Quantitative static analysis of Programs applied to HLS?

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

Lille1 (USTL)/LIFL
Lille, France
Verification / Static Analysis in circuits/HLS:

- Circuits are verified with Model Checking (boolean).
- As said Florent, we aims to develop higher-level tools, so let’s focus on C compilation.
- As we have C, we can use our « favorite » techniques.
  - Numerical Properties for the « control » part, for instance to bound the number of iterations (VHDL synthesis).
Infinite state programs (Static) verification:

- **safety** properties
- Infinite state: indecidability of most properties
  - interactive decision/proof
  - class restriction + abstractions.
  - if indecidable or too costly: approximations. Here: overapproximations ie « conservative » verification.
History of Abstract Interpretation

- French topic but now well used.
- Teams: ENS ULM, Verimag (Grenoble), Inria Popart team (Grenoble).
- Other People: Podelski (DE), Gulwani (US), Reps (US), . . .
- Static Analysis in HLS: Compsys (ENS Lyon) and future inria team in Lille (embedded systems) ?.
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**
0 ≤ x ≤ 142 is an invariant for "loop".
Problem Formalisation

\[ \mathcal{R}_k = \text{set of valuations at control point } k : \]

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

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

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

▶ (reccurent) equation system \( \mathcal{R}_k = F(\mathcal{R}_k) \), fixpoint.
\( \mathcal{R}_{init} = \mathbb{R} = \top, \ R_{loop}^{1} = \{0\}, \) puis \( \{0, 1\}, \ldots \) the least fixpoint is \( \{0, 1 \ldots 100\} \).
Fixpoint computation

- (Internal) représentation of valuations and computations

- Resolution convergence

▶ Approximation
Fixpoint computation

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

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

- Resolution convergence

▶ Approximation
Fixpoint computation

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

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

- Resolution convergence widening operator, with replacing

\[ R_0, R_1 = F(R_0), R_2 = F(F(R_0)), \ldots \] not convergent
by

\[ P_0, P_1 = P_0 \nabla F(P_0), P_1 \nabla F(P_1) \ldots \] convergent
On the example

- $x := 0$
- $x \leq 99 \rightarrow x++$
- $x > 100$

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

Convex polyhedra representation:

- Effective and efficient algorithmic (emptyness test, union, affine transformation . . .)
Fixpoint system resolution - 2

**Widening**: $P \nabla Q$ : limit extrapolation.  
$P \nabla Q$ constraints : take $Q$ constraints and remove those which are not saturated by $P$.

$\nabla = \{ x = y = 0 \} = \{ 0 \leq y \leq x \leq 0 \}$
Analysis example - 1

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

while \((x<=100)\) do
    read(b);
    if b then
        \(x:=x+2\)
    else begin
        \(x:=x+1;\)
        \(y:=y+1;\)
    end;
endif
endwhile

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

\(x \leq 100 \rightarrow\)
\(x := x + 1\)
\(y := y + 1\)

\(x \geq 100\)
\(x := x + 2\)
Example - 2

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

\begin{align*}
x \leq 100 & \rightarrow x := x + 1 \\
y & := y + 1 \\
x \geq 100 & \rightarrow x := x + 2
\end{align*}
Example - 2

\[ x \leq 100 \rightarrow \quad x := x + 1 \]
\[ y := y + 1 \]
\[ x \geq 100 \rightarrow \quad x := x + 2 \]

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

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

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

\[
\begin{align*}
\text{pin} & : x \leq 100 & \rightarrow & x := x + 1 \\
& & \rightarrow & y := y + 1 \\
& & \rightarrow & x \geq 100 \\
\text{pout} & : x \leq 100 & \rightarrow & x := x + 2
\end{align*}
\]
Example - 2

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

\[ x \leq 100 \rightarrow x := x + 1 \]
\[ y := y + 1 \]
\[ x \geq 100 \rightarrow x := x + 2 \]
Example - 2

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

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

\[ p \]

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

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

\[ x \geq 100 \]

\[ p_{in} \]

\[ p \]

\[ p_{out} \]

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

\[ p_{in} \]

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

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

\[ p \]

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

\[ p_{out} \]

\[ x \geq 100 \]
Linear Relation Analysis

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

Approximation is due to:
- Convex hulls
- **Widening** (my Phd.)
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).

▶ http://laure.gonnord.org/pro/aspic/aspic.html
Invariants of the example

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

\[
\begin{align*}
(x, y) & := (0, 0) \\
q_{\text{in}} & \\
p_{\text{in}} & \\
p_{\text{out}} & \rightarrow x \geq 100
\end{align*}
\]
Verification of **numerical** programs. « Proof » of non reachability:

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

- Numerical invariants of energy automata (from sensors)
  L. Samper et F. Maraninchi (Verimag, Grenoble)
Applications - 4

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

► With A. Darte, P. Feautrier, C. Alias.
► Demo?
Applications

Summary

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

Invariant generation

- Approximative: intervals, octogons, arrays, floating point...
- Exact: integer sets, intervals, booleans, constant propagation, ...
- Interprocedural, ...

and combinations
Tools

An interesting tool:

http://frama-c.cea.fr/

and also Apron Analyser:

pop-art.inrialpes.fr/interproc/interprocweb.cgi
Who/Where

**DART Team** in Inria Lille:

- codesigning specialized embedded systems (software and hardware)
- Compiling/synthesis issues, network issues, hardware issues, high level programming, and also **parallel** processing.
- Upgoing work on FPGA’s: expressing the architecture, architecture exploration (help for decision)
- Currently building a **group** to bring verification into the developpement process: rewriting, static analysis, model verification, . . . .

▶ postdocs positions . . .