

This module verifies the correctness of the algorithm used to implement *DeletePrefix* in the go-immutable-radix project. (<https://github.com/hashicorp/go-immutable-radix>)

```

MODULE RadixDeletePrefix
EXTENDS FiniteSets, Integers, Sequences, TLC
INSTANCE RadixTrees

  Set of characters to use for the alphabet of generated strings.
  CONSTANT Alphabet

  Length of input strings generated
  CONSTANT MinLength, MaxLength
  ASSUME
     $\wedge \{MinLength, MaxLength\} \subseteq Nat$ 
     $\wedge MinLength \leq MaxLength$ 
     $\wedge MinLength > 0$ 

  Number of unique elements to construct the radix tree with. This
  is a set of numbers so you can test with inputs of multiple sizes.
  CONSTANT ElementCounts
  ASSUME ElementCounts  $\subseteq Nat$ 

  Inputs is the set of input strings valid for the tree.
  Inputs  $\triangleq \text{UNION } \{[1 \dots n \rightarrow Alphabet] : n \in MinLength \dots MaxLength\}$ 

  InputSets is the full set of possible inputs we can send to the radix tree.
  InputSets  $\triangleq \{T \in \text{SUBSET } Inputs : Cardinality(T) \in ElementCounts\}$ 

```

```

  TRUE iff seq is prefixed with prefix.
  HasPrefix(seq, prefix)  $\triangleq$ 
     $\wedge Len(seq) \geq Len(prefix)$ 
     $\wedge \forall i \in 1 \dots Len(prefix) : seq[i] = prefix[i]$ 

  Remove prefix from seq.
  TrimPrefix(seq, prefix)  $\triangleq [i \in 1 \dots (Len(seq) - Len(prefix)) \mapsto seq[i + Len(prefix)]]$ 

```

```

DeletePrefix should be equivalent to the tree without inputs that have that prefix.
This purposely doesn't model the "delete" algorithm at all: only the end result
of what the tree should contain.
Expected(input, prefix)  $\triangleq Range(RadixTree(\{value \in input : \neg HasPrefix(value, prefix)\}))$ 

--algorithm delete_prefix
variables
  input  $\in InputSets$ ,
  prefix  $\in Inputs$ ,

```

```

    root = RadixTree(input),
    newChild =  $\langle \rangle$ ,
    search = {},
    result = {};

define
    We determine if newChild is “nil” by checking if it has the empty domain,
    since a non-null child will be a function with domains prefix, value, etc.
    NewChildNull  $\triangleq$  DOMAIN newChild = {}
end define ;

    Precondition:  $Len(n.Edges) = 1$ 
procedure mergeChild( $n = \langle \rangle$ )
begin
    MergeChild:
    with
        label = CHOOSE  $x \in \text{DOMAIN } n.Edges : \text{TRUE}$ , we know we have only one edge
        child =  $n.Edges[label]$ 
    do
         $n.Prefix := n.Prefix \circ child.Prefix \parallel$ 
         $n.Value := child.Value \parallel$ 
         $n.Edges := child.Edges$  ;
    end with ;

    ExitMergeChild:
    return ;
end procedure ;

procedure deletePrefix( $n = \langle \rangle$ ,  $nRoot = \text{FALSE}$ )
variables searchLabel =  $\langle \rangle$  ;
begin
    DeletePrefix:
    Check for key exhaustion
    if  $Len(search) = 0$  then
        newChild := [
            Prefix  $\mapsto n.Prefix$ ,
            Value  $\mapsto \langle \rangle$ ,
            Edges  $\mapsto \langle \rangle$ 
        ] ;
        return ;
    end if ;

    FindEdge:
    Look for an edge
    searchLabel := search[1] ;
    if  $\neg searchLabel \in \text{DOMAIN } n.Edges$  then
        NoEdge:

```

```

    newChild :=  $\langle \rangle$ ;
    return;
end if;

ConsumeAndRecurse:
with child = n.Edges[searchLabel] do
    if  $\neg \text{HasPrefix}(\text{child.Prefix}, \text{search}) \wedge \neg \text{HasPrefix}(\text{search}, \text{child.Prefix})$  then
        newChild :=  $\langle \rangle$ ;
        return;
    else
        Consume the search prefix
        if  $\text{Len}(\text{child.Prefix}) > \text{Len}(\text{search})$  then
            search :=  $\langle \rangle$ ;
        else
            search :=  $\text{SubSeq}(\text{search}, \text{Len}(\text{child.Prefix}) + 1, \text{Len}(\text{search}))$ 
        end if;

        call deletePrefix(child, FALSE);
    end if;
end with;

ExitIfNoChild:
if NewChildNull then
    newChild :=  $\langle \rangle$ ;
    return;
end if;

ModifyNode:
if  $\text{Len}(\text{newChild.Value}) = 0 \wedge \text{Cardinality}(\text{DOMAIN } \text{newChild.Edges}) = 0$  then
    n.Edges :=  $[\text{label} \in \text{DOMAIN } n.Edges \setminus \{\text{searchLabel}\} \mapsto n.Edges[\text{label}]]$ ;

    if  $\neg nRoot \wedge \text{Cardinality}(\text{DOMAIN } n.Edges) = 1 \wedge \text{Len}(n.Value) = 0$  then
        call mergeChild(n);
    end if;
else
    n.Edges[searchLabel] := newChild
end if;

ReturnDeletePrefix:
newChild := n;
return;
end procedure;

```

This entire algorithm is almost 1:1 translated where possible from the actual implementation in *iter.go*. That's the point: we're trying to verify our algorithm is correct for all inputs.

Source: <https://github.com/hashicorp/go-immutable-radix/blob/f63f49c0b598a5ead21c5015fb4d08fe7e3c21ea/iter.go> \neq L16

```
begin
Begin:
  search := prefix ;
  call deletePrefix(root, TRUE) ;

SetNewRoot:
  if  $\neg$ NewChildNull then
    root := newChild ;
  end if ;

AssertExpected:
  assert Range(root) = Expected(input, prefix) ;
end algorithm ;
```

!!!NOTE !!! The rest of the file is auto-generated based on the *PlusCal* above. For those who are reading this to learn TLA+/*PlusCal*, you can stop reading here.

BEGIN TRANSLATION (*chksum*(*pcal*) = "9c23a03f" \wedge *chksum*(*tla*) = "cb0cd4a5")

Parameter *n* of procedure *mergeChild* at line 63 col 22 changed to *n_*

VARIABLES *input*, *prefix*, *root*, *newChild*, *search*, *result*, *pc*, *stack*

define statement

NewChildNull \triangleq DOMAIN *newChild* = {}

VARIABLES *n_*, *n*, *nRoot*, *searchLabel*

vars \triangleq \langle *input*, *prefix*, *root*, *newChild*, *search*, *result*, *pc*, *stack*, *n_*, *n*, *nRoot*, *searchLabel* \rangle

Init \triangleq **Global variables**
 \wedge *input* \in *InputSets*
 \wedge *prefix* \in *Inputs*
 \wedge *root* = *RadixTree*(*input*)
 \wedge *newChild* = \langle \rangle
 \wedge *search* = {}
 \wedge *result* = {}
Procedure *mergeChild*
 \wedge *n_* = \langle \rangle
Procedure *deletePrefix*
 \wedge *n* = \langle \rangle
 \wedge *nRoot* = FALSE
 \wedge *searchLabel* = \langle \rangle
 \wedge *stack* = \langle \rangle
 \wedge *pc* = "Begin"

$$\begin{aligned}
MergeChild &\triangleq \wedge pc = \text{"MergeChild"} \\
&\wedge \text{LET } label \triangleq \text{CHOOSE } x \in \text{DOMAIN } n_ . Edges : \text{TRUEIN} \\
&\quad \text{LET } child \triangleq n_ . Edges[label] \text{IN} \\
&\quad \quad n_ ' = [n_ \text{ EXCEPT } !.Prefix = n_ . Prefix \circ child.Prefix, \\
&\quad \quad \quad !.Value = child.Value, \\
&\quad \quad \quad !.Edges = child.Edges] \\
&\wedge pc' = \text{"ExitMergeChild"} \\
&\wedge \text{UNCHANGED } \langle input, prefix, root, newChild, search, result, \\
&\quad \quad \quad stack, n, nRoot, searchLabel \rangle \\
\\
ExitMergeChild &\triangleq \wedge pc = \text{"ExitMergeChild"} \\
&\wedge pc' = Head(stack).pc \\
&\wedge n_ ' = Head(stack).n_ \\
&\wedge stack' = Tail(stack) \\
&\wedge \text{UNCHANGED } \langle input, prefix, root, newChild, search, \\
&\quad \quad \quad result, n, nRoot, searchLabel \rangle \\
\\
mergeChild &\triangleq MergeChild \vee ExitMergeChild \\
\\
DeletePrefix &\triangleq \wedge pc = \text{"DeletePrefix"} \\
&\wedge \text{IF } Len(search) = 0 \\
&\quad \text{THEN } \wedge newChild' = \left[\begin{array}{l} Prefix \mapsto n.Prefix, \\ Value \mapsto \langle \rangle, \\ Edges \mapsto \langle \rangle \end{array} \right] \\
&\quad \quad \wedge pc' = Head(stack).pc \\
&\quad \quad \wedge searchLabel' = Head(stack).searchLabel \\
&\quad \quad \wedge n' = Head(stack).n \\
&\quad \quad \wedge nRoot' = Head(stack).nRoot \\
&\quad \quad \wedge stack' = Tail(stack) \\
&\quad \text{ELSE } \wedge pc' = \text{"FindEdge"} \\
&\quad \quad \wedge \text{UNCHANGED } \langle newChild, stack, n, nRoot, \\
&\quad \quad \quad searchLabel \rangle \\
&\wedge \text{UNCHANGED } \langle input, prefix, root, search, result, n_ \rangle \\
\\
FindEdge &\triangleq \wedge pc = \text{"FindEdge"} \\
&\wedge searchLabel' = search[1] \\
&\wedge \text{IF } \neg searchLabel' \in \text{DOMAIN } n.Edges \\
&\quad \text{THEN } \wedge pc' = \text{"NoEdge"} \\
&\quad \text{ELSE } \wedge pc' = \text{"ConsumeAndRecurse"} \\
&\wedge \text{UNCHANGED } \langle input, prefix, root, newChild, search, result, \\
&\quad \quad \quad stack, n_ , n, nRoot \rangle \\
\\
NoEdge &\triangleq \wedge pc = \text{"NoEdge"} \\
&\wedge newChild' = \langle \rangle
\end{aligned}$$

$$\begin{aligned}
& \wedge pc' = \text{Head}(\text{stack}).pc \\
& \wedge \text{searchLabel}' = \text{Head}(\text{stack}).\text{searchLabel} \\
& \wedge n' = \text{Head}(\text{stack}).n \\
& \wedge nRoot' = \text{Head}(\text{stack}).nRoot \\
& \wedge \text{stack}' = \text{Tail}(\text{stack}) \\
& \wedge \text{UNCHANGED } \langle \text{input}, \text{prefix}, \text{root}, \text{search}, \text{result}, n_- \rangle \\
\text{ConsumeAndRecurse} & \triangleq \wedge pc = \text{"ConsumeAndRecurse"} \\
& \wedge \text{LET } child \triangleq n.\text{Edges}[\text{searchLabel}] \text{ IN} \\
& \quad \text{IF } \neg \text{HasPrefix}(child.\text{Prefix}, \text{search}) \wedge \neg \text{HasPrefix}(\text{search}, child.\text{Prefix}) \\
& \quad \quad \text{THEN } \wedge \text{newChild}' = \langle \rangle \\
& \quad \quad \quad \wedge pc' = \text{Head}(\text{stack}).pc \\
& \quad \quad \quad \wedge \text{searchLabel}' = \text{Head}(\text{stack}).\text{searchLabel} \\
& \quad \quad \quad \wedge n' = \text{Head}(\text{stack}).n \\
& \quad \quad \quad \wedge nRoot' = \text{Head}(\text{stack}).nRoot \\
& \quad \quad \quad \wedge \text{stack}' = \text{Tail}(\text{stack}) \\
& \quad \quad \quad \wedge \text{UNCHANGED } \text{search} \\
& \quad \quad \text{ELSE } \wedge \text{IF } \text{Len}(child.\text{Prefix}) > \text{Len}(\text{search}) \\
& \quad \quad \quad \quad \text{THEN } \wedge \text{search}' = \langle \rangle \\
& \quad \quad \quad \quad \text{ELSE } \wedge \text{search}' = \text{SubSeq}(\text{search}, \text{Len}(child.\text{Prefix}) + 1, \text{Len}(\text{search})) \\
& \quad \quad \wedge n' = child \\
& \quad \quad \wedge nRoot' = \text{FALSE} \\
& \quad \quad \wedge \text{stack}' = \langle [\text{procedure} \mapsto \text{"deletePrefix"}, \\
& \quad \quad \quad pc \mapsto \text{"ExitIfNoChild"}, \\
& \quad \quad \quad \text{searchLabel} \mapsto \text{searchLabel}, \\
& \quad \quad \quad n \mapsto n, \\
& \quad \quad \quad nRoot \mapsto nRoot] \rangle \\
& \quad \quad \quad \quad \circ \text{stack} \\
& \quad \quad \wedge \text{searchLabel}' = \langle \rangle \\
& \quad \quad \wedge pc' = \text{"DeletePrefix"} \\
& \quad \quad \wedge \text{UNCHANGED } \text{newChild} \\
& \quad \wedge \text{UNCHANGED } \langle \text{input}, \text{prefix}, \text{root}, \text{result}, n_- \rangle \\
\text{ExitIfNoChild} & \triangleq \wedge pc = \text{"ExitIfNoChild"} \\
& \wedge \text{IF } \text{NewChildNull} \\
& \quad \text{THEN } \wedge \text{newChild}' = \langle \rangle \\
& \quad \quad \wedge pc' = \text{Head}(\text{stack}).pc \\
& \quad \quad \wedge \text{searchLabel}' = \text{Head}(\text{stack}).\text{searchLabel} \\
& \quad \quad \wedge n' = \text{Head}(\text{stack}).n \\
& \quad \quad \wedge nRoot' = \text{Head}(\text{stack}).nRoot \\
& \quad \quad \wedge \text{stack}' = \text{Tail}(\text{stack}) \\
& \quad \text{ELSE } \wedge pc' = \text{"ModifyNode"} \\
& \quad \quad \wedge \text{UNCHANGED } \langle \text{newChild}, \text{stack}, n, nRoot, \\
& \quad \quad \quad \text{searchLabel} \rangle \\
& \wedge \text{UNCHANGED } \langle \text{input}, \text{prefix}, \text{root}, \text{search}, \text{result}, n_- \rangle
\end{aligned}$$

$$\begin{aligned}
\text{ModifyNode} &\triangleq \wedge pc = \text{"ModifyNode"} \\
&\wedge \text{IF } \text{Len}(\text{newChild.Value}) = 0 \wedge \text{Cardinality}(\text{DOMAIN } \text{newChild.Edges}) = 0 \\
&\quad \text{THEN } \wedge n' = [n \text{ EXCEPT } !.Edges = [label \in \text{DOMAIN } n.Edges \setminus \{searchLabel\} \mapsto n.Edges \\
&\quad \quad \wedge \text{IF } \neg nRoot \wedge \text{Cardinality}(\text{DOMAIN } n'.Edges) = 1 \wedge \text{Len}(n'.Value) = 0 \\
&\quad \quad \quad \text{THEN } \wedge \wedge n_-' = n' \\
&\quad \quad \quad \quad \wedge stack' = \langle [procedure \mapsto \text{"mergeChild"}, \\
&\quad \quad \quad \quad \quad pc \mapsto \text{"ReturnDeletePrefix"}, \\
&\quad \quad \quad \quad \quad n_- \mapsto n_-] \rangle \\
&\quad \quad \quad \quad \quad \circ stack \\
&\quad \quad \quad \quad \wedge pc' = \text{"MergeChild"} \\
&\quad \quad \quad \text{ELSE } \wedge pc' = \text{"ReturnDeletePrefix"} \\
&\quad \quad \quad \quad \wedge \text{UNCHANGED } \langle stack, n_- \rangle \\
&\quad \text{ELSE } \wedge n' = [n \text{ EXCEPT } !.Edges[searchLabel] = \text{newChild}] \\
&\quad \quad \wedge pc' = \text{"ReturnDeletePrefix"} \\
&\quad \quad \quad \wedge \text{UNCHANGED } \langle stack, n_- \rangle \\
&\quad \wedge \text{UNCHANGED } \langle input, prefix, root, \text{newChild}, search, result, \\
&\quad \quad nRoot, searchLabel \rangle \\
\\
\text{ReturnDeletePrefix} &\triangleq \wedge pc = \text{"ReturnDeletePrefix"} \\
&\wedge \text{newChild}' = n \\
&\wedge pc' = \text{Head}(stack).pc \\
&\wedge searchLabel' = \text{Head}(stack).searchLabel \\
&\wedge n' = \text{Head}(stack).n \\
&\wedge nRoot' = \text{Head}(stack).nRoot \\
&\wedge stack' = \text{Tail}(stack) \\
&\wedge \text{UNCHANGED } \langle input, prefix, root, search, result, n_- \rangle \\
\\
\text{deletePrefix} &\triangleq \text{DeletePrefix} \vee \text{FindEdge} \vee \text{NoEdge} \vee \text{ConsumeAndRecurse} \\
&\vee \text{ExitIfNoChild} \vee \text{ModifyNode} \vee \text{ReturnDeletePrefix} \\
\\
\text{Begin} &\triangleq \wedge pc = \text{"Begin"} \\
&\wedge search' = prefix \\
&\wedge \wedge n' = root \\
&\quad \wedge nRoot' = \text{TRUE} \\
&\quad \wedge stack' = \langle [procedure \mapsto \text{"deletePrefix"}, \\
&\quad \quad pc \mapsto \text{"SetNewRoot"}, \\
&\quad \quad searchLabel \mapsto searchLabel, \\
&\quad \quad n \mapsto n, \\
&\quad \quad nRoot \mapsto nRoot] \rangle \\
&\quad \quad \circ stack \\
&\quad \wedge searchLabel' = \langle \rangle \\
&\quad \wedge pc' = \text{"DeletePrefix"} \\
&\quad \wedge \text{UNCHANGED } \langle input, prefix, root, \text{newChild}, result, n_- \rangle \\
\\
\text{SetNewRoot} &\triangleq \wedge pc = \text{"SetNewRoot"} \\
&\wedge \text{IF } \neg \text{NewChildNull}
\end{aligned}$$

```

      THEN  $\wedge \text{root}' = \text{newChild}$ 
      ELSE  $\wedge \text{TRUE}$ 
            $\wedge \text{root}' = \text{root}$ 
 $\wedge \text{pc}' = \text{"AssertExpected"}$ 
 $\wedge \text{UNCHANGED } \langle \text{input}, \text{prefix}, \text{newChild}, \text{search}, \text{result}, \text{stack},$ 
                 $\text{n-}, \text{n}, \text{nRoot}, \text{searchLabel} \rangle$ 

 $\text{AssertExpected} \triangleq \wedge \text{pc} = \text{"AssertExpected"}$ 
 $\wedge \text{Assert}(\text{Range}(\text{root}) = \text{Expected}(\text{input}, \text{prefix}),$ 
             $\text{"Failure of assertion at line 157, column 3."})$ 
 $\wedge \text{pc}' = \text{"Done"}$ 
 $\wedge \text{UNCHANGED } \langle \text{input}, \text{prefix}, \text{root}, \text{newChild}, \text{search},$ 
                 $\text{result}, \text{stack}, \text{n-}, \text{n}, \text{nRoot}, \text{searchLabel} \rangle$ 

 $\text{Allow infinite stuttering to prevent deadlock on termination.}$ 
 $\text{Terminating} \triangleq \text{pc} = \text{"Done"} \wedge \text{UNCHANGED vars}$ 

 $\text{Next} \triangleq \text{mergeChild} \vee \text{deletePrefix} \vee \text{Begin} \vee \text{SetNewRoot} \vee \text{AssertExpected}$ 
 $\vee \text{Terminating}$ 

 $\text{Spec} \triangleq \text{Init} \wedge \square[\text{Next}]_{\text{vars}}$ 

 $\text{Termination} \triangleq \Diamond(\text{pc} = \text{"Done"})$ 

 $\text{END TRANSLATION}$ 

```

```

\ * Modification History
\ * Last modified Fri Jul 02 11:27:05 PDT 2021 by mitchellh
\ * Created Wed Jun 30 10:05:52 PDT 2021 by mitchellh

```