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
  1.1 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 (nexts g x@xs) (x#ys))

else dfs2 g xs (dfs2 g (nexts g x) (x#ys))

(\langle proof \rangle)

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

termination dfs2
(\langle proof \rangle)

lemma dfs-app: dfs g (xs@ys) zs = dfs g ys (dfs g xs zs)
(\langle proof \rangle)

lemma dfs2 g xs ys = dfs g xs ys
(\langle proof \rangle)

1.4 Basic Properties

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

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

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)
(\langle proof \rangle)

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

lemma Image-closed-trancl: assumes r " X \subseteq X shows r* " X = X
(\langle proof \rangle)

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

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

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

lemma dfs-subset-reachable-visit-nodes: set (dfs g xs ys) \subseteq reachable g xs \cup set ys
(\langle proof \rangle)

3
1.5 Correctness

**Theorem** \( \text{dfs-eq-reachable}: \text{set} (\text{dfs } g \ xs \ []) = \text{reachable } g \ xs \)

\[ \langle \text{proof} \rangle \]

**Theorem** \( y \in \text{set} (\text{dfs } g \ [x] \ []) = ((x,y) \in (\text{set } g)^*) \)

\[ \langle \text{proof} \rangle \]

1.6 Executable Code

**consts** \( \text{Node} :: \text{int} \Rightarrow \text{node} \)

**code-datatype** \( \text{Node} \)

**instantiation** \( \text{node} :: \text{equal} \)

**begin**

**definition** \( \text{equal-node} :: \text{node} \Rightarrow \text{node} \Rightarrow \text{bool} \)

**where**

\[ \langle \text{code del} \rangle: \text{equal-node} = \text{HOL.eq} \]

**instance** \( \langle \text{proof} \rangle \)

**end**

**declare** \( \langle \text{code abort: HOL.equal} :: \text{node} \Rightarrow \text{node} \Rightarrow \text{bool} \rangle \)

**export-code** \( \text{dfs dfs2 in SML file dfs.ML} \)

**end**