Compiler Construction

Various Dataflow Analysis

Optimizing Compiler

Dataflow analysis is the first step
towards optimizing compilers

An **dataflow analysis** of a CFG collects information about the execution of the program (for instance, how definitions and uses are related to each other). An

Optimizing Compiler transforms programs to improve their efficiency without changing their output.

Optimizing Compiler

- How definitions and uses are related to each other?
- What value a variable may have at a given point?
- Constant propagation?
- Common sub-expression elimination?
- Copy propagation?
- Dead Code Elimination?
- ...?

Full employment theorem for compiler writer

> *Computability theory shows that it will always be possible to invent new optimizing transformations*

> *It can be proven that for each ”optimizing compiler” there is another one that beats it (which is therefore ”more optimal”).*

Reaching definitions (1/2)

For many optimizations we need to see if a particular assignment of t can affect the value of t at another point in the program.

Definition

An ambiguous definition is a statement that might or might not assign a temporary t. For instance, a call may sometimes modify t and sometimes not.

Reaching definitions (2/2)

Reaching definitions can be expressed as a solution of dataflow equations

$$\begin{aligned} \text{begin}[n] &= \bigcup_{p \in \text{pred}[n]} \text{end}[p] \\ \text{end}[n] &= \text{gen}[n] \cup (\text{begin}[n] \setminus \text{kill}[n]) \end{aligned}$$

Terminology

- **gen**: when enter this statement, we know that we will reach its end
- **kills**: any statement that invalidates a *gen*
- **begin[n]**: which statements can reach the begining of statement *n*
- **end[n]**: which statements can reach the end of statement *n*

Example (1/2)

```
    a := 5
    c := 1
L1: if c > a goto L2
    c := c + c
    goto L1
L2: a := c - a
    c := 0
```

Example (2/2)

	gen	*kills*	*begin*	*end*	*begin*	*end*	*begin*	*end*
1	1	6						
2	2	4,7						
3								
4	4	2,7						
5								
6	6	1						
7	7	2,4						

$$\begin{aligned} \text{begin}[n] &= \bigcup_{p \in \text{pred}[n]} \text{end}[p] \\ \text{end}[n] &= \text{gen}[n] \cup (\text{begin}[n] \setminus \text{kills}[n]) \end{aligned}$$

Example (2/2)

			1st step					
	gen	*kills*	*begin*	*end*	*begin*	*end*	*begin*	*end*
1	1	6		1				
2	2	4,7	1	1,2				
3			1,2	1,2				
4	4	2,7	1,2	1,4				
5			1,4	1,4				
6	6	1	1,2	2,6				
7	7	2,4	2,6	6,7				

$$\mathrm{begin}[n] = \bigcup_{p \in \mathrm{pred}[n]} \mathrm{end}[p]$$

$$\mathrm{end}[n] = \mathrm{gen}[n] \cup (\mathrm{begin}[n] \setminus \mathrm{kills}[n])$$

Example (2/2)

	gen	kills	1st step begin	1st step end	2nd step begin	2nd step end	begin	end
1	1	6		1		1		
2	2	4,7	1	1,2	1	1,2		
3			1,2	1,2	1,2,4	1,2,4		
4	4	2,7	1,2	1,4	1,2,4	1,4		
5			1,4	1,4	1,4	1,4		
6	6	1	1,2	2,6	1,2,4	2,4,6		
7	7	2,4	2,6	6,7	2,4,6	6,7		

$$\text{begin}[n] = \bigcup_{p \in \text{pred}[n]} \text{end}[p]$$

$$\text{end}[n] = \text{gen}[n] \cup (\text{begin}[n] \setminus \text{kills}[n])$$

Example (2/2)

			1st step		2nd step		3rd step	
	gen	*kills*	*begin*	*end*	*begin*	*end*	*begin*	*end*
1	1	6		1		1		1
2	2	4,7	1	1,2	1	1,2	1	1,2
3			1,2	1,2	1,2,4	1,2,4	1,2,4	1,2,4
4	4	2,7	1,2	1,4	1,2,4	1,4	1,2,4	1,4
5			1,4	1,4	1,4	1,4	1,4	1,4
6	6	1	1,2	2,6	1,2,4	2,4,6	1,2,4	2,4,6
7	7	2,4	2,6	6,7	2,4,6	6,7	2,4,6	6,7

$$\begin{aligned} \mathrm{begin}[n] &= \bigcup_{p \in \mathrm{pred}[n]} \mathrm{end}[p] \\ \mathrm{end}[n] &= \mathrm{gen}[n] \cup (\mathrm{begin}[n] \setminus \mathrm{kills}[n]) \end{aligned}$$

Example (2/2)

			1st step		2nd step		3rd step	
	gen	*kills*	*begin*	*end*	*begin*	*end*	*begin*	*end*
1	1	6		1		1		1
2	2	4,7	1	1,2	1	1,2	1	1,2
3			1,2	1,2	1,2,4	1,2,4	1,2,4	1,2,4
4	4	2,7	1,2	1,4	1,2,4	1,4	1,2,4	1,4
5			1,4	1,4	1,4	1,4	1,4	1,4
6	6	1	1,2	2,6	1,2,4	2,4,6	1,2,4	2,4,6
7	7	2,4	2,6	6,7	2,4,6	6,7	2,4,6	6,7

$$\begin{aligned} \text{begin}[n] &= \bigcup_{p \in \text{pred}[n]} \text{end}[p] \\ \text{end}[n] &= \text{gen}[n] \cup (\text{begin}[n] \setminus \text{kills}[n]) \end{aligned}$$

Constant folding example: only one definition of a reaches statement 3, so we can replace $c > a$ by $c > 5$.

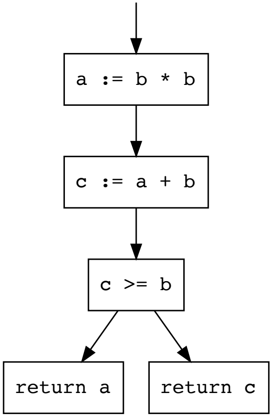
Common subexpression elimination

Can we eliminate duplicate computation?

$$\begin{aligned} \text{begin}[n] &= \bigcap_{p \in \text{pred}[n]} \text{end}[p] \\ \text{end}[n] &= \text{gen}[n] \cup (\text{begin}[n] \setminus \text{kills}[n]) \end{aligned}$$

In this situation, the sets are now sets of expressions.

Conservative Approximation



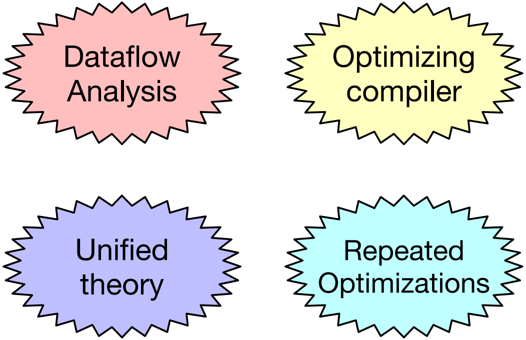
Other optimizations

- Copy Propagation
- Dead code elimination
- Alias analysis
- Lazy Code Motion
- ...

Applying optimizations repeatedly

- **Cutoff**: perform no more than k rounds
- **Cascading analysis**: predict the cascade of effects of an optimization. Value numbering is a typical case of cascading analysis
- **Incremental dataflow analysis**: patch the dataflow after applying an optimization.

Summary


Dataflow Analysis
Optimizing compiler
Unified theory
Repeated Optimizations