

A Verified Translation of Multitape Turing Machines into Singletape Turing Machines

Christian Dalvit and René Thiemann

April 18, 2024

Abstract

We define single- and multitape Turing machines (TMs) and verify a translation from multitape TMs to singletape TMs. In particular, the following results have been formalized: the accepted languages coincide, and whenever the multitape TM runs in $\mathcal{O}(f(n))$ time, then the singletape TM has a worst-time complexity of $\mathcal{O}(f(n)^2 + n)$. The translation is applicable both on deterministic and non-deterministic TMs.

Contents

1	Introduction	2
2	Preparations	2
3	Multitape Turing Machines	3
4	Singletape Turing Machines	5
5	Renamings for Singletape Turing Machines	8
6	Translating Multitape TMs to Singletape TMs	11
6.1	Definition of the Translation	11
6.2	Soundness of the Translation	16
6.2.1	R-Phase	16
6.2.2	S-Phase	17
6.2.3	E-Phase	18
6.2.4	Simulation of multitape TM by singletape TM	20
6.2.5	Simulation of singletape TM by multitape TM	21
6.2.6	Main Results	21
6.3	Main Results with Proper Renamings	21

1 Introduction

In 1965 Hartmanis and Stearns proved that multitape Turing machines (TMs) can be simulated by singletape Turing machines [1]. Since then, alternative approaches for translating multitape TMs to singletape TMs have been formulated [2, 3]. In this AFP entry we define a translation which has the usual quadratic overhead in running time.

For the design of the translation we had to choose between the approach how to encode the k tapes of a multitape TM onto a single tape.

In the textbooks [2, 3] the k tapes t_1, \dots, t_n are stored sequentially onto a single tape $t_1\# \dots \#t_n$ via a separator $\#$. The technical problem with this definition is that once a tape t_i needs to be enlarged to the right, the later tape content $\#t_{i+1}\# \dots \#t_n$ needs to be shifted correspondingly.

To avoid this problem, we followed the idea in the original work of Hartmanis et al. where the k -tapes are stored on top of each other, i.e., basically for the tape alphabet Γ of the multitape TM we switch to Γ^k in the singletape TM. As a consequence, the formal translation could be kept simple, in particular no tape shifts need to be performed.

2 Preparations

theory *TM-Common*

imports

HOL-Library.FuncSet

begin

A direction of a TM: go right, go left, or neutral (stay)

datatype *dir* = *R* | *L* | *N*

fun *go-dir* :: *dir* \Rightarrow *nat* \Rightarrow *nat* **where**

go-dir *R* *n* = *Suc* *n*

| *go-dir* *L* *n* = *n* - 1

| *go-dir* *N* *n* = *n*

lemma *finite-UNIV-dir*[*simp, intro*]: *finite* (*UNIV* :: *dir* set)

<proof>

hide-const (**open**) *L R N*

lemma *fin-funcsetI*[*intro*]: *finite* *A* \Longrightarrow *finite* ((*UNIV* :: '*a* :: *finite* set) \rightarrow *A*)

<proof>

lemma *finite-UNIV-fun-dir*[*simp, intro*]: *finite* (*UNIV* :: ('*k* :: *finite* \Rightarrow *dir*) set)

<proof>

lemma *relpow-transI*: $(x, y) \in R^{\sim n} \Longrightarrow (y, z) \in R^{\sim m} \Longrightarrow (x, z) \in R^{\sim (n+m)}$

<proof>

lemma *relpow-mono*: **fixes** $R :: 'a \text{ rel}$ **shows** $R \subseteq S \implies R^{\sim n} \subseteq S^{\sim n}$
<proof>

lemma *finite-infinite-inj-on*: **assumes** A : *finite* ($A :: 'a \text{ set}$) **and** inf : *infinite* ($UNIV :: 'b \text{ set}$)
shows $\exists f :: 'a \Rightarrow 'b$. *inj-on* f A
<proof>

lemma *gauss-sum-nat2*: $(\sum i < (n :: nat). i) = (n - 1) * n \text{ div } 2$
<proof>

lemma *aux-sum-formula*: $(\sum i < n. 10 + 5 * i) \leq 3 * n^2 + 7 * (n :: nat)$
<proof>

end

3 Multitape Turing Machines

theory *Multitape-TM*

imports

TM-Common

begin

Turing machines can be either defined via a datatype or via a locale. We use TMs with left endmarker and dedicated accepting and rejecting state from which no further transitions are allowed. Deterministic TMs can be partial.

Having multiple tapes, tape positions, directions, etc. is modelled via functions of type $'k \Rightarrow 'whatever$ for some finite index type $'k$.

The input will always be provided on the first tape, indexed by $0 :: 'k$.

datatype $('q, 'a, 'k) mttm = MTTM$
(Q-tm: 'q set)
'a set
(Γ-tm: 'a set)
'a
'a
('q × ('k ⇒ 'a) × 'q × ('k ⇒ 'a) × ('k ⇒ dir)) set
'q
'q
'q

datatype $('a, 'q, 'k) mt-config = Config_M$
(mt-state: 'q)
'k ⇒ nat ⇒ 'a
(mt-pos: 'k ⇒ nat)

locale *multitape-tm* =
fixes
 $Q :: 'q \text{ set}$ **and**
 $\Sigma :: 'a \text{ set}$ **and**
 $\Gamma :: 'a \text{ set}$ **and**
 $\text{blank} :: 'a$ **and**
 $LE :: 'a$ **and**
 $\delta :: ('q \times ('k \Rightarrow 'a) \times 'q \times ('k \Rightarrow 'a) \times ('k :: \{\text{finite}, \text{zero}\} \Rightarrow \text{dir})) \text{ set}$ **and**
 $s :: 'q$ **and**
 $t :: 'q$ **and**
 $r :: 'q$

assumes
 $\text{fin-}Q$: $\text{finite } Q$ **and**
 $\text{fin-}\Gamma$: $\text{finite } \Gamma$ **and**
 $\Sigma\text{-sub-}\Gamma$: $\Sigma \subseteq \Gamma$ **and**
 sQ : $s \in Q$ **and**
 tQ : $t \in Q$ **and**
 rQ : $r \in Q$ **and**
 blank : $\text{blank} \in \Gamma$ $\text{blank} \notin \Sigma$ **and**
 LE : $LE \in \Gamma$ $LE \notin \Sigma$ **and**
 tr : $t \neq r$ **and**
 $\delta\text{-set}$: $\delta \subseteq (Q - \{t, r\}) \times (UNIV \rightarrow \Gamma) \times Q \times (UNIV \rightarrow \Gamma) \times (UNIV \rightarrow UNIV)$ **and**
 δLE : $(q, a, q', a', d) \in \delta \implies a k = LE \implies a' k = LE \wedge d k \in \{\text{dir.N}, \text{dir.R}\}$

begin

lemma δ : **assumes** $(q, a, q', b, d) \in \delta$
shows $q \in Q$ $a k \in \Gamma$ $q' \in Q$ $b k \in \Gamma$
 $\langle \text{proof} \rangle$

lemma $\text{fin-}\Sigma$: $\text{finite } \Sigma$
 $\langle \text{proof} \rangle$

lemma $\text{fin-}\delta$: $\text{finite } \delta$
 $\langle \text{proof} \rangle$

lemmas $tm = sQ$ $\Sigma\text{-sub-}\Gamma$ $\text{blank}(1)$ $LE(1)$

fun $\text{valid-config} :: ('a, 'q, 'k) \text{ mt-config} \Rightarrow \text{bool}$ **where**
 $\text{valid-config } (\text{Config}_M q w n) = (q \in Q \wedge (\forall k. \text{range } (w k) \subseteq \Gamma) \wedge (\forall k. w k 0 = LE))$

definition $\text{init-config} :: 'a \text{ list} \Rightarrow ('a, 'q, 'k) \text{ mt-config}$ **where**
 $\text{init-config } w = (\text{Config}_M s (\lambda k n. \text{if } n = 0 \text{ then } LE \text{ else if } k = 0 \wedge n \leq \text{length } w \text{ then } w ! (n-1) \text{ else blank}) (\lambda -. 0))$

lemma valid-init-config : $\text{set } w \subseteq \Sigma \implies \text{valid-config } (\text{init-config } w)$
 $\langle \text{proof} \rangle$

inductive-set $step :: ('a, 'q, 'k) mt\text{-}config\ rel$ **where**
 $step: (q, (\lambda k. ts\ k\ (n\ k)), q', a, dir) \in \delta \implies$
 $(Config_M\ q\ ts\ n, Config_M\ q'\ (\lambda k. (ts\ k)(n\ k := a\ k)) (\lambda k. go\text{-}dir\ (dir\ k)\ (n\ k))) \in step$

lemma $valid\text{-}step$: **assumes** $step: (\alpha, \beta) \in step$
and $val: valid\text{-}config\ \alpha$
shows $valid\text{-}config\ \beta$
 $\langle proof \rangle$

definition $Lang :: 'a\ list\ set$ **where**
 $Lang = \{w . set\ w \subseteq \Sigma \wedge (\exists w' n. (init\text{-}config\ w, Config_M\ t\ w'\ n) \in step^{\widehat{*}})\}$

definition $deterministic$ **where**
 $deterministic = (\forall q\ a\ p1\ b1\ d1\ p2\ b2\ d2. (q, a, p1, b1, d1) \in \delta \longrightarrow (q, a, p2, b2, d2) \in \delta \longrightarrow (p1, b1, d1) = (p2, b2, d2))$

definition $upper\text{-}time\text{-}bound :: (nat \Rightarrow nat) \Rightarrow bool$ **where**
 $upper\text{-}time\text{-}bound\ f = (\forall w\ c\ n. set\ w \subseteq \Sigma \longrightarrow (init\text{-}config\ w, c) \in step^{\sim n} \longrightarrow n \leq f\ (length\ w))$
end

fun $valid\text{-}mttm :: ('q, 'a, 'k :: \{finite, zero\})mttm \Rightarrow bool$ **where**
 $valid\text{-}mttm\ (MTTM\ Q\ \Sigma\ \Gamma\ bl\ le\ \delta\ s\ t\ r) = multitape\text{-}tm\ Q\ \Sigma\ \Gamma\ bl\ le\ \delta\ s\ t\ r$

fun $Lang\text{-}mttm :: ('q, 'a, 'k :: \{finite, zero\})mttm \Rightarrow 'a\ list\ set$ **where**
 $Lang\text{-}mttm\ (MTTM\ Q\ \Sigma\ \Gamma\ bl\ le\ \delta\ s\ t\ r) = multitape\text{-}tm.Lang\ \Sigma\ bl\ le\ \delta\ s\ t$

fun $det\text{-}mttm :: ('q, 'a, 'k :: \{finite, zero\})mttm \Rightarrow bool$ **where**
 $det\text{-}mttm\ (MTTM\ Q\ \Sigma\ \Gamma\ bl\ le\ \delta\ s\ t\ r) = multitape\text{-}tm.deterministic\ \delta$

fun $upperb\text{-}time\text{-}mttm :: ('q, 'a, 'k :: \{finite, zero\})mttm \Rightarrow (nat \Rightarrow nat) \Rightarrow bool$
where
 $upperb\text{-}time\text{-}mttm\ (MTTM\ Q\ \Sigma\ \Gamma\ bl\ le\ \delta\ s\ t\ r)\ f = multitape\text{-}tm.upper\text{-}time\text{-}bound\ \Sigma\ bl\ le\ \delta\ s\ f$

end

4 Singletape Turing Machines

theory $Singletape\text{-}TM$
imports
 $TM\text{-}Common$
begin

Turing machines can be either defined via a datatype or via a locale. We

use TMs with left endmarker and dedicated accepting and rejecting state from which no further transitions are allowed. Deterministic TMs can be partial.

```
datatype ('q,'a)tm = TM
  (Q-tm: 'q set)
  'a set
  (Γ-tm: 'a set)
  'a
  'a
  ('q × 'a × 'q × 'a × dir) set
  'q
  'q
  'q
```

```
datatype ('a, 'q) st-config = Configs
  'q
  nat ⇒ 'a
  nat
```

```
locale singletape-tm =
```

```
fixes
```

```
  Q :: 'q set and
  Σ :: 'a set and
  Γ :: 'a set and
  blank :: 'a and
  LE :: 'a and
  δ :: ('q × 'a × 'q × 'a × dir) set and
  s :: 'q and
  t :: 'q and
  r :: 'q
```

```
assumes
```

```
  fin-Q: finite Q and
  fin-Γ: finite Γ and
  Σ-sub-Γ: Σ ⊆ Γ and
  sQ: s ∈ Q and
  tQ: t ∈ Q and
  rQ: r ∈ Q and
  blank: blank ∈ Γ blank ∉ Σ and
  LE: LE ∈ Γ LE ∉ Σ and
  tr: t ≠ r and
  δ-set: δ ⊆ (Q - {t,r}) × Γ × Q × Γ × UNIV and
  δLE: (q, LE, q', a', d) ∈ δ ⇒ a' = LE ∧ d ∈ {dir.N, dir.R}
```

```
begin
```

```
lemma δ: assumes (q,a,q',b,d) ∈ δ
shows q ∈ Q a ∈ Γ q' ∈ Q b ∈ Γ
⟨proof⟩
```

```
lemma finΣ: finite Σ
```

<proof>

lemmas $tm = sQ \Sigma\text{-sub-}\Gamma \text{ blank}(1) LE(1)$

fun $valid\text{-}config :: ('a, 'q) \text{ st-config} \Rightarrow \text{bool}$ **where**
 $valid\text{-}config (Config_S q w n) = (q \in Q \wedge range w \subseteq \Gamma)$

definition $init\text{-}config :: 'a \text{ list} \Rightarrow ('a, 'q) \text{ st-config}$ **where**
 $init\text{-}config w = (Config_S s (\lambda n. \text{if } n = 0 \text{ then } LE \text{ else if } n \leq \text{length } w \text{ then } w ! (n-1) \text{ else blank } 0))$

lemma $valid\text{-}init\text{-}config: set w \subseteq \Sigma \Longrightarrow valid\text{-}config (init\text{-}config w)$
<proof>

inductive-set $step :: ('a, 'q) \text{ st-config rel}$ **where**
 $step: (q, ts n, q', a, dir) \in \delta \Longrightarrow$
 $(Config_S q ts n, Config_S q' (ts(n := a))) (go\text{-}dir dir n) \in step$

lemma $stepI: (q, a, q', b, dir) \in \delta \Longrightarrow ts n = a \Longrightarrow ts' = ts(n := b) \Longrightarrow n' =$
 $go\text{-}dir dir n \Longrightarrow q1 = q \Longrightarrow q2 = q'$
 $\Longrightarrow (Config_S q1 ts n, Config_S q2 ts' n') \in step$
<proof>

lemma $valid\text{-}step: \text{assumes } step: (\alpha, \beta) \in step$
and $val: valid\text{-}config \alpha$
shows $valid\text{-}config \beta$
<proof>

definition $Lang :: 'a \text{ list set}$ **where**
 $Lang = \{w . set w \subseteq \Sigma \wedge (\exists w' n. (init\text{-}config w, Config_S t w' n) \in step^{\sim*})\}$

definition $deterministic$ **where**
 $deterministic = (\forall q a p1 b1 d1 p2 b2 d2. (q, a, p1, b1, d1) \in \delta \longrightarrow (q, a, p2, b2, d2) \in \delta \longrightarrow (p1, b1, d1) = (p2, b2, d2))$

definition $upper\text{-}time\text{-}bound :: (nat \Rightarrow nat) \Rightarrow \text{bool}$ **where**
 $upper\text{-}time\text{-}bound f = (\forall w c n. set w \subseteq \Sigma \longrightarrow (init\text{-}config w, c) \in step^{\sim n} \longrightarrow n \leq f (\text{length } w))$
end

fun $valid\text{-}tm :: ('q, 'a) \text{ tm} \Rightarrow \text{bool}$ **where**
 $valid\text{-}tm (TM Q \Sigma \Gamma bl le \delta s t r) = \text{singletape}\text{-}tm Q \Sigma \Gamma bl le \delta s t r$

fun $Lang\text{-}tm :: ('q, 'a) \text{ tm} \Rightarrow 'a \text{ list set}$ **where**
 $Lang\text{-}tm (TM Q \Sigma \Gamma bl le \delta s t r) = \text{singletape}\text{-}tm.Lang \Sigma bl le \delta s t r$

fun $det\text{-}tm :: ('q, 'a) \text{ tm} \Rightarrow \text{bool}$ **where**
 $det\text{-}tm (TM Q \Sigma \Gamma bl le \delta s t r) = \text{singletape}\text{-}tm.deterministic \delta$

```

fun upperb-time-tm :: ('q,'a)tm  $\Rightarrow$  (nat  $\Rightarrow$  nat)  $\Rightarrow$  bool where
  upperb-time-tm (TM Q  $\Sigma$   $\Gamma$  bl le  $\delta$  s t r) f = singletape-tm.upper-time-bound  $\Sigma$ 
  bl le  $\delta$  s f

```

```

context singletape-tm
begin

```

A deterministic step (in a potentially non-deterministic TM) is a step without alternatives. This will be useful in the translation of multitape TMs. The simulation is mostly deterministic, and only at very specific points it is non-deterministic, namely at the points where the multitape-TM transition is chosen.

```

inductive-set dstep :: ('a, 'q) st-config rel where
  dstep: (q, ts n, q', a, dir)  $\in \delta \Longrightarrow$ 
    ( $\bigwedge$  q1' a1 dir1. (q, ts n, q1', a1, dir1)  $\in \delta \Longrightarrow$  (q1', a1, dir1) = (q', a, dir))  $\Longrightarrow$ 
    (ConfigS q ts n, ConfigS q' (ts(n := a))) (go-dir dir n)  $\in$  dstep

```

```

lemma dstepI: (q, a, q', b, dir)  $\in \delta \Longrightarrow$  ts n = a  $\Longrightarrow$  ts' = ts(n := b)  $\Longrightarrow$  n' =
  go-dir dir n  $\Longrightarrow$  q1 = q  $\Longrightarrow$  q2 = q'
   $\Longrightarrow$  ( $\bigwedge$  q'' b' dir'. (q, a, q'', b', dir')  $\in \delta \Longrightarrow$  (q'', b', dir') = (q', b, dir))
   $\Longrightarrow$  (ConfigS q1 ts n, ConfigS q2 ts' n')  $\in$  dstep
  <proof>

```

```

lemma dstep-step: dstep  $\subseteq$  step
  <proof>

```

```

lemma dstep-inj: assumes (x,y)  $\in$  dstep
  and (x,z)  $\in$  step
shows z = y
  <proof>

```

```

lemma dsteps-inj: assumes (x,y)  $\in$  dstep $\sim$ n
  and (x,z)  $\in$  step $\sim$ m
  and  $\neg$  ( $\exists$  u. (z,u)  $\in$  step)
shows  $\exists$  k. m = n + k  $\wedge$  (y,z)  $\in$  step $\sim$ k
  <proof>

```

```

lemma dsteps-inj': assumes (x,y)  $\in$  dstep $\sim$ n
  and (x,z)  $\in$  step $\sim$ m
  and m  $\geq$  n
shows  $\exists$  k. m = n + k  $\wedge$  (y,z)  $\in$  step $\sim$ k
  <proof>
end
end

```

5 Renamings for Singletape Turing Machines

```

theory STM-Renaming

```

```

imports
  Singletape-TM
begin

locale renaming-of-singletape-tm = singletape-tm Q Σ Γ blank LE δ s t r
  for Q :: 'q set and Σ :: 'a set and Γ blank LE δ s t r
    + fixes ra :: 'a ⇒ 'b
    and rq :: 'q ⇒ 'p
    assumes ra: inj-on ra Γ
    and rq: inj-on rq Q
begin

abbreviation rd where rd ≡ map-prod rq (map-prod ra (map-prod rq (map-prod
ra (λ d :: dir. d))))

sublocale ren: singletape-tm rq ' Q ra ' Σ ra ' Γ ra blank ra LE rd ' δ rq s rq t rq
r
⟨proof⟩

fun rc :: ('a, 'q) st-config ⇒ ('b, 'p) st-config where
  rc (Configs q tc pos) = Configs (rq q) (ra o tc) pos

lemma ren-init: rc (init-config w) = ren.init-config (map ra w)
  ⟨proof⟩

lemma ren-step: assumes (c, c') ∈ step
  shows (rc c, rc c') ∈ ren.step
  ⟨proof⟩

lemma ren-steps: assumes (c, c') ∈ step∗
  shows (rc c, rc c') ∈ ren.step∗
  ⟨proof⟩

lemma ren-steps-count: assumes (c, c') ∈ step∞
  shows (rc c, rc c') ∈ ren.step∞
  ⟨proof⟩

lemma ren-Lang-forward: assumes w ∈ Lang
  shows map ra w ∈ ren.Lang
  ⟨proof⟩

abbreviation ira where ira ≡ the-inv-into Γ ra
abbreviation irq where irq ≡ the-inv-into Q rq

interpretation inv: renaming-of-singletape-tm rq ' Q ra ' Σ ra ' Γ ra blank ra LE
rd ' δ rq s rq t rq r ira irq
  ⟨proof⟩

lemmas inv-simps[simp] = the-inv-into-f-f[OF ra] the-inv-into-f-f[OF rq]

```

lemma *inv-ren-Sigma*: $\text{ira } 'ra \text{ } ' \Sigma = \Sigma$ $\langle \text{proof} \rangle$

lemma *inv-ren-Gamma*: $\text{ira } 'ra \text{ } ' \Gamma = \Gamma$ $\langle \text{proof} \rangle$

lemma *inv-ren-t*: $\text{irq } (rq \ t) = t$ $\langle \text{proof} \rangle$

lemma *inv-ren-s*: $\text{irq } (rq \ s) = s$ $\langle \text{proof} \rangle$

lemma *inv-ren-r*: $\text{irq } (rq \ r) = r$ $\langle \text{proof} \rangle$

lemma *inv-ren-blank*: $\text{ira } (ra \ \text{blank}) = \text{blank}$ $\langle \text{proof} \rangle$

lemma *inv-ren-LE*: $\text{ira } (ra \ LE) = LE$ $\langle \text{proof} \rangle$

lemma *inv-ren- δ* : $\text{inv.rd } 'rd \text{ } ' \delta = \delta$
 $\langle \text{proof} \rangle$

lemmas *inv-ren* = *inv-ren-t inv-ren-s inv-ren-r inv-ren- δ inv-ren-Gamma inv-ren-Sigma*
inv-ren-blank inv-ren-LE

lemma *inv-ren-Lang*: $\text{inv.ren.Lang} = \text{Lang}$ $\langle \text{proof} \rangle$

lemma *ren-Lang-backward*: **assumes** $v \in \text{ren.Lang}$
shows $\exists w. v = \text{map } ra \ w \wedge w \in \text{Lang}$
 $\langle \text{proof} \rangle$

lemma *ren-Lang*: $\text{ren.Lang} = \text{map } ra \text{ } ' \text{Lang}$
 $\langle \text{proof} \rangle$

lemma *ren-det*: **assumes** *deterministic*
shows *ren.deterministic*
 $\langle \text{proof} \rangle$

lemma *ren-upper-time*: **assumes** *upper-time-bound f*
shows *ren.upper-time-bound f*
 $\langle \text{proof} \rangle$

end

lemma *tm-renaming*: **assumes** $\text{valid-tm } (tm :: ('q, 'a)tm)$
and $\text{inj-on } (ra :: 'a \Rightarrow 'b) \ (\Gamma\text{-tm } tm)$
and $\text{inj-on } (rq :: 'q \Rightarrow 'p) \ (Q\text{-tm } tm)$
shows $\exists tm' :: ('p, 'b)tm.$
 $\text{valid-tm } tm' \wedge$
 $\text{Lang-tm } tm' = \text{map } ra \text{ } ' \text{Lang-tm } tm \wedge$
 $(\text{det-tm } tm \longrightarrow \text{det-tm } tm') \wedge$
 $(\forall f. \text{upperb-time-tm } tm \ f \longrightarrow \text{upperb-time-tm } tm' \ f)$
 $\langle \text{proof} \rangle$

end

6 Translating Multitape TMs to Singletape TMs

In this section we define the mapping from a multitape Turing machine to a singletape Turing machine. We further define soundness of the translation via several relations which establish a connection between configurations of both kinds of Turing machines.

The translation works both for deterministic and non-deterministic TMs. Moreover, we verify a quadratic overhead in runtime.

theory *Multi-Single-TM-Translation*

imports

Multitape-TM

Singletape-TM

STM-Renaming

begin

6.1 Definition of the Translation

datatype *'a tuple-symbol* = *NO-HAT 'a* | *HAT 'a*

datatype (*'a, 'k*) *st-tape-symbol* = *ST-LE (⊢)* | *TUPLE 'k ⇒ 'a tuple-symbol* | *INP 'a*

datatype *'a sym-or-bullet* = *SYM 'a* | *BULLET (·)*

datatype (*'a, 'q, 'k*) *st-states* =

R₁ 'a sym-or-bullet |

R₂ |

S₀ 'q |

S 'q 'k ⇒ 'a sym-or-bullet |

S₁ 'q 'k ⇒ 'a |

E₀ 'q 'k ⇒ 'a 'k ⇒ dir |

E 'q 'k ⇒ 'a sym-or-bullet 'k ⇒ dir |

Er 'q 'k ⇒ 'a sym-or-bullet 'k ⇒ dir 'k set |

El 'q 'k ⇒ 'a sym-or-bullet 'k ⇒ dir 'k set |

Em 'q 'k ⇒ 'a sym-or-bullet 'k ⇒ dir 'k set

type-synonym (*'a, 'q, 'k*)*mt-rule* = *'q × ('k ⇒ 'a) × 'q × ('k ⇒ 'a) × ('k ⇒ dir)*

context *multitape-tm*

begin

definition *R1-Set* **where** *R1-Set* = *SYM ' Σ ∪ {·}*

definition *gamma-set* :: (*'k ⇒ 'a tuple-symbol*) *set* **where**
gamma-set = (*UNIV :: 'k set*) → *NO-HAT ' Γ ∪ HAT ' Γ*

definition *Γ'* :: (*'a, 'k*) *st-tape-symbol set* **where**
Γ' = *TUPLE ' gamma-set ∪ INP ' Σ ∪ {⊢}*

definition $func\text{-}set = (UNIV :: 'k\ set) \rightarrow SYM\ ' \Gamma \cup \{\cdot\}$

definition $blank' :: ('a, 'k)\ st\text{-}tape\text{-}symbol\ \mathbf{where}\ blank' = TUPLE\ (\lambda\ -. NO\text{-}HAT\ blank)$

definition $hatLE' :: ('a, 'k)\ st\text{-}tape\text{-}symbol\ \mathbf{where}\ hatLE' = TUPLE\ (\lambda\ -. HAT\ LE)$

definition $encSym :: 'a \Rightarrow ('a, 'k)\ st\text{-}tape\text{-}symbol\ \mathbf{where}\ encSym\ a = (TUPLE\ (\lambda\ i.\ if\ i = 0\ then\ NO\text{-}HAT\ a\ else\ NO\text{-}HAT\ blank))$

definition $add\text{-}inp :: ('k \Rightarrow 'a\ tuple\text{-}symbol) \Rightarrow ('k \Rightarrow 'a\ sym\text{-}or\text{-}bullet) \Rightarrow ('k \Rightarrow 'a\ sym\text{-}or\text{-}bullet)\ \mathbf{where}$
 $add\text{-}inp\ inp\ inp2 = (\lambda\ k.\ case\ inp\ k\ of\ HAT\ s \Rightarrow SYM\ s\ | \ - \Rightarrow inp2\ k)$

definition $project\text{-}inp :: ('k \Rightarrow 'a\ sym\text{-}or\text{-}bullet) \Rightarrow ('k \Rightarrow 'a)\ \mathbf{where}$
 $project\text{-}inp\ inp = (\lambda\ k.\ case\ inp\ k\ of\ SYM\ s \Rightarrow s)$

definition $compute\text{-}idx\text{-}set :: ('k \Rightarrow 'a\ tuple\text{-}symbol) \Rightarrow ('k \Rightarrow 'a\ sym\text{-}or\text{-}bullet) \Rightarrow 'k\ set\ \mathbf{where}$
 $compute\text{-}idx\text{-}set\ tup\ ys = \{i.\ tup\ i \in HAT\ ' \Gamma \wedge ys\ i \in SYM\ ' \Gamma\}$

definition $update\text{-}ys :: ('k \Rightarrow 'a\ tuple\text{-}symbol) \Rightarrow ('k \Rightarrow 'a\ sym\text{-}or\text{-}bullet) \Rightarrow ('k \Rightarrow 'a\ sym\text{-}or\text{-}bullet)\ \mathbf{where}$
 $update\text{-}ys\ tup\ ys = (\lambda\ k.\ if\ k \in (compute\text{-}idx\text{-}set\ tup\ ys)\ then\ \cdot\ else\ ys\ k)$

definition $replace\text{-}sym :: ('k \Rightarrow 'a\ tuple\text{-}symbol) \Rightarrow ('k \Rightarrow 'a\ sym\text{-}or\text{-}bullet) \Rightarrow ('k \Rightarrow 'a\ tuple\text{-}symbol)\ \mathbf{where}$
 $replace\text{-}sym\ tup\ ys = (\lambda\ k.\ if\ k \in (compute\text{-}idx\text{-}set\ tup\ ys)\ then\ (case\ ys\ k\ of\ SYM\ a \Rightarrow NO\text{-}HAT\ a)\ else\ tup\ k)$

definition $place\text{-}hats\text{-}to\text{-}dir :: dir \Rightarrow ('k \Rightarrow 'a\ tuple\text{-}symbol) \Rightarrow ('k \Rightarrow dir) \Rightarrow 'k\ set \Rightarrow ('k \Rightarrow 'a\ tuple\text{-}symbol)\ \mathbf{where}$
 $place\text{-}hats\text{-}to\text{-}dir\ dir\ tup\ ds\ I = (\lambda\ k.\ (case\ tup\ k\ of\ NO\text{-}HAT\ a \Rightarrow if\ k \in I \wedge ds\ k = dir\ then\ HAT\ a\ else\ NO\text{-}HAT\ a\ | HAT\ a \Rightarrow HAT\ a))$

definition $place\text{-}hats\text{-}R :: ('k \Rightarrow 'a\ tuple\text{-}symbol) \Rightarrow ('k \Rightarrow dir) \Rightarrow 'k\ set \Rightarrow ('k \Rightarrow 'a\ tuple\text{-}symbol)\ \mathbf{where}$
 $place\text{-}hats\text{-}R = place\text{-}hats\text{-}to\text{-}dir\ dir.R$

definition $place\text{-}hats\text{-}M :: ('k \Rightarrow 'a\ tuple\text{-}symbol) \Rightarrow ('k \Rightarrow dir) \Rightarrow 'k\ set \Rightarrow ('k \Rightarrow 'a\ tuple\text{-}symbol)\ \mathbf{where}$
 $place\text{-}hats\text{-}M = place\text{-}hats\text{-}to\text{-}dir\ dir.N$

definition $place\text{-}hats\text{-}L :: ('k \Rightarrow 'a\ tuple\text{-}symbol) \Rightarrow ('k \Rightarrow dir) \Rightarrow 'k\ set \Rightarrow ('k \Rightarrow 'a\ tuple\text{-}symbol)\ \mathbf{where}$
 $place\text{-}hats\text{-}L = place\text{-}hats\text{-}to\text{-}dir\ dir.L$

definition $\delta' ::$

$((a, 'q, 'k) \text{ st-states} \times (a, 'k) \text{ st-tape-symbol} \times (a, 'q, 'k) \text{ st-states} \times (a, 'k) \text{ st-tape-symbol} \times \text{dir})\text{set}$

where

$$\begin{aligned} \delta' = & \{(R_1 \cdot, \vdash, R_1 \cdot, \vdash, \text{dir}.R)\} \\ & \cup (\lambda x. (R_1 \cdot, \text{INP } x, R_1 (\text{SYM } x), \text{hatLE}', \text{dir}.R)) \text{ ' } \Sigma \\ & \cup (\lambda (a,x). (R_1 (\text{SYM } a), \text{INP } x, R_1 (\text{SYM } x), \text{encSym } a, \text{dir}.R)) \text{ ' } (\Sigma \times \Sigma) \\ & \cup \{(R_1 \cdot, \text{blank}', R_2, \text{hatLE}', \text{dir}.L)\} \\ & \cup (\lambda a. (R_1 (\text{SYM } a), \text{blank}', R_2, \text{encSym } a, \text{dir}.L)) \text{ ' } \Sigma \\ & \cup (\lambda x. (R_2, x, R_2, x, \text{dir}.L)) \text{ ' } (\Gamma' - \{\vdash\}) \\ & \cup \{(R_2, \vdash, S_0 s, \vdash, \text{dir}.N)\} \\ & \cup (\lambda q. (S_0 q, \vdash, S q (\lambda -. \cdot), \vdash, \text{dir}.R)) \text{ ' } (Q - \{t,r\}) \\ & \cup (\lambda (q,\text{inp},t). (S q \text{inp}, \text{TUPLE } t, S q (\text{add-inp } t \text{inp}), \text{TUPLE } t, \text{dir}.R)) \text{ ' } (Q \\ & \times (\text{func-set} - (\text{UNIV} \rightarrow \text{SYM ' } \Gamma)) \times \text{gamma-set}) \\ & \cup (\lambda (q,\text{inp},a). (S q \text{inp}, a, S_1 q (\text{project-inp } \text{inp}), a, \text{dir}.L)) \text{ ' } (Q \times (\text{UNIV} \rightarrow \\ & \text{SYM ' } \Gamma) \times (\Gamma' - \{\vdash\})) \\ & \cup (\lambda ((q,a,q',b,d),t). (S_1 q a, t, E_0 q' b d, t, \text{dir}.N)) \text{ ' } (\delta \times \Gamma') \\ & \cup (\lambda ((q,a,d),t). (E_0 q a d, t, E q (\text{SYM } o a) d, t, \text{dir}.N)) \text{ ' } ((Q \times (\text{UNIV} \rightarrow \\ & \Gamma) \times \text{UNIV}) \times \Gamma') \\ & \cup (\lambda (q,d). (E q (\lambda -. \cdot) d, \vdash, S_0 q, \vdash, \text{dir}.N)) \text{ ' } (Q \times \text{UNIV}) \\ & \cup (\lambda (q,\text{ys},\text{ds},t). (E q \text{ys } \text{ds}, \text{TUPLE } t, E r q (\text{update-ys } t \text{ys}) \text{ds} (\text{compute-idx-set} \\ & t \text{ys}), \text{TUPLE}(\text{replace-sym } t \text{ys}), \text{dir}.R)) \text{ ' } (Q \times \text{func-set} \times \text{UNIV} \times \text{gamma-set}) \\ & \cup (\lambda (q,\text{ys},\text{ds},I,t). (E r q \text{ys } \text{ds } I, \text{TUPLE } t, E m q \text{ys } \text{ds } I, \text{TUPLE}(\text{place-hats-R} \\ & t \text{ds } I), \text{dir}.L)) \text{ ' } (Q \times \text{func-set} \times \text{UNIV} \times \text{UNIV} \times \text{gamma-set}) \\ & \cup (\lambda (q,\text{ys},\text{ds},I,t). (E m q \text{ys } \text{ds } I, \text{TUPLE } t, E l q \text{ys } \text{ds } I, \text{TUPLE}(\text{place-hats-M} \\ & t \text{ds } I), \text{dir}.L)) \text{ ' } (Q \times \text{func-set} \times \text{UNIV} \times \text{UNIV} \times \text{gamma-set}) \\ & \cup (\lambda (q,\text{ys},\text{ds},I,t). (E l q \text{ys } \text{ds } I, \text{TUPLE } t, E q \text{ys } \text{ds}, \text{TUPLE}(\text{place-hats-L} \\ & t \text{ds } I), \text{dir}.N)) \text{ ' } (Q \times \text{func-set} \times \text{UNIV} \times \text{UNIV} \times \text{gamma-set}) \\ & \cup (\lambda (q,\text{ys},\text{ds},I). (E l q \text{ys } \text{ds } I, \vdash, E q \text{ys } \text{ds}, \vdash, \text{dir}.N)) \text{ ' } (Q \times \text{func-set} \times \\ & \text{UNIV} \times \text{Pow}(\text{UNIV})) \text{ — first switch into E state, so E phase is always finished in} \\ & \text{E state} \end{aligned}$$

definition $Q' =$

$$\begin{aligned} & R_1 \text{ ' } R1\text{-Set} \cup \{R_2\} \cup \\ & S_0 \text{ ' } Q \cup (\lambda (q,\text{inp}). S q \text{inp}) \text{ ' } (Q \times \text{func-set}) \cup (\lambda (q,a). S_1 q a) \text{ ' } (Q \times (\text{UNIV} \\ & \rightarrow \Gamma)) \cup \\ & (\lambda (q,a,d). E_0 q a d) \text{ ' } (Q \times (\text{UNIV} \rightarrow \Gamma) \times \text{UNIV}) \cup \\ & (\lambda (q,a,d). E q a d) \text{ ' } (Q \times \text{func-set} \times \text{UNIV}) \cup \\ & (\lambda (q,a,d,I). E r q a d I) \text{ ' } (Q \times \text{func-set} \times \text{UNIV} \times \text{UNIV}) \cup \\ & (\lambda (q,a,d,I). E m q a d I) \text{ ' } (Q \times \text{func-set} \times \text{UNIV} \times \text{UNIV}) \cup \\ & (\lambda (q,a,d,I). E l q a d I) \text{ ' } (Q \times \text{func-set} \times \text{UNIV} \times \text{UNIV}) \end{aligned}$$

lemma $\text{compute-idx-range}[\text{simp},\text{intro}]$:

assumes $tup \in \text{gamma-set}$

assumes $\text{ys} \in \text{func-set}$

shows $\text{compute-idx-set } tup \text{ys} \in \text{UNIV}$

<proof>

lemma *update-ys-range*[*simp,intro*]:
assumes $tup \in \text{gamma-set}$
assumes $ys \in \text{func-set}$
shows $\text{update-ys } tup \ ys \in \text{func-set}$
 $\langle \text{proof} \rangle$

lemma *replace-sym-range*[*simp,intro*]:
assumes $tup \in \text{gamma-set}$
assumes $ys \in \text{func-set}$
shows $\text{replace-sym } tup \ ys \in \text{gamma-set}$
 $\langle \text{proof} \rangle$

lemma *tup-hat-content*:
assumes $tup \in \text{gamma-set}$
assumes $tup \ x = \text{HAT } a$
shows $a \in \Gamma$
 $\langle \text{proof} \rangle$

lemma *tup-no-hat-content*:
assumes $tup \in \text{gamma-set}$
assumes $tup \ x = \text{NO-HAT } a$
shows $a \in \Gamma$
 $\langle \text{proof} \rangle$

lemma *place-hats-to-dir-range*[*simp, intro*]:
assumes $tup \in \text{gamma-set}$
shows $\text{place-hats-to-dir } d \ tup \ ds \ I \in \text{gamma-set}$
 $\langle \text{proof} \rangle$

lemma *place-hats-range*[*simp,intro*]:
assumes $tup \in \text{gamma-set}$
shows $\text{place-hats-R } tup \ ds \ I \in \text{gamma-set}$ **and**
 $\text{place-hats-L } tup \ ds \ I \in \text{gamma-set}$ **and**
 $\text{place-hats-M } tup \ ds \ I \in \text{gamma-set}$
 $\langle \text{proof} \rangle$

lemma *fin-R1-Set*[*intro,simp*]: *finite R1-Set*
 $\langle \text{proof} \rangle$

lemma *fin-gamma-set*[*intro,simp*]: *finite gamma-set*
 $\langle \text{proof} \rangle$

lemma *fin- Γ'* [*intro,simp*]: *finite Γ'*
 $\langle \text{proof} \rangle$

lemma *fin-func-set*[*simp,intro*]: *finite func-set*
 $\langle \text{proof} \rangle$

lemma *memberships*[*simp,intro*]: $\vdash \in \Gamma'$
 $\cdot \in R1\text{-Set}$
 $x \in \Sigma \implies \text{SYM } x \in R1\text{-Set}$
 $x \in \Sigma \implies \text{encSym } x \in \Gamma'$
 $\text{blank}' \in \Gamma'$
 $\text{hatLE}' \in \Gamma'$
 $x \in \Sigma \implies \text{INP } x \in \Gamma'$
 $y \in \text{gamma-set} \implies \text{TUPLE } y \in \Gamma'$
 $(\lambda \cdot. \cdot) \in \text{func-set}$
 $f \in \text{UNIV} \rightarrow \text{SYM } ' \Gamma \implies f \in \text{func-set}$
 $g \in \text{UNIV} \rightarrow \Gamma \implies \text{SYM } \circ g \in \text{func-set}$
 $f \in \text{UNIV} \rightarrow \text{SYM } ' \Gamma \implies \text{project-inp } f k \in \Gamma$
 $\langle \text{proof} \rangle$

lemma *add-inp-func-set*[*simp,intro*]: $b \in \text{gamma-set} \implies a \in \text{func-set} \implies \text{add-inp } b a \in \text{func-set}$
 $\langle \text{proof} \rangle$

lemma *automation*[*simp*]: $\bigwedge a b A B. (S a b \in (\lambda x. \text{case } x \text{ of } (x1, x2) \Rightarrow S x1 x2) ' (A \times B)) \longleftrightarrow (a \in A \wedge b \in B)$
 $\bigwedge a b A B. (S_1 a b \in (\lambda x. \text{case } x \text{ of } (x1, x2) \Rightarrow S_1 x1 x2) ' (A \times B)) \longleftrightarrow (a \in A \wedge b \in B)$
 $\bigwedge a b c A B C. (E_0 a b c \in (\lambda x. \text{case } x \text{ of } (x1, x2, x3) \Rightarrow E_0 x1 x2 x3) ' (A \times B \times C)) \longleftrightarrow (a \in A \wedge b \in B \wedge c \in C)$
 $\bigwedge a b c A B C. (E a b c \in (\lambda x. \text{case } x \text{ of } (x1, x2, x3) \Rightarrow E x1 x2 x3) ' (A \times B \times C)) \longleftrightarrow (a \in A \wedge b \in B \wedge c \in C)$
 $\bigwedge a b c d A B C. (Er a b c d \in (\lambda x. \text{case } x \text{ of } (x1, x2, x3, x4) \Rightarrow Er x1 x2 x3 x4) ' (A \times B \times C)) \longleftrightarrow (a \in A \wedge b \in B \wedge (c,d) \in C)$
 $\bigwedge a b c d A B C. (Em a b c d \in (\lambda x. \text{case } x \text{ of } (x1, x2, x3, x4) \Rightarrow Em x1 x2 x3 x4) ' (A \times B \times C)) \longleftrightarrow (a \in A \wedge b \in B \wedge (c,d) \in C)$
 $\bigwedge a b c d A B C. (El a b c d \in (\lambda x. \text{case } x \text{ of } (x1, x2, x3, x4) \Rightarrow El x1 x2 x3 x4) ' (A \times B \times C)) \longleftrightarrow (a \in A \wedge b \in B \wedge (c,d) \in C)$
 $\text{blank}' \neq \vdash$
 $\vdash \neq \text{blank}'$
 $\text{blank}' \neq \text{INP } x$
 $\text{INP } x \neq \text{blank}'$
 $\langle \text{proof} \rangle$

interpretation *st: singletape-tm* $Q' (\text{INP } ' \Sigma) \Gamma' \text{blank}' \vdash \delta' R_1 \cdot S_0 t S_0 r$
 $\langle \text{proof} \rangle$

lemma *valid-st: singletape-tm* $Q' (\text{INP } ' \Sigma) \Gamma' \text{blank}' \vdash \delta' (R_1 \cdot) (S_0 t) (S_0 r)$
 $\langle \text{proof} \rangle$

Determinism is preserved.

lemma *det-preservation: deterministic* $\implies \text{st.deterministic}$
 $\langle \text{proof} \rangle$

6.2 Soundness of the Translation

lemma *range-mt-pos*:

$\exists i. \text{Max} (\text{range} (\text{mt-pos } cm)) = \text{mt-pos } cm \ i$
 $\text{finite} (\text{range} (\text{mt-pos } (cm :: ('a, 'q, 'k) \text{ mt-config})))$
 $\text{range} (\text{mt-pos } cm) \neq \{\}$
 $\langle \text{proof} \rangle$

lemma *max-mt-pos-step*: **assumes** $(cm, cm') \in \text{step}$

shows $\text{Max} (\text{range} (\text{mt-pos } cm')) \leq \text{Suc} (\text{Max} (\text{range} (\text{mt-pos } cm)))$
 $\langle \text{proof} \rangle$

lemma *max-mt-pos-init*: $\text{Max} (\text{range} (\text{mt-pos} (\text{init-config } w))) = 0$

$\langle \text{proof} \rangle$

lemma *INP-D*: **assumes** $\text{set } x \subseteq \text{INP } \Sigma$

shows $\exists w. x = \text{map } \text{INP } w \wedge \text{set } w \subseteq \Sigma$
 $\langle \text{proof} \rangle$

6.2.1 R-Phase

fun *enc* :: $('a, 'q, 'k) \text{ mt-config} \Rightarrow \text{nat} \Rightarrow ('a, 'k) \text{ st-tape-symbol}$

where $\text{enc} (\text{Config}_M \ q \ tc \ p) \ n = \text{TUPLE } (\lambda k. \text{if } p \ k = n \ \text{then } \text{HAT } (tc \ k \ n) \ \text{else } \text{NO-HAT } (tc \ k \ n))$

inductive *rel-R₁* :: $((('a, 'k) \text{ st-tape-symbol}, ('a, 'q, 'k) \text{ st-states}) \text{st-config} \Rightarrow 'a \ \text{list} \Rightarrow \text{nat} \Rightarrow \text{bool})$ **where**

$n = \text{length } w \Longrightarrow$

$tc' \ 0 = \vdash \Longrightarrow$

$p' \leq n \Longrightarrow$

$(\bigwedge i. i < p' \Longrightarrow \text{enc} (\text{init-config } w) \ i = tc' (\text{Suc } i)) \Longrightarrow$

$(\bigwedge i. i \geq p' \Longrightarrow tc' (\text{Suc } i) = (\text{if } i < n \ \text{then } \text{INP } (w \ ! \ i) \ \text{else } \text{blank}')) \Longrightarrow$

$(p' = 0 \Longrightarrow q' = \cdot) \Longrightarrow$

$(\bigwedge p. p' = \text{Suc } p \Longrightarrow q' = \text{SYM } (w \ ! \ p)) \Longrightarrow$

$\text{rel-R}_1 (\text{Config}_S (R_1 \ q') \ tc' (\text{Suc } p')) \ w \ p'$

lemma *rel-R₁-init*: **shows** $\exists cs1. (\text{st.init-config} (\text{map } \text{INP } w), cs1) \in \text{st.dstep} \wedge \text{rel-R}_1 \ cs1 \ w \ 0$

$\langle \text{proof} \rangle$

lemma *rel-R₁-R₁*: **assumes** $\text{rel-R}_1 \ cs0 \ w \ j$

and $j < \text{length } w$

and $\text{set } w \subseteq \Sigma$

shows $\exists cs1. (cs0, cs1) \in \text{st.dstep} \wedge \text{rel-R}_1 \ cs1 \ w \ (\text{Suc } j)$

$\langle \text{proof} \rangle$

inductive *rel-R₂* :: $((('a, 'k) \text{ st-tape-symbol}, ('a, 'q, 'k) \text{ st-states}) \text{st-config} \Rightarrow 'a \ \text{list} \Rightarrow \text{nat} \Rightarrow \text{bool})$ **where**

$tc' \ 0 = \vdash \Longrightarrow$

$(\bigwedge i. \text{enc}(\text{init-config } w) i = \text{tc}'(\text{Suc } i)) \implies$
 $p \leq \text{length } w \implies$
 $\text{rel-R}_2(\text{Config}_S R_2 \text{tc}' p) w p$

lemma *rel-R₁-R₂*: **assumes** *rel-R₁ cs0 w (length w)*
and *set w \subseteq Σ*
shows $\exists cs1. (cs0, cs1) \in \text{st.dstep} \wedge \text{rel-R}_2 cs1 w (\text{length } w)$
 $\langle \text{proof} \rangle$

lemma *rel-R₂-R₂*: **assumes** *rel-R₂ cs0 w (Suc j)*
and *set w \subseteq Σ*
shows $\exists cs1. (cs0, cs1) \in \text{st.dstep} \wedge \text{rel-R}_2 cs1 w j$
 $\langle \text{proof} \rangle$

inductive *rel-S₀* :: $((\text{'a}, \text{'k}) \text{st-tape-symbol}, (\text{'a}, \text{'q}, \text{'k}) \text{st-states}) \text{st-config} \Rightarrow (\text{'a}, \text{'q}, \text{'k}) \text{mt-config} \Rightarrow \text{bool}$ **where**
 $\text{tc}' 0 = \vdash \implies$
 $(\bigwedge i. \text{tc}'(\text{Suc } i) = \text{enc}(\text{Config}_M q \text{tc } p) i) \implies$
 $\text{valid-config}(\text{Config}_M q \text{tc } p) \implies$
 $\text{rel-S}_0(\text{Config}_S(S_0 q) \text{tc}' 0) (\text{Config}_M q \text{tc } p)$

lemma *rel-R₂-S₀*: **assumes** *rel-R₂ cs0 w 0*
and *set w \subseteq Σ*
shows $\exists cs1. (cs0, cs1) \in \text{st.dstep} \wedge \text{rel-S}_0 cs1 (\text{init-config } w)$
 $\langle \text{proof} \rangle$

If we start with a proper word w as input on the singletape TM, then via the R-phase one can switch to the beginning of the S-phase (*rel-S₀*) for the initial configuration.

lemma *R-phase*: **assumes** *set w \subseteq Σ*
shows $\exists cs. (\text{st.init-config}(\text{map } \text{INP } w), cs) \in \text{st.dstep} \wedge (\exists + 2 * \text{length } w) \wedge$
 $\text{rel-S}_0 cs (\text{init-config } w)$
 $\langle \text{proof} \rangle$

6.2.2 S-Phase

inductive *rel-S* :: $((\text{'a}, \text{'k}) \text{st-tape-symbol}, (\text{'a}, \text{'q}, \text{'k}) \text{st-states}) \text{st-config} \Rightarrow (\text{'a}, \text{'q}, \text{'k}) \text{mt-config} \Rightarrow \text{nat} \Rightarrow \text{bool}$ **where**
 $\text{tc}' 0 = \vdash \implies$
 $(\bigwedge i. \text{tc}'(\text{Suc } i) = \text{enc}(\text{Config}_M q \text{tc } p) i) \implies$
 $\text{valid-config}(\text{Config}_M q \text{tc } p) \implies$
 $(\bigwedge i. \text{inp } i = (\text{if } p \ i < \ p' \ \text{then } \text{SYM}(\text{tc } i \ (p \ i)) \ \text{else } \cdot)) \implies$
 $\text{rel-S}(\text{Config}_S(S q \text{inp}) \text{tc}'(\text{Suc } p')) (\text{Config}_M q \text{tc } p) p'$

lemma *rel-S₀-S*: **assumes** *rel-S₀ cs0 cm*
and *mt-state cm \notin {t,r}*
shows $\exists cs1. (cs0, cs1) \in \text{st.dstep} \wedge \text{rel-S } cs1 \text{ cm } 0$

<proof>

lemma *rel-S-mem*: **assumes** *rel-S* (*Config_S* (*S q inp*) *tc' p'*) *cm j*
shows *inp* \in *func-set* \wedge *q* \in *Q* \wedge (\exists *t*. *tc'* (*Suc i*) = *TUPLE t* \wedge *t* \in *gamma-set*)

<proof>

lemma *rel-S-S*: **assumes** *rel-S cs0 cm p'*
p' \leq Max (range (mt-pos cm))
shows \exists *cs1*. (*cs0*, *cs1*) \in *st.dstep* \wedge *rel-S cs1 cm (Suc p')*
<proof>

inductive *rel-S₁* :: (*'a*, *'k*) *st-tape-symbol*, (*'a*, *'q*, *'k*) *st-states*) *st-config* \Rightarrow (*'a*, *'q*,
'k) *mt-config* \Rightarrow *bool* **where**
tc' 0 = \vdash \Longrightarrow
(\bigwedge *i*. *tc'* (*Suc i*) = *enc (Config_M q tc p) i*) \Longrightarrow
valid-config (Config_M q tc p) \Longrightarrow
(\bigwedge *i*. *inp i = tc i (p i)*) \Longrightarrow
(\bigwedge *i*. *p i < p'*) \Longrightarrow
p' = Suc (Max (range p)) \Longrightarrow
rel-S₁ (Config_S (S₁ q inp) tc' p') (Config_M q tc p)

lemma *rel-S-S₁*: **assumes** *rel-S cs0 cm p'*
p' = Suc (Max (range (mt-pos cm)))
shows \exists *cs1*. (*cs0*, *cs1*) \in *st.dstep* \wedge *rel-S₁ cs1 cm*
<proof>

If we start the S-phase (in *rel-S₀*), and the multitape-TM is not in a final state, then we can move to the end of the S-phase (in *rel-S₁*).

lemma *S-phase*: **assumes** *rel-S₀ cs cm*
and *mt-state cm \notin {t, r}*
shows \exists *cs'*. (*cs*, *cs'*) \in *st.dstep* \wedge (\exists + *Max (range (mt-pos cm))*) \wedge *rel-S₁ cs'*
cm
<proof>

6.2.3 E-Phase

context

fixes *rule* :: (*'a*, *'q*, *'k*) *mt-rule*

begin

inductive-set *δ step* :: (*'a*, *'q*, *'k*) *mt-config rel* **where**

δ step: rule = (q, a, q1, b, dir) \Longrightarrow

rule \in δ \Longrightarrow

(\bigwedge *k*. *ts k (n k) = a k*) \Longrightarrow

(\bigwedge *k*. *ts' k = (ts k)(n k := b k)*) \Longrightarrow

(\bigwedge *k*. *n' k = go-dir (dir k) (n k)*) \Longrightarrow

(*Config_M q ts n*, *Config_M q1 ts' n'*) \in *δ step*

end

lemma *step-to- δ step*: $(c1, c2) \in \text{step} \implies \exists \text{ rule}. (c1, c2) \in \delta\text{step rule}$
 ⟨proof⟩

lemma *δ step-to-step*: $(c1, c2) \in \delta\text{step rule} \implies (c1, c2) \in \text{step}$
 ⟨proof⟩

inductive *rel-E₀* :: $((a, 'k) \text{ st-tape-symbol}, (a, 'q, 'k) \text{ st-states})\text{st-config}$
 $\implies (a, 'q, 'k) \text{ mt-config} \implies (a, 'q, 'k) \text{ mt-config} \implies (a, 'q, 'k)\text{mt-rule} \implies \text{bool}$ **where**

$tc' \ 0 = \vdash \implies$
 $(\bigwedge i. tc' (Suc\ i) = \text{enc} (\text{Config}_M\ q\ tc\ p)\ i) \implies$
 $\text{valid-config} (\text{Config}_M\ q\ tc\ p) \implies$
 $\text{rule} = (q, a, q1, b, d) \implies$
 $(\text{Config}_M\ q\ tc\ p, \text{Config}_M\ q1\ tc1\ p1) \in \delta\text{step rule} \implies$
 $(\bigwedge i. p\ i < p') \implies$
 $p' = \text{Suc} (\text{Max} (\text{range}\ p)) \implies$
 $\text{rel-E}_0 (\text{Config}_S\ (E_0\ q1\ b\ d)\ tc'\ p') (\text{Config}_M\ q\ tc\ p) (\text{Config}_M\ q1\ tc1\ p1)\ \text{rule}$

For the transition between S and E phase we do not have deterministic steps. Therefore we add two lemmas: the former one is for showing that multitape can be simulated by singletape, and the latter one is for the inverse direction.

lemma *rel-S₁-E₀-step*: **assumes** *rel-S₁* $cs\ cm$
and $(cm, cm1) \in \text{step}$
shows $\exists \text{ rule } cs1. (cs, cs1) \in \text{st.step} \wedge \text{rel-E}_0\ cs1\ cm\ cm1\ \text{rule}$
 ⟨proof⟩

lemma *rel-S₁-E₀-st-step*: **assumes** *rel-S₁* $cs\ cm$
and $(cs, cs1) \in \text{st.step}$
shows $\exists cm1\ \text{rule}. (cm, cm1) \in \text{step} \wedge \text{rel-E}_0\ cs1\ cm\ cm1\ \text{rule}$
 ⟨proof⟩

fun *enc2* :: $(a, 'q, 'k) \text{ mt-config} \implies (a, 'q, 'k) \text{ mt-config} \implies \text{nat} \implies \text{nat} \implies (a, 'k)$
st-tape-symbol
where *enc2* $(\text{Config}_M\ q\ tc\ p) (\text{Config}_M\ q1\ tc1\ p1)\ p'\ n = \text{TUPLE} (\lambda k. \text{if } p\ k < p'$
 $\text{then if } p\ k = n \text{ then HAT } (tc\ k\ n) \text{ else NO-HAT } (tc\ k\ n)$
 $\text{else if } p1\ k = n \text{ then HAT } (tc1\ k\ n) \text{ else NO-HAT } (tc1\ k\ n))$

inductive *rel-E* :: $((a, 'k) \text{ st-tape-symbol}, (a, 'q, 'k) \text{ st-states})\text{st-config}$
 $\implies (a, 'q, 'k) \text{ mt-config} \implies (a, 'q, 'k) \text{ mt-config} \implies (a, 'q, 'k)\text{mt-rule} \implies \text{nat} \implies$
bool **where**
 $tc' \ 0 = \vdash \implies$
 $(\bigwedge i. tc' (Suc\ i) = \text{enc2} (\text{Config}_M\ q\ tc\ p) (\text{Config}_M\ q1\ tc1\ p1)\ p'\ i) \implies$
 $\text{valid-config} (\text{Config}_M\ q\ tc\ p) \implies$
 $\text{rule} = (q, a, q1, b, d) \implies$
 $(\text{Config}_M\ q\ tc\ p, \text{Config}_M\ q1\ tc1\ p1) \in \delta\text{step rule} \implies$
 $\text{bo} = (\lambda k. \text{if } p\ k < p' \text{ then SYM } (b\ k) \text{ else } \cdot) \implies$
 $\text{rel-E} (\text{Config}_S\ (E\ q1\ bo\ d)\ tc'\ p') (\text{Config}_M\ q\ tc\ p) (\text{Config}_M\ q1\ tc1\ p1)\ \text{rule } p'$

lemma *rel-E₀-E*: **assumes** *rel-E₀ cs cm cm1 rule*
shows $\exists cs1. (cs, cs1) \in st.dstep \wedge rel-E\ cs1\ cm\ cm1\ rule\ (Suc\ (Max\ (range\ (mt-pos\ cm))))$
 $\langle proof \rangle$

lemma *rel-E-S₀*: **assumes** *rel-E cs cm cm1 rule 0*
shows $\exists cs1. (cs, cs1) \in st.dstep \wedge rel-S_0\ cs1\ cm1$
 $\langle proof \rangle$

lemma *dsteps-to-steps*: $a \in st.dstep \rightsquigarrow n \implies a \in st.step \rightsquigarrow n$
 $\langle proof \rangle$

lemma *δ' -mem*: **assumes** $tup \in A$
and $f ' A \subseteq \delta'$
shows $f\ tup \in \delta'$
 $\langle proof \rangle$

lemma *rel-E-E*: **assumes** *rel-E cs cm cm1 rule (Suc p')*
shows $\exists cs1. (cs, cs1) \in st.dstep \rightsquigarrow 4 \wedge rel-E\ cs1\ cm\ cm1\ rule\ p'$
 $\langle proof \rangle$

lemma *E-phase*: **assumes** *rel-E₀ cs cm cm1 rule*
shows $\exists cs'. (cs, cs') \in st.dstep \rightsquigarrow (6 + 4 * Max\ (range\ (mt-pos\ cm))) \wedge rel-S_0\ cs'\ cm1$
 $\langle proof \rangle$

6.2.4 Simulation of multitape TM by singletape TM

lemma *step-simulation*: **assumes** *rel-S₀ cs cm*
and $(cm, cm') \in step$
shows $\exists cs'. (cs, cs') \in st.step \rightsquigarrow (10 + 5 * Max\ (range\ (mt-pos\ cm))) \wedge rel-S_0\ cs'\ cm'$
 $\langle proof \rangle$

lemma *steps-simulation-main*: **assumes** *rel-S₀ cs cm*
and $Max\ (range\ (mt-pos\ cm)) \leq N$
and $(cm, cm') \in step \rightsquigarrow n$
shows $\exists m\ cs'. (cs, cs') \in st.step \rightsquigarrow m \wedge rel-S_0\ cs'\ cm' \wedge m \leq sum\ (\lambda\ i. 10 + 5 * (N + i))\ \{.. < n\} \wedge Max\ (range\ (mt-pos\ cm')) \leq N + n$
 $\langle proof \rangle$

lemma *steps-simulation-rel-S₀*: **assumes** *rel-S₀ cs (init-config w)*
and $(init-config\ w, cm') \in step \rightsquigarrow n$
shows $\exists m\ cs'. (cs, cs') \in st.step \rightsquigarrow m \wedge rel-S_0\ cs'\ cm' \wedge m \leq 3 * n^2 + 7 * n$
 $\langle proof \rangle$

lemma *simulation-with-complexity*: **assumes** $w: set\ w \subseteq \Sigma$
and $steps: (init-config\ w, Config_M\ q\ mtape\ p) \in step \rightsquigarrow n$
shows $\exists\ stape\ k. (st.init-config\ (map\ INP\ w), Config_S\ (S_0\ q)\ stape\ 0) \in st.step \rightsquigarrow k$

$\wedge k \leq 2 * \text{length } w + 3 * n^2 + 7 * n + 3$
 ⟨proof⟩

lemma *simulation*: $\text{map } \text{INP} \text{ ' } \text{Lang} \subseteq \text{st.Lang}$
 ⟨proof⟩

6.2.5 Simulation of singletape TM by multitape TM

lemma *rev-simulation*: $\text{st.Lang} \subseteq \text{map } \text{INP} \text{ ' } \text{Lang}$
 ⟨proof⟩

lemma *rev-simulation-complexity*: **assumes** $w: \text{set } w \subseteq \Sigma$
and $\text{steps}: (\text{st.init-config } (\text{map } \text{INP } w), \text{cs}) \in \text{st.steps}^{\sim n}$
and $n: n \geq 2 * \text{length } w + 3 * k^2 + 7 * k + 3$
shows $\exists \text{cm}. (\text{init-config } w, \text{cm}) \in \text{step}^{\sim k}$
 ⟨proof⟩

6.2.6 Main Results

theorem *language-equivalence*: $\text{st.Lang} = \text{map } \text{INP} \text{ ' } \text{Lang}$
 ⟨proof⟩

theorem *upper-time-bound-quadratic-increase*: **assumes** *upper-time-bound* f
shows $\text{st.upper-time-bound } (\lambda n. 3 * (f n)^2 + 13 * f n + 2 * n + 12)$
 ⟨proof⟩
end

6.3 Main Results with Proper Renamings

By using the renaming capabilities we can get rid of the *map INP* in the language equivalence theorem. We just assume that there will always be enough symbols for the renaming, i.e., an infinite supply of fresh names is available.

theorem *multitape-to-singletape*: **assumes** *valid-mttm* $(\text{mttm} :: ('p, 'a, 'k :: \{\text{finite}, \text{zero}\}) \text{mttm})$

and *infinite* $(\text{UNIV} :: 'q \text{ set})$
and *infinite* $(\text{UNIV} :: 'a \text{ set})$
shows $\exists \text{tm} :: ('q, 'a) \text{tm}. \text{valid-tm } \text{tm} \wedge$
 $\text{Lang-mttm } \text{mttm} = \text{Lang-tm } \text{tm} \wedge$
 $(\text{det-mttm } \text{mttm} \longrightarrow \text{det-tm } \text{tm}) \wedge$
 $(\text{upperb-time-mttm } \text{mttm } f \longrightarrow \text{upperb-time-tm } \text{tm } (\lambda n. 3 * (f n)^2 + 13 * f n$
 $+ 2 * n + 12))$
 ⟨proof⟩

end

References

- [1] J. Hartmanis and R. E. Stearns. On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, 117:285–306, 1965.
- [2] J. E. Hopcroft. *Introduction to automata theory, languages, and computation*. 3. edition, 2014.
- [3] M. Sipser. *Introduction to the theory of computation*. 2. edition, 2006.