AI-2 Assignment 1 – Logic Programming

Name: Michail Pettas

Course: 5DV181

Assignment: AI-2, Assignment 1 – Logic Programming

Date: 30-11-25

1 Section 1 – ASP Programs

1.1 Program 1 – Conflict-free Sets (`asp/conflict_free.lp`)

- **Intended rules:** `{ in(A) } :- arg(A).` guesses candidate extensions, while `:- in(A), in(B), att(A,B).` eliminates conflicting ones.
- **Guessing part:** the sole choice rule over `in/1`.
- **Checking part:** the integrity constraint forbidding internal attacks.

1.2 Program 2 – Admissible Sets (`asp/admissible.lp`)

- **Intended rules:** extends Program 1 with `defeated/1` plus a defence constraint so each in-argument is attacked only by arguments counter-attacked by the set.
- **Guessing part:** identical choice rule over `in/1`.
- **Checking part:** conflict-free constraint and `:- in(A), att(B,A), not defeated(B).` enforcing admissibility.

1.3 Program 3 – Stable Extensions (`asp/stable.lp`)

- **Intended rules:** builds on Program 2 and demands every outside argument is attacked by the chosen set via `:- arg(A), not in(A), not defeated(A).`
- **Guessing part:** choice rule over `in/1`.
- **Checking part:** conflict-free constraint, `defeated/1` helper, and the stability constraint.

1.4 Program 4 – Preferred Extensions (`asp/preferred.lp`)

- **Intended rules:** reuses the admissible core but incorporates a saturation-based maximality test (adapted from ASPARTIX's `prefex_gringo.lp`). A secondary guess `inN/1` hypothesizes a strict admissible superset; if such a witness survives, the candidate is rejected.
- **Guessing part:** admissible choice over `in/1`.
- **Checking part:** admissibility constraints plus the saturation block ensuring `spoil` triggers whenever a larger admissible set exists.
- **Execution note:** run `clingo asp/preferred.lp <framework>.lp` to enumerate preferred extensions without Lua post-processing.

2 Section 2 – ASP vs. Imperative Programming

2.1 Imperative Algorithm (Program 5 – `python/preferred.py`)

- **High-level description:** performs a conflict-free backtracking search, validates admissibility at leaves, then filters inclusion-maximal sets to obtain preferred extensions.
- **Key steps:**
  1. Sort arguments by (out+in) degree to guide branching.
  2. Recursively include/exclude arguments while preserving conflict freedom.
  3. Check admissibility after assigning all arguments.
  4. Filter maximal sets.
- **Complexity:** worst-case $O(2^n(n+m))$ with $n = |AR|$ and $m = |\text{attacks}|$; ordering heuristics provide modest pruning in sparse graphs.

2.2 Performance Comparison

- **Environment:** Windows 11 Pro, Intel i7-1185G7 @ 3.0 GHz, 16 GB RAM, Clingo 5.4.0, CPython 3.13.9.
- **Method:** `python/benchmark.py --runs 5 --instances examples/example1.lp ... example6.lp` for lecture examples and `--runs 3` for random frameworks. The helper runs both `clingo asp/preferred.lp` and `python/preferred.py --benchmark` on each instance and averages wall-clock time.
- **Notes:** timings include solver start-up; random instances come from `examples/` for reproducibility.
- **Timeout handling:** for `random_n100_*` instances, both tools are run under matching 60 s limits (Clingo via `--time-limit=60 --quiet=1`, Python via a PowerShell watchdog). Values reported as >60000 ms indicate the guard was intentionally hit.

Instance	#Args	#Attacks	ASP (ms)	Python (ms)	#Pref
example1.lp	1	1	7.25	61.06	1
example2.lp	2	2	6.44	56.96	2
example3.lp	3	2	6.95	57.85	2
example4.lp	4	3	6.74	54.75	2
example5.lp	3	3	8.90	57.11	1
example6.lp	5	6	6.97	54.14	2
random_n10_a05_seed1.lp	10	5	13.12	62.18	1
random_n10_a10_seed2.lp	10	10	7.20	58.34	1
random_n10_a20_seed3.lp	10	20	7.22	54.94	1
random_n20_d025_seed7.lp	20	112	7.75	56.45	1
random_n100_a10_seed4.lp	100	10	>60000	>60000	–
random_n100_a20_seed5.lp	100	20	>60000	>60000	–
random_n100_a100_seed6.lp	100	100	>60000	>60000	–
random_n100_a150_seed7.lp	100	150	>60000	>60000	–

2.3 Discussion

- **Relative speed:** Clingo solves each finished benchmark in ≤ 13 ms, whereas Python stays between 54–62 ms principally due to interpreter start-up and a naive backtracking search.
- **Impact of density:** Clingo's propagation keeps timings nearly flat across 5–112 attacks. The Python implementation experiences little variation because it still explores much of the search tree.
- **Scalability limits:** Past roughly 20 arguments the Python solver becomes the bottleneck. For the required `random_n100_*` cases we deliberately enforced a strict 60 s budget on *both* solvers to keep experiments manageable. Clingo therefore terminates via `--time-limit=60` after generating 0.84–1.4 million partial models, and the Python DFS is killed by a matching PowerShell watchdog. The >60000 entries in Table 1 explicitly signal the budget was hit rather than accidental crashes, documenting the present scalability ceiling.
- **Developer effort:** Declarative encodings stayed short and modular, whereas the Python solver required more parsing, bookkeeping, and benchmarking infrastructure.
- **Tooling insights:** `python/preferred.py --show-admissible` was useful for sanity checks, and `python/benchmark.py` guarantees reproducible comparisons under identical conditions.

2.4 Future Work

- Add memoisation, symmetry breaking, or multiprocessing to the Python solver.
- Experiment with alternative ASP maximality encodings or solver configuration.
- Automate long-run experiments with watchdog scripts so timeouts are recorded consistently.

Appendix

Timeout Command Logs

```
$ clingo --time-limit=60 --quiet=1 asp/preferred.lp examples/random_n100_a10_seed4.lp
Answer: 1407653 ...
TIME LIMIT   : 1
Models       : 1407653+
Time         : 60.006s (Solving: 60.00s 1st Model: 0.00s Unsat: 0.00s)

$ clingo --time-limit=60 --quiet=1 asp/preferred.lp examples/random_n100_a20_seed5.lp
Answer: 1341699 ...
TIME LIMIT   : 1
Models       : 1341699+
Time         : 60.011s (Solving: 60.01s 1st Model: 0.00s Unsat: 0.00s)

$ clingo --time-limit=60 --quiet=1 asp/preferred.lp examples/random_n100_a100_seed6.lp
Answer: 1010729 ...
TIME LIMIT   : 1
Models       : 1010729+
Time         : 60.002s (Solving: 60.00s 1st Model: 0.00s Unsat: 0.00s)

$ clingo --time-limit=60 --quiet=1 asp/preferred.lp examples/random_n100_a150_seed7.lp
Answer: 841655 ...
TIME LIMIT   : 1
Models       : 841655+
Time         : 60.006s (Solving: 60.00s 1st Model: 0.00s Unsat: 0.00s)

$ pwsh -Command "cd c:/Users/vavyl/Desktop/Methods; $job = Start-Job {... python/preferred.py
Python solver timed out after 60s on random_n100_a10_seed4.lp
```

(Repeat analogous logs for `random_n100_a20_seed5.lp`, `random_n100_a100_seed6.lp`, and `random_n100_a150_s`