Depth-First Search

Toshiaki Nishihara  Yasuhiko Minamide

April 19, 2020

Abstract

Depth-first search of a graph is formalized with function. It is shown that it visits all of the reachable nodes from a given list of nodes. Executable ML code of depth-first search is obtained with code generation feature of Isabelle/HOL. The formalization contains two implementations of depth-first search: one by stack and one by nested recursion. They are shown to be equivalent. The termination condition of the version with nested-recursion is shown by the method of inductive invariants.

Contents

1 Depth-First Search
   1.1 Definition of Graphs ........................................... 1
   1.2 Depth-First Search with Stack ................................. 2
   1.3 Depth-First Search with Nested-Recursion ................. 2
   1.4 Basic Properties ............................................... 3
   1.5 Correctness .................................................... 4
   1.6 Executable Code ............................................... 4

1 Depth-First Search

theory DFS
imports Main
begin

1.1 Definition of Graphs

typedecl node
type-synonym graph = (node * node) list

primrec nexts :: [graph, node] ⇒ node list
where
nexts [] n = []
| nexts (e#es) n = (if fst e = n then snd e # nexts es n else nexts es n)
definition nextss :: [graph, node list] ⇒ node set
  where nextss g xs = set g "" set xs

lemma nexts-set: y ∈ set (nexts g x) = ((x,y) ∈ set g)
  ⟨proof⟩

lemma nextss-Cons: nextss g (x#xs) = set (nexts g x) ∪ nextss g xs
  ⟨proof⟩

definition reachable :: [graph, node list] ⇒ node set
  where reachable g xs = (set g)* "" set xs

1.2 Depth-First Search with Stack

definition nodes-of :: graph ⇒ node set
  where nodes-of g = set (map fst g @ map snd g)

lemma [simp]: x /∈ nodes-of g ⇒ nexts g x = []
  ⟨proof⟩

lemma [simp]: finite (nodes-of g − set ys)
  ⟨proof⟩

function
dfs :: graph ⇒ node list ⇒ node list ⇒ node list
  where
dfs-base: dfs g [] ys = ys
  | dfs-inductive: dfs g (x#xs) ys = (if List.member ys x then dfs g xs ys
  else dfs g (nexts g x@xs) (x#ys))
  ⟨proof⟩

termination
  ⟨proof⟩

  • The second argument of dfs is a stack of nodes that will be visited.
  • The third argument of dfs is a list of nodes that have been visited already.

1.3 Depth-First Search with Nested-Recursion

function
dfs2 :: graph ⇒ node list ⇒ node list ⇒ node list
  where
dfs2 g [] ys = ys
  | dfs2-inductive:
    dfs2 g (x#xs) ys = (if List.member ys x then dfs2 g xs ys
else dfs2 g xs (dfs2 g (nexts g x) (x#ys))

(*proof*)

**lemma dfs2-invariant:** dfs2-dom \( (g, xs, ys) \implies set ys \subseteq set (dfs2 g xs ys) \)
(*proof*)

**termination dfs2**
(*proof*)

**lemma dfs-app:** dfs g (xs@ys) zs = dfs g ys (dfs g xs zs)
(*proof*)

**lemma dfs2 g xs ys = dfs g xs ys**
(*proof*)

### 1.4 Basic Properties

**lemma visit-subset-dfs:** set ys \subseteq set (dfs g xs ys)
(*proof*)

**lemma next-subset-dfs:** set xs \subseteq set (dfs g xs ys)
(*proof*)

**lemma nextss-closed-dfs**
(*rule-format*):

nextss g ys \subseteq set xs \cup set ys \implies nextss g (dfs g xs ys) \subseteq set (dfs g xs ys)
(*proof*)

**lemma nextss-closed-dfs:** nextss g (dfs g xs []) \subseteq set (dfs g xs [])
(*proof*)

**lemma Image-closed-trancl:** assumes \( r^\circ X \subseteq X \) shows \( r^* \circ X = X \)
(*proof*)

**lemma reachable-closed-dfs:** reachable g xs \subseteq set (dfs g xs [])
(*proof*)

**lemma reachable-nexts:** reachable g (nexts g x) \subseteq reachable g [x]
(*proof*)

**lemma reachable-append:** reachable g (xs @ ys) = reachable g xs \cup reachable g ys
(*proof*)

**lemma dfs-subset-reachable-visit-nodes:** set (dfs g xs ys) \subseteq reachable g xs \cup set ys
(*proof*)
1.5 Correctness

\textbf{theorem} \textit{dfs-eq-reachable}: \textit{set }\textit{(dfs g xs [] )} = \textit{reachable g xs} \\
\hspace{1cm} \langle \textit{proof}\rangle

\textbf{theorem} \textit{y} \in \textit{set }\textit{(dfs g [x] [] )} = ((x,y) \in (\textit{set g})^*) \\
\hspace{1cm} \langle \textit{proof}\rangle

1.6 Executable Code

\textbf{consts} \textit{Node} :: \textit{int }\Rightarrow \textit{node}

\textbf{code-datatype} \textit{Node}

\textbf{instantiation} \textit{node} :: \textit{equal} \\
\hspace{1cm} \textbf{begin}

\textbf{definition} \textit{equal-node} :: \textit{node }\Rightarrow \textit{node }\Rightarrow \textit{bool} \\
\hspace{1cm} \textbf{where}

\hspace{2cm} [\textit{code del}]: \textit{equal-node} = \textit{HOL.eq}

\textbf{instance} \langle \textit{proof}\rangle

\hspace{1cm} \textbf{end}

\textbf{declare} [[\textit{code abort: HOL.equal} :: \textit{node }\Rightarrow \textit{node }\Rightarrow \textit{bool}]]

\textbf{export-code} \textit{dfs dfs2} \textbf{in} \textit{SML file} \langle \textit{dfs.ML}\rangle

\hspace{1cm} \textbf{end}