ECS 165B Database Systems
Dr. Bertram Ludäscher

Winter 2014
January 10

GROUP PROJECT 1 – Graph-Data Visualization Tool (GDT)

Due: Friday, January 31 (Form a group and get started early! Submission details via class list)

Project Overview. Graph data is ubiquitous: Whether it is social networks (e.g., Facebook), the (semantic) web with its linked pages or "Linked Open Data", whether we study provenance graphs and lineage trees (e.g., family trees, pedigrees, genealogies, phylogenies) or program execution traces, parse trees, XML documents, `makefile` dependencies, etc. – in all these cases we have graph data that needs to be stored, queried, analyzed, and visualized. The goal of this group project is to develop a small tool or toolkit *GraphvizDataTools (GDT)* for generating and visualizing graphs from some given input data. For the actual graph visualization, we employ a well-known, simple, yet effective graph visualization tool called `GraphViz`, see http://www.graphviz.org/.[1]

There is a common structure to all project variants below:

1. We are given some data D, either rather explicitly (e.g., in form of a PostgreSQL table, CSV file, XML file, etc.), or more implicitly (e.g., the edges in a Rule-Goal graph of a Datalog program, the dependencies in a `makefile`, the hyperlinks in a web site, etc.)
2. We need to create a "graph view" from the given data, i.e., extract and transform (parts of) the data D so that we can think of it as a set GV of labeled edges $x \xrightarrow{\ell} y$. Thus, we can think of the result as a table $GV(N, L, N)$ whose rows are labeled edges, linking nodes $x, y \in N$ via an edge with label $\ell \in L$.
3. We need to create a `GraphViz` input file F from GV, i.e., F is is an ASCII file, using the `DOT` language[2]. When creating F from GV we might want to apply a separate *stylesheet* S which describes how to render the different *node types* NT and *edge types* ET. For example, when rendering a Rule-Goal graph, we might use NT to distinguish rule nodes (squares) from goal nodes (ovals). Edge types ET can then be defined based on the types of the two nodes that make up the edge and/or the label ℓ of that edge. The style sheet S can be modeled as a table that associates shapes, colors, etc. with node and edge types.
4. Finally, we render F, either using `GraphViz`, or some other tool that understands `DOT` files.

All of the project variants below are based on the common framework described in (1–4) above:

PostgreSQL Data Viewer (GDT-PDV). The user defines GV as a PostgreSQL view on top of the given database D. Node and edge types are also defined as views over GV and/or D. The stylesheet S is provided either "on demand" via a CSV file, or is given as a fixed table. Standard use case: the user picks a particular view GV (there should be at least two variants), a stylesheet S and the tool generates and the visualizes F. Some example data will be provided, e.g., for family relationships, or from Mondial or GapMinder, and should be used to demonstrate the system. Note: You can develop GDT-PDV as a Postgres command-line tool, or embed the necessary SQL, e.g., in Java, Python, or another language.

PostgreSQL Schema Viewer (GDT-PSV). This is a variant of GDT-PDV, where the data to be displayed is the user's database schema, stored in the PostgreSQL system catalog. That is, we think of the user's database schema (say Mondial) as a graph consisting of nodes (representing tables and their attributes) and edges, representing, e.g., foreign keys between tables, and the association between table names and attributes, i.e., column names.

[1]Check out the Graph Gallery there! Also see the Graphviz Wikipedia page.
[2]http://en.wikipedia.org/wiki/DOT_language

Datalog Result Viewer (GDT-DRV). In this variant, D is a Datalog database/file. As in GDT-PDV, a view GV is defined which is combined subsequently with a stylesheet S to generate F. Note: you might want to "wrap" your Datalog system (e.g., DES-Datalog or DLV) within another programming or scripting language, e.g., Java or Python.

Rule-Goal Graph Viewer (GDT-RGV). Here, instead of rendering data, we want to render the Datalog program (this is somewhat similiar to GDT-PSV, but instead of showing just the table schema, the tools generates the Rule-Goal graph). You can use, e.g., Python or a parser generator to parse the given input Datalog program and generate F from it. A fixed style sheet can be used here.

XML Tree Viewer (GDT-XTV). An XML document can be viewed as a tree. With GDT-XTV the user can load an XML document and rendered it via `GraphViz`. Stylesheets should be used here, e.g., to render XML tags differently from attribute/value pairs.

XML DTD Viewer (GDT-DTD). An XML DTD can be thought of as a set of grammar rules $lhs \rightarrow rhs$, where the lhs is an XML element type E, and rhs is the content description of E-elements. This "DTD grammar" has a natural tree representation. GDT-DTD should create this tree. Stylesheets should be used to render different element types differently.

Conceptual Model Viewer (GDT-CMV). Here the input D is an ER diagram or UML diagram in a suitable format, e.g., simple ASCII. The GDT-CMV tool then transfers D into a `DOT` file F, applying a stylesheet S along the way. Here, you can make up your own convenient language for ER or UML, then generate the visualization from it.

MyData Viewer (GDT-MDV). If you have access to a dataset that interests you and that you think could lend itself to an interesting graph visualization, let me know as soon as possible! Thus you would be developing a "custom data viewer" for the data that interests you. Main requirement: the solution should fit reasonably well the general framework (1–4) above.

About Group Projects. Group projects should involve 2 or (ideally) 3 students. Even if you can easily do a project by yourself, view it as an opportunity to learn from each other and to experience team work. As always, but in particular for these types of projects: What you get out of it, depends on what you put into it. These projects (and the ones following) are less as a means to test whether you understand the class material (that's done via individual assignments and exams), but rather as an opportunity to practice your programming skills, work with some new technologies, etc. In particular, you should apply software engineering principles and best practice.

Submission details and specific reporting formats, along with any other questions on the project will be discussed through the class forum (i.e., mailing list and web site). Please also keep a "project notebook" to document your meetings, discussion results, decisions, work packages, etc. Having a project notebook (e.g., a shared Google doc) will make creating documentation for your project submission a snap.[3]

[3] In the biological sciences, the related lab notebooks are mission critical and a good inspiration for project notebooks.