Computing Summaries

for Interprocedural Analysis

Ashish Tiwari

Tiwari@csl.sri.com

Computer Science Laboratory

SRI International

Menlo Park CA 94025

`http://www.csl.sri.com/~tiwari`

Joint work with Sumit Gulwani, Microsoft Research

Outline of this Talk

- The Assertion Checking Problem
- Example
- Interprocedural Analysis
- A methodology for interprocedural backward analysis
- Special Cases: Abstract domains defined by
  - Linear Arithmetic
  - Uninterpreted Symbols
- Conclusion

Assertion Checking Problem

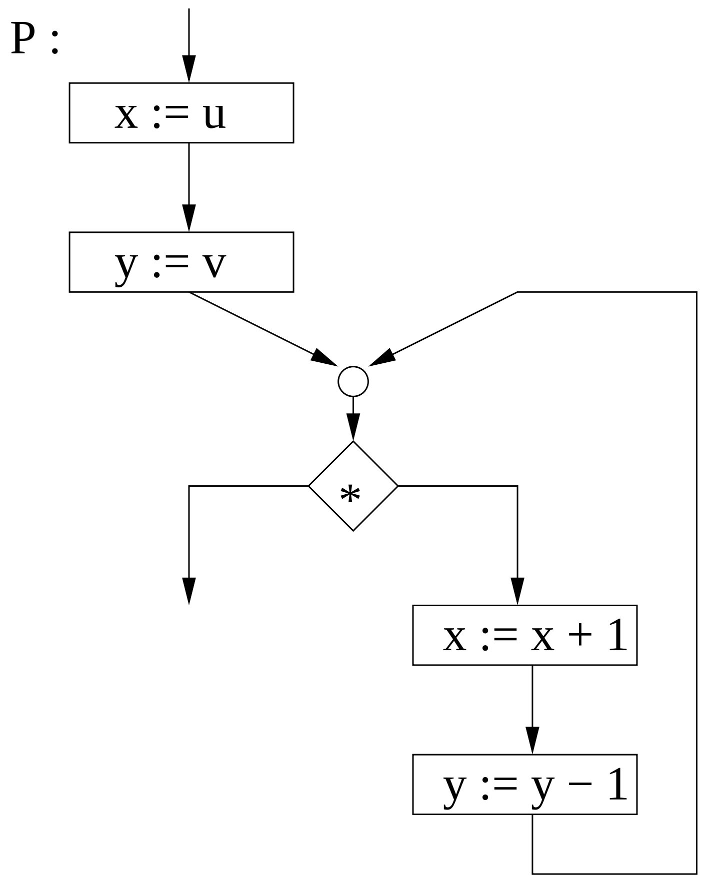
Given a program P annotated with an assertion ϕ
verify that ϕ evaluates to true in every *run* of P

$$P \in \mathbf{P}, \quad \mathbf{P} := \text{set of all programs in some programming model}$$
$$\phi \in \Phi, \quad \Phi := \text{set of all assertions in some assertion language}$$

This problem is undecidable for even simple $\mathbf{P}$ and Φ

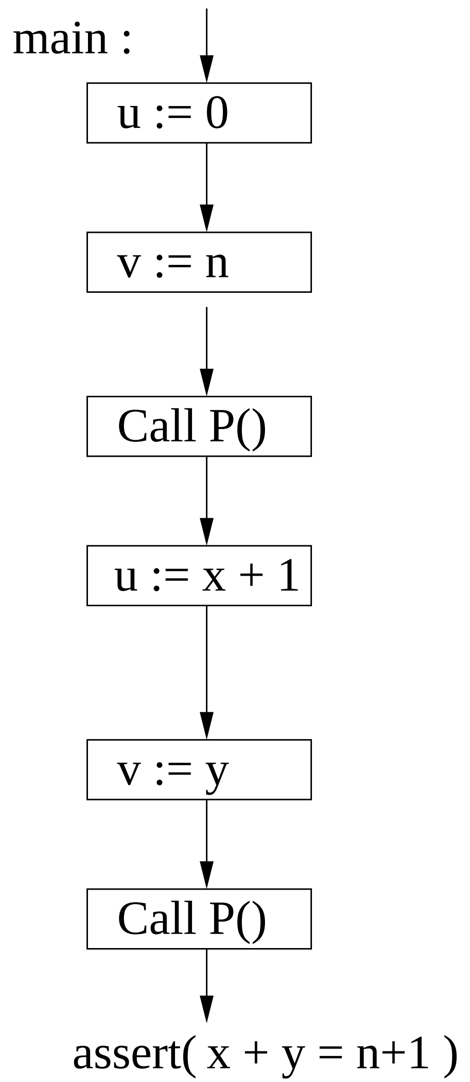
An Example

```
P() { // inputs: u,v
  x := u ;
  y := v ;
  while (*) {
    x := x + 1 ;
    y := y - 1 ;
  }
  // return x,y
}
```



An Example

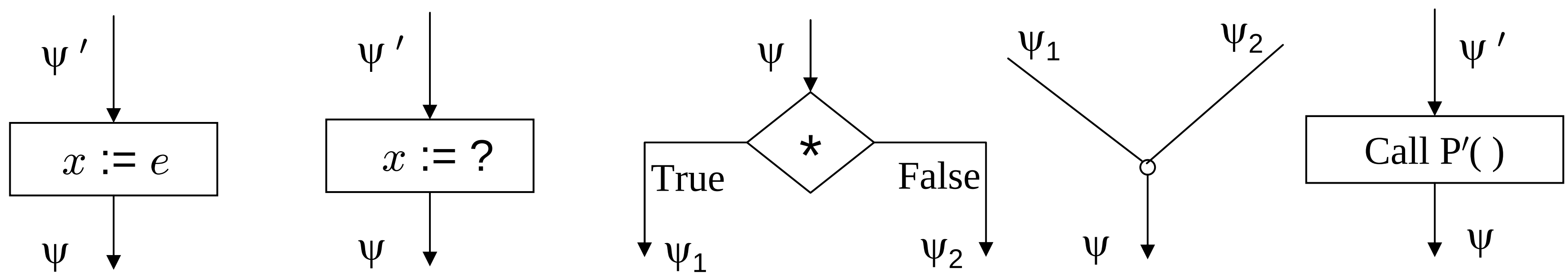
```
main() {
 u := 0 ;
 v := n ;
 Call P() ;
 u := x + 1 ;
 v := y ;
 Call P() ;
 assert(x + y == n+1)
}
```



Program Model

Programming Model in the example:

- Assignments: $x := e$, $x := ?$
- Nondeterminisitic conditionals: if (*)
- Join: Control flow merge
- Procedure call node: Call P()



(a) Assignment Node (b) Non-deterministic Assignment Node (c) Non-deterministic Conditional Node (d) Join Node (e) Procedure Call Node

Known Results on Assertion Checking

Nodes	Expr. Lang.	Complexity	Ref.
(a)-(d)	Lin Arith	PTime	[Karr 77,...]
(a)-(d)	UFS	PTime	[(Gulwani,Necula 04), (Müller-Olm, Rüthing, Seidl)]
(a)-(d)	UFS + LA	co-NP-hard	[Gulwani,T. 06]
(a)-(d)*	UFS + LA	decidable	[Gulwani,T. 06]

For generalizations of above results to other abstract domains and program models, see [Gulwani, T. VMCAI 07]

What about program models with procedure calls?

New Results

Present a general framework for interprocedural analysis

Nodes	Expr. Lang.	Complexity	Ref.
(a)-(e)	Lin Arith	PTime	[Müller-Olm and Seidl ’04, this paper]
(a)-(e)	Unary UFS	PTime	[this paper]
(a)-(e)	UFS	Open	

Some results on interprocedural analysis on UFS abstraction, but under restrictions, given by Müller-Olm, Seidl, and Steffen (ESOP’05)

Interprocedural Analysis

Two approaches for interprocedural analysis:

1. Inlining
2. Computing Summaries

Interprocedural Analysis: Inlining

```
P() {
  [ u + v == n+1 ]
  x := u;
  y := v;
  [ x + y == n+1 ]
  while (*) {
    x++;
    y--;
  }
  [ x + y == n+1 ]
}
```

```
main() {
 u := 0;
 v := n;
 Call P();
   [ x + 1 + y == n+1 ]
 u := x + 1;
 v := y;
   [ u + v == n+1 ]
 Call P();
   [ x + y == n+1 ]
 assert(x + y == n+1)
}
```

Interprocedural Analysis: Inlining

```
P() {
  [ u + v == n ]
  x := u;
  y := v;
  [ x + y == n ]
  while (*) {
    x++;
    y--;
  }
  [ x + y == n ]
}
```

```
main() {
   [ n + 0 == n ]
 u := 0;
 v := n;
   [ u + v == n ]
 Call P();
   [ x + 1 + y == n+1 ]
 u := x + 1;
 v := y;
   [ u + v == n+1 ]
 Call P();
   [ x + y == n+1 ]
 assert(x + y == n+1)
}
```

Interprocedural Analysis

Inlining: Re-analyzes P()

Summary Computation: Compute a summary of a procedure just once and use it to backward propagate across Call P() nodes

In the example, we required:

$$[\ ?\ ] \quad \text{Call P()} \quad [\ x + y = n + 1\ ]$$

$$[\ ?\ ] \quad \text{Call P()} \quad [\ x + y = n\ ]$$

Main idea: Propagate back a set of generic assertions

For example: $\alpha x + \beta y = \gamma$

Generic Assertions

Assertion that involves context-variables apart from regular program variables.

Examples of context-variables and their possible instantiations:

$$
\begin{array}{rcl}
\alpha(\_\_) & \mapsto & f(f(\_\_)),\ 2(\_\_),\ \_\_ + 1 \\
\beta(\_\_{}_1, \_\_{}_2) & \mapsto & 2(\_\_{}_1) + \_\_{}_2,\ f(\_\_{}_1, f(\_\_{}_2))
\end{array}
$$

A generic term: $\alpha(x) + \beta(y)$

A generic assertion: $\alpha(x) + \beta(y) = \gamma$

Complete Set of Generic Assertions

$\mathcal{A}$ is a complete set of generic assertions if,

for any generic assertion A_1, there exists $A_2 \in \mathcal{A}$ s.t.

$$A_1 = A_2\sigma$$

Expr. Lang.	Complete Set
Lin. Arith.	$\{\sum_{i\in V} \alpha_i x_i = \alpha\}$
Unary UFS	$\{\alpha(x_1) = \beta(x_2) \mid x_1, x_2 \in V,\ x_1 \not\equiv x_2\}$

We need a finite complete set of generic assertions

Computing Procedure Summaries

Summary := $\{(\psi_i, A_i) \mid [\psi_i]\,$Call P()$\,[A_i]\,, \quad A_i \in \mathcal{A}\}$

Method to compute procedure summaries:

1. WP based backward propagation over generic assertions
2. For procedure call nodes: requires matching current ψ with an assertion in $\mathcal{A}$ and using its current summary

$$\left[\bigwedge_i \psi_i' \sigma_i\right] \text{ Call P() } \left[\bigwedge_i B_i\right]$$

if (ψ_i', A_i) is in current summary of P() and $B_i = A_i \sigma_i$.

Computing Summaries: Linear Arithmetic

P() {
    $[true]$
    $x := u;$
    $y := v;$
    $[\alpha(x+1) + \beta(y-1) == \gamma,$
    $\alpha x + \beta y == \gamma]$
    while (*) {
        $x++;$
        $y--;$
    }
    $[\alpha x + \beta y == \gamma]$
}

P() {
    $[\alpha - \beta == 0, \alpha u + \beta v == \gamma]$
    $x := u;$
    $y := v;$
    $[\alpha - \beta == 0,$
    $\alpha x + \beta y == \gamma]$
    while (*) {
        $x++;$
        $y--;$
    }
    $[\alpha x + \beta y == \gamma]$
}

Summary: $\{(\alpha == \beta \wedge \alpha u + \beta v == \gamma, \;\; \alpha x + \beta y == \gamma)\}$

Computing Summaries: Linear Arithmetic

- Termination: There can be at most $k^2 + k + 1$ independent facts over the variables $\{\alpha_i x_j,\ \ \alpha_i,\ \ \gamma\}$ where $i, j \in \{1, \ldots, k\}$
- Since every fact is a linear equation over these $k^2 + k + 1$ variables
- Complexity of interprocedural assertion checking: $O(nk^{10})$ where n = number of program points and k = live variables
- Assuming arithmetic operations take $O(1)$ time

Using Summaries: Linear Arithmetic

main() {

$[0 + n == n]$

$u := 0;$

$v := n;$

$[1 - 1 == 0,\ u + v == n]$

Call P(); $\quad$ // $\alpha \mapsto 1, \beta \mapsto 1, \gamma \mapsto n$

$[x + 1 + y == n + 1]$

$u := x + 1;$

$v := y;$

$[1 - 1 == 0, u + v == n + 1]$

Call P(); $\quad$ // $\alpha \mapsto 1, \beta \mapsto 1, \gamma \mapsto n + 1$

$[x + y == n + 1]$

assert($x + y == n + 1$)

}

Computing Summaries: Unary UFS

The same general idea works.

- Complete Set of Generic Assertions: $\{\alpha(x) == \beta(y) \mid x, y \in V\}$, α and β are strings over the unary symbols
- Backward propagation gives generic assertions: $\{\alpha(C(x)) == \beta(D(y))\}$
- Termination: Any finite set of such assertions is essentially equivalent to a set containing at most two equations
- Summary: $\{(\psi_{xy}, \alpha(x) == \beta(y)) \mid x, y \in V, \; [\psi_{xy}]$ Call P() $[\alpha(x) == \beta(y)]\}$ where ψ_{xy} contains at most $k(k-1)/2 + 1$ equations
- All this takes polynomial number of string operations

However, programs can succinctly represent really large strings

Computing Summaries: Unary UFS: Large Strings

Consider the n procedures $P_0, \ldots, P_{n-1}$:

$$P_i(x_i)\ \{\ t := P_{i-1}(x_i);\ y_i := P_{i-1}(t);\ \mathit{return}(y_i); \}$$
$$P_0(x_0)\ \{\ y_0 := f x_0;\ \mathit{return}(y_0); \}$$

The summary of procedure P_i is:

$$(\alpha == f^{2^i}\ \wedge\ \beta = \epsilon,\ \ \alpha x_i == \beta y_i)$$

Computing Summaries: Unary UFS: Representation

- SCFGs: *singleton context-free grammars*
  A CFG where each nonterminal represents *exactly* one (terminal) string.
- An SCFG can represent strings in an exponentially succinct way
- We use SCFGs to represent strings during our interprocedural analysis
- Plandowski (1994) showed that equality (largest common prefix) checking of two strings represented as SCFGs can be done in PTime
- Summaries can be computed in time $O(nk^6 T_{base}(n))$ on the abstraction of unary symbols.

Computing Summaries: General Case

Interprocedural analysis on a logical lattice defined by Th:

- Finite complete set of generic assertions
- Finite essential ascending chain property: Every increasing sequence of generic assertions (over k regular variables) finitely essentially converges

What is essential equivalence?

In case of non-deterministic programs, do not need to distinguish between ϕ and $Unif(\phi)$

ψ is essentially equivalent to ψ' if $\psi\sigma$ and $\psi'\sigma$ have the same set of unifiers for every σ that assigns context variables to a ground term with holes

Conclusion

Presented a general framework for interprocedural analysis

Nodes	Expr. Lang.	Complexity	Ref.
(a)-(e)	Lin Arith	PTime	[Müller-Olm and Seidl ’04, this paper]
(a)-(e)	Unary UFS	PTime	[this paper]
(a)-(e)	UFS	Open	

Main ideas:

- Summary computation requires dealing with context variables
- Context unification can be used to simplify assertions to essentially equivalent assertions for non-det programs