \{(D, [Sym\ X])\}\ else$

$if\ eps_final\ q\ X\ then\ \{(F, [Sym\ X])\}\ else$
 $(\lambda(q, \alpha). (St\ q, map\ Sym\ \alpha))\ \delta\varepsilon\ M\ q\ X)$

$| scan_dpda_delta_eps\ F\ X = \{(D, [X])\}$

$| scan_dpda_delta_eps\ _ _ = \{\}$

definition $scan_dpda :: ('q\ st_scan, 'a, 's\ sym_scan)\ pda$ **where**

$scan_dpda =$

$(| init_state = Q0',$

$init_symbol = X0,$

$final_states = scan_dpda_final_states,$

$delta = scan_dpda_delta, delta_eps = scan_dpda_delta_eps\ |)$

2.2.2 Determinism

The automaton $scan_dpda$ is deterministic:

lemma $dpda_scan_dpda: dpda\ scan_dpda$

proof (*standard, goal_cases*)

case ($1\ p\ a\ Z$)

have $finite\ (scan_dpda_delta\ p\ a\ Z)$

by (*induction p a Z rule: scan_dpda_delta.induct*) (*auto simp: finite_delta*)

then show $?case$ **by** (*simp add: scan_dpda_def*)

next

case ($2\ p\ Z$)

have $finite\ (scan_dpda_delta_eps\ p\ Z)$

by (*induction p Z rule: scan_dpda_delta_eps.induct*) (*auto simp: finite_delta_eps*)

then show $?case$ **by** (*simp add: scan_dpda_def*)

next

case ($3\ q\ a\ X$)

have $scan_dpda_delta\ q\ a\ X \neq \{\} \longrightarrow scan_dpda_delta_eps\ q\ X = \{\}$

proof (*induction q a X rule: scan_dpda_delta.induct*)

case ($1\ q\ a\ X$)

then show $?case$ **proof**

assume $a: scan_dpda_delta\ (St\ q)\ a\ (Sym\ X) \neq \{\}$

have $*: \delta\varepsilon\ M\ q\ X = \{\}$ **proof** (*rule ccontr*)

assume $c: \delta\varepsilon\ M\ q\ X \neq \{\}$

```

    from a c have  $\delta M q a X \neq \{\}$  by simp
    hence  $\delta\varepsilon M q X = \{\}$ 
    using  $\delta\_nonempty[of q a X]$  by satx
    with c show False by satx
  qed
  hence **:  $(q, [], [X]) \rightsquigarrow$  by simp
  from ** have ***:  $\neg eps\_final q X$ 
  by (force simp: eps\_final\_def)
  from ** have ****:  $\neg eps\_nonfinal q X$ 
  by (force simp: eps\_nonfinal\_def)
  from * *** **** show  $scan\_dpda\_delta\_eps (St q) (Sym X) = \{\}$  by simp
  qed
  qed simp_all
  then show ?case by (simp add: scan\_dpda\_def)
next
  case (4 q a X)
  have  $scan\_dpda\_delta q a X = \{\} \vee (\exists p \alpha. scan\_dpda\_delta q a X = \{(p, \alpha)\})$ 
  by (induction q a X rule: scan\_dpda\_delta.induct, auto) (use  $\delta\_singleton$  in force)+
  then show ?case by (simp add: scan\_dpda\_def)
next
  case (5 q X)
  have  $scan\_dpda\_delta\_eps q X = \{\} \vee (\exists p \alpha. scan\_dpda\_delta\_eps q X = \{(p, \alpha)\})$ 
  by (induction q X rule: scan\_dpda\_delta\_eps.induct, auto) (use  $\delta\varepsilon\_singleton$  in force)+
  then show ?case by (simp add: scan\_dpda\_def)
  qed

```

2.2.3 Equivalence

sublocale $scan: dpda\ scan_dpda$
 using $dpda_scan_dpda$.

We abbreviate the definitions of $scan_dpda$ with index s :

notation $scan.step_1$ (infix $\rightsquigarrow_s$ 55)

notation $scan.steps$ (infix $\rightsquigarrow_s^*$ 55)

abbreviation $stack_with_X0 :: 's\ list \Rightarrow 's\ sym_scan\ list$ **where**
 $stack_with_X0\ \alpha \equiv map\ Sym\ \alpha\ @\ [X0]$

lemma $scan_dpda_first_step$:

assumes $(Q0', w, [X0]) \rightsquigarrow_s (q, u, \alpha)$

shows $q = St\ (init_state\ M) \wedge u = w \wedge \alpha = [Sym\ (init_symbol\ M), X0]$

using $assms\ scan.step_1_rule$ **by** (simp add: scan_dpda_def)

lemma $scan_dpda_step_from_St$:

assumes $(St\ q, w, \alpha) \rightsquigarrow_s (p, u, \gamma)$

shows $(\exists p'. p = St\ p') \vee p = F \vee p = D$

proof –

from *assms* **obtain** X **where**
 $(\exists \beta. (p, \beta) \in \text{scan_dpda_delta_eps } (St\ q)\ X) \vee (\exists a\ \beta. (p, \beta) \in \text{scan_dpda_delta } (St\ q)\ a\ X)$ **(is** $?a \vee ?b$ **)**
using *scan.step1_rule_ext scan_dpda_def* **by** *fastforce*
then consider $(a)\ ?a \mid (b)\ ?b$ **by** *blast*
thus *?thesis*
proof *cases*
case a
then show *?thesis* **by** $(\text{induction } St\ q\ X\ \text{rule: scan_dpda_delta_eps.induct})$ $(\text{auto split: if_splits})$
next
case b
then obtain a **where** $\exists \beta. (p, \beta) \in \text{scan_dpda_delta } (St\ q)\ a\ X$ **by** *blast*
then show *?thesis* **by** $(\text{induction } St\ q\ a\ X\ \text{rule: scan_dpda_delta.induct})$ $(\text{auto split: if_splits})$
qed
qed

lemma *scan_dpda_step_from_F*:
assumes $(F, w, \alpha) \rightsquigarrow_s (q, u, \gamma)$
shows $q = D$
using *assms scan.step1_rule_ext* $[of\ F\ w\ \alpha\ q\ u\ \gamma]$ *scan_dpda_def* **by** *auto*

lemma *scan_dpda_step_to_F*:
assumes $(q, w, X \# \alpha) \rightsquigarrow_s (F, u, \gamma)$
shows $u = w \wedge (\exists q' X'. q = St\ q' \wedge X = \text{Sym } X' \wedge \text{eps_final } q' X')$
proof $-$
from *assms* **have** *cases*: $(\exists \beta. u = w \wedge \gamma = \beta @ \alpha \wedge (F, \beta) \in \text{scan_dpda_delta_eps } q\ X)$
 $\vee (\exists a\ \beta. w = a \# u \wedge \gamma = \beta @ \alpha \wedge (F, \beta) \in \text{scan_dpda_delta } q\ a\ X)$ **(is** $?a \vee ?b$ **)**
using *scan.step1_rule* $[of\ q\ w\ X\ \alpha\ F\ u\ \gamma]$ *scan_dpda_def* **by** *simp*
from *cases* **consider** $(a)\ ?a \mid (b)\ ?b$ **by** *blast*
then show *?thesis*
proof *cases*
case a
then show *?thesis* **by** $(\text{induction } q\ X\ \text{rule: scan_dpda_delta_eps.induct})$ $(\text{auto split: if_splits})$
next
case b
then obtain a **where** $\exists \beta. w = a \# u \wedge \gamma = \beta @ \alpha \wedge (F, \beta) \in \text{scan_dpda_delta } q\ a\ X$ **by** *blast*
then show *?thesis* **by** $(\text{induction } q\ a\ X\ \text{rule: scan_dpda_delta.induct})$ $(\text{auto split: if_splits})$
qed
qed

lemma *scan_dpda_step_from_D*:
assumes $(D, w, \alpha) \rightsquigarrow_s (q, u, \gamma)$

shows $q = D$
using *assms* *scan.step1_rule_ext*[of $D \ w \ \alpha \ q \ u \ \gamma$] *scan_dpda_def* **by** *auto*

lemma *scan_dpda_steps_from_St*:
assumes $(St \ q, \ w, \ \alpha) \rightsquigarrow_s^* (p, \ u, \ \gamma)$
shows $(\exists p'. \ p = St \ p') \vee p = F \vee p = D$
using *assms* **by** (*induction* $(St \ q, \ w, \ \alpha) \ (p, \ u, \ \gamma)$ *arbitrary: p u γ rule: scan.steps_induct2_bw, simp*)
(use scan_dpda_step_from_St scan_dpda_step_from_F scan_dpda_step_from_D
in *blast*)

lemma *scan_dpda_step_to_St*:
assumes $(q, \ w, \ \alpha) \rightsquigarrow_s (St \ p, \ u, \ \gamma)$
shows $q = Q0' \vee (\exists q'. \ q = St \ q')$
using *assms* *scan_dpda_step_from_F*[of $w \ \alpha \ St \ p \ u \ \gamma$] *scan_dpda_step_from_D*[of $w \ \alpha \ St \ p \ u \ \gamma$]
by (*metis st_scan.exhaust st_scan.simps*(5))

lemma *scan_dpda_stack_with_X0*:
assumes $(St \ q, \ w, \ stack_with_X0 \ \alpha) \rightsquigarrow_s^* (St \ p, \ u, \ \gamma)$
shows $\exists \gamma'. \ \gamma = stack_with_X0 \ \gamma'$
using *assms* **proof** (*induction* $(St \ q, \ w, \ stack_with_X0 \ \alpha) \ (St \ p, \ u, \ \gamma)$ *arbitrary: p u γ rule: scan.steps_induct2_bw*)
case *(step r u γ v β)*
from *step(1,2)* **obtain** r' **where** $r_St[simp]: r = St \ r'$
using *scan_dpda_steps_from_St*[of $q \ w \ stack_with_X0 \ \alpha \ r \ u \ \gamma$] *scan_dpda_step_to_St*[of $r \ u \ \gamma \ p \ v \ \beta$] **by** *blast*
from *step(3)*[*OF* r_St] **obtain** γ'' **where** $\gamma_X0: \gamma = stack_with_X0 \ \gamma''$ **by** *blast*
from *step(2)* **obtain** $X \ \gamma' \ \beta'$ **where** $\gamma_def: \gamma = X \# \gamma'$ **and** $\beta_def: \beta = \beta' @ \gamma'$ **and**
cases: $(St \ p, \ \beta') \in scan_dpda_delta_eps \ (St \ r') \ X \vee (\exists a. \ (St \ p, \ \beta') \in scan_dpda_delta \ (St \ r') \ a \ X)$ **(is ?a $\vee$?b)**
using *scan.step1_rule_ext*[of $r \ u \ \gamma \ St \ p \ v \ \beta$] *scan_dpda_def* **by** *auto*
from *cases* **consider** $(a) \ ?a \mid (b) \ ?b$ **by** *blast*
then **have** $X_and_beta_def: (\exists X'. \ X = Sym \ X') \wedge (\exists \beta''. \ \beta' = map \ Sym \ \beta'')$
proof *cases*
case a
then **show** *?thesis* **by** (*induction* $St \ r' \ X$ *rule: scan_dpda_delta_eps.induct*)
(auto split: if_splits)
next
case b
then **obtain** a **where** $(St \ p, \ \beta') \in scan_dpda_delta \ (St \ r') \ a \ X$ **by** *blast*
then **show** *?thesis* **by** (*induction* $St \ r' \ a \ X$ *rule: scan_dpda_delta.induct*) *(auto split: if_splits)*
qed
from $X_and_beta_def$ [*THEN* *conjunct1*] $\gamma_X0 \ \gamma_def$ **have** $\exists \gamma'''. \ \gamma' = stack_with_X0 \ \gamma'''$
by (*metis hd_append list.sel*(1,3) *map_tl sym_scan.distinct*(1) *tl_append_if*)

with $X_and_ \beta'_def[THEN\ conjunct2]$ β_def **show** $?case$
by $(metis\ append.assoc\ map_append)$
qed $blast$

lemma $scan_dpda_trans$:
assumes $(St\ q,\ map\ Sym\ \alpha) \in \delta\ scan_dpda\ (St\ p)\ a\ (Sym\ X)$
shows $(q,\ \alpha) \in \delta\ M\ p\ a\ X$
using $assms$ **by** $(auto\ simp:\ scan_dpda_def\ inj_map_eq_map[OF\ inj_Sym]\ split:\ if_splits)$

lemma $scan_dpda_eps$:
assumes $(St\ q,\ map\ Sym\ \alpha) \in \delta\epsilon\ scan_dpda\ (St\ p)\ (Sym\ X)$
shows $(q,\ \alpha) \in \delta\epsilon\ M\ p\ X$
using $assms$ **by** $(auto\ simp:\ scan_dpda_def\ inj_map_eq_map[OF\ inj_Sym]\ split:\ if_splits)$

lemma $scan_dpda_step$:
assumes $(St\ q,\ w,\ map\ Sym\ \alpha) \rightsquigarrow_s (St\ p,\ u,\ map\ Sym\ \gamma)$
shows $(q,\ w,\ \alpha) \rightsquigarrow (p,\ u,\ \gamma)$
proof –
from $assms$ **obtain** $X\ \alpha'$ **where** $\alpha Sym_def: map\ Sym\ \alpha = Sym\ X\ \# map\ Sym\ \alpha'$ **and** **rule**:
 $(\exists \beta. u = w \wedge map\ Sym\ \gamma = \beta @ map\ Sym\ \alpha' \wedge (St\ p,\ \beta) \in \delta\epsilon\ scan_dpda\ (St\ q)\ (Sym\ X)) \vee$
 $(\exists a\ \beta. w = a\ \# u \wedge map\ Sym\ \gamma = \beta @ map\ Sym\ \alpha' \wedge (St\ p,\ \beta) \in \delta\ scan_dpda\ (St\ q)\ a\ (Sym\ X))$
using $scan.step1_rule_ext[of\ St\ q\ w\ map\ Sym\ \alpha\ St\ p\ u\ map\ Sym\ \gamma]$ **by** $auto$
from αSym_def **have** $\alpha_def: \alpha = X\ \#\alpha'$
using $inj_map_eq_map[OF\ inj_Sym]$ **by** $auto$
from **rule** **have** $(\exists \beta. u = w \wedge map\ Sym\ \gamma = map\ Sym\ \beta @ map\ Sym\ \alpha' \wedge (St\ p,\ map\ Sym\ \beta) \in \delta\epsilon\ scan_dpda\ (St\ q)\ (Sym\ X)) \vee$
 $(\exists a\ \beta. w = a\ \# u \wedge map\ Sym\ \gamma = map\ Sym\ \beta @ map\ Sym\ \alpha' \wedge (St\ p,\ map\ Sym\ \beta) \in \delta\ scan_dpda\ (St\ q)\ a\ (Sym\ X))$
using $append_eq_map_conv[where\ ?f = Sym]$ **by** $metis$
then **have** $(\exists \beta. u = w \wedge \gamma = \beta @ \alpha' \wedge (p,\ \beta) \in \delta\epsilon\ M\ q\ X) \vee (\exists a\ \beta. w = a\ \# u \wedge \gamma = \beta @ \alpha' \wedge (p,\ \beta) \in \delta\ M\ q\ a\ X)$
using $scan_dpda_trans\ scan_dpda_eps$ **by** $(metis\ inj_Sym\ inj_map_eq_map\ map_append)$
with α_def **show** $?thesis$
using $step1_rule$ **by** $simp$
qed

lemma $scan_dpda_stepX0$:
assumes $(St\ q,\ w,\ stack_with_X0\ \alpha) \rightsquigarrow_s (St\ p,\ u,\ stack_with_X0\ \gamma)$
shows $(q,\ w,\ \alpha) \rightsquigarrow (p,\ u,\ \gamma)$
proof –
from $assms$ **obtain** $X\ \alpha'$ **where** $\alpha_def: stack_with_X0\ \alpha = X\ \#\alpha'$ **and**
 $cases: (\exists \beta. (St\ p,\ \beta) \in scan_dpda_delta_eps\ (St\ q)\ X) \vee (\exists a\ \beta. (St\ p,\ \beta) \in scan_dpda_delta\ (St\ q)\ a\ X)$ **(is** $?a \vee ?b$ **)**

```

    using scan.step1_rule_ext[of St q w stack_with_X0  $\alpha$  St p u stack_with_X0
 $\gamma$ ] scan_dpda_def by force
    from cases consider (a) ?a | (b) ?b by blast
    then have  $\exists X'. X = \text{Sym } X'$ 
    proof cases
      case a
      then show ?thesis by (induction St q X rule: scan_dpda_delta_eps.induct)
    auto
  next
  case b
  then obtain a where  $\exists \beta. (St p, \beta) \in \text{scan\_dpda\_delta } (St q) a X$  by blast
  then show ?thesis by (induction St q a X rule: scan_dpda_delta.induct) auto
qed
with  $\alpha$ _def have map Sym  $\alpha \neq []$  by auto
then have  $(St q, w, \text{map Sym } \alpha) \rightsquigarrow_s (St p, u, \text{map Sym } \gamma)$ 
  using scan.step1_stack_drop[OF assms] by simp
then show ?thesis
  using scan_dpda_step by presburger
qed

```

The original automaton can mimic the steps of the automaton `scan_dpda` in the old states:

```

lemma scan_dpda_stepsX0:
  assumes  $(St q, w, \text{stack\_with\_X0 } \alpha) \rightsquigarrow_{s*} (St p, u, \text{stack\_with\_X0 } \gamma)$ 
  shows  $(q, w, \alpha) \rightsquigarrow^* (p, u, \gamma)$ 
using assms proof (induction  $(St q, w, \text{stack\_with\_X0 } \alpha) (St p, u, \text{stack\_with\_X0 } \gamma)$ 
arbitrary: p u  $\gamma$  rule: scan.steps_induct2_bw)
  case base
  then show ?case
    by (simp add: inj_Sym steps_refl)
next
  case (step r u  $\beta$  v)
  obtain r' where r_def:  $r = St r'$ 
  using scan_dpda_steps_from_St[OF step(1)] scan_dpda_step_to_St[OF step(2)]
by blast
  obtain  $\beta'$  where  $\beta\_def: \beta = \text{stack\_with\_X0 } \beta'$ 
  using scan_dpda_stack_with_X0[OF step(1)[simplified r_def]] by blast
  from step(3)[OF r_def  $\beta\_def$ ] have *:  $(q, w, \alpha) \rightsquigarrow^* (r', u, \beta')$  .
  have **:  $(r', u, \beta') \rightsquigarrow (p, v, \gamma)$ 
  using scan_dpda_stepX0[OF step(2)[simplified r_def  $\beta\_def$ ]] .
  show ?case
    using steps_trans[OF * step1_steps[OF **]] .
qed

```

If a pair of a state and a stack symbol allows infinite epsilon steps then the rest of the stack content stays untouched:

```

lemma stack_cycle_drop:
  assumes  $\forall i. \exists p \alpha. (q, [], [X]) \rightsquigarrow(i) (p, [], \alpha)$ 
  and  $(q, [], X \# \gamma) \rightsquigarrow^* (r, [], \beta)$ 

```

```

    shows  $\exists Y \beta'. \beta = Y \# \beta' @ \gamma \wedge (q, [], [X]) \rightsquigarrow^* (r, [], Y \# \beta')$ 
using assms(2) proof (induction (q, [] :: 'a list, X# $\gamma$ ) (r, [] :: 'a list,  $\beta$ ) arbitrary:
r  $\beta$  rule: steps_induct2_bw)
  case base
  then show ?case
    by (simp add: steps_refl)
next
  case (step p w  $\alpha$  r  $\beta$ )
  have w_def: w = []
  using decreasing_word[OF step(1)] by simp
  from step(3)[OF w_def] obtain Y  $\beta'$  where  $\alpha\_def: \alpha = Y \# \beta' @ \gamma$  and p1:
(q, [], [X])  $\rightsquigarrow^* (p, [], Y \# \beta')$  by blast
  from step(2)  $\alpha\_def$  obtain  $\beta''$  where  $\beta\_def: \beta = \beta'' @ \beta' @ \gamma$ 
  using step1_rule[of p w Y  $\beta' @ \gamma$  r []  $\beta$ ] by blast
  from step(2)[unfolded w_def  $\alpha\_def$   $\beta\_def$ ] have p2: (p, [], Y  $\# \beta'$ )  $\rightsquigarrow$  (r, [],
 $\beta'' @ \beta'$ )
  using step1_stack_drop[of p [] Y  $\# \beta' \gamma$  r []  $\beta'' @ \beta'$ ] by simp
  have *: (q, [], [X])  $\rightsquigarrow^* (r, [], \beta'' @ \beta')$ 
  using steps_trans[OF p1 step1_steps[OF p2]] .
  from * obtain n where p3: (q, [], [X])  $\rightsquigarrow$ (n) (r, [],  $\beta'' @ \beta'$ )
  using stepn_steps[of q [] [X] r []  $\beta'' @ \beta'$ ] by presburger
  from assms(1) obtain s  $\zeta$  where p4: (q, [], [X])  $\rightsquigarrow$ (Suc n) (s, [],  $\zeta$ ) by presburger
  from p4 have **: (r, [],  $\beta'' @ \beta'$ )  $\rightsquigarrow$  (s, [],  $\zeta$ )
  using stepn_split_last[of n q [] [X] s []  $\zeta$ ] dpda_stepn_det[OF p3] by auto
  from  $\beta\_def$  * show ?case
  using step1_nonempty_stack[OF **] by auto
qed

```

The automaton *scan_dpda* can either mimic the steps of the original automaton or detect a cycle:

```

lemma scan_dpda_steps:
  assumes (q, w,  $\alpha$ )  $\rightsquigarrow^* (p, u, \gamma)$ 
  shows (St q, w, map Sym  $\alpha$ )  $\rightsquigarrow_{s^*}^* (St p, u, map Sym \gamma) \vee$ 
    ( $\exists r X \beta. (St q, w, map Sym \alpha) \rightsquigarrow_{s^*}^* (St r, u, Sym X \# map Sym \beta) \wedge$ 
    (r, [], X# $\beta$ )  $\rightsquigarrow^* (p, [], \gamma) \wedge (\forall i. \exists s \Delta. (r, [], [X]) \rightsquigarrow(i) (s, [], \Delta)))$ 
using assms proof (induction (q, w,  $\alpha$ ) (p, u,  $\gamma$ ) arbitrary: p u  $\gamma$  rule: steps_induct2_bw)
  case base
  then show ?case
    by (simp add: scan.steps_refl)
next
  case (step p u  $\gamma$  r v  $\beta$ )
  from step(2) obtain X  $\gamma'$  where  $\gamma\_def: \gamma = X \# \gamma'$  and
    cases: ( $\exists \zeta. v = u \wedge \beta = \zeta @ \gamma' \wedge (r, \zeta) \in \delta \varepsilon M p X$ )  $\vee$  ( $\exists a \zeta. u = a \# v \wedge$ 
 $\beta = \zeta @ \gamma' \wedge (r, \zeta) \in \delta M p a X$ ) (is ?a  $\vee$  ?b)
  using step1_rule_ext[of p u  $\gamma$  r v  $\beta$ ] by blast
  from cases consider (a) ?a | (b) ?b by blast
  then show ?case
  proof cases
    case a

```

```

    then obtain  $\zeta$  where  $v\_def: u = v$  and  $\beta\_def: \beta = \zeta @ \gamma'$  and  $elem: (r, \zeta) \in \delta\varepsilon M p X$  by blast
    from  $step(3)$  consider  $(a1) (St q, w, map Sym \alpha) \rightsquigarrow_s^* (St p, u, map Sym \gamma)$ 
  |
     $(a2) \exists r X \beta. (St q, w, map Sym \alpha) \rightsquigarrow_s^* (St r, u, Sym X \# map Sym \beta) \wedge (r, [], X \# \beta) \rightsquigarrow^* (p, [], \gamma) \wedge (\forall i. \exists s \Delta. (r, [], [X]) \rightsquigarrow(i) (s, [], \Delta))$  by blast
    then show ?thesis
    proof cases
      case a1
        consider  $(a11) \forall i. \exists s \Delta. (p, [], [X]) \rightsquigarrow(i) (s, [], \Delta) \mid (a12) \neg(\forall i. \exists s \Delta. (p, [], [X]) \rightsquigarrow(i) (s, [], \Delta))$  by blast
        then show ?thesis
        proof cases
          case a11
            from  $a1[unfolded v\_def \gamma\_def]$  have *:  $(St q, w, map Sym \alpha) \rightsquigarrow_s^* (St p, v, Sym X \# map Sym \gamma')$  by simp
            from  $elem \beta\_def$  have **:  $(p, [], X \# \gamma') \rightsquigarrow^* (r, [], \beta)$ 
            using  $step1\_rule[of p [] X \gamma' r [] \beta]$   $step1\_steps$  by blast
            from * ** a11 show ?thesis by blast
          next
            case a12
              then have  $\neg eps\_nonfinal p X \wedge \neg eps\_final p X$ 
              by (auto simp:  $eps\_nonfinal\_def eps\_final\_def$ )
              with  $elem$  have  $(St r, map Sym \zeta) \in \delta\varepsilon scan\_dpda (St p) (Sym X)$ 
              by (auto simp:  $scan\_dpda\_def$ )
              with  $v\_def \gamma\_def \beta\_def$  have *:  $(St p, u, map Sym \gamma) \rightsquigarrow_s (St r, v, map Sym \beta)$ 
              using  $scan.step1\_rule[of St p u Sym X map Sym \gamma' St r u map Sym \zeta @ map Sym \gamma']$  by simp
              show ?thesis
              using  $scan.steps\_trans[OF a1 scan.step1\_steps[OF *]]$  by satr
            qed
          next
            case a2
              then obtain  $s Y \mu$  where *:  $(St q, w, map Sym \alpha) \rightsquigarrow_s^* (St s, u, Sym Y \# map Sym \mu)$  and  $spath: (s, [], Y \# \mu) \rightsquigarrow^* (p, [], \gamma)$ 
              and **:  $\forall i. \exists t \Delta. (s, [], [Y]) \rightsquigarrow(i) (t, [], \Delta)$  by blast
              from  $elem \gamma\_def \beta\_def$  have ***:  $(p, [], \gamma) \rightsquigarrow (r, [], \beta)$ 
              using  $step1\_rule[of p [] X \gamma' r [] \beta]$  by simp
              have ****:  $(s, [], Y \# \mu) \rightsquigarrow^* (r, [], \beta)$ 
              using  $steps\_trans[OF spath step1\_steps[OF ***]]$  .
              from *[ $unfolded v\_def$ ] ** **** show ?thesis by blast
            qed
          next
            case b
              then obtain  $a \zeta$  where  $u\_def: u = a \# v$  and  $\beta\_def: \beta = \zeta @ \gamma'$  and  $elem: (r, \zeta) \in \delta M p a X$  by blast
              from  $elem$  have  $eps\_empty: \delta\varepsilon M p X = \{\}$ 

```

```

    using  $\delta\_nonempty[of\ p\ a\ X]$  by blast
  from  $step(\mathcal{J})$  consider  $(t)$   $(St\ q,\ w,\ map\ Sym\ \alpha) \rightsquigarrow_s^* (St\ p,\ u,\ map\ Sym\ \gamma)$ 
    |  $(f)\ \exists r\ X\ \beta. (St\ q,\ w,\ map\ Sym\ \alpha) \rightsquigarrow_s^* (St\ r,\ u,\ Sym\ X\ \# \ map\ Sym\ \beta) \wedge$ 
 $(r,\ [],\ X\ \# \ \beta) \rightsquigarrow^* (p,\ [],\ \gamma) \wedge (\forall i. \exists s\ \Delta. (r,\ [],\ [X]) \rightsquigarrow(i)\ (s,\ [],\ \Delta))$  by blast
  then have *:  $(St\ q,\ w,\ map\ Sym\ \alpha) \rightsquigarrow_s^* (St\ p,\ u,\ map\ Sym\ \gamma)$ 
  proof cases
    case  $t$ 
    then show ?thesis .
  next
    case  $f$ 
    then obtain  $s\ Y\ \mu$  where  $f1: (s,\ [],\ Y\ \# \ \mu) \rightsquigarrow^* (p,\ [],\ \gamma)$  and  $f2: \forall i. \exists s'$ 
 $\mu'. (s,\ [],\ [Y]) \rightsquigarrow(i)\ (s',\ [],\ \mu')$  by blast
    with  $\gamma\_def$  obtain  $\gamma''$  where  $(s,\ [],\ [Y]) \rightsquigarrow^* (p,\ [],\ X\ \# \ \gamma'')$ 
      using  $stack\_cycle\_drop[OF\ f2\ f1]$  by blast
    then obtain  $n$  where  $f3: (s,\ [],\ [Y]) \rightsquigarrow(n)\ (p,\ [],\ X\ \# \ \gamma'')$ 
      using  $stepn\_steps[of\ s\ []\ [Y]\ p\ []\ X\ \# \ \gamma'']$  by presburger
    from  $eps\_empty$  have  $(p,\ [],\ X\ \# \ \gamma'') \rightsquigarrow|$  by simp
    with  $f2\ f3\ max\_eps\_steps$  have  $False$  by  $(meson\ Suc\_n\_not\_le\_n)$ 
    then show ?thesis ..
  qed
  from  $eps\_empty$  have  $(p,\ [],\ [X]) \rightsquigarrow|$  by auto
  then have  $\neg eps\_nonfinal\ p\ X \wedge \neg eps\_final\ p\ X$ 
    unfolding  $eps\_nonfinal\_def\ eps\_final\_def$  using  $step1\_stepn\_one$  by blast
  with  $elem$  have  $(St\ r,\ map\ Sym\ \zeta) \in \delta\ scan\_dpda\ (St\ p)\ a\ (Sym\ X)$ 
    by  $(auto\ simp: scan\_dpda\_def)$ 
  with  $u\_def\ \gamma\_def\ \beta\_def$  have **:  $(St\ p,\ u,\ map\ Sym\ \gamma) \rightsquigarrow_s (St\ r,\ v,\ map\ Sym\ \beta)$ 
    using  $scan.step1\_rule[of\ St\ p\ u\ Sym\ X\ map\ Sym\ \gamma'\ St\ r\ v\ map\ Sym\ \beta]$  by
  simp
  show ?thesis
    using  $scan.steps\_trans[OF\ * scan.step1\_steps[OF\ **]]$  by simp
  qed
  qed

```

The language of the automaton $scan_dpda$ and of the original automaton are the same, i.e. they are equivalent:

```

lemma lang_scan_dpda:
   $scan.accept\_final = accept\_final$ 
proof
  show  $scan.accept\_final \subseteq accept\_final$ 
  proof
    fix  $w$ 
    assume  $w \in scan.accept\_final$ 
    then obtain  $q\ \gamma$  where  $q\_final: q \in scan\_dpda\_final\_states$  and  $scan\_path:$ 
 $(Q0',\ w,\ [X0]) \rightsquigarrow_s^* (q,\ [],\ \gamma)$ 
    unfolding  $scan.accept\_final\_def$  using  $scan\_dpda\_def$  by auto
    from  $q\_final$  have cases:  $(\exists q' \in final\_states\ M. q = St\ q') \vee q = F$  (is ?a  $\vee$ 
    ?b)
    unfolding  $scan\_dpda\_final\_states\_def$  by auto

```

```

then have  $\exists p\ u\ \alpha. (Q0', w, [X0]) \rightsquigarrow_s (p, u, \alpha) \wedge (p, u, \alpha) \rightsquigarrow_{s*} (q, [], \gamma)$ 
  using scan.steps_not_refl_split_first[OF scan_path] by blast
then have  $p: (St\ (init\_state\ M), w, [Sym\ (init\_symbol\ M), X0]) \rightsquigarrow_{s*} (q, [], \gamma)$ 
  using scan_dpda_first_step by blast
from cases consider (a) ?a | (b) ?b by blast
then show  $w \in accept\_final$ 
proof cases
  case a
  then obtain  $q'$  where  $q\_def: q = St\ q'$  and  $q'\_final: q' \in final\_states\ M$ 
by blast
  from  $p[simplified\ q\_def]$  obtain  $\gamma'$  where  $\gamma\_def: \gamma = stack\_with\_X0\ \gamma'$ 
  using scan_dpda_stack_with_X0[of init_state M w [init_symbol M]]  $q'\ []$ 
] by auto
  from  $p[simplified\ q\_def\ \gamma\_def]$  have  $(init\_state\ M, w, [init\_symbol\ M]) \rightsquigarrow^* (q', [], \gamma')$ 
  using scan_dpda_stepsX0[of init_state M w [init_symbol M]]  $q'\ []\ \gamma'$  by
simp
  with  $q'\_final$  show ?thesis
  unfolding accept_final_def by blast
next
  case b
  obtain  $p\ u\ \alpha$  where  $fss: (St\ (init\_state\ M), w, [Sym\ (init\_symbol\ M), X0]) \rightsquigarrow_{s*} (p, u, \alpha)$ 
  and  $ls: (p, u, \alpha) \rightsquigarrow_s (F, [], \gamma)$ 
  using scan.steps_not_refl_split_last[OF  $p[simplified\ b]$ ] by blast
  obtain  $X\ \alpha'$  where  $\alpha\_def: \alpha = X\#\alpha'$ 
  using scan.step1_nonempty_stack[OF  $ls$ ] by blast
  obtain  $p'\ X'$  where  $u\_def: u = []$  and  $p\_def: p = St\ p'$  and  $X\_def: X = Sym\ X'$  and  $epath: eps\_final\ p'\ X'$ 
  using scan_dpda_step_to_F[OF  $ls[simplified\ \alpha\_def]$ ] by blast
  from  $fss[simplified\ p\_def]$  obtain  $\alpha''$  where  $\alpha\_with\_X0: \alpha = stack\_with\_X0\ \alpha''$ 
  using scan_dpda_stack_with_X0[of init_state M w [init_symbol M]]  $p'\ u$ 
] by auto
  from  $fss[simplified\ p\_def\ u\_def\ \alpha\_with\_X0]$  have  $*: (init\_state\ M, w, [init\_symbol\ M]) \rightsquigarrow^* (p', [], \alpha'')$ 
  using scan_dpda_stepsX0[of init_state M w [init_symbol M]]  $p'\ []\ \alpha''$  by
simp
  from  $\alpha\_def\ \alpha\_with\_X0\ X\_def$  obtain  $\alpha'''$  where  $\alpha''\_def: \alpha'' = X'\#\alpha'''$ 
  by (metis Nil_is_map_conv append_Cons append_Nil hd_Cons_tl list.map_sel(1) list.sel(1) sym_scan.distinct(1) sym_scan.inject)
  from  $epath[unfolded\ eps\_final\_def]$  obtain  $i\ r\ \beta$  where  $path: (p', [], [X^i]) \rightsquigarrow(i)\ (r, [], \beta)$  and  $r\_final: r \in final\_states\ M$  by blast
  from  $path$  have  $(p', [], [X^i]) \rightsquigarrow^* (r, [], \beta)$ 
  using stepn_steps[of  $p'\ []\ [X^i]\ r\ []\ \beta$ ] by auto
  with  $\alpha''\_def$  have  $** : (p', [], \alpha'') \rightsquigarrow^* (r, [], \beta @ \alpha''')$ 
  using steps_stack_app[of  $p'\ []\ [X^i]\ r\ []\ \beta\ \alpha'''$ ] by simp
  from  $r\_final$  show ?thesis

```

```

      unfolding accept_final_def using steps_trans[OF * **] by blast
    qed
  qed
next
  show accept_final  $\subseteq$  scan.accept_final
proof
  fix w
  assume w  $\in$  accept_final
  then obtain q  $\gamma$  where q_final:  $q \in \text{final\_states } M$  and p:  $(\text{init\_state } M, w, [\text{init\_symbol } M]) \rightsquigarrow^* (q, [], \gamma)$ 
  unfolding accept_final_def by blast
  have fs:  $(Q0', w, [X0]) \rightsquigarrow_s (St (\text{init\_state } M), w, [\text{Sym } (\text{init\_symbol } M), X0])$ 
  using scan_dpda_def scan.step1_rule by auto
  from scan_dpda_steps[OF p] consider (a)  $(St (\text{init\_state } M), w, [\text{Sym } (\text{init\_symbol } M)]) \rightsquigarrow_{s*} (St q, [], \text{map Sym } \gamma)$ 
  | (b)  $\exists r X \beta. (St (\text{init\_state } M), w, [\text{Sym } (\text{init\_symbol } M)]) \rightsquigarrow_{s*} (St r, [], \text{Sym } X \# \text{map Sym } \beta) \wedge$ 
   $(r, [], X \# \beta) \rightsquigarrow^* (q, [], \gamma) \wedge (\forall i. \exists s \Delta. (r, [], [X]) \rightsquigarrow(i) (s, [], \Delta))$  by auto
  then show  $w \in \text{scan.accept\_final}$ 
proof cases
  case a
  have  $(St (\text{init\_state } M), w, [\text{Sym } (\text{init\_symbol } M), X0]) \rightsquigarrow_{s*} (St q, [], \text{stack\_with\_X0 } \gamma)$ 
  using scan.steps_stack_app[OF a] by simp
  then have  $(Q0', w, [X0]) \rightsquigarrow_{s*} (St q, [], \text{stack\_with\_X0 } \gamma)$ 
  using scan.step1_steps[OF fs] scan.steps_trans[of  $Q0' w [X0] St (\text{init\_state } M) w [\text{Sym } (\text{init\_symbol } M), X0] St q [] \text{stack\_with\_X0 } \gamma$ ] by simp
  with q_final show ?thesis
  unfolding scan.accept_final_def using scan_dpda_def scan_dpda_final_states_def
  by auto
  next
  case b
  then obtain r X  $\beta$  where pscan:  $(St (\text{init\_state } M), w, [\text{Sym } (\text{init\_symbol } M)]) \rightsquigarrow_{s*} (St r, [], \text{Sym } X \# \text{map Sym } \beta)$  and
  pr:  $(r, [], X \# \beta) \rightsquigarrow^* (q, [], \gamma)$  and cycle:  $\forall i. \exists s \Delta. (r, [], [X]) \rightsquigarrow(i) (s, [], \Delta)$  by blast
  have *:  $(St (\text{init\_state } M), w, [\text{Sym } (\text{init\_symbol } M), X0]) \rightsquigarrow_{s*} (St r, [], \text{stack\_with\_X0 } (X \# \beta))$ 
  using scan.steps_stack_app[OF pscan] by simp
  have r_final:  $\exists \gamma'. (r, [], [X]) \rightsquigarrow^* (q, [], \gamma')$ 
  using stack_cycle_drop[OF cycle pr] by auto
  from r_final q_final have e1:  $\neg \text{eps\_nonfinal } r X$ 
  unfolding eps_nonfinal_def using stepn_steps[of r [] [X] q [] dpda_stepn_det[of
  _ r [] [X] q []]] by blast
  from cycle r_final q_final have e2:  $\text{eps\_final } r X$ 
  unfolding eps_final_def using stepn_steps[of r [] [X] q []]] by blast
  from e1 e2 have  $(St r, [], \text{Sym } X \# \text{map Sym } \beta) \rightsquigarrow_s (F, [], \text{Sym } X \# \text{map Sym } \beta)$ 

```

```

    using scan_dpda_def scan.step1_rule[of St r [] Sym X map Sym β F [] Sym
X # map Sym β] by simp
    then have **: (St r, [], stack_with_X0 (X#β))  $\rightsquigarrow_s$  (F, [], stack_with_X0
(X#β))
    using scan.steps_stack_app[of St r [] Sym X # map Sym β F [] Sym X #
map Sym β] by simp
    from fs * ** have (Q0', w, [X0])  $\rightsquigarrow_s$ * (F, [], stack_with_X0 (X # β))
    using scan.step1_steps scan.steps_trans by metis
    then show ?thesis
    unfolding scan.accept_final_def using scan_dpda_def scan_dpda_final_states_def
by auto
qed
qed
qed

```

2.2.4 Scan Property

```

lemma D_consumes: (D, w, X#α)  $\rightsquigarrow_s$ * (D, [], X#α)
proof (induction w)
  case Nil
  then show ?case
  by (simp add: scan.steps_refl)
next
  case (Cons a w)
  have (D, [X]) ∈ scan_dpda_delta D a X by simp
  then have *: (D, a#w, X#α)  $\rightsquigarrow_s$  (D, w, X#α)
  using scan.step1_rule[of D a#w X α D w X#α] by (simp add: scan_dpda_def)
  show ?case
  using scan.steps_trans[OF scan.step1_steps[OF *] Cons] .
qed

```

definition $eps_inf :: 'q \Rightarrow 's \Rightarrow bool$ **where**
 $eps_inf\ q\ X \equiv \forall i. \exists p\ \alpha. (q, [], [X]) \rightsquigarrow(i) (p, [], \alpha)$

definition $eps_infs :: 'q \Rightarrow 's\ list \Rightarrow bool$ **where**
 $eps_infs\ q\ \alpha \equiv \forall i. \exists p\ \gamma. (q, [], \alpha) \rightsquigarrow(i) (p, [], \gamma)$

If a pair of a state and a stack symbol does not allow infinite epsilon steps, then the stack symbol will be consumed:

```

lemma inf_consumes_sym:
  assumes eps_infs q (X#α)
  and  $\neg eps\_inf\ q\ X$ 
  shows  $\exists p. (q, [], X\#\alpha) \rightsquigarrow^* (p, [], \alpha)$ 
proof (rule ccontr)
  assume asm:  $\nexists p. (q, [], X\#\alpha) \rightsquigarrow^* (p, [], \alpha)$ 
  have  $\exists p\ \gamma. (q, [], X\#\alpha) \rightsquigarrow(i) (p, [], \gamma @ \alpha) \wedge \gamma \neq [] \wedge (q, [], [X]) \rightsquigarrow(i) (p, [], \gamma)$ 
  for i
  proof (induct i)
    case 0

```

```

    have (q, [], X # α)  $\rightsquigarrow$ (0) (q, [], [X] @ α)  $\wedge$  [X]  $\neq$  []  $\wedge$  (q, [], [X])  $\rightsquigarrow$ (0) (q, [],
[X]) by simp
    then show ?case by metis
next
case (Suc i)
  then obtain p γ where p: (q, [], X # α)  $\rightsquigarrow$ (i) (p, [], γ @ α) and γ_def: γ  $\neq$ 
[] and p1: (q, [], [X])  $\rightsquigarrow$ (i) (p, [], γ) by blast
  from assms(1)[unfolded eps_infs_def] obtain r β where *: (q, [], X # α)
 $\rightsquigarrow$ (Suc i) (r, [], β) by blast
  then have st: (p, [], γ @ α)  $\rightsquigarrow$  (r, [], β)
  using stepn_split_last[of i q [] X # α r [] β] dpda_stepn_det[OF p] by auto
  from γ_def obtain Y γ' where γ_def2: γ = Y # γ'
  using list.exhaust by blast
  from st[unfolded γ_def2] obtain γ'' where β_def: β = γ'' @ γ' @ α
  using step1_rule[of p [] Y γ' @ α r [] β] by auto
  from *[unfolded β_def] asm have **: γ'' @ γ'  $\neq$  []
  using stepn_steps[of q [] X # α r [] γ'' @ γ' @ α] by force
  from st[unfolded β_def] γ_def have st1: (p, [], γ)  $\rightsquigarrow$  (r, [], γ'' @ γ')
  using step1_stack_drop[of p [] γ α r [] γ'' @ γ'] by simp
  from p1 st1 have ***: (q, [], [X])  $\rightsquigarrow$ (Suc i) (r, [], γ'' @ γ') by simp
  from *[unfolded β_def] ** *** show ?case
  by (metis append.assoc)
qed
then have eps_inf q X
  by (metis eps_inf_def)
with assms(2) show False by satx
qed

lemma some_pair_inf:
  assumes eps_infs q α
  and  $\bigwedge p X \gamma. (q, [], \alpha) \rightsquigarrow^* (p, [], X \# \gamma) \longrightarrow \neg \text{eps\_inf } p X$ 
  shows False
using assms proof (induction α arbitrary: q)
  case Nil
  then show ?case
  by (force simp: eps_infs_def)
next
case (Cons X α)
  from Cons(3)[of q X α] have e:  $\neg \text{eps\_inf } q X$ 
  by (simp add: steps_refl)
  obtain p where p: (q, [], X # α)  $\rightsquigarrow^*$  (p, [], α)
  using inf_consumes_sym[OF Cons(2) e] by blast
  then obtain i where p1: (q, [], X # α)  $\rightsquigarrow$ (i) (p, [], α)
  using stepn_steps[of q [] X # α p [] α] by blast
  have *: eps_infs p α unfolding eps_infs_def proof
    fix j
    from Cons(2)[unfolded eps_infs_def] obtain r β where p2: (q, [], X # α)  $\rightsquigarrow$ (i
+ j) (r, [], β) by blast
    have (p, [], α)  $\rightsquigarrow$ (j) (r, [], β)

```

```

    using split_path[OF p2, of i] dpda_stepn_det[OF p1] by auto
    then show  $\exists r \beta. (p, [], \alpha) \rightsquigarrow(j) (r, [], \beta)$  by blast
qed
from Cons(3) have **:  $\bigwedge r Y \gamma. (p, [], \alpha) \rightsquigarrow^* (r, [], Y \# \gamma) \longrightarrow \neg \text{eps\_inf } r Y$ 
    using steps_trans[OF p] by blast
from Cons(1)[OF * **] show ?case .
qed

```

If a configuration allows infinite epsilon steps, it will eventually reach a pair of a state and a stack symbol that allows infinite epsilon steps:

lemma *inf_reaches*:

```

    assumes eps_infs q  $\alpha$ 
    shows  $\exists p X \gamma. (q, [], \alpha) \rightsquigarrow^* (p, [], X \# \gamma) \wedge \text{eps\_inf } p X$ 
using assms some_pair_inf by blast

```

lemma *eps_inf_to_D*:

```

    assumes eps_inf q X
    shows  $(St\ q, w, Sym\ X \# \alpha) \rightsquigarrow_s^* (D, [], Sym\ X \# \alpha)$ 
proof -
    from assms consider (a)  $\text{eps\_nonfinal } q X$  | (b)  $\neg \text{eps\_nonfinal } q X \wedge \text{eps\_final } q X$ 
    case a
    unfolding eps_inf_def eps_nonfinal_def eps_final_def by blast
    then show ?thesis proof cases
    case a
    then have  $(D, [Sym\ X]) \in \text{scan\_dpda\_delta\_eps } (St\ q) (Sym\ X)$  by simp
    then have st:  $(St\ q, w, Sym\ X \# \alpha) \rightsquigarrow_s (D, w, Sym\ X \# \alpha)$ 
    using scan.step1_rule[of St q w Sym X  $\alpha$  D w Sym X  $\# \alpha$ ] by (simp add: scan_dpda_def)
    show ?thesis
    using scan.steps_trans[OF scan.step1_steps[OF st] D_consumes] .
    next
    case b
    then have  $(F, [Sym\ X]) \in \text{scan\_dpda\_delta\_eps } (St\ q) (Sym\ X)$  by simp
    then have st1:  $(St\ q, w, Sym\ X \# \alpha) \rightsquigarrow_s (F, w, Sym\ X \# \alpha)$ 
    using scan.step1_rule[of St q w Sym X  $\alpha$  F w Sym X  $\# \alpha$ ] by (simp add: scan_dpda_def)
    have  $(D, [Sym\ X]) \in \text{scan\_dpda\_delta\_eps } F (Sym\ X)$  by simp
    then have st2:  $(F, w, Sym\ X \# \alpha) \rightsquigarrow_s (D, w, Sym\ X \# \alpha)$ 
    using scan.step1_rule[of F w Sym X  $\alpha$  D w Sym X  $\# \alpha$ ] by (simp add: scan_dpda_def)
    show ?thesis
    using scan.steps_trans[OF scan.step1_steps[OF st1] scan.steps_trans[OF scan.step1_steps[OF st2] D_consumes]] .
    qed
qed

```

lemma *scan_dpda_eps_inf*:

```

    assumes  $(q, w, \alpha) \rightsquigarrow^* (p, w, X \# \gamma)$ 
    and eps_inf p X

```

```

    shows  $\exists Y \beta. (St\ q, w, map\ Sym\ \alpha) \rightsquigarrow_s^* (D, [], Sym\ Y \# \beta)$ 
  proof -
    consider (a)  $(St\ q, w, map\ Sym\ \alpha) \rightsquigarrow_s^* (St\ p, w, Sym\ X \# map\ Sym\ \gamma) \mid$ 
      (b)  $\exists r\ Y\ \beta. (St\ q, w, map\ Sym\ \alpha) \rightsquigarrow_s^* (St\ r, w, Sym\ Y \# map\ Sym\ \beta)$ 
   $\wedge (\forall i. \exists s\ \zeta. (r, [], [Y]) \rightsquigarrow(i) (s, [], \zeta))$ 
    using scan_dpda_steps[OF assms(1)] by auto
    then show ?thesis proof cases
      case a
        have *:  $(St\ p, w, Sym\ X \# map\ Sym\ \gamma) \rightsquigarrow_s^* (D, [], Sym\ X \# map\ Sym\ \gamma)$ 
          using eps_inf_to_D[OF assms(2)] by simp
        show ?thesis
          using scan.steps_trans[OF a *] by blast
      next
        case b
          then obtain  $r\ Y\ \beta$  where  $p: (St\ q, w, map\ Sym\ \alpha) \rightsquigarrow_s^* (St\ r, w, Sym\ Y \#$ 
             $map\ Sym\ \beta)$  and  $r\_inf: \forall i. \exists s\ \zeta. (r, [], [Y]) \rightsquigarrow(i) (s, [], \zeta)$  by blast
          from  $r\_inf$  have  $r\_eps\_inf: eps\_inf\ r\ Y$ 
            by (simp add: eps_inf_def)
          have *:  $(St\ r, w, Sym\ Y \# map\ Sym\ \beta) \rightsquigarrow_s^* (D, [], Sym\ Y \# map\ Sym\ \beta)$ 
            using eps_inf_to_D[OF  $r\_eps\_inf$ ] by simp
          show ?thesis
            using scan.steps_trans[OF p *] by blast
    qed
  qed

lemma scan_dpda_neps_infs:
  assumes  $\neg eps\_infs\ q\ \alpha$ 
  shows  $\exists p\ \gamma. (St\ q, w, map\ Sym\ \alpha) \rightsquigarrow_s^* (St\ p, w, map\ Sym\ \gamma) \wedge (\forall X\ \gamma'. \gamma =$ 
 $X \# \gamma' \longrightarrow \delta\varepsilon\ M\ p\ X = \{\})$ 
  proof -
    from assms[unfolded eps_infs_def] obtain  $i$  where  $\forall p\ \gamma. \neg(q, [], \alpha) \rightsquigarrow(i) (p,$ 
 $[], \gamma)$  by blast
    then have  $\exists j\ p\ \gamma. j < i \wedge (q, [], \alpha) \rightsquigarrow(j) (p, [], \gamma) \wedge (p, [], \gamma) \rightsquigarrow \mid$  proof
      (induction  $i$ )
      case 0
        then show ?case
          by (auto simp: steps_refl)
      next
        case (Suc  $i$ )
        consider (a)  $\forall p\ \gamma. \neg(q, [], \alpha) \rightsquigarrow(i) (p, [], \gamma) \mid$  (b)  $\exists p\ \gamma. (q, [], \alpha) \rightsquigarrow(i) (p, [],$ 
 $\gamma)$  by blast
        then show ?case proof cases
          case a
            from Suc(1)[OF a] show ?thesis
              by (auto simp: less_Suc_eq)
          next
            case b
              then obtain  $p\ \gamma$  where *:  $(q, [], \alpha) \rightsquigarrow(i) (p, [], \gamma)$  by blast
              with Suc(2) have **:  $(p, [], \gamma) \rightsquigarrow \mid$ 

```

```

    using decreasing_word step1_steps by fastforce
    from * ** show ?thesis by blast
qed
qed
then obtain j p γ where j < i and p: (q, [], α) ~~~(j) (p, [], γ) and nst: (p, [],
γ) ~~~| by blast
from p have p1: (q, [], α) ~~~* (p, [], γ)
  using stepn_steps[of q [] α p [] γ] by auto
consider (a) (St q, [], map Sym α) ~~~_s* (St p, [], map Sym γ) |
  (b) ∃ r X β. (St q, [], map Sym α) ~~~_s* (St r, [], Sym X # map Sym β)
  ∧ (∀ i. ∃ s Δ. (r, [], [X]) ~~~(i) (s, [], Δ))
  using scan_dpda_steps[OF p1] by blast
then show ?thesis proof cases
  case a
  then have *: (St q, w, map Sym α) ~~~_s* (St p, w, map Sym γ)
    using scan.steps_word_app[of St q [] map Sym α St p [] map Sym γ w] by
simp
  from nst have **: ∀ X γ'. γ = X # γ' → δε M p X = {} by auto
  from * ** show ?thesis by blast
next
  case b
  then obtain r X β where p2: (St q, [], map Sym α) ~~~_s* (St r, [], Sym X #
map Sym β) and cycle: ∀ i. ∃ s Δ. (r, [], [X]) ~~~(i) (s, [], Δ) by blast
  from p2 have p3: (St q, [], stack_with_X0 α) ~~~_s* (St r, [], stack_with_X0
(X # β))
    using scan.steps_stack_app[where ?β = [X0]] by fastforce
  have (q, [], α) ~~~* (r, [], X # β)
    using scan_dpda_stepsX0[OF p3] .
  then obtain n where p4: (q, [], α) ~~~(n) (r, [], X # β)
    using stepn_steps[of q [] α r [] X # β] by blast
  have ∀ k. ∃ s Δ. (q, [], α) ~~~(k) (s, [], Δ) proof
    fix k
    show ∃ s Δ. (q, [], α) ~~~(k) (s, [], Δ) proof (cases k < n)
      case True
      then have k_leq: k ≤ n by simp
      then obtain s u ζ where *: (q, [], α) ~~~(k) (s, u, ζ)
        using split_path[OF p4 k_leq] by blast
      from * have u_def: u = []
        using stepn_steps[of q [] α s u ζ] decreasing_word[of q [] α s u ζ] by auto
      from * [unfolded u_def] show ?thesis by blast
    next
      case False
      then have k_beq: k ≥ n by simp
      with cycle obtain s ζ where (r, [], [X]) ~~~(k-n) (s, [], ζ) by presburger
      then have p5: (r, [], X # β) ~~~(k-n) (s, [], ζ @ β)
        using stepn_stack_app[where ?β = β] by fastforce
      from k_beq show ?thesis
        using stepn_trans[OF p4 p5] by auto
    qed
  qed

```

```

qed
with assms show ?thesis
  by (simp add: eps_infs_def)
qed
qed

lemma scan_dpda_scans_St:
 $\exists p X \gamma. (St\ q, w, stack\_with\_X0\ \alpha) \rightsquigarrow_s^* (p, [], X\#\gamma) \wedge \delta\ scan\_dpda\ p\ a\ X \neq \{\}$ 
proof (induction w arbitrary: q  $\alpha$ )
  case Nil
  then show ?case proof (cases eps_infs q  $\alpha$ )
    case True
    obtain p X  $\gamma$  where p:  $(q, [], \alpha) \rightsquigarrow_s^* (p, [], X\#\gamma)$  and eps_p:  $eps\_inf\ p\ X$ 
    using inf_reaches[OF True] by blast
    obtain Y  $\beta$  where  $(St\ q, [], map\ Sym\ \alpha) \rightsquigarrow_s^* (D, [], Sym\ Y\ \#\ \beta)$ 
    using scan_dpda_eps_inf[OF p eps_p] by blast
    then have *:  $(St\ q, [], stack\_with\_X0\ \alpha) \rightsquigarrow_s^* (D, [], Sym\ Y\ \#\ \beta @ [X0])$ 
    using scan.steps_stack_app[where ? $\beta = [X0]$ ] by fastforce
    have **:  $\delta\ scan\_dpda\ D\ a\ (Sym\ Y) \neq \{\}$ 
    by (simp add: scan_dpda_def)
    from * ** show ?thesis by blast
  next
  case False
  obtain p  $\gamma$  where p:  $(St\ q, [], map\ Sym\ \alpha) \rightsquigarrow_s^* (St\ p, [], map\ Sym\ \gamma)$  and
 $\gamma\_nempty: (\forall X\ \gamma'. \gamma = X\#\gamma' \longrightarrow \delta\varepsilon\ M\ p\ X = \{\})$ 
  using scan_dpda_neps_infs[OF False] by blast
  from p have p1:  $(St\ q, [], stack\_with\_X0\ \alpha) \rightsquigarrow_s^* (St\ p, [], stack\_with\_X0\ \gamma)$ 
  using scan.steps_stack_app[where ? $\beta = [X0]$ ] by simp
  show ?thesis proof (cases  $\gamma$ )
    case Nil
    with p1 show ?thesis
    by (force simp: scan_dpda_def)
  next
  case (Cons X  $\gamma'$ )
  with  $\gamma\_nempty$  have  $\delta\varepsilon\ M\ p\ X = \{\}$  by simp
  then have  $\delta\ scan\_dpda\ (St\ p)\ a\ (Sym\ X) \neq \{\}$ 
  by (simp add: scan_dpda_def)
  with p1 Cons show ?thesis by auto
  qed
qed
next
case IH: (Cons b w)
show ?case proof (cases eps_infs q  $\alpha$ )
  case True
  obtain p X  $\gamma$  where p:  $(q, [], \alpha) \rightsquigarrow_s^* (p, [], X\#\gamma)$  and eps_p:  $eps\_inf\ p\ X$ 
  using inf_reaches[OF True] by blast
  from p have p1:  $(q, b\#\!w, \alpha) \rightsquigarrow_s^* (p, b\#\!w, X\#\gamma)$ 
  using steps_word_app[of q []  $\alpha$  p []  $X\#\gamma\ b\#\!w$ ] by simp

```

```

obtain  $Y \beta$  where  $(St\ q, b \# w, map\ Sym\ \alpha) \rightsquigarrow_{s*} (D, [], Sym\ Y \# \beta)$ 
using  $scan\_dpda\_eps\_inf[OF\ p1\ eps\_p]$  by blast
then have  $*$ :  $(St\ q, b \# w, stack\_with\_X0\ \alpha) \rightsquigarrow_{s*} (D, [], Sym\ Y \# \beta @ [X0])$ 
using  $scan.steps\_stack\_app[where\ ?\beta = [X0]]$  by fastforce
have  $**$ :  $\delta\ scan\_dpda\ D\ a\ (Sym\ Y) \neq \{\}$ 
by  $(simp\ add: scan\_dpda\_def)$ 
from  $*$  show  $?thesis$  by blast
next
case False
obtain  $p\ \gamma$  where  $p: (St\ q, b \# w, map\ Sym\ \alpha) \rightsquigarrow_{s*} (St\ p, b \# w, map\ Sym\ \gamma)$ 
and  $\gamma\_empty: (\forall\ X\ \gamma'. \gamma = X \# \gamma' \longrightarrow \delta\epsilon\ M\ p\ X = \{\})$ 
using  $scan\_dpda\_neps\_infs[OF\ False]$  by blast
from  $p$  have  $p1: (St\ q, b \# w, stack\_with\_X0\ \alpha) \rightsquigarrow_{s*} (St\ p, b \# w, stack\_with\_X0\ \gamma)$ 
using  $scan.steps\_stack\_app[where\ ?\beta = [X0]]$  by simp
show  $?thesis$  proof  $(cases\ \gamma)$ 
case Nil
have  $(D, [X0]) \in scan\_dpda\_delta\ (St\ p)\ b\ X0$  by simp
with Nil have  $st: (St\ p, b \# w, stack\_with\_X0\ \gamma) \rightsquigarrow_s (D, w, [X0])$ 
using  $scan.step1\_rule[of\ St\ p\ b \# w\ X0\ []\ D\ w\ [X0]]$  by  $(simp\ add: scan\_dpda\_def)$ 
have  $*$ :  $(St\ q, b \# w, stack\_with\_X0\ \alpha) \rightsquigarrow_{s*} (D, [], [X0])$ 
using  $scan.steps\_trans[OF\ p1\ scan.steps\_trans[OF\ scan.step1\_steps[OF\ st]\ D\_consumes]]$  .
have  $**$ :  $\delta\ scan\_dpda\ D\ a\ X0 \neq \{\}$ 
by  $(simp\ add: scan\_dpda\_def)$ 
from  $*$  show  $?thesis$  by blast
next
case  $(Cons\ X\ \gamma')$ 
with  $\gamma\_empty$  have  $\delta\epsilon\ M\ p\ X = \{\}$  by simp
then consider  $(a)\ (D, [Sym\ X]) \in scan\_dpda\_delta\ (St\ p)\ b\ (Sym\ X) \mid (b)$ 
 $\exists\ r\ \beta. (St\ r, map\ Sym\ \beta) \in scan\_dpda\_delta\ (St\ p)\ b\ (Sym\ X)$ 
by  $(cases\ \delta\ M\ p\ b\ X = \{\})\ fastforce+$ 
then show  $?thesis$  proof cases
case  $a$ 
with Cons have  $st: (St\ p, b \# w, stack\_with\_X0\ \gamma) \rightsquigarrow_s (D, w, stack\_with\_X0\ \gamma)$ 
using  $scan.step1\_rule[of\ St\ p\ b \# w\ Sym\ X\ stack\_with\_X0\ \gamma'\ D\ w\ stack\_with\_X0\ \gamma]$  by  $(simp\ add: scan\_dpda\_def)$ 
from Cons have  $*$ :  $(St\ q, b \# w, stack\_with\_X0\ \alpha) \rightsquigarrow_{s*} (D, [], stack\_with\_X0\ \gamma)$ 
using  $scan.steps\_trans[OF\ p1\ scan.steps\_trans[OF\ scan.step1\_steps[OF\ st]]\ D\_consumes[of\ w\ Sym\ X\ stack\_with\_X0\ \gamma']]$  by simp
have  $**$ :  $\delta\ scan\_dpda\ D\ a\ (Sym\ X) \neq \{\}$ 
by  $(simp\ add: scan\_dpda\_def)$ 
from  $*$  show  $?thesis$  by auto
next
case  $b$ 
then obtain  $r\ \beta$  where  $(St\ r, map\ Sym\ \beta) \in scan\_dpda\_delta\ (St\ p)\ b$ 

```

```

(Sym X) by blast
with Cons have st: (St p, b#w, stack_with_X0 γ)  $\rightsquigarrow_s$  (St r, w, stack_with_X0
(β@γ'))
  using scan.step1_rule[of St p b#w Sym X stack_with_X0 γ' St r w
stack_with_X0 (β@γ')] by (simp add: scan_dpda_def)
  from IH have *: ∃ p X γ. (St r, w, stack_with_X0 (β@γ'))  $\rightsquigarrow_{s*}$  (p, [], X
# γ) ∧ δ scan_dpda p a X ≠ {} by presburger
  from * show ?thesis
    using scan.steps_trans[OF scan.steps_trans[OF p1 scan.step1_steps[OF
st]]] by blast
  qed
  qed
  qed
  qed

```

For every input word, the automaton *scan_dpda* scans the entire input and, moreover, ends in a configuration where no epsilon step is possible:

```

lemma scan_dpda_scans:
  ∃ q X α. (init_state scan_dpda, w, [init_symbol scan_dpda])  $\rightsquigarrow_{s*}$  (q, [], X#α) ∧
  δ scan_dpda q a X ≠ {}
proof -
  have (St (init_state M), stack_with_X0 [init_symbol M]) ∈ scan_dpda_delta_eps
  Q0' X0 by simp
  then have *: (init_state scan_dpda, w, [init_symbol scan_dpda])  $\rightsquigarrow_s$  (St (init_state
M), w, stack_with_X0 [init_symbol M])
    using scan.step1_rule[of Q0' w X0 [] St (init_state M) w stack_with_X0
[init_symbol M]] by (simp add: scan_dpda_def)
  obtain p X γ where **: (St (init_state M), w, stack_with_X0 [init_symbol
M])  $\rightsquigarrow_{s*}$  (p, [], X#γ) and d: δ scan_dpda p a X ≠ {}
    using scan_dpda_scans_St[of init_state M w [init_symbol M]] by blast
  have (init_state scan_dpda, w, [init_symbol scan_dpda])  $\rightsquigarrow_{s*}$  (p, [], X#γ)
    using scan.steps_trans[OF scan.step1_steps[OF *] **] .
  with d show ?thesis by blast
qed
end

```

2.3 Complement Construction

After ensuring that the automaton scans its entire input word, we are left with constructing the deterministic pushdown automaton that recognizes the complement language. The main idea is to keep track of whether a final state has been visited since the last true step using a second component for states. If no final state has been visited, the complement automaton enters a final state of its own just before the next true step.

2.3.1 Definition

An S1-state indicates that a final state has been visited since the last true step, whereas an S2-state indicates that no final state has been visited since the last true step. S3-states are the final states of the complement automaton:

datatype 'q s123 = S1 'q | S2 'q | S3 'q

instance s123 :: (finite) finite

proof

have *: UNIV = {t. $\exists q. t = S1\ q$ } $\cup$ {t. $\exists q. t = S2\ q$ } $\cup$ {t. $\exists q. t = S3\ q$ }
 by auto (metis s123.exhaust)
 show finite (UNIV :: 'a s123 set)
 by (simp add: * full_SetCompr_eq)

qed

lemma inj_S1: inj S1

by (simp add: inj_def)

lemma inj_S2: inj S2

by (simp add: inj_def)

We can now assume the scan property proved in the last subsection:

locale complement_dpda = dpda M **for** M :: ('q :: finite, 'a :: finite, 's :: finite)
 pda +

assumes M_path: $\exists q\ X\ \alpha. (init_state\ M, w, [init_symbol\ M]) \rightsquigarrow^* (q, [], X\#\alpha)$
 $\wedge \delta\ M\ q\ a\ X \neq \{\}$

begin

definition comp_dpda_init_state :: 'q s123 **where**

comp_dpda_init_state $\equiv$ if init_state M $\in$ final_states M then S1 (init_state M) else S2 (init_state M)

definition comp_dpda_final_states :: 'q s123 set **where**

comp_dpda_final_states $\equiv$ range S3

fun comp_dpda_delta :: 'q s123 $\Rightarrow$ 'a $\Rightarrow$'s $\Rightarrow$ ('q s123 $\times$'s list) set **where**

comp_dpda_delta (S1 q) a X = ($\lambda(p, \alpha). \text{if } p \in \text{final_states } M \text{ then } (S1\ p, \alpha)$
 else (S2 p, α)) ' $\delta\ M\ q\ a\ X$
 | comp_dpda_delta (S3 q) a X = ($\lambda(p, \alpha). \text{if } p \in \text{final_states } M \text{ then } (S1\ p, \alpha)$
 else (S2 p, α)) ' $\delta\ M\ q\ a\ X$
 | comp_dpda_delta _ _ _ = $\{\}$

fun comp_dpda_delta_eps :: 'q s123 $\Rightarrow$'s $\Rightarrow$ ('q s123 $\times$'s list) set **where**

comp_dpda_delta_eps (S1 q) X = ($\lambda(p, \alpha). (S1\ p, \alpha)$) ' $\delta\varepsilon\ M\ q\ X$
 | comp_dpda_delta_eps (S2 q) X = ($\lambda(p, \alpha). \text{if } p \in \text{final_states } M \text{ then } (S1\ p, \alpha)$
 else (S2 p, α)) ' $\delta\varepsilon\ M\ q\ X$
 $\cup (\text{if } \exists a. \delta\ M\ q\ a\ X \neq \{\} \text{ then } \{(S3\ q, [X])\} \text{ else } \{\})$
 | comp_dpda_delta_eps _ _ = $\{\}$

definition *comp_dpda* :: ('q s123, 'a, 's) pda **where**

```

comp_dpda =
  (| init_state = comp_dpda_init_state,
    init_symbol = init_symbol M,
    final_states = comp_dpda_final_states,
    delta = comp_dpda_delta, delta_eps = comp_dpda_delta_eps |)

```

2.3.2 Determinism

lemma *image_singleton_if_inj*:

assumes *inj* *f*

shows $(\exists x. A = \{x\}) \longleftrightarrow (\exists x. f \cdot A = \{x\})$

using *assms* **by** (*metis image_empty image_insert image_inv_f*)

The automaton *comp_dpda* is deterministic:

lemma *dpda_comp_dpda*: *dpda comp_dpda*

proof (*standard, goal_cases*)

case (1 *p a Z*)

have *finite* (*comp_dpda_delta* *p a Z*)

by (*induction* *p a Z* *rule: comp_dpda_delta.induct*) (*auto simp: finite_delta*)

then show ?*case*

by (*simp add: comp_dpda_def*)

next

case (2 *p Z*)

have *finite* (*comp_dpda_delta_eps* *p Z*)

by (*induction* *p Z* *rule: comp_dpda_delta_eps.induct*) (*auto simp: finite_delta_eps*)

then show ?*case*

by (*simp add: comp_dpda_def*)

next

case (3 *q a X*)

then show ?*case*

proof

assume $\delta \text{ comp_dpda } q \ a \ X \neq \{\}$

then have *comp_dpda_delta* *q a X* $\neq \{\}$

by (*simp add: comp_dpda_def*)

then have *comp_dpda_delta_eps* *q a X* $= \{\}$

by (*induction* *q a X* *rule: comp_dpda_delta.induct*) (*auto simp: delta_nonempty*)

then show $\delta \varepsilon \text{ comp_dpda } q \ a \ X = \{\}$

by (*simp add: comp_dpda_def*)

qed

next

case (4 *q a X*)

let ?*f* = $(\lambda(p, \alpha). \text{ if } p \in \text{final_states } M \text{ then } (S1 \ p, \alpha) \text{ else } (S2 \ p, \alpha))$

have *: *inj* ?*f*

by (*simp add: inj_def*)

have *comp_dpda_delta* *q a X* $= \{\}$ $\vee (\exists p \ \gamma. \text{ comp_dpda_delta } q \ a \ X = \{(p, \gamma)\})$

proof (*induction* *q a X* *rule: comp_dpda_delta.induct*)

case (1 *q a X*)

```

    then show ?case
    using  $\delta\_singleton[of\ q\ a\ X]$   $image\_singleton\_if\_inj[OF\ *,\ of\ \delta\ M\ q\ a\ X]$  by
simp
  next
    case (2  $q\ a\ X$ )
    then show ?case
    using  $\delta\_singleton[of\ q\ a\ X]$   $image\_singleton\_if\_inj[OF\ *,\ of\ \delta\ M\ q\ a\ X]$  by
simp
    qed simp
    then show ?case
    by (simp add:  $comp\_dpda\_def$ )
  next
    case (5  $q\ X$ )
    have  $comp\_dpda\_delta\_eps\ q\ X = \{\}$   $\vee (\exists p\ \gamma. comp\_dpda\_delta\_eps\ q\ X = \{(p, \gamma)\})$ 
    proof (induction  $q\ X$  rule:  $comp\_dpda\_delta\_eps.induct$ )
    case (1  $q\ X$ )
    then show ?case
    using  $\delta\epsilon\_singleton[of\ q\ X]$  by auto
  next
    case (2  $q\ X$ )
    consider (a)  $\exists a. \delta\ M\ q\ a\ X \neq \{\}$  | (b)  $\neg(\exists a. \delta\ M\ q\ a\ X \neq \{\})$  by blast
    then show ?case
    proof cases
    case a
    then have  $\delta\epsilon\ M\ q\ X = \{\}$ 
    using  $\delta\_nonempty[of\ q\ X]$  by blast
    then show ?thesis by simp
  next
    case b
    let ?f =  $(\lambda(p, \alpha). if\ p \in final\_states\ M\ then\ (S1\ p, \alpha)\ else\ (S2\ p, \alpha))$ 
    have *:  $inj\ ?f$ 
    by (simp add:  $inj\_def$ )
    from b show ?thesis
    using  $\delta\epsilon\_singleton[of\ q\ X]$   $image\_singleton\_if\_inj[OF\ *,\ of\ \delta\epsilon\ M\ q\ X]$  by
simp
    qed
    qed simp
    then show ?case
    by (simp add:  $comp\_dpda\_def$ )
  qed

```

2.3.3 Complementation

sublocale $comp: dpda\ comp_dpda$
 using $dpda_comp_dpda$.

We abbreviate the definitions of $comp_dpda$ with index c :

notation $comp.step_1$ (**infix** $\rightsquigarrow_c$ 55)
notation $comp.steps$ (**infix** $\rightsquigarrow_{c*}$ 55)

```

notation comp.stepsn (( $\_ / \rightsquigarrow_c'(\_)/ \_$ ) [55, 0, 55] 55)
notation comp.steps1 (infix  $\rightsquigarrow_c + 55$ )

lemma comp_dpda_step_from_S1:
  assumes (S1 q, [],  $\alpha$ )  $\rightsquigarrow_c$  (p, [],  $\gamma$ )
  shows  $\exists p'. p = S1\ p'$ 
using assms comp.step1_rule_ext[of S1 q []  $\alpha$  p []  $\gamma$ ] by (auto simp: comp_dpda_def)

lemma comp_dpda_steps_from_S1:
  assumes (S1 q, [],  $\alpha$ )  $\rightsquigarrow_{c*}$  (p, [],  $\gamma$ )
  shows  $\exists p'. p = S1\ p'$ 
using assms comp_dpda_step_from_S1 comp.decreasing_word
  by (induction (S1 q, [] :: 'a list,  $\alpha$ ) (p, [] :: 'a list,  $\gamma$ ) arbitrary: p  $\gamma$  rule: comp.steps_induct2_bw) fastforce+)

lemma comp_dpda_nonfinal_stepsS2:
  assumes (q, [],  $\alpha$ )  $\rightsquigarrow_{c*}$  (p, [],  $\gamma$ )
  and  $\bigwedge r\ \beta. (q, [], \alpha) \rightsquigarrow_{c*} (r, [], \beta) \longrightarrow r \notin \text{final\_states } M$ 
  shows (S2 q, [],  $\alpha$ )  $\rightsquigarrow_{c*}$  (S2 p, [],  $\gamma$ )
using assms proof (induction (q, [] :: 'a list,  $\alpha$ ) (p, [] :: 'a list,  $\gamma$ ) arbitrary: q  $\alpha$  rule: steps_induct2)
  case 1
  then show ?case
    by (simp add: comp.steps_refl)
next
  case (2 q  $\alpha$  r u  $\beta$ )
  from 2(1) obtain X  $\alpha'$   $\zeta$  where  $\alpha\_def: \alpha = X\#\alpha'$  and  $u\_def: u = []$  and *:
 $\beta = \zeta @ \alpha'$  and elem: (r,  $\zeta$ )  $\in \delta\varepsilon\ M\ q\ X$ 
  using step1_rule_ext[of q []  $\alpha$  r u  $\beta$ ] by blast
  from 2(4)[of r  $\beta$ ] have  $r \notin \text{final\_states } M$ 
  using step1_steps[OF 2(1)[unfolded u_def]] by simp
  with elem have (S2 r,  $\zeta$ )  $\in \text{comp\_dpda\_delta\_eps } (S2\ q)\ X$  by force
  with  $\alpha\_def\ *$  have **: (S2 q, [],  $\alpha$ )  $\rightsquigarrow_c$  (S2 r, [],  $\beta$ )
  using comp.step1_rule[of S2 q []  $X\ \alpha'\ S2\ r$  []  $\beta$ ] by (simp add: comp_dpda_def)
  from 2(4) have nr:  $\bigwedge s\ \mu. (r, [], \beta) \rightsquigarrow_{c*} (s, [], \mu) \longrightarrow s \notin \text{final\_states } M$ 
  using steps_trans[OF step1_steps[OF 2(1)], unfolded u_def] by blast
  from 2(3)[OF u_def nr] have ***: (S2 r, [],  $\beta$ )  $\rightsquigarrow_{c*}$  (S2 p, [],  $\gamma$ ) .
  show ?case
    using comp.steps_trans[OF comp.step1_steps[OF **] ***] .
qed

```

The automaton *comp_dpda* mimics the steps of the original automaton in S2-states, provided that the word read so far is not accepted:

```

lemma comp_dpda_nonfinal_steps:
  assumes (q', w,  $\alpha$ )  $\rightsquigarrow_{c*}$  (p, u,  $\gamma$ )
  and  $w \neq u$ 
  and  $\bigwedge r\ \beta. (q', w, \alpha) \rightsquigarrow_{c*} (r, u, \beta) \longrightarrow r \notin \text{final\_states } M$ 
  and  $q = S1\ q' \vee q = S2\ q'$ 
  shows (q, w,  $\alpha$ )  $\rightsquigarrow_{c*}$  (S2 p, u,  $\gamma$ )

```

```

using assms proof (induction ( $q'$ ,  $w$ ,  $\alpha$ ) ( $p$ ,  $u$ ,  $\gamma$ ) arbitrary:  $q \ q' \ w \ \alpha$  rule:
steps_induct2)
  case ( $2 \ q' \ w \ \alpha \ r \ v \ \beta$ )
    from  $2(1)$  obtain  $X \ \alpha'$  where  $\alpha\_def$ :  $\alpha = X \# \alpha'$  and cases:
      ( $\exists \zeta. v = w \wedge \beta = \zeta @ \alpha' \wedge (r, \zeta) \in \delta\varepsilon \ M \ q' \ X$ )  $\vee$  ( $\exists a \ \zeta. w = a \# v \wedge \beta =$ 
 $\zeta @ \alpha' \wedge (r, \zeta) \in \delta \ M \ q' \ a \ X$ ) (is  $?a \vee ?b$ )
      using step1_rule_ext[of  $q' \ w \ \alpha \ r \ v \ \beta$ ] by blast
      from  $2(5)$  have  $rp\_nonfinal$ :  $\bigwedge ra \ \beta'. (r, v, \beta) \rightsquigarrow^* (ra, u, \beta') \longrightarrow ra \notin fi$ -
 $nal\_states \ M$ 
      using steps_trans[OF step1_steps[OF  $2(1)$ ]] by blast
      from cases consider ( $a$ )  $?a$  | ( $b$ )  $?b$  by blast
      then show  $?case$ 
    proof cases
      case  $a$ 
        then obtain  $\zeta$  where  $*$ :  $v = w$  and  $**$ :  $\beta = \zeta @ \alpha'$  and elem:  $(r, \zeta) \in \delta\varepsilon \ M$ 
 $q' \ X$  by blast
        from elem  $2(6)$  have  $(S1 \ r, \zeta) \in comp\_dpda\_delta\_eps \ q \ X \vee (S2 \ r, \zeta) \in$ 
 $comp\_dpda\_delta\_eps \ q \ X$  by force
        with  $* \ ** \ \alpha\_def$  have  $***$ :  $(q, w, \alpha) \rightsquigarrow_c (S1 \ r, v, \beta) \vee (q, w, \alpha) \rightsquigarrow_c (S2 \ r,$ 
 $v, \beta)$ 
        using comp.step1_rule[of  $q \ w \ X \ \alpha' \ _ \ v \ \beta$ ] by (simp add: comp_dpda_def)
        from  $2(4) \ *$  have  $v\_neq$ :  $v \neq u$  by simp
        from  $2(3)$ [OF  $v\_neq \ rp\_nonfinal$ ] have  $****$ :  $(S1 \ r, v, \beta) \rightsquigarrow_{c*} (S2 \ p, u, \gamma)$ 
by simp
        from  $2(3)$ [OF  $v\_neq \ rp\_nonfinal$ ] have  $*****$ :  $(S2 \ r, v, \beta) \rightsquigarrow_{c*} (S2 \ p, u, \gamma)$ 
by simp
        from  $*** \ **** \ *****$  show  $?thesis$ 
        using comp.step1_steps comp.steps_trans by blast
      next
        case  $b$ 
          then obtain  $a \ \zeta$  where  $*$ :  $w = a \# v$  and  $**$ :  $\beta = \zeta @ \alpha'$  and elem:  $(r, \zeta)$ 
 $\in \delta \ M \ q' \ a \ X$  by blast
          show  $?thesis$ 
          proof (cases  $v = u$ )
            case True
              from  $2(5)$  have  $r\_nonfinal$ :  $r \notin final\_states \ M$ 
              using step1_steps[OF  $2(1)$ ][unfolded True]] by simp
              from  $2(6)$  consider ( $s1$ )  $q = S1 \ q' \mid (s2) \ q = S2 \ q'$  by blast
              then have  $p1$ :  $(q, w, \alpha) \rightsquigarrow_{c*} (S2 \ r, v, \beta)$ 
              proof cases
                case  $s1$ 
                  with elem  $r\_nonfinal$  have  $(S2 \ r, \zeta) \in comp\_dpda\_delta \ q \ a \ X$  by force
                  with  $* \ ** \ \alpha\_def$  have  $***$ :  $(q, w, \alpha) \rightsquigarrow_c (S2 \ r, v, \beta)$ 
                  using comp.step1_rule[of  $q \ w \ X \ \alpha' \ S2 \ r \ v \ \beta$ ] by (simp add: comp_dpda_def)
                  show  $?thesis$ 
                  using comp.step1_steps[OF  $***$ ] .
                next
                  case  $s2$ 
                    with elem have  $(S3 \ q', [X]) \in comp\_dpda\_delta\_eps \ q \ X$  by auto

```

```

    with  $\alpha\_def$  have ***:  $(q, w, \alpha) \rightsquigarrow_c (S3\ q', w, \alpha)$ 
    using comp.step1_rule[of  $q\ w\ X\ \alpha'\ S3\ q'\ w\ \alpha$ ] by (simp add: comp_dpda_def)
    from elem r_nonfinal have  $(S2\ r, \zeta) \in comp\_dpda\_delta\ (S3\ q')\ a\ X$  by
force
    with * **  $\alpha\_def$  have ****:  $(S3\ q', w, \alpha) \rightsquigarrow_c (S2\ r, v, \beta)$ 
    using comp.step1_rule[of  $S3\ q'\ w\ X\ \alpha'\ S2\ r\ v\ \beta$ ] by (simp add:
comp_dpda_def)
    show ?thesis
    using comp.steps_trans[OF comp.step1_steps[OF ***] comp.step1_steps[OF
****]] .
qed
from 2(2)[unfolded True] have a1:  $(r, [], \beta) \rightsquigarrow^* (p, [], \gamma)$ 
    using steps_word_app[of  $r\ []\ \beta\ p\ []\ \gamma\ u$ ] by simp
    from rp_nonfinal[unfolded True] have a2:  $\bigwedge ra\ \beta'. (r, [], \beta) \rightsquigarrow^* (ra, [], \beta')$ 
 $\longrightarrow ra \notin final\_states\ M$ 
    using steps_word_app[of  $r\ []\ \beta\ \_ []\ \_ u$ ] by simp
    have  $(S2\ r, [], \beta) \rightsquigarrow_c^* (S2\ p, [], \gamma)$ 
    using comp_dpda_nonfinal_stepsS2[OF a1 a2] .
    with True have p2:  $(S2\ r, v, \beta) \rightsquigarrow_c^* (S2\ p, u, \gamma)$ 
    using comp.steps_word_app[of  $S2\ r\ []\ \beta\ S2\ p\ []\ \gamma\ u$ ] by simp
    show ?thesis
    using comp.steps_trans[OF p1 p2] .
next
case False
from 2(6) consider  $(s1)\ q = S1\ q' \mid (s2)\ q = S2\ q'$  by blast
then have p:  $(q, w, \alpha) \rightsquigarrow_c^* (S1\ r, v, \beta) \vee (q, w, \alpha) \rightsquigarrow_c^* (S2\ r, v, \beta)$ 
proof (cases)
case s1
    with elem have  $(S1\ r, \zeta) \in comp\_dpda\_delta\ q\ a\ X \vee (S2\ r, \zeta) \in$ 
comp_dpda_delta  $q\ a\ X$  by force
    with * **  $\alpha\_def$  have  $(q, w, \alpha) \rightsquigarrow_c (S1\ r, v, \beta) \vee (q, w, \alpha) \rightsquigarrow_c (S2\ r, v,$ 
 $\beta)$ 
    using comp.step1_rule[of  $q\ w\ X\ \alpha'\ \_ v\ \beta$ ] by (simp add: comp_dpda_def)
    then show ?thesis
    using comp.step1_steps by blast
next
case s2
    with elem have  $(S3\ q', [X]) \in comp\_dpda\_delta\_eps\ q\ X$  by auto
    with  $\alpha\_def$  have ***:  $(q, w, \alpha) \rightsquigarrow_c (S3\ q', w, \alpha)$ 
    using comp.step1_rule[of  $q\ w\ X\ \alpha'\ S3\ q'\ w\ \alpha$ ] by (simp add: comp_dpda_def)
    from elem have  $(S1\ r, \zeta) \in comp\_dpda\_delta\ (S3\ q')\ a\ X \vee (S2\ r, \zeta) \in$ 
comp_dpda_delta  $(S3\ q')\ a\ X$  by force
    with * **  $\alpha\_def$  have ****:  $(S3\ q', w, \alpha) \rightsquigarrow_c (S1\ r, v, \beta) \vee (S3\ q', w, \alpha)$ 
 $\rightsquigarrow_c (S2\ r, v, \beta)$ 
    using comp.step1_rule[of  $S3\ q'\ w\ X\ \alpha'\ \_ v\ \beta$ ] by (simp add: comp_dpda_def)
    from *** **** show ?thesis
    using comp.step1_steps comp.steps_trans by metis
qed
from 2(3)[OF False rp_nonfinal] have p1:  $(S1\ r, v, \beta) \rightsquigarrow_c^* (S2\ p, u, \gamma)$  by

```

```

simp
  from 2(3)[OF False rp_nonfinal] have p2: (S2 r, v,  $\beta$ )  $\rightsquigarrow_c^*$  (S2 p, u,  $\gamma$ ) by
simp
  from p p1 p2 show ?thesis
  using comp.steps_trans by blast
qed
qed
qed simp

```

The automaton *comp_dpda* transitions to an S1-state if the target state is a final state:

```

lemma comp_dpda_final_steps:
  assumes (q', w,  $\alpha$ )  $\rightsquigarrow_+$  (p, u,  $\gamma$ )
  and p  $\in$  final_states M
  and q = S1 q'  $\vee$  q = S2 q'
  shows (q, w,  $\alpha$ )  $\rightsquigarrow_c^+$  (S1 p, u,  $\gamma$ )
using assms proof (induction (q', w,  $\alpha$ ) (p, u,  $\gamma$ ) arbitrary: q q' w  $\alpha$  rule:
steps1_induct)
  case (base q' w  $\alpha$ )
  from base(1) obtain X  $\alpha'$  where  $\alpha\_def$ :  $\alpha = X \# \alpha'$  and cases:
    ( $\exists \beta. u = w \wedge \gamma = \beta @ \alpha' \wedge (p, \beta) \in \delta\varepsilon M q' X$ )  $\vee$  ( $\exists a \beta. w = a \# u \wedge \gamma$ 
 $= \beta @ \alpha' \wedge (p, \beta) \in \delta M q' a X$ ) (is ?a  $\vee$  ?b)
  using step1_rule_ext[of q' w  $\alpha$  p u  $\gamma$ ] by blast
  from cases consider (a) ?a | (b) ?b by blast
  then show ?case
proof cases
  case a
  then obtain  $\beta$  where *:  $u = w$  and **:  $\gamma = \beta @ \alpha'$  and elem:  $(p, \beta) \in \delta\varepsilon M$ 
q' X by blast
  from base(2,3) elem have (S1 p,  $\beta$ )  $\in$  comp_dpda_delta_eps q X by force
  with * **  $\alpha\_def$  have ***:  $(q, w, \alpha) \rightsquigarrow_c (S1 p, u, \gamma)$ 
  using comp.step1_rule[of q w X  $\alpha'$  S1 p u  $\gamma$ ] by (simp add: comp_dpda_def)
  show ?thesis
  using comp.steps1_step[OF ***] .
next
  case b
  then obtain a  $\beta$  where *:  $w = a \# u$  and **:  $\gamma = \beta @ \alpha'$  and elem:  $(p, \beta)$ 
 $\in \delta M q' a X$  by blast
  from base(3) consider (c) q = S1 q' | (d) q = S2 q' by blast
  then show ?thesis
proof cases
  case c
  with elem base(2) have (S1 p,  $\beta$ )  $\in$  comp_dpda_delta q a X by force
  with * **  $\alpha\_def$  have ***:  $(q, w, \alpha) \rightsquigarrow_c (S1 p, u, \gamma)$ 
  using comp.step1_rule[of q w X  $\alpha'$  S1 p u  $\gamma$ ] by (simp add: comp_dpda_def)
  show ?thesis
  using comp.steps1_step[OF ***] .
next
  case d

```

```

with elem have  $(S3\ q', [X]) \in comp\_dpda\_delta\_eps\ q\ X$  by auto
with  $\alpha\_def$  have ***:  $(q, w, \alpha) \rightsquigarrow_c (S3\ q', w, \alpha)$ 
using comp.step1_rule[of  $q\ w\ X\ \alpha'\ S3\ q'\ w\ \alpha$ ] by (simp add: comp_dpda_def)
from elem base(2) have  $(S1\ p, \beta) \in comp\_dpda\_delta\ (S3\ q')\ a\ X$  by force
with ***  $\alpha\_def$  have ****:  $(S3\ q', w, \alpha) \rightsquigarrow_c (S1\ p, u, \gamma)$ 
using comp.step1_rule[of  $S3\ q'\ w\ X\ \alpha'\ S1\ p\ u\ \gamma$ ] by (simp add: comp_dpda_def)
show ?thesis
using comp.steps1_trans[OF comp.steps1_step[OF ***] comp.steps1_step[OF
****]] .
qed
qed
next
case (step  $q'\ w\ \alpha\ r\ v\ \beta$ )
from step(1) obtain  $X\ \alpha'$  where  $\alpha\_def: \alpha = X \# \alpha'$  and cases:
 $(\exists \beta'. v = w \wedge \beta = \beta' @ \alpha' \wedge (r, \beta') \in \delta\varepsilon\ M\ q'\ X) \vee (\exists a\ \beta'. w = a \# v \wedge$ 
 $\beta = \beta' @ \alpha' \wedge (r, \beta') \in \delta\ M\ q'\ a\ X)$  (is ?c  $\vee$  ?d)
using step1_rule_ext[of  $q'\ w\ \alpha\ r\ v\ \beta$ ] by blast
from step(5) consider (a)  $q = S1\ q' \mid$  (b)  $q = S2\ q'$  by blast
then show ?case
proof cases
case a
from cases consider (c) ?c  $\mid$  (d) ?d by blast
then show ?thesis
proof cases
case c
then obtain  $\zeta$  where *:  $v = w$  and **:  $\beta = \zeta @ \alpha'$  and elem:  $(r, \zeta) \in \delta\varepsilon$ 
 $M\ q'\ X$  by blast
from elem a have  $(S1\ r, \zeta) \in comp\_dpda\_delta\_eps\ q\ X$  by auto
with ***  $\alpha\_def$  have ***:  $(q, w, \alpha) \rightsquigarrow_c (S1\ r, v, \beta)$ 
using comp.step1_rule[of  $q\ w\ X\ \alpha'\ S1\ r\ v\ \beta$ ] by (simp add: comp_dpda_def)
from step(3)[OF step(4)] have ****:  $(S1\ r, v, \beta) \rightsquigarrow_{c+} (S1\ p, u, \gamma)$  by simp
show ?thesis
using comp.steps1_trans[OF comp.steps1_step[OF ***] ****] .
next
case d
then obtain  $a\ \zeta$  where *:  $w = a \# v$  and **:  $\beta = \zeta @ \alpha'$  and elem:  $(r, \zeta)$ 
 $\in \delta\ M\ q'\ a\ X$  by blast
from elem a have  $(S1\ r, \zeta) \in comp\_dpda\_delta\ q\ a\ X \vee (S2\ r, \zeta) \in$ 
 $comp\_dpda\_delta\ q\ a\ X$  by force
with ***  $\alpha\_def$  have ***:  $(q, w, \alpha) \rightsquigarrow_c (S1\ r, v, \beta) \vee (q, w, \alpha) \rightsquigarrow_c (S2\ r,$ 
 $v, \beta)$ 
using comp.step1_rule[of  $q\ w\ X\ \alpha'\ \_v\ \beta$ ] by (simp add: comp_dpda_def)
from step(3)[OF step(4)] have ****:  $(S1\ r, v, \beta) \rightsquigarrow_{c+} (S1\ p, u, \gamma)$  by simp
from step(3)[OF step(4)] have *****:  $(S2\ r, v, \beta) \rightsquigarrow_{c+} (S1\ p, u, \gamma)$  by simp
from *** ***** show ?thesis
using comp.steps1_step comp.steps1_trans by blast
qed
next
case b

```

```

from cases consider (c) ?c | (d) ?d by blast
then show ?thesis
proof cases
  case c
    then obtain  $\zeta$  where *:  $v = w$  and **:  $\beta = \zeta @ \alpha'$  and elem:  $(r, \zeta) \in \delta\varepsilon$ 
     $M q' X$  by blast
    from elem b have  $(S1\ r, \zeta) \in comp\_dpda\_delta\_eps\ q\ X \vee (S2\ r, \zeta) \in$ 
     $comp\_dpda\_delta\_eps\ q\ X$  by force
    with * **  $\alpha\_def$  have ***:  $(q, w, \alpha) \rightsquigarrow_c (S1\ r, v, \beta) \vee (q, w, \alpha) \rightsquigarrow_c (S2\ r,$ 
     $v, \beta)$ 
    using comp.step1_rule[of  $q\ w\ X\ \alpha' \_ v\ \beta$ ] by (simp add: comp_dpda_def)
    from step(3)[OF step(4)] have ****:  $(S1\ r, v, \beta) \rightsquigarrow_{c+} (S1\ p, u, \gamma)$  by simp
    from step(3)[OF step(4)] have *****:  $(S2\ r, v, \beta) \rightsquigarrow_{c+} (S1\ p, u, \gamma)$  by simp
    from *** ***** show ?thesis
    using comp.steps1_step comp.steps1_trans by blast
  next
  case d
    then obtain a  $\zeta$  where *:  $w = a \# v$  and **:  $\beta = \zeta @ \alpha'$  and elem:  $(r, \zeta)$ 
     $\in \delta\ M\ q' a\ X$  by blast
    from elem b have  $(S3\ q', [X]) \in comp\_dpda\_delta\_eps\ q\ X$  by auto
    with  $\alpha\_def$  have ***:  $(q, w, \alpha) \rightsquigarrow_c (S3\ q', w, \alpha)$ 
    using comp.step1_rule[of  $q\ w\ X\ \alpha'\ S3\ q'\ w\ \alpha$ ] by (simp add: comp_dpda_def)
    from elem have  $(S1\ r, \zeta) \in comp\_dpda\_delta\ (S3\ q')\ a\ X \vee (S2\ r, \zeta) \in$ 
     $comp\_dpda\_delta\ (S3\ q')\ a\ X$  by force
    with * **  $\alpha\_def$  have ****:  $(S3\ q', w, \alpha) \rightsquigarrow_c (S1\ r, v, \beta) \vee (S3\ q', w, \alpha)$ 
     $\rightsquigarrow_c (S2\ r, v, \beta)$ 
    using comp.step1_rule[of  $S3\ q'\ w\ X\ \alpha' \_ v\ \beta$ ] by (simp add: comp_dpda_def)
    from step(3)[OF step(4)] have *****:  $(S1\ r, v, \beta) \rightsquigarrow_{c+} (S1\ p, u, \gamma)$  by simp
    from step(3)[OF step(4)] have *****:  $(S2\ r, v, \beta) \rightsquigarrow_{c+} (S1\ p, u, \gamma)$  by
    simp
    from *** ***** ***** show ?thesis
    using comp.steps1_step comp.steps1_trans bymetis
  qed
qed
qed

```

If the automaton *comp_dpda* ends up in an S3-state while reading a word, then no state reached while reading that same word is final:

```

lemma comp_dpda_steps_nonfinalS1:
  assumes  $(S1\ q, w, \alpha) \rightsquigarrow_{c*} (S3\ p, [], \gamma)$ 
  and  $(q, w, \alpha) \rightsquigarrow^* (r, [], \beta)$ 
  shows  $r \notin final\_states\ M$ 
proof
  assume r_final:  $r \in final\_states\ M$ 
  from assms(1) obtain n where p1:  $(S1\ q, w, \alpha) \rightsquigarrow_{c(n)} (S3\ p, [], \gamma)$ 
  using comp.stepn_steps[of  $S1\ q\ w\ \alpha\ S3\ p\ []\ \gamma$ ] by blast
  have np: comp.no_step1  $(S3\ p, [], \gamma)$ 
  using comp.step1_rule_ext[of  $S3\ p\ []\ \gamma$ ] by (simp add: comp_dpda_def)
  have  $(S1\ q, w, \alpha) \rightsquigarrow_{c*} (S1\ r, [], \beta)$ 

```

```

proof (cases (q, w, α) = (r, [], β))
  case True
  then show ?thesis
    by (simp add: comp.steps_refl)
next
  case False
  have q1: (q, w, α)  $\rightsquigarrow^+$  (r, [], β)
    using steps_steps1[OF assms(2) False] .
  have q2: (S1 q, w, α)  $\rightsquigarrow_c^+$  (S1 r, [], β)
    using comp_dpda_final_steps[OF q1 r_final] by simp
  show ?thesis
    using comp.steps1_steps[OF q2] .
qed
then obtain m where p2: (S1 q, w, α)  $\rightsquigarrow_c(m)$  (S1 r, [], β)
  using comp.stepn_steps[of S1 q w α S1 r [] β] by blast
have m_leq_n: m ≤ n
  using comp.max_eps_steps[OF p1 np p2] .
have (S1 r, [], β)  $\rightsquigarrow_c(n - m)$  (S3 p, [], γ)
  using comp.split_path[OF p1 m_leq_n] comp.dpda_stepn_det[OF p2] by blast
then have *: (S1 r, [], β)  $\rightsquigarrow_{c*}$  (S3 p, [], γ)
  using comp.stepn_steps[of S1 r [] β S3 p [] γ] by blast
show False
  using comp_dpda_steps_from_S1[OF *] by simp
qed

lemma comp_dpda_steps_nonfinalS2:
  assumes (S2 q, w, α)  $\rightsquigarrow_{c*}$  (S3 p, [], γ)
  and (q, w, α)  $\rightsquigarrow^+$  (r, [], β)
  shows r ∉ final_states M
proof
  assume r_final: r ∈ final_states M
  from assms(1) obtain n where p1: (S2 q, w, α)  $\rightsquigarrow_c(n)$  (S3 p, [], γ)
  using comp.stepn_steps[of S2 q w α S3 p [] γ] by blast
  have np: comp.no_step1 (S3 p, [], γ)
  using comp.step1_rule_ext[of S3 p [] γ] by (simp add: comp_dpda_def)
  obtain m where p2: (S2 q, w, α)  $\rightsquigarrow_c(m)$  (S1 r, [], β)
  using comp.steps1_steps[OF comp_dpda_final_steps[OF assms(2) r_final, of S2 q, simplified]] comp.stepn_steps[of S2 q w α S1 r [] β] by blast
  have m_leq_n: m ≤ n
  using comp.max_eps_steps[OF p1 np p2] .
  have (S1 r, [], β)  $\rightsquigarrow_c(n - m)$  (S3 p, [], γ)
  using comp.split_path[OF p1 m_leq_n] comp.dpda_stepn_det[OF p2] by blast
  then have *: (S1 r, [], β)  $\rightsquigarrow_{c*}$  (S3 p, [], γ)
  using comp.stepn_steps[of S1 r [] β S3 p [] γ] by blast
  show False
  using comp_dpda_steps_from_S1[OF *] by simp
qed

```

The language of the automaton *comp_dpda* is the complement language:

```

lemma lang_comp_dpda:
  comp.accept_final = - accept_final
proof
  show comp.accept_final  $\subseteq$  - accept_final
  proof
    fix w
    assume w  $\in$  comp.accept_final
    then obtain q  $\gamma$  where p: (init_state comp_dpda, w, [init_symbol comp_dpda])
     $\rightsquigarrow_{c*}$  (q, [],  $\gamma$ ) and q_final: q  $\in$  final_states comp_dpda
    unfolding comp.accept_final_def by blast
    from q_final obtain q' where q_def: q = S3 q'
    by (auto simp: comp_dpda_def comp_dpda_final_states_def)
    have  $\bigwedge r \beta$ . (init_state M, w, [init_symbol M])  $\rightsquigarrow_r$  (r, [],  $\beta$ )  $\longrightarrow$  r  $\notin$  final_states
    M
  proof
    fix r  $\beta$ 
    assume asm: (init_state M, w, [init_symbol M])  $\rightsquigarrow_r$  (r, [],  $\beta$ )
    consider (a) init_state M  $\in$  final_states M | (b) init_state M  $\notin$  final_states
    M by satx
    then show r  $\notin$  final_states M
    proof cases
      case a
      with p q_def have *: (S1 (init_state M), w, [init_symbol M])  $\rightsquigarrow_{c*}$  (S3 q',
    [],  $\gamma$ )
      by (simp add: comp_dpda_def comp_dpda_init_state_def)
      show ?thesis
      using comp_dpda_steps_nonfinalS1[OF * asm] .
    next
      case b
      consider (c) r = init_state M | (d) r  $\neq$  init_state M by satx
      then show ?thesis
      proof cases
        case c
        with b show ?thesis by simp
      next
        case d
        then have *: (init_state M, w, [init_symbol M])  $\rightsquigarrow_+$  (r, [],  $\beta$ )
        using steps_steps1[OF asm] by simp
        from b p q_def have **: (S2 (init_state M), w, [init_symbol M])  $\rightsquigarrow_{c*}$ 
    (S3 q', [],  $\gamma$ )
        by (simp add: comp_dpda_def comp_dpda_init_state_def)
        show ?thesis
        using comp_dpda_steps_nonfinalS2[OF ** *] .
      qed
    qed
  qed
  then show w  $\in$  - accept_final
  by (auto simp: accept_final_def)
  qed

```

```

next
  show  $- \text{accept\_final} \subseteq \text{comp.accept\_final}$ 
  proof
    fix w
    assume  $w \in - \text{accept\_final}$ 
    then have nonfinal:  $\bigwedge r \beta. (\text{init\_state } M, w, [\text{init\_symbol } M]) \rightsquigarrow^* (r, [], \beta)$ 
     $\longrightarrow r \notin \text{final\_states } M$ 
    by (auto simp: accept_final_def)
    from  $M\_path[\text{of } w]$  obtain  $q \ X \ \alpha \ a$  where  $p: (\text{init\_state } M, w, [\text{init\_symbol } M]) \rightsquigarrow^* (q, [], X \# \alpha)$  and  $\text{delta\_M}: \delta \ M \ q \ a \ X \neq \{\}$  by blast
    consider (a)  $w = []$  | (b)  $w \neq []$  by satx
    then have *:  $(\text{init\_state } \text{comp\_dpda}, w, [\text{init\_symbol } \text{comp\_dpda}]) \rightsquigarrow_c^* (S2 \ q, [], X \# \alpha)$ 
    proof cases
      case a
      from nonfinal[unfolded a] have  $\text{init\_state } M \notin \text{final\_states } M$ 
      using steps_refl[of  $\text{init\_state } M \ [] \ [\text{init\_symbol } M]$ ] by simp
      with a show ?thesis
      using comp_dpda_nonfinal_stepsS2[OF p[unfolded a] nonfinal[unfolded a]]
    by (simp add: comp_dpda_def comp_dpda_init_state_def)
    next
      case b
      with p nonfinal show ?thesis
      using comp_dpda_nonfinal_steps[of  $\text{init\_state } M \ w \ [\text{init\_symbol } M] \ q \ [] \ X \# \alpha$ ] by (simp add: comp_dpda_def comp_dpda_init_state_def)
    qed
    from delta_M have **:  $(S2 \ q, [], X \# \alpha) \rightsquigarrow_c (S3 \ q, [], X \# \alpha)$ 
    using comp.step1_rule[of  $S2 \ q \ [] \ X \ \alpha \ S3 \ q \ [] \ X \# \alpha$ ] comp_dpda_def by auto
    have  $(\text{init\_state } \text{comp\_dpda}, w, [\text{init\_symbol } \text{comp\_dpda}]) \rightsquigarrow_c^* (S3 \ q, [], X \# \alpha)$ 
    using comp.steps_trans[OF * comp.step1_steps[OF **]] .
    then show  $w \in \text{comp.accept\_final}$ 
    using comp.accept_final_def comp_dpda_def comp_dpda_final_states_def
  by force
  qed
  qed
end

```

By constructing the two automata one after another, we get the final lemma stating that deterministic context-free languages are closed under complementation:

```

lemma complement_dpda:
  assumes  $\text{dpda } (M :: ('q :: \text{finite}, 'a :: \text{finite}, 's :: \text{finite}) \text{ pda})$ 
  shows  $\exists M' :: ('q \text{ st\_scan } s123, 'a, 's \text{ sym\_scan}) \text{ pda}.$ 
   $\text{dpda } M' \wedge \text{pda.accept\_final } M' = - \text{pda.accept\_final } M$ 
  proof -
    let ?SM =  $\text{dpda\_scan.scan\_dpda } M :: ('q \text{ st\_scan}, 'a, 's \text{ sym\_scan}) \text{ pda}$ 
    have  $\text{dpda\_sm}: \text{dpda } ?SM$ 

```

```

    using assms dpda_scan.dpda_scan_dpda dpda_scan_def by blast
    have *:  $\bigwedge a w. \exists q X \alpha. \text{pda.steps } ?SM (\text{init\_state } ?SM, w, [\text{init\_symbol } ?SM])$ 
       $(q, [], X \# \alpha) \wedge \text{delta } ?SM q a X \neq \{\}$ 
    using assms dpda_scan.scan_dpda_scans dpda_scan_def by blast
    have L1:  $\text{pda.accept\_final } ?SM = \text{pda.accept\_final } M$ 
    using assms dpda_scan.lang_scan_dpda dpda_scan.intro by auto
    let ?CM = complement_dpda.comp_dpda ?SM :: ('q st_scan s123, 'a, 's sym_scan)
pda
    from dpda_sm * have dpda_cm: dpda ?CM
    using complement_dpda.dpda_comp_dpda complement_dpda.intro complement_dpda_axioms_def
by blast
    from dpda_sm * have L2:  $\text{pda.accept\_final } ?CM = \text{UNIV} - \text{pda.accept\_final } ?SM$ 
    using complement_dpda.lang_comp_dpda complement_dpda_axioms_def complement_dpda_def by blast
    from dpda_cm L1 L2 show ?thesis by blast
qed

end

```

References

- [1] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Publishing Company, 1979.