Description of the *Loco* Library

Pierre Castéran & Vincent Filou  
Laboratoire Bordelais de Recherche en Informatique, LabRI, UMR 5800, Université Bordeaux 1,  
351 Cours de la Libération, 33405 Talence Cedex, France  
{pierre.casteran, vincent.filou}@labri.fr  
A3PAT ANR-05-BLAN-0146 and RIMEL ANR-06-SETIN-015  

October 12, 2009

1 Introduction

*Loco* is a set of libraries for Coq 8.2 (http://coq.inria.fr) about local computation systems, available at http://www.labri.fr/~casteran/Loco. Theoretical descriptions of local computation systems can be found at the Visidia project page: http://www.labri.fr/projet/visidia. We mainly present the contents of the *Examples/* sublibrary. The contents of the other modules is briefly sketched, since many details are open to change and improvement.

To compile the library, just type `make` at its top-level.

2 Overview of the library’s structure

The library contains currently four subdirectories:

**Prelude**: This directory contains some additional definitions and facts about well-ordered relations, finite sets and maps.

**Graphs**: This directory contains an *ad-hoc* library on finite, non-directed labelled graphs. It is far from being complete: we put only the definitions and facts which we needed for our work on local computation systems. A special module *Graph_Admitted* contains some folklore lemmas whose proof has a priority lower than the rest of our development.

**GRS**: This directory is the adaptation to Coq of the theory of local computations. It is clearly a work in progress.

**Examples**: This directory contains some formal correction proofs, as well as two impossibility proofs. Its contents is described in the next sections.
3 Specifications

Formal specifications are described under the form of tasks, i.e. input/output relations between labeled graphs.

3.1 Computing the degree of each vertex

The module Examples/Degree_Task defines a task for computing the degree of each vertex of any finite, undirected connected graph. The input is any unlabeled graph $G$ (i.e. labeled with Coq’s unit singleton type). The output is the same underlying graph, in which each vertex is labeled with its degree in $G$. Edges are not labeled in the output states.

3.2 Election task

The output state of an election is a graph labeling built on a type with only two values: elected and beaten. An election task must terminate on a state in which exactly one vertex is labeled with elected. This kind of task is defined in the module Examples/Election_Task.

The task Election_Trees_Task is associated to election in trees in which each vertex is initially labeled with its degree.

3.3 Spanning Tree Construction

The module Examples/SpanningTree_Task defines a task for computing a spanning tree of any finite, undirected connected graph. The input is any labeled graph $G$ where vertices are initially labeled with booleans, with exactly one vertex labeled true, and edges are unlabeled. The output state must be the graph $G$ in which vertices and edges are labeled with booleans, and the vertices and edges carrying the value true determine a spanning tree of $G$.

4 Certified Local Computation Systems

4.1 Computing the degree of each vertex (LC0, Implicit Termination Detection)

The module Examples/Degree_Implicit_Term contains a LC0 relabeling system for computing the degree of each vertex of the graph $G$ (see [3.1]).

Each vertex is labeled with the type option nat and each edge with a boolean value. The relabeling rule is shown in figure [1]: A None-labeled vertex updates its label so that it becomes Some $n$ where $n$ is its degree in the graph.
4.2 Computing the degree of each vertex (LC1, Local Termination Detection)

The module Examples/Degree_LC1 contains a LC1 relabeling system for computing the degree of each vertex of the graph $G$ (see 3.1), with local termination detection.

The relabeling rule is shown in figure 2.

4.3 Spanning Tree Computation (LC0, Local Termination Detection)

The module Examples/SpanningTree_LC0 contains a LC0 relabeling system for building a spanning tree of any connected, finite, simple undirected graph, with exactly one initially marked (i.e. true-labeled) vertex (see section 3.3), with local termination detection. The relabeling rule is shown in figure 3, an example of execution is presented in figure 4.

$$R : \begin{array}{ccc}
A & f & A \\
\bullet & \bullet & \bullet
\end{array} \rightarrow \begin{array}{ccc}
A & 1 & A \\
\bullet & \bullet & \bullet
\end{array}$$

Figure 3: Spanning Tree Computation: a LC0 Relabeling Rule
4.4 Vertex Election in a Tree (LC0, Local Termination Detection)

The module Examples/Election_Local_Term contains a LC0 relabeling system for electing a vertex in a tree, each vertex of which is initially labeled with its degree (see section 3.2).

Vertices are labeled in option nat and edges in unit. The relabeling rule is represented in figure 5.

5 Impossibility Proofs

5.1 Computing the degree of each vertex (LC0, Local Termination Detection)

The module Examples/No_Degree_LC0 contains a formal proof that there exists no LC0 relabeling system with local termination detection for computing the degree of each vertex in an initially unlabeled graph.

The proof is done by contradiction, using lemma extension_lemma from module GRS/LC0. Roughly speaking, any behaviour of a LC0-system on a labeled graph G can be extended on a supergraph G'. Such a system would then give the same degree to the vertex a in G and G' (see figure 6).
Figure 6: The degree of $a$ in $G$ is not equal to the degree of $a$ in $G'$

5.2 Election in an unlabeled tree (LC0, Local Termination Detection)

The module `Examples/No_Election_LC0` contains a formal proof that there exists no LC0 relabeling system with local termination detection for vertex election in an initially unlabeled tree.

The proof is done by contradiction. Such a system would elect at least two vertices of the graph $G$ of figure 7. The proof uses a lemma allowing to compose two executions of a LC0 system on two disjoint labeled graphs (lemma `composition_lemma` of module `GRS/LC0`).
Figure 7: Any LC0, LTD system for election in uniformly labeled trees would elect two vertices of $G$. 