What is static analysis by abstract interpretation?

Example of static analysis (input)

```
n := n0;
i := n;
while (i <> 0) do
    j := 0;
    while (j <> i) do
        j := j + 1
    od;
i := i - 1
od
```

Example of static analysis (output)

```
{n0>0}
n := n0;
{n0=n,n0>=0}
i := n;
{n0=1,n0=n,n0>=0}
while (i <> 0) do
    {n0=n,i>=1,n0>=i}
    j := 0;
    {n0=n,j=0,i>=1,n0>=i}
    while (j <> i) do
        {n0=n,j>=0,i>=j+1,n0>=i}
        j := j + 1
    od;
    {n0=n,j>=1,i>=j,n0>=i}
    {i+1=j,n0=n,i>=0,n0>=i+1}
i := i - 1
od
{n0=n,i=0,n0>=0}
```
Example of static analysis

**Verification**: absence of runtime errors;

**Abstraction**: polyhedral abstraction (affine inequalities);

**Theory**: abstract interpretation.

Potential impact of runtime errors

- 50% of the security attacks on computer systems are through buffer overruns\(^1\)!
- Embedded computer system crashes easily result from overflows of various kinds.

\(^1\) See for example the Microsoft Security Bulletins MS02-065, MS04-011, etc.
A very informal introduction to the principles of abstract interpretation

Semantics

The *concrete semantics* of a program formalizes (is a mathematical model of) the set of all its possible executions in all possible execution environments.

Graphic example: Possible behaviors

Undecidability

- The concrete mathematical semantics of a program is an “infinite” mathematical object, *not computable*;
- All non trivial questions on the concrete program semantics are *undecidable*.

Example: Kurt Gödel argument on termination
- Assume termination($P$) would always terminates and returns true iff $P$ always terminates on all input data;
- The following program yields a contradiction
  \[ P \equiv \text{while termination}(P) \text{ do skip od} \]
The *safety properties* of a program express that no possible execution in any possible execution environment can reach an erroneous state.

### Safety proofs

- A **safety proof** consists in proving that the intersection of the program concrete semantics and the forbidden zone is empty;
- **Undecidable** problem (the concrete semantics is not computable);
- Impossible to provide completely automatic answers with finite computer resources and neither human interaction nor uncertainty on the answer.\(^2\)

---

\(^2\) e.g. probabilistic answer.
Abstract interpretation

- consists in considering an *abstract semantics*, that is to say a superset of the concrete semantics of the program;
- hence the abstract semantics covers all possible concrete cases;
- correct: if the abstract semantics is safe (does not intersect the forbidden zone) then so is the concrete semantics.

Formal methods

Formal methods are abstract interpretations, which differ in the way to obtain the abstract semantics:
- "*model checking*":
  - the abstract semantics is given manually by the user;
  - in the form of a finitary model of the program execution;
  - can be computed automatically, by techniques relevant to static analysis.
“deductive methods”:
- the abstract semantics is specified by verification conditions;
- the user must provide the abstract semantics in the form of inductive arguments (e.g. invariants);
- can be computed automatically by methods relevant to static analysis.

“static analysis”: the abstract semantics is computed automatically from the program text according to pre-defined abstractions (that can sometimes be tailored automatically/manually by the user).

Required properties of the abstract semantics
- sound so that no possible error can be forgotten;
- precise enough (to avoid false alarms);
- as simple/abstract as possible (to avoid combinatorial explosion phenomena).

Graphic example: Erroneous abstraction — I

Graphic example: Erroneous abstraction — II
Abstract domains

**Standard abstractions**
- that serve as a *basis* for the design of static analyzers:
  - abstract program data,
  - abstract program basic operations;
  - abstract program control (iteration, procedure, concurrency, ...);
- can be *parametrized* to allow for manual adaptation to the application domains.

**Graphic example: Standard abstraction by intervals**

**Graphic example: A more refined abstraction**
A very informal introduction to static analysis algorithms

Trace semantics

- Consider (possibly infinite) traces that is series of states corresponding to executions described by discrete transitions;
- The collection of all such traces, starting from the initial states, is the trace semantics.

Graphic example: Small-steps transition semantics
Trace semantics, intuition

Prefix trace semantics

Prefixes of a finite trace

Prefixes of an infinite trace
Prefix trace semantics

Trace semantics: maximal finite and infinite behaviors
Prefix trace semantics: finite prefixes of the maximal behaviors

Abstraction

This is an abstraction. For example:

- Trace semantics: \( \{a^n b \mid n \geq 0\} \)
- Prefix trace semantics: \( \{a^n \mid n \geq 0\} \cup \{a^n b \mid n \geq 0\} \)

Is there of possible behavior with infinitely many successive \( a \)?
- Trace semantics: no
- Prefix trace semantics: I don’t know

Least Fixpoint Prefix Trace Semantics

\[
\text{Prefixes} = \{\bullet \mid \bullet \text{ is an initial state}\} \\
\cup \{\ldots \bullet \ldots \bullet \in \text{Prefixes} \mid \bullet \ldots \bullet \in \text{Prefixes} \}
\]

- In general, the equation \( \text{Prefixes} = F(\text{Prefixes}) \) may have multiple solutions;
- Choose the least one for subset inclusion \( \subseteq \).
- Abstractions of this equation lead to effective iterative analysis algorithms.
Collecting semantics

- Collect all states that can appear on some trace at any given discrete time:

Collecting abstraction

- This is an abstraction. Does the red trace exists?
  Trace semantics: no, collecting semantics: I don’t know.

Graphic example: collecting semantics
Collecting semantics in fixpoint form
Graphic example: collecting semantics in fixpoint form
Graphic example: collecting semantics in fixpoint form

Interval Abstraction (in iterative fixpoint form)
Graphic example: traces of intervals in fixpoint form
Graphic example: traces of intervals in fixpoint form

Possible discrete trajectories

---

Graphic example: traces of intervals in fixpoint form

Possible discrete trajectories

Graphic example: traces of intervals in fixpoint form

Abstraction by Galois connections
Abstracting sets (i.e. properties)

- Choose an abstract domain, replacing sets of objects (states, traces, …) $S$ by their abstraction $\alpha(S)$
- The abstraction function $\alpha$ maps a set of concrete objects to its abstract interpretation;
- The inverse concretization function $\gamma$ maps an abstract set of objects to concrete ones;
- Forget no concrete objects: (abstraction from above) $S \subseteq \gamma(\alpha(S))$.

Interval abstraction $\alpha$

\[ \alpha(\{x : [1, 99], y : [2, 77]\}) \]

Interval concretization $\gamma$

\[ \gamma(\{x : [1, 99], y : [2, 77]\}) \]

The abstraction $\alpha$ is monotone

\[ X \subseteq Y \Rightarrow \alpha(X) \subseteq \alpha(Y) \]
The concretization $\gamma$ is monotone

\[ \{ x : [33, 89], y : [48, 61] \} \subseteq \{ x : [1, 99], y : [2, 90] \} \]

\[ X \subseteq Y \Rightarrow \gamma(X) \subseteq \gamma(Y) \]

The $\gamma \circ \alpha$ composition is extensive

\[ \{ x : [1, 99], y : [2, 77] \} \]

The $\alpha \circ \gamma$ composition is reductive

\[ \{ x : [1, 99], y : [2, 77] \} \]

Correspondance between concrete and abstract properties

- The pair $\langle \alpha, \gamma \rangle$ is a Galois connection:
  \[ \langle \wp(S), \subseteq \rangle \xrightarrow{\gamma} \langle D, \subseteq \rangle \]

- $\langle \wp(S), \subseteq \rangle \xrightarrow{\alpha} \langle D, \subseteq \rangle$ when $\alpha$ is onto (equivalently $\alpha \circ \gamma = 1$ or $\gamma$ is one-to-one).

\[ \alpha \circ \gamma(Y) = \subseteq Y \]
Galois connection

\[ (\mathcal{D}, \subseteq) \xrightarrow{\gamma} (\overline{\mathcal{D}}, \subseteq) \]

iff

\[ \forall x, y \in \mathcal{D} : x \subseteq y \implies \alpha(x) \subseteq \alpha(y) \]
\[ \land \forall x, \overline{y} \in \overline{\mathcal{D}} : \overline{x} \subseteq \overline{y} \implies \gamma(x) \subseteq \gamma(y) \]
\[ \land \forall x \in \mathcal{D} : x \subseteq \gamma(\alpha(x)) \]
\[ \land \forall \overline{y} \in \overline{\mathcal{D}} : \alpha(\gamma(\overline{y})) \subseteq \overline{x} \]

iff

\[ \forall x \in \mathcal{D}, \overline{y} \in \overline{\mathcal{D}} : \alpha(x) \subseteq \overline{y} \iff x \subseteq \gamma(\overline{y}) \]

Example: Set of traces to trace of intervals abstraction

Set of traces:
\[ \alpha_1 \downarrow \]
Trace of sets:
\[ \alpha_2 \downarrow \]
Trace of intervals

Example: Set of traces to reachable states abstraction

Set of traces:
\[ \alpha_1 \downarrow \]
Trace of sets:
\[ \alpha_3 \downarrow \]
Reachable states

Composition of Galois Connections

The composition of Galois connections:

\[ (\mathcal{L}, \leq) \xrightarrow{\gamma_1} (\mathcal{M}, \subseteq) \]
and:

\[ (\mathcal{M}, \subseteq) \xrightarrow{\gamma_2} (\mathcal{N}, \leq) \]

is a Galois connection:

\[ (\mathcal{L}, \leq) \xrightarrow{\gamma_1 \circ \gamma_2} (\mathcal{N}, \leq) \]
Convergence acceleration by widening/narrowing

Graphic example: upward iteration with widening

x(t)

Possible discrete trajectories

Initial states

Interval transition

Interval transition with widening
Graphic example: upward iteration with widening

\[ x(t) \]

Possible discrete trajectories

Interval transition with widening

Graphic example: stability of the upward iteration

\[ x(t) \]

Possible discrete trajectories

Interval widening

- \( \mathcal{L} = \{ \bot \} \cup \{ [\ell, u] \mid \ell, u \in \mathbb{Z} \cup \{-\infty\} \land u \in \mathbb{Z} \cup \{\} \land \ell \leq u \} \)
- The widening extrapolates unstable bounds to infinity:
  \[ \bot \uparrow X = X \]
  \[ X \uparrow \bot = X \]
  \[ [\ell_0, u_0] \uparrow [\ell_1, u_1] = \begin{cases} -\infty & \text{if } \ell_1 < \ell_0 \\ \ell_0 & \text{if } u_1 > u_0 \\ +\infty & \text{else } u_0 \end{cases} \]

Not monotone. For example \([0, 1] \subseteq [0, 2]\) but \([0, 1] \uparrow [0, 2] = [0, +\infty] \not\subseteq [0, 2] = [0, 2] \uparrow [0, 2]\)

Example: Interval analysis (1975)

Program to be analyzed:

\begin{verbatim}
x := 1;
1:
   while x < 10000 do
2:
      x := x + 1
3:
   od;
4:
\end{verbatim}
Example: Interval analysis (1975)

Equations (abstract interpretation of the semantics):

\[
\begin{align*}
    & X_1 = [1, 1] \\
    & X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\
    & X_3 = X_2 \oplus [1, 1] \\
    & X_4 = (X_1 \cup X_3) \cap [10000, +\infty]
\end{align*}
\]

```
x := 1;
while x < 10000 do
    x := x + 1
od;
```

Example: Interval analysis (1975)

Resolution by chaotic increasing iteration:

\[
\begin{align*}
    & X_1 = [1, 1] \\
    & X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\
    & X_3 = X_2 \oplus [1, 1] \\
    & X_4 = (X_1 \cup X_3) \cap [10000, +\infty]
\end{align*}
\]

```
x := 1;
while x < 10000 do
    x := x + 1
od;
```

Example: Interval analysis (1975)

Increasing chaotic iteration:

\[
\begin{align*}
    & X_1 = [1, 1] \\
    & X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\
    & X_3 = X_2 \oplus [1, 1] \\
    & X_4 = (X_1 \cup X_3) \cap [10000, +\infty]
\end{align*}
\]

```
x := 1;
while x < 10000 do
    x := x + 1
od;
```

Example: Interval analysis (1975)

Increasing chaotic iteration:

\[
\begin{align*}
    & X_1 = [1, 1] \\
    & X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\
    & X_3 = X_2 \oplus [1, 1] \\
    & X_4 = (X_1 \cup X_3) \cap [10000, +\infty]
\end{align*}
\]

```
x := 1;
while x < 10000 do
    x := x + 1
od;
```
Example: Interval analysis (1975)
Increasing chaotic iteration:

\[
\begin{align*}
x &:= 1; \\
\text{1: while } x < 10000 \text{ do} \\
\quad x &:= x + 1 \\
\text{2: } X_1 = [1, 1] \\
\quad X_2 = (X_1 \cup X_3) \cap [10000, +\infty] \\
\quad X_3 = X_2 \oplus [1, 1] \\
\quad X_4 = (X_1 \cup X_3) \cap [10000, +\infty]
\end{align*}
\]

Example: Interval analysis (1975)
Increasing chaotic iteration:

\[
\begin{align*}
x &:= 1; \\
\text{1: while } x < 10000 \text{ do} \\
\quad x &:= x + 1 \\
\text{2: } X_1 = [1, 1] \\
\quad X_2 = [1, 2] \\
\quad X_3 = [2, 3] \\
\quad X_4 = 0
\end{align*}
\]

Example: Interval analysis (1975)
Increasing chaotic iteration: convergence !

\[
\begin{align*}
x &:= 1; \\
\text{1: while } x < 10000 \text{ do} \\
\quad x &:= x + 1 \\
\text{2: } X_1 = [1, 1] \\
\quad X_2 = [1, 2] \\
\quad X_3 = [2, 3] \\
\quad X_4 = 0
\end{align*}
\]

Example: Interval analysis (1975)
Increasing chaotic iteration: convergence !!

\[
\begin{align*}
x &:= 1; \\
\text{1: while } x < 10000 \text{ do} \\
\quad x &:= x + 1 \\
\text{2: } X_1 = [1, 1] \\
\quad X_2 = [1, 3] \\
\quad X_3 = [2, 3] \\
\quad X_4 = 0
\end{align*}
\]
Example: Interval analysis (1975)

Increasing chaotic iteration: convergence !!!!

\[
\begin{align*}
   x &:= 1; \\
   X_1 &= [1, 1] \\
   X_2 &= (X_1 \cup X_3) \cap [-\infty, 9999] \\
   X_3 &= X_2 \oplus [1, 1] \\
   X_4 &= (X_1 \cup X_3) \cap [10000, +\infty] \\
   \end{align*}
\]

1: while \( x < 10000 \) do

2: \( x := x + 1 \)

3: \( X_3 = [2, 4] \)

4: \( X_4 = 0 \)

Example: Interval analysis (1975)

Increasing chaotic iteration: convergence !!!!!

\[
\begin{align*}
   x &:= 1; \\
   X_1 &= [1, 1] \\
   X_2 &= (X_1 \cup X_3) \cap [-\infty, 9999] \\
   X_3 &= X_2 \oplus [1, 1] \\
   X_4 &= (X_1 \cup X_3) \cap [10000, +\infty] \\
   \end{align*}
\]

1: while \( x < 10000 \) do

2: \( x := x + 1 \)

3: \( X_3 = [2, 5] \)

4: \( X_4 = 0 \)
Example: Interval analysis (1975)

Increasing chaotic iteration:

\[
\begin{align*}
\text{x} & := 1; \\
\text{while } x < 10000 \text{ do} & \\
\text{x} & := x + 1 \\
\od; & \\
\end{align*}
\]

\[
\begin{align*}
X_1 &= [1, \infty] \\
X_2 &= (X_1 \cup X_3) \cap [-\infty, 9999] \\
X_3 &= X_2 \ominus [1, 1] \\
X_4 &= (X_1 \cup X_3) \cap [10000, +\infty] \\
\end{align*}
\]

Example: Interval analysis (1975)

Convergence speed-up by widening:

\[
\begin{align*}
\text{x} & := 1; \\
\text{while } x < 10000 \text{ do} & \\
\text{x} & := x + 1 \\
\od; & \\
\end{align*}
\]

\[
\begin{align*}
X_1 &= [1, 1] \\
X_2 &= (X_1 \cup X_3) \cap [-\infty, 9999] \\
X_3 &= X_2 \ominus [1, 1] \\
X_4 &= (X_1 \cup X_3) \cap [10000, +\infty] \\
\end{align*}
\]

Example: Interval analysis (1975)

Decreasing chaotic iteration:

\[
\begin{align*}
\text{x} & := 1; \\
\text{while } x < 10000 \text{ do} & \\
\text{x} & := x + 1 \\
\od; & \\
\end{align*}
\]

\[
\begin{align*}
X_1 &= [1, 1] \\
X_2 &= (X_1 \cup X_3) \cap [-\infty, 9999] \\
X_3 &= X_2 \ominus [1, 1] \\
X_4 &= (X_1 \cup X_3) \cap [10000, +\infty] \\
\end{align*}
\]

Example: Interval analysis (1975)

Decreasing chaotic iteration:

\[
\begin{align*}
\text{x} & := 1; \\
\text{while } x < 10000 \text{ do} & \\
\text{x} & := x + 1 \\
\od; & \\
\end{align*}
\]

\[
\begin{align*}
X_1 &= [1, 1] \\
X_2 &= (X_1 \cup X_3) \cap [-\infty, 9999] \\
X_3 &= X_2 \ominus [1, 1] \\
X_4 &= (X_1 \cup X_3) \cap [10000, +\infty] \\
\end{align*}
\]
Example: Interval analysis (1975)

Decreasing chaotic iteration:

\[
\begin{align*}
x & := 1; \\
x & := x + 1 \quad \left\{ \begin{array}{l}
X_1 = [1, 1] \\
X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\
X_3 = X_2 \oplus [1, 1] \\
X_4 = (X_1 \cup X_3) \cap [10000, +\infty] 
\end{array} \right.
\end{align*}
\]

Final solution:

\[
\begin{align*}
x & := 1; \\
x & := x + 1 \quad \left\{ \begin{array}{l}
X_1 = [1, 1] \\
X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\
X_3 = X_2 \oplus [1, 1] \\
X_4 = (X_1 \cup X_3) \cap [10000, +\infty] 
\end{array} \right.
\end{align*}
\]

Result of the interval analysis:

\[
\begin{align*}
x & := 1; \\
x & := x + 1 \quad \left\{ \begin{array}{l}
X_1 = [1, 1] \\
X_2 = [1, 9999] \\
X_3 = [2, +10000] \\
X_4 = 0
\end{array} \right.
\end{align*}
\]

Checking absence of runtime errors with interval analysis:

\[
\begin{align*}
x & := 1; \\
x & := x + 1 \quad \left\{ \begin{array}{l}
X_1 = [1, 1] \\
X_2 = [1, 9999] \\
X_3 = [2, +10000] \\
X_4 = [+10000, +10000]
\end{array} \right. \quad \text{\(\leftarrow\) no overflow}
\end{align*}
\]
Refinement of abstractions

Approximations of an [in]finite set of points:

\[
\{\ldots, (19, 77), \ldots, (20, 03), \ldots \}
\]

From Below: dual \(^3\) + combinations.

\(^3\) Trivial for finite states (liveness model-checking), more difficult for infinite states (variant functions).

Effective computable approximations of an [in]finite set of points; Signs \(^4\)

\[
\{ x \geq 0 \\
y \geq 0
\]

Effective computable approximations of an \textit{[in]finite set of points}; \textit{Intervals} $^5$

\begin{align*}
\{ x \in [19, 77] \\
y \in [20, 03]
\end{align*}

\hspace{1cm}

Effective computable approximations of an \textit{[in]finite set of points}; \textit{Octagons} $^6$

\begin{align*}
\{ 1 \leq x \leq 9 \\
x + y \leq 77 \\
1 \leq y \leq 9 \\
x - y \leq 99
\end{align*}

\hspace{1cm}

Effective computable approximations of an \textit{[in]finite set of points}; \textit{Polyhedra} $^7$

\begin{align*}
19x + 77y \leq 2004 \\
20x + 03y \geq 0
\end{align*}

\hspace{1cm}

Effective computable approximations of an \textit{[in]finite set of points}; \textit{Simple congruences} $^8$

\begin{align*}
x = 19 \text{ mod } 77 \\
y = 20 \text{ mod } 99
\end{align*}

---


Effective computable approximations of an infinite set of points; Linear congruences

\[ \begin{align*}
1x + 9y &= 7 \mod 8 \\
2x - 1y &= 9 \mod 9
\end{align*} \]


Effective computable approximations of an infinite set of points; Trapezoidal linear congruences

\[ \begin{align*}
1x + 9y &\in [0, 77] \mod 10 \\
2x - 1y &\in [0, 99] \mod 11
\end{align*} \]


Refinement of iterates

Graphic example: Refinement required by false alarms
Graphic example: partitionned upward iteration with widening

Graphic example: partitionned upward iteration with widening

Interval widening with threshold set

- The threshold set $T$ is a finite set of numbers (plus $+\infty$ and $-\infty$),
- $[a, b] \bigwedge_{T} [a', b'] = \begin{cases} \max \{ \ell \in T \mid \ell \leq a' \} & \text{if } a' < a, \\ \text{else } a, \\ \min \{ h \in T \mid h \geq b' \} & \text{if } b' > b, \\ \text{else } b \end{cases}

- Examples (intervals):
  - sign analysis: $T = \{-\infty, 0, +\infty\}$;
  - strict sign analysis: $T = \{-\infty, -1, 0, +1, +\infty\}$;
  - $T$ is a parameter of the analysis.

Forward/reachability analysis
Example of iterated forward/backward analysis

Arithmetical mean of two integers \( x \) and \( y \):

\[
\begin{align*}
\{ x \geq y \} \\
\text{while } (x <> y) \text{ do} \\
\{ x \geq y + 2 \} \\
\quad x := x - 1; \\
\quad \{ x \geq y + 1 \} \\
\quad y := y + 1 \\
\quad \{ x \geq y \} \\
\od \\
\{ x = y \}
\end{align*}
\]

Necessarily \( x \geq y \) for proper termination

Example of iterated forward/backward analysis

Adding an auxiliary counter \( k \) decremented in the loop body and asserted to be null on loop exit:

\[
\begin{align*}
\{ x = y + 2k, x \geq y \} \\
\text{while } (x <> y) \text{ do} \\
\{ x = y + 2k, x \geq y + 2 \} \\
\quad k := k - 1; \\
\quad \{ x = y + 2k + 2, x \geq y + 2 \} \\
\quad x := x - 1; \\
\quad \{ x = y + 2k + 1, x \geq y + 1 \} \\
\quad y := y + 1 \\
\quad \{ x = y + 2k, x \geq y \} \\
\od \\
\{ x = y, k = 0 \} \\
\text{assume } (k = 0) \\
\{ x = y, k = 0 \}
\end{align*}
\]

Moreover the difference of \( x \) and \( y \) must be even for proper termination.
### Applications of abstract interpretation

- **Theoretical applications of abstract interpretation**
  - Static Program Analysis [POPL '77,78,79] including Data-flow Analysis [POPL '79,00], Set-based Analysis [FPCA '95], etc
  - Syntax Analysis [TCS 290(1) 2002]
  - Hierarchies of Semantics (including Proofs) [POPL '92, TCS 277(1–2) 2002]
  - Typing [POPL '97]
  - Model Checking [POPL '00]
  - Program Transformation [POPL '02]
  - Software watermarking [POPL '04]

### Industrial applications of abstract interpretation

- **Program analysis and manipulation**: a small rate of false alarms is acceptable
  - AiT: worst case execution time\(^{11}\)
  - StackAnalyzer: stack usage analysis\(^{11}\)
- **Program verification**: no false alarms is acceptable
  - TVLA: A system for generating abstract interpreters
  - Astrée: verification of absence of run-time errors\(^{11}\)

\(^{11}\) applied to the primary flight control software of the Airbus A340/600 and A380 fly-by-wire systems
Seminal papers


Recent surveys


Course Content

Anticipated Content of Course 16.399: Abstract Interpretation

- Today: an informal overview of abstract interpretation;

- The software verification problem (undecidability, complexity, test, simulation, specification, formal methods (deductive methods, model-checking, static analysis) and their limitations, intuitive notion of approximation, false alarms);

- Mathematical foundations (naive set theory, first order classical logic, lattice theory, fixpoints);
– **Semantics of programming languages** (abstract syntax, operational semantics, inductive definitions, example of a simple imperative language, grammar and interpreter of the language, trace semantics);

– **Program specification and manual proofs** (safety properties, Hoare logic, predicate transformers, liveness properties, linear-time temporal logic (LTL));

– **Order-theoretic approximation** (abstraction, closures, Galois connections, fixpoint abstraction, widening, narrowing, reduced product, absence of best approximation, refinement);

– **Numerical abstract domains** (intervals, affine equalities, congruences, octagons, polyhedra);

– **Symbolic abstract domains** (abstraction of sequences, trees and graphs, BDDs, word and tree automata, pointer analysis);

– **Case studies** (abstractions used in ASTREE and TVLA);

– **Principal of static analysis by abstract interpretation** (reachability analysis of a transition system, finite approximation, model-checking, infinite approximation, static analysis, program-based versus language-based analysis, limitations of finite approximations);

– **Design of a generic structural abstract interpreter** (collecting semantics, non-relational and relational analysis, converage acceleration by widening/narrowing);

– **Static analysis** (forward reachability analysis, backward analysis, iterated forward/backward analysis, inevitability analysis, termination)

### Anticipated Home Work of Course 16.399: Abstract Interpretation

– A **reading assignment** of the slides for each course and of a recommended recently published research article related to that course;

– A **personnal project** on the design and implementation of a **static analyzer** of numerical programs (which frontend will be provided)
Assigned reading for course 1

Patrick Cousot.
Abstract Interpretation Based Formal Methods and Future Challenges.

Anticipated Grading of Course 16.399:

Abstract Interpretation

The course is letter graded.
10% Class participation
15% Presentation 1
15% Presentation 2
40% Personal project
20% Final written exam

The two presentations of research papers are in CS conference format (25mn of talk and 5mn of questions) to be selected by the students in the list of assigned readings; to be held outside of lecture hours — times TBA.