Approximation of invariant sets by abstract interpretation and interval analysis

Damien Massé
with Luc Jaulin and Thomas Le Mézo

LabSTICC
Université de Bretagne Occidentale
Brest, France

Prasys seminar
Interval analysis represents an efficient approach for reliable computation.

- Manipulated elements are boxes.
- Safe bounds are computed.
- With efficient libraries to manipulate non-linear expressions.
Abstract interpretation

Abstract interpretation is a formal framework (initially) designed for static program analysis.

- Manipulated elements are in (any) abstract domains.
- Main goal is the safe approximations of fixpoints (for programs).
- Efficient libraries on numerical abstract domains, mainly for linear expressions.
Boxes are a well-known numerical abstract domain (Cousot & Cousot, 1976).
In static analysis, it is widely considered as being too imprecise. Faced with imprecision, attitude varies:

- people using interval analysis use bisections (if the dimension is low);
- people using abstract interpretation change the domain to relational (at least “weakly relational”) domain.
Boxes (or intervals)

Boxes are used to over-approximate sets of (tuple of) reals, using interval hulls (a box is a Cartesian product of intervals).

\[
\begin{pmatrix}
1 & 0 \\
-1 & 0 \\
0 & 1 \\
0 & -1
\end{pmatrix}
\begin{pmatrix}
x_1 \\
x_2
\end{pmatrix}
\leq
\begin{pmatrix}
\max x_1 \\
- \min x_1 \\
\max x_2 \\
- \min x_2
\end{pmatrix}
\]

The domain of boxes is a Moore family (i.e. closed by union). The function \( \rho \) abstracting to a box is a closure operator. Moore families (and closure operators) represent abstract domains.
Other Moore families: octagons [Mine, 2001]

Octagons is the most well-known example of (weakly) numerical relational domain, with relations of the form: $\pm x_i \pm x_j \leq c$.

Number of constraints for dim. $n$: $2n^2$.

Advantages: more precise than boxes.

Drawbacks: maybe too many constraints. Slower than boxes.
Template polyhedral domains [Sankaranarayanan et al., 2005]

Domains with fixed constraint matrix are called Template polyhedral domains.
Example with:

\[
T = \begin{pmatrix}
-1 & 0 \\
-1 & 3 \\
4 & 3 \\
1 & -4 \\
-2 & -3
\end{pmatrix}
\]

Advantages: possible to customize the template.
Drawbacks: in general, linear programming must be used to compute convex union, intersection.
Boxes and octagons are, of course, particular cases.
Abstract domains and abstract operators

Construction of abstract domains:
- definition;
- basic operations (closed union, intersection, emptiness test...);
- complex operations (depends on the intended use): numerical or symbolic operations.

Given a monotonic operator $\phi : \wp(\mathbb{R}^n) \rightarrow \wp(\mathbb{R}^n)$, a safe abstract operator $\phi^\#$ on abstract elements must satisfy:

$$\phi^\# \supseteq \rho \circ \phi \circ \rho$$

Replacing $\phi$ by $\phi^\#$ enables to make safe (over-)approximations of computations on the abstract domain.

The best abstract operator is of course $\rho \circ \phi \circ \rho$, but it is not always computable. Generally, one must make a trade-off between precision, efficiency and generality.
Example: paver with octagons

Replacing boxes with octagons to represent sets (with bisection):

With optimal operators

Non-optimal operators
Invariant sets

Let’s consider more complex sets, defined using a (deterministic) differential equation (with $x \in \mathbb{R}^n$ a state vector):

$$\dot{x} = f(x)$$

or (non-deterministic) differential inclusion:

$$\dot{x} \in F(x)$$

We consider solutions of these equations in $\mathbb{R}^+ \rightarrow \mathbb{R}^n$ (trajectories), and define:

- a **positive invariant** $B$ is a set of states from which all trajectory stays in $B$;
- a **capture basin** $C$ of a target $T$ is a set is states from which at least one trajectory goes to $C$;
- a **viability kernel** $K$ is a set of states from which at least one trajectory stays in $K$. 
Eulerian approach

Existing methods mostly use guaranteed integration over small intervals of time.
Eulerian approach: decompose the state space and approximate the sub-paths restricted to each subset of the state space.

A simpler approach (proposed by Luc Jaulin and Thomas Le Mézo) would be to represent only the transitions between each subset.
Maze (Luc Jaulin and Thomas Le Mézo)

The modelisation (abstraction) of the state space uses a paving $\mathcal{P}$ of boxes (which can be bisected to increase the precision), and doors at the boundary of each box:

- **input doors** for ingoing oriented paths;
- **output doors** for outgoing oriented paths.

With a valuation of the doors, the result is a maze.
Maze and control flow graph

Given the differential equation/inclusion and a paving gives something similar to a control flow graph, where each door is a vertex:

\[ x' = x \quad dt = 0 \]
\[ \exists dt \geq 0 \quad \exists x : [0, dt] \rightarrow P \]
\[ \dot{x} = f(x) \text{ et } x(dt) \in o_3 \]
\[ x = x(0) \in i_4 \text{ et } x' = x(dt) \]

The result is a **discretization approach** with arbitrary time between transitions from input to output doors (transitions from output to input doors are timeless).
Program semantics

Static analysis by abstract interpretation consider programs as transition systems over an \textit{infinite} set of states $\Sigma$ (to simplify, we consider the transition relation $\tau$ to be \textit{total}).

Note: here, one vertex = one state, not one program point.

A \textit{trace} is an (infinite) sequence of successive states (i.e. in $\mathbb{N} \rightarrow \Sigma$). The discretization approach assumes the existence of a relation between traces and oriented paths.
Temporal properties

Properties generally analysed are *temporal properties* (reachability, termination...), computed for set of traces.

Examples of *state-based temporal properties* (and their CTL expression):

- states from which one trace leads to $\theta (\EF \theta)$ ($\Rightarrow$ “capture basin”).
- states from which all traces lead to $\theta (\AF \theta)$;
- states from which there exists one trace staying in $\sigma (\EG \sigma)$ ($\Rightarrow$ “viability kernel”).
- states from which all traces stay in $\sigma (\AG \sigma)$ ($\Rightarrow$ “positive invariant”).
Example

Note that $\text{EF} \theta = \text{AG} \theta$ and $\text{AF} \theta = \text{EG} \theta$.

All these properties can be expressed using fixpoint semantics over monotone operators (called predicate transformers).
Predicate transformers

Two operators $\text{pre}$ and $\tilde{\text{pre}}$ on $\wp(\Sigma)$ are commonly used:

$$\text{pre}(X) = \{ s \mid \exists s' \in X, s \xrightarrow{\tau} s' \}$$

$$\tilde{\text{pre}}(X) = \{ s \mid \forall s' \in \Sigma, s \xrightarrow{\tau} s' \Rightarrow s' \in X \}$$

- $\text{pre}$ and $\tilde{\text{pre}}$ are monotonic (note that $\tilde{\text{pre}}$ is anti-monotone w.r.t. $\tau$);
- $\forall X, \tilde{\text{pre}}(X) = \Sigma \setminus \text{pre}(\Sigma \setminus X)$.
- when $\tau$ is deterministic, $\text{pre} = \tilde{\text{pre}}$;
Fixpoint semantics

If $s$ satisfies $\text{EF}\theta$, then its predecessors satisfy $\text{EF}\theta$. So $\text{EF}\theta$ is (almost) a fixpoint of $\text{pre}$.

More precisely:

- $\text{EF}\theta$ is the smallest set (least fixpoint) stable by the function $X \mapsto (\theta \cup \text{pre}(X))$:
  $$\text{EF}\theta = \text{lfp}(\theta \cup \text{pre})$$

- $\text{EG}\sigma$ is the largest set (greatest fixpoint) stable by the function $X \mapsto (\sigma \cap \text{pre}(X))$:
  $$\text{EG}\sigma = \text{gfp}(\sigma \cap \text{pre})$$

$\text{AF}\theta$ and $\text{AG}\sigma$ are similar, using $\widetilde{\text{pre}}$:

- $\text{AF}\theta = \text{lfp}(\theta \cup \widetilde{\text{pre}})$
- $\text{AG}\sigma = \text{gfp}(\sigma \cap \widetilde{\text{pre}})$
Construction of fixpoint semantics

The common constructive approach for fixpoints use the constructive form of the *Knaster-Tarski theorem*, e.g., starting with $\emptyset$ (of $\Sigma$), to repeatedly apply the function ("Kleene iterations") until convergence.

However:

- convergence is not guaranteed, even after $\omega$ iterations;
- the fixpoint may not (even) be memory representable.
Abstraction of fixpoint semantics

Using abstract domains, one can safely *over*-approximate the fixpoint semantics, e.g. with \( \text{pre}^\# \supseteq \rho \circ \text{pre} \circ \rho \):

\[
\text{lfp}(\theta \cup \text{pre}^\#) \subseteq \text{lfp}(\theta \cup \text{pre})
\]

Abstract domains use (almost always) over-approximations. Hence *under*-approximations requires to compute the complement, e.g.:

\[
\text{lfp}(\theta \cup \text{pre}) \subseteq \Sigma \setminus (\text{gfp}(\overline{\theta} \cap \overline{\text{pre}^\#}))
\]

Now the (abstract) fixpoints are memory representable, but the convergence is still not guaranteed.
Abstraction of the predicate transformers

An simple abstraction uses cones to represent all possibles directions of \( f(x) \) inside a box \([x]\).

abstract pre operator (from blue to red) abstract \( \tilde{\text{pre}} \) operator

To abstract \( \tilde{\text{pre}} \), we restrict the differential inclusion to a finite set of functions (controls). This restriction is itself an overapproximation (for the \( \tilde{\text{pre}} \) operator).

The approximated transition relation defines an affine program: transitions between states follow affine (polyhedral) constraints (of the form \( A \cdot \begin{pmatrix} X \\ X' \end{pmatrix} \leq B \)).
And so...

Van Der Pol basin of capture (target $\theta$ in red), deterministic:

- underapproximation by overapproximating $EG\bar{\theta}$ (greatest fixpoint, starting from the negation of the target);
- overapproximation using $EF\theta$ (least fixpoint, starting from the target).

Note: bisection is applied on the yellow zone to increase the precision.
And so (2)... Van Der Pol viability kernel with two controls, \( \sigma \) being the whole square without the center:

- underapproximation by overapproximating \( AF\bar{\sigma} \) (least fixpoint using \( \text{pre} \));
- overapproximation using \( EG\sigma \) (greatest fixpoint using \( \text{pre} \)).
Relation paths/traces

The whole computation relies on a (assumed) relation between the semantics of the graph (traces) and the semantics of the differential equation:

1. every oriented path can be associated to an (infinite) trace;
2. every trace can be associated to an oriented path.

Unfortunately, this assumption does not hold: a path may be associated to a finite trace, and infinite traces may represent sub-path (on limited time).
When a path is not an infinite trace

Easy case: when a path does not cross an infinite number of doors, it means that it “terminates” inside a box.
⇒ we just add a new infinitely looping state for this box.

Boxes with this state are those for which the “cone” has opposite vectors.
When an trace is not a (complete) path

Many cases where an infinite trace does not represent a whole path:
- when an infinite timeless cycle appears (at boundaries between boxes);
- or when a path crosses an infinite number of doors during a finite time.

Except for reachability analyses ($\text{EF}t$), the problem appears even when we try to remove obvious cycles.

No obvious way to solve this problem in the general case.
To summarize, two theoretical problems with practical consequences:
- the convergence of the fixpoint computation is still not guaranteed (though it seems to work, for now, on simple examples);
- spurious traces are taken into account.

Removing these traces would mean getting a fixpoint (the set is still stable by the predicate operators), but *not* the least (or greatest) fixpoint (trying to get “more than infinite” traces).

In this case, Kleene iterations are not usable (they cannot go “past” a fixpoint), so is there other methods to compute fixpoints?
Policy iteration

Policy iteration (in the context of abstract interpretation) is a technique which uses convex optimization to compute exact abstract fixpoint for specific kind of programs, in finite time. More specifically, for affine programs (checked) and template polyhedral abstract domain (checked), one can, using linear programming:

- compute the least fixpoint of the \( \text{pre}^\# \) operator in finite time [Gawlitza and Seidl, 2007];
- and also the greatest fixpoint of the \( \text{pre}^\# \) operator in finite time [Massé, 2012].

The extension to \( \tilde{\text{pre}}^\# \) is quite easy.
Policy iteration (intuition for the lfp)

Policy iteration can be seen as an extension of Newton method. Let’s consider a monotonic function $f : \mathbb{R} \to \mathbb{R}$ with $f = \max f_i$ where each $f_i$ is concave and the fixpoints of $f_i$ are computable ($f_i$ are the policies). Then:

- each fixpoint of $f$ is a fixpoint of (at least one) $f_i$;
- the least fixpoint of $f$ is computable.

- $p_1$, $p_2$, $p_3$ and $p_4$ are successive policy fixpoints, creating an increasing chain of pre-fixpoints;
- no policy can be selected twice;
- the algorithm stops when it reaches a fixpoint.

Policy iteration extends this idea to $\mathbb{R}^n$. 
Policy selection

In the case of lfp \( \text{pre} \), a policy can be seen as an “increasing” cycle of doors (containing a subpath going inside the current set).

Policies for lfp \( \overline{\text{pre}} \) are more abstract, creating different cycles for different controls, but represent broadly the same principle.
Beyond the least fixpoint

Since timeless cycles do not modify the current state values, they correspond to specific policies, locally equal to the identity function. These policies are never selected when computing the least fixpoint. By forcing their selection (e.g., stating that they represent $x \mapsto x + \epsilon$ instead of $x$, one can go beyond the least fixpoint).
Conclusions

Relationships between approximating differential inclusion sets and temporal property sets.
Lot of work to do, theoretical:

- formalisation of this approach, for timeless and limited time cycles;
- specific case of the greatest fixpoint;
- what about the modification of the paving (bisection)? other abstract domains (and higher dimensions)?

and practical:

- current implementations of policy iteration not suitable for the maze framework;
- for initial computations, Kleene iterations may be more efficient; how to switch to policy iteration when needed?