Twined Language Specification

Lucas Weissman Zach Sturdevant

May 13, 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

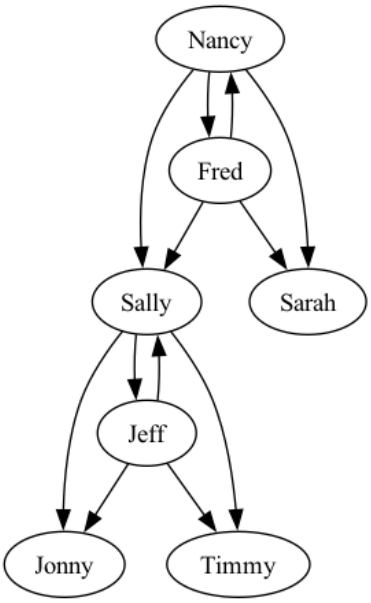


Figure 1: Family Relationship

   (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

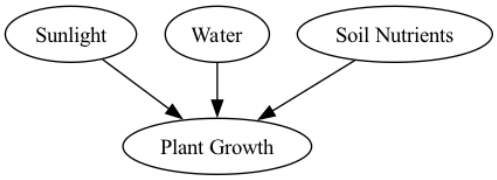


Figure 2: Plant Growth Relationship

(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

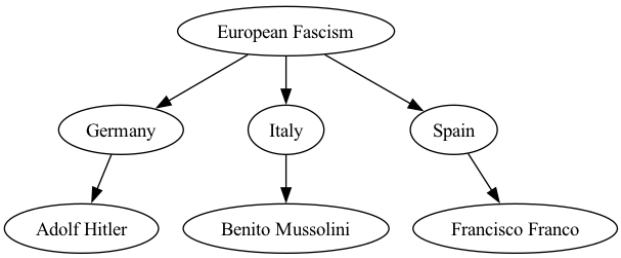


Figure 3: Famious Eureopean Fascists

(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** >
	|	< **edgeList** >
	|	< **nodeName** >
	|	< **listOfNodes** >
< **Node** >	::=	{ < **String** >, < **nodeName** > }
< **nodeName** >	::=	< **String** >
< **edgeList** >	::=	(< **nodeName** > +)
< **listOfNodes** >	::=	< **Node** > +

Semantics

Syntax	Abstract Syntax	Prec./Assoc.	Meaning
<NodeName>	Name of string	N/A	NodeName is a primative represented using an f# string
<EdgeList>	NodeName list	N/A	An edgeList is a list of NodeNames stored in an f# list
<Node>	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
<NodeList>	Node list	N/A	A series of nodes in the form {node} {node} separated by whitespace including newlines these are used to store all nodes in a graph
<NodeInfo>	NodeInfo of String	N/A	primative of information that will be associated with a node represented with an f# string.
<NodeName> := <NodeInfo>	Definition of NodeName := NodeInfo	N/A	A node is a combining form that stores a NodeName and a list of nodes that the named node is connected to in an f# list

1. Primitive values:

(a) NodeName: A series of characters digits or spaces that represent the name of a node. stored in a string.

(b) EdgeList: A series of NodeNames, or 0 stored in the form (NodeName, NodeName,) that contains the NodeNames of nodes connected to the node, stored in a list.

2. Combining forms:

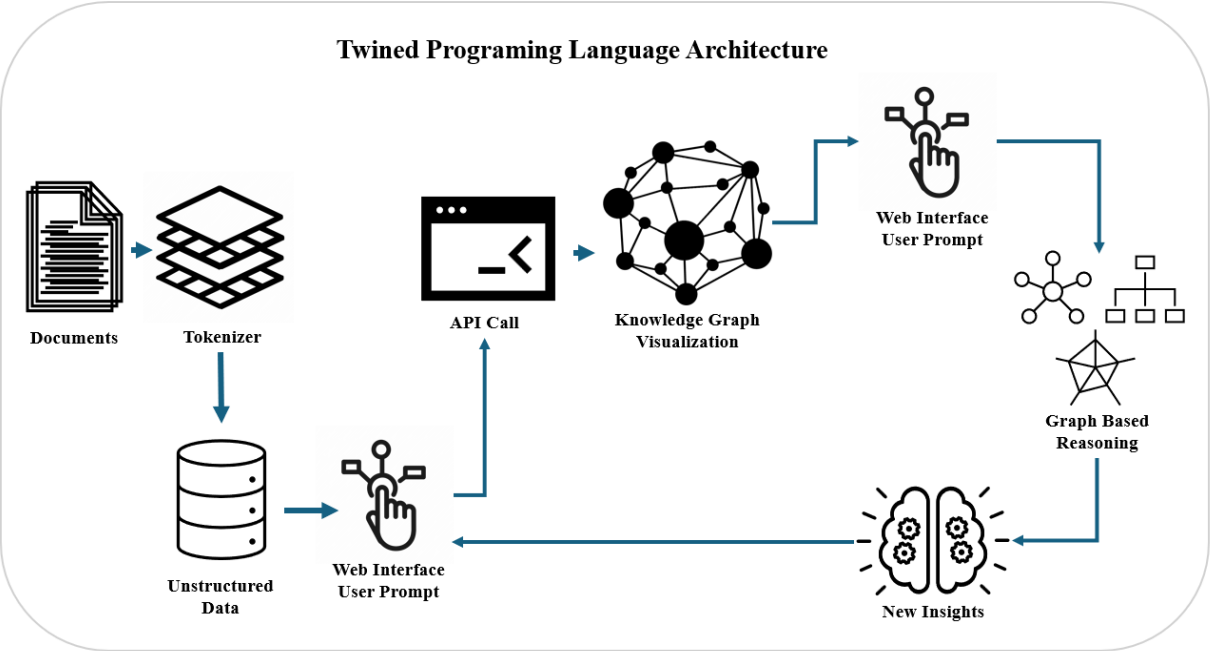
(a) Node: A tuple of the form NodeName, (EdgeList) that represents a full node, which is the name of the node and its EdgeList. In the future the node name will be used to access a dictionary that contains the information associted with that node

(b) NodeList: A series of nodes in the form Node Node separated by whitespace including newlines these are used to represent all nodes in a graph

3. Evalutation:

(a) The programs in our language read inputs given in a txt file, it parses the text file and separating the series of nodes and their edges.

(b) When the program is evaluated it produces a graph using the nodes and their edgelist on the screen


Twined Programing Language Architecture
Documents
Tokenizer
Unstructured Data
Web Interface User Prompt
API Call
Knowledge Graph Visualization
Web Interface User Prompt
Graph Based Reasoning
New Insights