Tries

Andreas Lochbihler and Tobias Nipkow

October 11, 2017

Abstract

This article formalizes the "trie" data structure invented by Fredkin [1]. It also provides a specialization where the entries in the trie are lists.

Contents

1 Trie

theory *Trie*
imports *HOL−Library.AList*
begin

datatype $('k, 'v)$ *trie* =
  Trie $'v$ *option* $('k * ('k,'v)trie)$ *list*

lemma *trie-induct* [*case-names Trie*, *induct type*]:
  $(\bigwedge vo\ kvs.\ (\bigwedge k\ t.\ (k, t) \in set\ kvs \Longrightarrow P\ t) \Longrightarrow P\ (Trie\ vo\ kvs)) \Longrightarrow P\ t$
⟨*proof*⟩

definition *empty-trie* :: ('*k*, '*v*) *trie* **where**
empty-trie = *Trie None* []

fun *is-empty-trie* :: ('*k*, '*v*) *trie* ⇒ *bool* **where**
is-empty-trie (*Trie v m*) = (*v* = *None* ∧ *m* = [])

fun *lookup-trie* :: ('*k*, '*v*) *trie* ⇒ '*k list* ⇒ '*v option* **where**
lookup-trie (*Trie v m*) [] = *v* |
lookup-trie (*Trie v m*) (*k*#*ks*) =
  (*case map-of m k of*
    None ⇒ *None* |
    Some st ⇒ *lookup-trie st ks*)

fun *update-with-trie* ::
  '*k list* ⇒ ('*v option* ⇒ '*v*) ⇒ ('*k*, '*v*) *trie* ⇒ ('*k*, '*v*) *trie* **where**
update-with-trie [] *f* (*Trie v ps*) = *Trie* (*Some*(*f v*)) *ps* |
update-with-trie (*k*#*ks*) *f* (*Trie v ps*) =
  Trie v (*AList.update-with-aux empty-trie k* (*update-with-trie ks f*) *ps*)

The function argument *f* of *update-with-trie* does not return an optional value because *None* could break the invariant that no empty tries are contained in a trie because *AList.update-with-aux* cannot recognise and remove empty tries. Therefore the delete function is implemented separately rather than via *update-with-trie*.

Do not use *update-with-trie* if most of the calls do not change the entry (because of the garbage this creates); use *lookup-trie* possibly followed by *update-trie*. This shortcoming could be addressed if *f* indicated that the entry is unchanged, eg by *None*.

definition *update-trie* :: '*k list* ⇒ '*v* ⇒ ('*k*, '*v*) *trie* ⇒ ('*k*, '*v*) *trie* **where**
update-trie ks v = *update-with-trie ks* (%-. *v*)

lemma *update-trie-induct*:
  ⟦⋀*v ps*. *P* [] (*Trie v ps*); ⋀*k ks v ps*. (⋀*x*. *P ks x*) ⟹ *P* (*k*#*ks*) (*Trie v ps*)⟧
⟹ *P xs t*
⟨*proof*⟩

lemma *update-trie-Nil*[*simp*]: *update-trie* [] *v* (*Trie vo ts*) = *Trie* (*Some v*) *ts*
⟨*proof*⟩

lemma *update-trie-Cons*[*simp*]: *update-trie* (*k*#*ks*) *v* (*Trie vo ts*) =
  Trie vo (*AList.update-with-aux* (*Trie None* []) *k* (*update-trie ks v*) *ts*)
⟨*proof*⟩

fun *delete-trie* :: '*key list* ⇒ ('*key*, '*val*) *trie* ⇒ ('*key*, '*val*) *trie*
where
delete-trie [] (*Trie vo ts*) = *Trie None ts* |

```
delete-trie (k#ks) (Trie vo ts) =
  (case map-of ts k of
    None ⇒ Trie vo ts |
    Some t ⇒ let t' = delete-trie ks t
          in if is-empty-trie t'
            then Trie vo (AList.delete-aux k ts)
            else Trie vo (AList.update k t' ts))

fun all-trie :: ('v ⇒ bool) ⇒ ('k, 'v) trie ⇒ bool where
all-trie p (Trie v ps) =
  ((case v of None ⇒ True | Some v ⇒ p v) ∧ (∀(k,t) ∈ set ps. all-trie p t))

fun invar-trie :: ('key, 'val) trie ⇒ bool where
invar-trie (Trie vo kts) =
  (distinct (map fst kts) ∧
   (∀(k, t) ∈ set kts. ¬ is-empty-trie t ∧ invar-trie t))
```

1.1 Empty trie

lemma *invar-empty* [*simp*]: *invar-trie empty-trie*
⟨*proof*⟩

lemma *is-empty-conv*: *is-empty-trie ts* ⟷ *ts* = *Trie None* []
⟨*proof*⟩

1.2 *lookup-trie*

lemma *lookup-empty* [*simp*]: *lookup-trie empty-trie* = *Map.empty*
⟨*proof*⟩

lemma *lookup-empty'* [*simp*]: *lookup-trie* (*Trie None* []) *ks* = *None*
⟨*proof*⟩

lemma *lookup-update*:
lookup-trie (*update-trie ks v t*) *ks'* = (*if ks* = *ks' then Some v else lookup-trie t ks'*)
⟨*proof*⟩

lemma *lookup-update'*:
lookup-trie (*update-trie ks v t*) = (*lookup-trie t*)(*ks* ↦ *v*)
⟨*proof*⟩

lemma *lookup-eq-Some-iff*:
assumes *invar*: *invar-trie* ((*Trie vo kvs*) :: ('*key*, '*val*) *trie*)
shows *lookup-trie* (*Trie vo kvs*) *ks* = *Some v* ⟷
  (*ks* = [] ∧ *vo* = *Some v*) ∨
  (∃ *k t ks'*. *ks* = *k* # *ks'* ∧
    (*k*, *t*) ∈ *set kvs* ∧ *lookup-trie t ks'* = *Some v*)
⟨*proof*⟩

lemma *lookup-eq-None-iff*:
assumes *invar*: *invar-trie* ((*Trie vo kvs*) :: (*'key*, *'val*) *trie*)
shows *lookup-trie* (*Trie vo kvs*) *ks* = *None* ⟷
    (*ks* = [] ∧ *vo* = *None*) ∨
    (∃ *k ks'*. *ks* = *k* # *ks'* ∧ (∀ *t*. (*k*, *t*) ∈ *set kvs* ⟶ *lookup-trie t ks'* = *None*))
⟨*proof*⟩

lemma *update-not-empty*: ¬ *is-empty-trie* (*update-trie ks v t*)
⟨*proof*⟩

lemma *invar-trie-update*: *invar-trie t* ⟹ *invar-trie* (*update-trie ks v t*)
⟨*proof*⟩

lemma *is-empty-lookup-empty*:
  invar-trie t ⟹ *is-empty-trie t* ⟷ *lookup-trie t* = *Map.empty*
⟨*proof*⟩

lemma *lookup-update-with-trie*:
  lookup-trie (*update-with-trie ks f t*) *ks'* =
  (*if ks'* = *ks then Some*(*f*(*lookup-trie t ks'*)) *else lookup-trie t ks'*)
⟨*proof*⟩

1.3 *delete-trie*

lemma *delete-eq-empty-lookup-other-fail*:
  ⟦ *delete-trie ks t* = *Trie None* []; *ks'* ≠ *ks* ⟧ ⟹ *lookup-trie t ks'* = *None*
⟨*proof*⟩

lemma *lookup-delete*: *invar-trie t* ⟹
  lookup-trie (*delete-trie ks t*) *ks'* =
  (*if ks* = *ks' then None else lookup-trie t ks'*)
⟨*proof*⟩

lemma *lookup-delete'*:
  invar-trie t ⟹ *lookup-trie* (*delete-trie ks t*) = (*lookup-trie t*)(*ks* := *None*)
⟨*proof*⟩

lemma *invar-trie-delete*:
  invar-trie t ⟹ *invar-trie* (*delete-trie ks t*)
⟨*proof*⟩

1.4 *update-with-trie*

lemma *nonempty-update-with-aux*: *AList.update-with-aux v k f ps* ≠ []
⟨*proof*⟩

lemma *nonempty-update-with-trie*: ¬ *is-empty-trie* (*update-with-trie ks f t*)
⟨*proof*⟩

lemma *invar-update-with-trie*:

invar-trie t $\Longrightarrow$ *invar-trie* (*update-with-trie ks f t*)
⟨*proof*⟩

1.5 Domain of a trie

lemma *dom-lookup*:
dom (*lookup-trie* (*Trie vo kts*)) =
($\bigcup$ *k*∈*dom* (*map-of kts*). *Cons k ' dom* (*lookup-trie* (*the* (*map-of kts k*)))) ∪
(*if vo* = *None then* {} *else* {[]})
⟨*proof*⟩

lemma *finite-dom-lookup*:
finite (*dom* (*lookup-trie t*))
⟨*proof*⟩

lemma *dom-lookup-empty-conv*: *invar-trie t* $\Longrightarrow$ *dom* (*lookup-trie t*) = {} $\longleftrightarrow$ *is-empty-trie t*
⟨*proof*⟩

1.6 Range of a trie

lemma *ran-lookup-Trie*: *invar-trie* (*Trie vo ps*) $\Longrightarrow$
ran (*lookup-trie* (*Trie vo ps*)) =
(*case vo of None* ⇒ {} | *Some v* ⇒ {*v*}) ∪ (*UN* (*k*,*t*) : *set ps*. *ran*(*lookup-trie t*))
⟨*proof*⟩

lemma *all-trie-eq-ran*:
invar-trie t $\Longrightarrow$ *all-trie P t* = (∀ *x* ∈ *ran*(*lookup-trie t*). *P x*)
⟨*proof*⟩

end

2 Tries (List Version)

theory *Tries*
imports *Trie*
begin

This is a specialization of tries where values are lists.

type-synonym ('*k*,'*v*)*tries* = ('*k*,'*v list*)*trie*

definition *lookup-tries* :: ('*k*,'*v*)*tries* ⇒ '*k list* ⇒ '*v list* **where**
lookup-tries t ks = (*case lookup-trie t ks of None* ⇒ [] | *Some vs* ⇒ *vs*)

definition *update-with-tries* ::
'*k list* ⇒ ('*v list* ⇒ '*v list*) ⇒ ('*k*, '*v*) *tries* ⇒ ('*k*, '*v*) *tries* **where**
update-with-tries ks f =
update-with-trie ks (λ*vo*. *case vo of None* ⇒ *f* [] | *Some vs* ⇒ *f vs*)

definition *insert-tries* :: *'k list* ⇒ *'v* ⇒ (*'k*,*'v*)*tries* ⇒ (*'k*,*'v*)*tries* **where**
insert-tries ks v =
update-with-trie ks (λ*vo*. *case vo of None* ⇒ [*v*] | *Some vs* ⇒ *v*#*vs*)

definition *inserts-tries* :: (*'v* ⇒ *'k list*) ⇒ *'v list* ⇒ (*'k*,*'v*)*tries* ⇒ (*'k*,*'v*)*tries* **where**
inserts-tries key = *fold* (%*v*. *insert-tries* (*key v*) *v*)

definition *tries-of-list* :: (*'v* ⇒ *'k list*) ⇒ *'v list* ⇒ (*'k*,*'v*)*tries* **where**
tries-of-list key vs = *inserts-tries key vs* (*Trie None* [])

definition *set-tries* :: (*'k*,*'v*)*tries* ⇒ *'v set* **where**
set-tries t = *Union* {*gs*. ∃ *a*. *gs* = *set*(*lookup-tries t a*)}

definition *all-tries* :: (*'v* ⇒ *bool*) ⇒ (*'k*,*'v*)*tries* ⇒ *bool* **where**
all-tries P = *all-trie* (*list-all P*)

2.1 *lookup-tries*

lemma *lookup-Nil*[*simp*]:
lookup-tries (*Trie vo ps*) [] = (*case vo of None* ⇒ [] | *Some vs* ⇒ *vs*)
⟨*proof*⟩

lemma *lookup-Cons*[*simp*]: *lookup-tries* (*Trie vo ps*) (*a*#*as*) =
(*case map-of ps a of None* ⇒ [] | *Some at* ⇒ *lookup-tries at as*)
⟨*proof*⟩

lemma *lookup-empty*[*simp*]: *lookup-tries* (*Trie None* []) *as* = []
⟨*proof*⟩

theorem *lookup-update*:
lookup-tries (*update-trie as vs t*) *bs* =
(*if as*=*bs then vs else lookup-tries t bs*)
⟨*proof*⟩

theorem *lookup-update-with*:
lookup-tries (*update-with-tries as f t*) *bs* =
(*if as*=*bs then f*(*lookup-tries t as*) *else lookup-tries t bs*)
⟨*proof*⟩

2.2 *insert-tries*, *inserts-tries*, *tries-of-list*

lemma *invar-insert-tries*: *invar-trie t* ⟹ *invar-trie*(*insert-tries as v t*)
⟨*proof*⟩

lemma *invar-inserts-tries*:
invar-trie t ⟹ *invar-trie* (*inserts-tries key xs t*)
⟨*proof*⟩

lemma *invar-of-list*: *invar-trie* (*tries-of-list key xs*)
⟨*proof*⟩

lemma *set-lookup-insert-tries*: *set* (*lookup-tries* (*insert-tries ks a t*) *ks'*) =
(*if ks'* = *ks then Set.insert a* (*set*(*lookup-tries t ks'*)) *else set*(*lookup-tries t ks'*))
⟨*proof*⟩

lemma *in-set-lookup-inserts-tries*:
(*v* ∈ *set*(*lookup-tries* (*inserts-tries key vs t*) (*key v*))) =
(*v* ∈ *set vs* ∪ *set*(*lookup-tries t* (*key v*)))
⟨*proof*⟩

lemma *in-set-lookup-of-list*:
v ∈ *set*(*lookup-tries* (*tries-of-list key vs*) (*key v*)) = (*v* ∈ *set vs*)
⟨*proof*⟩

lemma *in-set-lookup-inserts-triesD*:
v ∈ *set*(*lookup-tries* (*inserts-tries key vs t*) *xs*) ⟹
v ∈ *set vs* ∪ *set*(*lookup-tries t xs*)
⟨*proof*⟩

lemma *in-set-lookup-of-listD*:
v ∈ *set*(*lookup-tries* (*tries-of-list f vs*) *xs*) ⟹ *v* ∈ *set vs*
⟨*proof*⟩

2.3 *set-tries*

lemma *set-tries-eq-ran*: *set-tries t* = *Union*(*set* ‘ *ran*(*lookup-trie t*))
⟨*proof*⟩

lemma *set-tries-empty*[*simp*]: *set-tries* (*Trie None* []) = {}
⟨*proof*⟩

lemma *set-tries-insert*[*simp*]:
set-tries (*insert-tries a x t*) = *Set.insert x* (*set-tries t*)
⟨*proof*⟩

lemma *set-insert-tries*:
set-tries (*inserts-tries key xs t*) =
set xs Un set-tries t
⟨*proof*⟩

lemma *set-tries-of-list*[*simp*]:
set-tries(*tries-of-list key xs*) = *set xs*
⟨*proof*⟩

lemma *in-set-lookup-set-triesD*:
x∈*set* (*lookup-tries t a*) ⟹ *x* ∈ *set-tries t*
⟨*proof*⟩

end

References

[1] E. Fredkin. Trie memory. *Commun. ACM*, 3(9):490–499, Sept. 1960.