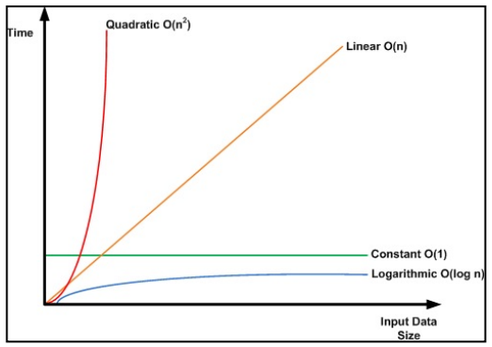
Recurrence Extraction from Lazy Programs

Caroline Churney, Norman Danner

Wesleyan University

Introduction to Cost Analysis

- **Cost**: number of operations required to obtain result of program
  - Measure of the efficiency of a program



Strict vs. Lazy Programming

- **Strict:** immediate evaluation
- **Lazy:** only evaluates if necessary for the result of the program

What is Clairvoyant Call by Value?

- Alternative model for lazy evaluation
- Utilizes the concept of nondeterminism
  - Interpreter makes choices during execution depending on the necessity of bindings for evaluation of the program
- Results in derivation tree with minimum cost or the "maximally lazy computation cost"

Goals

- Gain intuitions surrounding the clairvoyant call by value approach to lazy cost analysis
- Extract cost recurrences from lazy programs
- Develop tools that allow us to track the evaluation and cost of various programs

Our Work

- Coded parser and interpreter to analyze the cost and operations of programs
- Studied principles of lazy cost analysis with guidance from Hackett and Hutton's work on clairvoyance

Future Work

- Extend Hackett and Hutton's work in order to formalize the recurrence extraction process for lazy programs
- Adapt our interpreter to a lazy language in order to track operations and cost of more lazy programs

For more information on Hackett and Hutton's clairvoyant model: Jennifer Hackett and Graham Hutton. 2019. Call-by-Need Is Clairvoyant Call-by-Value. Proc. ACM Program. Lang. 3, ICFP, Article 114 (August 2019), 23 pages. https://doi.org/10.1145/3341718

Reverse

```
fun rev xs =
   case xs of
        nil –> nil
        x′:xs′ –>
             let
               a′=rev xs′
               n=nil
               b′=x′:n
             in
               app a′ b′
```

```
fun app xs ys =
   case xs of
        nil –> ys
        x′:xs′ –>
             let
               a = app xs′ ys
             in
               x′:a
```

Strict

rev [1…n]
=rev[2…n]@[1]
T(n-1)
.
.
.
=[n…2]@[1]
T_{app}(n-1)
.
.
.
=[n…1]

Lazy

rev [1…n]
=rev[2…n]@[1]
T(n-1)
.
.
.
=(e::es)@[1]
T_{app}(1)
= e::(es@[1])

Recurrences

If append has a constant cost of 1, then strict evaluation results in quadratic time and lazy evaluation results in linear time.

Strict

T(0)=0
T(n)=T(n-1)+n-1

Lazy

T(0)=0
T(n)=T(n-1)+1