

From Photoshop to Programming Languages

An algebraic approach to computation

Brian Sutton

8 February 2005

Haskell: An unusual programming language

```
> 5 + 3
```

`<-- human input`

```
8
```

`<-- Haskell response`

```
> let coins = [25, 10, 5, 1]
```

```
> coins
```

```
[25, 10, 5, 1]
```

```
> head coins
```

```
25
```

```
> tail coins
```

```
[10, 5, 1]
```

```
> 100 : coins
```

```
[100, 25, 10, 5, 1]
```

Haskell: An unusual programming language

The infinite sequence 7, 7, 7, 7, 7,

```
> let sevens = 7 : sevens
```

```
> sevens
```

What will happen?

Haskell: An unusual programming language

The infinite sequence 7, 7, 7, 7, 7,

```
> let sevens = 7 : sevens
```

```
> sevens
```

What will happen?

- ▶ Error: "sevens" is undefined.

Haskell: An unusual programming language

The infinite sequence 7, 7, 7, 7, 7,

```
> let sevens = 7 : sevens
```

```
> sevens
```

What will happen?

- ▶ Error: "sevens" is undefined.
- ▶ Infinite loop.

Haskell: An unusual programming language

The infinite sequence 7, 7, 7, 7, 7,

```
> let sevens = 7 : sevens
```

```
> sevens
```

What will happen?

- ▶ Error: "sevens" is undefined.
- ▶ Infinite loop.
- ▶ Unravel the very fabric of the space-time continuum.

Haskell: An unusual programming language

The infinite sequence 7, 7, 7, 7, 7,

```
> let sevens = 7 : sevens
```

```
> sevens
```

```
[7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7,  
, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, ...
```

What will happen?

- ▶ Error: "sevens" is undefined.
- ▶ Infinite loop.
- ▶ Unravel the very fabric of the space-time continuum.

Haskell: An unusual programming language

`list !! n` gives the n th term of `list`.

```
> let coins = [25, 10, 5, 1]
```

```
> coins !! 2
```

```
5
```

What will happen now?

```
> let sevens = 7 : sevens
```

```
> sevens !! 2
```

Let's try it...

Haskell: An unusual programming language

`list !! n` gives the n th term of `list`.

```
> let coins = [25, 10, 5, 1]
```

```
> coins !! 2
```

```
5
```

What will happen now?

```
> let sevens = 7 : sevens
```

```
> sevens !! 2
```

```
7
```

Let's try it... **It works!**

`sevens` is an infinite sequence, and we can actually use it.

Plan

Two problems.

1. Simplify “Photoshop products.”
2. Understand infinite sequences in Haskell.

```
> let sevens = 7 : sevens
```

```
> sevens !! 2
```

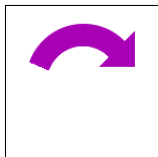
```
7
```

Similar solutions.

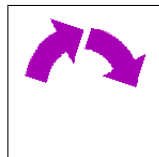
Arithmetic with Photoshop



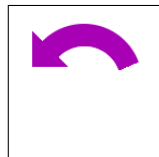
identity



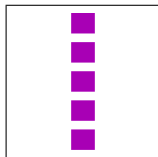
90° cw



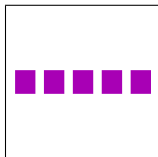
180°



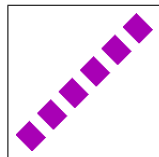
90° ccw



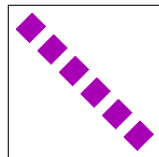
mirror across
vertical



mirror across
horizontal



mirror across
slash



mirror across
backslash

Arithmetic with Photoshop

We can “perform arithmetic” on the eight transformations.

$x \cdot y$ means “do y , then x .”

Examples:

- ▶ $(90^\circ \text{ cw}) \cdot (90^\circ \text{ cw}) = 180^\circ$
- ▶ $(90^\circ \text{ cw}) \cdot (\text{mirror across vertical}) = (\text{mirror across slash})$

In fact, all eight transformations can be produced from just two,

$$r = 90^\circ \text{ cw}$$

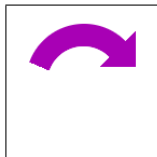
$$s = \text{mirror across vertical}$$

Arithmetic with Photoshop

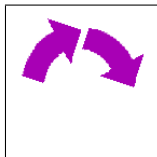
$r = 90^\circ$ cw, $s =$ mirror across vertical, $\epsilon =$ empty product



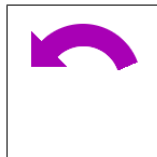
ϵ



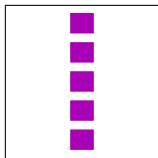
r



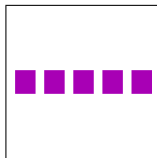
$r \cdot r$



$r \cdot r \cdot r$



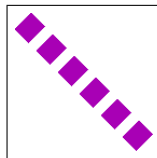
s



$r \cdot r \cdot s$



$r \cdot s$



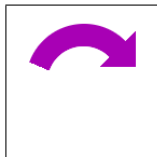
$s \cdot r$

Arithmetic with Photoshop

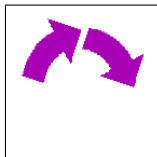
$r = 90^\circ$ cw, $s = \text{mirror across vertical}$, $\epsilon = \text{empty product}$



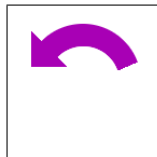
ϵ



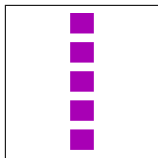
r



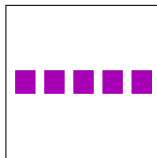
$r \cdot r$



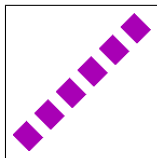
$r \cdot r \cdot r$



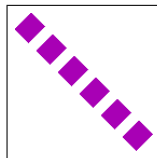
s



$r \cdot r \cdot s$



$r \cdot s$



$s \cdot r$

Modern Algebra: the dihedral group with eight elements.

Arithmetic with Photoshop

Simplify

$$r \cdot s \cdot r \cdot r \cdot s \cdot s \cdot r \cdot s \cdot r \cdot r \cdot r.$$

The answer should be one of the eight canonical forms:



ϵ



r



$r \cdot r$



$r \cdot r \cdot r$



s



$r \cdot r \cdot s$



$r \cdot s$



$s \cdot r$

Arithmetic with Photoshop

Simplify

$$r \cdot s \cdot r \cdot r \cdot s \cdot s \cdot r \cdot s \cdot r \cdot r \cdot r.$$

The answer should be one of the eight canonical forms:



ϵ



r



$r \cdot r$



$r \cdot r \cdot r$



s



$r \cdot r \cdot s$



$r \cdot s$



$s \cdot r$

The word problem:

Example:

$$r \cdot s \cdot s \cdot r \cdot s \cdot r \stackrel{?}{=} s \cdot r \cdot r \cdot r \cdot s$$

Answer: Identity \Leftrightarrow same canonical forms.

Plan

Two problems.

1. Compute canonical forms on a computer.

$$r \cdot s \cdot s \cdot r \Rightarrow r \cdot r$$

2. Understand infinite sequences in Haskell.

```
> let sevens = 7 : sevens
```

```
> sevens !! 2
```

```
7
```

Similar solutions.

Computing canonical forms

Basic idea

Input: u

Output: canonical form \bar{u}

1. Substitute.
2. Repeat.

Identities

▶ $s \cdot s = \epsilon$

▶ $r \cdot r \cdot r \cdot r = \epsilon$

▶ $r \cdot r \cdot r \cdot s = s \cdot r$

Computing canonical forms

Basic idea

Input: u

Output: canonical form \bar{u}

1. Substitute.
2. Repeat.

Identities

▶ $s \cdot s = \epsilon$

▶ $r \cdot r \cdot r \cdot r = \epsilon$

▶ $r \cdot r \cdot r \cdot s = s \cdot r$

Compute the canonical form for $r \cdot s \cdot r$.

$$r \cdot s \cdot r$$

Computing canonical forms

Basic idea

Input: u

Output: canonical form \bar{u}

1. Substitute.
2. Repeat.

Identities

▶ $s \cdot s = \epsilon$

▶ $r \cdot r \cdot r \cdot r = \epsilon$

▶ $r \cdot r \cdot r \cdot s = s \cdot r$

Compute the canonical form for $r \cdot s \cdot r$.

$$r \cdot s \cdot r$$

Computing canonical forms

Basic idea

Input: u

Output: canonical form \bar{u}

1. Substitute.
2. Repeat.

Identities

▶ $s \cdot s = \epsilon$

▶ $r \cdot r \cdot r \cdot r = \epsilon$

▶ $r \cdot r \cdot r \cdot s = s \cdot r$

Compute the canonical form for $r \cdot s \cdot r$.

$$\begin{aligned} r \cdot s \cdot r \\ = r \cdot r \cdot r \cdot r \cdot s \end{aligned}$$

Computing canonical forms

Basic idea

Input: u

Output: canonical form \bar{u}

1. Substitute.
2. Repeat.

Identities

▶ $s \cdot s = \epsilon$

▶ $r \cdot r \cdot r \cdot r = \epsilon$

▶ $r \cdot r \cdot r \cdot s = s \cdot r$

Compute the canonical form for $r \cdot s \cdot r$.

$$r \cdot s \cdot r$$

$$= r \cdot r \cdot r \cdot r \cdot s$$

Computing canonical forms

Basic idea

Input: u

Output: canonical form \bar{u}

1. Substitute.
2. Repeat.

Identities

▶ $s \cdot s = \epsilon$

▶ $r \cdot r \cdot r \cdot r = \epsilon$

▶ $r \cdot r \cdot r \cdot s = s \cdot r$

Compute the canonical form for $r \cdot s \cdot r$.

$$\begin{aligned} r \cdot s \cdot r \\ = r \cdot r \cdot r \cdot r \cdot s \end{aligned}$$

Computing canonical forms

Basic idea

Input: u

Output: canonical form \bar{u}

1. Substitute.
2. Repeat.

Identities

▶ $s \cdot s = \epsilon$

▶ $r \cdot r \cdot r \cdot r = \epsilon$

▶ $r \cdot r \cdot r \cdot s = s \cdot r$

Compute the canonical form for $r \cdot s \cdot r$.

$$\begin{aligned} & r \cdot s \cdot r \\ &= r \cdot r \cdot r \cdot r \cdot s \\ &= \epsilon \cdot s \end{aligned}$$

Computing canonical forms

Basic idea

Input: u

Output: canonical form \bar{u}

1. Substitute.
2. Repeat.

Identities

▶ $s \cdot s = \epsilon$

▶ $r \cdot r \cdot r \cdot r = \epsilon$

▶ $r \cdot r \cdot r \cdot s = s \cdot r$

Compute the canonical form for $r \cdot s \cdot r$.

$$r \cdot s \cdot r$$

$$= r \cdot r \cdot r \cdot r \cdot s$$

$$= \epsilon \cdot s$$

Computing canonical forms

Basic idea

Input: u

Output: canonical form \bar{u}

1. Substitute.
2. Repeat.

Identities

▶ $s \cdot s = \epsilon$

▶ $r \cdot r \cdot r \cdot r = \epsilon$

▶ $r \cdot r \cdot r \cdot s = s \cdot r$

Compute the canonical form for $r \cdot s \cdot r$.

$$\begin{aligned}r \cdot s \cdot r \\&= r \cdot r \cdot r \cdot r \cdot s \\&= \epsilon \cdot s \\&= s\end{aligned}$$

Computing canonical forms

Basic idea

Input: u

Output: canonical form \bar{u}

1. Substitute.
2. Repeat.

Identities

▶ $s \cdot s = \epsilon$

▶ $r \cdot r \cdot r \cdot r = \epsilon$

▶ $r \cdot r \cdot r \cdot s = s \cdot r$

But what prevents a computer from trying the following?

$$r \cdot s \cdot r$$

Computing canonical forms

Basic idea

Input: u

Output: canonical form \bar{u}

1. Substitute.
2. Repeat.

Identities

▶ $s \cdot s = \epsilon$

▶ $r \cdot r \cdot r \cdot r = \epsilon$

▶ $r \cdot r \cdot r \cdot s = s \cdot r$

But what prevents a computer from trying the following?

$$r \cdot s \cdot r$$

Computing canonical forms

Basic idea

Input: u

Output: canonical form \bar{u}

1. Substitute.
2. Repeat.

Identities

▶ $s \cdot s = \epsilon$

▶ $r \cdot r \cdot r \cdot r = \epsilon$

▶ $r \cdot r \cdot r \cdot s = s \cdot r$

But what prevents a computer from trying the following?

$$r \cdot s \cdot r$$

$$= r \cdot r \cdot r \cdot r \cdot s$$

Computing canonical forms

Basic idea

Input: u

Output: canonical form \bar{u}

1. Substitute.
2. Repeat.

Identities

▶ $s \cdot s = \epsilon$

▶ $r \cdot r \cdot r \cdot r = \epsilon$

▶ $r \cdot r \cdot r \cdot s = s \cdot r$

But what prevents a computer from trying the following?

$$r \cdot s \cdot r$$

$$= r \cdot r \cdot r \cdot r \cdot s$$

Computing canonical forms

Basic idea

Input: u

Output: canonical form \bar{u}

1. Substitute.
2. Repeat.

Identities

▶ $s \cdot s = \epsilon$

▶ $r \cdot r \cdot r \cdot r = \epsilon$

▶ $r \cdot r \cdot r \cdot s = s \cdot r$

But what prevents a computer from trying the following?

$$r \cdot s \cdot r$$

$$= r \cdot r \cdot r \cdot r \cdot s$$

Computing canonical forms

Basic idea

Input: u

Output: canonical form \bar{u}

1. Substitute.
2. Repeat.

Identities

▶ $s \cdot s = \epsilon$

▶ $r \cdot r \cdot r \cdot r = \epsilon$

▶ $r \cdot r \cdot r \cdot s = s \cdot r$

But what prevents a computer from trying the following?

$$r \cdot s \cdot r$$

$$= r \cdot r \cdot r \cdot r \cdot s$$

$$= r \cdot s \cdot r$$

Computing canonical forms

Basic idea

Input: u

Output: canonical form \bar{u}

1. Substitute.
2. Repeat.

Identities

▶ $s \cdot s = \epsilon$

▶ $r \cdot r \cdot r \cdot r = \epsilon$

▶ $r \cdot r \cdot r \cdot s = s \cdot r$

But what prevents a computer from trying the following?

$$r \cdot s \cdot r$$

$$= r \cdot r \cdot r \cdot r \cdot s$$

$$= r \cdot s \cdot r$$

Computing canonical forms

Basic idea

Input: u

Output: canonical form \bar{u}

1. Substitute.
2. Repeat.

Identities

▶ $s \cdot s = \epsilon$

▶ $r \cdot r \cdot r \cdot r = \epsilon$

▶ $r \cdot r \cdot r \cdot s = s \cdot r$

But what prevents a computer from trying the following?

$$r \cdot s \cdot r$$

$$= r \cdot r \cdot r \cdot r \cdot s$$

$$= r \cdot s \cdot r$$

$$= r \cdot r \cdot r \cdot r \cdot s$$

Computing canonical forms

Basic idea

Input: u

Output: canonical form \bar{u}

1. Substitute.
2. Repeat.

Identities

▶ $s \cdot s = \epsilon$

▶ $r \cdot r \cdot r \cdot r = \epsilon$

▶ $r \cdot r \cdot r \cdot s = s \cdot r$

But what prevents a computer from trying the following?

$$r \cdot s \cdot r$$

$$= r \cdot r \cdot r \cdot r \cdot s$$

$$= r \cdot s \cdot r$$

$$= r \cdot r \cdot r \cdot r \cdot s$$

$$= r \cdot s \cdot r$$

Computing canonical forms

Basic idea

Input: u

Output: canonical form \bar{u}

1. Substitute.
2. Repeat.

Identities

▶ $s \cdot s = \epsilon$

▶ $r \cdot r \cdot r \cdot r = \epsilon$

▶ $r \cdot r \cdot r \cdot s = s \cdot r$

But what prevents a computer from trying the following?

$$r \cdot s \cdot r$$

$$= r \cdot r \cdot r \cdot r \cdot s$$

$$= r \cdot s \cdot r$$

$$= r \cdot r \cdot r \cdot r \cdot s$$

$$= r \cdot s \cdot r$$

$$= \dots$$

Computing canonical forms

Basic idea

Input: u

Output: canonical form \bar{u}

1. Substitute.
2. Repeat.

Identities

▶ $s \cdot s = \epsilon$

▶ $r \cdot r \cdot r \cdot r = \epsilon$

▶ $r \cdot r \cdot r \cdot s = s \cdot r$

Three difficulties:

1. Which direction? LHS \rightarrow RHS or RHS \rightarrow LHS?
2. Which identity to apply?
3. When to stop?

Computing canonical forms

1. Which direction?

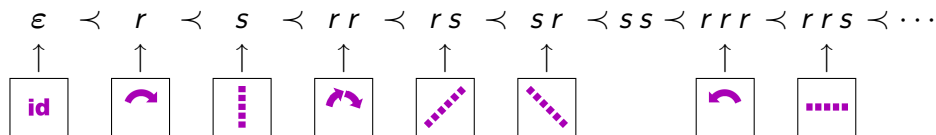
Words are ordered, first by length, then alphabetically.

$\epsilon \prec r \prec s \prec rr \prec rs \prec sr \prec ss \prec rrr \prec rrs \prec \dots$

Computing canonical forms

1. Which direction?

Words are ordered, first by length, then alphabetically.

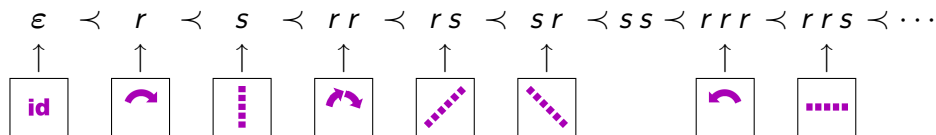


The canonical forms are at the bottom of the order.

Computing canonical forms

1. Which direction?

Words are ordered, first by length, then alphabetically.



The canonical forms are at the bottom of the order.

Which direction? Always move *down* the order.

Computing canonical forms

Three difficulties:

1. Which direction?
2. Which rule to apply?
3. When to stop?

Computing canonical forms

Three difficulties:

1. Which direction? down the ordering
2. Which rule to apply?
3. When to stop?

Lesson 1: $\epsilon \prec r \prec s \prec rs \prec ss \prec rrr \prec \dots$

Use rewriting rules instead of identities.

▶ $ss \rightarrow \epsilon$

▶ $rrrr \rightarrow \epsilon$

▶ $rrrs \rightarrow sr$

Computing canonical forms

Three difficulties:

1. Which direction? down the ordering
2. Which rule to apply?
3. When to stop? don't stop until you have to

Lesson 1: $\epsilon \prec r \prec s \prec rs \prec ss \prec rrr \prec \dots$

Use rewriting rules instead of identities.

▶ $ss \rightarrow \epsilon$

▶ $rrrr \rightarrow \epsilon$

▶ $rrrs \rightarrow sr$

Because of the ordering, eventually you have to stop.

Computing canonical forms

2. Which rule to apply?

▶ $ss \rightarrow \epsilon$

▶ $rrrr \rightarrow \epsilon$

▶ $rrrs \rightarrow sr$

Computing canonical forms

2. Which rule to apply?

▶ $ss \rightarrow \epsilon$

▶ $rrrr \rightarrow \epsilon$

▶ $rrrs \rightarrow sr$

Problem

Solution

$rrrss$ \Rightarrow rrr
 \Rightarrow srs

Computing canonical forms

2. Which rule to apply?

▶ $ss \rightarrow \epsilon$

▶ $rrrr \rightarrow \epsilon$

▶ $rrrs \rightarrow sr$

Problem

$rrrss$ \Rightarrow rrr
 \Rightarrow srs

Solution

Add rule

▶ $srs \rightarrow rrr$

Computing canonical forms

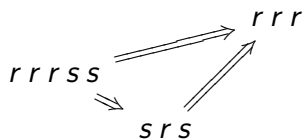
2. Which rule to apply?

▶ $ss \rightarrow \epsilon$

▶ $rrrr \rightarrow \epsilon$

▶ $rrrs \rightarrow sr$

Problem



Solution

Add rule

▶ $srs \rightarrow rrr$

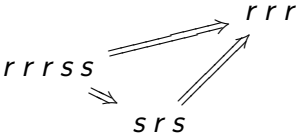
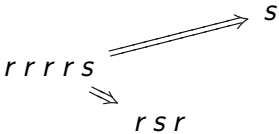
Computing canonical forms

2. Which rule to apply?

▶ $ss \rightarrow \epsilon$

▶ $rrrr \rightarrow \epsilon$

▶ $rrrs \rightarrow sr$

Problem	Solution
 <p>$rrrss$ \Rightarrow rrr $rrrss$ \Rightarrow srs srs \Rightarrow rrr</p>	<p>Add rule</p> <p>▶ $srs \rightarrow rrr$</p>
 <p>$rrrrs$ \Rightarrow s $rrrrs$ \Rightarrow rsr</p>	

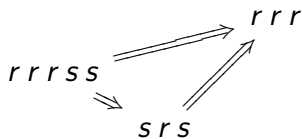
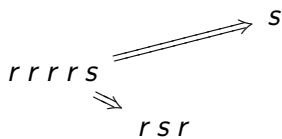
Computing canonical forms

2. Which rule to apply?

▶ $ss \rightarrow \epsilon$

▶ $rrrr \rightarrow \epsilon$

▶ $rrrs \rightarrow sr$

Problem	Solution
 <p>A diagram showing the string $rrrss$ on the left. A double arrow points down and right to srs. From srs, two double arrows point up and right to rrr.</p>	Add rule ▶ $srs \rightarrow rrr$
 <p>A diagram showing the string $rrrrs$ on the left. A double arrow points down and right to rsr. From rsr, a double arrow points up and right to s.</p>	Add rule ▶ $rsr \rightarrow s$

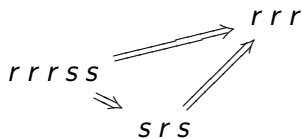

Computing canonical forms

2. Which rule to apply?

▶ $ss \rightarrow \epsilon$

▶ $rrrr \rightarrow \epsilon$

▶ $rrrs \rightarrow sr$

Problem	Solution
	Add rule ▶ $srs \rightarrow rrr$
	Add rule ▶ $rsr \rightarrow s$

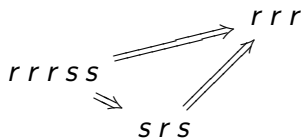

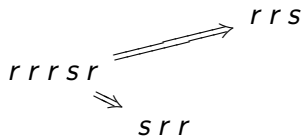
Computing canonical forms

2. Which rule to apply?

▶ $ss \rightarrow \epsilon$

▶ $rrrr \rightarrow \epsilon$

▶ $rrrs \rightarrow sr$

Problem	Solution
	Add rule ▶ $srs \rightarrow rrr$
	Add rule ▶ $rsr \rightarrow s$
	

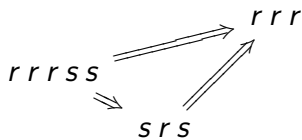

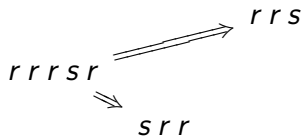
Computing canonical forms

2. Which rule to apply?

▶ $ss \rightarrow \epsilon$

▶ $rrrr \rightarrow \epsilon$

▶ $rrrs \rightarrow sr$

Problem	Solution
 <p>$rrrss$ \Rightarrow rrr $rrrss$ \Rightarrow srs srs \Rightarrow rrr</p>	Add rule ▶ $srs \rightarrow rrr$
 <p>$rrrrs$ \Rightarrow s $rrrrs$ \Rightarrow rsr rsr \Rightarrow s</p>	Add rule ▶ $rsr \rightarrow s$
 <p>$rrrsr$ \Rightarrow rrs $rrrsr$ \Rightarrow srr srr \Rightarrow rrs</p>	Add rule ▶ $srr \rightarrow rrs$

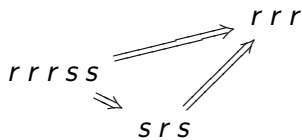

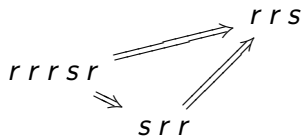
Computing canonical forms

2. Which rule to apply?

▶ $ss \rightarrow \epsilon$

▶ $rrrr \rightarrow \epsilon$

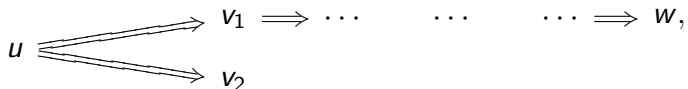
▶ $rrrs \rightarrow sr$

Problem	Solution
	Add rule ▶ $srs \rightarrow rrr$
	Add rule ▶ $rsr \rightarrow s$
	Add rule ▶ $srr \rightarrow rrs$

Computing canonical forms

Confluent rewriting system: “No dead ends.”

If



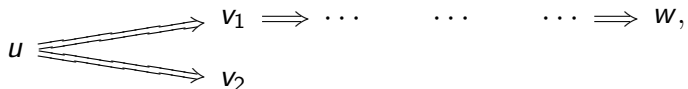
then there exists



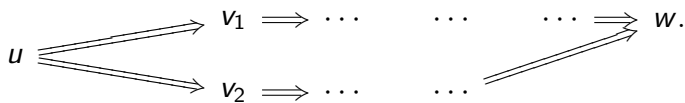
Computing canonical forms

Confluent rewriting system: “No dead ends.”

If



then there exists



Lemma. The following rules are confluent.

- ▶ $ss \rightarrow \varepsilon$
- ▶ $srs \rightarrow rrr$
- ▶ $rsr \rightarrow s$
- ▶ $rrrr \rightarrow \varepsilon$
- ▶ $srr \rightarrow rrs$
- ▶ $rrrs \rightarrow sr$

Computing canonical forms

Three difficulties:

1. Which direction?
2. Which rule to apply?
3. When to stop?

Computing canonical forms

Three difficulties:

1. Which direction? down the ordering
2. Which rule to apply?
3. When to stop?

Lesson 1: $\varepsilon \prec r \prec s \prec rs \prec ss \prec \dots$

Use rewriting rules instead of identities.

Computing canonical forms

Three difficulties:

1. Which direction? down the ordering
2. Which rule to apply?
3. When to stop? don't stop until you have to

Lesson 1: $\varepsilon \prec r \prec s \prec rs \prec ss \prec \dots$

Use rewriting rules instead of identities.

Because of the ordering, eventually you have to stop.

Computing canonical forms

Three difficulties:

1. Which direction? down the ordering
2. Which rule to apply? doesn't matter
3. When to stop? don't stop until you have to

Lesson 1: $\epsilon \prec r \prec s \prec rs \prec ss \prec \dots$

Use rewriting rules instead of identities.

Because of the ordering, eventually you have to stop.

Lesson 2: Use a confluent set of rules, e.g.

$$\blacktriangleright ss \rightarrow \epsilon \qquad \blacktriangleright srs \rightarrow rrr$$

$$\blacktriangleright rsr \rightarrow s \qquad \blacktriangleright rrrr \rightarrow \epsilon$$

$$\blacktriangleright srr \rightarrow rrs \qquad \blacktriangleright rrrs \rightarrow sr$$

Computing canonical forms

“Simplify” any Photoshop product. Solution:

Algorithm

Input: u

Output: canonical form \bar{u}

1. Substitute.
2. Repeat.

Rewriting rules

- ▶ $ss \rightarrow \epsilon$
- ▶ $srs \rightarrow rrr$
- ▶ $rsr \rightarrow s$
- ▶ $rrrr \rightarrow \epsilon$
- ▶ $srr \rightarrow rrs$
- ▶ $rrrs \rightarrow sr$

Computing canonical forms

“Simplify” any Photoshop product. Solution:

Algorithm

Input: u

Output: canonical form \bar{u}

1. Substitute.
2. Repeat.

Rewriting rules

- | | |
|-----------------------------|-------------------------------|
| ▶ $ss \rightarrow \epsilon$ | ▶ $srs \rightarrow rrr$ |
| ▶ $rsr \rightarrow s$ | ▶ $rrrr \rightarrow \epsilon$ |
| ▶ $srr \rightarrow rrs$ | ▶ $rrrs \rightarrow sr$ |

Theorem. The algorithm is correct.

Proof.

1. Always get an answer. (Ordering \Rightarrow termination.)
2. Answer is well defined. (Confluence \Rightarrow no dead ends.)
3. Input \sim output. (Rules come from identities.)
4. Answer is one of the eight canonical forms. (Next slide.)

Computing canonical forms

“Simplify” any Photoshop product. Solution:

Algorithm

Input: u

Output: canonical form \bar{u}

1. Substitute.
2. Repeat.

Rewriting rules

- | | |
|-----------------------------|-------------------------------|
| ▶ $ss \rightarrow \epsilon$ | ▶ $srs \rightarrow rrr$ |
| ▶ $rsr \rightarrow s$ | ▶ $rrrr \rightarrow \epsilon$ |
| ▶ $srr \rightarrow rrs$ | ▶ $rrrs \rightarrow sr$ |

Conclusion of proof:

4. Answer is one of the eight canonical forms.
 - 4.1 Length ≥ 4 : Can be shortened.
 - 4.2 Length ≤ 3 : Check exhaustively.

Plan

Two problems.

1. Compute canonical forms on a computer.

$$r s s r \Rightarrow r r$$

2. Understand infinite sequences in Haskell.

```
> let sevens = 7 : sevens
```

```
> sevens !! 2
```

```
7
```

Similar solutions.

Rewriting in Haskell

```
list !! 0 = head list [1]
```

```
list !! n = (tail list) !! (n-1) [2]
```

```
> let sevens = 7 : sevens [3]
```

```
> sevens !! 2 [4]
```

```
sevens !! 2
```

Rewriting in Haskell

`list !! 0 = head list` [1]

`list !! n = (tail list) !! (n-1)` [2]

`> let sevens = 7 : sevens` [3]

`> sevens !! 2` [4]

`sevens !! 2 = (7 : sevens) !! 2` <3>

Rewriting in Haskell

`list !! 0 = head list` [1]

`list !! n = (tail list) !! (n-1)` [2]

```
> let sevens = 7 : sevens [3]
```

```
> sevens !! 2 [4]
```

`sevens !! 2 = (7 : sevens) !! 2`
`= sevens !! 1` <2>

Rewriting in Haskell

```
list !! 0 = head list [1]
```

```
list !! n = (tail list) !! (n-1) [2]
```

```
> let sevens = 7 : sevens [3]
```

```
> sevens !! 2 [4]
```

```
sevens !! 2 = (7 : sevens) !! 2  
            = sevens !! 1  
            = (7 : sevens) !! 1 <3>
```

Rewriting in Haskell

```
list !! 0 = head list [1]
```

```
list !! n = (tail list) !! (n-1) [2]
```

```
> let sevens = 7 : sevens [3]
```

```
> sevens !! 2 [4]
```

```
sevens !! 2 = (7 : sevens) !! 2  
            = sevens !! 1  
            = (7 : sevens) !! 1  
            = sevens !! 0 <2>
```

Rewriting in Haskell

```
list !! 0 = head list [1]
```

```
list !! n = (tail list) !! (n-1) [2]
```

```
> let sevens = 7 : sevens [3]
```

```
> sevens !! 2 [4]
```

```
sevens !! 2 = (7 : sevens) !! 2  
            = sevens !! 1  
            = (7 : sevens) !! 1  
            = sevens !! 0  
            = (7 : sevens) !! 0 <3>
```

Rewriting in Haskell

```
list !! 0 = head list [1]
```

```
list !! n = (tail list) !! (n-1) [2]
```

```
> let sevens = 7 : sevens [3]
```

```
> sevens !! 2 [4]
```

```
sevens !! 2 = (7 : sevens) !! 2
```

```
            = sevens !! 1
```

```
            = (7 : sevens) !! 1
```

```
            = sevens !! 0
```

```
            = (7 : sevens) !! 0
```

```
            = 7
```

```
<1>
```

Rewriting in Haskell

Haskell uses rewriting rules.

In some ways, similar to the Photoshop example.

In other ways, different.

What rewriting rule describes a for loop?

None! No more for loops!

- ▶ Paulson, L. C. *ML for the Working Programmer*.
- ▶ www.haskell.org
- ▶ <http://www-math.mit.edu/~bsutton>

Haskell: An unusual programming language

```
> let plus = zipWith (+)
```

```
> let fib = 1 : 1 : (fib 'plus' tail fib)
```

```
> take 7 fib
```

```
[1, 1, 2, 3, 5, 8, 13]
```