構成的アルゴリズム論
Left Reductions

胡 振江

東京大学 計数工学科

2008 年 1 月 28 日

A Problem

Min-Max Problem

Given is a list of lists of numbers. Required is an efficient algorithm for computing the minimum of the maximum numbers in each list. More succinctly, we want to compute

$$\mathit{minimax} \;=\; \downarrow/ \cdot \uparrow/*$$

as efficiently as possible.

Outline

1. Well-defineness of General Equations
2. Definition of Left Reduction
3. Implementation of Left Reduction
4. Application: The Minimax Problem

General Equations

So far we have mainly seen examples of homomorphisms. It is instructive to determine the conditions under which a general set of equations

$$\begin{array}{lcl} h\ [\,] & = & e \\ h\ [a] & = & f\ \ a \\ h\ (x +\!\!\!+\, y) & = & H(x, y, h\ x, h\ y) \end{array}$$

determines a unique function h, not necessarily a homomorphism.

Another Representation of h

Consider the equations

$$
\begin{array}{lcl}
h'\ [\,] & = & ([], e) \\
h'\ [a] & = & ([a], f\ \ a) \\
h'\ (x +\!\!+\, y) & = & h'\ x \oplus h'\ y \\
 & & \textbf{where}\ (x, u) \oplus (y, v) = (x +\!\!+\, y, H(x, y, u, v))
\end{array}
$$

If h' is a well-defined function (a well-defined homomorphism), then so is h, because we have

$$h = \pi_2 \cdot h'$$

What is the condition for h' to be well-defined homomorphism?

Well-Defineness of Homomorphic Equations

Fact: h' defined by equations

$$
\begin{array}{lcl}
h'\ [\,] & = & ([], e) \\
h'\ [a] & = & ([a], f\ \ a) \\
h'\ (x \mathbin{+\!\!+} y) & = & h'\ x \oplus h'\ y \\
 & & \textbf{where } (x,u) \oplus (y,v) = (x \mathbin{+\!\!+} y, H(x,y,u,v))
\end{array}
$$

is well-defined if $(R, \oplus, ([], e))$ forms a monoid:

1. $([], e)$ is the unit of $\oplus$;
2. $\oplus$ is associative.

Monoid Condition

Translating the monoid condition into conditions on e and H gives the following three conditions.

1. $H(x, [], u, e) = u$
2. $H([], y, e, v) = v$
3. $H(x \mathbin{+\!\!+} y, z, H(x, y, u, v), w) = H(x, y \mathbin{+\!\!+} z, u, H(y, z, v, w))$

An Example

Longest All-Even Initial Segment Problem

$$laei\ [2,4,8,6,5,2,4] = [2,3,8,6]$$

$$\begin{array}{lcl}
laei\ [\,] & = & [\,] \\
laei\ [a] & = & \textbf{if}\ even\ a\ \textbf{then}\ [a]\ \textbf{else}\ [\,] \\
laei\ (x +\!\!+\, y) & = & \textbf{if}\ laei\ x = x\ \textbf{then}\ laei\ x +\!\!+\ laei\ y\ \textbf{else}\ laei\ x
\end{array}$$

In this example,

$$\begin{array}{lcl}
e & = & [\,] \\
H(x,y,u,v) & = & \textbf{if}\ u = x\ \textbf{then}\ u +\!\!+\, v\ \textbf{else}\ u
\end{array}$$

Exercise: Prove that $\forall x.\ \#(laei\ x) \leq \#x$.
Exercise: Prove that *laei* is well-defined.
[Hint: Use the fact that $\#u \leq \#x$ and $\#v \leq \#y$.]

Lemma. *laei* is not a homomorphism

Proof. Suppose

$$laei(x +\!\!+ y) = laei\ x \oplus laei\ y$$

for some operator $\oplus$. Since $laei\ [2,1] = [2]$, $laei\ [4] = [4]$ and $laei\ [2] = [2]$, we have

$$\begin{aligned} laei\ [2,1,4] &= laei\ [2,1] \oplus laei\ [4] \\ &= [2] \oplus [4] \\ &= laei\ [2] \oplus laei\ [4] \\ &= laei\ [2,4] \end{aligned}$$

This is a contridition, since $laei\ [2,1,4] = [2]$ and $laei\ [2,4] = [2,4]$.

Outline

1 Well-defineness of General Equations

2 Definition of Left Reduction

3 Implementation of Left Reduction

4 Application: The Minimax Problem

Left Reduction

$$\oplus \not\rightarrow_e [x_1, x_2, \ldots, x_n] = (((e \oplus x_1) \oplus x_2) \oplus \cdots) \oplus x_n$$

In the monoid view of lists, the formal definition of $\oplus \not\rightarrow_e$ is as follows.

$$\begin{array}{lcl} \oplus \not\rightarrow_e [] & = & e \\ \oplus \not\rightarrow_e [a] & = & e \oplus a \\ \oplus \not\rightarrow_e (x +\!\!+ y) & = & \oplus \not\rightarrow_{e'} y \ \textbf{where} \ e' = \oplus \not\rightarrow_e x \end{array}$$

$\oplus \not\to_e$: A Well-Defined Function

There is an instructive alternative way of seeing that $\oplus \not\to_e$ is well-defined. Define h by

$$\begin{array}{lcl} h\ [] & = & id \\ h\ [a] & = & (\oplus a) \\ h\ (x +\!\!+ y) & = & h\ y \cdot h\ x \end{array}$$

Obviously, h is a homomorphism from $([a], +\!\!+, [])$ to $(\beta \to \beta, \cdot, id_\beta)$. Now we have

$$\oplus \not\to_e x = h\ x\ e$$

and so $\oplus \not\to_e$ is well-defined.

Left Reduction is Important

Every set of equations of the following form

$$\begin{array}{lcl} f\ [] & = & e \\ f\ (x \mathbin{+\!\!+} [a]) & = & F(a, x, f\ x) \end{array}$$

can be defined in terms of a left reduction:

$$f = \pi_2 \cdot \oplus \not\to_{e'}$$

where

$$\begin{array}{lcl} e' & = & ([], e) \\ (x, u) \oplus a & = & (x \mathbin{+\!\!+} [a], F(a, x, u)) \end{array}$$

Specialization Lemma

Every homomorphism on lists can be expressed as a left (or also a right) reduction. More precisely,

$$
\begin{array}{l}
\oplus/ \cdot f* \;=\; \odot \not\rightarrow_e \\
\quad \text{where} \\
\qquad e \;=\; id_{\oplus} \\
\qquad a \odot b \;=\; a \oplus f\ b
\end{array}
$$

Exercise: Prove the specialization lemma.

Outline

1. Well-defineness of General Equations
2. Definition of Left Reduction
3. Implementation of Left Reduction
4. Application: The Minimax Problem

Left Reductions and Loops

A left reduction $\oplus \not\rightarrow_e x$ can be translated into the following program in a conventional *imperative* language.

```
|[ var r;
   r := e;
   for b in x
      do r := r oplus b;
   return r
]|
```

Left Zeros

Left reductions require that the argument list be traversed in its entirety. Such a traversal can be cut short if we recognize the possibility that an operator may have *left-zeros*.

Definition: Left Zero

ω is a left-zero of $\oplus$ if for all a the following holds.

$$\omega \oplus a = \omega$$

Exercise: Prove that if ω is a left-zero of $\oplus$ then

$$\oplus \not\rightarrow_{\omega} x = \omega$$

for all x. (by induction on snoc list x.)

Implementation of Left Reduction with Left-zero Check

From the fact that $\oplus \not\rightarrow_e (x +\!\!+ y) = \oplus \not\rightarrow_{(\oplus \not\rightarrow_e x)}\ y$, we have the following program for left-reduction.

```
|[ var r;
   r := e;
   for b in x while not left-zero(r)
      do r := r oplus b;
   return r
]|
```

Outline

1. Well-defineness of General Equations
2. Definition of Left Reduction
3. Implementation of Left Reduction
4. Application: The Minimax Problem

Minimax

Let us return to the problem of computing

$$minimax \;=\; \downarrow / \cdot \uparrow / *$$

efficiently. Using the specialization lemma, we can write

$$minimax \;=\; \odot \not\rightarrow_{\infty}$$

where ∞ is the identity element of $\downarrow /$, and

$$a \odot x \;=\; a \downarrow (\uparrow / x).$$

Exercise: Prove that $-\infty$ is the left-zero of $\odot$.

Since $\downarrow$ distributes through $\uparrow$ we have

$$a \odot x \;=\; a \downarrow (\uparrow / x) \;=\; \uparrow /(a \downarrow) * x$$

Using the specialization lemma a second time, we have

$$\begin{aligned} a \odot x \;&=\; \oplus_a \not\rightarrow_{-\infty} x \\ \text{where } b \oplus_a c \;&=\; b \uparrow (a \downarrow c) \end{aligned}$$

Exercise: Prove that a is the left-zero of $\oplus_a$.

An Efficient Implementation of `minimax xs`

```
|[ var a; a := infinity;
   for x in xs while a <> -infinity
      do a := a odot x;
   return a
]|
```

where the assignment `a := a odot x` can be implemented by the loop:

```
|[ var b; b := -infinity;
   for c in x while c <> a
      do b := b max (a min c);
   a := b
]|
```

About the Final Examination

- 時間：2 月 4 日 8:30-10:00
- 場所：6 号館 63 号室
- 教科書、講義資料など持ち込み可
- 成績：全体の 70%

ご協力ありがとうございました。