Inferring **Sufficient Conditions**
with **Backward Polyhedral Under-Approximations**

Antoine Miné

CNRS & École normale supérieure
Paris, France

**Workshop on Numerical and Symbolic Abstract Domains**
10 September 2012
Deauville
Inferring invariants is a well-studied problem, with many applications
(proving correctness, optimising, ...)

Favorite method: abstract interpretation using abstract domains
(scalable, terminates thanks to $\triangledown$, modular, flexible)

We want to infer sufficient conditions
for a given property to hold on a given program...
...and still use abstractions

Focus on numeric properties
$\implies$ numeric abstract domains (polyhedra)
sufficient conditions
  - transition systems
  - programs
  - applications

design effective abstract under-approximations
  - general algebraic properties
  - polyhedral operators
    (tests, assignments, loops, non-linear expressions)

Disclaimer
Work in progress, much research to be done
My goal: convince you that it might be interesting
Sufficient Conditions
Transition Systems

- \( \Sigma \): set of states
- \( \tau \subseteq \Sigma \times \Sigma \): transition relation, possibly non-deterministic
- initial states \( i_1, i_2, \ldots \) and final states \( f_1, f_2, \ldots \)
Given a set $I$ of initial states, compute the set $\text{inv}(I)$ of reachable states.
Sufficient Conditions

Transition Systems: Invariants

\[ \text{inv}(I) = \text{lfp}_I \lambda X. X \cup \text{post}(X) \]

where \( \text{post}(X) \overset{\text{def}}{=} \{ \sigma \in \Sigma \mid \exists \sigma' \in X: (\sigma', \sigma) \in \tau \} \)

Smallest set containing \( I \) and invariant by \( \tau \)
Given a target invariant set $T$, compute the largest set $I$ such that $\text{inv}(I) \subseteq T$.
\[ \text{cond}(T) = \text{gfp}_T \lambda X. X \cap \tilde{\text{pre}}(X) \]

where \( \tilde{\text{pre}}(X) \overset{\text{def}}{=} \{ \sigma \in \Sigma \mid \forall \sigma' \in \Sigma: (\sigma, \sigma') \in \tau \implies \sigma' \in X \} \)

\[ \implies \text{cond}(T) \text{ is the largest set } I \text{ such that } \text{inv}(I) \subseteq T \]
Example Program Analysis

Simple (non-deterministic) loop example

```
j := [0;10]
for (i = 0; i < 100; i++) {
    j = j + [0;1];
}
assert (j <= 105);
```
Simple (non-deterministic) loop example

```plaintext
j := [0;10]
for (i = 0; i < 100; i++) {
    j = j + [0;1];
}
assert (j <= 105);
```

**Forward invariant inference:**

final values of $j$

$\Rightarrow j \in [0;110]$, the assertion can be violated
Simple (non-deterministic) loop example

\[
\begin{align*}
  j & := [0;10] \\
  \text{for } (i = 0; i < 100; i++) & \{ \\
    j & = j + [0;1]; \\
  \} \\
  \text{assert } (j \leq 105);
\end{align*}
\]

**Classic backward analysis:**

initial values of \( j \) that \textit{may} lead to correct program termination

\[\Rightarrow j \in [0; 10] \quad \text{(we can always choose } 0 \in [0; 1])\]
Simple (non-deterministic) loop example

\[
j := [0;10]
\]

\[
\text{for (} i = 0; i < 100; i++ \text{) }
\]
\[
\quad j = j + [0;1];
\]
\[
}\}
\]
\[
\text{assert (} j <= 105);\]

**Backward sufficient condition analysis:**

initial values of \( j \) that **always** lead to correct program termination

\[
\implies j \in [0; 5] \quad \text{(works even if we choose always 1 } \in [0; 1] \text{)}
\]
Analysis Specificities

- backwards
- handling of non-determinism
- under-approximation
Analysis Specificities

- backwards

- handling of non-determinism

$$\widetilde{\text{pre}}(X) \neq \text{pre}(X) \overset{\text{def}}{=} \{ \sigma \in \Sigma \mid \exists \sigma' \in X : (\sigma, \sigma') \in \tau \}$$

we could use $$\widetilde{\text{pre}}(X) = \neg(\text{pre}(\neg X))$$

but abstract domains are seldom closed under $$\neg$$

$$\Rightarrow$$ classic backward operators cannot be recycled easily

- under-approximation
Analysis Specificities

- backwards
- handling of non-determinism
- under-approximation

\[ I' \subseteq \text{cond}(T) \implies \text{inv}(I') \subseteq T \]
\[ \implies \text{soundness requires under-approximations} \]

abstract domains seldom have best under-approximations (vs. best over-approximations via Galois connection)

we could use an abstract $X$ to represent $\neg X$
\[ \implies \text{turns over-approximations into under-approximations} \]
but $\neg X$ seldom represents an interesting property
\[ \implies \text{classic backward operators cannot be recycled easily} \]
Analysis Specificities

- backwards
- handling of non-determinism
- under-approximation

\[ \implies \text{design new, non-optimal operators on existing domains} \]
Applications

- sufficient condition for correctness
  application to contract inference

- run-time check hoisting

- counter-example inference
Applications

- sufficient condition for correctness
- run-time check hoisting

original

```c
for (i=0; i<n; i++)
    assert (i>=0 && i<N);
    t[i] = 1;
```

`→`

optimized

```c
if (n<=N)
    for (i=0; i<n; i++)
        t[i] = 1;
else
    original program
```

- counter-example inference
Applications

- sufficient condition for correctness
- run-time check hoisting
- counter-example inference
  - sufficient initial conditions for the assertion to fail
    $\Rightarrow$ under-approximate $\text{cond}(T)$ as before
  - proof that the assertion is eventually reached
    liveness property!
    $\Rightarrow$ instrument with a decreasing counter
    (idea from Cousot Cousot POPL'12)
Backward Polyhedral
Under-Approximations
Backward Versions

Programs are decomposed into atomic instructions \( i \) with well-known concrete transfer functions \( \tau \{ i \} \)

\[ \implies \text{derive concrete backwards transfer functions } \overleftarrow{\tau} \{ i \} \]

from concrete forward ones \( \tau \{ i \} \)

\[ \text{Backward version of } f : \mathcal{P}(X) \to \mathcal{P}(Y) \]

\[ \overleftarrow{f} : \mathcal{P}(Y) \to \mathcal{P}(X) \]

\[ \overleftarrow{f} (B) \overset{\text{def}}{=} \{ a \in X \mid f(\{a\}) \subseteq B \} \]

Core properties:

- \( \overleftarrow{\text{pre}} = \overleftarrow{\text{post}} \)
- \( \overleftarrow{f} \) is monotonic and a complete \( \cap \)-morphism
- \( \mathcal{P}(X) \xleftarrow{f} \xrightarrow{\overleftarrow{f}} \mathcal{P}(Y) \) (if \( f \) is a complete \( \cup \)-morphism)
Algebraic Properties

- \( \mathcal{f} \cup \mathcal{g} = \mathcal{f} \cap \mathcal{g} \) (but \( \mathcal{f} \cap \mathcal{g} \supset \mathcal{f} \cup \mathcal{g} \))
  (control-flow join)

- \( \mathcal{f} \circ \mathcal{g} = \mathcal{g} \circ \mathcal{f} \)
  (instruction sequence)

- \( f \subseteq g \iff \mathcal{f} \subseteq \mathcal{g} \)
  (approximation)

- \( \lambda x. \text{lfp}_x (\lambda z. z \cup \mathcal{f}(z)) = \lambda y. \text{gfp}_y (\lambda z. z \cap \mathcal{f}(z)) \)
  (loops)

\[\rightarrow\] break-down, abstract and combine backward functions
Polyhedra

domain of convex closed polyhedra

Dual representations:

Constraints

Generators

(Cousot & Halbwachs, POPL 1978)
Concrete Semantics:

we have \( \tau \{ a \cdot x \leq b? \} \Rightarrow R = \{ \rho \in R \mid a \cdot \rho(x) \leq b \} \)
Concrete Semantics:

we have  \( \tau \{ a \cdot x \leq b? \} \)  \( R = \{ \rho \in R \mid a \cdot \rho(x) \leq b \} \)

and so  \( \leftarrow \tau \{ a \cdot x \leq b? \} \)  \( R = R \cup \{ \rho \mid a \cdot \rho(x) > b \} \)
Abstract Polyhedral Semantics: \[ \{ a \cdot x \leq b \}^{\#} P \]

- remove \( a \cdot x \leq b \) from the constraint set
- remove all constraints redundant with \( a \cdot x \leq b \)

\[ \Rightarrow \text{under-approximation, not optimal} \]

Note: \( \lambda P. P \) always under-approximates \( \{ e? \} \)
Modeling If-Then-Else

Forward semantics: \( y + [0; 1] \geq 0 \)

\[
\tau\{ \text{if } (y + [0; 1] \geq 0) \{ t \} \text{ else } \{ e \} \} = \\
(\tau\{ t \} \circ \tau\{ y + [0; 1] \geq 0 \}) \cup (\tau\{ e \} \circ \tau\{ y + [0; 1] < 0 \})
\]

where \( \tau\{ y + [0; 1] \geq 0 \} R = \{ (x, y) \in R \mid y \geq -1 \} \)

\( \tau\{ y + [0; 1] < 0 \} R = \{ (x, y) \in R \mid y < 0 \} \)
Modeling If-Then-Else

Backward semantics:

\[ \{ \text{if } (y + [0; 1] \geq 0) \{ t \} \text{ else } \{ e \} \} = (\{ y + [0; 1] \geq 0 \} \circ \{ t \}) \cap (\{ y + [0; 1] < 0 \} \circ \{ e \}) \]

where

\[ \{ y + [0; 1] \geq 0 \} R = R \cup \{(x, y) \mid y < -1\} \]

\[ \{ y + [0; 1] < 0 \} R = R \cup \{(x, y) \mid y \geq 0\} \]
Backward Polyhedral Under-Approximations

Modeling If-Then-Else

\[ \tau \{ \cdot \} \text{ is not strict} \]
\[ \Rightarrow \text{ we can recover from a coarse under-approximation, even } \emptyset \]

**Example:** the analysis finds the else branch dead but continues nevertheless
Concrete Semantics:

we have \[ \tau \{ x := \text{?} \} R = \{ (x, y) \mid \exists v: (v, y) \in R \} \]
Concrete Semantics:

we have $\tau \{ x := ? \} \ R = \{ (x, y) \mid \exists v : (v, y) \in R \}$

and so $\leftarrow \tau \{ x := ? \} \ R = \{ (x, y) \mid \forall v : (v, y) \in R \}$
Theorem
If $R$ is convex closed, then $\leftarrow^\tau \{ x := ? \} R$ is either $R$ or $\emptyset$.

Abstract Polyhedral Semantics:

$$\leftarrow^\tau \{ x := ? \} P = \begin{cases} P & \text{if } \tau \{ x := ? \} P = P \\ \emptyset & \text{otherwise} \end{cases}$$

(exact)
Modeling Assignments

**Example:** $x := x + [a; b]$ (exact)

\[\begin{align*}
\tau \{ x := x + [a; b] \} R &= \{ (x + v, y) \mid (x, y) \in R, \ v \in [a; b] \} \\
\overleftarrow{\tau} \{ x := x + [a; b] \} R &= \{ (x, y) \mid \forall v \in [a; b]: (x + v, y) \in R \} \\
\overleftarrow{\tau} \{ x := x + [a; b] \}^\# P &= \tau \{ x := x - a \}^\# P \cap^\# \tau \{ x := x - b \}^\# P
\end{align*}\]

**General case:**

- $\tau \{ x := e \}$ can be synthesized using tests and projections
  \[\overleftarrow{\tau} \{ x := e \} \quad \text{and} \quad \overleftarrow{\tau} \{ x := e \}^\#\]
- $\tau \{ x := ? \}$ over-approximates $\tau \{ x := e \}$, so $\overleftarrow{\tau} \{ x := ? \}$
- under-approximates $\overleftarrow{\tau} \{ x := e \}$, so $\overleftarrow{\tau} \{ x := ? \}^\#$
Loops

Concrete Semantics:

\[
\tau \{ \text{while } (e) \{ b \} \} X \equiv \\
\tau \{ \neg e? \} (\text{lfp}_X \lambda Y. Y \cup (\tau \{ b \} \circ \tau \{ e? \})) Y)
\]

\[
\text{abstract lfp}_X F \text{ as the limit of } \left\{ \begin{array}{l}
X_0 \overset{\text{def}}{=} X \\
X_{i+1} \overset{\text{def}}{=} X_i \triangledown F(X_i)
\end{array} \right\}
\]

where

- \(X^\#\) and \(F^\#\) over-approximate \(X\) and \(F\)
- the widening \(\triangledown\) ensures convergence
  (e.g., remove unstable constraints)
Lower Widening

**Backward Abstract Semantics:**

Abstract $\text{gfp}_X F$ as the limit of

\[
\begin{align*}
X_0^\# & \overset{\text{def}}{=} X^\# \\
X_{i+1}^\# & \overset{\text{def}}{=} X_i^\# \wedge F^\#(X_i^\#)
\end{align*}
\]

where

- $X^\#$ and $F^\#$ under-approximate $X$ and $F$
- $\wedge$ is a **lower widening**
  - $A^\# \wedge B^\#$ under-approximates $A^\# \cap B^\#$
  - $\forall(X_n^\#)_{n \in \mathbb{N}}$: the sequence $Y_0^\# \overset{\text{def}}{=} X_0^\#$, $Y_{i+1}^\# \overset{\text{def}}{=} Y_i^\# \wedge X_{i+1}^\#$ stabilizes in finite time: $\exists i: Y_{i+1}^\# = Y_i^\#$
**Example Polyhedral Lower Widening:**

- remove unstable **generators**
- optionally use thresholds

▽ introduced formally by Cousot, along ▽ but no instance until now

▽ is not a narrowing!
Expression Abstraction

**Forward Expression Abstraction:**

Idea: replace $\tau \{ x := e \} R$ with $\tau \{ x := f \} R$ when

- $x := f$ is simpler to abstract than $x := e$
- $\forall \rho \in R: \llbracket e \rrbracket \rho \subseteq \llbracket f \rrbracket \rho$ (soundness)

**Backward Expression Abstraction:**

**Theorem**

$$\forall X : \overset{\leftarrow}{\tau} \{ x := e \} X \supseteq (\overset{\leftarrow}{\tau} \{ v := f \} X) \cap R$$

$\implies$ replace $\overset{\leftarrow}{\tau} \{ x := e \} X^\sharp$ with $(\overset{\leftarrow}{\tau} \{ x := f \} X^\sharp) \cap R^\sharp$

e.g.: $\overset{\leftarrow}{\tau} \{ x := y \ast z \}^\sharp P \longrightarrow (\overset{\leftarrow}{\tau} \{ x := [0; 1] \ast z \}^\sharp P) \cap Q$

if $Q \Rightarrow y \in [0; 1]$

*over-approximating e under-approximates $\overset{\leftarrow}{\tau} \{ x := e \}!*$

(also works on tests)
Extremely simple proof-of-concept

- analyzes a toy-language
- forward invariant + backward sufficient condition analysis
- abstract interpretation by induction on the syntax
- polyhedral abstractions based on Apron
- on-line analysis: http://www.di.ens.fr/~mine/banal
- sources freely available (in Ocaml)

Example analyses:
- simple loops (for \(...\) j = j + [0;1])
- bubble sort (Cousot Halbwachs 78)
Short-Comings

- Abstraction of tests

\[ \Rightarrow \text{how to choose now to maximize the end result?} \]

- Lower widening

\[ \Rightarrow \text{widening is the usual suspect} \]

- Experiments
Related Work

- \( w(l)p \) calculus  \((Dijkstra 75, Morris 97)\)
- (finite) model-checking  \((Dams 96)\)
- over-approximating backward abstract interpretation  
  \((Bourdoncle 93, Rival 05)\)
- under-approximations
  - exact disjunctive completions
  - path enumeration  \((Moy 08)\)
  - domains closed by complement  \(\text{\textcolor{red}{(Lev-Ami et al. 07)}}\)
- higher-order abstract interpretation
  - abstract lower closure operators  \(\text{\textcolor{red}{(Massé 02)}}\)
  - under-approximation on powersets  \(\text{\textcolor{red}{(Schmidt 04)}}\)
Conclusion

It **seems possible** to:
- infer **sound** sufficient conditions
- for **non-deterministic, infinite-state** programs
- using non-trivial **under-approximations**
- on **infinite-state** abstract domains

Our contribution:
1. **general properties** for compositional analysis design
2. example abstract transfer functions on **polyhedra**

Much more work to do to make it practical!