A brief introduction to
static program analysis

David Schmidt

Kansas State University

www.cis.ksu.edu/~schmidt

What is static program analysis?

It is the extraction of a program's properties in advance of the program's execution.

Example properties:

- the program will not generate a run-time exception (error)
- the program will generate an output that has a desirable property
- the program's internal statements have desirable properties that admit optimization

Standard techniques:

- type checking
- iterative dataflow analysis
- theorem provimg

An example Python program

Let n be some input integer:

```
p0 : i = n;  x = 0;
p1 : while i != 0 :
       p2 : x = x + 1;  i = i - 1
p3 : print x
```

What properties can we extract?

- the program will not generate a type-mismatch exception
- the definitions (assignments) at point p_0 possibly reach p_3
- the program satisfies the postcondition, $x = n$

Type checking the Python program

p_0: `i = n;  x = 0;`
p_1: `while i != 0 :`
    p_2: `x = x + 1;  i = i - 1`
p_3: `print x`

The program is well-typed: it won't generate a mismatch exception.

$$\Gamma \vdash 0 : \mathrm{int} \qquad \frac{\Gamma \vdash e_1 : \mathrm{int} \quad \Gamma \vdash e_2 : \mathrm{int}}{\Gamma \vdash e_1 + e_2 : \mathrm{int}} \qquad \frac{\Gamma \vdash e_1 : \mathrm{int} \quad \Gamma \vdash e_2 : \mathrm{int}}{\Gamma \vdash e_1 == e_2 : \mathrm{bool}}$$

$$\frac{\Gamma \vdash e : \tau}{\Gamma \vdash x = e : \Gamma + [x \mapsto \tau]} \qquad \frac{\Gamma \vdash c_1 : \Gamma_1 \quad \Gamma_1 \vdash c_2 : \Gamma_2}{\Gamma \vdash c_1 ; c_2 : \Gamma_2} \qquad \frac{\Gamma \vdash e : \mathrm{bool} \quad \Gamma \vdash c : \Gamma}{\Gamma \vdash \mathit{while}\ e \colon c : \Gamma}$$

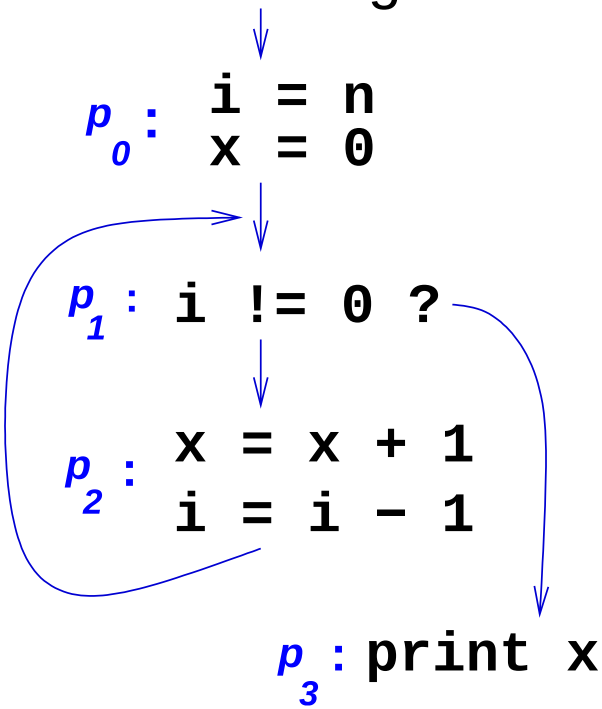
Although it's drawn as an (inverted) deduction, type checking is implemented as a traversal of the program's parse tree:

$$\frac{[\,] \vdash \texttt{i = n} \ [i \mapsto \mathrm{int}] \qquad [i \mapsto \mathrm{int}] \vdash \texttt{x = 0} : \Gamma_0 \qquad \dfrac{\Gamma_0 \vdash \texttt{i != n} : \Gamma_0 \qquad \dfrac{\Gamma_0 \vdash \texttt{x = x + 1} : \Gamma_0 \qquad \Gamma_0 \vdash \texttt{i = i - 1} : \Gamma_0}{\Gamma_0 \vdash \texttt{x = x + 1; i = i - 1} : \Gamma_0}}{\Gamma_0 \vdash \texttt{while i != 0: x = x + 1; i = i - 1} : \Gamma_0}}{[\,] \vdash \texttt{i = n; x = 0; while i != 0: x = x + 1; i = i - 1} : \Gamma_0}$$

Let $\Gamma_0 = [\, x \mapsto \mathrm{int},\ i \mapsto \mathrm{int}]$

Reaching definitions calculated by dataflow analysis

Does the assignment at p_i reach point p_j?



$$\mathrm{in}_{p_i} = \bigcup_{p' \in \mathrm{pred}\, p_i} \mathrm{out}_{p'}$$

$$\mathrm{out}_{p_i} = \mathrm{in}_{p_i} - \{p_x | p_x \equiv x = e\} \cup \{p_i\},$$

for $p_i \equiv x' = e'$

$$\mathrm{out}_{p_i} = \mathrm{in}_{p_i}, \quad \text{for } p_i \equiv e?$$

For the example program, the equations for reaching definitions are solved iteratively as

$$\mathrm{in}_{p_0} = \{\} \qquad \mathrm{in}_{p_1} = \{p_0, p_2\}$$

$$\mathrm{in}_{p_2} = \{p_0, p_2\} \quad \mathrm{in}_{p_3} = \{p_0, p_2\}$$

Partial correctness proved within Hoare logic

```
p0 : i = n;  x = 0;
p1 : while i != 0 :
        p2 : x = x + 1;  i = i - 1
p3 : print x
```

$$\{[e/x]P\}\ x = e\ \{P\}$$

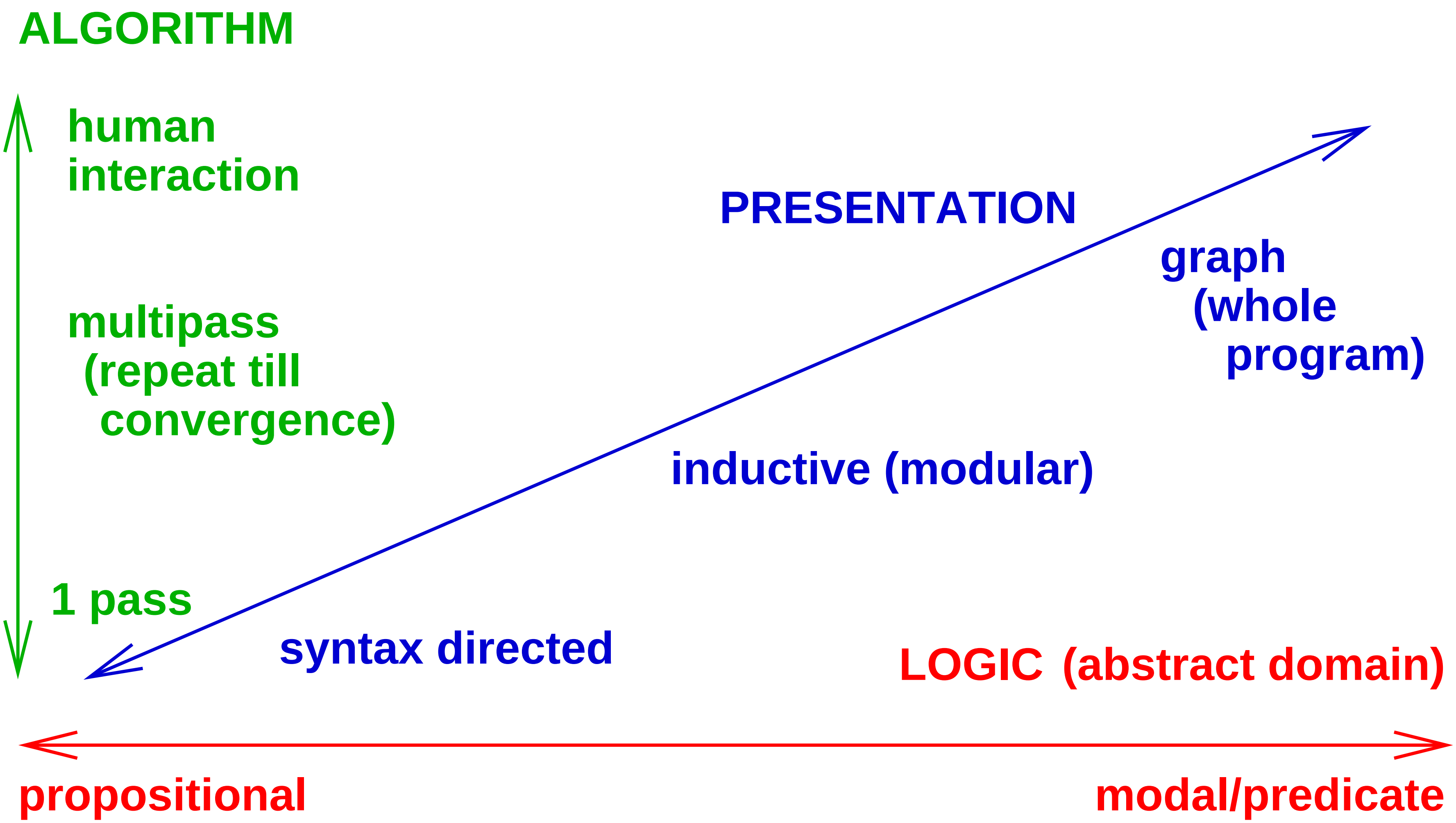
$$\frac{\{P\}\ c1\ \{Q\}\ c2\ \{R\}}{\{P\}\ c1; c2\ \{R\}}$$

$$\frac{\{e \wedge P\}\ c\ \{P\}}{\{P\}\ \textit{while}\ e :\ c\ \{\neg e \wedge P\}}$$

$$\frac{\dfrac{\dfrac{\{x+1=n-i+1\}\ \texttt{x = x + 1}\ \{x=n-i+1\}\ \texttt{i = i - 1}\ \{x=n-i\}}{\{i \mathrel{!}= 0\ \&\ x=n-i\}\ \texttt{x = x + 1; i = i - 1}\ \{x=n-i\}}}{\{x=n-i\}\ \texttt{while i != 0: x = x + 1; i = i - 1}\ \{x=n\}}}{\{true\}\ \texttt{i = n; x = 0}\ \{x=n-i\}\ \texttt{while i != 0: x = x + 1; i = i - 1}\{x=n\}}$$

One must discover the loop invariant, $x = n - i$, to accomplish the proof.

Three axes of static analyses



Some standard static analyses

	logic	algorithm	presentation
type checking	propositional logic: $\mathtt{int}$, $\mathtt{int} \to \mathtt{bool}$	one pass (traverse syntax tree)	syntax-directed
ML type inference	shallow $\forall$ logic: $\forall \alpha . \alpha \to \alpha$	one pass + unification	syntax-directed
a.i.-based dataflow analysis	propositional logic (token sets)	iterate until convergence	graph-based
model checking	LTL, ACTL (modal-like logics)	iterate forever (!)	graph-based
theorem proving, LF	predicate logics	human interaction	inductive, stated axiomatically

A "hybrid" analysis: predicate abstraction

We wish to prove that $\mathtt{z} \geq \mathtt{x} \wedge \mathtt{z} \geq \mathtt{y}$ at p_3:

***p*₀** **:** `if x < y`
***p*₁** **:** `  then z = y`
***p*₂** **:** `  else z = x`
p*₃** **:** ***exit

$$\begin{tikzcd}[column sep=small]
 & p_0, \langle ?, ?, ? \rangle \arrow[dl] \arrow[dr] & \\
p_1, \langle t, ?, ? \rangle \arrow[d] & & p_2, \langle f, ?, ? \rangle \arrow[d] \\
p_3, \langle t, t, t \rangle & & p_3, \langle f, t, t \rangle
\end{tikzcd}$$

We choose three predicates,

$$\phi_1 = \mathtt{x} < \mathtt{y}$$

$$\phi_2 = \mathtt{z} \geq \mathtt{x}$$

$$\phi_2 = \mathtt{z} \geq \mathtt{y}$$

and compute their values at the program's points. The predicates' values come from the domain, $\{t, f, ?\}$. (Read $?$ as $t \vee f$.)

At all occurrences of p_3 in the abstract trace, $\phi_2 \wedge \phi_3$ holds.

When a goal is undecided, refinement is necessary

Prove $\phi_0 \equiv \mathtt{x} \geq \mathtt{y}$ at p_4:

```
p0 : if !(x >= y)
p1 : then { i = x;
       p2 :  x = y;
       p3 :  y = i;
p4 : }
```

$$\begin{array}{ccc} & p_0, \langle ? \rangle & \\ \swarrow & & \searrow \\ p_1, \langle f \rangle & & p_4, \langle t \rangle \\ p_2, \langle f \rangle & & \\ p_3, \langle t \rangle & & \\ p_4, \langle ? \rangle & & \end{array}$$

To decide the goal, we must refine the state by adding a needed auxiliary predicate: $wp(\mathtt{y} = \mathtt{i}, \mathtt{x} \geq \mathtt{y}) = (\mathtt{x} \geq \mathtt{i}) \equiv \phi_1$.

$$\begin{array}{ccc} & p_0, \langle ?, ? \rangle & \\ \swarrow & & \searrow \\ p_1, \langle f, ? \rangle & & p_4, \langle t \rangle \\ p_2, \langle f, t \rangle & & \\ p_3, \langle t, t \rangle & & \\ p_4, \langle t, t \rangle & & \end{array}$$

because $\mathtt{x} \not\geq \mathtt{y}$ and $\mathtt{x} \geq \mathtt{i}$ imply $\mathtt{y} > \mathtt{i}$ implies $\mathtt{x}_{new} \geq \mathtt{i}$

But incremental predicate refinement cannot synthesize many interesting loop invariants. For this example:

p_0 : `i = n; x = 0;`
p_1 : `while  i != 0  {`
  p_2 : `x = x + 1;  i = i - 1;`
  `}`
p_3 : `goal: x = n`

We find that the initial predicate set, $P_0 \equiv \{i = 0, x = n\}$, does not validate the loop body.

The first refinement suggests we add $P_1 \equiv \{i = 1, x = n - 1\}$ to the program state, but this fails to validate a loop that iterates more than once.

Refinement stage j adds predicates $P_j \equiv \{i = j, x = n - j\}$; the refinement process continues forever!

The loop invariant is $x = n - i$:-)

Mr. Gedell will present interesting combinations and variations of the standard analyses....

References This talk: www.cis.ksu.edu/~schmidt/presentations

1. A. Aho and J. Ullman. Principles of Compiler Design. Addison Wesley, 1977.
2. K. Apt and G. Plotkin. Ten years of Hoare's logic: a survey, part 1. ACM TOPLAS 3 (1981).
3. E. Clarke, O. Grumberg, and D. Peled. Model Checking. MIT Press 1999.
4. P. Cousot and R.Cousot. Abstract interpretation. ACM POPL 1977.
5. F. Nielson, H.R. Nielson, and C. Hankin. Principles of Program Analysis. Springer, 1999.
6. D. Schmidt. Introduction to static analysis and abstract interpretation. School on Semantics and Applications, 2003. http://santos.cis.ksu.edu/schmidt/Escuela03/home.html