Fundamentals of Program Analysis
+ Generation of Linear Prg. Invariants

Markus Müller-Olm
Westfälische Wilhelms-Universität Münster, Germany

2nd Tutorial of
SPP RS3: Reliably Secure Software Systems

Schloss Buchenau, September 3-6, 2012
Dream of Automatic Analysis

main()
{ x=17;
  if (x>63)
  { y=17;x=10;x=x+1;}
  else
  { x=42;
    while (y<99)
    { y=x+y;x=y+1;}
    y=11; }
  x=y+1;
  printf(x);
}

G(Φ → FΨ)

specification of property
Fundamental Problem

Rice's Theorem (informal version):
All non-trivial semantic properties of programs from a Turing-complete programming language are undecidable.

Consequence:
For Turing-complete programming languages:
Automatic analyzers of semantic properties, which are both correct and complete are impossible.
What can we do about it?

- Give up „automatic“: interactive approaches:
  - proof calculi, theorem provers, ...

- Give up „sound“: ???

- Give up „complete“: approximative approaches:
  - Approximate analyses:
    - data flow analysis, abstract interpretation, type checking, ...
  - Analyse weaker formalism:
    - model checking, reachability analysis, equivalence- or preorder-checking, ...
What can we do about it?

- Give up „automatic“: interactive approaches:
  - proof calculi, theorem provers, …

- Give up „sound“: ???

- Give up „complete“: approximative approaches:
  - Approximate analyses:
    - data flow analysis, abstract interpretation, type checking, …
  - Analyse weaker formalism:
    - model checking, reachability analysis, equivalence- or preorder-checking, …
Overview

- Introduction
- Fundamentals of Program Analysis
  Excursion 1
- Interprocedural Analysis
  Excursion 2
- Analysis of Parallel Programs
  Excursion 3
- Conclusion

Apology for not giving proper credit in these lectures!
Overview

- Introduction
- **Fundamentals of Program Analysis**
  - Excursion 1
- Interprocedural Analysis
  - Excursion 2
- Analysis of Parallel Programs
  - Excursion 3
- Conclusion

Apology for not giving proper credit in these lectures!
From Programs to Flow Graphs

main() {
    x=17;
    if (x>63) {
        y=17;x=10;x=x+1;
    } else {
        x=x+42;
        while (y<99) {
            y=x+y;x=y+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

- Node 0: \(x = 17\)
- Node 1: \(y > 63\)
- Node 2: \(y := 17\)
- Node 3: \(x := 10\)
- Node 4: \(x := x + 1\)
- Node 5: \(x = x + 42\)
- Node 6: \(y < 99\)
- Node 7: \(y := 11\)
- Node 8: \(x := y + 1\)
- Node 9: \(x := y + 1\)
- Node 10: \(y := y + 1\)
- Node 11: \(x := x + 1\)
Live Variables Analysis

Graph showing variable assignments and conditions.
Interpretation of Partial Orders in Approximate Program Analysis

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

$\sqcup X$ for $X \subseteq L$, where $(L, \sqsubseteq)$ is the partial order:
the most precise information consistent with all informations $x \in X$.

Example:
order for live variables analysis:
- $(P(Var), \subseteq)$ with $Var =$ set of variables in the program

Remark:
often dual interpretation in the literature!
Complete Lattice

Complete lattice \((L, \sqsubseteq)\):
- a partial order \((L, \sqsubseteq)\) for which the least upper bound, \(\sqcup X\), exists for all \(X \subseteq L\).

In a complete lattice \((L, \sqsubseteq)\):
- \(\cap X\) exists for all \(X \subseteq L\):
  \[\cap X = \sqcup \{ x \in L \mid x \sqsubseteq X \}\]
- least element \(\bot\) exists:
  \[\bot = \cup L = \cap \emptyset\]
- greatest element \(\top\) exists:
  \[\top = \cap \emptyset = \cup L\]

Example:
- for any set \(A\) let \(P(A) = \{X \mid X \subseteq A\}\) (power set of \(A\)).
- \((P(A), \sqsubseteq)\) is a complete lattice.
- \((P(A), \supseteq)\) is a complete lattice.
Specifying Live Variables Analysis by a Constraint System

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

\[
\begin{align*}
A[\text{fin}] & \supseteq \text{init}, \quad \text{for fin, the termination node} \\
A[u] & \supseteq f_e(A[v]), \quad \text{for each edge } e = (u, s, v)
\end{align*}
\]

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

\(f_e: P(\text{Var}) \to P(\text{Var}), \quad f_e(x) = x \setminus \text{kill}_e \cup \text{gen}_e, \quad \text{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“ (whatever this means).

2. The smallest solution is called MFP-solution; it comprises a value $\text{MFP}[u] \in L$ for each program point $u$.

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

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

The Goal:
Find for each program point expressions or variables that have a constant value at this program point

Enabled Optimizations:
- Replace constant variables or expressions at compile time by their value → smaller and faster code
- Eliminate unreachable branches in the program → smaller code

Remarks:
Constant expressions and variables appear often, e.g.:
- const-declarations in PASCAL
- final attributes in JAVA-interfaces, ...
- values computed out of declared constants
Constant Propagation

\((\rho(x), \rho(y), \rho(z))\)
A Lattice for Constant Propagation

An order $\sqsubseteq$ on $\mathbb{Z} \cup \{\top\}$:

$\top$ $\rightarrow$ "unknown value"

\[
\begin{array}{c}
\ldots \ \ -2 \ \ -1 \ \ 0 \ \ 1 \ \ 2 \ \ \ldots \\
\end{array}
\]

$L_{\text{CP}} = \{\rho \mid \rho : \text{Var}_G \rightarrow (\mathbb{Z} \cup \{\top\}) \cup \{\bot\}\}

\rho \sqsubseteq_{\text{CP}} \rho' \iff \rho = \bot \vee (\rho \neq \bot \land \rho' \neq \bot \land \forall x \in \text{Var}_G : \rho(x) \sqsubseteq \rho'(x))$

Remark: $(L_{\text{CP}}, \sqsubseteq_{\text{CP}})$ is a complete lattice.
Constant Propagation

\[(\rho(x), \rho(y), \rho(z))\]
Specifying Constant Propagation by a Constraint System

Sei $G = (N,E,st,te)$ ein Flussgraph über $BA_{std}$.

Compute (smallest) solution over $(L,\sqsubseteq) = (L_{CP},\sqsubseteq_{CP})$ of:

$V[st] \sqsubseteq init$, for $st$, the start note
$V[v] \sqsubseteq f_{e}(V[u])$, for each edge $e = (u,s,v)$

where:

$init = T_{CP} \in L_{CP}$ is the mapping $T_{CP}(x) = T$ and

$f_{e}: L_{CP} \rightarrow L_{CP}$ is defined by

$$f_{e}(\rho) =_{df} \begin{cases} 
\rho\{[t]_{CP}(\rho) / x\}, & \text{if } e = (u,x:=t,v) \text{ und } \rho \neq \bot \\
\rho, & \text{otherwise}
\end{cases}$$
Specifying Constant Propagation by a Constraint System

Remarks:

1. Again, every solution is „correct“ (whatever this means).

2. Again, the smallest solution is called **MFP-solution**; it comprises a value $\text{MFP}[u] \in L$ for each program point $u$.

3. The MFP-solution is the **most precise** one.
Backwards vs. Forward Analyses

Live Variables Analysis is a **Backwards Analysis**, i.e.:

- analysis info flows from target node to source node of an edge
- the initial inequality is for the termination node of the flow graph

\[ A[te] \supseteq \text{init}, \quad \text{for } te, \text{ the termination point} \]
\[ A[u] \supseteq f_e(A[v]), \quad \text{for each edge } e = (u, s, v) \in E \]

Dually, constant propagation is a **Forward Analyses** i.e.:

- analysis info flows from source node to target node of an edge.
- the initial inequality is for the start node of the flow graph

\[ A[st] \supseteq \text{init}, \quad \text{for } st, \text{ the start node} \]
\[ A[v] \supseteq f_e(A[u]), \quad \text{for each edge } e = (u, s, v) \in E \]

Other examples: reaching definitions, available expressions, ...
Monotone Data-Flow Problems

**Goal:**
- A generic notion that captures what is common for different analyses

**Advantages:**
- Study general properties of data flow problems independently of concrete analysis questions
- Build efficient, generic implementations
Monotone Data-Flow Problems

Definition:

A monotone data-flow problem is a tuple $P = ((L, \sqsubseteq), F, (N, E), st, init)$ consisting of:

- a complete lattice $(L, \sqsubseteq)$.  
  The elements of $L$ are called (data-flow facts).
- a set $F$ of transfer functions $f : L \rightarrow L$, such that:
  - each $f \in F$ is monotone: $\forall x, y \in L : x \sqsubseteq y \Rightarrow f(x) \sqsubseteq f(y)$
  - $id \in F$
  - $F$ is closed under composition: $\forall f, g \in F : f \circ g \in F$.
- A graph $(N, E)$ with a finite set of edges $N$;
  each node of the graph is annotated with a transfer function $f \in F$:
  
  $E \subseteq N \times F \times N$.
- $st \in N$ is a designated initial node.
- $init \in L$ is a designated initial information.
Constraint System for a Data-Flow Problem

Let \( P = ((L,\sqsubseteq),F,(N,E),u_0,\text{init}) \) be a data-flow problem.

Compute (smallest) solution over \((L,\sqsubseteq)\) of the followi constraint system:

\[
\begin{align*}
A[st] & \sqsubseteq \text{init}, \quad \text{for st, the start node} \\
A[v] & \sqsubseteq f(A[u]), \quad \text{for each node } e = (u,f,v) \in E
\end{align*}
\]

Note:
Here, information flows from nodes to their successor nodes only. Hence, for backwards analyses the direction of the edges must be reversed when mapping it to the corresponding data-flow problem.


Constraint System for a Data-Flow Problem

Remarks:

1. Again, every solution is „correct“ (whatever this means).

2. Again, the smallest solution is called MFP-solution; it comprises a value $\text{MFP}[u] \in L$ for each program point $u$.

3. The MFP-solution is the most precise one.
Three Questions

- Do (smallest) solutions always exist?
- How to compute the (smallest) solution?
- How to justify that a solution is what we want?
Three 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,
\[
\begin{align*}
\text{lfp}(f) &= \sqcap \{ x \in L \mid f(x) \sqsubseteq x \} & \text{least pre-fixpoint} \\
\text{gfp}(f) &= \sqcup \{ x \in L \mid x \sqsubseteq f(x) \} & \text{greatest post-fixpoint}
\end{align*}
\]
Knaster-Tarski Fixpoint Theorem

L:

- $\top \top \top \top$
- $\bot \bot \bot \bot$
- $\text{pre-fixpoints of } f$
- $\text{gfp}(f)$
- $\text{fixpoints of } f$
- $\text{lfp}(f)$
- $\text{post-fixpoints of } f$

Picture from: Nielson/Nielson/Hankin, *Principles of Program Analysis*
Smallest Solutions Always Exist

- 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.
Three Questions

- Do (smallest) solutions always exist?
- How to compute the (smallest) solution?
- How to justify that a solution is what we want?
Workset-Algorithm

\[ W = \emptyset; \]

forall (program points \( v \)) \{ \( A[v] = \bot; \) \( W = W \cup \{ v \}; \) \}

\( A[st] = \text{init}; \)

while \( W \neq \emptyset \) \{ \\
\( u = \text{Extract}(W); \)

forall (s, v with \( e = (u, s, v) \) edge) \{ \\
\( t = f_e(A[u]); \)

if \( \neg(t \subseteq A[v]) \) \{ \\
\( A[v] = A[v] \cup t; \)
\( W = W \cup \{ v \}; \)

\}

\}

\}
Invariants of the Main Loop

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

b1) \( A[st] \supseteq init \)

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

If and when workset algorithm terminates:

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

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

Hence, with a): \( A[u] = MFP[u] \) f.a. \( u \) ☺
How to Guarantee Termination

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

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

- Lattice \((L, \sqsubseteq)\) has infinite ascending chains:
  \[ \Rightarrow \text{algorithm may not terminate; } \]
  use \textit{widening operators} in order to enforce termination
\( \nabla : L \times L \rightarrow L \) is called a \textit{widening operator} iff

1) \( \forall x, y \in L : x \sqcup y \sqsubseteq x \nabla y \)

2) for all sequences \((l_n)_n\), the (ascending) chain \((w_n)_n\)

\[ w_0 = l_0, \quad w_{i+1} = w_i \nabla l_{i+1} \text{ for } i > 0 \]

stabilizes eventually.
Workset-Algorithm with Widening

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

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

b1) \[ A[st] \supseteq \text{init} \]

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

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

Upon termination, we have:

\[ A \text{ is a solution of the constraint system by b1)&b2)} \]

\[ \Rightarrow A[u] \supseteq \text{MFP}[u] \] 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<=i and 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...

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

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

$[0,0] \subset [0,1] \subset [0,2] \subset \ldots$

A widening operator:

$$[l_0, u_0] \triangledown [l_1, u_1] = [l_2, u_2],$$

where

$$l_2 = \begin{cases} l_0 & \text{if } l_0 \leq l_1 \\ -\infty & \text{otherwise} \end{cases} \quad \text{and} \quad u_2 = \begin{cases} u_0 & \text{if } u_0 \geq u_1 \\ +\infty & \text{otherwise} \end{cases}$$

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

$\neg(i < 42) \quad \neg(0 \leq i < 42)$

$A_1 := A + i$

$M[A_1] := i$

$i := i + 1$

$\Rightarrow$ Result is far too imprecise!
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!
Find out, prg. point 7 is unreachable!

\[
\begin{array}{l}
\text{i:=0} \\
\neg (i < 42) \\
\neg (0 \leq i < 42) \\
\end{array}
\]

\[
\begin{array}{l}
\text{0} \\
\text{1} \\
\text{8} \\
\text{2} \\
\text{3} \\
\text{7} \\
\text{4} \\
\text{5} \\
\text{6} \\
\text{7} \\
\text{8} \\
\end{array}
\]

\[
\begin{array}{l}
\text{i<42} \\
\text{0 \leq i < 42} \\
\text{A_1 := A + i} \\
\text{M[A_1] := i} \\
\text{i := i+1} \\
\end{array}
\]

\[
\begin{array}{|c|c|c|c|c|}
\hline
& 1 & 2 & 3 \\
\hline
\text{l} & -\infty & -\infty & -\infty \\
\text{u} & +\infty & +\infty & +\infty \\
\text{l} & 0 & 0 & 0 \\
\text{u} & 0 & 41 & 41 \\
\text{l} & 0 & 0 & 0 \\
\text{u} & 0 & 41 & 41 \\
\text{l} & 1 & 1 & 1 \\
\text{u} & 42 & 42 & 42 \\
\text{l} & \bot & \bot & \bot \\
\text{u} & 42 & +\infty & +\infty \\
\hline
\end{array}
\]
Remedy 2: Narrowing

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

\[ i := 0 \]
\[ \neg (i < 42) \]
\[ i < 42 \]
\[ \neg (0 \leq i < 42) \]
\[ 0 \leq i < 42 \]
\[ A_1 := A + i \]
\[ M[A_1] := i \]
\[ i := i + 1 \]

\[ \Rightarrow \quad \text{We get the exact result in this example (but not guaranteed)}! \]
Remarks

- Can use a work-list instead of a work-set
- Special iteration strategies in special situations
- Semi-naive iteration (later!)
- Narrowing operators
Three 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
Three 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

Execution Semantics

Abstraction

MOP-solution

sound?

MFP-solution

sound?

precise?

sound?

how precise?
\[ MOP[y] = \emptyset \cup \{ y \} = \{ y \} \]
Meet-Over-All-Paths Solution (MOP)

Definition:

The transfer function \( f_\pi : L \rightarrow L \) of a path \( \pi : v_0 f_0 \cdots f_{k-1} v_k, k \geq 0 \), is:

\[
f_\pi = f_{k-1} \circ \ldots \circ f_0.
\]

The MOP-solution is:

\[
\text{MOP}[v] = \biguplus \{ f_\pi(\text{init}) \mid \pi \in \text{Paths}[st,v] \} \text{ für alle } v \in \mathbb{N}.
\]
Coincidence Theorem

Definition:
A data-flow problem is positively-distributive if
\[
f(\biguplus X) = \biguplus \{ f(x) \mid x \in X \}
\]
for all sets \( \emptyset \neq X \subseteq L \) and transfer functions \( f \in F \).

Theorem:
For any instance of a positively-distributive data-flow problem:
\[
MOP[u] = MFP[u]
\]
for all program points \( u \) (if all program points reachable).

Remark:
A data-flow problem is positively-distributive if a) and b) hold:
(a) it is distributive: \( f(x \biguplus y) = f(x) \biguplus f(y) \) f.a. \( f \in F \), \( x, y \in L \).
(b) it is effective: the lattice \( L \) does not have infinite ascending chains.

Remark: All bitvector frameworks are distributive and effective.
Recall: Lattice for Constant Propagation

lattice $L: \{ \rho \mid \rho : \text{Var} \rightarrow (\mathbb{Z} \cup \{\top\})\} \cup \{\bot\}$

$\rho \sqsubseteq \rho' \iff \rho = \bot \lor$

$(\rho, \rho' \neq \bot \land \forall x: \rho(x) \sqsubseteq \rho'(x))$
(\(\rho(x), \rho(y), \rho(z)\))

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

Recall:

We assume transfer functions in a data-flow problem to be monotone i.e.:

\[ x \sqsubseteq y \implies f(x) \sqsubseteq f(y) \quad \text{for all } f \in F, x, y \in L. \]

Theorem:

For any data-flow problem:

\[ \text{MOP}[u] \sqsubseteq \text{MFP}[u] \quad \text{for all program points } u. \]
Assessing Data Flow Frameworks

Execution Semantics

Abstraction

MOP-solution

sound

MFP-solution

sound

precise, if distrib.
Where Flow Analysis Looses Precision

Execution semantics → MOP → MFP → Widening

Potential loss of precision
Three 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:
  \((D, \sqsubseteq) = (P(\text{Edges}^*), \subseteq)\) or \((D, \sqsubseteq) = (P(\text{Stmt}^*), \subseteq)\)

- sets of reaching states („Collecting Semantics“):
  \((D, \sqsubseteq) = (P(\Sigma^*), \subseteq)\) with \(\Sigma = \text{Var} \rightarrow \text{Val}\)
Assume a universally-disjunctive abstraction function $\alpha : D \rightarrow D^\#$.

**Correct abstract interpretation:**

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

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

**Correct and precise abstract interpretation:**

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

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

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

Constraint system for reaching runs:

\[ R[st] \supseteq \{ \varepsilon \}, \quad \text{for } st, \text{ the start node} \]
\[ R[v] \supseteq R[u] \cdot \{ \langle e \rangle \}, \quad \text{for each edge } e = (u,s,v) \]

Operational justification:

Let \( R[u] \) be components of smallest solution over \( P(\text{Edges}^*) \). Then

\[ R[u] = R^{op}[u] \overset{\text{def}}{=} \{ r \in \text{Edges}^* | st \xrightarrow{r} u \} \quad \text{for all } u \]

Prove:

a) \( R^{op}[u] \) satisfies all constraints \( \quad \text{(direct)} \)
\[ \Rightarrow R[u] \subseteq R^{op}[u] \quad \text{f.a. } u \]

b) \( w \in R^{op}[u] \Rightarrow w \in R[u] \quad \text{(by induction on } |w|) \)
\[ \Rightarrow R^{op}[u] \subseteq R[u] \quad \text{f.a. } u \]
Abstract Interpretation

Constraint system for reaching runs:

\[ R[st] \supseteq \{\varepsilon\}, \quad \text{for } st, \text{ the start node} \]
\[ R[v] \supseteq R[u] \cdot \{\langle e \rangle\}, \quad \text{for each edge } e = (u, s, v) \]

Derive the analysis:

Replace
\[ \{\varepsilon\} \quad \text{by } \text{init} \]
\[ \bullet \cdot \{\langle e \rangle\} \quad \text{by } f_e \]

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 \( \alpha_{\text{MOP}} : \mathcal{P}(\text{Edges}^+) \rightarrow L \) by

\[
\alpha_{\text{MOP}}(R) = \sqcup \{ f_r(\text{init}) \mid r \in R \} \quad \text{where} \quad f_\varepsilon = \text{ld}, \quad f_{s(\varepsilon)} = f_s \circ f_s
\]

Remark:
If all transfer functions \( f_e \) are monotone, the abstraction is correct, hence:

\[
\alpha_{\text{MOP}}(R[u]) \subseteq R^# [u] \quad \text{f.a. prg. points} \ u
\]

If all transfer function \( f_e \) are universally-distributive, i.e.,

\[
f(\sqcup X) = \sqcup \{ f(x) \mid x \in X \} \quad \text{for all sets} \ X \subseteq L
\]

the abstraction is correct and precise, hence:

\[
\alpha_{\text{MOP}}(R[u]) = R^# [u] \quad \text{f.a. prg. points} \ u
\]

Justifies MOP vs. MFP theorems (*cum grano salis*).
Overview

- Introduction
- Fundamentals of Program Analysis

Excursion 1

- Interprocedural Analysis
  Excursion 2

- Analysis of Parallel Programs
  Excursion 3

- Conclusion
Challenges for Automatic Analysis

- **Data aspects:**
  - infinite number domains
  - dynamic data structures (e.g. lists of unbounded length)
  - pointers
  - ...

- **Control aspects:**
  - recursion
  - concurrency
  - creation of processes / threads
  - synchronization primitives (locks, monitors, communication stmts ...)
  - ...

⇒ infinite/unbounded state spaces
Classifying Analysis Approaches

control aspects

data aspects

analysis techniques
(My) Main Interests of Recent Years

Data aspects:
- algebraic invariants over $\mathbb{Q}$, $\mathbb{Z}$, $\mathbb{Z}_m$ ($m = 2^n$) in sequential programs, partly with recursive procedures
- invariant generation relative to Herbrand interpretation

Control aspects:
- recursion
- concurrency with process creation / threads
- synchronization primitives, in particular locks/monitors

Technics:
- fixpoint-based
- automata-based
- (linear) algebra
- syntactic substitution-based techniques
- ...
Overview

- Introduction
- Fundamentals of Program Analysis
  Excursion 1
- Interprocedural Analysis
  Excursion 2
- Analysis of Parallel Programs
  Excursion 3
- Conclusion
A Note on Karr’s Algorithm

Markus Müller-Olm

Joint work with
Helmut Seidl (TU München)

ICALP 2004, Turku, July 12-16, 2004
What this Excursion is About...

0

\[ x_1 := 1 \]
\[ x_2 := 1 \]
\[ x_3 := 1 \]

1

\[ x_1 := x_1 + 1 \]
\[ x_2 := 2x_2 - 2x_1 + 5 \]
\[ x_3 := x_3 + x_2 \]

2

\[ x_3 = x_1^2 \]
\[ x_2 = 2x_1 - 1 \]
Affine Programs

- Basic Statements:
  - affine assignments: \( x_1 := x_1 - 2x_3 + 7 \)
  - unknown assignments: \( x_i := ? \)
    \[ \rightarrow \text{abstract too complex statements} \]

- Affine Programs:
  - control flow graph \( G = (N, E, st) \), where
    - \( N \) finite set of program points
    - \( E \subseteq N \times \text{Stmt} \times N \) set of edges
    - \( st \in N \) start node

- Note: non-deterministic instead of guarded branching
The Goal: Precise Analysis

Given an affine program, determine for each program point

- all valid affine relations:
  \[ a_0 + \sum a_i x_i = 0 \quad a_i \in \mathbb{Q} \]

More ambitious goal:
- determine all valid polynomial relations (of degree \( \leq d \)):
  \[ p(x_1, \ldots, x_k) = 0 \quad p \in \mathbb{Q}[x_1, \ldots, x_n] \]
Applications of Affine (and Polynomial) Relations

- Data-flow analysis:
  - definite equalities: \( x = y \)
  - constant detection: \( x = 42 \)
  - discovery of symbolic constants: \( x = 5yz + 17 \)
  - complex common subexpressions: \( xy + 42 = y^2 + 5 \)
  - loop induction variables

- Program verification
  - strongest valid affine (or polynomial) assertions (cf. Petri Net invariants)

- RS3:
  - Improve precision of PDG-based IFC analysis (with Gregor Snelting (KIT, Karlsruhe) and his group)
Karr’s Algorithm

- Determines valid affine relations in programs.

- **Idea:** Perform a data-flow analysis maintaining for each program point a set of affine relations, i.e., a linear equation system.

- **Fact:** Set of valid affine relations forms a vector space of dimension at most $k+1$, where $k = \#\text{program variables}$.

  $\Rightarrow$ can be represented by a basis.

  $\Rightarrow$ forms a complete lattice of height $k+1$.

[Karr, 1976]
Deficiencies of Karr’s Algorithm

- Basic operations are complex
  - „non-invertible“ assignments
  - union of affine spaces

- $O(n \cdot k^4)$ arithmetic operations
  - $n$ size of the program
  - $k$ number of variables

- Number may grow to exponential size
Our Contribution

- Reformulation of Karr’s algorithm:
  - basic operations are simple
  - $O(n \cdot k^3)$ arithmetic operations
  - numbers stay of polynomial length: $O(n \cdot k^2)$

Moreover:
- generalization to polynomial relations of bounded degree
- show, algorithm finds all affine relations in „affine programs“

- Ideas:
  - represent affine spaces by affine bases instead of lin. eq. syst.
  - use semi-naive fixpoint iteration
  - keep a reduced affine basis for each program point during fixpoint iteration
Affine Basis
Concrete Collecting Semantics

Smallest solution over subsets of $\mathbb{Q}^k$ of:

$V[st] \supseteq \mathbb{Q}^k$

$V[v] \supseteq f_s(V[u])$, for each edge $(u,s,v)$

where

$f_{x_i:=t}(X) = \{x[x_i \mapsto t(x)] \mid x \in X\}$

$f_{x_i:=?}(X) = \{x[x_i \mapsto c] \mid x \in X, c \in \mathbb{Q}\}$

First goal: compute affine hull of $V[u]$ for each $u$. 
Abstraction

Affine hull:

\[ \text{aff}(X) = \{ \sum \lambda_i x_i \mid x_i \in X, \lambda_i \in \mathbb{Q}, \sum \lambda_i = 1 \} \]

The affine hull operator is a closure operator:

\[ \text{aff}(X) \supseteq X, \text{aff}(\text{aff}(X)) = X, \ X \subseteq Y \Rightarrow \text{aff}(X) \subseteq \text{aff}(Y) \]

⇒ Affine subspaces of \( \mathbb{Q}^k \) ordered by set inclusion form a complete lattice:

\[ (D, \sqsubseteq) = \left( \{ X \subseteq \mathbb{Q}^k \mid \text{aff}(X) = X \}, \subseteq \right). \]

Affine hull is even a precise abstraction:

Lemma: \( f_s(\text{aff}(X)) = \text{aff}(f_s(X)) \).
Abstract Semantics

Smallest solution over \((D, \sqsubseteq)\) of:

\[ V^\#[st] \supseteq \mathbb{Q}^k \]
\[ V^\#[v] \supseteq f_s(V^\#[u]) , \text{ for each edge } (u,s,v) \]

Lemma: \( V^\#[u] = \text{aff}(V[u]) \) for all program points \( u \).
Basic Semi-naive Fixpoint Algorithm

\[
\text{forall } (v \in N) \ G[v] = \emptyset; \\
G[st] = \{0, e_1, \ldots, e_k\}; \\
W = \{(st,0),(st,e_1),\ldots,(st,e_k)\}; \\
\text{while } W \neq \emptyset \ { \\
 \quad (u,x) = \text{Extract}(W); \\
 \quad \text{forall } (s,v \text{ with } (u,s,v) \in E) \ { \\
 \quad \quad t = \left[ [s] \right] x; \\
 \quad \quad \text{if } (t \notin \text{aff}(G[v])) \ { \\
 \quad \quad \quad G[v] = G[v] \cup \{t\}; \\
 \quad \quad \quad W = W \cup \{(v,t)\}; \\
 \quad \quad \} \\
 \quad } \\
\}
\]
Example

0
\[ x_1 := 1 \]
\[ x_2 := 1 \]
\[ x_3 := 1 \]

1
\[ x_1 := x_1 + 1 \]
\[ x_2 := 2x_2 - 2x_1 + 5 \]
\[ x_3 := x_3 + x_2 \]

2

\begin{pmatrix}
0 \\
0 \\
0 \\
0 \\
\end{pmatrix}
\begin{pmatrix}
0 \\
0 \\
0 \\
1 \\
\end{pmatrix}
\begin{pmatrix}
1 \\
1 \\
1 \\
1 \\
\end{pmatrix}
\begin{pmatrix}
2 \\
3 \\
4 \\
3 \\
\end{pmatrix}
\begin{pmatrix}
3 \\
5 \\
5 \\
7 \\
\end{pmatrix}
\begin{pmatrix}
4 \\
4 \\
7 \\
16 \\
\end{pmatrix}

\epsilon \text{ aff } \left\{ \begin{pmatrix}
1 \\
1 \\
1 \\
\end{pmatrix}, \begin{pmatrix}
2 \\
3 \\
4 \\
\end{pmatrix}, \begin{pmatrix}
3 \\
5 \\
9 \\
\end{pmatrix} \right\}
Correctness

Theorem:
   a) Algorithm terminates after at most $nk + n$ iterations of the loop, where $n = |N|$ and $k$ is the number of variables.
   b) For all $v \in N$, we have $aff(G_{fin}[v]) = V^#[v]$.

Invariants for b)
   I1: $\forall v \in N : G[v] \subseteq V[v]$ and $\forall (u, x) \in W : x \in V[u]$.
   I2: $\forall (u, s, v) \in E : aff\left(G[v] \cup \{[[s]] x | (u, x) \in W\}\right) \equiv f_s(aff(G[u]))$. 
Complexity

Theorem:

a) The affine hulls $V^*[u] = \text{aff}(V[u])$ can be computed in time $O(n \cdot k^3)$, where $n = |N| + |E|$.

b) In this computation only arithmetic operations on numbers with $O(n \cdot k^2)$ bits are used.

Store diagonal basis for membership tests.
Propagate original vectors.
Point + Linear Basis
Example

\[ \begin{align*}
0 & : x_1 := 1 \\
1 & : x_2 := x_1 + 1 \\
2 & : x_3 := x_2 - 2x_1 + 5 \\
& : x_3 := x_3 + x_2
\end{align*} \]
Determining Affine Relations

Lemma: a is valid for $X \iff a$ is valid for $\text{aff}(X)$.

$\Rightarrow$ suffices to determine the affine relations valid for affine bases; can be done with a linear equation system!

Theorem:

a) The vector spaces of all affine relations valid at the program points of an affine program can be computed in time $O(n \cdot k^3)$.

b) This computation performs arithmetic operations on integers with $O(n \cdot k^2)$ bits only.
Example

\[ a_0 + a_1 x_1 + a_2 x_2 + a_3 x_3 = 0 \]
\[ \iff a_0 + 2a_1 + 3a_2 + 4a_3 = 0 \]
\[ 1a_1 + 2a_2 = 0 \]
\[ 2a_3 = 0 \]

\[ \iff a_0 = a_2, \ a_1 = -2a_2, \ a_3 = 0 \]

\[ \implies 2x_1 - x_2 - 1 \] is valid at 2

\[
\begin{align*}
\begin{pmatrix} 2 & 3 & 4 \\ 3 & 5 & 9 \\ 4 & 7 & 16 \end{pmatrix} & , \\
\begin{pmatrix} 1 & 2 \\ 0 & 0 \end{pmatrix} & \end{align*}
\]
Also in the Paper

- Non-deterministic assignments
- Bit length estimation
- Polynomial relations
- Affine programs + affine equality guards
  - validity of affine relations undecidable
End of Excursion 1
Overview

- Introduction
- Fundamentals of Program Analysis
  Excursion 1
- Interprocedural Analysis
  Excursion 2
- Analysis of Parallel Programs
  Excursion 3
- Conclusion
Interprocedural Analysis

Main:
- \( \text{R}() \)
- \( \text{P}() \)
  - \( \text{c} := \text{a} + \text{b} \)

P():
- \( \text{c} := \text{a} + \text{b} \)

R():
- \( \text{c} := \text{a} + \text{b} \)
- \( \text{a} := 7 \)
- \( \text{c} := \text{a} + \text{b} \)
- \( \text{a} := 7 \)

Q():
- \( \text{a} := 7 \)
- \( \text{c} := \text{a} + \text{b} \)

Call edges:
- \( \text{P}() \rightarrow \text{Q}() \)
- \( \text{Q}() \rightarrow \text{P}() \)

Procedures:
- \( \text{call} \)
- \( \text{recursion} \)
Running Example:
(Definite) Availability of the single expression \(a+b\)

The lattice:

\[
\begin{array}{c}
\text{false} & \text{a+b not available} \\
\mid & \\
\text{true} & \text{a+b available}
\end{array}
\]

Initial value: 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.

Main:

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

The lattice:

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]

\[ \text{false} \]

\[ \text{true} \]
Context-Insensitive Analysis

Conservative assumption: Information flows from each call node to entry of procedure and from exit of procedure back to return point.
Assume a universally-disjunctive abstraction function $\alpha : D \to D^\#$.

Correct abstract interpretation:

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

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

Correct and precise abstract interpretation:

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

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

Use this as a guideline for designing correct (and precise) analyses!
Example Flow Graph

Main:

$e_0 : c := a + b$

$e_1 : P()$

$e_2 : a := 7$

$e_3 : P()$

$e_4 : c := a + b$

The lattice:

false

true
Let’s Apply Our Abstract Interpretation Recipe: Constraint System for Feasible Paths

Operational justification:

\[
S(u) = \left\{ r \in \text{Edges}^* \mid st_p \xrightarrow{r} u \right\} \quad \text{for all } u \text{ in procedure } p
\]

\[
S(p) = \left\{ r \in \text{Edges}^* \mid st_p \xrightarrow{r} \varepsilon \right\} \quad \text{for all procedures } p
\]

\[
R(u) = \left\{ r \in \text{Edges}^* \mid \exists w \in \text{Nodes}^* : st_{\text{Main}} \xrightarrow{r} uw \right\} \quad \text{for all } u
\]

Same-level runs:

\[
S(p) \supseteq S(r_p) \quad \text{for all } p \quad r_p \text{ return point of } p
\]

\[
S(st_p) \supseteq \{ \varepsilon \} \quad \text{for all } p \quad st_p \text{ entry point of } p
\]

\[
S(v) \supseteq S(u) \cdot \{ \langle e \rangle \} \quad e = (u,s,v) \text{ base edge}
\]

\[
S(v) \supseteq S(u) \cdot S(p) \quad e = (u,p,v) \text{ call edge}
\]

Reaching runs:

\[
R(st_{\text{Main}}) \supseteq \{ \varepsilon \} \quad \text{for all } p \quad st_{\text{Main}} \text{ entry point of } \text{Main}
\]

\[
R(v) \supseteq R(u) \cdot \{ \langle e \rangle \} \quad e = (u,s,v) \text{ basic edge}
\]

\[
R(v) \supseteq R(u) \cdot S(p) \quad e = (u,p,v) \text{ call edge}
\]

\[
R(st_p) \supseteq R(u) \quad e = (u,p,v) \text{ call edge, } st_p \text{ entry point of } p
\]
Context-Sensitive Analysis

Summary-based approaches:

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

Classic types of summary information:

Functional approach: [Sharir/Pnueli 81, Knoop/Steffen: CC´92]
Use (monotonic) functions on data flow informations!

Relational approach: [Cousot/Cousot: POPL´77]
Use relations (of a representable class) on data flow informations!

Call-string-based approaches: e.g [Sharir/Pnueli 81], [Khedker/Karkare: CC´08]
- Analysis relative to finite portion of call stack
- Applicable to arbitrary lattices
- Sometimes less precise than summary-based approaches
Formalization of Functional Approach

Abstractions:

Abstract same-level runs with $\alpha_{\text{Funct}} : \text{Edges}^* \rightarrow (L \rightarrow L)$:

$$\alpha_{\text{Funct}}(R) = \{ f_r \mid r \in R \}$$

for $R \subseteq \text{Edges}^*$

Abstract reaching runs with $\alpha_{\text{MOP}} : \text{Edges}^* \rightarrow L$:

$$\alpha_{\text{MOP}}(R) = \{ f_r(\text{init}) \mid r \in R \}$$

for $R \subseteq \text{Edges}^*$

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

$$S^#(p) \sqsubseteq S^#(r_p)$$

$r_p$ return point of $p$

$$S^#(st_p) \sqsubseteq \text{id}$$

$st_p$ entry point of $p$

$$S^#(v) \sqsubseteq f_e^# \circ S^#(u)$$

$e = (u,s,v)$ base edge

$$S^#(v) \sqsubseteq S^#(p) \circ S^#(u)$$

$e = (u,p,v)$ call edge

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

$$R^#(st_{\text{Main}}) \sqsubseteq \text{init}$$

$st_{\text{Main}}$ entry point of $\text{Main}$

$$R^#(v) \sqsubseteq f_e^#(R^#(u))$$

$e = (u,s,v)$ basic edge

$$R^#(v) \sqsubseteq S^#(p)(R^#(u))$$

$e = (u,p,v)$ call edge

$$R^#(st_p) \sqsubseteq R^#(u)$$

$e = (u,p,v)$ call edge, $st_p$ entry point of $p$
Observations:

Just three monotone functions on lattice $L$:

- $\lambda x . \text{false}$
- $\lambda x . x$
- $\lambda x . \text{true}$

$f \circ g : L \rightarrow L$:

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

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

Main:

\[ a = a + b \]

\[ P() \rightarrow Q() \rightarrow R() \]

\[ P() \rightarrow Q() \rightarrow R() \]

\[ P: c = a + b \]

\[ R() \]

\[ R: c = a + b, a = 7 \]

\[ Q: P() \]

\[ \text{the lattice:} \]

\[ k \rightarrow i \rightarrow g \]
Context-Sensitive Analysis, 2. Phase

Main:

P() → false
Q() → true
R() → true

P: i

Q: k

R: g

the lattice:

false
true
Functional Approach

Theorem:

Correctness: For any monotone framework:
\[ \alpha_{\text{MOP}}(R[u]) \subseteq R^#[u] \quad \text{f.a. } u \]

Completeness: For any universally-distributive framework:
\[ \alpha_{\text{MOP}}(R[u]) = R^#[u] \quad \text{f.a. } u \]

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.
Overview

- Introduction
- Fundamentals of Program Analysis
  - Excursion 1
- Interprocedural Analysis
  - Excursion 2
- Analysis of Parallel Programs
  - Excursion 3
- Conclusion
Excursion Mainly Based On ...

MMO + Helmut Seidl (TU München)
Precise Interprocedural Analysis through Linear Algebra, POPL 2004

MMO + Helmut Seidl (TU München)
Analysis of Modular Arithmetic
ESOP 2005 + TOPLAS 2007
Finding Invariants...

Main:

0

1

x₁ := x₂

2

x₃ := 0

3

x₁ - x₂ - x₃ = 0

P()

4

x₁ := x₁ - x₂ - x₃

5

x₁ = 0

P:

6

x₃ := x₃ + 1

7

x₁ := x₁ + x₂ + 1

P()

8

x₁ := x₁ - x₂

9

x₁ - x₂ - x₃ - x₂ x₃ = 0
... 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
- improving PDG-based IFC analysis (with G. Snelting’s group, KIT)
- ...
A Program Abstraction

Affine programs:

- affine assignments: \( x_1 := x_1 - 2x_3 + 7 \)

- unknown assignments: \( x_i := ? \)
  \[ \rightarrow \quad \text{abstract too complex statements!} \]

- 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:
  \[ a_0 + \sum a_i x_i = 0 \quad a_i \in R \]

- determine all valid polynomial relations (of degree $\leq d$):
  \[ p(x_1,\ldots,x_k) = 0 \quad p \in R[x_1,\ldots,x_n] \]

... and all this in polynomial time (unit cost measure) !!!
Infinity Dimensions

push-down

arithmetic
Use a Standard Approach for Interprocedural Generalization of Karr’s Algo?

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

Relational approach [Cousot/Cousot, 1977]
- Idea: summarize each procedure by approximation of I/O relation
- Problem: not exact

Call-string approach [Sharir/Pnueli, 1981], [Khedker/Karkare: CC´08]
- Idea: take a finite piece of the run-time stack into account
- Problem: not exact
Relational Analysis is Not Strong Enough

Main:

0

\[ x := 1 \]

1

\[ x := 1 \]

2

P()

\[ x = 1 \]

P:

3

\[ x := 2 \cdot x \]

4

\[ x := x \]

5

\[ x := x - 1 \]

True relational semantics of P:

$x_{post}$

2

1

0

1

2

3

$x_{pre}$

Best affine approximation:
Towards the Algorithm ...
Concrete Semantics of an Execution Path

- Every execution path $\pi$ induces a linear transformation on extended program states:

$$\pi$$

$$\left[\begin{array}{c}
x_1 := 2x_1 + 3x_2 + 4, \\
x_2 := 5x_1 + 6
\end{array}\right](v)

= \left[\begin{array}{c}
x_2 := 5x_1 + 6
\end{array}\right]\left(\left[\begin{array}{c}
x_1 := 2x_1 + 3x_2 + 4
\end{array}\right](v)\right)

= \left[\begin{array}{ccc}
1 & 0 & 0 \\
0 & 1 & 0 \\
6 & 5 & 0
\end{array}\right]\left[\begin{array}{ccc}
1 & 0 & 0 \\
4 & 2 & 3 \\
0 & 0 & 1
\end{array}\right](v)

= \left[\begin{array}{ccc}
1 & 0 & 0 \\
4 & 2 & 3 \\
26 & 10 & 15
\end{array}\right](v)

\text{with } v = \left[\begin{array}{c}
1 \\
\nu_1 \\
\nu_2
\end{array}\right]

\left[\pi\right], \text{ the transformation matrix for path } \pi
Observation 1

Extended states ...
- ... form a vector space of dimension $k+1 = O(k)$.
- Subspaces form a lattice of height $k+1 = O(k)$

Transformation matrices ...
- ... form a vector space of dimension $k \cdot (k+1)+1 = O(k^2)$.
- Subspaces from a complete lattice of height $k \cdot (k+1)+1 = O(k^2)$. 
Observation 2

Affine relation is valid for a set $M$ of extended states iff it is valid for span $M$, i.e.:

$$a_0 + a_1 v_1 + ... + a_k v_k \quad \text{for all} \quad \begin{pmatrix} 1 \\ v_1 \\ v_2 \end{pmatrix} \in V$$

$$\Leftrightarrow a_0 v_0 + a_1 v_1 + ... + a_k v_k \quad \text{for all} \quad \begin{pmatrix} v_0 \\ v_1 \\ v_2 \end{pmatrix} \in \text{span } V$$

$\Rightarrow$ Suffices to compute span of reaching states !! (cf. Excursion 1)
Observation 3

\[
\text{span}\ \{Mv \mid M \in P, \ v \in V\}
\]
\[
= \text{span}\ \{Mv \mid M \in \text{span}P, \ v \in \text{span}V\}
\]

$\Rightarrow$ **Span of transformation matrices** of paths through procedure can be used as precise summary !!
Let’s Apply Our Abstract Interpretation Recipe: Constraint System for Feasible Paths

Operational justification:

\[ S(u) = \{ r \in \text{Edges}^* \mid \text{begin entry point of } p \rightarrow u \} \] for all \( u \) in procedure \( p \)

\[ S(p) = \{ r \in \text{Edges}^* \mid \text{begin entry point of } p \rightarrow \epsilon \} \] for all procedures \( p \)

\[ R(u) = \{ r \in \text{Edges}^* \mid \exists \omega \in \text{Nodes}^* : \text{begin entry point of } Main \rightarrow u \omega \} \] for all \( u \)

Same-level runs:

\[ S(p) \supseteq S(r_p) \] \( r_p \) return point of \( p \)

\[ S(st_p) \supseteq \{ \epsilon \} \] \( st_p \) entry point of \( p \)

\[ S(v) \supseteq S(u) \cdot \{ (e) \} \] \( e = (u,s,v) \) base edge

\[ S(v) \supseteq S(u) \cdot S(p) \] \( e = (u,p,v) \) call edge

Reaching runs:

\[ R(st_{\text{Main}}) \supseteq \{ \epsilon \} \] \( st_{\text{Main}} \) entry point of \( Main \)

\[ R(v) \supseteq R(u) \cdot \{ (e) \} \] \( e = (u,s,v) \) basic edge

\[ R(v) \supseteq R(u) \cdot S(p) \] \( e = (u,p,v) \) call edge

\[ R(st_p) \supseteq R(u) \] \( e = (u,p,v) \) call edge, \( st_p \) entry point of \( p \)
Algorithm for Computing Affine Relations

1) Compute (for each prg. point and procedure u) a basis $B$ with:
   \[ \text{Span } B = \text{Span } \{ \pi \mid \pi \in S(u) \} \]
   by a precise abstract interpretation.

2) Compute (for each prg. point u) a basis $B$ with:
   \[ \text{Span } B = \text{Span } \{ \pi(v) \mid \pi \in R(u), v \in \mathbb{F}^{k+1} \} \]
   by a precise abstract interpretation.

3) Solve the linear equation system:
   \[ a_0 v_0 + a_1 v_1 + \ldots + a_k v_k = 0 \quad \text{for all } (v_0, \ldots, v_k)^T \in R(u) \]
Constraint Systems obtained by Abstract Interpretation

Same-level runs:
\[ S(p) \sqsupseteq S(r_p) \quad r_p \text{ return point of } p \]
\[ S(st_p) \sqsupseteq \text{span } \{ I \} \quad st_p \text{ entry point of } p, \ I \text{ identity matrix} \]
\[ S(v) \sqsupseteq \{ [e] \} \cdot S(u) \quad e = (u,s,v) \text{ base edge, } \cdot \text{ matrix product lifted to sets} \]
\[ S(v) \sqsupseteq S(u) \cdot S(p) \quad e = (u,p,v) \text{ call edge} \]

Reaching runs:
\[ R(st_{Main}) \sqsupseteq F^{k+1} \quad st_{Main} \text{ entry point of } Main \]
\[ R(v) \sqsupseteq \{ [e] v \mid v \in R(u) \} \quad e = (u,s,v) \text{ basic edge} \]
\[ R(v) \sqsupseteq \{ Mv \mid M \in S(p), v \in R(u) \} \quad e = (u,p,v) \text{ call edge} \]
\[ R(st_p) \sqsupseteq R(u) \quad e = (u,p,v) \text{ call edge, } st_p \text{ entry point of } p \]

All computations can be and are performed on bases !
Theorem

In an affine program:

- We can compute bases for the following vector spaces:
  \[ \alpha_S(S(u)) = \text{Span} \{ \pi \mid \pi \in S(u) \} \quad \text{for all } u. \]
  \[ \alpha_R(R(u)) = \text{Span} \{ \pi \cdot v \mid \pi \in R(u), v \in \mathbb{F}^{k+1} \} \quad \text{for all } u. \]

- The vector spaces
  \[ \{ a \in \mathbb{F}^{k+1} \mid \text{affine relation } a \text{ is valid at } u \} \]
  can be computed precisely for all prg. points \( u \).

- The time complexity is \textbf{linear} in the program size and \textbf{polynomial} in the number of variables
  \[ O(n \cdot k^8) \]
  (\( n \) size of the program, \( k \) number of variables)
An Example

Main:

0
\( x_1 := x_2 \)

1
\( x_3 := 0 \)

2

P()

3
\( x_1 := x_1 - x_2 - x_3 \)

4

P:

0
\( x_1 := x_1 + x_2 + 1 \)

1
\( x_3 := x_3 + 1 \)

2
\( x_1 := x_1 - x_2 \)

3

P()

4

⇒ stable!
An Example

Main:
0 → 1
  x₁ := x₂
  x₃ := 0

1 → 2
  x₃ := 0

2 → 3
  x₁ := x₁ - x₂ - x₃

3 → 4
  x₁ := x₁ - x₂

4 → 0
  x₁ := x₂
  x₃ := x₃ + 1

P:
0 → 1
  x₃ := x₃ + 1

1 → 2
  x₁ := x₁ + x₂ + 1

2 → 3
  x₃ := x₃ + 1

3 → 4
  x₁ := x₁ + x₂ + 1

4 → 0
  x₁ := x₂
  x₃ := x₃ + 1

⇒ stable!
An Example

Main:

$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

$x_1 := x_2$

$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

$x_3 := 0$

$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

$P()$

$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 2 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 2 & 0 \end{pmatrix}$

$x_1 := x_1 - x_2 - x_3$

$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 2 & 0 \end{pmatrix}$

$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 2 & 0 \end{pmatrix}$

$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 2 & 0 \end{pmatrix}$
An Example

\[ a_0 + a_1 x_1 + a_2 x_2 + a_3 x_3 = 0 \] is valid at 3

\[
\begin{pmatrix}
1 & 0 & 0 & 0 & a_0 \\
1 & 2 & 0 & 2 & a_1 \\
1 & 1 & 1 & 0 & a_2 \\
1 & 3 & 1 & 2 & a_3
\end{pmatrix}
= 0
\]

\[ a_0 = 0 \land a_2 = a_3 = -a_1 \]

Just the affine relations of the form

\[ a x_1 - a x_2 - a x_3 = 0 \quad (\text{for } a \in \mathbb{F}) \]

are valid at 3, in particular:

\[ x_1 - x_2 - x_3 = 0 \]
Extensions

Also in the papers:

- Local variables, value parameters, return values
- Computing polynomial relations of bounded degree
- Affine pre-conditions
- Computing over modular rings (e.g. modulo $2^w$) or PIR
End of Excursion 2
Overview

- Introduction
- Fundamentals of Program Analysis
  - Excursion 1
- Interprocedural Analysis
  - Excursion 2
- Analysis of Parallel Programs
  - Excursion 3
- Conclusion
Interprocedural Analysis of Parallel Programs

Main:
\[ c := a + b \]

\[ P() \]

\[ Q() \| P() \]

\[ R() \]

P:
\[ c := a + b \]

Q:
\[ a := 7 \]
\[ c := a + b \]

P:
\[ a := 7 \]
\[ c := a + b \]

R:
\[ R() \| Q() \]

parallel call edge
Interleaving- Operator $\otimes$
(Shuffle-Operator)

Example:

\[
\langle a, b \rangle \otimes \langle x, y \rangle = \{ \langle a, b, x, y \rangle, \langle a, x, b, y \rangle, \langle a, x, y, b \rangle, \langle x, a, b, y \rangle, \langle x, a, y, b \rangle, \langle x, y, a, b \rangle \}
\]
Constraint System for Same-Level Runs

Operational justification:

\[ S(u) = \left\{ r \in \text{Edges}^* \mid st_p \stackrel{r}{\rightarrow} u \right\} \quad \text{for all } u \text{ in procedure } p \]
\[ S(p) = \left\{ r \in \text{Edges}^* \mid st_p \stackrel{r}{\rightarrow} \varepsilon \right\} \quad \text{for all procedures } p \]

Same-level runs:

\[ S(p) \supseteq S(r_p) \quad r_p \text{ return point of } p \]
\[ S(st_p) \supseteq \{\varepsilon\} \quad st_p \text{ entry point of } p \]
\[ S(v) \supseteq S(u) \cdot \langle\{e\}\rangle \quad e = (u, s, v) \text{ base edge} \]
\[ S(v) \supseteq S(u) \cdot S(p) \quad e = (u, p, v) \text{ call edge} \]
\[ S(v) \supseteq S(u) \cdot (S(p_0) \otimes S(p_1)) \quad e = (u, p_0 \parallel p_1, v) \text{ parallel call edge} \]

[Seidl/Steffen: ESOP 2000]
Constraint System for a Variant of Reaching Runs

Operational justification:

\[
R(u,q) = \{ r \in \text{Edges}^* | \exists c \in \text{Config} : st_q \xrightarrow{r} c, \text{At}_u(c) \}
\]

for program point \( u \) and procedure \( q \)

\[
P(q) = \{ r \in \text{Edges}^* | \exists c \in \text{Config} : st_q \xrightarrow{r} c \}
\]

Reaching runs:

\[
R(u,q) \supseteq S(u) \quad \text{u program point in procedure } q
\]
\[
R(u,q) \supseteq S(v) \cdot R(u,p) \quad \text{e} = (v,p,\_\_\_) \text{ call edge in proc. } q
\]
\[
R(u,q) \supseteq S(v) \cdot (R(u,p_i) \otimes P(p_{i-1})) \quad \text{e} = (v,p_o || p_{i-1},\_\_\_) \text{ parallel call edge in proc. } q, \; i = 0,1
\]

Interleaving potential:

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

**(Shuffle-Operator)**

Example:

\[
\langle a,b \rangle \otimes \langle x,y \rangle = \left\{ \langle a,b,x,y \rangle, \langle a,x,b,y \rangle, \langle a,x,y,b \rangle, \langle x,a,b,y \rangle, \langle x,a,y,b \rangle, \langle x,y,a,b \rangle \right\}
\]

The only new ingredient:

interleaving operator \( \otimes \) must be abstracted !

😊
Case: Availability of Single Expression

[Seidl/Steffen: ESOP 2000]

Abstract shuffle operator:

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

\[ f_1 \otimes^\# f_2 := f_1 \cdot f_2 \sqcup f_2 \cdot f_1 \]

Main lemma:

\[ \forall f_j \in \{g,k,i\} : f_1 \circ \ldots \circ f_{j+1} \circ \ldots \circ f_j = f_j \]

Treat other (separable) bitvector problems analogously...

\[ \Rightarrow \text{precise interprocedural analyses for all bitvector problems!} \]
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 (generate)

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

**Kill Potential:**

\[
KP(p) \sqsupseteq T \quad \text{if } p \text{ contains reachable edge } e \text{ with } f_e = k
\]

\[
KP(p) \sqsupseteq KP(q) \quad \text{if } p \text{ calls } q, q \parallel _, \text{ or } _\parallel q \text{ at some reachable edge}
\]

**Possible Interference:**

\[
PI(p) \sqsupseteq PI(q) \quad \text{if } q \text{ contains reachable call to } p
\]

\[
PI(p_i) \sqsupseteq PI(q) \sqcup KP(p_{i-1}) \quad \text{if } q \text{ contains reachable parallel call } p_0 \parallel p_1, \ i = 0,1
\]

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

\[
R^*(st_{Main}) \sqsupseteq \text{init} \quad \text{st}_{Main} \text{ entry point of } Main
\]

\[
R^*(v) \sqsupseteq f_e(R^*(u)) \quad e = (u,s,v) \text{ basic edge}
\]

\[
R^*(v) \sqsupseteq S^*(p)(R^*(u)) \quad e = (u,p,v) \text{ call edge}
\]

\[
R^*(st_p) \sqsupseteq R^*(u) \quad e = (u,p,v) \text{ call edge, } st_p \text{ entry point of } p
\]

\[
R^*(v) \sqsupseteq PI(p) \quad v \text{ reachable prg. point in } p
\]
Overview

- Introduction
- Fundamentals of Program Analysis
  Excursion 1
- Interprocedural Analysis
  Excursion 2
- Analysis of Parallel Programs
  Excursion 3
- Conclusion
Precise Fixpoint-Based Analysis of Programs with Thread-Creation and Procedures

Markus Müller-Olm
Westfälische Wilhelms-Universität Münster

Joint work with:
Peter Lammich
[same place]

CONCUR 2007
(My) Main Interests of Recent Years

Data aspects
- algebraic invariants over $\mathbb{Q}$, $\mathbb{Z}$, $\mathbb{Z}_m$ ($m = 2^n$) in sequential programs, partly with recursive procedures
- invariant generation relative to Herbrand interpretation

Control aspects
- recursion
- concurrency with process creation / threads
- synchronization primitives, in particular locks/monitors

Technics used
- fixpoint-based
- automata-based
- (linear) algebra
- syntactic substitution-based techniques
- ...
Another Program Model

Procedures

Recursive procedure calls

Basic actions

Spawn commands

Entry point, \( e_q \), of \( Q \)

Return point, \( x_q \), of \( Q \)

Spawning and calling procedures

- **P:**
  - \( 0 \): A
  - \( 1 \): call P
  - \( 2 \): B
  - \( 3 \): spawn Q

- **Q:**
  - \( 4 \): C
  - \( 5 \): call Q
  - \( 6 \): D
  - \( 7 \): Return point, \( x_q \), of \( Q \)
Spawns are Fundamentally Different

P:  
0
  ↓ A
  → 1
  ↓ call P
  → 2
  ↓ B
  → 3

Q:  
4
  ↓ C
  → 5
  ↓ call Q
  → 6
  ↓ D
  → 7

P induces trace language: \[ L = \bigcup \{ A^n \cdot (B^m \otimes (C^i \cdot D^j)) \mid n \geq m \geq 0, i \geq j \geq 0 \} \]

Cannot characterize L by constraint system with “•” and “⊗”.

[Bouajjani, MO, Touili: CONCUR 2005]
Gen/Kill-Problems

- Class of simple but important DFA problems

- Assumptions:
  - Lattice \((L, \sqsubseteq)\) is distributive
  - Transfer functions have form \(f_e(l) = (l \sqcap \text{kill}_e) \cup \text{gen}_e\) with \(\text{kill}, \text{gen} \in L\)

- Examples:
  - bitvector problems, e.g.
  - available expressions, live variables, very busy expressions, ...
Data Flow Analysis

Goal:

Compute, for each program point $u$:

- **Forward analysis:**
  \[ MOP^F[u] = \alpha^F(\text{Reach}[u]) \], where
  \[ \alpha^F(X) = \{ f_w(x_0) \mid w \in X \} \]

- **Backward analysis:**
  \[ MOP^B[u] = \alpha^B(\text{Leave}[u]) \], where
  \[ \alpha^B(X) = \{ f_w(\bot) \mid w^R \in X \} \]

Reach\$u\$ = \{ $w \mid \exists c : \{ e_{\text{Main}} \} \xrightarrow{w} c \land at_u(c) \}$

Leave\$u\$ = \{ $w \mid \exists c : \{ e_{\text{Main}} \} \xrightarrow{\ast} c \xrightarrow{w} \bot \land at_u(c) \}$

\[ at_u(c) \iff \exists w : (uw) \in c \]

\[ f_w = f_{e_n} \circ \cdots \circ f_{e_1}, \text{ for } w = e_1 \cdots e_n \]
Data Flow Analysis

Goal:
Compute, for each program point $u$:

- Forward analysis: $MOP^F[u] = \alpha^F(\text{Reach}[u])$, where $\alpha^F(X) = \sqcup \{ f_w(x_0) | w \in X \}$
- Backward analysis: $MOP^B[u] = \alpha^B(\text{Leave}[u])$, where $\alpha^B(X) = \sqcup \{ f_w(\bot) | w^R \in X \}$

Problem for programs with threads and procedures:

We cannot characterize Reach[$u$] and Leave[$u$] by a constraint system with operators „concatenation“ and „interleaving“.
One Way Out

- Derive alternative characterization of MOP-solution:
  - reason on level of execution paths
  - exploit properties of gen/kill-problems

- Characterize the path sets occurring as least solutions of constraint systems

- Perform analysis by abstract interpretation of these constraint systems

[Lammich/MO: CONCUR 2007]
Forward Analysis
Directly Reaching Paths and Potential Interleaving

**Reaching path:** a suitable interleaving of the red and blue paths

**Directly reaching path:** the red path

**Potential interference:** set of edges in the blue paths (note: no order information!)

Formalization by augmented operational semantics with markers (see paper)
Forward MOP-solution

**Theorem:** For gen/kill problems:

\[ \text{MOP}^F[u] = \alpha^F(\text{DReach}[u]) \sqcup \alpha^\text{Pl}(\text{Pl}[u]), \]

where \( \alpha^\text{Pl}(X) = \sqcup \{ \text{gen}_e \mid e \in X \} \).

**Remark**

- DReach[u] and Pl[u] can be characterized by constraint systems (see paper)
- \( \alpha^F(\text{DReach}[u]) \) and \( \alpha^\text{Pl}(\text{Pl}[u]) \) can be computed by an abstract interpretation of these constraint systems
Characterizing Directly Reaching Paths

Same level paths:

- \([\text{init}] \quad S[e_q] \supseteq \{\varepsilon\}\)
  for \(q \in P\)

- \([\text{base}] \quad S[v] \supseteq S[u]; e\)
  for \(e = (u, \text{base } _, v) \in E\)

- \([\text{call}] \quad S[v] \supseteq S[u]; e; S[r_q]; \text{ret}\)
  for \(e = (u, \text{call } q, v) \in E\)

- \([\text{spawn}] \quad S[v] \supseteq S[u]; e\)
  for \(e = (u, \text{spawn } q, v) \in E\)

Directly reaching paths:

- \([\text{init}] \quad R[e_{\text{main}}] \supseteq \{\varepsilon\}\)
  for \(u \in N_p\)

- \([\text{reach}] \quad R[u] \supseteq R[e_p]; S[u]\)
  for \(e = (u, \text{call } q, _) \in E\)

- \([\text{callp}] \quad R[e_q] \supseteq R[u]; e\)
  for \(e = (u, \text{spawn } q, _) \in E\)

- \([\text{spawnp}] \quad R[e_q] \supseteq R[u]; e\)
  for \(e = (u, \text{spawn } q, _) \in E\)
Backwards Analysis
Directly Leaving Paths and Potential Interleaving

Leaving path: a suitable interleaving of orange, black and parts of blue paths

Directly leaving path: a suitable interleaving of orange and black paths

Potential interference: the edges in the blue paths

Formalization by augmented operational semantics with markers (see paper)
Interleaving from Threads created in the Past

**Theorem:** For gen/kill problems:

\[ \text{MOP}^B[u] = \alpha^B(\text{DLeave}[u]) \sqcup \alpha^\Pi(\text{PI}[u]), \]

where \( \alpha^\Pi(E) = \sqcup \{ \text{gen}_e \mid e \in E \} \).

**Remark**

- We know no simple characterization of \( \text{DLeave}[u] \) by a constraint system.

- Main problem: Threads generated in a procedure instance survive that instance.
Representative Directly Leaving Paths

A representative directly leaving path:

1 2 3 4 5

. . . . . . . . .
Interleaving from Threads created in the Future

Lemma
\[ \alpha^B(DLeave[u]) = \alpha^B(RDLeave[u]) \] (for gen/kill problems).

Corollary
\[ MOP^B[u] = \alpha^B(RDLeave[u]) \sqcup \alpha^{PI}(PI[u]) \] (for gen/kill problems).

Remark
- RDLeave[u] and PI[u] can be characterized by constraint systems (see paper)
- \( \alpha^B(RDLeave[u]) \) and \( \alpha^{PI}(PI[u]) \) can be computed by an abstract interpretation of these constraint systems
Also in the Paper

- Formalization of these ideas
  - constraint systems for path sets
  - validation with respect to operational semantics

- Parallel calls in combination with threads
  - threads become trees instead of stacks ...

- Analysis of running time:
  - global information in time linear in the program size
Summary

- Forward- and backward gen/kill-analysis for programs with threads and procedures
- More efficient than automata-based approach
- More general than known fixpoint-based approach
- Recent work: Precise analysis in presence of locks/monitors (see papers at SAS 2008, CAV 2009, VMCAI 2011)
End of Excursion 3
Conclusion

- Program analysis very broad topic
- Provides generic analysis techniques for (software) systems
- Here just one path through the forest
- Many interesting topics not covered
Thank you!