Needed Computations
Shortcutting Needed Steps

Sergio Antoy
Portland State University
Joint work with Jacob Johannsen and Steven Libby

Termgraph 2014 – Vienna, Austria
Thanks NSF CCF-1317249

Outline

- Part 1: an *optimal* implementation.
- Part 2: a *better* one.

What is the meaning of "optimal"?

A bit of theory

The fundamental result of computations in Orthogonal Term Rewriting Systems:

Every reducible term has a redex that is reduced in **every** computation to its normal form, if it exists.

Significant consequences and some subtleties ...

Just do it

- If you discover that a redex is needed, then reduce it.
- Can't do any better.
- Basis for optimal computations (including non-determinism and narrowing).

However

Needed redexes:

- can't always be known,
- move around,
- may be cloned.

Our Environment

- *Graph* not term rewriting.
- Inductively sequential systems.
- Needed redexes easily found.
- Redex has identity, no clones.

Inductive sequentiality

Operations are defined by constructor-based left-linear rules.

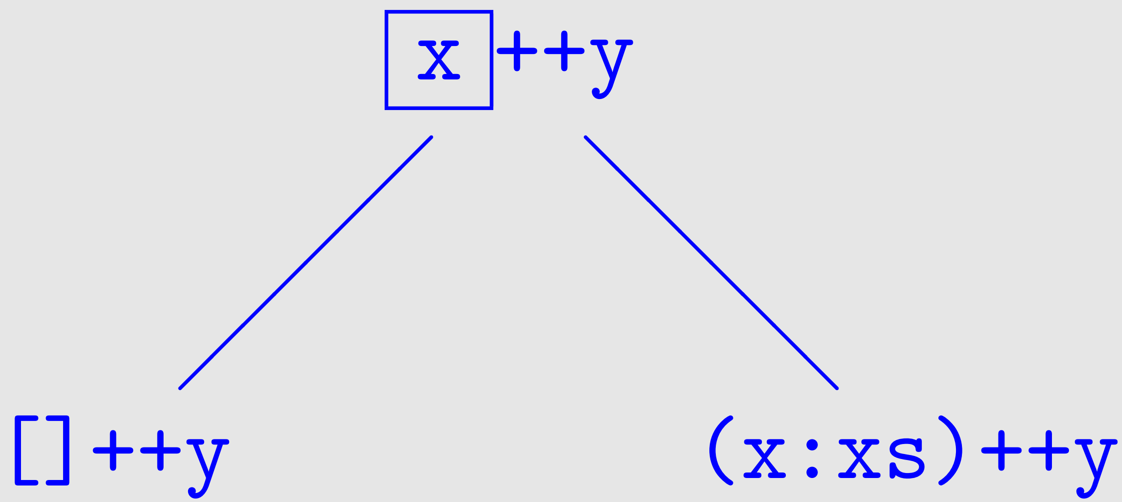
A definition, e.g., the usual list concatenation:

$$\texttt{[] ++ y} \rightarrow \texttt{y}$$
$$\texttt{(x : xs) ++ y} \rightarrow \texttt{x : (xs ++ y)}$$

is like a structural induction case distinction.

Definitional Trees

A hierarchical structure of the rules of an operation that captures the structural induction-like case distinction of a definition:



Compilation

- Data: graphs labeled by signature symbols.
- Two functions: **H** (head) and **N**(norm).
- **H** derives argument to head constructor form.
  Generated by a traversal of definitional trees.
- **N** derives argument to constructor form.
  Obtained from signature.

Object Code (**H**)

Head function for ++.
Pattern matching is a notational convenience.
Apply rules in textual order.

```
H([]++[]) = []
H([]++(y:ys)) = (y:ys)
H([]++y) = H(y)
H((x:xs)++y) = x:(xs++y)
H(x++y) = H(H(x)++y)
```

Object Code (**N**)

Norm function for List constructors and `++`.
Pattern matching is a notational convenience.

```
N([]) = []
N(x:xs) = N(x):N(xs)
N(x++y) = N(H(x++y))
```

Properties

- Call-by-value (innermost).
- "As if" only needed redexes are reduced.
- Normalizing computations (for values).
- A "very good" implementation/strategy.
  - optimal
  - no lookahead
  - simple

A more Efficient Implementation

- Transform object code
- Shortcut some rewrite steps
- Fewer allocated nodes
- Fewer (pattern) matched nodes
- “Fewer” may be zero

Transformation

Two phases:

- Remove application of **H** to a variable (specialize the variable).
- Introduce composition of **H** and symbol (from the above).

Phase 1 Example

An earlier rule of **H** for `++`:

$$\mathbf{H}(\texttt{[]++y}) = \mathbf{H}(\texttt{y})$$

specialize `y`:

$$\mathbf{H}(\texttt{[]++(u++v)}) = \mathbf{H}(\texttt{u++v})$$

do same for every other (list) operation.

Phase 2 Example

Rule obtained from phase 1:

$$\mathbf{H}(\texttt{[]++(u++v))} = \mathbf{H}(\texttt{u++v})$$

Introduce $\mathbf{H}_{\texttt{++}}$ as $\mathbf{H} \circ \texttt{++}$:

$$\mathbf{H}_{\texttt{++}}(\texttt{[],u++v}) = \mathbf{H}_{\texttt{++}}(\texttt{u,v})$$

do same everywhere in object code.
H is no longer invoked.

Benchmark Program

Define:

```
length [] = 0
length (_:xs) = 1 + length xs
```

Object code:

$$\mathbf{H}_{\mathtt{length}}(\mathtt{[]}) = \mathtt{0}$$
$$\mathbf{H}_{\mathtt{length}}(\mathtt{\_:xs})) = \mathbf{H}_{+}(\mathtt{1,length(xs)})$$
$$\ldots$$

Integers are built-in, addition defined accordingly.

Benchmark Results

l_1 and l_2 are long lists.

`length(`l_1`++`l_2`)`	C_R	T_R	O_R
rewrite steps	10	6	6
shortcut steps	0	4	4
node allocations	20	16	12
node matches	40	26	18

Entries are normalized with respect to the number of rewrite steps of C_R.

Pushing the Idea Further

O_R replaces in phase 1 rule:

$$\mathbf{H}(\texttt{length(\_:xs)}) \texttt{ = } \mathbf{H}(\texttt{1+length(xs)})$$

with:

$$\mathbf{H}(\texttt{length(\_:xs)}) \texttt{ = } \mathbf{H}(\texttt{1+}\mathbf{H}(\texttt{length(xs)}))$$

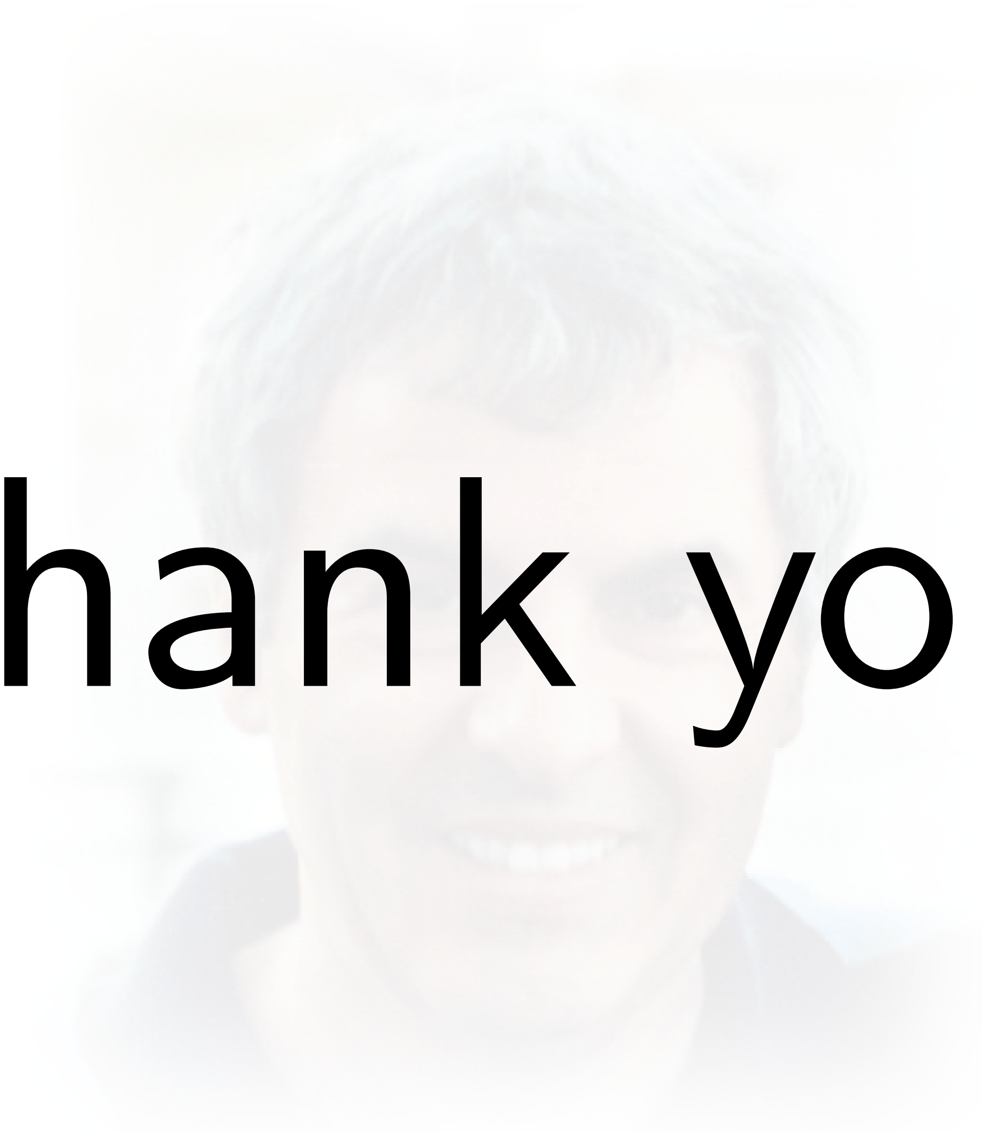
and then applies phase 2 of the transformation:

$$\mathbf{H}_{\texttt{length}}\texttt{(\_:xs))} \texttt{ = } \mathbf{H}_{+}\texttt{(1,}\mathbf{H}_{\texttt{length}}\texttt{(xs))}$$

Can still save the allocations of `1`.

Conclusion

- Two implementations of rewriting.
- One remarkably simple and optimal.
- One shortcuts needed steps (rewriting?).
- Executes with same building blocks.
- Conceptually interesting (optimal strategies).
- Practically useful (fewer resources).



Thank you