Computing invariants of programs
Calcul Formel team seminar

Laure Gonnord
http://laure.gonnord.org/pro/

Lille1 (USTL)/LIFL
Lille, France

Juin 2012
Working on C programs (or *flowcharts programs*), we aim to:

- Optimize the compilation process
- Prove safety properties
- Prove termination

▶ Static Analyses, of different kinds. Here we focus on *abstract interpretation*
Goal of Abstract Interpretation

Propagating **information** about program variables (numerical, arrays, ...) in order to get **invariants**.

- We focus on **numerical variables** here.
1. Program to transition systems/CFG

2. The problem and its fixpoint formulation

3. Fixpoint computation by means of abstract interpretation

4. Application domains
**Model - Notations**

**Numerical** properties verification on control flow graphs with **affine** actions and tests:

\[ AX \leq B \rightarrow X := CX + D \]

\( \tau : g \rightarrow a \)

(counter automata, interpreted automata)

- \( A, C \) matrices, \( B, D \) vectors.
- « natural » semantic.
- Objective: generating invariants for **each control point**
Vocabulary

- State: couple $(pc, val)$
- Initial states
- Reachable states.

- Reachability is undecidable.
From C programs to counter automata

Our tool C2FSM (Feautrier):
- Turns a sequential C file into a counter automaton.
- Performs **safe** abstractions of non numerical variables, structures, behaviors.

▶ other tools based on LLVM, Rose, ... still a research problem
1. Program to transition systems/CFG

2. The problem and its fixpoint formulation

3. Fixpoint computation by means of abstract interpretation

4. Application domains
The problem and its fixpoint formulation

Model - Example

0 ≤ x ≤ 142 is an invariant for "loop".
Problem Formalisation

\( \mathcal{R}_k \) = set of **valuations** at control point \( k \):

\[
\mathcal{R}_k = a_1 (\mathcal{R}_{k_1} \cap g_1) \cup a_2 (\mathcal{R}_{k_2} \cap g_2) \cup a_3 (\mathcal{R}_k \cap g_3)
\]
\( \mathcal{R}_k = \text{set of valuations at control point } k : \)

\[
\mathcal{R}_k = a_1 (\mathcal{R}_{k_1} \cap g_1) \cup a_2 (\mathcal{R}_{k_2} \cap g_2) \cup a_3 (\mathcal{R}_k \cap g_3)
\]

− (reccurent) equation system \( \mathcal{R}_k = F(\mathcal{R}_k) \), \textbf{fixpoint}, with \( \mathcal{R}_k^0 \)

= initial val of variables.
The problem and its fixpoint formulation

FixPoint - Exemple

\[
\begin{align*}
\text{init} & \quad x := 0 \\
\text{loop} & \quad x \leq 99 \\
& \quad \rightarrow x ++ \\
\text{end} & \quad x \geq 100
\end{align*}
\]

\[\mathcal{R}_{\text{init}} = \mathbb{R} = \top, \quad \mathcal{R}_{\text{loop}}^{1} = \{0\}, \text{ puis } \{0, 1\}, \ldots \text{ the least fixpoint is } \{0, 1 \ldots 100\}.\]
Fixpoint computation - Problems

- Representation (infinite sets, integers, \ldots )
- Computation of the transition relation
- Convergence

- **Linear relation Analysis**, ie abstract interpretation with polyhedra.
1. Program to transition systems/CFG
2. The problem and its fixpoint formulation
3. Fixpoint computation by means of abstract interpretation
4. Application domains
Fixpoint computation with polyhedra

- (Internal) representation of valuations and computations

- Transition function :
- Resolution convergence
Fixpoint computation with polyhedra

- (Internal) representation of valuations and computations
  convex polyhedra:

  \[
  P_k = \begin{cases} \top & \text{if } k = k_{\text{init}} \\ \bigsqcup_{(k, g, a, k')} a(P_{k'} \sqcap g) & \text{else} \end{cases}
  \]

- Transition function:
- Resolution convergence
Fixpoint computation with polyhedra

- (Internal) representation of valuations and computations
  convex polyhedra:

  \[
  P_k = \begin{cases} 
  \top & \text{if } k = k_{\text{init}} \\
  \bigcup_{(k,g,a,k')} a(P_{k'}, \cap g) & \text{else}
  \end{cases}
  \]

- Transition function: see later
- Resolution convergence **widening operator**, with replacing

  \[
  R_0, R_1 = F(R_0), R_2 = F(F(R_0)), \ldots \text{not convergent}
  \]

  by

  \[
  P_0, P_1 = P_0 \nabla F(P_0), P_1 \nabla F(P_1) \ldots \text{“machine” convergent}
  \]

▶ Approximation
On the example

\[
P_{\text{fin}}^{\text{loop}} = \{0 \leq x \leq 100\} \text{ if the analysis is precise enough}
\]
Resolution of the fixpoint system - 1

(pb 1 and 2) Convex polyhedra representation + transition function:

- Effective and efficient algorithmic (emptiness test, union, affine transformation . . . )
Fixpoint computation by means of abstract interpretation

Fixpoint system resolution - 2

(pb 3) Coping with termination.

**Widening**: $P \triangledown Q$ : limit extrapolation.

$P \triangledown Q$ constraints : take $Q$ constraints and remove those which are not saturated by $P$.

\[
\begin{align*}
x = y &= 0 \\
0 &\leq y \leq x \leq 1 \\
0 &\leq y \leq x
\end{align*}
\]

**Trick (!)**: \(\{x = y = 0\} = \{0 \leq y \leq x \leq 0\}\)
Fixpoint system resolution - 3

The widening operator being designed, we compute \((x \subseteq F(x))\)

\[ P_0, P_1 = P_0 \triangleleft F(P_0), P_2 = P_1 \triangleleft F(P_1) \ldots \]

finite computation instead of:

\[ P_0, F(P_0), F^2(P_0), \ldots \]

which can be infinite.

Theorem

(Cousot/Cousot 77) Iteratively computing the reachable states from the entry point with the domain operators and applying widening at entry nodes of loops converges in a finite number of steps to a overapproximation of the least invariant (postfixpoint).

- The widening operators must satisfy the non ascending chain condition (see Cousot/Cousot 1977).
Fixpoint system resolution - 4

Iteration strategy:
(Bourdoncle,1992) Computing strongly connected subcomponents and iterate inside each:

Gray nodes are **widening nodes**
Analysis example - 1

\[ x := 0; y := 0 \]

\[ \text{while } (x <= 100) \text{ do} \]

\[ \text{read}(b); \]

\[ \text{if } b \text{ then} \]

\[ x := x + 2 \]

\[ \text{else begin} \]

\[ x := x + 1; \]

\[ y := y + 1; \]

\[ \text{end;} \]

\[ \text{endif} \]

\[ \text{endwhile} \]
Example - 2

\[ x \leq 100 \rightarrow \]
\[ x := x + 1 \]
\[ y := y + 1 \]

\[ x \geq 100 \rightarrow \]
\[ x := x + 2 \]

\[ (x, y) := (0, 0) \]
Example - 2

\[
\begin{align*}
(x, y) & := (0, 0) \\
x & \leq 100 \rightarrow x := x + 1 \\
y & := y + 1 \\
x & \geq 100 \rightarrow x := x + 2
\end{align*}
\]
Example - 2

$$x \leq 100 \rightarrow 
\begin{align*}
(x, y) := (0, 0) \\
x := x + 1 \\
y := y + 1
\end{align*}$$

$$x \geq 100 \rightarrow 
\begin{align*}
x := x + 2
\end{align*}$$


$$p_{in} \rightarrow (x, y) := (0, 0) \rightarrow p \rightarrow x \leq 100 \rightarrow 
\begin{align*}
x := x + 1 \\
y := y + 1
\end{align*} \\
x \geq 100 \rightarrow p_{out}$$
Example - 2

Fixpoint computation by means of abstract interpretation

Laure Gonnord (USTL/LIFL) Computing invariants of programs 2012
Example - 2

\[
p_{\text{in}} \quad (x, y) := (0, 0) \quad p \quad x \leq 100 \rightarrow \\
x := x + 1 \quad p \quad x \leq 100 \rightarrow \\
y := y + 1 \quad x := x + 2 \quad p \quad x \geq 100 \rightarrow \\
\]

\[
p_{\text{out}} 
\]
Example - 2

Let's consider the following program:

\[
\begin{align*}
(x, y) &:= (0, 0) \\
&\quad x \leq 100 \rightarrow x := x + 1 \\
&\quad y := y + 1 \\
&\quad x \geq 100 \rightarrow x := x + 2 \\
&\quad (x, y) := (0, 0) \\
&\quad x \geq 100
\end{align*}
\]

The initial state is \((x, y) = (0, 0)\). The program updates `x` and `y` based on their values and continues to do so until `x` becomes greater than or equal to 100. The process is visualized in the diagram, which shows the transitions between states and the invariant properties that hold at each step.
Example - 2

\[ x \leq 100 \rightarrow \]
\[ p \]
\[ x := x + 1 \]
\[ y := y + 1 \]
\[ x \geq 100 \rightarrow \]
\[ p_{out} \]

\[ (x, y) := (0, 0) \]

\[ x \leq 100 \rightarrow \]
\[ x := x + 2 \]
Example - 2

\[(x, y) := (0, 0)\]

\[x \leq 100 \rightarrow x := x + 1\]
\[y := y + 1\]
\[x \geq 100\]

\[x \leq 100 \rightarrow x := x + 2\]
Example - 2

\( (x, y) := (0, 0) \)

\[
\begin{align*}
\text{if } x \leq 100 & \rightarrow \\
x & := x + 1 \\
y & := y + 1 \\
x & \geq 100 \rightarrow
\end{align*}
\]

\[
\begin{align*}
\text{if } x \leq 100 & \rightarrow \\
x & := x + 2
\end{align*}
\]
Demo: Aspic characteristics

**ASPIC**: Accelerated Symbolic Polyhedral Invariant Computation

- **Input**: the automaton is described in textual language (Fast) with or without proof goal (formula). Non deterministic and random operations ($x := ?$)
- **Classical computation + accelerations**
- **Output**: invariants (+ diagnostic).

Linear Relation Analysis

Complexity increases with:
- number of control points
- number of numerical variables

Approximation is due to:
- Convex hulls
- **Widening** (my PhD.)
1. Program to transition systems/CFG
2. The problem and its fixpoint formulation
3. Fixpoint computation by means of abstract interpretation
4. Application domains
Applications - 1

- Verification of **numerical** programs. « Proof » of non reachability:

- (non) reachability in counter automata coming from a SystemC Semantic (100 control points, J. Cornet (ST))
Applications - 2

By encoding in counter automata:

- Programs with **lists**: R. Iosif and S. Perarnau (Verimag, Grenoble)

- Programs with **pointers**: A. Sangnier and A. Finkel (LSV, Cachan)
Applications - 3

Work with COMPSYS (ENS Lyon): WTC (worst time complexity) estimation:
- compilation + static analysis
- scheduling

With A. Darte, P. Feautrier, C. Alias.
Conclusion

Linear relation analysis:
- computes numerical invariants
- on counter automata
- is not exact but sure (overapproximations)
- is performant
- is useful for other areas