Hidden Markov Models

Simon Wimmer

December 7, 2022

Abstract

This entry contains a formalization of hidden Markov models [3] based on Johannes Hölzl's formalization of discrete time Markov chains [1]. The basic definitions are provided and the correctness of two main (dynamic programming) algorithms for hidden Markov models is proved: the forward algorithm for computing the likelihood of an observed sequence, and the Viterbi algorithm for decoding the most probable hidden state sequence. The Viterbi algorithm is made executable including memoization.

Hidden markov models have various applications in natural language processing. For an introduction see Jurafsky and Martin [2].

Contents

1 Hidden Markov Models

theory *Hidden-Markov-Model*
imports
Markov-Models.Discrete-Time-Markov-Chain Auxiliary
HOL−Library.IArray
begin

1.1 Definitions

Definition of Markov Kernels that are closed w.r.t. to a set of states.

locale *Closed-Kernel* =
fixes $K :: 's \Rightarrow 't\ pmf$ **and** $S :: 't\ set$
assumes *finite*: *finite S*
and *wellformed*: $S \neq \{\}$
and *closed*: $\forall\ s.\ K\ s \subseteq S$

An HMM is parameterized by a Markov kernel for the transition probabilites between internal states, a Markov kernel for the output probabilities of observations, and a fixed set of observations.

locale *HMM-defs* =
fixes $\mathcal{K} :: 's \Rightarrow 's\ pmf$ **and** $\mathcal{O} :: 's \Rightarrow 't\ pmf$ **and** $\mathcal{O}_s :: 't\ set$

locale *HMM* =
HMM-defs + *O*: *Closed-Kernel* $\mathcal{O}\ \mathcal{O}_s$
begin

lemma *observations-finite*: *finite* $\mathcal{O}_s$
and *observations-wellformed*: $\mathcal{O}_s \neq \{\}$
and *observations-closed*: $\forall\ s.\ \mathcal{O}\ s \subseteq \mathcal{O}_s$
⟨*proof*⟩

end

Fixed set of internal states.

locale *HMM2-defs* = *HMM-defs* $\mathcal{K}\ \mathcal{O}$ **for** $\mathcal{K} :: 's \Rightarrow 's\ pmf$ **and** $\mathcal{O} :: 's \Rightarrow 't\ pmf$ +
fixes $\mathcal{S} :: 's\ set$

locale *HMM2* = *HMM2-defs* + *HMM* + *K*: *Closed-Kernel* $\mathcal{K}\ \mathcal{S}$
begin

lemma *states-finite*: *finite* $\mathcal{S}$
and *states-wellformed*: $\mathcal{S} \neq \{\}$
and *states-closed*: $\forall\ s.\ \mathcal{K}\ s \subseteq \mathcal{S}$
⟨*proof*⟩

end

The set of internal states is now given as a list to iterate over. This is needed for the computations on HMMs.

locale *HMM3-defs* = *HMM2-defs* $\mathcal{O}_s\ \mathcal{K}$ **for** $\mathcal{O}_s :: 't\ set$ **and** $\mathcal{K} :: 's \Rightarrow 's\ pmf$ +
fixes *state-list* :: *'s list*

locale *HMM3* = *HMM3-defs* - - $\mathcal{O}_s\ \mathcal{K}$ + *HMM2* $\mathcal{O}_s\ \mathcal{K}$ **for** $\mathcal{O}_s :: 't\ set$ **and** $\mathcal{K} :: 's \Rightarrow 's\ pmf$ +
assumes *state-list-*$\mathcal{S}$: *set state-list* = $\mathcal{S}$

context *HMM-defs*
begin

no-notation (*ASCII*) *comp* (**infixl** *o* 55)

The "default" observation.

definition
$obs \equiv SOME\ x.\ x \in \mathcal{O}_s$

lemma (**in** *HMM*) *obs*:
$obs \in \mathcal{O}_s$
⟨*proof*⟩

The HMM is encoded as a Markov chain over pairs of states and observations. This is the Markov chain's defining Markov kernel.

definition
$K \equiv \lambda\ (s_1,\ o_1 :: 't).\ bind\text{-}pmf\ (\mathcal{K}\ s_1)\ (\lambda\ s_2.\ map\text{-}pmf\ (\lambda\ o_2.\ (s_2,\ o_2))\ (\mathcal{O}\ s_2))$

sublocale *MC-syntax K* ⟨*proof*⟩

Uniform distribution of the pairs $(s,\ o)$ for a fixed state s.

definition $I\ (s :: 's) = map\text{-}pmf\ (\lambda\ x.\ (s,\ x))\ (pmf\text{-}of\text{-}set\ \mathcal{O}_s)$

The likelihood of an observation sequence given a starting state s is defined in terms of the trace space of the Markov kernel given the uniform distribution of pairs for s.

definition
$likelihood\ s\ os = T'\ (I\ s)\ \{\omega \in space\ S.\ \exists\ o_0\ xs\ \omega'.\ \omega = (s,\ o_0)\ \#\#\ xs\ @-\ \omega' \wedge map\ snd\ xs = os\}$

abbreviation (*input*) $L\ os\ \omega \equiv \exists\ xs\ \omega'.\ \omega = xs\ @-\ \omega' \wedge map\ snd\ xs = os$

lemma *likelihood-alt-def*: $likelihood\ s\ os = T'\ (I\ s)\ \{(s,\ o)\ \#\#\ xs\ @-\ \omega'\ |o\ xs\ \omega'.\ map\ snd\ xs = os\}$
⟨*proof*⟩

1.2 Iteration Rule For Likelihood

lemma *L-Nil*:
$L\ []\ \omega = True$
⟨*proof*⟩

lemma *emeasure-T-observation-Nil*:
$T\ (s,\ o_0)\ \{\omega \in space\ S.\ L\ []\ \omega\} = 1$
⟨*proof*⟩

lemma *L-Cons*:
$L\ (o\ \#\ os)\ \omega \longleftrightarrow snd\ (shd\ \omega) = o \wedge L\ os\ (stl\ \omega)$
⟨*proof*⟩

lemma *L-measurable*[*measurable*]:
$Measurable.pred\ S\ (L\ os)$
⟨*proof*⟩

lemma *init-measurable*[*measurable*]:
$Measurable.pred\ S\ (\lambda x.\ \exists\, o_0\ xs\ \omega'.\ x = (s,\ o_0)\ \#\#\ xs\ @-\ \omega' \wedge map\ snd\ xs = os)$
(**is** $Measurable.pred\ S\ ?f$)
⟨*proof*⟩

lemma *T-init-observation-eq*:
$T\ (s,\ o)\ \{\omega \in space\ S.\ L\ os\ \omega\} = T\ (s,\ o')\ \{\omega \in space\ S.\ L\ os\ \omega\}$
⟨*proof*⟩

Shows that it is equivalent to define likelihood in terms of the trace space starting at a single pair of an internal state s and the default observation *obs*.

lemma (**in** *HMM*) *likelihood-init*:
$likelihood\ s\ os = T\ (s,\ obs)\ \{\omega \in space\ S.\ L\ os\ \omega\}$

⟨*proof*⟩

lemma *emeasure-T-observation-Cons*:
T (*s*, *o*$_0$) {$\omega \in$ *space S*. *L* (*o*$_1$ # *os*) ω} =
(∫$^+$ *t*. *ennreal* (*pmf* (*O t*) *o*$_1$) ∗ *T* (*t*, *o*$_1$) {$\omega \in$ *space S*. *L os* ω} ∂(*K s*)) (**is** *?l* = *?r*)
⟨*proof*⟩

1.3 Computation of Likelihood

fun *backward* **where**
backward s [] = *1* |
backward s (*o* # *os*) = (∫$^+$ *t*. *ennreal* (*pmf* (*O t*) *o*) ∗ *backward t os* ∂*measure-pmf* (*K s*))

lemma *emeasure-T-observation-backward*:
emeasure (*T* (*s*, *o*)) {$\omega \in$ *space S*. *L os* ω} = *backward s os*
⟨*proof*⟩

lemma (**in** *HMM*) *likelihood-backward*:
likelihood s os = *backward s os*
⟨*proof*⟩

end

context *HMM2*
begin

fun (**in** *HMM2-defs*) *forward* **where**
forward s t-end [] = *indicator* {*t-end*} *s* |
forward s t-end (*o* # *os*) =
(∑ *t* ∈ $\mathcal{S}$. *ennreal* (*pmf* (*O t*) *o*) ∗ *ennreal* (*pmf* (*K s*) *t*) ∗ *forward t t-end os*)

lemma *forward-split*:
forward s t (*os1* @ *os2*) = (∑ *t'* ∈ $\mathcal{S}$. *forward s t' os1* ∗ *forward t' t os2*)
if *s* ∈ $\mathcal{S}$
⟨*proof*⟩

lemma (**in** −)
(∑ *t* ∈ *S*. *f t*) = *f t* **if** *finite S t* ∈ *S* ∀ *s* ∈ *S* − {*t*}. *f s* = *0*
thm *sum.empty sum.insert sum.mono-neutral-right*[*of S* {*t*}]
⟨*proof*⟩

lemma *forward-backward*:
(∑ *t* ∈ $\mathcal{S}$. *forward s t os*) = *backward s os* **if** *s* ∈ $\mathcal{S}$
⟨*proof*⟩

theorem *likelihood-forward*:
likelihood s os = (∑ *t* ∈ $\mathcal{S}$. *forward s t os*) **if** ‹*s* ∈ $\mathcal{S}$›
⟨*proof*⟩

1.4 Definition of Maximum Probabilities

abbreviation (*input*) *V os as* $\omega \equiv (\exists\ \omega'.\ \omega =$ *zip as os* @− $\omega')$

definition
max-prob s os =
Max {*T'* (*I s*) {$\omega \in$ *space S*. ∃ *o* ω'. ω = (*s*, *o*) ## *zip as os* @− ω'}
| *as*. *length as* = *length os* ∧ *set as* ⊆ $\mathcal{S}$}

fun *viterbi-prob* **where**
viterbi-prob s t-end [] = *indicator* {*t-end*} *s* |

viterbi-prob s t-end (o # os) =
*(MAX t ∈ 𝒮. ennreal (pmf (𝒪 t) o * pmf (𝒦 s) t) * viterbi-prob t t-end os)*

definition
is-decoding s os as ≡
T' (I s) {ω ∈ space S. ∃ o ω'. ω = (s, o) ## zip as os @− ω'} = max-prob s os ∧
length as = length os ∧ set as ⊆ 𝒮

1.5 Iteration Rule For Maximum Probabilities

lemma *emeasure-T-state-Nil*:
T (s, o₀) {ω ∈ space S. V [] as ω} = 1
⟨*proof*⟩

lemma *max-prob-T-state-Nil*:
Max {T (s, o) {ω ∈ space S. V [] as ω} | as. length as = length [] ∧ set as ⊆ 𝒮} = 1
⟨*proof*⟩

lemma *V-Cons*: *V (o # os) (a # as) ω ⟷ fst (shd ω) = a ∧ snd (shd ω) = o ∧ V os as (stl ω)*
⟨*proof*⟩

lemma *measurable-V*[*measurable*]:
Measurable.pred S (λω. V os as ω)
⟨*proof*⟩

lemma *init-V-measurable*[*measurable*]:
*Measurable.pred S (λx. ∃ o ω'. x = (s, o) ## zip as os @− ω') (***is** *Measurable.pred S ?f)*
⟨*proof*⟩

lemma *max-prob-Cons'*:
Max {T (s, o₁) {ω ∈ space S. V (o # os) as ω} | as. length as = length (o # os) ∧ set as ⊆ 𝒮} =
(
*MAX t ∈ 𝒮. ennreal (pmf (𝒪 t) o * pmf (𝒦 s) t) **
(MAX as ∈ {as. length as = length os ∧ set as ⊆ 𝒮}. T (t, o) {ω ∈ space S. V os as ω})
*) (***is** *?l = ?r)*
and *T-V-Cons*:
T (s, o₁) {ω ∈ space S. V (o # os) (t # as) ω}
*= ennreal (pmf (𝒪 t) o * pmf (𝒦 s) t) * T (t, o) {ω ∈ space S. V os as ω}*
*(***is** *?l' = ?r')*
if *length as = length os*
⟨*proof*⟩

lemmas *max-prob-Cons = max-prob-Cons'*[*OF length-replicate*]

1.6 Computation of Maximum Probabilities

lemma *T-init-V-eq*:
T (s, o) {ω ∈ space S. V os as ω} = T (s, o') {ω ∈ space S. V os as ω}
⟨*proof*⟩

lemma *T'-I-T*:
T' (I s) {ω ∈ space S. ∃ o ω'. ω = (s, o) ## zip as os @− ω'} = T (s,o) {ω ∈ space S. V os as ω}
⟨*proof*⟩

lemma *max-prob-init*:
max-prob s os = Max {T (s,o) {ω ∈ space S. V os as ω} | as. length as = length os ∧ set as ⊆ 𝒮}
⟨*proof*⟩

lemma *max-prob-Nil*[*simp*]:
max-prob s [] = 1
⟨*proof*⟩

lemma *Max-start*:
(*MAX* $t \in \mathcal{S}$. (*indicator* $\{t\}$ s :: *ennreal*)) = *1* **if** $s \in \mathcal{S}$
⟨*proof*⟩

lemma *Max-V-viterbi*:
(*MAX* $t \in \mathcal{S}$. *viterbi-prob* s t *os*) =
Max $\{T\ (s,\ o)\ \{\omega \in$ *space* S. V *os as* $\omega\} \mid$ *as*. *length as* = *length os* $\wedge$ *set as* $\subseteq \mathcal{S}\}$ **if** $s \in \mathcal{S}$
⟨*proof*⟩

lemma *max-prob-viterbi*:
(*MAX* $t \in \mathcal{S}$. *viterbi-prob* s t *os*) = *max-prob s os* **if** $s \in \mathcal{S}$
⟨*proof*⟩

end

1.7 Decoding the Most Probable Hidden State Sequence

context *HMM3*
begin

fun *viterbi* **where**
viterbi s t-end [] = ([], *indicator* $\{$*t-end*$\}$ *s*) |
viterbi s t-end (*o* # *os*) = *fst* (
argmax snd (*map*
(λ*t*. *let* (*xs*, *v*) = *viterbi t t-end os in* (*t* # *xs*, *ennreal* (*pmf* ($\mathcal{O}$ *t*) *o* ∗ *pmf* ($\mathcal{K}$ *s*) *t*) ∗ *v*))
state-list))

lemma *state-list-nonempty*:
state-list ≠ []
⟨*proof*⟩

lemma *viterbi-viterbi-prob*:
snd (*viterbi s t-end os*) = *viterbi-prob s t-end os*
⟨*proof*⟩

context
begin

private fun *val-of* **where**
val-of s [] [] = *1* |
val-of s (*t* # *xs*) (*o* # *os*) = *ennreal* (*pmf* ($\mathcal{O}$ *t*) *o* ∗ *pmf* ($\mathcal{K}$ *s*) *t*) ∗ *val-of t xs os*

lemma *val-of-T*:
val-of s as os = *T* (*s*, o_1) $\{\omega \in$ *space S*. *V os as* $\omega\}$ **if** *length as* = *length os*
⟨*proof*⟩

lemma *viterbi-sequence*:
snd (*viterbi s t-end os*) = *val-of s* (*fst* (*viterbi s t-end os*)) *os*
if *snd* (*viterbi s t-end os*) > *0*
⟨*proof*⟩

lemma *viterbi-valid-path*:
length as = *length os* $\wedge$ *set as* $\subseteq \mathcal{S}$ **if** *viterbi s t-end os* = (*as*, *v*)
⟨*proof*⟩

definition
viterbi-final s os = *fst* (*argmax snd* (*map* (λ *t*. *viterbi s t os*) *state-list*))

lemma *viterbi-finalE*:
obtains *t* **where**

$t \in \mathcal{S}$ *viterbi-final s os* = *viterbi s t os*
snd (*viterbi s t os*) = *Max* ((λ*t*. *snd* (*viterbi s t os*)) ‘ $\mathcal{S}$)
⟨*proof*⟩

theorem *viterbi-final-max-prob*:
assumes *viterbi-final s os* = (*as*, *v*) $s \in \mathcal{S}$
shows *v* = *max-prob s os*
⟨*proof*⟩

theorem *viterbi-final-is-decoding*:
assumes *viterbi-final s os* = (*as*, *v*) *v* > *0* $s \in \mathcal{S}$
shows *is-decoding s os as*
⟨*proof*⟩

end

end

end

2 Implementation

theory *HMM-Implementation*
imports
Hidden-Markov-Model
Monad-Memo-DP.State-Main
begin

2.1 The Forward Algorithm

locale *HMM4* = *HMM3* - - - $\mathcal{O}_s$ $\mathcal{K}$ **for** $\mathcal{O}_s$ :: *'t set* **and** $\mathcal{K}$ :: *'s* ⇒ *'s pmf* +
assumes *states-distinct*: *distinct state-list*

context *HMM3-defs*
begin

context
fixes *os* :: *'t iarray*
begin

Alternative definition using indices into the list of states. The list of states is implemented as an immutable array for better performance.

function *forward-ix-rec* **where**
forward-ix-rec s t-end n = (*if n* ≥ *IArray.length os then indicator* {*t-end*} *s else*
(∑ *t* ← *state-list*.
ennreal (*pmf* ($\mathcal{O}$ *t*) (*os* !! *n*)) ∗ *ennreal* (*pmf* ($\mathcal{K}$ *s*) *t*) ∗ *forward-ix-rec t t-end* (*n* + *1*)))

⟨*proof*⟩
termination
⟨*proof*⟩

Memoization

memoize-fun *forward-ix*$_m$: *forward-ix-rec*
with-memory *dp-consistency-mapping*
monadifies (*state*) *forward-ix-rec.simps*[*unfolded Let-def*]
term *forward-ix*$_m$*'*
memoize-correct
⟨*proof*⟩

The main theorems generated by memoization.

context
includes *state-monad-syntax*
begin
thm *forward-ix$_m$'.simps forward-ix$_m$-def*
thm *forward-ix$_m$.memoized-correct*
end

end

definition
forward-ix os = forward-ix-rec (IArray os)

definition
likelihood-compute s os ≡
if s ∈ set state-list then Some (∑ t ← state-list. forward s t os) else None

end

Correctness of the alternative definition.

lemma (**in** *HMM3*) *forward-ix-drop-one*:
forward-ix (o # os) s t (n + 1) = forward-ix os s t n
⟨*proof*⟩

lemma (**in** *HMM4*) *forward-ix-forward*:
forward-ix os s t 0 = forward s t os
⟨*proof*⟩

Instructs the code generator to use this equation instead to execute *forward*. Uses the memoized version of *forward-ix*.

lemma (**in** *HMM4*) *forward-code* [*code*]:
forward s t os = fst (run-state (forward-ix$_m$' (IArray os) s t 0) Mapping.empty)
⟨*proof*⟩

theorem (**in** *HMM4*) *likelihood-compute*:
likelihood-compute s os = Some x ⟷ s ∈ 𝒮 ∧ x = likelihood s os
⟨*proof*⟩

2.2 The Viterbi Algorithm

context *HMM3-defs*
begin

context
fixes *os* :: *'t iarray*
begin

Alternative definition using indices into the list of states. The list of states is implemented as an immutable array for better performance.

function *viterbi-ix-rec* **where**
viterbi-ix-rec s t-end n = (if n ≥ IArray.length os then ([], indicator {t-end} s) else
fst (
argmax snd (map
(λt. let (xs, v) = viterbi-ix-rec t t-end (n + 1) in
*(t # xs, ennreal (pmf (𝒪 t) (os !! n) * pmf (𝒦 s) t) * v))*
state-list)))

⟨*proof*⟩
termination
⟨*proof*⟩

Memoization

memoize-fun *viterbi-ix$_m$*: *viterbi-ix-rec*
  with-memory *dp-consistency-mapping*
  monadifies (*state*) *viterbi-ix-rec.simps*[*unfolded Let-def*]

memoize-correct
  ⟨*proof*⟩

The main theorems generated by memoization.

context
  includes *state-monad-syntax*
begin
thm *viterbi-ix$_m$'.simps viterbi-ix$_m$-def*
thm *viterbi-ix$_m$.memoized-correct*
end

end

definition
  viterbi-ix os = *viterbi-ix-rec* (*IArray os*)

end

context *HMM3*
begin

lemma *viterbi-ix-drop-one*:
  viterbi-ix (*o* # *os*) *s t* (*n* + *1*) = *viterbi-ix os s t n*
  ⟨*proof*⟩

lemma *viterbi-ix-viterbi*:
  viterbi-ix os s t 0 = *viterbi s t os*
  ⟨*proof*⟩

lemma *viterbi-code* [*code*]:
  viterbi s t os = *fst* (*run-state* (*viterbi-ix$_m$'* (*IArray os*) *s t 0*) *Mapping.empty*)
  ⟨*proof*⟩

end

2.3 Misc

lemma *pmf-of-alist-support-aux-1*:
  assumes ∀ (-, *p*) ∈ *set μ*. *p* ≥ *0*
  shows (*0* :: *real*) ≤ (*case map-of μ x of None* ⇒ *0* | *Some p* ⇒ *p*)
  ⟨*proof*⟩

lemma *pmf-of-alist-support-aux-2*:
  assumes ∀ (-, *p*) ∈ *set μ*. *p* ≥ *0*
    and *sum-list* (*map snd μ*) = *1*
    and *distinct* (*map fst μ*)
  shows ∫$^+$ *x*. *ennreal* (*case map-of μ x of None* ⇒ *0* | *Some p* ⇒ *p*) ∂*count-space UNIV* = *1*
  ⟨*proof*⟩

lemma *pmf-of-alist-support*:
  assumes ∀ (-, *p*) ∈ *set μ*. *p* ≥ *0*
    and *sum-list* (*map snd μ*) = *1*
    and *distinct* (*map fst μ*)
  shows *set-pmf* (*pmf-of-alist μ*) ⊆ *fst* ‘ *set μ*
  ⟨*proof*⟩

Defining a Markov kernel from an association list.

locale *Closed-Kernel-From* =
  fixes K :: $('s \times ('t \times real)\ list)\ list$
    and S :: $'t\ list$
  assumes *wellformed*: $S \neq []$
      and *closed*: $\forall\ (s, \mu) \in set\ K.\ \forall\ (t, \text{-}) \in set\ \mu.\ t \in set\ S$
      and *is-pmf*:
        $\forall\ (\text{-}, \mu) \in set\ K.\ \forall\ (\text{-}, p) \in set\ \mu.\ p \geq 0$
        $\forall\ (\text{-}, \mu) \in set\ K.\ distinct\ (map\ fst\ \mu)$
        $\forall\ (s, \mu) \in set\ K.\ sum\text{-}list\ (map\ snd\ \mu) = 1$
      and *is-unique*:
        $distinct\ (map\ fst\ K)$
begin

definition
  $K'\ s \equiv case\ map\text{-}of\ (map\ (\lambda\ (s, \mu).\ (s, PMF\text{-}Impl.pmf\text{-}of\text{-}alist\ \mu))\ K)\ s\ of$
  $None \Rightarrow return\text{-}pmf\ (hd\ S)\ |$
  $Some\ s \Rightarrow s$

sublocale *Closed-Kernel* K' *set* S
  ⟨*proof*⟩

definition [*code*]:
  $K1 = map\text{-}of\ (map\ (\lambda\ (s, \mu).\ (s, map\text{-}of\ \mu))\ K)$

lemma *pmf-of-alist-aux*:
  assumes $(s, \mu) \in set\ K$
  shows
    $pmf\ (pmf\text{-}of\text{-}alist\ \mu)\ t = (case\ map\text{-}of\ \mu\ t\ of$
      $None \Rightarrow 0$
    $|\ Some\ p \Rightarrow p)$
  ⟨*proof*⟩

lemma *unique*: $\mu = \mu'$ **if** $(s, \mu) \in set\ K\ (s, \mu') \in set\ K$
  ⟨*proof*⟩

lemma (**in** −) *map-of-NoneD*:
  $x \notin fst\ `\ set\ M$ **if** $map\text{-}of\ M\ x = None$
  ⟨*proof*⟩

lemma K'*-code* [*code-post*]:
  $pmf\ (K'\ s)\ t = (case\ K1\ s\ of$
      $None \Rightarrow (if\ t = hd\ S\ then\ 1\ else\ 0)$
    $|\ Some\ \mu \Rightarrow case\ \mu\ t\ of$
        $None \Rightarrow 0$
      $|\ Some\ p \Rightarrow p$
)
  ⟨*proof*⟩

end

2.4 Executing Concrete HMMs

locale *Concrete-HMM-defs* =
  fixes $\mathcal{K}$ :: $('s \times ('s \times real)\ list)\ list$
    and $\mathcal{O}$ :: $('s \times ('t \times real)\ list)\ list$
    and $\mathcal{O}_s$ :: $'t\ list$
    and $\mathcal{K}_s$ :: $'s\ list$
begin

definition
$\mathcal{K}'\ s \equiv$ *case map-of* (*map* (λ (s, μ). (s, *PMF-Impl.pmf-of-alist* μ)) $\mathcal{K}$) s *of*
None $\Rightarrow$ *return-pmf* (*hd* $\mathcal{K}_s$) |
Some $s \Rightarrow s$

definition
$\mathcal{O}'\ s \equiv$ *case map-of* (*map* (λ (s, μ). (s, *PMF-Impl.pmf-of-alist* μ)) $\mathcal{O}$) s *of*
None $\Rightarrow$ *return-pmf* (*hd* $\mathcal{O}_s$) |
Some $s \Rightarrow s$

end

locale *Concrete-HMM = Concrete-HMM-defs +*
assumes *observations-wellformed'*: $\mathcal{O}_s \neq []$
and *observations-closed'*: $\forall$ (s, μ) $\in$ *set* $\mathcal{O}$. $\forall$ (t, -) $\in$ *set* μ. $t \in$ *set* $\mathcal{O}_s$
and *observations-form-pmf'*:
$\forall$ (-, μ) $\in$ *set* $\mathcal{O}$. $\forall$ (-, p) $\in$ *set* μ. $p \geq 0$
$\forall$ (-, μ) $\in$ *set* $\mathcal{O}$. *distinct* (*map fst* μ)
$\forall$ (s, μ) $\in$ *set* $\mathcal{O}$. *sum-list* (*map snd* μ) = 1
and *observations-unique*:
distinct (*map fst* $\mathcal{O}$)
assumes *states-wellformed*: $\mathcal{K}_s \neq []$
and *states-closed*: $\forall$ (s, μ) $\in$ *set* $\mathcal{K}$. $\forall$ (t, -) $\in$ *set* μ. $t \in$ *set* $\mathcal{K}_s$
and *states-form-pmf*:
$\forall$ (-, μ) $\in$ *set* $\mathcal{K}$. $\forall$ (-, p) $\in$ *set* μ. $p \geq 0$
$\forall$ (-, μ) $\in$ *set* $\mathcal{K}$. *distinct* (*map fst* μ)
$\forall$ (s, μ) $\in$ *set* $\mathcal{K}$. *sum-list* (*map snd* μ) = 1
and *states-unique*:
distinct (*map fst* $\mathcal{K}$) *distinct* $\mathcal{K}_s$
begin

interpretation *O*: *Closed-Kernel-From* $\mathcal{O}$ $\mathcal{O}_s$
rewrites $O.K' = \mathcal{O}'$
⟨*proof*⟩

interpretation *K*: *Closed-Kernel-From* $\mathcal{K}$ $\mathcal{K}_s$
rewrites $K.K' = \mathcal{K}'$
⟨*proof*⟩

lemmas *O-code* = *O.K'-code O.K1-def*
lemmas *K-code* = *K.K'-code K.K1-def*

sublocale *HMM-interp*: *HMM4* $\mathcal{O}'$ *set* $\mathcal{K}_s$ $\mathcal{K}_s$ *set* $\mathcal{O}_s$ $\mathcal{K}'$
⟨*proof*⟩

end

end

3 Example

theory *HMM-Example*
imports
HMM-Implementation
HOL−Library.AList-Mapping
begin

We would like to implement mappings as red-black trees but they require the key type to be linearly ordered. Unfortunately, *HOL−Analysis* fixes the product order to the element-wise order and thus we cannot restore a linear order, and the red-black tree implementation (from *HOL−Library*) cannot be

used.

The ice cream example from Jurafsky and Martin [2].

definition
states = [*''start''*, *''hot''*, *''cold''*, *''end''*]

definition *observations* :: *int list* **where**
observations = [*0*, *1*, *2*, *3*]

definition
kernel =
[
(*''start''*, [(*''hot''*,*0.8* :: *real*), (*''cold''*,*0.2*)]),
(*''hot''*, [(*''hot''*,*0.6* :: *real*), (*''cold''*,*0.3*), (*''end''*, *0.1*)]),
(*''cold''*, [(*''hot''*,*0.4* :: *real*), (*''cold''*,*0.5*), (*''end''*, *0.1*)]),
(*''end''*, [(*''end''*, *1*)])
]

definition
emissions =
[
(*''hot''*, [(*1*, *0.2*), (*2*, *0.4*), (*3*, *0.4*)]),
(*''cold''*, [(*1*, *0.5*), (*2*, *0.4*), (*3*, *0.1*)]),
(*''end''*, [(*0*, *1*)])
]

global-interpretation *Concrete-HMM kernel emissions observations states*
defines
viterbi-rec = *HMM-interp.viterbi-ix*$_m$*'*
and *viterbi* = *HMM-interp.viterbi*
and *viterbi-final* = *HMM-interp.viterbi-final*
and *forward-rec* = *HMM-interp.forward-ix*$_m$*'*
and *forward* = *HMM-interp.forward*
and *likelihood* = *HMM-interp.likelihood-compute*
⟨*proof*⟩

lemmas [*code*] = *HMM-interp.viterbi-ix*$_m$*'.simps*[*unfolded O-code K-code*]

lemmas [*code*] = *HMM-interp.forward-ix*$_m$*'.simps*[*unfolded O-code K-code*]

value *likelihood ''start''* [*1*, *1*, *1*]

If we enforce the last observation to correspond to *''end''*, then *forward* and *likelihood* yield the same result.

value *likelihood ''start''* [*1*, *1*, *1*, *0*]

value *forward ''start'' ''end''* [*1*, *1*, *1*, *0*]

value *forward ''start'' ''end''* [*3*, *3*, *3*, *0*]

value *forward ''start'' ''end''* [*3*, *1*, *3*, *0*]

value *forward ''start'' ''end''* [*3*, *1*, *3*, *1*, *0*]

value *viterbi ''start'' ''end''* [*1*, *1*, *1*, *0*]

value *viterbi ''start'' ''end''* [*3*, *3*, *3*, *0*]

value *viterbi ''start'' ''end''* [*3*, *1*, *3*, *0*]

value *viterbi ''start'' ''end'' [3, 1, 3, 1, 0]*

If we enforce the last observation to correspond to *''end''*, then *viterbi* and *viterbi-final* yield the same result.

value *viterbi-final ''start'' [3, 1, 3, 1, 0]*

value *viterbi-final ''start'' [1, 1, 1, 1, 1, 1, 1, 0]*

value *viterbi-final ''start'' [1, 1, 1, 1, 1, 1, 1, 1]*

end

References

[1] J. Hölzl. Markov chains and Markov decision processes in Isabelle/HOL. *Journal of Automated Reasoning*, 2017.

[2] D. Jurafsky and J. H. Martin. *Speech and language processing.* 2017.

[3] A. A. Markov. Essai d'une recherche statistique sur le texte du roman "Eugene Onegin" illustrant la liaison des epreuve en chain ('Example of a statistical investigation of the text of "Eugene Onegin" illustrating the dependence between samples in chain'). *Izvistia Imperatorskoi Akademii Nauk (Bulletin de l'Académie Impériale des Sciences de St.-Pétersbourg)*, 7:153–162, 1913. English translation by Morris Halle, 1956.