Visualization and Manipulation of Pedigree Diagrams

Limsoon Wong
limsoon@krdl.org.sg
Kent Ridge Digital Labs, 21 Heng Mui Keng Terrace, Singapore 119613

Abstract

The Pedigree Visualizer is a system for visualization of pedigree diagrams. It accepts a simple text-based specification of a pedigree diagram. The pedigree diagram is then layout automatically. Both GIF- and PS-formatted output files are produced. In addition, the Pedigree Visualizer also provides a rich set of functions for the manipulation and management of large pedigree files.

Keywords: pedigree diagram, graph layout

1 Introduction

A pedigree is a set of individuals related as spouse, sibling, and offspring. There are some aesthetic considerations when drawing pedigree diagrams. Overlap should be avoided between individuals. Spouses should be adjacent to each other. Siblings should also be adjacent to each other. Parents should be directly above their children. Link crossings and sharp bends should be avoided. Links should be keep as short as possible. Symmetry and balance should be favoured. And so on.

According to a recent paper of Tores and Barillot [7], it is difficult to find softwares that draw pedigree diagrams perfectly. We are intrigued by their remarks and decide to explore the problem of building a system for the automatic layout of pedigree diagrams. We have at our command three general and powerful packages developed for previous applications: the Kleisli Query System [2, 3] developed for writing ad-hoc queries over heterogeneous complex data sources that we can use for manipulation of large pedigree databases; the Graphviz package [4, 5] developed for automatic layout and drawing of directed graphs that we can adapt for drawing pedigree diagrams; and the Pizzkell/Kleisli suite developed for interfacing various programming languages to Kleisli and its data exchange format that we can use to interface Kleisli to Graphviz. We feel that by customizing these three general tools a good pedigree visualization and management system can be constructed.

The Pedigree Visualizer is the result of our effort. It is a system that performs automatic layout of pedigree according to the criteria set forth above. There are some challenges that it has to overcome: how to let a pedigree be described textually, how to layout the pedigree neatly, how to allow a pedigree to be manipulated and transformed in a high-level manner, and how to detect inconsistencies in the pedigree. By properly utilizing the three general tools mentioned above, we are able to overcome these challenges with surprising ease.

This paper presents the functionalities of the Pedigree Visualizer and describes its implementation. The paper is organized as follows. Section 2 illustrates, by means of a demonstration, the automatic layout and drawing of pedigree diagrams using the Pedigree Visualizer. Section 3 highlights certain problematic pedigrees that are difficult to draw neatly, such as pedigrees involving individuals with multiple mates or with consanguineous mates. It also discusses the solutions to these problems in the Pedigree Visualizer. In the course of the discussion, the more powerful operators in the Pedigree Visualizer are described. Section 4 briefly introduces the simpler operators in the Pedigree Visualizer that are used for editing a pedigree diagram. Section 5 presents the architecture and implementation of the Pedigree Visualizer. It introduces the Kleisli Exchange Format. It gives example Kleisli/CPL codes that implement two of the functions in the Pedigree Visualizer. It also describes the layout algorithm. Section 6 contains remarks on our experience in implementing the Pedigree Visualizer.

2 Demonstration

The basic operation of the Pedigree Visualizer is very simple. The user first input the specification of his pedigree diagram, as shown in Figure 1. The input provided by the user in Figure 1 is required to be in the Kleisli Exchange Format, which will be described in Section 5. In addition, the Pedigree Visualizer also allows the user to upload entire pedigree data file that is too large for cut-and-paste into the interface shown in Figure 1.

The Pedigree Visualizer then computes an automatic layout of the diagram. It then produces both a GIF version and a Postscript version of the diagram. The GIF version corresponding to the example input in Figure 1 is shown in Figure 2. As can be seen, the output follows the established conventions for pedigree diagram. Male individuals are placed inside rectangles. Females individuals are placed inside ovals. Individuals of unspecified sex are placed inside diamonds. Afflicted individuals are shaded. Mates are placed side-by-side and immediately above their offsprings. Siblings are also placed side-by-side. Individuals of the same generation are also placed at the same horizontal level. There are two further items to note on this figure. Near the top of the diagram displayed is a button called "source", a button called "operations" and a button called "clean-up". The "source" button is for exporting an edited or transformed pedigree diagram in Kleisli Exchange Format. The other two buttons provide the user access to a variety of operators for manipulating the pedigree diagram that we will describe in Section 4 and Section 3. Near the bottom of the diagram is a line that says "No isolated individuals". An "isolated" individual is an individual that is not connected to any other individual in the diagram. The Pedigree Visualizer automatically checks for such individuals in the pedigree diagram that it draws and lists these individuals explicitly at the bottom of the diagram.

While we do not show the Postscript version that is produced, it has an interesting feature worth mentioning here. Since a pedigree diagram can be quite large, it can be difficult to view if we shrink it to fit a single A4-sized page. Therefore, the Pedigree Visualizer produces the Postscript version of the diagram in a special way. It automatically "tiles" the diagram into several pages. These pages, after printing, can then be easily pasted side-by-side to form the actual pedigree diagram.

3 Problem Cases

The standard convention for drawing pedigree diagrams seem to be very "Christian": the convention produces very nice pedigree diagrams provided (a) each person has only 1 mate, (b) kins do not mate, and (c) mates are treated as orphan (i.e., we are not interested in the ancestry of the wife if we are drawing the pedigree of the husband.) If any of these conditions are violated, the pedigree diagram would not look nice.

Unfortunately, in real life, these conditions are frequently violated. The most obvious violations are multiple matings and consanguineous matings. Take Figure 2 for example. If "6" mates with "13," then we have a mating that is both consanguineous ("13" is a niece of "6") and multiple ("7" is another mate of "6"). The result is a pedigree diagram with many cross-overs and loops, as shown in Figure 3. The reason is that the mating causes "13" to be placed at the same generation level as "6", since mates must be placed side-by-side according to convention. That in turn causes "11" and "12" to move up one level, since they are siblings of "13" and siblings must be placed on the same level according to convention.

Therefore, we need to analyse such problem cases and provide solutions for them in the Pedigree Visualizer. We discuss these problem cases and our solutions here.

3.1 Consanguineous Mates

The mating of two kins is likely to cause an individual and his siblings to be "promoted" up a generation or two to that of his mate. This is likely to introduce loops into the pedigree diagram.

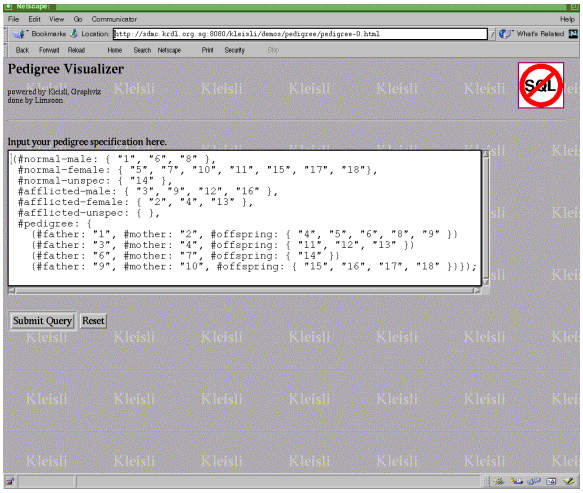

Figure 1: An example input.

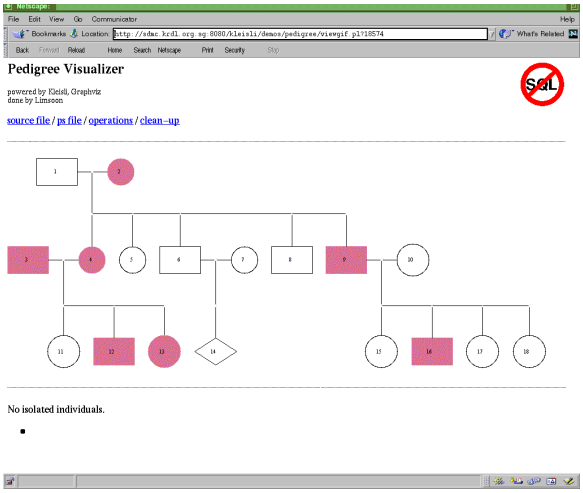

Figure 2: An example output.

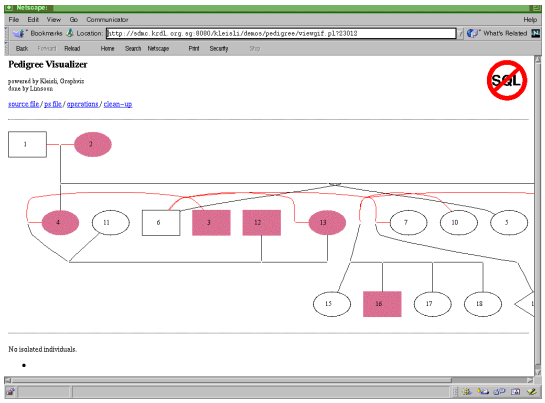

Figure 3: A bad pedigree diagram caused by the consanguineous multiple mating of "6" and "13."

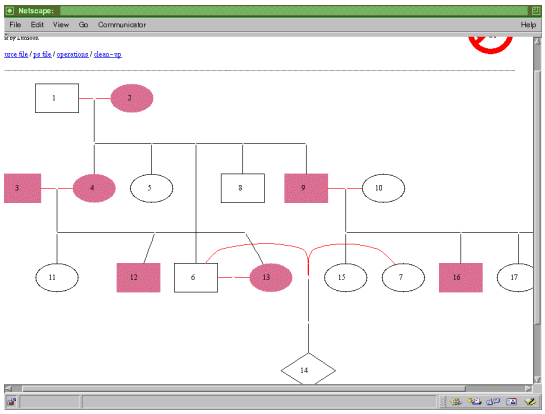

Figure 4: A relaxed pedigree diagram derived from Figure 3 by allowing siblings to occupy different levels.

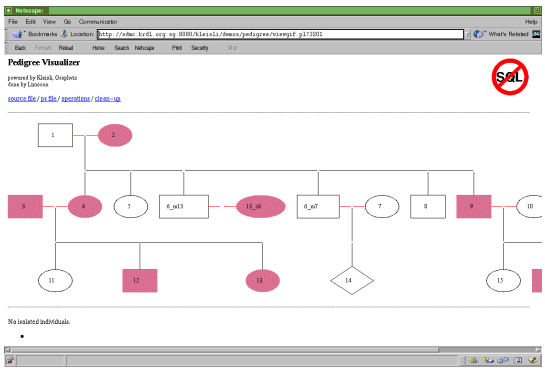

Figure 5: A repaired pedigree diagram by converting the consanguineous multiple matings of "6" into "normal" ones using aliases.

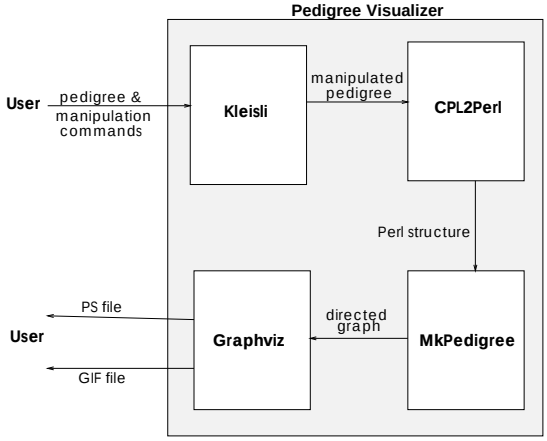

Figure 6: Architecture of the Pedigree Visualizer.

A solution is to relax the requirement that siblings be placed on the same level. Then we can move the mate from the older generation down to the level of the mate from the younger generation. This usually reduces cris-crosses and loops, as shown in Figure 4, which is exactly the same pedigree as Figure 3. This is the default solution made automatically by the Pedigree Visualizer when it has to draw a consanguineous mating unit. However, this is not a complete solution since cris-crosses are merely reduced, not eliminated.

An alternative solution is to "convert" a consanguineous mating into a "normal" mating. This conversion is done by introducing an alias. If x and y are in a consanguineous mating unit, we create an alias for one of them, preferably the one from a younger generation. For convenience, let us assume that an alias $x_{\textit{consanguineous with } y}$ is created for x. Then the mating unit of x and y is replaced by the mating unit of $x_{\textit{consanguineous with } y}$ and y in which $x_{\textit{consanguineous with } y}$ is treated as an "orphan" mate. That is, we do not insert a link between $x_{\textit{consanguineous with } y}$ and the parental mating unit of x. We also do not delete x. The rationale is as follows. Recall that x is at a younger generation than y. So x stays where it is. On the other hand, its alias $x_{\textit{consanguineous with } y}$ is placed besides y, nearer to the top of the pedigree where it is more conspicuous.

The Pedigree Visualizer provides a function that the user can invoke to detect consanguineous matings and to automatically convert these consanguineous matings into normal matings using the solution described above.

3.2 Multiple Mates

Recall that mates must be drawn side-by-side. This is not possible if an individual has more than 2 mates. A solution is to "convert" multiple matings into single matings. However, this conversion is done differently from that of consanguineous matings. If an individual x has multiple mates y_1, ..., y_n, we introduce aliases $x_{\textit{multiple with } y_1}$, ..., $x_{\textit{multiple with } y_n}$ in place of x. That is, we remove x from the pedigree and instead let $x_{\textit{multiple with } y_1}$ mates with y_1, .., $x_{\textit{multiple with } y_n}$ mates with y_n and link $x_{\textit{multiple with } y_1}$, ..., $x_{\textit{multiple with } y_n}$ to the parental mating unit of x. The links to the parental mating unit of x are needed to connect these aliases into the pedigree of x at the right place.

The Pedigree Visualizer provides a function that the user can invoke to detect multiple matings and to automatically convert these multiple matings into single matings using the solution described above. Figure 5 is derived from the problematic pedigree diagram of Figure 3 by the Pedigree Visualizer in two steps. First, the multiple matings of "6" are converted into two single matings, one of which is consanguineous with "13." Second, the consanguineous mating is converted into a normal one.

3.3 Multiple Families

A normal pedigree diagram descends from a single ancestral mating unit. If a descendant mates with an outsider, this outsider mate is portrayed in the pedigree as an orphan mate. That is, the ancestors and relatives of this orphan mate are excluded from the pedigree diagram. If the pedigree of the outsider mate are displayed, the resulting "integrated" pedigree diagram is necessarily full of cris-crosses because it is clearly impossible to position both sets of ancestors directly above this mating pair.

As it happens, the Pedigree Visualizer can also be used to "manage" integrated pedigrees. A mating unit and its immediate offsprings can be thought of as a core unit. The pedigree specification to the Pedigree Visualizer consists of a set of core units. These core units are *not* required to descend from a common ancestral mating unit. In other words, the pedigrees of multiple large families can be combined in a single input to the Pedigree Visualizer.

The Pedigree Visualizer does its best to layout the combined pedigree, which necessarily has many cris-crosses. However, the Pedigree Visualizer also provides a few useful pedigree extraction functions in this situation. We describe two below. It has a function where the user is allowed to choose several individuals and to automatically extracts their descendents and then to display the pedigree

restricted to these individuals and their descendents. It also has a function where the user is allowed to choose several individuals and to automatically extracts their ancestors and then to display the pedigree restricted to these individuals and their ancestors. Using these functions, the user can use the Pedigree Visualizer as a pedigree database to maintain a combined pedigree of all individuals of interest—being confident that, on an ad-hoc basis, he can use these functions to rapidly extracts suitable parts of the combined pedigree for viewing and research.

3.4 Errors in Data

There is another situation where a pedigree diagram cannot be layout properly. The user may not supply an error-free pedigree specification. For example, he may specify that an individual has two distinct parental mating units. He may also declare an individual to be afflicted in one place and normal in another place. Or he may specify a male as a mother or a female as a father. Or he may specify an individual to be one of his own descendents. This situation is expected to arise most often when two separate pedigree databases or files are merged and parts of them conflict with each other.

The Pedigree Visualizer provides a function to automatically merge two pedigree files. It also provides a function to check for all inconsistencies mentioned above in a pedigree specification. Once any inconsistency is detected, the user can use a number of manipulation functions to correct it. These manipulation functions are described in the next section.

4 Manipulations

Besides the more sophisticated functions described in the previous section such as extracting ancestors, extracting descendents, correcting multiple matings, correcting consanguineous matings, merging pedigree files, and determining inconsistencies, the Pedigree Visualizer also provides many simpler functions for editing and manipulating a pedigree diagram in a local smaller manner. We briefly describe some of these functions below.

The functions that affect individuals are (a) a function to delete specified individuals, which also delete relationships that these individuals participate in; (b) a function to rename an individual; (c) a function to create a new individual; and (d) a function to change the classification of an individual, viz. male vs female and normal vs afflicted.

The functions that affect mating units are: (a) a function to delete all mating units involving some specified individuals; (b) a function to delete a specified mating unit; and (c) a function to mate two specified individuals.

The functions that affect parent-child links are: (a) a function to create a parent-child link; (b) a function to delete a parent-child link; and (c) a function to "orphanize" an individual by removing his link to his parental mating unit. In (a) and (b), the child and both his parents must be specified. In (c), only the child needs to be specified.

5 Implementation

The typical process from the input of a pedigree specification to manipulations of the pedigree to the output of a resulting pedigree diagram as a GIF or Postscript file is depicted in Figure 6. This process has to go through four different modules in the Pedigree Visualizer: the Kleisli Query System, the CPL2Perl module, the MkPedigree module, and the Graphviz module. In the rest of this section, we now briefly describe the implementation of this process and these modules.

Logical	Lexical	Remarks
Unit	`()`	
Booleans	`true` `false`	

Numbers	`123`
	`123.123`
	`~123`
	`~123.123`
		Positive numbers
		
		Negative numbers

Strings	`"a string"`	String is put inside double quotes

Records	`(#`l_1`:` O_1`,`
	⋮`,`
	`#`l_n`:` O_n`)`
		Record is put inside round brackets.
		The label-value triplets enumerate the fields of the record

Variants	`<#`l`:` O`>`	Variant is put inside angle brackets
Sets	`{` O_1`,` ⋮`,` O_n`}`	Set is put inside curly brackets
Bags	`{|` O_1`,` ⋮`,` O_n`|}`	Bag is put inside curly-bar brackets
Lists	`[` O_1`,` ⋮`,` O_n`]`	List is put inside square brackets

Figure 7: The basic form of the Kleisli Exchange Format.

5.1 Input

The input to the Pedigree Visualizer is expected to be a pedigree specification either in the form of a data file or a data stream layout in the Kleisli Exchange Format.

The Kleisli Exchange Format is a very simple and flexible self-describing exchange format. The format consist of a lexical layer and a logical layer. The logical layer provides for the following data structuring concepts: sets, bags, lists, records, and variants—which corresponds to the data types supported by the Kleisli Query System [2, 3]. The lexical layer specifies how data corresponding to these structural concepts are layout in a data stream. The important property of the lexical layer is that each data structuring concept is given an unambiguous encoding. As a result, the data items can be parsed without reference to any schema. The basic form of the exchange format is given in Figure 7. Punctuation/indentation marks such as colon, commas, tabs, newlines, and spaces are not significant. The semicolon is used to indicate the end of a complex object.

Any data in this format can be directly manipulated by the Kleisli Query System, just like any data in a relational table can be directly manipulated by a relational database system. Any data in this format can also be directly parsed into Perl objects, Java objects, and other internal structures in other programming languages by a suite of Pizzkell/Kleisli interfaces. As a result any data in this format can also be manipulated by Perl, Java, and other programming languages as internal logical objects—as opposed to character strings—via these Pizzkell/Kleisli interfaces. The CPL2Perl module in the Pedigree Visualizer is one of these Pizzkell/Kleisli interfaces.

The Kleisli Exchange Format is designed for the exchange of general complex structures, so long as these structures are built up using sets, bags, lists, records, and variants. The pedigree specification is just a very simple instance of such structures. In the Pedigree Visualizer, it expects a pedigree

specification to conform to the following "schema" or structure:

```
(#normal-male: {string}, #normal-female: {string}, #normal-unspec: {string},
 #afflicted-male: {string}, #afflicted-female: {string}, #afflicted-unspec: {string},
 #pedigree: { (#father: string, #mother: string, #offspring: {string})})
```

That is, it should be a record consisting of the following seven fields: `normal-male`, which is a set of strings representing the names of normal males in the pedigree; `normal-female`, which is a set of strings representing the names of normal females in the pedigree; `normal-unspec`, which is a set of strings representing the names of normal individuals of unspecified sex in the pedigree; `afflicted-male`, which is a set of strings representing the names of afflicted males in the pedigree; `afflicted-female`, which is a set of strings representing the names of afflicted females in the pedigree; `afflicted-unspec`, which is a set of strings representing the names of afflicted individuals of unspecified sex in the pedigree; and `pedigree`, which is itself a set of records tabulating the parent-child relationships of all individuals in the pedigree. An instance of a pedigree specification layout according to the Kleisli Exchange Format is given in Figure 1.

5.2 Manipulations

After the input pedigree specification is supplied, the Pedigree Visualizer allows the user to apply a variety of analysis and manipulation functions mentioned in Sections 3 and 4. These functions are developed on top of the Kleisli Query System.

The Kleisli Query System [2, 3] is an advanced broad-scale integration technology that has proved useful in the bioinformatics arena [1, 6]. Many bioinformatics problems require access to data sources that are high in volume, highly heterogeneous and complex, constantly evolving, and geographically dispersed. Solutions to these problems usually involve multiple carefully sequenced steps and require information to be passed smoothly between the steps. Kleisli is designed to handle these requirements directly by providing a high-level query language, CPL, that can be used to express complicated transformation across multiple data sources in a clear and simple way.

The capability of Kleisli to express complicated transformation is precisely what we exploit to implement the various pedigree manipulation functions of the Pedigree Visualizer. We provide below a couple of examples for illustration.

Example 5.1 The function to "orphanize" a set of individuals `INDIVIDUALS` in a pedigree `OLD` to produce a new pedigree `NEW`. Recall from Section 4 that this is a function to remove these individuals from all parent-child relationships. It is implemented using Kleisli as follows:[1]

```
writefile
  (#normal-male: OLD.#normal-male, #normal-female: OLD.#normal-female,
   #normal-unspec: OLD.#normal-unspec, #afflicted-male: OLD.#afflicted-male,
   #afflicted-female: OLD.#afflicted-female, #afflicted-unspec:  OLD.#afflicted-unspec,
   #pedigree:
     { (#father:x.#father, #mother:x.#mother, #offspring:x.#offspring set-diff INDIVIDUALS)
     | \x <- OLD.#pedigree })
to NEW using stdout;
```

Example 5.2 Recall from Section 3 the function to "merge" two separate pedigree specifications `SPECA` and `SPECB` into a combined pedigree specification `NEW`. In this function, we need to take the union of the respective sets of normal and afflicted individuals from `SPECA` and `SPECB`. But we cannot

[1] The syntax of the Kleisli/CPL query language is as follows. $(\#l_1 : e_1, ..., \#l_n : e_n)$ constructs a record with fields l_1, ..., l_n with values e_1, ..., e_n respectively. $e.\#l$ returns the value in the l field of the record e. $\{f(x) \mid \backslash x \leftarrow e, C(x)\}$ constructs a set consisting of every object $f(x)$ where x is an object in the set e and the test $C(x)$ is true. The query language also has a rich set of operators such as set-diff which computes the difference of two sets and $\{+\}$ which computes the union of two sets.

take the simple union of the `pedigree` field of `SPECA` and `SPECB`. Instead, for each parental pair in these two pedigree specifications, we have to first find their separate sets of children and then take the union of these sets of children. The implementation in Kleisli/CPL is given below.

```
writefile
 (#normal-male: SPECA.#normal-male {+} SPECB.#norma-male,
  #normal-female: SPECA.#normal-female {+} SPECB.#norma-female,
  #normal-unspec: SPECA.#normal-unspec {+} SPECB.#norma-unspec,
  #afflicted-male: SPECA.#afflicted-male {+} SPECB.#afflicted-male,
  #afflicted-female: SPECA.#afflicted-female {+} SPECB.#afflicted-female,
  #afflicted-unspec: SPECA.#afflicted-unspec {+} SPECB.#afflicted-unspec,
  #pedigree:
   { (#father: p.#dad, #mother: p.#mum, #offspring:
        { c
        | \q <- SPECA.#pedigree {+} SPECB.#pedigree,
          q.#father = p.#dad, q.#mother = p.#mum,
          \c <- q.#offspring })
   | \p <- {(#dad:x.#father, #mum:x.#mother)| \x <- SPECA.#pedigree {+} SPECB.#pedigree}})
to NEW using stdout;
```

As can be seen, it is straightforward to express queries and thus to implement pedigree manipulation functions in Kleisli.

5.3 Output

The output of the Pedigree Visualizer is the GIF- or Postscript-formatted pedigree diagram. This step involves three modules of the Pedigree Visualizer: CPL2Perl, MkPedigree, and Graphviz. The input to this step is a manipulated or transformed pedigree specification layout in a data stream conforming to the Kleisli Exchange Format. The CPL2Perl module, which is a generic parser for the Kleisli Exchange Format, is used to convert this data stream into a structured Perl object. The MkPedigree module, which is implemented in Perl, takes this Perl object and produces a directed graph specification file. Finally, the Graphviz module takes this directed graph specification and generates a layout for the pedigree diagram in GIF and Postscript formats. Let us now proceed to the details.

The problem of "optimizing" the drawing of a pedigree, making it as visually attractive as possible, is a non-trivial one [7]. Nevertheless, in the absence of consanguineous loops and individuals with multiple mates, it is possible to efficiently produce fairly neat pedigree diagrams using a four-pass heuristic-based technique originally developed for directed graphs [4]. Very roughly, the technique works as follow. In the first pass, an optimal rank assignment is found using a network simplex algorithm that minimizes $\sum_{(v,w)\text{ is an edge}}(\text{rank}(w)-\text{rank}(v))$ subject to the constraint that $\text{rank}(w)-\text{rank} > 1$. In the second pass, the vertex order within each rank is set by iterating a few heuristics for local transpositions to reduce crossings. The heuristics are: (i) order a vertex by the median position of its parents; (ii) bias the median to the side where vertices are more closely packed; and (iii) swap adjacent vertices to reduce crossings. In the third pass, optimal coordinates for nodes are found by minimizing $\sum_{(v,w)\text{ is an edge}} |x_w - x_v|$ subject to sufficient space separation is kept between all vertices. In the last past, splines are made to draw the links. This algorithm makes neat drawings and is very efficient. It is implemented in the Graphviz module [5].

However, the Graphviz module was implemented as a general package for the automatic layout of directed graphs so that it can serve as a building blocks for many other applications. As a consequence, it accepts a general directed graph specification in a form that is very different from our pedigree specification. In essence, it accepts a list of arcs of the form $x \rightarrow y$, which specifies an arc is to be drawn from the node x to the node y. Unfortunately, in a pedigree diagram, the convention is that a link from a parental mating unit to a child is not drawn from either parent to that child; rather than,

it is drawn from the middle of the link connecting the two parents vertically down to the horizontal link connecting through all the children. This is not compatible with Graphviz's expectation.

To solve this problem, the Pedigree Visualizer uses the MkPedigree module, which is specially designed to convert a Perl structure representing a pedigree specification into a directed graph specification. We now describe the conversion process. The MkPedigree module has to generate some extra invisible nodes as follows. Given a mating unit of x and y, and children z_1, ..., z_n. The MkPedigree module generates extra invisible nodes xy, xyz_1, ..., xyz_n. It then tells Graphviz to draw the graph $\{x \to xy \to y, xy \to xyz_1, xyz_1 \to \cdots \to xyz_n, xyz_1 \to z_1, ..., xyz_n \to z_n\}$. It also tells Graphviz that x, xy, and y have the same rank, and that xyz_1, ..., xyz_n have the same rank. Graphviz does place nodes having the same rank at the same level. This causes horizontal arcs to be drawn connecting x, xy, and y, and horizontal arcs to be drawn connecting xyz_1, ..., xyz_n. Then Graphviz draws vertical arcs from xy down to xyz_1 and from each xyz_i to z_i. In other words, these invisible nodes serve as the connecting points in the middle of edges linking individuals, as needed by the drawing convention for pedigree diagrams. An example input to Graphviz produced by MkPedigree corresponding to the pedigree specified in Figure 1 is given in Figure 8; the corresponding output from Graphviz is of course the diagram in Figure 2.

```
digraph aGraph {
size="7.5,10"; center=true; concentrate=true;
fontname="SSERIFF"; fontsize=20; edge [dir=none];
/* normal male */ node[fontsize=20,shape=box];
"1"; "6"; "8";
/* normal female */ node [shape=ellipse];
"5"; "7"; "10"; "11"; "15"; "17"; "18";
/* normal unspecified */ node [shape=diamond];
"14";
/* afflicted male */ node [shape=box,style=filled,color=palevioletred];
"3"; "9"; "12"; "16";
/* afflicted female */ node [shape=ellipse,style=filled,color=palevioletred];
"2"; "4"; "13";
/* internal nodes */ node [style=invis,fontsize=0,height=0,width=0,shape=plaintext];
 "1_X_2"; "X_4"; "X_5"; "X_6"; "X_8"; "X_9";
"3_X_4"; "X_11"; "X_12"; "X_13";
"6_X_7"; "X_14";
"9_X_10"; "X_15"; "X_16"; "X_17"; "X_18";
/* spouses */
{rank=same;"1";"1_X_2";"2"} "1"->"1_X_2"->"2";
{rank=same;"3";"3_X_4";"4"} "3"->"3_X_4"->"4";
{rank=same;"6";"6_X_7";"7"} "6"->"6_X_7"->"7";
{rank=same;"9";"9_X_10";"10"} "9"->"9_X_10"->"10";
/* parent-child */
"1_X_2"->"X_4"; "X_4"->"4"; "X_5"->"5"; "X_6"->"6"; "X_8"->"8"; "X_9"->"9";
"3_X_4"->"X_11"; "X_11"->"11"; "X_12"->"12"; "X_13"->"13";
"6_X_7"->"X_14"; "X_14"->"14";
"9_X_10"->"X_15"; "X_15"->"15"; "X_16"->"16"; "X_17"->"17"; "X_18"->"18";
/* siblings */
{rank=same;"X_4";"X_5";"X_6";"X_8";"X_9";} "X_4"->"X_5"->"X_6"->"X_8"->"X_9";
{rank=same;"X_11";"X_12";"X_13";} "X_11"->"X_12"->"X_13";
{rank=same;"X_15";"X_16";"X_17";"X_18";} "X_15"->"X_16"->"X_17"->"X_18"; }
```

Figure 8: A specification corresponding to the input in Figure 1 for Graphviz to draw the pedigree diagram in Figure 2.

6 Remarks

The Pedigree Visualizer provides an automatic means for the layout and drawing of pedigree diagrams in a visually appealing manner. In addition, it provides a rich set of functions useful in the management and manipulation of large collection pedigrees.

There are several major general concepts involved in its implementation. A generic and flexible data exchange format useful for representing pedigrees and other complex data. A generic and powerful query system useful for implementing complex data transformations, including those functions of the Pedigree Visualizer. A generic and powerful system for the layout and drawing of directed graphs useful for drawing pedigrees and other kinds of diagrams. These concepts are embodied respectively by the followings modules of the Pedigree Visualizer: CPL2Perl, Kleisli, and Graphviz.

The generic concepts are combined and specialized for pedigree diagrams by simple Kleisli/CPL scripts implementing all the functions of the Pedigree Visualizer and by the special MkPedigree module that interfaces the Kleisli module to the Graphviz module via the CPL2Perl module. As all the Kleisli, CPL2Perl, and Graphviz modules are pre-existing components, the only effort involved in building the Pedigree Visualizer is in writing these scripts and in implementing the MkPedigree module.

In fact, relying on the power of the Kleisli Query System, the Pedigree Visualizer can be substantially generalized by directly interfacing it with medical record databases. In a typical situation, a pedigree specification is prepared for a specific disease. We can imagine a medical record database capturing the marital and parental relationships of a large population and their medical histories. Suppose we want pedigrees of people with a particular disease X. A Kleisli/CPL query can be easily written against this medical record database to identify all individuals afflicted with X. A second Kleisli/CPL query can also be easily written against this database to extract all ancestors and descendents of these afflicted individuals. Then a third Kleisli/CPL query can be written to construct the pedigree specification using these previously extracted information. After which, the Pedigree Visualizer can be brought into play to present the extracted (combined) pedigree. Finally, analysis functions on pedigrees such as determining if a trait is dominant or recessive and probability that a specified individual is heterozygous and so on can also be incorporated into the Pedigree Visualizer.

Availability. The Pedigree Visualizer is at `sdmc.krdl.org.sg:8080/kleisli/demos/pedigree`. The Kleisli Query System is available from Kris Informatics, Inc.; `www.kris-inc.com`. The Graphviz package is available from AT&T Research Labs; `www.research.att.com`. The CPL2Perl and other packages in the Pizzkell/Kleisli suite are available from Kent Ridge Digital Labs.

References

[1] Benton, D., Bioinformatics—principles and potential of a new multidisciplinary tool, *Trends in Biotechnology*, 14:261–272, 1996.

[2] Chung, S.Y. and Wong, L., Kleisli, a new tool for data integration in biology, *Trends in Biotechnology*, 17(9):351–355, 1999.

[3] Davidson, S. *et al.*, BioKleisli: A digital library for biomedical researchers, *International Journal of Digital Libraries*, 1(1):36–53, 1997.

[4] Ganser, E.R. *et al.*, A technique for drawing directed graphs, *IEEE Trans. Software Engineering*, 19(3):214–230, 1993.

[5] Gansner, E.R. and North, S.C., An open graph visualization system and its applications to software engineering, *Software—Practice and Experience*. to appear.

[6] Karp, P.D., Database links are a foundation for interoperability, *Trends in Biotechnology*, 14:273–279, 1996.

[7] Tores, F. and Barillot, E., Optimizing pedigree drawing using interval graph theory, *Currents in Computational Molecular Biology*, 194–195, 2000.