The least-effort cabinet formation

Johan Jeuring
Lambert Meertens
CWI & RUU

1 Introduction

This note is motivated by the latest Dutch elections, and the formation of a government following on these elections. We derive an algorithm which minimizes the effort needed to form a government (the reader is supposed to be familiar with the Bird–Meertens formalism). This algorithm is an application of a theorem about problems which can be specified by means of

$$\downarrow_{\#}/ \cdot p \triangleleft \cdot \mathsf{segs}.$$

The Dutch House of Representatives consists of 150 seats, occupied by nine parties. The parties can be ordered from left to right according to their political views.

It is generally acknowledged that a government should be supported by at least 80 seats from the House of Representatives. Furthermore, if a number of parties form a government, and there exists a party the political views of which are in between the political views of the parties forming a government, this party joins the parties in the formation of the government (this deviates from the current state of affairs in the Netherlands). Forming a cabinet becomes more difficult when more parties are involved in the negotiations.

The problem we consider is to minimize the effort of forming a cabinet. Suppose we represent the House of Representatives as a list of natural numbers. The length of the list is equal to the number of parties in the House, the parties are ordered according to their political views, and every natural number represents the number of seats a party occupies in the House. It follows from the above facts that a government is supported by a segment of the list, the sum of which is at least 80. Since forming a cabinet becomes more difficult when more parties are involved, we want to find the shortest segment satisfying the above requirement. The specification of our problem reads

$$\downarrow_{\#}/ \cdot \mathsf{sgs} \triangleleft \cdot \mathsf{segs},$$

where the predicate **sgs** (sum greater than seventy–nine) is defined by

$$\mathsf{sgs} = (\geq 80) \cdot +/.$$

2 A promotion theorem

In this section we prove a theorem which gives the conditions under which $\downarrow_\#/ \cdot p \triangleleft \cdot \mathsf{segs}$ equals the composition of a projection function with a left–reduction.

Using the Segment Decomposition Theorem, see [Bir87], we have

$$\begin{array}{rl} & \downarrow_\#/ \cdot p \triangleleft \cdot \mathsf{segs} \\ = & \text{SDT} \\ & \downarrow_\#/ \cdot (\downarrow_\#/ \cdot p \triangleleft \cdot \mathsf{tails})* \cdot \mathsf{inits} \end{array}$$

Note that the part $\downarrow_\#/ \cdot (\downarrow_\#/ \cdot p \triangleleft \cdot \mathsf{tails})*$ from the last formula is a homomorphism. We abbreviate $\downarrow_\#/ \cdot p \triangleleft \cdot \mathsf{tails}$ to sp. The following theorem, which is not proved, gives a solution for problems of the form $h \cdot \mathsf{inits}$, where h is a homomorphism of a specific form. We have

Theorem 1 *(inits–promotion Theorem). Let h be a homomorphism $\oplus/ \cdot f*$ where f is a left–reduction $(\odot \not\rightarrow i)$. Then*

$$(h \cdot \mathsf{inits}, f) = (\otimes \not\rightarrow e),$$

where $e = (i, i)$, and $\otimes$ is defined by

$$(x, y) \otimes a = (x \oplus (y \odot a), y \odot a).$$

In order to apply the inits–promotion Theorem we want to find a left–reduction for $\downarrow_\#/ \cdot p \triangleleft \cdot \mathsf{tails}$. The 'Theory of Lists' developped so far provides lemmas which give a left–reduction for functions of the form $\uparrow_\#/ \cdot p \triangleleft \cdot \mathsf{tails}$ provided p satisfies some conditions. Therefore, we try to express $\downarrow_\#/ \cdot p \triangleleft \cdot \mathsf{tails}$ as $\uparrow_\#/ \cdot q \triangleleft \cdot \mathsf{tails}$ for some predicate q. The following three lemmas contain statements of the form

$$f = g \text{ on } \Delta.$$

By this statement we mean that $f\ d = g\ d$ for all $d \in \Delta$. Using this construct, it is possible to restrict the domain of equality of functions. Let Ξ be the set of lists which satisfy the predicate p, and let Θ be the set of of lists which satisfy $\neg\ p$ (clearly $\alpha* = \Xi \cup \Theta$). We have

Lemma 1 *Let p be a predicate such that $\neg\ p$ is postfix–closed, and define q by $q = \neg\ p \cdot \mathsf{tl}$. Then*

$$\mathsf{sp} = \uparrow_\#/ \cdot q \triangleleft \cdot \mathsf{tails} \text{ on } \Xi.$$

We abbreviate $\uparrow_\#/ \cdot q \triangleleft \cdot \mathsf{tails}$ to lq. This lemma is proved using the following two lemmas, which can be proved using induction.

Lemma 2 *Let $\neg\ p$ be a postfix–closed predicate. Then*

$$p \triangleleft \cdot \mathsf{tails} = [\,]^\bullet \text{ on } \Theta.$$

Lemma 3 *Let $\neg\, p$ be a postfix–closed predicate. Then*

$$\mathsf{lq} = \mathsf{id} \text{ on } \Theta.$$

Proof. We prove Lemma 1 by induction. Since $\neg\, p$ is postfix–closed, $\neg\, p\,[\,]$ holds, and hence $[\,]$ is not an element of Ξ, which proves the base case.

For the induction step, assume

$$p\ x \Rightarrow (\mathsf{sp}\ x = \mathsf{lq}\ x).$$

If $\neg\, p\ x \mathbin{+\!\!+} [a]$ holds, then again $[a] \mathbin{+\!\!+} x$ is not an element from Ξ and we are finished. Suppose $p\ [a] \mathbin{+\!\!+} x$ holds. By a straightforward calculation we obtain

$$\mathsf{sp}\ [a] \mathbin{+\!\!+} x = ([a] \mathbin{+\!\!+} x) \downarrow_{\#} (\mathsf{sp}\ x).$$

We distinguish two cases. First, suppose $\neg\, p\ x$ holds. Then by Lemma 2 we have

$$p \triangleleft\ \mathsf{tails}\ x = [\,],$$

and hence

$$([a] \mathbin{+\!\!+} x) \downarrow_{\#} (\mathsf{sp}\ x) = ([a] \mathbin{+\!\!+} x) \downarrow_{\#} 1_{\downarrow_{\#}} = [a] \mathbin{+\!\!+} x.$$

On the other hand, $q\ [a] \mathbin{+\!\!+} x = \neg\, p\ x$ also holds, and by applying Lemma 3 we find

$$\mathsf{lq}\ [a] \mathbin{+\!\!+} x = [a] \mathbin{+\!\!+} x.$$

Finally, if $p\ x$ holds, then

$$\begin{array}{ll} & ([a] \mathbin{+\!\!+} x) \downarrow_{\#} (\mathsf{sp}\ x) \\ = & \text{definition of } \mathsf{tails}\text{, assumption} \\ & \mathsf{sp}\ x \\ = & \text{induction hypothesis} \\ & \mathsf{lq}\ x \\ = & \text{assumption} \\ & \mathsf{lq}\ [a] \mathbin{+\!\!+} x \end{array}$$

□

The lemmas which give a left–reduction for $\uparrow_{\#}/ \cdot p \triangleleft \cdot \mathsf{tails}$ all require the predicate p to be prefix–closed. Suppose $\neg\, p$ is prefix–closed. Then we have for nonempty x

$$\begin{array}{ll} & q\ x \mathbin{+\!\!+} [a] \\ = & \text{definition of } q \\ & \neg\, p\ \mathsf{tl}\ x \mathbin{+\!\!+} [a] \\ = & \neg\, p \text{ is prefix–closed} \\ & \neg\, p\ \mathsf{tl}\ x \\ = & \text{definition of } q \\ & q\ x \end{array}$$

Hence q is prefix–closed on lists of length at least two. Furthermore, $q\ [a] = \mathsf{True}$ for all elements a. Since $q\ [\,]$ is undefined, q is not prefix–closed. We call a predicate p *almost prefix–closed* if and only if it holds for all singletons and satisfies

$$p\ x +\!\!+ [a] \Rightarrow p\ x,$$

for all elements a and all nonempty lists x. The predicate q is almost prefix–closed according to the discussion above. We have the following variant of the Sliding Tails Lemma (see [BGJ89]).

Lemma 4 *Suppose p is an almost prefix-closed predicate. Then*

$$\uparrow_{\#}/ \cdot p \triangleleft \cdot \mathsf{tails} = (\oplus \not\rightarrow e),$$

where e is some fictitious element ω, and the operator $\oplus$ is defined by

$$\begin{array}{rll} x \oplus a = & x +\!\!+ [a] & \textit{if } p\ x +\!\!+ [a] \\ & (\mathsf{tl}\ x) \oplus a & \textit{if } (\neg\ p\ x +\!\!+ [a]) \wedge x \neq [\,] \\ & [a] & \textit{if } x = \omega \end{array}$$

The equality given in Lemma 1 holds only for elements in Ξ. Hence sp is a left–reduction for elements in Ξ. Suppose z is an element of Θ. Then, by Lemma 3, $\mathsf{lq}\ z = z$. However, $\mathsf{sp}\ z = 1_{\downarrow_{\#}}$, as is verified by an easy calculation. We have the following equivalence

$$(\mathsf{sp}\ z = 1_{\downarrow_{\#}}) \Leftrightarrow (\neg\ p\ \mathsf{lq}\ z).$$

We define

$$\begin{array}{rll} x \downarrow_{\$} y = & x & \text{if } \neg\ p\ y \\ & x \downarrow_{\#} y & \text{otherwise} \end{array}$$

Note that the operator $\downarrow_{\$}$ is associative but not commutative. By the above equivalence, we have

$$\downarrow_{\#}/ \cdot (\downarrow_{\#}/ \cdot p \triangleleft \cdot \mathsf{tails}) * \cdot\ \mathsf{inits} = \downarrow_{\$}/ \cdot (\uparrow_{\#}/ \cdot q \triangleleft \cdot \mathsf{tails}) * \cdot\ \mathsf{inits}.$$

The following theorem is obtained by applying the inits–promotion Theorem and the variant of the Sliding Tails Lemma.

Theorem 2 *Let h be the homomorphism $\downarrow_{\#}/ \cdot p\triangleleft$, where $\neg\ p$ is segment–closed. Let q be the predicate $\neg\ p \cdot \mathsf{tl}$. Then*

$$h \cdot \mathsf{segs} = \pi_1 \cdot (\odot \not\rightarrow i),$$

where $i = (\omega, \omega)$, and the operator $\odot$ is defined by

$$(x, y) \odot a = (x \downarrow_{\$} (y \oplus a), y \oplus a),$$

where the operator $\oplus$ is defined by

$$\begin{array}{rll} x \oplus a = & x +\!\!+ [a] & \textit{if } q\ x +\!\!+ [a] \\ & (\mathsf{tl}\ x) \oplus a & \textit{if } (\neg\ q\ x +\!\!+ [a]) \wedge x \neq [\,] \\ & [a] & \textit{if } x = \omega \end{array}$$

3 The solution

Consider the problem described in the introduction. It is required to find an efficient algorithm for

$$\downarrow_{\#}/ \cdot \mathsf{sgs} \triangleleft \cdot \mathsf{segs}.$$

The predicate **sgs** satisfies

$$\neg\ \mathsf{sgs}\ x = (+/\ x) < 80.$$

Since the list x consists of natural numbers, clearly the predicate $\neg$ **sgs** is segment–closed. We apply Theorem 2, and obtain

$$\downarrow_{\#}/ \cdot \mathsf{sgs} \triangleleft \cdot \mathsf{segs} = \pi_1 \cdot (\odot \not\rightarrow i),$$

where $i = (\omega, \omega)$, and the operator $\odot$ is defined by

$$(x, y) \odot a = (x \downarrow_{\mathsf{s}} (y \oplus a), y \oplus a),$$

where the operator $\oplus$ is defined by

$$x \oplus a = \begin{array}{ll} x +\!\!\!+ [a] & \text{if } +/\ x +\!\!\!+ [a] < 80 \\ (\mathsf{tl}\ x) \oplus a & \text{if } (+/\ x +\!\!\!+ [a] \geq 80) \wedge x \neq [\,] \\ [a] & \text{if } x = \omega \end{array}$$

This is a linear–time algorithm computing the least–effort cabinet formation.

Acknowledgements. Maarten Fokkinga and Hans Zantema made useful comments

References

[BGJ89] R.S. Bird, J. Gibbons, and G. Jones. Formal derivation of a pattern matching algorithm. *Science of Computer Programming*, 12:93–104, 1989.

[Bir87] R.S. Bird. An introduction to the theory of lists. In M. Broy, editor, *Logic of Programming and Calculi of Discrete Design*, pages 5–42, Springer–Verlag, 1987.