Sources of Redundancy

DEPT. OF REDUNDANCY DEPT.

INTRODUCTION

A catalogue of optimizing transformations which a compiler can make on a program is presented in this paper. The catalogue does not pretend to include all possible transformations; what is presented is a categorization of most of the transformations which are currently reasonably well understood.

The basic purpose of the transformations presented is to improve the execution time of compiled programs. Some attention is given to execution space, but no attention is given to optimizing compile time or to the more general questions of total job or system optimization.

The term optimization is a misnomer in that it source: http://www.cs.rochester.edu/~keith/512/RiceOnly/1971-allen-catalogue.pdf

Macros

• Is there redundancy in the code below?

```c
#define larger(x,y) (x)>(y)? (x): (y)
#define smaller(x,y) (x)>(y)? (y): (x)
... ...
I = larger(a+b, a-b)
S = smaller(a+b, a-b)
```
Redundant Expression Elimination

Function Abstraction

• Is there redundancy in the code below?

```c
if (find_min(list) > 0)
    return find_max(list);
```

Finding Common Subexpression

Finding Identical Expressions

• Assumptions
  • straight line code
  • different names means different variables

• Example

```c
m = 2 * y * z
n = 3 * y * z
o = 2 * y - z
```

• Optimization?
  • side effects?
  • Representation?
  • Potential issues?
Problems

- Same name does not mean same value
- Side effects?
- Other potential issues?

The Idea: Assigning Numbers to Values

- The example from Allen-Cocke, 1971

\[
\begin{align*}
A \to A \cdot B \\
C & \to A \\
D & \to C \cdot B
\end{align*}
\]

Local Value Numbering

Value Numbering

- Assumptions
  - Straight-line code, at most 2 operands on rhs
  - Algorithm for finding redundancy

- Problems
  - Assignments, pointers, order of operands, constants, code generation

Value Numbering

- Example

\[
\begin{align*}
x & \to a \cdot b \\
c & \to a \\
y & \to c \cdot b
\end{align*}
\]

- Key property
  - Number the values: same number \(\to\) same value

- Extensions
  - What about "d = d + a"?
  - Stewart method
    - e = f + g
    - h = e - f
EAC, Chapter 8

- Time to substantiate the class slogan
- What is code optimization?
  - “is to discover, at compile time, information about the run-time behavior of the program and to use that information to improve the code generated by the compiler”
- Engineering A Compiler, Cooper & Torczon
- Extensions later
  - it finds more than just redundancy
  - what other attributes are useful?
  - it doesn’t have to be just performance
  - it doesn’t have to be compile time
  - it doesn’t have to be code generation

Summary

- Sources of redundancy
  - data abstraction
  - language implementation
  - modular design, code reuse (e.g. libraries)
- Redundancy removal
  - representation of code
  - name based redundancy analysis
  - the problem of re-assignments
- Value numbering
  - two names have same value if their numbers are the same

Next
- attendance required
- weekly Friday recitation session 11-12pm, Room 601.
- compiler implementation in class next Monday

Reviews of EAC on Amazon

- “Between the Tiger and the Dragon. I found the book to be a nice balance between the deep theory of Aho et al’s Dragon book and the implementation focus of Appel’s Tiger book.”
- “By contrast, this book (Cooper/Torczon) is not only digestible (nice presentation, not overly terse), but it also covers new and interesting algorithms and data-structures.”
- “Pseudo code was given ... but there were always special cases... I liked the constant summaries, but when I faced the questions at the end of the chapters, I quickly realized I hadn’t digested the material fully”
- “Concise, implementation-oriented, pragmatic but thoughtful”