Manual for Abstract Interpreter Project

Goktug Saatcioglu, NYU Computer Science

05.10.2019

This is the documentation for the abstract interpret project for Prof. Cousot's Abstract Interpretation course. All project files are written in OCaml and required libraries are described later on in the document.

The project is split into two folders: `domains` and `src`. The folder `src` contains two folders where the first is called `no-ptr`. This folder contains the implementation of a lexer, parser, abstract syntax tree, fixpoint calculator and a generic abstract interpreter for the language without pointers. Similarly, the second folder `ptr` includes the same implementations of `no-ptr` but extended to the pointer language. So, abstract domains that do not deal with pointers will be used with the contents of `no-ptr` and abstract domains that also have pointers will be used with the contents of `ptr`.
The folder `domains` includes the implementation of several abstract domains. They are as follows (listed in the format "`folder-name`: implemented domain"):

- `bottop`: a simple abstract domain with the elements bot and top (only used for testing purposes),
- `congruence`: Cartesian congruence domain (fb),
- `constancy`: Cartesian constancy domain (fb),
- `interval`: Cartesian interval domain (fb),
- `octagon-apron`: octagon domain using Apron library (f),
- `parity`: Cartesian parity domain (fb),
- `poly-elina`: polyhedral domain using Elina library (f),
- `potential-points-to`: flow-sensitive Cartesian potential points-to domain (f),
- `potential-points-to-interval`: combination of potential points-to and intervals (fb),
- `red-prod-congruences-intervals`: reduced product of congruences and intervals (fb),
- `red-prod-parity-signs`: reduced product of parity and signs (fb),
- `sign`: Cartesian sign domain (fb),
- `top`: a simple abstract domain where everything is top (only used for testing purposes).

Here (f) stands for forward analysis and (fb) stands for forward-backward analysis and indicate which of the analyses an implementation supports. Furthermore, each implementation comes with a widening and narrowing operator which can be trivial/non-trivial depending on the domain.

There is a `makefile` that can be used to compile and run the domains. The `makefile` works by accessing one of the folders inside `domains` and creating a symbolic link to the relevant files depending on what the analysis is (so which domain you wish to use). Similarly, depending on the context, the makefile accesses one of the folders inside `src` and create the relevant symbolic links. Then it proceeds to compile the program using `ocamlc` and outputs an executable named `main`. The following commands can be used with the terminal command `make`.

- `all`: run all domains

- `analyseTop`: compile for top domain
- `analyseBotTop`: compile for bot-top domain
- `analyseParity`: compile for parity domain
- `analyseConstancy`: compile for constancy domain
- `analyseSign`: compile for sign domain
- `analyseCongruence`: compile for congruence domain
- `analyseOctagonApron`: compile for octagon domain with Apron library
- `analysePolyElina`: compile for polyehdral domain with Elina library
- `analyseReducedSignParity`: compile for reduced product of signs and parity
- `analyseReducedCongruenceInterval`: compile for reduced product of congruences and intervals
- `analysePotentialPointsTo`: compile for potential points-to domain
- `analysePotentialPointsToInterval`: compile for combination of potential points-to and intervals

So, the format for compilation is, for example, `make analyseParity` (which compiles for the parity analysis). Furthermore, the following utility commands are available.

- `help`: print all available compilation commands
- `delete`: delete all compilation files including `main`
- `clean`: delete all compilation files except the symbolic links and `main`
- `removeSys`: delete all symbolic links
- `removeMain`: delete the created `main` executable

Upon compiling the interpreter will run a few example programs and print out the results of the analysis. Please note that there may be some formatting issues when printing and resizing of the terminal window may be required. Any printer file can be modified by navigating to the `domains` folder and selecting from the available folders the file that begins with `printer`. Changing the variable `labelmargin` could be useful to change the behavior of the printer if desired. Finally, to analyze more programs use the following command in the terminal after compiling a domain.

- `echo "your_program" | ./main`: where the string `"your_program"` is a valid program whose syntax is as defined in the lecture notes (can be no pointer or pointer language depending on which domain is compiled)

The program has been written using version 4.06 of OCaml and tested on macOS Mojave. Furthermore, if using the Apron or Elina library the following libraries are needed: `apron`, `camlidl`, `conf-gmp`, `conf-m4`, `conf-mpfr`, `conf-ppl`, `elina`, `mlgmpidl`, and `ocamlfind`. All libraries can be installed using the `opam` package manager. Furthermore, you will need the `gcc` compiler for ANSI C, the `GMP` C library, the GNU m4 library and the `MPFR` C library. These can all be installed by installing XCode and/or using `brew`.

Please note that while the programs have been tested and debugged there may still be minor issues with the implementation logic. Please report all bugs to `gs2417@nyu.edu` so that they may be fixed. Furthermore, the implementation of the combination of the potential points-to and intervals may contain bigger issues due to the author's doubts when implementing this domain. Again, please report any issues to the above given email address. Also, note that all analyses should compile and run as they have been all tested on whether there are any compilation issues.

The initial files of `abstractDomain.mli`, `abstractSyntax.ml`, `abstractSyntaxExpressions.ml`, `abstractTree.ml`, `abstractTree.mli`, `lexer.mll`, `parser.mly`, `abstractDomainParity.ml` and `printerParity.ml` in the folders `src/no-ptr` and `domains/parity` have been provided by Prof. Cousot. All other files including changes to the aforementioned files have been implemented by Goktug Saatcioglu.