Program analysis

▶ in CAP, we discuss programs manipulating programs
compute something with a program as an input
▶ another program
▶ a property of the program what it does (not)
→ accept/reject, transform the initial program

▶ we shall focus on smaller scale languages
1. small imperative language: IMP
   1.1 Abstract Interpretation (automatic, the program is the only input)
   1.2 Hoare triples (interaction with the user)
2. small functional language: Fun
   2.1 type inference
   2.2 abstract machines and compilation
   2.3 intermediate representations

▶ in breadth rather than in depth
▶ few proofs (see references on the www page)
▶ prerequisites: order theory, semantics

Abstract Interpretation

With material from the courses by A. Miné and P. Roux

1. The method

Analysing programs

▶ typical questions we want to ask / bugs we want to avoid
  x = a/b make sure b ≠ 0
  x = t[i] make sure i is within the bounds of t
  i = i+1 make sure there is no overflow

▶ Abstract Interpretation can also be used to perform more refined analyses

Know everything about all possible runs of the program

▶ annotate nodes of (some kind of) Control Flow Graphs with labels ℓ ∈ L
▶ during execution, a program state (ℓ, σ) consists of
  a control state ℓ ∈ L and
  an environment (memory state) σ ∈ V → Z

▶ concrete semantics (meaning) of the program
  write a recursive equation involving sets of environments
  we are interested in the least fixpoint of some operator acting on \( P(V → Z) \)
  this fixpoint yields a function of type \( L → P(V → Z) \)
  associating a set of possible stores (memory states) to every label in the program

the least fixpoint exists (Knaster-Tarski’s theorem)

. . . but there is no hope of computing it (either impossible/undecidable or too costly)

Computing an abstraction

"I took a speed reading course and read War and Peace in twenty minutes.
It involves Russia."
Let's get rough

- instead of computing the concrete semantics, compute an abstract semantics
- be less precise, and more computable
- scale down our ambitions, and strike a balance
- rough but sound

the abstract semantics contains the concrete semantics

some examples of abstractions
demo-signs.pdf demo-cstes.pdf

The general method: $D$ and $D^\#$, via $\gamma$

the concretisation function $\gamma : D^\# \rightarrow D$

- $\gamma$ should be monotone
- $a \in D^\#$ is a sound abstraction of $c \in D$ if $c \subseteq \gamma(a)$
- $g : D^\# \rightarrow D^\#$ is a sound abstraction of $f : D \rightarrow D$ if $\forall a \in D^\#, (f \circ \gamma)(a) \subseteq (\gamma \circ g)(a)$

move from the concrete semantics to the abstract semantics:
from $R_i = \bigcup_{(\ell, c) \in A} [\ell] R_j$ to $\sigma^\#_i = \bigcup_{(\ell, c) \in A} [\ell]^\# \sigma^\#_j$

- $\sigma^\#_i$, $\sigma^\#_j$: abstract environments
- $[\cdot]^\#$: abstract transfer function

Insuring that an answer is provided

we want effective computations

- everything should be computable in $D^\#$
  - representation of elements of $D^\#$
  - $\bigcup$, $\bigcap$, $\ldots$
- computing the abstract semantics
  - computing $\sigma^\#_i$
    - relies on the definition of abstract operators $+\#$, $-\#$, $\ldots$
  - computing the least fixpoint
    - Kleene iterations $\perp, F(\perp), F(F(\perp)), \ldots$
    - a finite number of them: stabilisation
      - ok if the lattice is of finite height
      - otherwise

TP next week

- you will be given a program that computes the abstract semantics according to a given value abstract domain
- you will define several value abstract domains, and see how the analysis of programs is affected
  - you might want to write down equations before coding $\subseteq^\# +^\# -^\# \ldots$
- all this in OCaml
  - you don’t need to be an expert OCaml programmer
  - basically, define (simple) types, and (simple) functions acting on such types
- install OCaml on your laptop

2. How it works
(and why — a glance at the mathematical justification)

on the board

The answers of Abstract Interpretation

Theorem (Soundness): $\forall \ell \in L, R_\ell \subseteq \gamma(a)\#$

because we use sound operators ($\bigcup^\#, +^\#, \ldots$) in $D^\#$, we keep over-approximating when computing the abstract semantics

cf. talking with toddlers

Abstract Interpretation: compute the abstract semantics, and check the required condition

- if the answer is “ok”, then it is “ok”
  - for example, $0$ is not among the possible values for $X$ at that point in the program
- if the answer is “no”, then work needs to be done

Widening

- the analysis must be able to answer in reasonable time
- in some cases, the abstract domain $D^\#$ is of unbounded height
  to guarantee convergence of the computation of the least fixpoint, we use a widening operator $\nabla : D^\# \times D^\# \rightarrow D^\#$ satisfying:
  - $\forall x^\#, y^\#, x^\# \nabla y^\# = x^\# [\nabla] x^\# \nabla y^\#$, soundness
  - for any sequence $(y_0^\#)_{i=0}^\infty$
    - the sequence $x_0^\# = y_0^\#$, $x_{i+1}^\# = x_i^\# \nabla y_{i+1}^\#$ satisfies $\exists n. x_n^\# = x_0^\#$, stabilisation
  - $\nabla$ “extrapolates”

for some nodes $\ell$ belonging to cycles in the CFG

$\nabla$ convergence of the iteration

- a narrowing operator can also be used to make the analysis more precise after applying widening

Further notions in Abstract Interpretation
References

course by Pierre Roux
AI in 3 lessons

course by Antoine Miné (and others)

much more detailed and in depth (M2)

see links from the course webpage

many thanks to Antoine and Pierre for allowing me to use their material

a peculiarity in terminology:

. prefixpoint \( f(x) \sqsubseteq x \)
. postfixpoint \( x \sqsubseteq f(x) \)

as seen, e.g., in L3IF

. . . they use the converse

Relational abstract domains

the concrete semantics is given by a function (which is difficult to compute) in \( L \rightarrow P(V \rightarrow \mathbb{Z}) \)

associating a set of possible memory states to every label in the program

we have described non relational analyses

\( P(V \rightarrow \mathbb{Z}) \) is abstracted into \( V \rightarrow P(\mathbb{Z}) \),
and then \( P(\mathbb{Z}) \) is abstracted into some \( D^\dagger \)

a relational abstract domain is some \( D^\dagger \) which is an abstraction of \( P(V \rightarrow \mathbb{Z}) \)

express that certain combinations of \( x \) and \( y \) are impossible (polyhedra, octagons)

Galois connections

\( \alpha \): monotone abstraction function

\[
\begin{align*}
(\mathcal{D}, \sqsubseteq) & \xrightarrow{\gamma} (\mathcal{D}^\dagger, \sqsubseteq^\dagger) \\
\alpha(x) \sqsubseteq^\dagger y^\dagger & \iff x \sqsubseteq \gamma(y^\dagger)
\end{align*}
\]

any \( x \in \mathcal{D} \) has a best abstraction \( \alpha(x) \)