KB

Usage of map<string, vector<vector<string>>> * FactMap	- This data structure was used to streamline the INFERENCE process by storing different bindings of actors in an easily accessible way. This means that in order to evaluate the validity of a fact, all a function would have to do is check for the string sequence's presence in this map. The map keys are strings that represent the Fact type. Each index contains a vector of string vectors that hold different combinations of actors that represent facts. - **Using maps** gives flexibility when removing Facts: removing an entire fact archetype means having to only use std::map's erase function.
toString()	This function was used to get string representation of the entire KB. This string representation would be able to be returned and used in writing to sockets (webserver), files (DUMP) and provide debugging info.

Rule

Rule Object	- **The rule object** was created with the five features that promote exception safe code: copy/move constructors and operator overloads, a default constructor, and a destructor. - The rule object has its own personalized **swap function** that allows the use of the copy/swap idiom. This idiom reduces the overhead of copying and moving the rule object. - Includes comparison operators for completion.

RB

Storing rules in a map:	Makes accessing rules easier, eliminates the need for a rule fetching function. It also allows us to easily check for the presence of a rule and avoid duplicate rules.
toString:	Returns a string representation of all rules in the RB. Calls each individual rule's toString and is called by DUMP.

Parse

Parsing a line	- **Parsing a line** is done through the ParseLine() function. The function will execute different commands depending on input; these include DUMP, INFERENCE, LOAD, and DROP. The function will call toString(), ParsePred(), ParseRule(), inference(), and Remove() (removing a fact/rule) respectively. - **Terminal input**: pressing q to leave program cleanly.

Query

Inference	**Traverse()**: a helper function that creates combinations of relevant actors
EvaluateFact()	Checks if a fact exists in the KB
ruleEvaluate()	- **ruleEvaluate()**:Evaluates the validity of a rule. Checks at key points to see if the rule has already been evaluated (exists as a fact in the KB) to eliminate unnecessary computation. - **Split between operateOR() and operateAND()**: the two logical operations for a rule have drastically different functionality when it comes to multithreading. OR would utilize the std::async function and AND would utilize std::threads and unique_locks. - **Boolean check shortcuts**: In the case of OR, if an evaluated component returned true the operateOR() function would automatically return true because one “true” value is all an or needs to evaluate to true. Likewise with operateAND, if a “false” value was encountered, the function would automatically return false to save operating time. - **Async**: operateOR is multithreaded through the use of C++11’s std::future and std::async. Each operateOR call starts a new std::async call. We can start all of these asynchronous threads because the value of an OR requires only one of its predicates to return true.

SRI

SRI	Manager class that creates and calls each of the objects.