What is static analysis?

Matt Might
matt.might.net
University of Utah
@mattmight
interpreter
abstract interpreter
static analyzer
What
Why
Too slow.
Too buggy.
Too insecure.
Why?
We can’t engineer.
Software engineering?
Software engineering?

How do they know the load limit on bridges, Dad?

They drive bigger and bigger trucks over the bridge until it breaks.

Then they weigh the last truck and rebuild the bridge.

Oh, I should've guessed.

Dear, if you don't know the answer, just tell him!
We can’t engineer.
We can’t predict.
Because Alan Turing said so!

Halt!
“Thou shalt not write an algorithm which determines whether a program halts.”
while P(x)

“the loop hole”

while P(x)
Interesting question?
Interesting question?

Undecidable.
Why we need software engineering
class MyActivity {
  public MyActivity() {
   activateMic();
  }
}

class MyActivity {
    public MyActivity() {
        activateMic();
    }
}

A problem has been detected and Windows has been shut down to prevent damage to your computer.
The problem seems to be caused by the following file: SPOMCON.SYS

If this is the first time you've seen this stop error screen, restart your computer. If this screen appears again, follow these steps:

- Check to make sure any new hardware or software is properly installed.
- If this is a new installation, ask your hardware or software manufacturer for any Windows updates you might need.
- If problems continue, disable or remove any newly installed hardware or software. Disable BIOS memory options such as caching or shadowing.
- If needed, use Safe Mode to remove or disable components, restart your computer, press F8 to select Advanced Startup Options, and then select Safe Mode.

Technical Information:
*** STOP: 0x00000050 (0xFDF004C2, 0x00000001, 0xBFBE7617, 0x00000000)
*** SPOMCON.SYS - Address FB0E7617 base at F8FE5000, DateStamp 3DDD07c
There’s a loop hole...
There’s a loop hole...

...in the loop hole.
static analysis = reasoning
MAX++

*a++ =

*0
MAX++

*a++ =

*0
Example: Sign analysis
What is the sign of $-3 \times 2$?
... -3 -2 -1 0 1 2 3 ...
... -3 -2 -1 0 1 2 3 ...

\[ -3 \times 2 \]
\[-3 \times 2 = -6\]
Let’s build it!
\[
\text{<exp> ::= <int>}
\]

\[
\text{<exp> ::= <exp> * <exp>}
\]

\[
\text{<exp> ::= <exp> = <exp>}
\]

\[
\text{<exp> ::= <exp> + <exp>}
\]
simple-eval : exp -> integer
simple-eval^ : exp -> abstract-integer
simple-eval : exp -> integer
\( \alpha : \text{integer} \rightarrow \text{abstract-integer} \)
simple-eval^ : exp -> abstract-integer
Example: Turing machines
How to approximate?
Make it finite!
The diagram shows a transition graph with states $q_1$, $q_2$, and $q_3$. The transitions are labeled with moves $(0,1,R)$ and $(1,0,L)$, indicating right and left movements, respectively. The final state $q_3$ is highlighted in green. There is a sequence of symbols and a set $\{0,1\}$ at the bottom of the diagram, indicating the possible inputs or states.
$\{0, 1\}$
Let’s do it for RTL.
What is static program analysis?

The halting problem asks whether the execution of a specific program for a given input will terminate.

The halting problem is famous for being undecidable.

That is, no algorithm can solve it for all programs and all inputs.

This complicates any attempt to predict program behavior: we can make predicting almost any program behavior equivalent to predicting the termination of a nearly identical program.

Static analyses are algorithms that do their best to defy the undecidability of the halting problem: they attempt to predict program behavior.

Predicting program behavior enables program optimization, security audits, automatic parallelization and, if accurate enough, correctness verification.
<prog> ::= <stmt> ... 

<stmt> ::= <label> : |
| goto <label> ; |
| <var> := <exp> ; |
| if <exp> goto <label> ; 

<exp> ::= <exp> + <exp> |
| <exp> * <exp> |
| <exp> = <exp> |
| <int> |
| <var>
\[ <\text{prog}> ::= <\text{stmt}> \ldots \]

\[ <\text{stmt}> ::= \text{\texttt{(label \ <label>\)}} \]
\[ | \quad \text{\texttt{(goto \ <label>\)}} \]
\[ | \quad \text{\texttt{(:= \ <var> \ <exp>\)}} \]
\[ | \quad \text{\texttt{(if \ <exp> \ goto \ <label>\)}} \]

\[ <\text{exp}> ::= \text{\texttt{(+ \ <exp> \ <exp>\)}} \]
\[ | \quad \text{\texttt{(* \ <exp> \ <exp>\)}} \]
\[ | \quad \text{\texttt{ (= \ <exp> \ <exp>\)}} \]
\[ | \quad <\text{int}> \]
\[ | \quad <\text{var}> \]
(struct state {stmts env})
stmts = stmt*

env = var #=> integer
injected : prog $\rightarrow$ state

step : state $\rightarrow$ state
; stmt-map : label => stmt*
(define stmt-map (make-hasheq))
preprocess : prog -> void
(define (preprocess stmts)
  (match stmts
    [(cons `(label ,label) rest)
      (hash-set! stmt-map label stmts)
      (preprocess rest)]
    [(cons _ rest)
      (preprocess rest)]
    [() (void)]))
Make it finite!
(struct state^ { stmts^ env^ })
\texttt{stmts}^* = \texttt{stmt}\ast

\texttt{env}^* = \texttt{var} \#=> \texttt{abstract-integer}
inject : prog -> state

step : state -> state
Questions?

matt.might.net
@mattmight