Complexity of iterative analysis

How long does iterative analysis take?

`l`: depth of loop nesting

`n`: # of stmts in loop

`t`: time to execute one flow function

`k`: height of lattice

Another example: integer range analysis

For each program point,
for each integer-typed variable,
calculate (an approximation to) the set of integer values
that can be taken on by the variable

- use info for constant folding comparisons,
  for eliminating array bounds checks,
  for (in)dependence testing of array accesses,
  for eliminating overflow checks

What domain to use?

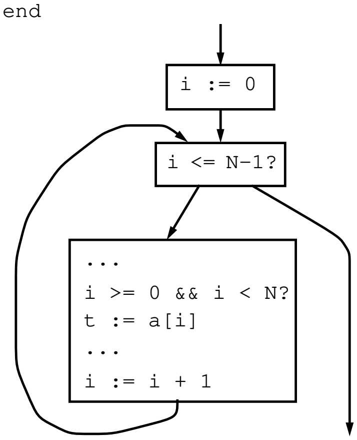
- what is its height?

What flow functions to use?

- are they monotonic?

Example

```
for i := 0 to N-1
  ... a[i] ...
end
```



Widening operators

If domain is tall, then can introduce artificial generalizations
(called **widenings**) when merging at loop heads

- can guarantee a small number of widenings possible

Abstract interpretation

[Another name for dataflow analysis, with different connotations]

Describe analysis in terms close to semantics of operations, enabling careful reasoning about correctness of analysis

Step 1: define concrete semantics of operators in IR, in terms of concrete domain

For each analysis:

Step 2: define an abstract semantics of operators in IR, in terms of abstract domain

Step 3: define abstraction function relating elements of concrete domain to elements of abstract domain

Step 4: prove abstract semantics is safe approximation to concrete semantics

Concrete semantics

Define real semantics of evaluating the IR; an interpreter

Concrete domain: map from vars to (values or variables)

Concrete flow functions:

$\text{Sem}_{\texttt{x := } N}$:
$\text{Info}_{succ} = \text{Info}_{pred}[\texttt{x} \to N]$

$\text{Sem}_{\texttt{x := \&y}}$:
$\text{Info}_{succ} = \text{Info}_{pred}[\texttt{x} \to \texttt{y}]$

$\text{Sem}_{\texttt{x := y + z}}$:
`let` rhs = $\text{Info}_{pred}(\texttt{y}) + \text{Info}_{pred}(\texttt{z})$ `in`
$\text{Info}_{succ} = \text{Info}_{pred}[\texttt{x} \to \text{rhs}]$

$\text{Sem}_{\texttt{*p := *q + *r}}$:
`let` rhs = $\text{Info}_{pred}(\text{Info}_{pred}(\texttt{q})) + \text{Info}_{pred}(\text{Info}_{pred}(\texttt{r}))$ `in`
$\text{Info}_{succ} = \text{Info}_{pred}[\text{Info}_{pred}(\texttt{p}) \to \text{rhs}]$

Concrete control flow

$\text{Sem}_{\texttt{if x < y goto L1 else L2}}$:
`if` $\text{Info}_{pred}(\texttt{x}) < \text{Info}_{pred}(\texttt{y})$
`then` $\text{Info}_{\texttt{L1}} = \text{Info}_{pred}$
`else` $\text{Info}_{\texttt{L2}} = \text{Info}_{pred}$

[Loops awkward in this format.
Recursion or machine configurations better.]

An abstraction: collecting semantics

Collecting semantics: abstract semantics whose domain is the set of possible values from the concrete domain

- collect together all values seen during any execution

Collecting domain: map from vars to sets of (values or variables)

- tuple domain of powerset domains, ordered by $\supseteq$

Collecting flow functions:

$\text{Sem}_{\texttt{x := } N}$:
$\text{Info}_{succ} = \text{Info}_{pred}[\texttt{x} \to \{N\}]$

$\text{Sem}_{\texttt{x := \&y}}$:
$\text{Info}_{succ} = \text{Info}_{pred}[\texttt{x} \to \{\texttt{y}\}]$

$\text{Sem}_{\texttt{x := y + z}}$:
`let` rhs = $\{ y+z \mid y \in \text{Info}_{pred}(\texttt{y}) \wedge z \in \text{Info}_{pred}(\texttt{z}) \}$ `in`
$\text{Info}_{succ} = \text{Info}_{pred}[\texttt{x} \to \text{rhs}]$

$\text{Sem}_{\texttt{*p := *q + *r}}$:
`let` rhs = $\{ y+z \mid q \in \text{Info}_{pred}(\texttt{q}) \wedge r \in \text{Info}_{pred}(\texttt{r}) \wedge y \in \text{Info}_{pred}(q) \wedge z \in \text{Info}_{pred}(r) \}$,
lhs = $\{ p \mid p \in \text{Info}_{pred}(\texttt{p}) \}$ `in`
$\text{Info}_{succ} = \text{Info}_{pred}[\ldots, p_i \to \text{Info}_{pred}(p_i) \cup \text{rhs}, \ldots], \forall p_i \in \text{lhs}$

Collecting control flow

Sem$_{\texttt{if x < y goto L1 else L2}}$:

`let` outcomes = { x < y | x ∈ Info$_{pred}$(`x`) ∧ y ∈ Info$_{pred}$(`y`) } `in`

`if` true ∈ outcomes `then` Info$_{\texttt{L1}}$ = Info$_{pred}$

`if` false ∈ outcomes `then` Info$_{\texttt{L1}}$ = Info$_{pred}$

Merge:

Info$_{succ}$ = {..., `x` → $\cup_{p \in pred}$ Info$_p$(`x`), ...}

Initial conditions: Info$_{entry}$ = {..., `x` → {}, ...}

Becoming computable

Concrete semantics is uncomputable:
must run program until it halts to get final answer

Collecting semantics is also uncomputable:
must run program until it halts to collect all intermediate answers

Need to abstract away some of the precision of collecting interpretation to get a computable abstract interpretation

- go from infinitely tall powerset domain to finite-height domain

E.g. constant propagation domain, range analysis domain, domain of signs, domain of possible pointer targets, ...

Abstraction function and correctness

To connect concrete and abstract domain(s), need abstraction function

Abstraction function *A* takes an element of input domain and returns the most specific corresponding element of abstract domain

- for collecting semantics: $A(d_{concrete}) = \{d_{concrete}\}$
- for more abstract semantics:
  squash out precision in collecting semantics somehow

Safety rule: if concrete semantics produces element $d_{concrete}$, abstract semantics must produce some element $d_{abstract}$ such that $A(d_{concrete}) \le d_{abstract}$

- true for collecting semantics
- relating collecting flow functions to abstract flow functions shouldn't be hard