# CS8803 – Knowledge Based AI

# Project 2

# Puyan Lotfi

## Implementation:

The implementation of my project 2 is far different from my project 1. To start, I wrote project 1 in lisp, but in project 2 I wrote it all in C++ and used Flex and Bison (lex and yacc) to construct a Lexer/Parser to read the input format as well as the plan/script format that I came up with which I based on the original input format.

Implementation details aside: my design is one thing that did not change much from project 1. In fact, I wanted to use something similar to scripts in project 1 but just had too much trouble with lisp semantics to get it all done. As a result, I don't use anything like pre and post conditions that most people might have in their projects. Instead I try to map plan scripts to data in my knowledge base, and once the mapping is made I use the mapped events to figure out what agents might have been associated in the "intention" of the plan.

## Core Components:

- **"Frame Registry":** A global mapping of symbols (ie token of text) to frames for the events and facts read in. This is what I refer to as the "knowledge base" or in code as the "GlobalFrameRegistry." (aicop.y) What's more is that this mapping can be thought of as a graph that interconnects related events. Most of the code that walks these related frames isn't used in project 2, but it was written while project 1 was being reimplemented.
- **"Plan Registry":** A global mapping of symbols to frames for the plans/script entries. In code this is called the "GlobalPlanRegistry".

- **Frames:** everything is build on top of frames (my frame class). Maps of frames, lists of frames, maps of lists of frames.
- **Context Free Lexer+Parser:** I only mention it because by implementing a context free grammar rather than throwing a bunch of string tokenizer code at the problem, I made it extremely easy for my self to reuse and extent the existing input format provided. If you open one of my plan txt files you'll see something very similar to the inputs events (minus the agent specific data), but with some added attributes like the name of the plan, the plan entry and the next plan entry names, and if the entry is the ending or starting entry of the plan/script.
- **Script/Plan Mapping + agent guessing code:** Once all the scripts/plans and event inputs are read in, there is code that is run that for every script tries to map each entry of the script to a event that may have happened and is stored in the knowledge base. Once a reasonable number of entries are mapped, it begins to infer what agent and co-agent to associate the intention of the plan to.
- **The Plan/Script:** A plan, or script, in my project is almost identical to an event except that it doesn't have any data specific to any agents. What is added is information about the name of the script as well as an order that events in the plan happen in.

## Design and Methodologies:

My methodology boils down to matching data from my scripts (that have an order, which is why I didn't implement pre and post conditions) to data in my knowledge base that consists of event and fact frames. Each entry in a script looks very similar to an event, except that any specifics related to agents or co-agents is removed. Every script has information about the intention of the potential actors. Once a match is made between data from the script and data from the knowledge base, the program tries to figure out which agents were most likely the key players of the story that the script is telling. I figure out key agent and the supporting agent base on the frequency that agents and co-agents occur in the

matching events from the knowledge base (agent is most frequent, co-agent is second most frequent).

## Self Critique and Evaluation:

My biggest bit of satisfaction with project 2 is that I was able to catch up to where I was in project 1 without relying on other people's code. My second biggest bit of satisfaction was with the use of a parser/lexer, since I enjoy using tools to get things done across disciplines. I am also fairly happy with letting the scripts handle the order of causal chains. The only way for a set of events to make sense is if it matches the order of the events in a script, and I see no reason to hard-code pre and post conditions.

Overall I am not very satisfied with the state of my project 2. I think my mapping code is very difficult to read and not very stream lined. I also think that my code will have problems if there are multiple causal chains that have different actors involved. I lot of this is because I went about thinking of the problem differently than many of my peers.

If I had more time I would use the code in the function GetAllAssociatedFrames() (this function traverses related frames and prints them out) to group all causally linked frames together. Then for each of these causal groups I would attempt to map every script just as I do currently. Also, I would employ some kind of hard-coded logic somewhere (probably in the input data, that then gets read in) to tell my program that certain agents (like rex-luthor, who has his hand in everything) are not always important in building causally related groups.

Another thing I am not too satisfied with is the way I determine the important agent in a given causal chain. Right now I just total the frequency that I come across that agent in a list of events, and I already have a lot of smarter ideas on how to do that.

## Knowledge Representation and Output:

Just to be explicit, I will display some of the input and output of my program, and focus on the robert-e-ford scenario.
The causal chain for this scenario is:

(  ( go-to, ( robert-e-ford,store,,,, )))
(  ( give, ( robert-e-ford,fake-check,clerk,,, )))
(  ( give, ( robert-e-ford,fake-id,clerk,,, )))
(  ( suspect, ( clerk,fraud,,,, )))
(  ( get-caught, ( robert-e-ford,,,,, )))
(  ( call, ( clerk,police,,,, )))
(  ( run-away, ( robert-e-ford,,,,, )))
(  ( chase-down, ( police-officers,robert-e-ford,,,, )))
(  ( get-away, ( robert-e-ford,accomplice,,,, )))

These events are somewhere in my knowledge base, with no order.

The script for this scenario is in rob-store-plan.txt. One line of the script looks like this:

(ACPLAN(rob-store-plan8 rob-store-plan9 rob-store-plan) (chase-down (police-officers , BLANK , , , ,)))

The information given in one line of a script tells what actions and general agents are involved (like police), and what event comes after. The BLANK filler can be used as a sort of helper to show that some agent or object belongs in that position of the frame, but I don't think it is really necessary given the way I implemented project 2.

So the list of events in rob-store-plan.txt is used to get all events in the causal chain out of the knowledge base. What happens next is that those events are

printed out, and use with the intention line of the script to figure out which agent had which intention to do what. The intention line of the rob-store-plan.txt script is:

(ACPLAN(rob-store-plan10 rob-store-planACPLAN-END rob-store-plan) (steal-merchandise-from (agent , , store , , ,)))

And the output for this chain is:

Another Plan: rob-store-plan
( ( go-to, ( robert-e-ford,store,,,, )))
( ( give, ( robert-e-ford,fake-check,clerk,,, )))
( ( give, ( robert-e-ford,fake-id,clerk,,, )))
( ( suspect, ( clerk,fraud,,,, )))
( ( get-caught, ( robert-e-ford,,,,, )))
( ( call, ( clerk,police,,,, )))
( ( run-away, ( robert-e-ford,,,,, )))
( ( chase-down, ( police-officers,robert-e-ford,,,, )))
( ( get-away, ( robert-e-ford,accomplice,,,, )))
Intention of robert-e-ford:
steal-merchandise-from store