Program Analysis of Sequential and Parallel Programs

Markus Müller-Olm
Westfälische Wilhelms-Universität Münster, Germany

14th Estonian Winter School in Computer Science
Palmse, Estonia, March, 1-6, 2009
(Automatic) Program Analysis

● What?
  ● Compute (or prove) properties of programs automatically without executing the program: „static analysis“

● Why?
  ● Validation/verification/debugging:
    ● find errors in programs
    ● guarantee correctness
    ● check programs from untrusted source (e.g. Internet)
    ● ...
  
  ● Programming systems:
    ● optimizing compilers
    ● IDEs (integrated development environments)
    ● CASE tools (CASE = Computer Aided Software Engineering)
    ● ...

...
Dream of Automatic Analysis

program

```c
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);
}
```

specification of property

\[ G(\Phi \rightarrow \Box \Psi) \]
Dream of Automatic Analysis

program

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);
}

analyzer for a fixed property

G(Φ → FΨ)

result
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:
    - type checking, flow analysis, abstract interpretation, ...
  - 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:
    - type checking, flow analysis, abstract interpretation, ...
  - Analyse weaker formalism:
    - model checking, reachability analysis, equivalence- or preorder-checking, ...
Overview

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

Apology for not giving proper credit in these lectures!
Overview

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

Live Variables

Dead Variables

x:=x+1

y:=17

y>63

¬(y>63)

x=17

y=x+y

x=x+42

y<99

¬(y<99)

x=y+1

y:=11

x:=x+1

x:=y+1

y:=y+1

x:=x+1

x:=10

y live

y live

x live

x dead
Live Variables Analysis

1. $x = x + 42$
2. $y > 63$
3. $y := 17$
4. $x := y + 1$
5. $x := x + 1$
6. $y < 99$
7. $y = x + y$
8. $y < 99$
9. $x = y + 1$
10. $x := y + 1$
11. $y := 11$

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: $y = x + y$
Node 8: $y < 99$
Node 9: $x = y + 1$
Node 10: $x := y + 1$
Node 11: $y := 11$
Remarks on Data-Flow Analysis

- Forward vs. backward analyses

- (Separable) bitvector analyses
  - forward: reaching definitions, available expressions, ...
  - backward: live/dead variables, very busy expressions, ...
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 \((L,\sqsubseteq)\):
- \(\bigwedge X\) exists for all \(X \subseteq L\): \[\bigwedge X = \sqcup \{x \in L \mid x \sqsubseteq X\}\]
- least element \(\bot\) exists: \[\bot = \sqcup L = \bigwedge \emptyset\]
- greatest element \(\top\) exists: \[\top = \sqcup \emptyset = \bigwedge L\]

Example:
- for any set \(A\) let \(P(A) = \{X \mid X \subseteq A\}\) (power set of \(A\)).
- \((P(A),\subseteq)\) 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:
- \( (P(\text{Var}), \subseteq) \) with \( \text{Var} = \) 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:

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

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

\[
f_e : P(\text{Var}) \rightarrow P(\text{Var}), \quad f_e(x) = x \setminus \text{kill}_e \cup \text{gen}_e, \quad \text{with}
\]

- \(\text{kill}_e = \) variables assigned at \(e\)
- \(\text{gen}_e = \) 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?
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.
- \(f : L \rightarrow L\) 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,
\[
\text{lfp}(f) = \sqcap \{ x \in L \mid f(x) \sqsubseteq x \} \quad \text{least pre-fixpoint}
\]
\[
\text{gfp}(f) = \sqcup \{ x \in L \mid x \sqsubseteq f(x) \} \quad \text{greatest post-fixpoint}
\]
Knaster-Tarski Fixpoint Theorem

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

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

0 \[ x=17 \]

1 \[ y=63 \] \[ \neg (y>63) \]

2 \[ \emptyset \]

3 \[ y:=17 \]

4 \[ x:=10 \]

5 \[ x=x+42 \]

6 \[ y<99 \]

7 \[ y=x+y \]

8 \[ \emptyset \]

10 \[ y:=11 \]

11 \[ x:=y+1 \]

x:=x+1
Invariants of the Main Loop

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

b1) $A[\text{fin}] \supseteq \text{init}$

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

If and when workset algorithm terminates:

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

$\Rightarrow A[u] \supseteq \text{MFP}[u]$ f.a. $u$

Hence, with a): $A[u] = \text{MFP}[u]$ f.a. $u$
How to Guarantee Termination

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

- Lattice \((L, \sqsubseteq)\) has no infinite ascending chains
  \(\Rightarrow\) algorithm terminates

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

\[ \nabla : L \times L \to L \text{ is called a } \textit{widening operator} \quad \text{iff} \]

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

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

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

stabilizes eventually.
Workset-Algorithm with Widening

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

Invariants of the Main Loop

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

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

b2) \( v \notin W \Rightarrow A[u] \sqsubseteq f_e(A[v]) \) 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)

\[ A[u] \sqsubseteq MFP[u] \] 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:

\[
\text{for } (i=0; \ i<42; \ i++) \\
\quad \text{if } (0\leq i \text{ and } i<42) \\
\quad \{ \\
\quad \quad A1 = A+i; \\
\quad \quad M[A1] = i; \\
\quad \} \\
\]

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

The lattice...

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

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

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

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

⇒ Result is far too imprecise!

Example taken from: H. Seidl, Vorlesung „Programmoptimierung“
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 (but not guaranteed) ! ☺
Remarks

- Can use work-list instead of work-set
- Special iteration strategies in special situations
- Semi-naive iteration
Program Analysis of Sequential and Parallel Programs II

Markus Müller-Olm
Westfälische Wilhelms-Universität Münster, Germany

14th Estonian Winter School in Computer Science
Palmse, Estonia, March, 1-6, 2009
Recall: Specifying Live Variables Analysis by a Constraint System

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

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

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

\[f_e: P(\text{Var}) \to P(\text{Var}), \quad f_e(x) = x \setminus \text{kill}_e \cup \text{gen}_e, \quad \text{with}\]

- \(\text{kill}_e = \text{variables assigned at } e\)
- \(\text{gen}_e = \text{variables used in an expression evaluated at } e\)
Recall: Questions

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

Execution Semantics

Abstraction

MOP-solution

sound?

how precise?

MFP-solution

sound?

precise?
Live Variables

MOP[y] = ∅ ∪ {y} = {y}

infinitely many such paths
Meet-Over-All-Paths Solution (MOP)

- 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 \textit{positively-distributive} if
\[ f(\sqcup X) = \sqcup \{ f(x) \mid x \in X \} \text{ for all } \emptyset \neq X \subseteq L, f \in F. \]

Theorem:

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

Remark:

A framework is positively-distributive if a) and b) hold:
(a) it is distributive: \[ f(x \sqcup y) = f(x) \sqcup f(y) \text{ f.a. } f \in F, x,y \in L \]
(b) it is effective: \[ L \text{ does not have infinite ascending chains}. \]

Remark:

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

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

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

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

pointwise join

$\top \quad \top(x) = \top \quad \text{f.a. } x \in \text{Var}$
\[ MOP[v] = (\top, \top, 5) \]
\((\rho(x), \rho(y), \rho(z))\)

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

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

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

Remark:
Any "reasonable" framework is monotone. ☺
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
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:
  \[(D, \sqsubseteq) = (P(Edges^*), \subseteq) \quad \text{or} \quad (D, \sqsubseteq) = (P(Stmt^*), \subseteq)\]

- sets of reaching states (collecting semantics):
  \[(D, \sqsubseteq) = (P(\Sigma^*), \sqsubseteq) \quad \text{with} \quad \Sigma = \text{Var} \to \text{Val}\]
Transfer Lemma

Situation:
complete lattices \((L, \sqsubseteq)\), \((L', \sqsubseteq')\)
monotonic functions \(f: L \rightarrow L\), \(g: L' \rightarrow L'\), \(\alpha: L \rightarrow L'\)

Definition:
Let \((L, \sqsubseteq)\) be a complete lattice.
\(\alpha : L \rightarrow L\) is called universally-disjunctive iff \(\forall X \subseteq L: \alpha(\sqcup X) = \sqcup \{ \alpha(x) | x \in X \}\).

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

Transfer Lemma:
Suppose \(\alpha\) is universally-disjunctive. Then:
(a) \(\alpha \circ f \sqsubseteq' g \circ \alpha \Rightarrow \alpha(\operatorname{lfp}(f)) \sqsubseteq' \operatorname{lfp}(g)\).
(b) \(\alpha \circ f = g \circ \alpha \Rightarrow \alpha(\operatorname{lfp}(f)) = \operatorname{lfp}(g)\).
Abstract Interpretation

Assume a universally-disjunctive abstraction function \( \alpha : D \rightarrow D^\# \).

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 \( \alpha (\text{MFP}[u]) \sqsubseteq^\# \text{MFP}^\#[u] \) 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 \( \alpha (\text{MFP}[u]) = \text{MFP}^\#[u] \) f.a. \( u \)

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

Constraint system for reaching runs:

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

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

Operational justification:

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

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

Prove:

a) \( R^{op}[u] \) satisfies all constraints \hspace{1cm} (direct)
   \[ \Rightarrow R[u] \subseteq R^{op}[u] \quad \text{f.a. } u \]

b) \( w \in R^{op}[u] \Rightarrow w \in R[u] \hspace{1cm} \text{(by induction on } |w|) \)
   \[ \Rightarrow R^{op}[u] \subseteq R[u] \quad \text{f.a. } u \]
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 $\alpha_{\text{MOP}} : \mathcal{P}(\text{Edges}^\ast) \rightarrow L$ by

$$\alpha_{\text{MOP}}(R) = \bigsqcup \{ f_r(\text{init}) \mid r \in R \} \quad \text{where } f_\epsilon = ld, \ f_{s\cdot(e)} = f_e \circ f_s$$

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

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

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

- Introduction
- Fundamentals of Program Analysis
- Interprocedural Analysis
- Analysis of Parallel Programs
- 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

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

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

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

Main:

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

P:

- \( c := a + b \)

Q:

- \( a := 7 \)
- \( c := a + b \)

R:

- \( a := 7 \)
- \( c := a + b \)

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

Procedures:

Recursion:
- \( \text{P}() \rightarrow \text{P}() \)
Running Example:
(Definite) Availability of the single expression $a+b$

The lattice:

```
false  a+b not available
    /  |
true   a+b available
```

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.

```
c:=a+b P()
false
a:=7 P()
c:=a+b P: Main:
false true true false true true false false true
☺☺ ☺☺
```

The lattice:
Context-Insensitive Analysis

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

The lattice:

```
Main:
false

true

c:=a+b

true true
false

true false

false

false

false
```

P:

```
false

true false

true false
```

P():

```
false

true false

true false
```

a:=7

false true

false

false

false

false

false
Let's Apply Our Abstract Interpretation Recipe: Constraint System for Feasible Paths

Operational justification:

\[
S(u) = \left\{ r \in \text{Edges}^* \mid \begin{array}{c} S_{t_{st}} \xrightarrow{r} u \end{array} \right\} \quad \text{for all } u \text{ in procedure } p
\]

\[
S(p) = \left\{ r \in \text{Edges}^* \mid \begin{array}{c} S_{t_{st}} \xrightarrow{r} \epsilon \end{array} \right\} \quad \text{for all procedures } p
\]

\[
R(u) = \left\{ r \in \text{Edges}^* \mid \exists \omega \in \text{Nodes}^* : \begin{array}{c} S_{t_{st}} \xrightarrow{r} u \omega \end{array} \right\} \quad \text{for all } u
\]

Same-level runs:

\[
S(p) \supseteq S(r_p)
\]

\[
S(st_p) \supseteq \{\epsilon\}
\]

\[
S(v) \supseteq S(u) \cdot \{\langle e \rangle\}
\]

\[
S(v) \supseteq S(u) \cdot S(p)
\]

Reaching runs:

\[
R(st_{Main}) \supseteq \{\epsilon\}
\]

\[
R(v) \supseteq R(u) \cdot \{\langle e \rangle\}
\]

\[
R(v) \supseteq R(u) \cdot S(p)
\]

\[
R(st_p) \supseteq R(u)
\]
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

Classic approaches for summary informations:

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

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

3) Call string approach: [Sharir/Pnueli 81], [Khedker/Karkare: CC´08]
   Analyse relative to finite portion of call stack !
Formalization of Functional Approach

Abstractions:

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

$$\alpha_{\text{Funct}}(R) = \bigsqcup \left\{ f_r \mid r \in R \right\} \text{ for } R \subseteq \text{Edges}^*$$

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

$$\alpha_{\text{MOP}}(R) = \bigsqcup \left\{ f_r(\text{init}) \mid r \in R \right\} \text{ for } R \subseteq \text{Edges}^*$$

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

- $S^#(p) \sqsubseteq S^#(r_p)$ $r_p$ return point of $p$
- $S^#(st_p) \sqsubseteq id$ $st_p$ 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 $p$
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:
\[ \alpha_{\text{MOP}}(R[u]) = R^\#[u] \quad \text{f.a. } u \]

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 (in general).
**Functional Approach for Availability of Single Expression Problem**

Observations:

Just three monotone functions on lattice $L$:

\[
\begin{align*}
\lambda x . \text{false} & \quad \text{k (ill)} \\
\lambda x . x & \quad \text{i (ignore)} \\
\lambda x . \text{true} & \quad \text{g (generate)}
\end{align*}
\]

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:

\[ c := a + b \]

P():

\[ c := a + b \]

Q():

\[ a := 7 \]

R():

\[ c := a + b \]

\[ a := 7 \]

the lattice:
Context-Sensitive Analysis, 2. Phase

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:

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

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.
Program Analysis of Sequential and Parallel Programs III

Markus Müller-Olm
Westfälische Wilhelms-Universität Münster, Germany

14th Estonian Winter School in Computer Science
Palmse, Estonia, March, 1-6, 2009
Outline of this Lecture

- Precise interprocedural analysis through linear algebra
  - algebraic invariants, fixpoints, recursion, linear algebra

- Regular symbolic analysis of dynamic networks of pushdown systems
  - automata-based, recursion, parallelism (thread creation)
Outline of this Lecture

• Precise interprocedural analysis through linear algebra
  • algebraic invariants, fixpoints, recursion, linear algebra

• Regular symbolic analysis of dynamic networks of pushdown systems
  • automata-based, recursion, parallelism (thread creation)
Finding Invariants...

Main:

0

1

x₁ := x₂

2

x₃ := 0

P()

3

x₁ - x₂ - x₃ = 0

x₁ := x₁ - x₂ - x₃

4

x₁ = 0

P:

5

P:

6

x₃ := x₃ + 1

7

x₁ := x₁ + x₂ + 1

P()

8

x₁ := x₁ - x₂

9

x₁ - x₂ - x₃ - x₆x₄ = 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
- ...
A Program Abstraction

Affine programs:

- affine assignments: \( x_1 := x_1 - 2x_3 + 7 \)
- unknown assignments: \( x_i := ? \) → 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?

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], [Khedker/Karkare: CC´08]
• Idea: take just a finite piece of run-time stack into account
• Problem: not exact

Relational approach  [Cousot/Cousot, 1977]
• Idea: summarize each procedure by approximation of I/O relation
• Problem: not exact (next slide)
Relational Analysis is Not Strong Enough

Main:

0

\[ x := 1 \]

1

P(())

2

\[ x = 1 \]

P:

3

\[ x := x \]

4

\[ x := 2 \cdot x - 1 \]

True relational semantics of P:

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

- Every execution path $\pi$ induces an **affine transformation** of the program state:

\[
\begin{bmatrix}
  x_1 := x_1 + x_2 + 1; \\
  x_3 := x_3 + 1
\end{bmatrix}(v)
\]

\[
= \begin{bmatrix}
  x_3 := x_3 + 1
\end{bmatrix}\left(\begin{bmatrix}
  x_1 := x_1 + x_2 + 1
\end{bmatrix}(v)\right)
\]

\[
= \begin{bmatrix}
  x_3 := x_3 + 1
\end{bmatrix}\begin{pmatrix}
  1 & 1 & 0 \\
  0 & 1 & 0 \\
  0 & 0 & 1
\end{pmatrix}\begin{pmatrix}
  v_1 \\
  v_2 \\
  v_3
\end{pmatrix} + \begin{pmatrix}
  1 \\
  0 \\
  1
\end{pmatrix}
\]

\[
= \begin{pmatrix}
  1 & 1 & 0 \\
  0 & 1 & 0 \\
  0 & 0 & 1
\end{pmatrix}\begin{pmatrix}
  v_1 \\
  v_2 \\
  v_3
\end{pmatrix} + \begin{pmatrix}
  1 \\
  0 \\
  1
\end{pmatrix}
\]
Affine Relations

- An affine relation can be viewed as a vector:

\[
\begin{bmatrix}
1 & -3 & 5 \\
1 & 0 & 1 \\
0 & 1 & 3 \\
\end{bmatrix}
\]

\[x_1 - 3x_2 + 5 = 0\] corresponds to \(a = \begin{bmatrix} 5 \\ 1 \\ 3 \\ 0 \end{bmatrix}\)
Affine Assignments induce linear wp-Transformations on Affine Relations

\{ x_2 + x_3 + 5 = 0 \} \quad x_1 := 4x_2 + x_3 + 3 \quad \{ x_1 - 3x_2 + 2 = 0 \}

A linear transformation:

\[
\begin{pmatrix}
1 & 3 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 4 & 1 & 0 \\
0 & 1 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
2 \\
1 \\
-3 \\
0
\end{pmatrix}
= \begin{pmatrix}
5 \\
0 \\
1 \\
1
\end{pmatrix}
\]
WP of Affine Relations

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

$$\left[ x_1 := x_1 + x_2 + 1; \ x_3 := x_3 + 1 \right]^T(a)$$

$$= \left[ x_1 := x_1 + x_2 + 1 \right]^T \left( \left[ x_3 := x_3 + 1 \right]^T(a) \right)$$

$$= \left[ x_1 := x_1 + x_2 + 1 \right]^T \begin{pmatrix} 1 & 0 & 0 & 1 \ 0 & 1 & 0 & 0 \ 0 & 0 & 1 & 0 \ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix}$$
Observations

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

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

- Matrices \( M \) form a vector space of dimension \((k+1) \times (k+1)\).

- Sub-spaces form a complete lattice of height \( O(k^2) \).
Let’s Apply Our Abstract Interpretation Recipe: Constraint System for Feasible Paths

Operational justification:

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

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

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

Same-level runs:

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

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

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

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

Reaching runs:

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

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

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

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

1) Compute a basis $B$ with:
   \[
   \text{Span } B = \text{Span } \{ [\pi]^T \mid \pi \text{ reaches } v \}
   \]
   for each program point by a precise abstract interpretation:

   Lattice: Subspaces of $\mathbb{IF}^{(k+1) \times (k+1)}$

   Replace:
   \[
   \begin{align*}
   \{ \varepsilon \} & \quad \text{by} \quad \{ I \} \quad \text{(I identity matrix)} \\
   \text{concatenation} & \quad \text{by} \quad \text{matrix product} \quad \text{(lifted to subspaces)} \\
   \{ \langle e \rangle \} & \quad \text{by} \quad \langle A_e \rangle \quad \text{for affine assignment edge } e = (u,s,v)
   \end{align*}
   \]

2) Solve the linear equation system:
   \[
   M \ a = 0 \quad \text{for all } M \in B
   \]
Theorem

In an affine program:

- The following vector spaces of matrices can be computed precisely:
  \[ \alpha(R(v)) = \text{Span} \{ [\pi]^T | \pi \in R(v) \} \] for each prg. point \( v \).

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

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

P:

0

\(x_3 := x_3 + 1\)

1

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

2

P()

3

\(x_1 := x_1 - x_2\)

4

\(\Rightarrow\) stable!
An Example

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

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

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

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

Just the affine relations of the form

\[ a_1 x_1 - a_4 x_2 - a_1 x_3 = 0 \quad (a_1 \in \mathbb{F}) \]

are valid at 3

\[
\text{Span}\left\{ \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \right\}
\]
Extensions

- In the paper (see handouts):
  - Local variables, value parameters, return values
  - Computing polynomial relations of degree $\leq d$
  - Affine pre-conditions
  - Formalization as an abstract interpretation

- In follow-up papers (see webpage):
  - Computing over modular rings (e.g. modulo $2^w$) or PIRs
  - Forward algorithm
Outline of this Lecture

- Precise interprocedural analysis through linear algebra
  - algebraic invariants, fixpoints, recursion, linear algebra

- Regular symbolic analysis of dynamic networks of pushdown systems
  - automata-based, recursion, parallelism (thread creation)
DPNs: Dynamic Pushdown-Networks

A *dynamic pushdown-network* (over a finite set of actions Act) consists of:

- $P$, a finite set of control symbols
- $\Gamma$, a finite set of stack symbols
- $\Delta$, a finite set of rules of the following form

$$p\gamma \xrightarrow{a} p_1w_1$$
$$p\gamma \xrightarrow{a} p_1w_1 \triangleright p_2w_2$$

(with $p, p_1, p_2 \in P$, $\gamma \in \Gamma$, $w_1, w_2 \in \Gamma^*$, $a \in \text{Act}$).
DPNs: Dynamic Pushdown-Networks

A State of a DPN is a word in \((\mathcal{P}\Gamma^*)^+\):

\[ p_1 w_1 p_2 w_2 \cdots p_k w_k \quad \text{(with } p_i \in \mathcal{P}, w_i \in \Gamma^*, k > 0) \]

... an infinite state space

The transition relation of a DPN:

\[ (p\gamma \xrightarrow{a} p_1 w_1) \in \Delta: \quad u p\gamma v \xrightarrow{a} u p_1 w_1 v \]

\[ (p\gamma \xrightarrow{a} p_1 w_1 \triangleright p_2 w_2) \in \Delta: \quad u p\gamma v \xrightarrow{a} u p_2 w_2 p_1 w_1 v \]
Example

Consider the following DPN with a single rule

\[ p\gamma \xrightarrow{a} p\gamma \gamma \gamma \gamma q\gamma \]

Transitions:

\[ p\gamma \]
\[ q\gamma q\gamma q\gamma \gamma \gamma \]
\[ q\gamma q\gamma q\gamma q\gamma q\gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \gamma \g
Reachability Analysis

Given:
- Model of a system: \( M \)
- Set of system states: \( \text{Bad} \)

Reachability analysis:
- Can a state from \( \text{Bad} \) be reached from an initial states of the system?
  \[ \exists \sigma_0, \ldots, \sigma_k : \text{Init} \ni \sigma_0 \rightarrow \cdots \rightarrow \sigma_k \in \text{Bad} \]

Applications:
- Check safety properties:
  \( \text{Bad} \) is a set of states to be avoided
- More applications by iterated computation of reachability sets for sub-models of the system model, e.g. data-flow analysis...
Reachability Analysis

Given:
- Model of a system: \( M \)
- Set of system states: \( \text{Bad} \)

Reachability analysis:
- Can a state from \( \text{Bad} \) be reached from an initial state of the system?
  \( \exists \sigma_0, \ldots, \sigma_k : \text{Init} \ni \sigma_0 \rightarrow^* \sigma_k \in \text{Bad} \)

Def.: - \( \text{pre}^*(X) = \{ \sigma \mid \exists \sigma' \in X : \sigma \rightarrow^* \sigma' \} \)
- \( \text{post}^*(X) = \{ \sigma \mid \exists \sigma' \in X : \sigma' \rightarrow^* \sigma \} \)

Equivalent formulations of reachability analysis:
- \( \text{pre}^*(\text{Bad}) \cap \text{Init} \neq \emptyset \)
- \( \text{post}^*(\text{Init}) \cap \text{Bad} \neq \emptyset \)

\( \Rightarrow \) Computation of \( \text{pre}^* \) or \( \text{post}^* \) is key to reachability analysis
Reachability Analysis of Finite State Systems

\[ \varphi_0 = \text{Init} \]

\[ \varphi_{i+1} = \varphi_i \cup \text{post}(\varphi_i) \]

\[ \text{post}(X) = \{ \sigma | \exists \sigma' \in X : \sigma \rightarrow \sigma' \} \]

\[ \Rightarrow \text{Bad reachable from initial state} \]
Reachability Analysis of Finite State Systems

\[ \varphi_0 = \text{Init} \]
\[ \varphi_{i+1} = \varphi_i \cup \text{post}(\varphi_i) \]
\[ \text{post}(X) = \{ \sigma | \exists \sigma' \in X : \sigma \rightarrow \sigma' \} \]

⇒ Bad not reachable from initial state
Problems with Infinite-State Systems

- State sets $\varphi_i$ can be infinite

  $\Rightarrow$ symbolic representation of (certain) infinite state sets

  Here: by finite automata
Example: Representation of an Infinite State Set of a DPN by a Word Automaton

An automaton $A$:

\[
\begin{align*}
\text{The regular set of states represented by } A : \\
L(A) &= (q\gamma q\gamma q^*)^* \\
\text{... an infinite set of states.}
\end{align*}
\]
Problems with Infinite-State Systems

- State sets $\varphi_i$ can be infinite
  - $\Rightarrow$ symbolic representation of (certain) infinite state sets
    
    Here: by finite (word) automata

- Iterated computation of reachability sets does not terminate in general
  - $\Rightarrow$ Methods for \textit{acceleration} of the computation
    
    Here: by computing with finite automata
Computing pre* for DPNs with Finite Automata

Theorem [Bouajjani, MO, Touili, 2005]

For every DPN and every regular state set R, pre*(R) is regular and can be computed in polynomial time.

Proof:

Generalization of a known technique for single pushdown systems: saturation of an automaton for R.

⇒ Reachability analysis is effective for regular sets Bad of states!
Example: Reachability Analysis for DPNs

Consider again DPN with the rule

\[ p\gamma \xrightarrow{a} p\gamma \triangleright q\gamma \]

and the infinite set of states

\[ \text{Bad} = \left( q\gamma q\gamma p\gamma^* \right)^* = L(A) \]

Analysis problem: can Bad be reached from p\gamma?
Example: Reachability Analysis for DPNs

1. **Step**: Saturate automaton for Bad with the DPN rule:

   \[ p\gamma \xrightarrow{a} p\gamma > q\gamma \]

   Resulting automaton \( A_{\text{pre}^*} \) represents \( \text{pre}^*(\text{Bad}) \)!

2. **Step**: Check, whether \( p\gamma \) is accepted by \( A_{\text{pre}^*} \) or not

   **Result**: Bad is reachable from \( p\gamma \), as \( A_{\text{pre}^*} \) accepts \( p\gamma \).
Modelling Programs with Procedures and Threads by DPNs

Main:
- \( n_1 \) to \( n_2 \): \( x := x + 1 \)
- \( n_2 \) to \( n_3 \): call Main
- \( n_3 \) to \( n_4 \): \( y := 0 \)
- \( n_4 \) to \( n_1 \): spawn Q

Q:
- \( m_1 \) to \( m_2 \): \( y := x \cdot y \)
- \( m_2 \) to \( m_3 \): call Q
- \( m_3 \) to \( m_4 \): \( x := y + 1 \)
- \( m_4 \) to \( m_1 \): spawn Q

\[ \begin{align*}
\#N_1 \xrightarrow{x:=x+1} & \#N_2 \\
\#N_2 \xrightarrow{\text{call}_p} & \#N_1 N_3 \\
\#N_3 \xrightarrow{y:=0} & \#N_4 \\
\#N_1 \xrightarrow{\text{spawn}_Q} & \#N_4 \triangleright \#M_1 \\
\#M_1 \xrightarrow{y:=x \cdot y} & \#M_2 \\
\#M_2 \xrightarrow{\text{call}_Q} & \#M_1 M_3 \\
\#M_3 \xrightarrow{x:=y+1} & \#M_4 \\
\#N_1 \xrightarrow{\text{skip}} & \#M_4
\end{align*} \]
Live Variables Analysis via Iterated pre[•]-computation

Observation

Variable $x$ is live at $u$

iff

$$e_{Main} \in \text{pre}^{*}(A t_u \cap \text{pre}^{*}_{\Delta_{\text{non-def}}} (\text{pre}_{\Delta_{\text{use}}} (Conf)))$$

Remark

This condition can be checked by computing with automata.
A Non-Representability Result

- P induces trace language: \( L = \bigcup \{ A^n \cdot (B^m \otimes (C^i \cdot D^j)) \mid n \geq m \geq 0, i \geq j \geq 0 \} \)
- \( L \) cannot be characterized by constraint system with operators “concatenation“ and “interleaving“
Forward Reachability Analysis of DPNs

Observation [Bouajjani, MO, Touili, 2005]
In general, \( \text{post}^*(R) \) is not regular, not even if \( R \) is finite.

Example:
Consider DPN with the rule
\[ p \gamma \xrightarrow{a} p \gamma q \gamma \]
Recall:
\[ p \gamma \]
\[ q \gamma \]
\[ q \gamma \]
\[ q \gamma \]
\[ q \gamma \]
\[ q \gamma \]
\[ \vdots \]
\[ \vdots \]
\[ \vdots \]
\[ \text{post}^*\{p \gamma \} = \{ (q \gamma)^k p \gamma^{k+1} | k \geq 0 \} \text{ is not regular.} \]

Theorem [Bouajjani, MO, Touili, 2005]
For every DPN, \( \text{post}^*(R) \) is contextfree if \( R \) is contextfree.
It can be computed in polynomial time.
A Little Bit of Synchronization ...

- CDPNs – Constrained Dynamic Pushdown Networks

- Idea: Threads can observe (stable regular patterns of) their children, but not vice versa

- States are represented by trees in order to mirror father/child relationship

- Use tree automata techniques for
  - representation of state sets and
  - symbolic computation of pre* (under certain conditions)

- See the paper (in the handouts)
Conclusion

- Program analysis very broad topic
- Provides generic analysis techniques
- Tradeoff Precision vs. Efficiency
- Here just one path through the forest
- Many interesting topics not covered
Thank you !