Advanced Topics in Artificial Intelligence

T-720-ATAI-2016

EXERCISE 5 (Final Project)

In the final programming exercise of this course you will pair up to do some seed programming and teaching of an AGI-aspiring cognitive architecture (NARS).

Download OpenNARS

OpenNARS is an open source implementation of NARS. Download version 1.7.0 from https://drive.google.com/folderview?id=0B8Z4Yige07tBUk5LSUtxSGY0eVk&usp=sharing . Note that you can also download the source code here, but if you're going to do that, it might be better to get it from GitHub: https://github.com/opennars/opennars. You will not need the source code for this exercise, but you should probably check out the wiki: https://github.com/opennars/opennars/wiki.

When you have unzipped the file, run OpenNARS_GUI.jar to start the program and start the interactive session by clicking the OpenNARS button. Resetting the memory does not appear to work, so if you want that, you'll have to restart the program.

To enter a sentence, press Ctrl + Enter. Documentation about an older but similar interface is here: https://github.com/opennars/opennars/wiki/Graphical-User-Interface. The Narsese grammar is here: https://github.com/opennars/opennars/wiki/Input-Output-Format.

Tip: You can't copy text that you've typed into the OpenNARS GUI. It is probably a good idea to write your commands somewhere else and then paste them into the GUI, so that you don't lose any work.

Your final project

The final programming assignment of this course is a bit different from the earlier ones. Due to the size, we ask you to pair up in order to somewhat mitigate the workload. This assignment is very open, and we will only give you fairly high level descriptions of the tasks that you should make NARS learn:

~~- Diagnosing, treating and preventing disease~~

~~- Dealing with numbers~~

~~- Natural language processing~~

- Game playing

There are roughly two different ways in which NARS can acquire a piece of knowledge: you can tell it directly, or the system can learn it. In this assignment, you will do both. In the first stage, you should think about the rules you want NARS to have. Then in the second stage, you'll start from the bare minimum and try to teach NARS as many interesting rules as possible by giving examples and asking questions. Example:

Part 1:

```
// transitivity: if x < y && y < z then x < z
<(&&, <$x --> (/, smaller, _, $y)>, <$y --> (/, smaller, _, $z)>) ==> <$x -->
(/, smaller, _, $z)>>.
<A --> (/, smaller, _, B)>.
<B --> (/, smaller, _, E)>.
<A --> (/, smaller, _, E)>?
```

Part 2:

```
<A --> (/, smaller, _, B)>.
<A --> (/, smaller, _, C)>.
<A --> (/, smaller, _, D)>.
//<A --> (/, smaller, _, E)>.
<B --> (/, smaller, _, C)>.
<B --> (/, smaller, _, D)>.
<B --> (/, smaller, _, E)>.
<C --> (/, smaller, _, D)>.
<C --> (/, smaller, _, E)>.
<D --> (/, smaller, _, E)>.
<A --> (/, smaller, _, E)>?
<(&&, <$x --> (/, smaller, _, $y)>, <$y --> (/, smaller, _, $z)>) ==> <$x -->
(/, smaller, _, $z)>>? // this is actually too complex to learn
```

Game playing

In this assignment you're going to teach NARS about playing a made-up tile-based role-playing game. The game has NPCs, weapons, armor, keys and chests (and whatever else you want). The protagonist can move to the location of any of these things to interact with them. Loose items can be picked up. Chests can be opened with the appropriate keys. Enemies can be hit with a weapon that you own, but they can

also hit you. You can define different weapon and armor types that are strong/weak against each other (influencing the probability of a one-hit-kill). When chests are opened or NPCs are vanquished, you can pick up their items.

You can also make some NPCs friendly, so they won't attack you and will just give/trade their items if you ask (which you should make preferable). You can add properties to tell beforehand whether an NPC is friendly. You can also make things like properties, chest contents, NPC's items, weapons and armor partially observable and define actions (e.g. "look") to gather more information.

All actions take time. One goal should be to avoid dying, while another goal could be acquiring an item, opening a chest or vanquishing an enemy. Maybe you want to open chest1, for which you need key1, which is carried by npc1, who also has a blunt weapon (e.g. a club) and padded armor (strong vs. blunt), so you need a sword (slashing) and your own padded armor, etc. This way you can see there's a whole sequence of actions that the player would need to complete.

Programming Narsese

You should first consider what knowledge and rules the system should eventually have. Define these rules in Narsese. Work incrementally, start simple, and continually test your rules. For instance, if you define the transitivity rule above, then you should test it out with some concrete instances. Next, you could define the non-symmetry rule `<<(*, smaller, $x, $y)> <=> (--, <(*, smaller, $y, $x)>)>.`, test that out, and then test the combination.

Document your code extremely well. Say what you're doing and why. Record the time steps at which you're putting things into the system, and the timesteps (and truth values) of answers to questions and actions. It may be difficult to get NARS to do very complicated, many-step, many-variable inferences. It may help to break things up into more, smaller inference chains. If you can't get NARS to do something, report what you did, what you expected, what really happened and what you think the problem is.

You have quite a bit of freedom in this assignment. Make sure that you use all 8 levels of NAL, as well as judgments, questions and goals. You're encouraged to

Teaching NARS

Now that you have an idea of the knowledge NARS should learn, think about how you

could teach these things if you could not directly encode them. Use as little starting knowledge as possible, and then use examples and questions to teach these concepts.

Again, you should start simple, and you don't have to take this all the way up to the most complex rules. If you can't get NARS to learn a general rule (such as transitivity above), then at least see if you can get it to generalize to some other examples. Try using shaping/chaining and the Socratic method to nudge NARS in the right direction and allow it to make smaller steps. If you can't get it to work, do the same as above: document what you tried, why/how you thought it would work, what really happened and what you think the problem is. In some cases you may want to write a separate program to generate a lot of input data for NARS to learn from (e.g. disease/symptom data for a lot of people, summation outcomes for a lot of numbers, a lot of sentences, or a lot of game enemy encounters).

Submit your solutions

Submit a zip file named "ex5_" followed by your names ".zip". Include all files that you wrote for this exercise. Depending on how you work, it may be a single text file with the Narsese code and lots of comments, or a main file (pdf/docx/txt) where you describe what you did and one or more Narsese files that you refer to. Please write your full name and kennitala into the file. Submit it in MySchool.