

Hidden Markov Models

Simon Wimmer

May 26, 2024

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	2
1.1	Definitions	2
1.2	Iteration Rule For Likelihood	3
1.3	Computation of Likelihood	4
1.4	Definition of Maximum Probabilities	4
1.5	Iteration Rule For Maximum Probabilities	5
1.6	Computation of Maximum Probabilities	5
1.7	Decoding the Most Probable Hidden State Sequence	6
2	Implementation	7
2.1	The Forward Algorithm	7
2.2	The Viterbi Algorithm	8
2.3	Misc	9
2.4	Executing Concrete HMMs	10
3	Example	11

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 ⇒ 't pmf and S :: 't set
  assumes finite: finite S
    and wellformed: S ≠ {}
    and closed: ∀ s. K s ⊆ S
```

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

```
locale HMM-defs =
  fixes K :: 's ⇒ 's pmf and O :: 's ⇒ 't pmf and Os :: 't set
```

```
locale HMM =
  HMM-defs + O: Closed-Kernel O Os
begin
```

```
lemma observations-finite: finite Os
  and observations-wellformed: Os ≠ {}
  and observations-closed: ∀ s. O s ⊆ Os
  ⟨proof⟩
```

end

Fixed set of internal states.

```
locale HMM2-defs = HMM-defs K O for K :: 's ⇒ 's pmf and O :: 's ⇒ 't pmf +
  fixes S :: 's set
```

```
locale HMM2 = HMM2-defs + HMM + K: Closed-Kernel K S
begin
```

```
lemma states-finite: finite S
  and states-wellformed: S ≠ {}
  and states-closed: ∀ s. K s ⊆ 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 Os K for Os :: 't set and K :: 's ⇒ 's pmf +
  fixes state-list :: 's list
```

```
locale HMM3 = HMM3-defs - - Os K + HMM2 Os K for Os :: 't set and K :: 's ⇒ 's pmf +
  assumes state-list-S: set state-list = 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

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

sublocale MC-syntax K *<proof>*

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

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

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: $likelihood\ s\ os = T'\ (I\ s)\ \{(s, o)\ \#\#\ xs\ @-\ \omega' \mid o\ xs\ \omega'.\ map\ snd\ xs = os\}$
<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:

$$likelihood\ s\ os = T\ (s, obs)\ \{\omega \in space\ S.\ L\ os\ \omega\}$$

$\langle \text{proof} \rangle$

lemma *emeasure-T-observation-Cons*:

$T (s, o_0) \{ \omega \in \text{space } S. L (o_1 \# os) \omega \} =$
 $(\int^+ t. \text{ennreal } (\text{pmf } (\mathcal{O} t) o_1) * T (t, o_1) \{ \omega \in \text{space } S. L os \omega \} \partial(\mathcal{K} s))$ (is ?l = ?r)
 $\langle \text{proof} \rangle$

1.3 Computation of Likelihood

fun *backward where*

backward s [] = 1 |
*backward s (o # os) = ($\int^+ t. \text{ennreal } (\text{pmf } (\mathcal{O} t) o) * \text{backward } t \text{ os } \partial \text{measure-pmf } (\mathcal{K} s)$)*

lemma *emeasure-T-observation-backward*:

emeasure (T (s, o)) { $\omega \in \text{space } S. L os \omega$ } = backward s os
 $\langle \text{proof} \rangle$

lemma (in *HMM*) *likelihood-backward*:

likelihood s os = backward s os
 $\langle \text{proof} \rangle$

end

context *HMM2*

begin

fun (in *HMM2-defs*) *forward where*

forward s t-end [] = indicator {t-end} s |
forward s t-end (o # os) =
 $(\sum t \in S. \text{ennreal } (\text{pmf } (\mathcal{O} t) o) * \text{ennreal } (\text{pmf } (\mathcal{K} s) t) * \text{forward } t \text{ t-end os})$

lemma *forward-split*:

*forward s t (os1 @ os2) = ($\sum t' \in S. \text{forward } s \ t' \ os1 * \text{forward } t' \ t \ os2$)*
if $s \in S$
 $\langle \text{proof} \rangle$

lemma (in $-$)

$(\sum t \in S. f t) = f t$ **if** *finite S* $t \in S \ \forall \ s \in S - \{t\}. f s = 0$
thm *sum.empty sum.insert sum.mono-neutral-right*[of $S \ \{t\}$]
 $\langle \text{proof} \rangle$

lemma *forward-backward*:

$(\sum t \in S. \text{forward } s \ t \ os) = \text{backward } s \ os$ **if** $s \in S$
 $\langle \text{proof} \rangle$

theorem *likelihood-forward*:

*likelihood s os = ($\sum t \in S. \text{forward } s \ t \ os)$ **if** $s \in S$*
 $\langle \text{proof} \rangle$

1.4 Definition of Maximum Probabilities

abbreviation (input) $V \ os \ as \ \omega \equiv (\exists \ \omega'. \ \omega = \text{zip } as \ os \ @- \ \omega')$

definition

max-prob s os =
 $\text{Max } \{ T' (I s) \{ \omega \in \text{space } S. \exists \ o \ \omega'. \ \omega = (s, o) \ \#\# \ \text{zip } as \ os \ @- \ \omega' \}$
 $\mid as. \text{length } as = \text{length } os \wedge \text{set } as \subseteq S \}$

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*:

$(\text{MAX } t \in \mathcal{S}. (\text{indicator } \{t\} s :: \text{ennreal})) = 1 \text{ if } s \in \mathcal{S}$
 $\langle \text{proof} \rangle$

lemma *Max-V-viterbi*:

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

lemma *max-prob-viterbi*:

$(\text{MAX } t \in \mathcal{S}. \text{viterbi-prob } s t os) = \text{max-prob } s os \text{ if } s \in \mathcal{S}$
 $\langle \text{proof} \rangle$

end

1.7 Decoding the Most Probable Hidden State Sequence

context *HMM3*

begin

fun *viterbi* **where**

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

lemma *state-list-nonempty*:

$\text{state-list} \neq []$
 $\langle \text{proof} \rangle$

lemma *viterbi-viterbi-prob*:

$\text{snd } (\text{viterbi } s t\text{-end } os) = \text{viterbi-prob } s t\text{-end } os$
 $\langle \text{proof} \rangle$

context

begin

private fun *val-of* **where**

$\text{val-of } s [] [] = 1 \mid$
 $\text{val-of } s (t \# xs) (o \# os) = \text{ennreal } (\text{pmf } (\mathcal{O} t) o * \text{pmf } (\mathcal{K} s) t) * \text{val-of } t xs os$

lemma *val-of-T*:

$\text{val-of } s as os = T (s, o_1) \{ \omega \in \text{space } S. V os as \omega \} \text{ if } \text{length } as = \text{length } os$
 $\langle \text{proof} \rangle$

lemma *viterbi-sequence*:

$\text{snd } (\text{viterbi } s t\text{-end } os) = \text{val-of } s (\text{fst } (\text{viterbi } s t\text{-end } os)) os$
if $\text{snd } (\text{viterbi } s t\text{-end } os) > 0$
 $\langle \text{proof} \rangle$

lemma *viterbi-valid-path*:

$\text{length } as = \text{length } os \wedge \text{set } as \subseteq \mathcal{S} \text{ if } \text{viterbi } s t\text{-end } os = (as, v)$
 $\langle \text{proof} \rangle$

definition

$\text{viterbi-final } s os = \text{fst } (\text{argmax } \text{snd } (\text{map } (\lambda t. \text{viterbi } s t os) \text{state-list}))$

lemma *viterbi-finalE*:

obtains t **where**

$t \in \mathcal{S}$ *viterbi-final s os* = *viterbi s t os*
 snd (*viterbi s t os*) = $Max ((\lambda t. snd$ (*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** $\mathcal{K} :: 's \Rightarrow 's$ *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* $n \geq IArray.length$ *os* *then indicator* {*t-end*} *s* *else*
 ($\sum t \leftarrow state-list.$
 $ennreal$ (*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-ixm'simps forward-ixm-def
thm forward-ixm.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-ixm' (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-ixm: 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-ixm' .simps viterbi-ixm-def  
thm viterbi-ixm.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-ixm' (IArray os) s t 0) Mapping.empty)  
  <proof>
```

```
end
```

2.3 Misc

```
lemma pmf-of-alist-support-aux-1:  
  assumes  $\forall (-, p) \in \text{set } \mu. p \geq 0$   
  shows  $(0 :: \text{real}) \leq (\text{case map-of } \mu \text{ } x \text{ of None } \Rightarrow 0 \mid \text{Some } p \Rightarrow p)$   
  <proof>
```

```
lemma pmf-of-alist-support-aux-2:  
  assumes  $\forall (-, p) \in \text{set } \mu. p \geq 0$   
  and sum-list (map snd  $\mu$ ) = 1  
  and distinct (map fst  $\mu$ )  
  shows  $\int^+ x. \text{ennreal } (\text{case map-of } \mu \text{ } x \text{ of None } \Rightarrow 0 \mid \text{Some } p \Rightarrow p) \partial \text{count-space UNIV} = 1$   
  <proof>
```

```
lemma pmf-of-alist-support:  
  assumes  $\forall (-, p) \in \text{set } \mu. p \geq 0$   
  and sum-list (map snd  $\mu$ ) = 1  
  and distinct (map fst  $\mu$ )  
  shows set-pmf (pmf-of-alist  $\mu$ )  $\subseteq$  fst ' set  $\mu$   
  <proof>
```

Defining a Markov kernel from an association list.

```

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

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

sublocale Closed-Kernel  $K'$  set  $S$ 
   $\langle \text{proof} \rangle$ 

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

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

lemma unique:  $\mu = \mu'$  if  $(s, \mu) \in \text{set } K (s, \mu') \in \text{set } K$ 
   $\langle \text{proof} \rangle$ 

lemma (in  $-$ ) map-of-NoneD:
   $x \notin \text{fst } ' \text{set } M$  if  $\text{map-of } M x = \text{None}$ 
   $\langle \text{proof} \rangle$ 

lemma K'-code [code-post]:
   $\text{pmf } (K' s) t = (\text{case } K1 s \text{ of}$ 
   $\text{None} \Rightarrow (\text{if } t = \text{hd } S \text{ then } 1 \text{ else } 0)$ 
   $\mid \text{Some } \mu \Rightarrow \text{case } \mu t \text{ of}$ 
   $\text{None} \Rightarrow 0$ 
   $\mid \text{Some } p \Rightarrow p$ 
   $)$ 
   $\langle \text{proof} \rangle$ 

end

```

2.4 Executing Concrete HMMs

```

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

```

definition

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

definition

$\mathcal{O}' s \equiv \text{case map-of } (\text{map } (\lambda (s, \mu). (s, \text{PMF-Impl.pmf-of-alist } \mu)) \mathcal{O}) s \text{ of}$
 $\text{None} \Rightarrow \text{return-pmf } (\text{hd } \mathcal{O}_s) \mid$
 $\text{Some } s \Rightarrow s$

end**locale** *Concrete-HMM* = *Concrete-HMM-defs* +

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

begin**interpretation** *O*: *Closed-Kernel-From* $\mathcal{O} \mathcal{O}_s$ **rewrites** $O.K' = \mathcal{O}'$ $\langle \text{proof} \rangle$ **interpretation** *K*: *Closed-Kernel-From* $\mathcal{K} \mathcal{K}_s$ **rewrites** $K.K' = \mathcal{K}'$ $\langle \text{proof} \rangle$ **lemmas** $O\text{-code} = O.K'\text{-code } O.K1\text{-def}$ **lemmas** $K\text{-code} = K.K'\text{-code } K.K1\text{-def}$ **sublocale** *HMM-interp*: *HMM4* $\mathcal{O}' \text{ set } \mathcal{K}_s \mathcal{K}_s \text{ set } \mathcal{O}_s \mathcal{K}'$ $\langle \text{proof} \rangle$ **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-ixm'  
and viterbi = HMM-interp.viterbi  
and viterbi-final = HMM-interp.viterbi-final  
and forward-rec = HMM-interp.forward-ixm'  
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'). *Izvestia Imperatorskoi Akademii Nauk (Bulletin de l'Académie Impériale des Sciences de St.-Pétersbourg)*, 7:153–162, 1913. English translation by Morris Halle, 1956.