Data-flow Analysis: Theoretical Foundations - Part 2

Y.N. Srikant

Department of Computer Science
Indian Institute of Science
Bangalore 560 012

NPTEL Course on Compiler Design

Foundations of Data-flow Analysis

- Basic questions to be answered
  1. Under what situations is the iterative DFA algorithm correct?
  2. How precise is the solution produced by it?
  3. Will the algorithm converge?
  4. What is the meaning of a "solution"?
- The above questions can be answered accurately by a DFA framework
- Further, reusable components of the DFA algorithm can be identified once a framework is defined
- A DFA framework $(D, V, \wedge, F)$ consists of

  D : A direction of the dataflow, either forward or backward

  V : A domain of values

  $\wedge$: A meet operator $(V, \wedge)$ form a semi-lattice

  F : A family of transfer functions, $V \longrightarrow V$

  F includes constant transfer functions for the ENTRY/EXIT nodes as well

Properties of the Iterative DFA Algorithm

- If the iterative algorithm converges, the result is a solution to the DF equations

  Proof: If the equations are not satisfied by the time the loop ends, atleast one of the *OUT* sets changes and we iterate again
- If the framework is monotone, then the solution found is the maximum fixpoint (MFP) of the DF equations
  An MFP solution is such that in any other solution, values of $IN[B]$ and $OUT[B]$ are $\leq$ the corresponding values of the MFP (i.e., less precise)

  Proof: We can show by induction that the values of $IN[B]$ and $OUT[B]$ only decrease (in the sense of $\leq$ relation) as the algorithm iterates

Properties of the Iterative DFA Algorithm (2)

- If the semi-lattice of the framework is monotone and is of finite height, then the algorithm is guaranteed to converge

  Proof: Dataflow values decrease with each iteration
  Max no. of iterations = height of the lattice × no. of nodes in the flow graph

Meaning of the Ideal Data-flow Solution

- Find all possible execution paths from the start node to the beginning of *B*
- (Assuming forward flow) Compute the data-flow value at the end of each path (using composition of transfer functions) and apply the $\wedge$ operator to these values to find their *glb*
- No execution of the program can produce a *smaller* value for that program point

$$IDEAL[B] \quad = \quad \bigwedge_{P,\ a\ possible\ execution\ path\ from\ start\ node\ to\ B} f_P(v_{init})$$

- Answers greater (in the sense of $\leq$) than IDEAL are incorrect (one or more execution paths have been ignored)
- Any value smaller than or equal to IDEAL is conservative, *i.e.,* safe (one or more infeasible paths have been included)
- Closer the value to IDEAL, more precise it is

Meaning of the Meet-Over-Paths Data-flow Solution

- Since finding all execution paths is an undecidable problem, we approximate this set to include all paths in the flow graph

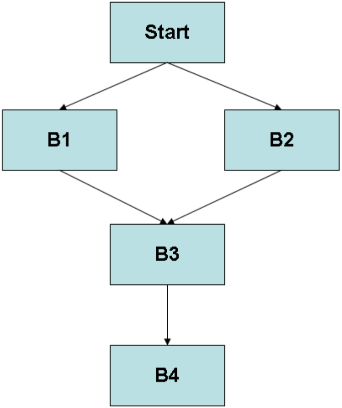
$$MOP[B] = \bigwedge_{P,\ a\ path\ from\ start\ node\ to\ B} f_P(v_{init})$$

- $MOP[B] \leq IDEAL[B]$, since we consider a superset of the set of execution paths

Meaning of the Maximum Fixpoint Data-flow Solution

- Finding all paths in a flow graph may still be impossible, if it has cycles
- The iterative algorithm does not try this
  - It visits all basic blocks, not necessarily in execution order
  - It applies the $\wedge$ operator at each join point in the flow graph
  - The solution obtained is the Maximum Fixpoint solution (MFP)
- If the framework is distributive, then the MOP and MFP solutions will be identical
- Otherwise, with just monotonicity, $MFP \leq MOP \leq IDEAL$, and the solution provided by the iterative algorithm is safe

Example to show $MFP \leq MOP$



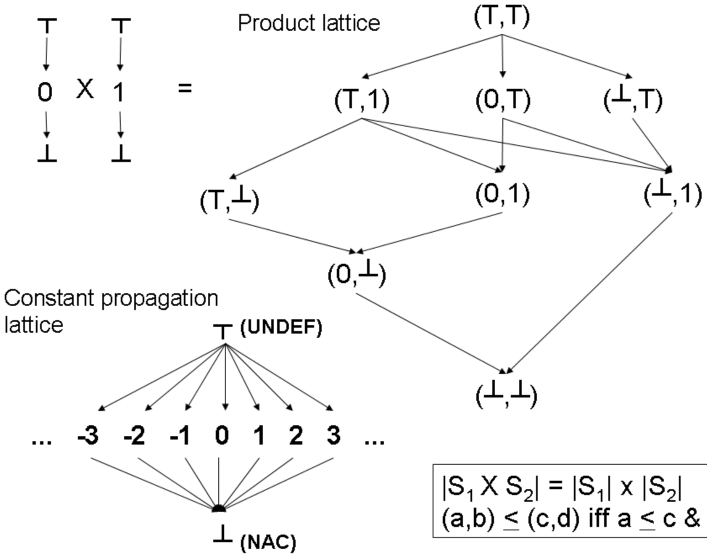
Example to show $MFP \leq MOP$ (2)

- There are two paths from Start to B4: $Start \rightarrow B1 \rightarrow B3 \rightarrow B4$ and $Start \rightarrow B2 \rightarrow B3 \rightarrow B4$
- $MOP[B4] = ((f_{B3} \cdot f_{B1}) \wedge (f_{B3} \cdot f_{B2}))(v_{init})$
- In the iterative algorithm, if we chose to visit the nodes in the order $(Start, B1, B2, B3, B4)$, then $IN[B4] = f_{B3}(f_{B1}(v_{init}) \wedge f_{B2}(v_{init}))$
- Note that the $\wedge$ operator is being applied differently here than in the *MOP* equation
- The two values above will be equal only if the framework is distributive
- With just monotonicity, we would have $IN[B4] \leq MOP[B4]$

Constant Propagation Framework - Data-flow Values

- The lattice for a single variable in the CP framework is shown in the next slide
- An example of product of two lattices is in the next slide
- DF values in the RD framework can also be considered as
  - values in a product of lattices of definitions
  - one lattice for each definition, with ϕ as $\top$ and {d} as the only other element
- The lattice of the DF values in the CP framework
  - Product of the semi-lattices of the variables (one lattice for each variable)

Product of Two Lattices and Lattice of Constants



$|S_1 \times S_2| = |S_1| \times |S_2|$

$(a,b) \leq (c,d)$ iff $a \leq c$ & $b \leq d$

CP Framework - The $\wedge$ (meet) Operator

- In a product lattice, $(a_1, b_1) \leq (a_2, b_2)$ iff $a_1 \leq_A a_2$ and $b_1 \leq_B b_2$ assuming $a_1, a_2 \in A$ and $b_1, b_2 \in B$
- Each variable is associated with a map m
- $m(v)$ is the abstract value (as in the lattice) of the variable v in a map m
- Each element of the product lattice is a similar, but “larger” map m
  - which is defined for all variables, and
  - where $m(v)$ is the abstract value of the variable v
- Thus, $m \leq m'$ (in the product lattice), iff for all variables v, $m(v) \leq m'(v)$, OR, $m \wedge m' = m''$, if $m''(v) = m(v) \wedge m'(v)$, for all variables v

Transfer Functions for the CP Framework

- Assume one statement per basic block
- Transfer functions for basic blocks containing many statements may be obtained by composition
- $m(v)$ is the abstract value of the variable v in a map m.
- The set F of the framework contains transfer functions which accept maps and produce maps as outputs
- F contains an identity map
- Map for the *Start* block is $m_0(v) = \mathit{UNDEF}$, for all variables v
- This is reasonable since all variables are undefined before a program begins

Transfer Functions for the CP Framework

- Let f_s be the transfer function of the statement s
- If $m' = f_s(m)$, then f_s is defined as follows
  1. If s is not an assignment, f_s is the identity function
  2. If s is an assignment to a variable x, then $m'(v) = m(v)$, for all $v \neq x$, provided, one of the following conditions holds
     (a) If the RHS of s is a constant c, then $m'(x) = c$
     (b) If the RHS is of the form $y + z$, then

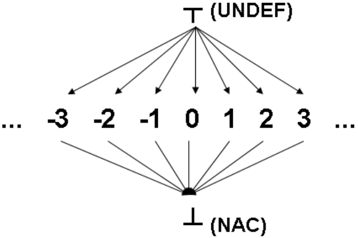
$$\begin{aligned} m'(x) &= m(y) + m(z),\ \textit{if}\ m(y)\ \textit{and}\ m(z)\ \textit{are constants} \\ &= NAC,\ \textit{if either}\ m(y)\ \textit{or}\ m(z)\ \textit{is}\ NAC \\ &= UNDEF,\ \textit{otherwise} \end{aligned}$$

     (c) If the RHS is any other expression, then $m'(x) = NAC$

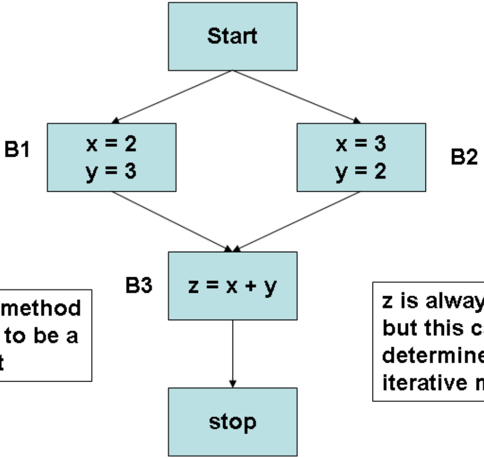
Monotonicity of the CP Framework

It must be noted that the transfer function ($m' = f_s(m)$) always produces a "lower" or same level value in the CP lattice, whenever there is a change in inputs

$m(y)$	$m(z)$	$m'(x)$
UNDEF	UNDEF	UNDEF
	c_2	UNDEF
	NAC	NAC
c_1	UNDEF	UNDEF
	c_2	$c_1 + c_2$
	NAC	NAC
NAC	UNDEF	NAC
	c_2	NAC
	NAC	NAC



Non-distributivity of the CP Framework



The iterative method determines z to be a non-constant

z is always a constant but this cannot be determined by the iterative method

Non-distributivity of the CF Framework - Example

- If f_1, f_2, f_3 are transfer functions of $B1, B2, B3$ (resp.), then $f_3(f_1(m_0) \wedge f_2(m_0)) < f_3(f_1(m_0)) \wedge f_3(f_2(m_0))$ as shown in the table, and therefore the CF framework is non-distributive

m	$m(x)$	$m(y)$	$m(z)$
m_0	UNDEF	UNDEF	UNDEF
$f_1(m_0)$	2	3	UNDEF
$f_2(m_0)$	3	2	UNDEF
$f_1(m_0) \wedge f_2(m_0)$	NAC	NAC	UNDEF
$f_3(f_1(m_0) \wedge f_2(m_0))$	NAC	NAC	NAC
$f_3(f_1(m_0))$	2	3	5
$f_3(f_2(m_0))$	3	2	5
$f_3(f_1(m_0)) \wedge f_3(f_2(m_0))$	NAC	NAC	5