Lab 7
Abstract Interpretation: Numerical Abstract Domains

Objective

- Write an abstract interpreter for the Mu language in Python.
- Implement classical finite abstract domains, and some infinite ones.

This lab is adapted from a lab by Pierre Roux, in the particular setting of Mu and PYTHON/ANTLR4.

No code is provided in the git. Pull for some examples.

Milestones for the 3 sessions

- Session 1: Operational code infrastructure, obtained by adapting code from previous lab, Makefile, and basic command line. Exercise 1 and 2 (augmenting the grammar and concrete evaluator with random and assert expressions). Adequating test files for the sign abstract domain. Starting the implementation of the sign abstract domain, in parallel with the abstract evaluator (Exercise 3).
- Session 2: Finishing the abstract evaluator and the sign abstract (Exercise 3), implementing the constant abstract domain (Exercise 4). Basic tests. Adapt the command line.
- Session 3: Interval lattice (Exercise 5 and 6), and automatic test infrastructure. Bonuses if everything else is working.

Your work is due on Thursday the 14th of December before midnight. We expect your archive name to be of the form `SurnameName.tgz` and the archive to extract itself in a new directory `SurnameName/`.
Your folder must contain your code, tests, and a Readme that explains what is implemented, the usage, your tests, what are the known bugs, how to solve them if you know, or any other interesting *factual* points that may help understanding you work.

7.1 Abstract Analyser for Mu for constant propagation

You are on your own. Based on what we did before, you have to implement an abstract interpreter for the Mu language.
Minimal printing for your evaluators are logs (concrete value or abstract domain), assert (a given property). You can also choose to print the invariant before every while loop and at the end of the program, similarly to what is done here : `http://pagai.forge.imag.fr/`

EXERCISE #1 ▶ **Language extension**
Extend the grammar with random and assert expressions, extend your mu concrete evaluator from Lab3 and test.

$$\begin{array}{lcll} expr & := & \ldots & | \quad \text{rand}(e1, e2) \\ stmt & := & \ldots & | \quad \text{assert}(b) \end{array}$$

EXERCISE #2 ▶ **Generic Static Analyser**
Implement a static analyser, test infrastructure for the *constant* abstract domain (see later for a lattice description). You may choose to do it using visitor or CFG. Try to be as generic as you can in your analyser since you will have to change abstract domains from the command line.

You will have to implement the following abstract domains: signs, constants, intervals (in bonus, polyhedra or the congruence abstract domain - see exercise sheet). For assert, the analyser should print `True` or `I do not know` if it is not capable of proving the assertion.

7.2 Finite height Abstract Domains

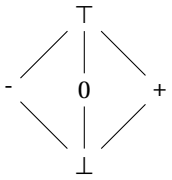
A cheat sheet about abstract domains can be found at the address:

`http://perso.ens-lyon.fr/pierre.roux/vas_2013_2014/rappels_domaines_abstraits.pdf`

(talk to your TA if you need help in reading the french there).

7.2.1 Signs

This domain makes it possible to find variables which are strictly positive or strictly negative, or zero, hence allowing to guarantee the correctness of more divisions.

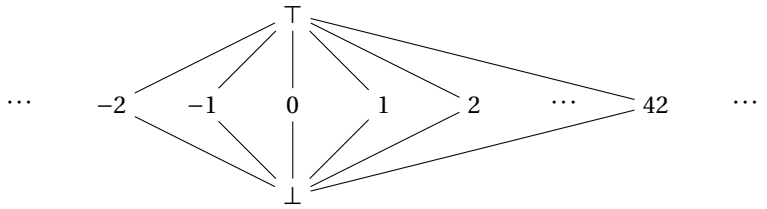


$$\begin{array}{lcl}\gamma(\top) & = & \mathbb{Z} \\ \gamma(+) & = & \{n \in \mathbb{Z} \mid n > 0\} \\ \gamma(-) & = & \{n \in \mathbb{Z} \mid n < 0\} \\ \gamma(0) & = & \{0\} \\ \gamma(\bot) & = & \emptyset\end{array}$$

EXERCISE #3 ▶ **Signs**
Implement this domain and see what happens on well-chosen examples.

7.2.2 Kildall

This domain makes it possible to find variables which are constants at a certain point in the program. It can also be used to simplify programs in a compiler.

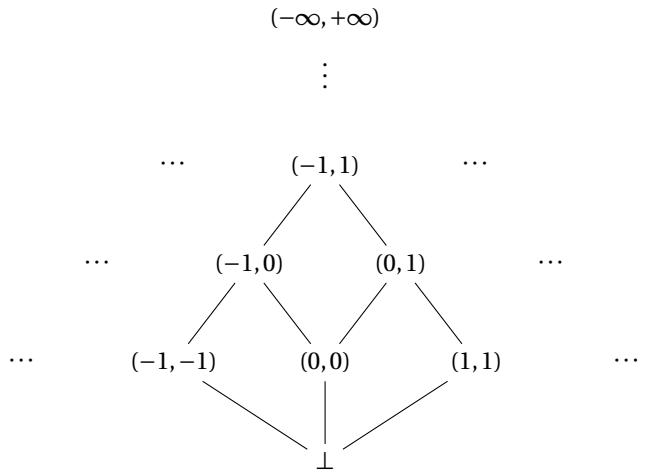


$$\begin{array}{lcl}\gamma(\top) & = & \mathbb{Z} \\ \gamma(n) & = & \{n\} \\ \gamma(\bot) & = & \emptyset\end{array}$$

EXERCISE #4 ▶ **Kildall**
Implement this domain and see what happens for the example `ex/ex08.mu`.

7.3 Infinite Height: intervals

In this section, we wish to implement a domain of intervals, where variables are interpreted by the range of values they can take.
The lattice is $(\mathscr{D}^\sharp, \sqsubseteq^\sharp)$ with $\mathscr{D}^\sharp = \bot \cup \{(n_1, n_2) \in (\mathbb{Z} \cup \{-\infty\}) \times (\mathbb{Z} \cup \{+\infty\}) \mid n_1 \leq n_2\}$.



$$\begin{array}{lcl}\gamma(-\infty, +\infty) & = &]\!]-\infty, +\infty[\![\\ \gamma(-\infty, n) & = &]\!]-\infty, n]\!] \\ \gamma(n, +\infty) & = & [\![n, +\infty[\![\\ \gamma(n_1, n_2) & = & [\![n_1, n_2]\!] \\ \gamma(\bot) & = & \emptyset\end{array}$$

Reminder: a widening operator can be used to accelerate the convergence of the fixpoint calculation. The idea is to extrapolate in the computation, so that we reach a result without going upwards ad infinitum in a lattice of unbounded height:

$$x^\sharp \triangledown y^\sharp = \begin{cases} [\![a,b]\!] & \text{if } x^\sharp = [\![a,b]\!], y^\sharp = [\![c,d]\!], c \geq a, d \leq b \\ [\![a,+\infty[\![& \text{if } x^\sharp = [\![a,b]\!], y^\sharp = [\![c,d]\!], c \geq a, d > b \\ ]\!-\infty,b]\!] & \text{if } x^\sharp = [\![a,b]\!], y^\sharp = [\![c,d]\!], c < a, d \leq b \\ ]\!-\infty,+\infty[\![& \text{if } x^\sharp = [\![a,b]\!], y^\sharp = [\![c,d]\!], c < a, d > b \\ y^\sharp & \text{if } x^\sharp = \bot \\ x^\sharp & \text{if } y^\sharp = \bot \end{cases}$$

EXERCISE #5 ▶ **Intervals**
Implement this domain. Firstly, implement without widening, then test, then implement widening. You will have to change your generic analyser to apply widening at loop heads. Test on well-chosen examples.

EXERCISE #6 ▶ **Descending sequence, widening delay**
On the following program:

```
i = 0; j = 0;
while (i < 10) {
  if (i <= 0) {
    j = 1;
    ++i;
  } else {
    ++i; } }
```

What interval does one get for variable `j`? First try to improve by using a descending sequence, then by a widening delay. Augment the command line of your tool.

7.4 Applications

EXERCISE #7 ▶ **Printing**
For every print statement, make the analyzer print the abstract value of the expression given to the print statement.

EXERCISE #8 ▶ **Division by zero**
Make your analyzer find divisions by zero.

There are two possible modes:

- either make it detect all cases when there is a risk of a division by zero (eg. `5 / random(-5, 5)`), or
- only cases when you are sure there is a division by zero.

7.5 Optional extensions

Only do this section if you have time. Exercises below are in ascending order of difficulty.

EXERCISE #9 ▶ **Arrays**
Add arrays to the Mu language. Describe their semantics and operations on the arrays on paper.

Add support for checking out of bound access to arrays (negative and overflow).

Add basic support for abstract values in the arrays. The easiest way to do it is to join (ie. make the union) abstract values of all variables in the array to a single value.

EXERCISE #10 ▶ **More types**
Add support for booleans, to refine what your analyzer infers when a condition is run.

Add support for floats, in addition to integers.

EXERCISE #11 ▶ **Polyhedra**
Add a new abstract domain, that uses a convex polyhedron as the type of an abstract value. You may use a third-party library if its license allows it.