Twined Language Specification

Lucas Weissman   Zach Sturdevant

May 22, 2024

Introduction

"Twined" is a knowledge programming language that aims to revolutionize the way we manage and interact with textual information. By converting unstructured text data into structured, navigable knowledge graphs, "Twined" enhances comprehension and analysis, providing a unique interactive interface that allows users to explore information and create insights through based reasoning visualizations.

"Twined" addresses the problem of inefficient management and interaction with the ever-growing volume of new information one can retain, making it a powerful tool for researchers, analysts, students, and anyone who works with text-based information and wants to uncover hidden connections, identify patterns, and derive meaningful conclusions. By providing a platform that encourages exploration, critical thinking, and the synthesis of ideas from various fields, "Twined" aims to replicate the transformative learning experience offered by liberal arts institutions by promoting users to twine the threads of information into their own personal tapestry of knowledge.

Design Principles

The design of "Twined" is guided by both aesthetic and technical ideas that aim to make knowledge more accessible and manageable for users. From an aesthetic standpoint, the language strives to provide a simple and intuitive starting point that gradually evolves into a powerful graphical user interface. This approach ensures that users of all skill levels can easily interact with the language and leverage its capabilities without being overwhelmed by complexity. By prioritizing user experience and usability, "Twined" aims to create an engaging and visually appealing environment that encourages exploration and discovery.

From a technical perspective, "Twined" uses the power of generative AI to enhance the efficiency and effectiveness of knowledge access and management. The language aims to minimize user input while maximizing the usefulness of the output, allowing users to obtain valuable insights and connections with minimal effort. As the language evolves, it will incorporate features that enable users to upload text and express their desired outcomes in natural language. "Twined" will then handle customized outputs to the user.

Examples

1. (a) Program 1 input

```
{Nancy, (Fred, Sally, Sarah,)}
{Fred, (Nancy, Sally, Sarah,)}
{Sally, (Jeff, Jonny, Timmy,)}
{Jeff, (Sally, Jonny, Timmy,)}
```
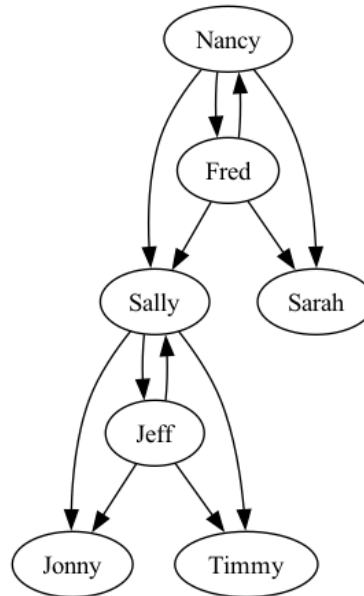
(b) Program 1 output

Figure 1: Family Relationship

(c) This graph represents the relationships between family individuals. Each node represents a person, and each edge represents a direct relationship between two individuals. For example, "Nancy" has relationships with "Sally," "Fred," and "Sarah," as depicted by the edges connecting their respective nodes.

2. (a) Program 2 input

```
{Sunlight, (PlantGrowth,)}
{Water, (PlantGrowth,)}
{SoilNutrients, (PlantGrowth,)}
```
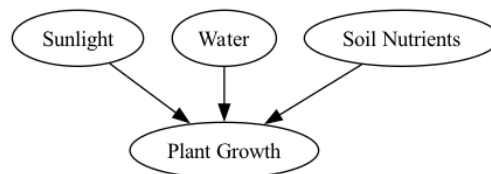
(b) Program 2 output

Figure 2: Plant Growth Relationship

(c) The graph represents factors affecting plant growth. Each node represents a factor such as "Sunlight," "Water," and "Soil Nutrients," and each edge represents the influence of these factors on "Plant Growth," as depicted by the edges connecting their respective nodes.

3. (a) Program 3 input

```
{European Fascism, (Germany, Italy, Spain,)}
{Germany, (Adolf Hitler,)}
{Italy, (Benito Mussolini,)}
{Spain,(Francisco Franco,)}
```
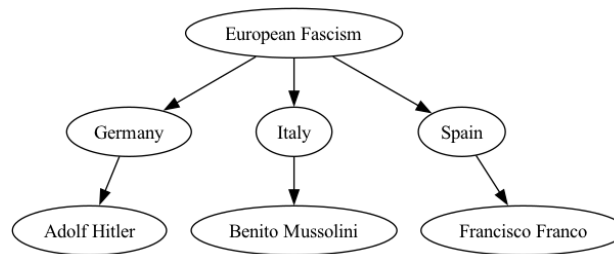
(b) Program 3 output

Figure 3: Famous Eureopean Fascists

(c) This graph shows connections between European Fascism and leaders of different Fascist countries. Each node represents a topic and how those topics are connected in a top down manner.

Language Concepts

To effectively use "Twined," users should understand the fundamental concepts of knowledge graphs and how they represent information. While the language aims to minimize the need for users to grasp low-level primitives, a basic understanding of nodes and edges is still beneficial. Nodes represent entities or concepts within the text, while edges signify the relationships or connections between them. These concepts enables "Twined" to create dynamic interactive knowledge graphs that highlight both hierarchical and associative relationships. By comprehending how these elements interact, users can better navigate and interpret the visual representation of their data.

"Twined" will provide an intuitive and user-friendly experience through an interface, for inputting their desired information to explore knowledge graphs. The user interface will offer tools for navigating the graph, filtering information, and discovering new connections.

Formal Syntax

< **expression** >	::=	< **Node** >
	|	< **NodeInfo** >
	|	< **EdgeList** >
	|	< **NodeName** >
	|	< **NodeList** >
	|	< **Assignment** >
	|	< **AssignmentList** >
	|	< **NodesAndAssignments** >
< **Node** >	::=	{ < **String** >, < **NodeName** > }
< **NodeName** >	::=	< **String** >
< **EdgeList** >	::=	(< **NodeName** > +)
< **NodeInfo** >	::=	< **String** >
< **NodeLists** >	::=	< **Node** > +
< **Assignment** >	::=	{ < **NodeName** >, < **NodeInfo** > }
< **AssignmentList** >	::=	< **Assignment** > +
< **NodesAndAssignments** >	::=	{ < **NodeList** >, < **AssignmentList** > }

Semantics

Syntax	Abstract Syntax	Prec./Assoc.	Meaning
NodeName	Name of string	N/A	NodeName is a primitive represented using an F# string
(name1, name2,)	EdgeList list	N/A	An EdgeList is a list of NodeNames stored in an F# list
NodeInfo	Info of string	N/A	NodeInfo is a primitive represented using an F# string
name, (name1,)	Node of (NodeName * EdgeList)	N/A	A node is a combining form that stores a NodeName and a list of nodes that that node is connected to in an F# list
{node} {node}	Node list	N/A	A series of nodes separated by whitespace including newlines. These are used to store all nodes in a graph
`^^NodeName := NodeInfo^^`	Assignment of NodeName := NodeInfo	N/A	A combining form that uses a NodeName and its associated information stored in an F# map
`^^NodeName := NodeInfo^^`	List of Assignments	N/A	A series of assignments to be stored in an environment
NodeList and AssignmentList	Tuple of NodeList and AssignmentList	N/A	A list of nodes that form the graph and the assignments of each node

Remaining Work

1. **Backend Enhancements**

   (a) Enable PDF and other input formats for summarization and graph creation.

   (b) Improve the parser for flexible inputs to prevent errors from missing punctuation.

   (c) Optimize parsing and evaluation algorithms for larger datasets and implement caching.

   (d) Refactor the backend for horizontal scaling and utilize cloud services for dynamic scaling.

2. **Frontend Improvements**

   (a) Develop a sophisticated, intuitive interface using modern frameworks like React or Vue.js.

   (b) Implement real-time feedback and enhance visualizations with interactive elements and theming options.

   (c) Utilize JavaScript libraries like D3.js for more dynamic and visually appealing graphs.

   (d) Provide export options for graphs in various formats.

3. **User Experience and Security**

   (a) Enhance the web interface for smoother interaction and better navigation tools.

   (b) Implement robust authentication systems, including OAuth and connection with student email.

   (c) Integrate OCR technology for extracting text from images and PDFs.

   (d) Add text-to-speech and speech-to-text features for enhanced accessibility.

4. **Advanced Features and Documentation**

   (a) Integrate machine learning for insights, recommendations, and automatic node categorization.

    (b) Develop thorough documentation and create a community support portal.

5. **Features Not Implemented** We had planned to implement several additional features and commands, such as direct interaction with OpenAI's interface for chatting, and the use of OCR technology to convert PNG/JPG images to work with notes on paper. Additionally, we attempted to work with PDF-to-text conversion. Unfortunately, due to time constraints, we were unable to fully develop and integrate these functionalities by the final submission.