# Compilation

## TP 9: Abstract Interpretation

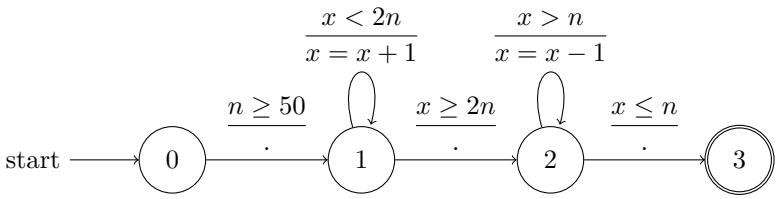
A. Isoard & C. Alias

Program analysis offers new tools to the programmer and to the compiler. Knowing properties on the program at compile time allow to do more complex code transformations, to predict more precisely the performance of the program and even verify critical properties over the program. In this TP we will study a powerful program analysis named: Abstract Interpretation.

This analysis is done in three steps: (i) transform the program into an automaton, (ii) associate to each transition an abstract interpretation and (iii) compute for each state the abstract domain of every possible memory state by following the abstract interpretation of the program (fix point computation).

```
var n:int, x:int;
begin
  assume n >= 50;
  while (x < 2*n) do
    x = x+1;
  done;
  while (x > n) do
    x = x-1;
  done;
end
```

(a) Pseudo code




(b) Automaton

Figure 1: Toy example

### Exercise 1. *Abstracting with Intervals*

This is the abstract interpretation seen in the lecture. For each instruction in the program we associate to each variable an interval representing the range of its possible values.

**Step by step.**

**State 0** How many variable(s) do we have? Give their initial abstract domains.

**State 1** What is the new domain of `n`? Can we deduce something on `x`? Why?

**State 2** What is the new domain of `x`? And after one loop? And after widening? And after one loop again (narrowing)?

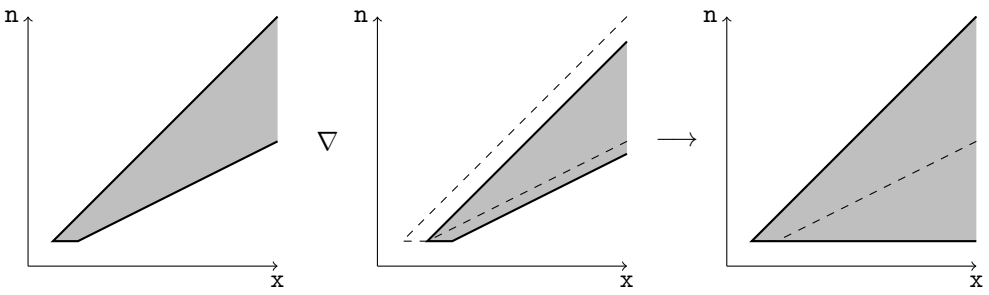
**State 3** What is the final domain of `x`?

### Exercise 2. *Abstracting with Polyhedrons*

To improve the precision of the analysis, we want to allow *inter-variables constraints*. In order to be able to arbitrarily compose abstract representations we restrict ourself to *affine constraints*, as this keep everything *computable* (with gauss eliminations). In our case we will only allow *conjunctions of affine inequalities*, thus an abstract space can be viewed as a *convex polyhedron* in a $n$-dimensional space where $n$ is the number of variables.

Union will be implemented using $\sqcup$ the *convex union* over polyhedrons. The figure **??** illustrate the widening operator. The idea is: $P \nabla Q$ is composed of the constraints of $P \sqcup Q$ that were already enforced by $P$.

As a side note, the previous abstraction was a particular case of this one, where the only allowed polyhedron were axis aligned rectangles.




$n \leq x$
$50 \leq n$
$x \leq 2n$

$n \leq x - 1$
$50 \leq n$
$x - 1 \leq 2n$

$n \leq x$
$50 \leq n$

Figure 2: Widening

**Questions.**

1. Describe in two different ways (with inequalities only) a non empty polyhedron. (clue: try a flat polyhedron)
2. Show that the widening operator depends on the representation.
3. (Bonus) Try to define a widening operator that does not depends on the representation, by using the generator representation: with vertices and rays instead of constraints.

**Step by step.**

**State 0** How many variable(s) do we have? Give the initial abstract domain.

**State 1** What is the new domain? Can we deduce something on `x`? Why?

**State 2** What is the new domain? And after one loop? And after widening? And after one loop again (narrowing)?

**State 3** What is the final domain?

## Exercise 3. *Real tools*

Abstract interpretation is actually used to verify properties over real programs, like absence of integer overflow, detection of aliasing, prediction of worst case execution time, ... A lot of different tools implement some kind of abstract interpretation. Lets try one of them.

**Manip.**

Try the code of figure **??** on the online Interproc analyser.[1]

Experiment with different abstract domains: *box* is our interval abstraction, *convex polyhedra* is our polyhedral abstraction. Try to guess (and verify) what *octagon* is.

Iteration/Widening options allow you to perform a certain number of standard iteration before resorting to widening, and allow you to specify the number of iteration to do during narrowing.

Try the other examples or custom made ones. Have fun!

**Interested? Want more?** Contact Laure Gonnord at `firstname.lastname@ens-lyon.fr` for internship/Ph.D opportunities! By the way, an other tool, Pagai[2] is capable of analysing C code (based on clang/llvm).

[1] online demonstrator `http://pop-art.inrialpes.fr/interproc/interprocweb.cgi`
[2] sources available at `http://pagai.forge.imag.fr/`