Comparing Graph Representations for Walks

Christoph Madlener

July 5, 2020

Contents

1 Findings on Walks

theory *Findings-Walks*
imports
AGF.DDFS
AGF.Berge
AGF.Graph
AGF.TA-Graphs
Graph-Theory.Digraph
Graph-Theory.Arc-Walk
Graph-Theory.Vertex-Walk
Ports-Overview
begin

This theory summarizes the findings from porting and comparing lemmas regarding walks (and specializations thereof) from different graph representations. First a short overview of the considered representations is given. Then similarities and differences in definitions and available lemmas will be emphasized. One focal point of this analysis was the proof automation obtainable in different representations. We conclude that – solely regarding walks – the underlying graph representation does not substantially impact

this aspect. Instead carefully considering rather minuscule differences in definitions and formulations of lemmas can enable automation. We will refer to relevant lemmas (and their proofs) to illustrate this in more detail. For the full overview of ported lemmas, see *Ports.Ports-Overview*.

1.1 Considered Graph Representations

The target of the ports was *AGF.DDFS*, which represents a graph simply as a set of pairs $(('a \times 'a)\ set)$, vertices are defined implicitly as *dVs*. This representation will be referred to as DDFS from now on. The original formalization defines walks inductively via *dpath*, [] is also a walk. Additionally there is *dpath-bet*, for specifying a start and end vertex, which enforces non-emptiness. Originally this was the smallest representation only containing some basic lemmas about paths and how a directed graph can model a transitive relation using walks (?):

thm *dpath-transitive-rel dpath-transitive-rel'*

A selection of lemmas regarding walks (and path, trails, etc.) has been ported from two undirected and two directed graph representations.

1.1.1 Undirected Graph Representations

AGF.Berge A graph is a set of (two-element) sets $('a\ set\ set)$, implicit vertices *Vs – graph-abs*. Referred to as Berge from now on.

Defines vertex walks via the inductive predicate *Berge.path* which allows empty walks. Also provides *walk-betw* for specifying start and end vertices. Note that *walk-betw* doesn't allow empty walks. *edges-of-path* is used to extract the edges of a given walk.

Besides some basic lemmas about walks, the formalization focuses on *edges-of-path* and goes on to develop theory about connected components and matchings. No notion of reachability is introduced.

AGF.Graph Like Berge, but with an explicit vertex set $('a\ graph)$ – *Graph.graph*. Referred to as Mitja from now on.

Defines *walk* and *walk-edges* which essentially correspond to *walk-betw* and *edges-of-path* from Berge. Also defines *closed-walk*s, *Walk.trail*s, *closed-trail* and *Walk.path*s. Shows the number of paths in a finite graphs is finite.

thm *finite-graph.paths-finite*

In addition to basic walk lemmas includes decomposition of walks at a given vertex and obtaining a path from a walk.

thm *graph.walk-vertex-decomp-is-walk-vertex-decomp*

thm *graph.walk-to-path-is-path*

Reachability is defined via the existence of a walk (*Walk.reachable*).

Furthermore defines weighted graphs and on top of that shortest paths, trees etc.

1.1.2 Directed Graph Representations

AGF.TA-Graphs A graph is defined via its adjacency relation ($'a \Rightarrow 'a \Rightarrow$ *bool*), implicit *Graph-Defs.vertices*. Referred to as TA-Graph from now on.

Walks are defined via the inductive *Graph-Defs.steps*, as nonempty lists of vertices. *Graph-Defs.run* formalizes infinite walks.

Reachability is defined as the reflexive transitive closure of the adjacency relation (*rtranclp*), resp. its transitive closure (*tranclp*).

Formalizes simulations and bisumiluations.

Graph-Theory.Digraph A graph is represented as a set of vertices, a set of edges and functions mapping edges to their head, resp. tail (($'a$, $'b$) *pre-digraph*)– *wf-digraph*. Referred to as Digraph from now on.

Formalizes both vertex (*vwalk*) and arc walks (*pre-digraph.awalk*). The vertex walk part introduces "joinability" of walks. Provides *vwalk-arcs* to compute the arcs belonging to a walk. Note, however, that this gives a list of pairs of vertices, which generally are not the arcs of the graph (which have an arbitrary type).

For arc walks also has *pre-digraph.trail* and *pre-digraph.apath*. Shows how to obtain a path from a walk.

thm *wf-digraph.apath-awalk-to-apath*

Reachability (*Digraph.reachable*) is defined via a custom reflexive transitive closure (*rtrancl-on*) of *arcs-ends* as to include potentially disconnected vertices.

Has lots of additional theory, including Euler trails, shortest paths, a simpler version representing the arcs as a set of pairs, and more.

1.2 Walk representation and basic reasoning

All of the above formalizations define walks as lists of either vertices or arcs. Hence, reasoning about walks is essentially reasoning about lists with some additional conditions imposed by the edges present in a given graph. After establishing the interactions between elementary list operations and walks, the reasoning is pretty much independent from the underlying graph representation. However, Digraph which arguably has the most abstract

approach, has to help itself by employing more helper definitions, in order to formulate lemmas in a manageable way (cf. *arcs-ends*, $\lambda G\ u\ p.$ *hd* (*pre-digraph.awalk-verts* $G\ u\ p$), etc.). All the other theories chose a more lightweight representation, allowing to work directly on the underlying graph.

An illustrative example for this are the proofs of the following theorems (and the lemmas leading up to them), where the automation has been set up identically:

thm *wf-digraph.apath-awalk-to-apath*
thm *apath-awalk-to-apath*

1.3 Arc walks vs. vertex walks

As mentioned before, walks can be defined as either lists of vertices or as lists of arcs. The first variant is present in all candidates. Arc walks, however, are only formalized in Digraph (*Graph-Theory.Arc-Walk*). Strictly concerning the reasoning, the two variants again do not differ in a meaningful way. *Graph-Theory.Vertex-Walk* mentions that vertex walks "do not really work with multigraphs". As it stands only Digraph is capable of representing multigraphs. Obtaining a vertex walk from an arc walk is straightforward.

thm *wf-digraph.awalk-imp-vwalk*

The converse direction is nontrivial for Digraph because one cannot recover the arcs used by a vertex walk. For the other (non multigraph) representations both directions should be obtainable. See the following to lemmas for reference:

thm *awalk-imp-dpath*
thm *dpath-imp-awalk*

1.4 Empty vs. nonempty walks

As mentioned before, the formalizations differ in whether they allow empty walks (i.e. []) or not. This has some implications on the automation when a proof is conducted with induction on the list representing a walk. Consider the proofs of the following lemmas, the first one from DDFS which allows empty walks, the second one from TA-Graph, which does not allow empty walks.

thm *append-dpath-pref*
thm *Graph-Defs.steps-appendD1*

In TA-Graph an additional case distinction is required in the induction step in order to complete the proof. This inconvenience could possibly be fixed with a custom induction scheme. Thus this aspect also does not lead to a favorable definition of walks, instead it remains a consideration to be made.

1.5 Implicit vs. explicit vertex sets

Another difference between the different formalizations is whether they define an implicit or explicit vertex set. An implicit vertex set does not allow for disconnected vertices. During the ports of lemmas about walks, this only came into play in one case. The following lemma from Mitja about walks in a (vertex-)induced subgraph took some additional care when formulating, in order to accommodate the implicit vertex set of DDFS.

thm *induced-subgraph.walk-supergraph-is-walk-subgraph*
thm *dpath-induced-subgraph-dpath*

1.6 Comparison of available Theory

The introduction of the different representations already briefly introduced what areas each of them covers. Most of them specialize at some point into different graph topics besides walks. Strictly speaking about walks, the most "complete" formalization is probably Digraph. One big reason for this of course being that it was the only representation under consideration which formalized arc-walks. It also has a very solid foundation for extending the existing walk lemmas (if that is even necessary).

As DDFS was the target of the ports it now also has arc-walks. At the same time it needs more work to clean up, unify the concepts introduced by different representations, and also some fleshing out of basic lemmas and intermediate results. While working directly with a set of edges works well for prototyping and some quick developments, it would probably be also helpful to work with locales in further development.

1.7 Conclusion

In the previous subsections we summarized the findings of porting lemmas from different graph representations to DDFS. The goal was to evaluate whether one formalization offers improved proof automation for lemmas about walks. We conclude that no meaningful advantages can be observed in this regard. At the same time it does not seem necessary to develop and maintain multiple graph representations.

Subsequently we investigated the differences in the definitions of walks and vertices. Although in this case the effects were observable in some cases, the miniscule reduction of effort does not merit departing from a wanted definition.

end