Static Program Analysis
- Origins and Applications -

• Origins:
  Compilers – checking the applicability of optimizing program transformations,
  examples:
  - moving computations from run time to compile time
  - avoiding redundant computations

• Hot application area:
  Verification – proving safety properties
  examples:
  - proving the absence of run-time errors
Literature used in the course

Problem:
Determine at each program point the values that program variables have every time control reaches this point.

Purpose:
Folding away (sub-)expressions whose operands are statically known, evaluate them at compile time instead of at run time.
a <- 1;
b <- 2;
c <- a + b;
c < 5

The necessary ingredients:
Disadvantage of this representation:
Need to associate information with positions before and after statements.

Alternative:

Statements at edges
Information at nodes
Outgoing edges at conditionals labeled with condition and negation of condition
Program Analysis should be Semantics-based

statements labeling edges have a (concrete) semantics, called \textit{concrete edge effects}.

They typically transform the state before the execution of the statement to the state after the execution of the statement.

For a semantics-based static program analysis edges also have an abstract semantics, called \textit{abstract edge effects}.

In the case of constant folding, they transform abstract variable bindings into abstract variable bindings.

Alternative specifications:
NNH: equalities
SWH: inequalities