Depth-First Search

Toshiaki Nishihara Yasuhiko Minamide

August 16, 2018

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** \( \text{nextss} :: \text{[graph, node list]} \Rightarrow \text{node set} \)

**where** \( \text{nextss} g \; \text{xs} = \text{set} g \; \cup \; \text{set} \; \text{xs} \)

**lemma** \( \text{nexts-set}: y \in \text{set} \; (\text{nexts} \; g \; x) = ((x,y) \in g) \)

**proof**

**lemma** \( \text{nextss-Cons}: \text{nextss} \; g \; (x \# \text{xs}) = \text{set} \; (\text{nexts} \; g \; x) \cup \text{nextss} \; g \; \text{xs} \)

**proof**

**definition** \( \text{reachable} :: \text{[graph, node list]} \Rightarrow \text{node set} \)

**where** \( \text{reachable} \; g \; \text{xs} = (\text{set} \; g) \ast \; \text{set} \; \text{xs} \)

### 1.2 Depth-First Search with Stack

**definition** \( \text{nodes-of} :: \text{graph} \Rightarrow \text{node set} \)

**where** \( \text{nodes-of} \; g = \text{set} \; (\text{map} \; \text{fst} \; g \; @ \; \text{map} \; \text{snd} \; g) \)

**lemma** \( \text{[simp]}: x /\in \; \text{nodes-of} \; g \Rightarrow \text{nexts} \; g \; x = [] \)

**proof**

**lemma** \( \text{[simp]}: \text{finite} \; (\text{nodes-of} \; g \; - \; \text{set} \; \text{ys}) \)

**proof**

**function**

\( \text{dfs} :: \text{graph} \Rightarrow \text{node list} \Rightarrow \text{node list} \Rightarrow \text{node list} \)

**where**

\( \text{dfs-base}: \text{dfs} \; g \; [] \; \text{ys} = \text{ys} \)

\| \text{dfs-inductive}: \text{dfs} \; g \; (x \# \text{xs}) \; \text{ys} = (\text{if} \; \text{List.member} \; \text{ys} \; x \; \text{then} \; \text{dfs} \; g \; \text{xs} \; \text{ys} \; \text{else} \; \text{dfs} \; g \; (\text{nexts} \; g \; x \# \text{xs}) \; (x \# \text{ys})) \)

**proof**

**termination**

**proof**

- The second argument of \( \text{dfs} \) is a stack of nodes that will be visited.
- The third argument of \( \text{dfs} \) is a list of nodes that have been visited already.

### 1.3 Depth-First Search with Nested-Recursion

**function**

\( \text{dfs2} :: \text{graph} \Rightarrow \text{node list} \Rightarrow \text{node list} \Rightarrow \text{node list} \)

**where**

\( \text{dfs2-base}: \text{dfs2} \; g \; [] \; \text{ys} = \text{ys} \)

\| \text{dfs2-inductive}: \text{dfs2} \; g \; (x \# \text{xs}) \; \text{ys} = (\text{if} \; \text{List.member} \; \text{ys} \; x \; \text{then} \; \text{dfs2} \; g \; \text{xs} \; \text{ys} \; \text{else} \; \text{dfs2} \; g \; (\text{nexts} \; g \; x \# \text{xs}) \; (x \# \text{ys})) \)
else dfs2 g xs (dfs2 g (nexts g x) (x#ys))

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

termination dfs2

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

lemma dfs2 g xs ys = dfs g xs ys

1.4 Basic Properties

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

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

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)

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

lemma Image-closed-trancl: assumes r "" X \subseteq X shows r* "" X = X

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

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

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

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

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

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

1.6 Executable Code

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

\textbf{code-datatype} Node

\textbf{instantiation} node :: equal
\begin{code}
\textbf{definition} equal-node :: node \Rightarrow node \Rightarrow bool
\textbf{where}
\texttt{[code del]}: equal-node = HOL.eq
\textbf{instance} \langle \textit{proof} \rangle
\end{code}
\textbf{end}

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

\textbf{export-code} dfs dfs2 in \textbf{SML file} dfs.ML
\textbf{end}