A Tutorial on Program Analysis

Markus Müller-Olm
Dortmund University

Thanks!

Helmut Seidl
(TU München)

and

Bernhard Steffen
(Universität Dortmund)

for discussions, inspiration, joint work, ...
Dream of Program Analysis

```
main(){
    int x;
    if (x > 0)
        y=x+1;
    else
        x=42;
    while (y>99)
        y=2*y;
    x=x+1;
    printf("x=");
}
```

![Diagram of program analysis]

Property specification

G(Φ → FΨ)

Purposes of Automatic Analysis

- Optimizing compilation
- Validation/Verification
  - Type checking
  - Functional correctness
  - Security properties
  - ...  
- Debugging
Dream of Program Analysis

program\hspace{2cm} analyzer\hspace{2cm} result

```
main();
  { x = 7;
    if (x<5)
    { y = 7;
      x = 32;
      if (y==2)
        { while (y<8)
          { y = y+y+1;
              y = 0;
          }
          x = y+1;
          printf(x);
        }
    }
  }
```

$G(\Phi \rightarrow F\Psi)$

property specification

Fundamental Limit

Rice's Theorem [Rice,1953]:

All non-trivial semantic questions about programs from a universal programming language are undecidable.
Two Solutions

Weaker formalisms
- analyze abstract models of systems
- e.g.: automata, labelled transition systems, ...

Approximate analyses
- yield sound but, in general, incomplete results
- e.g.: detects some instead of all constants

Model checking
Flow analysis
Abstract interpretation
Type checking

Weaker Formalisms

Program

Abstract model

Exact analyzer for abstract model

Approximate

Exact
Overview

- Introduction
- Fundamentals of Program Analysis
- Interprocedural Analysis
- Analysis of Parallel Programs
- Invariant Generation
- Conclusion

Apology for not giving detailed credit!

Credits

- Pioneers of Iterative Program Analysis:
  - Kildall, Wegbreit, Kam & Ullman, Karr, ...
- Abstract Interpretation:
  - Cousot/Cousot, Halbwachs, ...
- Interprocedural Analysis:
  - Sharir & Pnueli, Knoop, Steffen, Rüthing, Sagiv, Reps, Wilhelm, Seidl, ...
- Analysis of Parallel Programs:
  - Knoop, Steffen, Vollmer, Seidl, ...
- And many more:
  - Apology ...
Overview

- Introduction
- Fundamentals of Program Analysis
- Interprocedural Analysis
- Analysis of Parallel Programs
- Invariant Generation
- Conclusion

---

From Programs to Flow Graphs

```c
main()
{  x=17;
   if (x>63)
      { y=17; x=10; x=x+1; }
   else
      { x=x+42;
          while (y<99)
             { y=x+y; x=x+1; }
          y=11;
      }
   x=y+1;
}
```
Dead Code Elimination

Goal:
find and eliminate assignments that compute values which are never used

Fundamental problem:
undecidability
→ use approximate algorithm:
e.g.: ignore that guards prohibit certain execution paths

Technique:
1) perform live variables analyses:
variable \( x \) is live at program point \( u \) iff
there is a path from \( u \) on which \( x \) is used before it is modified

2) eliminate assignments to variables that are not live at the target point

Live Variables
Live Variables Analysis

```
\[ x = x + 42 \]
\[ y > 63 \]
\[ y := 17 \]
\[ x := y + 1 \]
\[ (y > 63) \]
\[ x := 10 \]
\[ y := 11 \]
\[ (y < 99) \]
\[ y = x + y \]
```

Remarks

- Forward vs. backward analyses
- (Separable) bitvector analyses
  - forward: reaching definitions, available expressions, ...
  - backward: live/dead variables, very busy expressions, ...
Partial Order

Partial order \((L, \sqsubseteq)\):
set \(L\) with binary relation \(\sqsubseteq \subseteq L \times L\) s.t.
- \(\sqsubseteq\) is reflexive:
  \[ \forall x \in L: \ x \sqsubseteq x \]
- \(\sqsubseteq\) is antisymmetric:
  \[ \forall x, y \in L: \ x \sqsubseteq y \implies \neg(y \sqsubseteq x) \]
- \(\sqsubseteq\) is transitive
  \[ \forall x, y, z \in L: \ (x \sqsubseteq y \land y \sqsubseteq z) \implies x \sqsubseteq z \]

For a subset \(X \subseteq L\):
\(\bigvee X\): least upper bound \((join)\), if it exists
\(\bigwedge X\): greatest lower bound \((meet)\), if it exists

Complete Lattice

- Complete lattice \((L, \sqsubseteq)\):
  - a partial order \((L, \sqsubseteq)\) for which \(\bigvee X\) exists for all \(X \subseteq L\).

- In a complete lattice \((L, \sqsubseteq)\):
  - \(\bigwedge X\) exists for all \(X \subseteq L\):
    \[ \bigwedge X = \bigcup \{ x \in L \mid x \subseteq X \} \]
  - least element \(\bot\) exists:
    \[ \bot = \bigcup L = \bigcap \emptyset \]
  - greatest element \(T\) exists:
    \[ T = \bigvee \emptyset = \bigcap L \]

- Example:
  - for any set \(A\) let \(P(A) = \{ X \mid X \subseteq A \}\).
  - \((P(A), \sqsubseteq)\) is a complete lattice.
  - \((P(A), \supseteq)\) is a complete lattice.
**Interpretation in Approximate Program Analysis**

\( x \sqsubseteq y \):
- \( x \) is more precise information than \( y \).
- \( y \) is a correct approximation of \( x \).

\( \sqcup X \) for \( X \subseteq L \):
the most precise information consistent with all informations \( x \in X \).

**Remark:**
often dual interpretation in the literature!

**Example:**
lattice for live variables analysis:
- \((P(\text{Var}), \sqsubseteq)\) with \(\text{Var} = \text{set of variables in the program}\)

---

**Specifying Live Variables Analysis by a Constraint System**

Compute (smallest) solution over \((L, \sqsubseteq) = (P(\text{Var}), \sqsubseteq)\) of:

\[
\begin{align*}
V^*[\text{fin}] & \sqsubseteq \text{init}, & \text{for fin, the termination node} \\
V^*[u] & \sqsubseteq f^*_e(V^*[v]), & \text{for each edge } e = (u,s,v)
\end{align*}
\]

where \(\text{init} = \text{Var}\),

\(f^*_e : P(\text{Var}) \rightarrow P(\text{Var}), \quad f^*_e(x) = x \setminus \text{kill}_e \cup \text{gen}_e\) with
- \(\text{kill}_e = \text{variables assigned at } e\)
- \(\text{gen}_e = \text{variables used in an expression evaluated at } e\)
Specifying Live Variables Analysis by a Constraint System

Remarks:

1. Every solution is „correct“.

2. The smallest solution is called MFP-solution; it comprises a value MFP[u] ∈ L for each program point u.

3. (MFP abbreviates „maximal fixpoint“ for traditional reasons.)

4. The MFP-solution is the most precise one.

Data-Flow Frameworks

- Correctness
  - generic properties of frameworks can be studied and proved

- Implementation
  - efficient, generic implementations can be constructed
Questions

- Do (smallest) solutions always exist?
- How to compute the (smallest) solution?
- How to justify that a solution is what we want?
Knaster-Tarski Fixpoint Theorem

Definitions:
Let \((L, \sqsubseteq)\) be a partial order.
- \(f : L \to L\) is monotonic iff \(\forall x, y \in L : x \sqsubseteq y \Rightarrow f(x) \sqsubseteq f(y)\).
- \(x \in L\) is a fixpoint of \(f\) iff \(f(x) = x\).

Fixpoint Theorem of Knaster-Tarski:
Every monotonic function \(f\) on a complete lattice \(L\) has a least fixpoint \(\text{lfp}(f)\) and a greatest fixpoint \(\text{gfp}(f)\).
More precisely,
\[
\text{lfp}(f) = \bigcap \{ x \in L \mid f(x) \sqsubseteq x \} \quad \text{least pre-fixpoint}
\]
\[
\text{gfp}(f) = \bigcup \{ x \in L \mid x \sqsubseteq f(x) \} \quad \text{greatest post-fixpoint}
\]

Source: Nielson/Nielson/Harkin, Principles of Program Analysis
Smallest Solutions Exist Always

- Define functional $F : L^n \rightarrow L^n$ from right hand sides of constraints such that:
  - $\sigma$ solution of constraint system \iff $\sigma$ pre-fixpoint of $F$

- Functional $F$ is monotonic.

- By Knaster-Tarski Fixpoint Theorem:
  - $F$ has a least fixpoint which equals its least pre-fixpoint.

Questions

- Do (smallest) solutions always exist?

- How to compute the (smallest) solution?

- How to justify that a solution is what we want?
Workset-Algorithm

\[ W = \emptyset; \]
\[ \text{forall (program points } v) \{ A[v] = \bot; W = W \cup \{v\}; \} \]
\[ A[\text{fin}] = \text{init}; \]
\[ \text{while } W \neq \emptyset \{ \]
\[ \quad v = \text{Extract}(W); \]
\[ \quad \text{forall (u,s with } e = (u,s,v) \text{ edge) } \{ \]
\[ \quad \quad t = t_e(A[v]); \]
\[ \quad \quad \text{if } \neg (t \subseteq A[u]) \{ \]
\[ \quad \quad \quad A[u] = A[u] \cup t; \]
\[ \quad \quad \quad W = W \cup \{u\}; \]
\[ \quad \} \]
\[ \} \]

Live Variables Analysis
Invariants of the Main Loop

a) \( A[u] \subseteq \text{MFP}[u] \) f.a. prg. points \( u \)

b1) \( A[\text{fin}] \not\subseteq \text{init} \)

b2) \( v \in W \Rightarrow A[u] \not\subseteq f_c(A[v]) \) f.a. edges \( e = (u,s,v) \)

If and when worklist algorithm terminates:

\( A \) is a solution of the constraint system by b1)&b2)

\[ \Rightarrow A[u] \not\subseteq \text{MFP}[u] \quad \text{f.a. } u \]

Hence, with a): \( A[u] = \text{MFP}[u] \) f.a. \( u \)

How to Guarantee Termination

- Lattice \( (L, \subseteq) \) has finite heights
  \[ \Rightarrow \text{algorithm terminates after at most} \]
  \[ \#\text{prg points} \cdot (\text{heights}(L)+1) \]
  iterations of main loop

- Lattice \( (L, \subseteq) \) has no infinite ascending chains
  \[ \Rightarrow \text{algorithm terminates} \]

- Lattice \( (L, \subseteq) \) has infinite ascending chains:
  \[ \Rightarrow \text{algorithm may not terminate;} \]
  use \textit{widening operators} in order to enforce termination
Widening Operator

\( \triangledown: L \times L \rightarrow L \) is called a **widening operator** iff

1) \( \forall x,y \in L : x \cup y \sqsubseteq x \triangledown y \)

2) for all ascending chains \((l_i)_n\), the ascending chain \((w_i)_n\) defined by
   \[ w_0 = l_0, \quad w_{i+1} = w_i \triangledown l_i \text{ for } i > 0 \]
   stabilizes eventually.

Workset-Algorithm with Widening

\[ W = \emptyset; \]
\[ \text{forall } (\text{program points } v) \{ A[v] = \bot; \quad W = W \cup \{v\}; \} \]
\[ A[\text{fin}] = \text{init}; \]
\[ \text{while } W \neq \emptyset \{ \]
\[ v = \text{Extract}(W); \]
\[ \text{forall } (u,s \text{ with } e = (u,s,v) \text{ edge}) \{ \]
\[ t = \ell_s(A[v]); \]
\[ \text{if } \neg(t \sqsubseteq A[u]) \{ \]
\[ A[u] = A[u] \triangledown t; \]
\[ W = W \cup \{u\}; \]
\[ \} \]
\[ \} \]
Invariants of the Main Loop

a) \( A[u] \subseteq MFP[u] \) f.a. prg. points \( u \)

b1) \( A[\text{fin}] \subseteq \text{init} \)

b2) \( v \in W \Rightarrow A[u] \nsubseteq f_e(A[v]) \) f.a. edges \( e = (u, s, v) \)

With a widening operator we enforce termination but we lose invariant a).

Upon termination, we have:

\( A \) is a solution of the constraint system by b1)&b2)

\( \Rightarrow A[u] \nsubseteq MFP[u] \) f.a. \( u \)

Compute a sound upper approximation (only)!

Example of a Widening Operator:
Interval Analysis

The goal

Find save interval for the values of program variables, e.g. of \( i \) in:

```c
for (i=0; i<42; i++)
    if (0\leq i \land i<42) {
        A1 = A+i;
        M[A1] = i;
    }
```

..., e.g., in order to remove the redundant array range check.
Example of a Widening Operator: Interval Analysis

The lattice...

\((L, \subseteq) = \left( \left\{ [l, u] \mid l \in \mathbb{Z} \cup (-\infty), u \in \mathbb{Z} \cup (+\infty), l \leq u \right\} \cup \emptyset, \subseteq \right)\)

... has infinite ascending chains, e.g.:

\([0, 0] \subset [0, 1] \subset [0, 2] \subset ...\)

A widening operator:

\([l_0, u_0] \lor [l_i, u_i] = [l_2, u_2], \text{ where}\)

\[l_2 = \begin{cases} l_0 & \text{if } l_0 \leq l_i \\ -\infty & \text{otherwise} \end{cases} \quad \text{and} \quad u_2 = \begin{cases} u_0 & \text{if } u_0 \geq u_i \\ +\infty & \text{otherwise} \end{cases}\]

A chain of maximal length arising with this widening operator:

\(\emptyset \subset [3, 7] \subset [3, +\infty] \subset [-\infty, +\infty]\)

Analyzing the Program with the Widening Operator

Result is far too imprecise!

Example taken from: H. Seidl, Vorlesung „Programmoptimierung“, WS 04/05
Remedy 1: Loop Separators

- Apply the widening operator only at a "loop separator" (a set of program points that cuts each loop).
- We use the loop separator {1} here.

Identify condition at edge from 2 to 3 as redundant !

Remedy 2: Narrowing

- Iterate again from the result obtained by widening --- Iteration from a prefix-point stays above the least fixpoint ! ---

We get the exact result in this example !
Remarks

- Can use a work-list instead of a work-set
- Special iteration strategies
- Semi-naive iteration

Questions

- Do (smallest) solutions always exist?
- How to compute the (smallest) solution?
- How to justify that a solution is what we want?
  - MOP vs MFP-solution
  - Abstract interpretation
Questions

- Do (smallest) solutions always exist?
- How to compute the (smallest) solution?
- How to justify that a solution is what we want?
  - MOP vs MFP-solution
  - Abstract interpretation

Assessing Data Flow Frameworks
Live Variables

\[
\begin{align*}
\emptyset & \quad x := 17 \\
y & := 17 \\
x & := 10 \\
x & := x + 1 \\
x & := y + 1 \\
\text{out}(x) & \\
\emptyset & \quad \{y\} \\
x & := 42 \\
x & := x + y \\
x & := y + 1 \\
y & := y + 1
\end{align*}
\]

ininitely many such paths

\[
\text{MOP}[v] = \emptyset \cup \{y\} = \{y\}
\]

Meet-Over-All-Paths Solution

- Forward Analysis
  \[
  \text{MOP}[u] := \bigsqcup_{p \in \text{Paths}[\text{entry}, u]} F_p(\text{init})
  \]

- Backward Analysis
  \[
  \text{MOP}[u] := \bigsqcup_{p \in \text{Paths}[u, \text{exit}]} F_p(\text{init})
  \]

- Here: „Join-over-all-paths“; MOP traditional name
**Coincidence Theorem**

**Definition:**
A framework is positively-distributive if
\[ f(L \uplus X) = \sqcup \{ f(x) \mid x \in X \} \text{ for all } \emptyset \neq X \subseteq L, f \in F. \]

**Theorem:**
For any instance of a positively-distributive framework:
\[ \text{MOP}[u] = \text{MFP}[u] \quad \text{for all program points } u. \]

**Remark:**
A framework is positively-distributive if a) and b) hold:
(a) it is distributive: \( f(x \uplus y) = f(x) \uplus f(y) \) f.a. \( f \in F, x, y \in L \)
(b) it is effective: \( L \) does not have infinite ascending chains.

**Remark:**
All bitvector frameworks are distributive and effective.

---

**Lattice for Constant Propagation**

unknown value

\[ \cdots -2 -1 0 1 2 \cdots \]

inconsistent value
Constant Propagation Framework & Instance

lattice $L$ \( \text{Var} \rightarrow (\mathbb{Z} \cup \{\bot, \top\}) \), = \text{Var} \rightarrow \text{ConstVal}

\( \subseteq \) \( \rho \subseteq \rho' \iff \forall x: \rho(x) \subseteq \rho'(x) \)

\( \sqcup \) pointwise join

\( \top \) \( \top (x) = \top \) f.a. \( x \in \text{Var} \)

control flow program graph

initial value \( \top \)

function space \( \{ f : D \rightarrow D \mid f \text{ monotone} \} \)

\( f \)

\( f_i(d) = \begin{cases} 
\delta(x \mapsto \lfloor e \rfloor^{CP} (d)) & \text{if } i \text{ annotated with } x := e \\
\delta & \text{otherwise}
\end{cases} \)

\((\rho(x), \rho(y), \rho(z))\)

\(x := 2\)

\(y := 3\)

\((2,3,5)\)

\(x := 3\)

\(y := 2\)

\(z := x+y\)

\(\text{out}(x)\)

\((3,2,5)\)

\(\text{MOP}[v] = (\top, \top, 5)\)
(ρ(x), ρ(y), ρ(z))

26 : 3

\[
\begin{align*}
(2, \top, \top) & \quad x := 2 \\
(3, \top, \top) & \quad x := 3 \\
(2,3, \top) & \quad y := 3 \\
(3,2, \top) & \quad y := 2 \\
(\top, \top, \top) & \quad z := x+y \\
(\top, \top, \top) & \quad \text{out}(x)
\end{align*}
\]

\[
\begin{align*}
\text{MFP}[v] & = (\top, \top, \top) \\
\text{MOP}[v] & = (\top, \top, 5)
\end{align*}
\]

---

**Correctness Theorem**

**Definition:**

A framework is **monotone** if for all \( f \in F, x, y \in L \):
\[
x \sqsubseteq y \quad \Rightarrow \quad f(x) \sqsubseteq f(y).
\]

**Theorem:**

In any monotone framework:
\[
\text{MOP}[i] \subseteq \text{MFP}[i] \quad \text{for all program points } i.
\]

**Remark:**

Any "reasonable" framework is monotone.
Assessing Data Flow Frameworks

Questions

- Do (smallest) solutions always exist?
- How to compute the (smallest) solution?
- How to justify that a solution is what we want?
  - MOP vs MFP-solution
  - Abstract interpretation
Abstract Interpretation

Often used as reference semantics:

- sets of reaching runs:
  \[(D, \subseteq) = (P(\text{Edges}'), \subseteq) \quad \text{or} \quad (D, \subseteq) = (P(\text{Stmt}'), \subseteq)\]

- sets of reaching states (collecting semantics):
  \[(D, \subseteq) = (P(\Sigma'), \subseteq) \quad \text{with} \quad \Sigma = \text{Var} \rightarrow \text{Val}\]

Transfer Lemma

**Situation:**

Complete lattices \((L, \subseteq), (L', \subseteq')\)

Monotonic functions \(f : L \rightarrow L, g : L' \rightarrow L', \alpha : L \rightarrow L'\)

**Definition:**

Let \((L, \subseteq)\) be a complete lattice.

\(\alpha : L \rightarrow L\) is called **universally-disjunctive** iff \(\forall X \subseteq L : \alpha(\bigcup X) = \bigcup \{ \alpha(x) \mid x \in X \}\).

**Remark:**

- \((\alpha, \gamma)\) is called **Galois connection** iff \(\forall x \in L, x' \in L' : \alpha(x) \subseteq' y \iff x \subseteq \gamma(y)\).
- \(\alpha\) is universally-disjunctive iff \(\exists \gamma L' \rightarrow L : (\alpha, \gamma)\) is Galois connection.

**Transfer Lemma:**

Suppose \(\alpha\) is universally-disjunctive. Then:

(a) \(\alpha \circ f \subseteq g \circ \alpha \quad \Rightarrow \quad \alpha(\text{lfp}(f)) \subseteq \text{lfp}(g)\).

(b) \(\alpha \circ f = g \circ \alpha \quad \Rightarrow \quad \alpha(\text{lfp}(f)) = \text{lfp}(g)\).
Abstract Interpretation

Assume a universally-disjunctive abstraction function $\alpha : D \to D^\#$.

**Correct abstract interpretation:**

Show $\alpha(\alpha(x_1,\ldots,x_k)) \subseteq^\# \alpha(\alpha(x_1),\ldots,\alpha(x_k))$  
f.a. $x_1,\ldots,x_k \in L$, operators $o$

Then $\alpha(MFP[u]) \subseteq^\# MFP^\#[u]$  
f.a. $u$

**Correct and precise abstract interpretation:**

Show $\alpha(\alpha(x_1,\ldots,x_k)) = o^\#(\alpha(x_1),\ldots,\alpha(x_k))$  
f.a. $x_1,\ldots,x_k \in L$, operators $o$

Then $\alpha(MFP[u]) = MFP^\#[u]$  
f.a. $u$

Use this as guideline for designing correct (and precise) analyses!

---

**Abstract Interpretation**

Constraint system for reaching runs:

$$R[st] \supseteq \{s\}, \quad \text{for } st, \text{ the start node}$$

$$R[v] \supseteq R[u] \cdot \{\{e\}\}, \quad \text{for each edge } e = (u, v)$$

**Operational justification:**

Let $R[u]$ be components of smallest solution over $Edges^\*$. Then

$$R[u] = R^\omega[u] = \{ r \in Edges^* | st \xrightarrow{r} u \} \quad \text{f.a. } u$$

**Prove:**

a) $R^\omega[u]$ satisfies all constraints  

   $\Rightarrow R[u] \subseteq R^\omega[u]$  

   (direct)

b) $w \in R^\omega[u] \Rightarrow w \in R[u]$  

   (by induction on $|w|$)

   $\Rightarrow R^\omega[u] \subseteq R[u] \text{ f.a. } u$
Abstract Interpretation

Constraint system for reaching runs:
\[ R[st] \supseteq \{ \varepsilon \}, \quad \text{for } st, \text{ the start node} \]
\[ R[v] \supseteq R[u] \cdot \{ \{ e \} \}, \quad \text{for each edge } e = (u, s, v) \]

Derive the analysis:
Replace
\[ \{ \varepsilon \} \quad \text{by} \quad \text{init} \]  
\[ (\bullet) \cdot \{ \{ e \} \} \quad \text{by} \quad f_e \]

Obtain abstracted constraint system:
\[ R^* [st] \supseteq \text{init}, \quad \text{for } st, \text{ the start node} \]
\[ R^* [v] \supseteq f_e (R^* [u]), \quad \text{for each edge } e = (u, s, v) \]

Abstract Interpretation

MOP-Abstraction:
Define \( \alpha_{\text{MOP}} : \text{Edges}^* \rightarrow L \) by
\[ \alpha_{\text{MOP}} (R) = \sqcup \{ f_r (\text{init}) \mid r \in R \} \quad \text{where } f_x = \text{id}, \ f_x (v) = f_e \circ f_s \]

Remark:
For all monotone frameworks the abstraction is correct:
\( \alpha_{\text{MOP}} (R[u]) \sqsubseteq R^*[u] \)  f.a. prg. points \( u \)

For all universally-distributive frameworks the abstraction is correct and precise:
\( \alpha_{\text{MOP}} (R[u]) = R^*[u] \)  f.a. prg. points \( u \)

Justifies MOP vs. MFP theorems (cum grano salis). ☹️
Where Flow Analysis Loses Precision

Execution Semantic → MOP → MFP → Widening

Loss of Precision

Overview

- Introduction
- Fundamentals of Program Analysis
- Interprocedural Analysis
- Analysis of Parallel Programs
- Invariant Generation
- Conclusion
Interprocedural Analysis

Running Example: Availability of the single expression $a+b$

The lattice:

- $\text{false}$
  - $a+b$ not available
- $\text{true}$
  - $a+b$ available

Initial value: $\text{false}$
**Intra-Procedural-Like Analysis**

Conservative assumption: procedure destroys all information; information flows from call node to entry point of procedure

**Context-Insensitive Analysis**

Conservative assumption: Information flows from each call node to entry of procedure and from exit of procedure back to return point
Context-Insensitive Analysis

Conservative assumption: Information flows from each call node to entry of procedure and from exit of procedure back to return point.

Constraint System for Feasible Paths

Operational justification:
\[
\begin{align*}
S(u) &= \{ r \in \text{Edges}^* \mid st_{u} \xrightarrow{r} u \} \quad \text{for all } u \text{ in procedure } p \\
S(p) &= \{ r \in \text{Edges}^* \mid st_{p} \xrightarrow{r} e \} \quad \text{for all procedures } p \\
R(u) &= \{ r \in \text{Edges}^* \mid \exists \omega \in \text{Nodes}^* : st_{\omega} \xrightarrow{r} e \} \quad \text{for all } u
\end{align*}
\]

Same-level runs:
\[
\begin{align*}
S(p) &\supseteq S(r_p) \quad r_p \text{ return point of } p \\
S(st_p) &\supseteq \{e\} \quad st_p \text{ entry point of } p \\
S(v) &\supseteq S(u) \cdot \{e\} \quad e = (u,s,v) \text{ base edge} \\
S(v) &\supseteq S(u) \cdot S(p) \quad e = (u,p,v) \text{ call edge}
\end{align*}
\]

Reaching runs:
\[
\begin{align*}
R(st_{\text{Main}}) &\supseteq \{e\} \quad st_{\text{Main}} \text{ entry point of } \text{Main} \\
R(v) &\supseteq R(u) \cdot \{e\} \quad e = (u,s,v) \text{ basic edge} \\
R(v) &\supseteq R(u) \cdot S(p) \quad e = (u,p,v) \text{ call edge} \\
R(st_u) &\supseteq R(u) \quad e = (u,p,v) \text{ call edge, } st_u \text{ entry point of } p
\end{align*}
\]
Context-Sensitive Analysis

Idea:

Phase 1: Compute summary information for each procedure...
... as an abstraction of same-level runs

Phase 2: Use summary information as transfer functions for procedure calls...
... in an abstraction of reaching runs

Summary information:

1) Functional approach:
   Use (monotonic) functions on data flow informations!

2) Relational approach:
   Use relations (of a representable class) on data flow informations!

3) etc...

Functional Approach for Availability of Single Expression Problem

Observations:

Just three monotone functions on lattice $L$:\
\[
\begin{array}{c|c}
  \lambda x. \text{false} & \text{false} \\
  \lambda x. x & \text{true} \\
  \lambda x. \text{true} & \text{true}
\end{array}
\]

Functional composition of two such functions $f, g : L \rightarrow L$:

\[
h \circ f = \begin{cases} 
  f & \text{if } h = i \\
  h & \text{if } h \in \{g, k\}
\end{cases}
\]

Analogous: precise interprocedural analysis for all (separable) bitvector problems in time linear in program size.
Context-Sensitive Analysis, 1. Phase

Main:

\[ c = a + b \]

the lattice:

\[ k \]
\[ i \]
\[ g \]

Context-Sensitive Analysis, 2. Phase

Main:

false

true

false

true

false

true

false

true

false

true

false

true

false

true

false

true

false

true
Formalization of Functional Approach

Abstractions:

Abstract same-level runs with $\alpha_{s\text{-s}} : \text{Edges}' \to (L \to \mathcal{L})$:

$$\alpha_{s\text{-s}}(R) = \bigsqcup \{ f \ | \ f \in R \} \quad \text{for } R \subseteq \text{Edges}'$$

Abstract reaching runs with $\alpha_{r\text{-r}} : \text{Edges}' \to L$:

$$\alpha_{r\text{-r}}(R) = \bigsqcup \{ f(\text{init}) \ | \ f \in R \} \quad \text{for } R \subseteq \text{Edges}'$$

1. Phase: Compute summary informations, i.e., functions:

$$S^*(p) \supseteq S^*(r_p) \quad r_p \text{ return point of } p$$
$$S^*(st_p) \supseteq \text{id} \quad st_p \text{ entry point of } p$$
$$S^*(v) \supseteq f \cdot S^*(u) \quad e = (u,s,v) \text{ base edge}$$
$$S^*(v) \supseteq S^*(p) \cdot S^*(u) \quad e = (u,p,v) \text{ call edge}$$

2. Phase: Use summary informations; compute on data flow informations:

$$R^*(st_{\text{Main}}) \supseteq \text{init} \quad st_{\text{Main}} \text{ entry point of } \text{Main}$$
$$R^*(v) \supseteq f \cdot R^*(u) \quad e = (u,s,v) \text{ basic edge}$$
$$R^*(v) \supseteq S^*(p) \cdot R^*(u) \quad e = (u,p,v) \text{ call edge}$$
$$R^*(st_p) \supseteq R^*(u) \quad e = (u,p,v) \text{ call edge, } st_p \text{ entry point of } p$$

Functional Approach

Theorem:

Correctness: For any monotone framework:

$$\alpha_{M\text{-Mon}}(R[u]) \subseteq R^*[u] \quad \text{f.a. } u$$

Completeness: For any universally-distributive framework:

$$\alpha_{M\text{-Distr}}(R[u]) = R^*[u] \quad \text{f.a. } u$$

Alternative condition:

framework positively-distributive & all prog. point dyn. reachable

Remark:

a) Functional approach is effective, if $L$ is finite...

b) ... but may lead to chains of length up to $|L| \cdot \text{height}(L)$ at each program point.
**Extensions**

- Parameters, return values, local variables can be handled also

**Overview**

- Introduction
- Fundamentals of Program Analysis
- Interprocedural Analysis
- Analysis of Parallel Programs
- Invariant Generation
- Conclusion
Interprocedural Analysis of Parallel Programs

Interleaving- Operator \( \otimes \)
(Shuffle-Operator)

Example:

\[
\langle a, b \rangle \otimes \langle x, y \rangle = \left\{ \begin{array}{l}
\langle a, b, x, y \rangle \\
\langle a, x, b, y \rangle, \langle a, x, y, b \rangle \\
\langle x, a, b, y \rangle, \langle x, a, y, b \rangle, \langle x, y, a, b \rangle
\end{array} \right. 
\]
Constraint System for Same-Level Runs

Operational justification:

\[
S(u) = \{ r \in \text{Edges}^* | S_{f_p} \rightarrow^r u \} \text{ for all } u \text{ in procedure } p \\
S(p) = \{ r \in \text{Edges}^* | S_{f_p} \rightarrow^r \epsilon \} \text{ for all procedures } p
\]

Same-level runs:

\[
S(p) \supseteq S(r_p) \quad \text{ } r_p \text{ return point of } p \\
S(st_p) \supseteq \{ \epsilon \} \quad \text{ } st_p \text{ entry point of } p \\
S(v) \supseteq S(u) \{ \{ e \} \} \quad e = (u, s, v) \text{ base edge} \\
S(v) \supseteq S(u) \cdot S(p) \quad e = (u, p, v) \text{ call edge} \\
S(v) \supseteq S(u) \cdot (S(p_1) \otimes S(p_2)) \quad e = (u, p, v) \text{ parallel call edge}
\]

Constraint System for Reaching Runs

Operational justification:

\[
R(u, q) = \{ r \in \text{Edges}^* | \exists c \in \text{Config} : S_{f_p} \rightarrow^r c, \text{ } A_{u}(c) \} \\
P(q) = \{ r \in \text{Edges}^* | \exists c \in \text{Config} : S_{f_p} \rightarrow^r c \}
\]

Reaching runs:

\[
R(u, q) \supseteq S(u) \quad u \text{ program point in procedure } q \\
R(u, q) \supseteq S(v) \cdot R(u, p) \quad e = (v, p, _) \text{ call edge} \\
R(u, q) \supseteq S(v) \cdot (R(u, p_1) \otimes R(p_2)) \quad e = (v, p, v) \text{ parallel call edge, } i = 0,1
\]

Interleaving potential:

\[
P(p) \supseteq R(u, p) \quad u \text{ program point and } p \text{ procedure}
\]
Interleaving- Operator $\otimes$
(Shuffle-Operator)

Example:

$$\langle a, b \rangle \otimes \langle x, y \rangle = \{ \ \langle a, b, x, y \rangle \ , \ \langle a, x, b, y \rangle , \langle a, x, y, b \rangle , \langle x, a, b, y \rangle , \langle x, a, y, b \rangle , \langle x, y, a, b \rangle \ \}$$

Only new ingredient:
interleaving operator $\otimes$ must be abstracted!

Case: Availability of Single Expression

Abstract shuffle operator:

<table>
<thead>
<tr>
<th></th>
<th>i</th>
<th>g</th>
<th>k</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>i</td>
<td>g</td>
<td>k</td>
<td></td>
</tr>
<tr>
<td>g</td>
<td>g</td>
<td>g</td>
<td>k</td>
<td></td>
</tr>
<tr>
<td>k</td>
<td>k</td>
<td>k</td>
<td>k</td>
<td></td>
</tr>
</tbody>
</table>

The lattice:

- k (ill)
- i (ignore)
- g (generate)

Main lemma:

$$\forall f_j \in \{ g, k, i \} : \begin{array}{c}
\overbrace{\underbrace{ \ldots \ldots \ldots \ldots } }^{\epsilon(i) + \epsilon(p) + \epsilon + 1}
\end{array} f_i \circ \ldots \circ f_j \circ \ldots \circ f_i = f_j$$

Treat other (separable) bitvector problems analogously...

$\Rightarrow$ precise interprocedural analyses for all bitvector problems!
**Bitvector Problems**

Problem of this algorithm:

- **Complexity:** quadratic in program size: quadratically many constraints for reaching runs!

- **Solution:** linear-time „search for killers“-algorithm.

**Idea of „Search for Killers“-Algorithm**

- **the basic lattice:**
  - false
  - true

- **the function lattice:**
  - `k (ill)`
  - `i (ignore)`
  - `g (enerate)`

⇒ perform „normal“ analysis but weaken information if a „killer“ can run in parallel!
Formalization of „Search for Killers“-Algorithm

Kill Potential:

\[ KP(p) = T \text{ if } p \text{ contains reachable edge } e \text{ with } f_e = k \]
\[ KP(p) = KP(q) \text{ if } p \text{ calls } q, q \parallel_{<} \text{ or } q \parallel_{=} \text{ at some reachable edge} \]

Possible Interference:

\[ P(l(p)) \supseteq P(l(q)) \text{ if } q \text{ contains reachable call to } p \]
\[ P(l(p_i)) \supseteq P(l(q)) \cup KP(p_{j,i}) \text{ if } q \text{ contains reachable parallel call } p_{j,i} \text{ at some reachable edge} \]

Weaken data flow information in 2nd phase if killer can run in ||:

\[ R^*(s_{\text{init}}) \supseteq \text{init} \]
\[ R^*(v) \supseteq f_{\text{edge}}(R^*(u)) \]
\[ R^*(v) \supseteq S^*(p)(R^*(u)) \]
\[ R^*(s_{\text{entry}}) \supseteq R^*(u) \]
\[ R^*(v) \supseteq P(l(p)) \]

where:

- \( s_{\text{init}} \) entry point of Main
- \( e = (u,s,v) \) basic edge
- \( e = (u,p,v) \) call edge
- \( s_{\text{entry}} \) entry point of \( p \)
- \( v \) reachable prog. point in \( p \)

Beyond Bitvector-Analysis:
Analysis of Transitive Dependences

- Analysis problem:
  - Is there an execution from \( u \) to \( v \) mediating a dependence from \( x \) to \( y \) ?
    - \( a:=x \ldots b:=a \ldots c:=b \ldots y:=c \)

- Anwendungen:
  - program slicing
  - faint-code-elimination
  - copy constants
  - information flow ☺
Complexity Results

In parallel programs: [MO/Seidl, STOC 2001]

- analysis of transitive dependences is ...
  - undecidable, interprocedurally
  - PSPACE-complete, intraprocedurally
  - already NP-complete for programs without loop
under assumption

„Basic statements are executed atomically“

Analysis of Transitive Dependences in Parallel Programs

```
<table>
<thead>
<tr>
<th>a := x</th>
</tr>
</thead>
<tbody>
<tr>
<td>x := 1;</td>
</tr>
<tr>
<td>x := 0;</td>
</tr>
<tr>
<td>a := 0;</td>
</tr>
<tr>
<td>write(a)</td>
</tr>
</tbody>
</table>
```

Nevertheless: a is constantly 0!
Algorithmic Potential

In parallel programs: [MO, TCS 2004]
- transitive dependences are computable (in exponential time), even interprocedurally, if (unrealistic) assumption "Basic statements are executed atomically" is abandoned!

Technique:
- a (complex) domain of "dependence traces"
- abstract operators ∗ and ◦ which are precise and correct abstractions of ; and ◦ relative to a non-atomic semantics.

Analysis of Transitive Dependences in Parallel Programs

atomic execution | non-atomic execution
---|---
a := x | p := x
x := 1; x := 0; x := 1; x := 0;
a := 0; a := p a := 0; a := 0;
write(a) write(a)

a is constantly 0 ! a is not constantly 0 !
Overview

- Introduction
- Fundamentals of Program Analysis
- Interprocedural Analysis
- Analysis of Parallel Programs
- Invariant Generation
- Conclusion

Finding Invariants...

Main:

1. \( x_1 := x_2 \)
2. \( x_3 := 0 \)
3. \( x_1 := X_1 \cdot x_2 \cdot x_3 \)
4. \( x_1 := 0 \)

P:

5. \( x_2 := x_2 \cdot x_3 \cdot x_2 \cdot x_3 = 0 \)
6. \( x_3 := x_3 + 1 \)
7. \( x_1 := x_1 + x_2 + 1 \)
8. \( x_1 := x_1 \cdot x_2 \)
9. \( x_1 := x_1 \cdot x_2 \)
... through Linear Algebra

- Linear Algebra
  - vectors
  - vector spaces, sub-spaces, bases
  - linear maps, matrices
  - vector spaces of matrices
  - Gaussian elimination
  - ...

Applications

- definite equalities: \( x = y \)
- constant propagation: \( x = 42 \)
- discovery of symbolic constants: \( x = 5yz+17 \)
- complex common subexpressions: \( xy+42 = y^2+5 \)
- loop induction variables
- program verification!
- ...

Spring School on Security, Jerusalem, April 25-29, 2005
A Program Abstraction

Affine programs:

- **affine assignments:** \( x_1 := x_1 - 2x_2 + 7 \)
- **unknown assignments:** \( x_i := ? \)
  \[ \rightarrow \text{abstract too complex statements!} \]
- **non-deterministic instead of guarded branching**

The Challenge

Given an affine program

(with procedures, parameters, local and global variables, ...)

over \( R \):

- \( R \) the field \( \mathbb{Q} \) or \( \mathbb{Z}_p \), a modular ring \( \mathbb{Z}_m \), the ring of integers \( \mathbb{Z} \),
  an effective PIR,...)

- **determine all valid affine relations:**

  \[ a_0 + \sum a_i x_i = 0 \quad a_i \in R \quad 5x + 7y - 42 = 0 \]

- **determine all valid polynomial relations (of degree \( \leq d \))**:

  \[ p(x_1, \ldots, x_n) = 0 \quad p \in R \llbracket x_1, \ldots, x_n \rrbracket \quad 5x^2 + 7z^3 - 42 = 0 \]

... and all this in polynomial time (unit cost measure) !!!
Finding Invariants in Affine Programs

- **Intraprocedural:**
  - [Karr 76]: affine relations over fields
  - [Granger 91]: affine congruence relations over \( \mathbb{Z} \)
  - [Gulwani/Necula 03]: affine relations over random \( \mathbb{Z}_p \) \( p \) prime
  - [MO/Seidl 04]: polynomial relations over fields

- **Interprocedural:**
  - [Horwitz/Reps/Sagiv 96]: linear constants
  - [MO/Seidl 04]: polynomial relations over fields
  - [Gulwani/Necula 05]: affine relations over random \( \mathbb{Z}_p \) \( p \) prime
  - [MO/Seidl 05]: polynomial relations over modular rings \( \mathbb{Z}_m \), \( m \in \mathbb{Z} \) and PIRs

Infinity Dimensions

```

push-down

```

arithmetic
Use a Standard Approach for Interprocedural Generalization of Karr?

Functional approach [Sharir/Pnueli, 1981], [Knoop/Steffen, 1992]
- Idea: summarize each procedure by function on data flow facts
- Problem: not applicable

Call-string approach [Sharir/Pnueli, 1981]
- Idea: take just a finite piece of run-time stack into account
- Problem: not exact

Relational analysis [Cousot, 1977]
- Idea: summarize each procedure by approximation of I/O relation
- Problem: not exact (next slide)

Relational Analysis is Not Strong Enough

Main:
- \(x := 1\)
- \(x := x\)
- \(x := 2 \cdot x - 1\)

P:
- \(x := 1\)
- \(P()\)

True relational semantics of P:
- \(x_{post}\): True relation between \(x_{pre}\) and \(x_{post}\)

Best affine approximation:
- Approximation of the true relation

Spring School on Security, Marseille, April 25-29, 2005
Towards the Algorithm ...

Concrete Semantics of an Execution Path

- Every execution path \( \pi \) induces an affine transformation of the program state:

\[
\begin{align*}
\left[ x_1 := x_1 + x_2 + 1; x_3 := x_3 + 1 \right](v) \\
= \left[ x_3 := x_3 + 1 \right] \left[ x_1 := x_1 + x_2 + 1 \right](v) \\
= \left[ x_3 := x_3 + 1 \right] \begin{pmatrix} 1 & 1 & 0 & v_1 \\ 0 & 1 & 0 & v_2 \\ 0 & 0 & 1 & v_3 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \\
= \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}
\end{align*}
\]
Affine Relations

- An affine relation can be viewed as a vector:

\[ 5 + x_1 - 2x_2 - x_3 = 0 \]

Corresponds to

\[ a = \begin{pmatrix} 5 \\ 1 \\ -2 \\ -1 \end{pmatrix} \]

WP of Affine Relations

- Every execution path \( \pi \) induces a linear transformation of affine post-conditions into their weakest pre-conditions:

\[
[x_i := x_i + x_2 + 1; \ x_2 := x_2 + 1] \ (a)
\]

\[
= [x_i := x_i + x_2 + 1] \left[ x_i := x_i + 1 \right]^T (a)
\]

\[
= \begin{pmatrix}
1 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
a_1 \\
a_2 \\
a_3 \\
a_4
\end{pmatrix}
\]

\[
= \begin{pmatrix}
1 & 1 & 0 & 1 \\
0 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
a_1 \\
a_2 \\
a_3 \\
a_4
\end{pmatrix}
\]
Observations

- Only the zero relation is valid at program start:
  
  \[ 0 : 0 + 0x_1 + \cdots + 0x_k = 0 \]

- Thus, relation \( a_0 + a_1 x_1 + \cdots + a_k x_k = 0 \) is valid at program point \( v \) iff
  \[ M a = 0 \quad \text{for all } M \in \{ [[\pi]^T] \mid \pi \text{ reaches } v \} \]
  and:
  
  \[ M a = 0 \quad \text{for all } M \in \text{Span} \{ [[\pi]^T] \mid \pi \text{ reaches } v \} \]

- Matrices \( M \) form the \( R \)-module \( R^{(k+1) \times (k+1)} \).

- Sub-modules form a complete lattice of height \( O(w \cdot k^2) \).

Algorithm for Computing Affine Relations

1) Compute a generating system \( G \) with:

\[ \text{Span } G = \text{Span} \{ [[\pi]^T] \mid \pi \text{ reaches } v \} \]

by a precise abstract interpretation.

2) Solve the linear equation system:

\[ M a = 0 \quad \text{for all } M \in G \]

⇒ Need algorithms for:

1) Keeping generating systems in echelon form.
2) Solving (homogeneous) linear equation systems.
**Theorem**

1) The R-modules of matrices
   \[ \text{Span} \{ [\pi]^T \mid \pi \text{ reaches } v \} \]
   can be computed using arithmetic in R.

2) The R-modules
   \[ \{ a \in R^{n+1} \mid \text{affine relation } a \text{ is valid at } v \} \]
   can be computed using arithmetic in R.

3) The time complexity is **linear** in the program size and **polynomial** in the number of variables (unit cost measure):
   e.g. \( O(n \cdot k^3) \) for \( R=\mathbb{Q} \)
   \( (n \text{ size of the program, } k \text{ number of variables}) \)

4) We do not know how to avoid **exponential growth of number sizes**
   in interprocedural analysis for \( R \in \{\mathbb{Q},\mathbb{Z}\} \).
   However: we can avoid exponential growth in intra-procedural algorithms!

**An Example**

Main:

```
0
```

```
1: x_1 := x_2
2: x_3 := 0
3: P()
4: x_1 := x_1 \cdot x_2 \cdot x_3
```

P:

```
0
```

```
1: x_3 := x_3 + 1
2: x_1 := x_1 + x_2 + 1
3: P()
4: x_1 := x_1 \cdot x_2
```

\[
\begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1 \\
\end{bmatrix}
\quad +
\begin{bmatrix}
0 & 1 & 1 \\
1 & 1 & 0 \\
0 & 0 & 1 \\
\end{bmatrix}
\quad =
\begin{bmatrix}
1 & 2 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1 \\
\end{bmatrix}
\Rightarrow \text{stable!}
\]
**An Example**

Just the affine relations of the form $a_{x}x_{1} - a_{y}x_{2} - a_{z}x_{3} = 0$ are valid at 3.

**Extensions**

- Local variables, value parameters, return values
- Computing polynomial relations of degree $\leq d$
- Affine pre-conditions
Precise Analysis through Algebra

- Algebra
  - Polynomial rings, ideals, Gröbner bases, ...
  - Hilbert’s Basis Theorem ensures termination.

- Polynomial programs (over \( \mathbb{Q} \)):
  - Polynomial assignments: \( x := xy - 5z \)
  - Negated polynomial guards: \( \neg (xy - 3z = 0) \)
  - The rest as for affine programs!

- Intraprocedural computation of “polynomial constants” [MO/Seidl 2002]
- Intraprocedural derivation of all valid polynomial relations of degree \( \leq d \) [MO/Seidl 2003]

A Polynomial Program

After \( n \) iterations at 2:

\[
\begin{align*}
    x &= \sum_{i=0}^{n} q^i \cdot \frac{q^{n+1} - 1}{q-1} \quad \text{(Horner’s method)} \\
    y &= q^{n+1} \\
    \Rightarrow x \cdot (q - 1) &= y - 1 \\
    \Rightarrow x \cdot q - x - y + 1 &= 0
\end{align*}
\]

At 3:

\[
x - y + 1 = 0
\]
Computing Polynomial Relations

\[
p_1 := (a + b + c)q + (d - a) \\
p_2 := (a + c - d)q - cq^2 + (d - a)
\]

1. \(x := 1\) \\
2. \(x := x \cdot q + 1\), \(y := y \cdot q\)

\[
p_3 := axq - ax + by + cq + d \\
p_4 := axq^2 + aq - axq - a + byq + cq + d \\
x := x \cdot (q - 1)
\]

\[
p_5 := ax + by + cq + d
\]

\[
\begin{align*}
    a + b + c &= 0 & d - a &= 0 \\
    a + c - d &= 0 & e &= 0 & d - a &= 0
\end{align*}
\]

\[\iff \ a = d = -b \quad c = 0\]

\[\Rightarrow \quad \text{All identities of the form } ax - ay + a = 0 \quad \text{are valid.} \]

Conclusion

- Program analysis very broad topic
- Provides generic analysis techniques
- Some topics not covered:
  - Analyzing pointers and heap structures
  - Automata-theoretic methods
  - (Software) model checking
  - ...

Spring School on Security, Menali, April 25-29, 2005