Project 4: Logic Programming

Paul Beggs
GitHub Link

October 8, 2025

1 Domain 1: Merit-Based Promoting

In this domain, the goal is to find people that are older (older implies more experienced), and overachievers in the corporation. It goes about carrying out this goal by combining different facts to make a choice for who would be the best promotion candidate. For example, `merit_corp_1.txt` examines who outperforms who between two people. That is, Alice is smarter than Bob, and a hard worker, but she is younger; nevertheless, she still outperforms Bob because age is not taken into consideration for performance. In the next example, `merit_corp_2.txt`, it's a bit more convoluted because we[1] have 3 people. The set-up is like this:

- Alice: Younger than Bob, smarter than Carol, and is a hard worker.
- Carol: Older than, and smarter than Bob, but less smart than Alice, and is also a hard worker.
- Bob: Older than Alice, but less smart than Carol, and is also not a hard worker.

These facts propagate into conclusions like Carol is smarter than Bob, or Alice is smarter than Bob (transitivity). These lead to more facts like Alice outperforms both Carol and Bob (because age is not a factor), but Carol is still promoted over Alice because she lacks age.

1.1 Strengths of the Domain

If we wanted to add additional factors such as some horn clause like `arrives_early`, this could be included in the promotion criteria as well. Of course, we would have to add in additional horn clause because we can only have 2 pre-conditions per horn clause, but it's well within scope to combine hard-working and arriving early into a fact that we can use with the promoting logic. It's also nice to have things like transitivity so that we can cut down on the amount of facts to add to the system. Having the system automatically propagate all interrelated relationships based on connected facts is a nice bonus.

[1] Please let me know if referring to myself as "we" should be avoided. I have been using LaTeX for math papers for so long, that I just instinctively refer to myself as "we" instead of writing "I" or "my."

1.2 Weaknesses of the Domain

At its current state, basing a promotion while only using factors such as age and performance is illogical and unfair. There are obviously many facets that go into promoting someone in a corporation. Additionally, since we only have true or false statements, we are only able to make choices based on absolute logic. This is exemplified by assuming a one-to-one function between age and experience. This is simply not the case. Also, having subjective "facts" like "Alice smarter than Bob" clearly leads to ethical problems. One person being "smarter" than another can take a myriad of forms, and only using true or false logic to determine this quality of a person can not be entirely captured with this system.

2 Domain 2: Building Computers

This domain has a lot of different rules because I wanted to explore two different situations. One where the system is bootable with all the right parts, and the other where the system has mismatched RAM types, so the build is not compatible. In the first problem file, we just go through all the components that we have: A case, motherboard, power supply unit (PSU), RAM, a CPU with no integrated graphics, and a GPU (we don't need to include a GPU if we have integrated graphics (i.e., `cpu_igpu`)). When we propagate facts, we see that since we have a case and a motherboard, we're able to install the motherboard, and since we have installed the motherboard, we can also install the CPU and the RAM. Then, we have to make sure that our RAM and motherboard is compatible, so we specify that they are to get a `memory_ready` PC. Thus, if also we have a way of properly displaying information (i.e., `display_ready`), then we are able to boot the system. For the other problem file, we do not have compatible RAM types, so the system is not compatible, and cannot be booted.

2.1 Strengths of the Domain

The negative constraint included in this domain allow for an immediate flag that can be used in an actual program like PCPartPicker. That is, we can assemble a build with all the relevant parts, but then when the system sees that we have an incompatible RAM-motherboard configuration, it could instantly flag the build. Of course, the domain would need a lot more work to get to that point, but it's possible. Being able to "install" the computer is valuable. It allows for the user to know that they are at a certain point in the build, and they are able to advance. In addition, since building a PC is mainly logic checking (like how PCPartPicker does it), I think it would be possible to fully develop a system that has every edge case nailed down when it comes to picking parts.

2.2 Weaknesses of the Domain

Not being able to specify certain brand names for the other parts, or the size of the components (e.g., Mini-ITX, Micro-ATX, etc.) is also limiting the functionality of the domain. However, as mentioned in the strengths, we could theoretically nail down every edge case to make it a closed system. Also, not having the ability to type in `has ram_gskill_ddr4` and automatically deduce that the system has RAM could also be improved upon.