# Maximum Marking Problems with Accumulative Weight Functions

Isao Sasano[1], Mizuhito Ogawa[2], and Zhenjiang Hu[3]

[1] RIEC, Tohoku University,
2-1-1 Katahira, Aoba-ku, Sendai 980-8577, Japan
sasano@riec.tohoku.ac.jp
[2] School of Information Science, JAIST,
1-1 Asahidai, Nomi-shi, Ishikawa 923-1292, Japan
mizuhito@jaist.ac.jp
[3] Department of Mathematical Informatics,
School of Information Science and Technology, University of Tokyo,
7-3-1 Hongo, Bunkyo-ku, Tokyo 113-8656, Japan
hu@mist.i.u-tokyo.ac.jp

**Abstract.** We present a new derivation of efficient algorithms for a class of optimization problems called maximum marking problems. We extend the class of weight functions used in the specification to allow for weight functions with accumulation, which is particularly useful when the weight of each element depends on adjacent elements. This extension of weight functions enables us to treat more interesting optimization problems such as a variant of the maximum segment sum problem and the fair bonus distribution problem. The complexity of the derived algorithm is linear with respect to the size of the input data.

**Keywords:** Program derivation, Maximum marking problem, Accumulative weight function, Optimization problem.

## 1 Introduction

One way to guarantee the correctness of programs is to derive programs from specification [PP96, BdM96]. For this approach to be practical, we need high-level theorems that provide solutions for a wide class of problems. Such theorems should also guide the programmer in casting the specification in a form that fulfills the prerequisite conditions of the theorems.

The optimization theorem presented by Sasano et al. [SHTO00, SHT01] is such a theorem, designed for solving the maximum marking problem [Bir01] (also called the maximum weightsum problem). The core of the theorem is generic dynamic programming; it clarifies a class of problems that can be solved by dynamic programming.

The maximum marking problem, MMP for short, is the problem of marking the entries of some given data structure $D$ to maximize a given weight function $w$ under a given constraint $p$. This covers a wide variety of problems

D.V. Hung and M. Wirsing (Eds.): ICTAC 2005, LNCS 3722, pp. 575–591, 2005.

[BLW87, BPT92] (by instantiating $D$, $p$ and $w$), including the well-known maximum segment sum problem [Bir89, Gri90], the maximum independent set problem [SHTO00], some knapsack problems [SHTO01], some optimized range problems in data mining [SHTO02], and the register allocation problem [OHS03].

MMP was first considered in a work on graph algorithms [BLW87], which showed that MMP can be solved in linear time for a certain class of graphs. Borie et al. [BPT92] presented a way to derive a linear time algorithm for MMP from properties described by logical formulae. Their work is elegant in theory; but prohibitive in practice due to the huge constant factor. Our work [SHTO00] facilitated a flexible description of the constraint $p$ by recursive functions and reduced the constant factor drastically. Our subsequent work [SHT01] extended the way the constraint $p$ is described with accumulation. Bird showed a relational derivation for MMP [Bir01], and we demonstrated how to apply the optimization theorem to program analysis [OHS03].

Existing theorems, in a functional [SHTO00], logical [BPT92], or relational [Bir01] setting, can deal with a general data structure $D$ and a powerful constraint $p$. However, they require the weight function $w$ to be in homomorphic form, and hence do not allow for some simple modifications of the weight functions. For instance, consider a variant of the maximum segment sum problem, where the sum is computed by alternately changing the sign. Even this simple example cannot be dealt with by the existing theorems, because the distributivity condition with respect to maximum does not hold, but is required by the theorems.

In this paper, we present two new optimization theorems (calculational rules) for deriving efficient algorithms for a wider class of MMP, by allowing weight functions to be accumulative both in a top-down and bottom-up way. These weight functions are useful when the weight of each element depends on adjacent elements. This extension enables us to treat more interesting optimization problems such as a variant of the maximum segment sum problem (which requires a top-down accumulative weight function) and the fair bonus distribution problem (which requires a bottom-up accumulative weight function). The derived algorithm is linear in the size of the data.

Throughout the paper we will use the notation of Haskell [PJH99], a functional language, to describe our derivation as well as derived programs.

## 2 Preliminaries

In this section we define maximum marking problems on polynomial data types.

We describe polynomial data types in the following form:

$$D\ \alpha = A_1\ (\alpha, D_1, \ldots, D_{n_1})\ |\ A_2\ (\alpha, D_1, \ldots, D_{n_2})\ |\ \cdots\ |\ A_k\ (\alpha, D_1, \ldots, D_{n_k})$$

where every $D_i$ is just $D\,\alpha$, and $A_i$'s are called data constructors, applied to an element of type $\alpha$ and bounded number of recursive components. Though they seem restrictive, these polynomial data types are powerful enough to cover

commonly used data types such as lists, binary trees, and rooted trees [BLW87]. Moreover, other data types like rose trees, a regular data type defined by

$$\mathit{RTree}\ \alpha = \mathit{Node}\ \alpha\ [\mathit{RTree}\ \alpha],$$

can be encoded by a polynomial data type (see Section 4.3). The fold operation *foldD* on $D\ \alpha$ is defined as follows:

$$\begin{aligned}&\mathit{foldD}\ \varphi_1\ \ldots\ \varphi_k = f\\ &\quad \textbf{where}\ f\ (A_i\ (x,\ t_1, \ldots, t_{n_i})) = \varphi_i\ (x,\ f\ t_1, \ldots, f\ t_{n_i}) \quad (i = 1, \ldots, k).\end{aligned}$$

Maximum marking problems are specified on polynomial data types in the following form:

$$\mathit{max}\ w \circ \mathit{filter}\ p \circ \mathit{gen}_D\ M.$$

The function $\mathit{gen}_D$ generates all possible markings by using a finite list of marks $M :: [\mathit{Mark}]$:

$$\mathit{gen}_D :: [\mathit{Mark}] \to D\ \alpha \to [D\ (\alpha, \mathit{Mark})].$$

*Mark* is the type of marks. We treat data types that have a single type parameter $\alpha$, and the elements of type $\alpha$ in the input data are the marking targets. We implement marking as a pair of an element and a mark, so the type of marked elements is $(\alpha, \mathit{Mark})$ and the type of marked data is $D\ (\alpha, \mathit{Mark})$.

The functions *max* and $\mathit{gen}_D$ are defined as follows:

$$\begin{aligned}
&\mathit{max}\ w\ [\,] && = \mathit{error}\ \text{"}\texttt{No solution.}\text{"}\\
&\mathit{max}\ w\ [x] && = x\\
&\mathit{max}\ w\ (x : xs) && = \mathit{bmax}\ w\ x\ (\mathit{max}\ w\ xs)\\
&\mathit{bmax}\ f\ a\ b && = \textbf{if}\ f\ a > f\ b\ \textbf{then}\ a\ \textbf{else}\ b\\
&\mathit{gen}_D\ M && = \mathit{foldD}\ \xi_1 \ldots \xi_k\\
&\xi_i\ (x, ts_1, \ldots, ts_{n_i}) && = [\,A_i\ (x^*, t_1, \ldots, t_{n_i}) \mid x^* \leftarrow [\,(x, m) \mid m \leftarrow M\,],\\
& && \qquad t_1 \leftarrow ts_1, \ldots, t_{n_i} \leftarrow ts_{n_i}]\\
& && \qquad (i = 1, \ldots, k)
\end{aligned}$$

Here we define mutumorphisms on the data type $D\ \alpha$.

**Definition 1 (Mutumorphisms).** *Functions $f_1, f_2, \ldots, f_n$ are mutumorphisms on a recursive data type $D\ \alpha$ if each function $f_i$ is defined mutually by*

$$\begin{aligned}&f_i\ (A_j\ (x,\ t_1, \ldots, t_{n_j})) = \varphi_{ij}\ (x,\ h\ t_1, \ldots, h\ t_{n_j})\\ &\quad \textbf{where}\ h = (f_1 \vartriangle f_2 \vartriangle \ldots \vartriangle f_n) \qquad (j = 1, \ldots, k).\end{aligned}$$

*Note that $f_1 \vartriangle f_2 \vartriangle \ldots \vartriangle f_n$ represents a function defined as follows:*

$$(f_1 \vartriangle f_2 \vartriangle \ldots \vartriangle f_n)\ x = (f_1\ x, f_2\ x, \ldots, f_n\ x).$$

We say that a function $f$ is *finite mutumorphic* [SHTO00] if the function $f$ is defined as mutumorphisms along with other functions, each of which has finite range. A finite mutumorphic function $f$ can be represented as a composition of a projection function $\pi$ whose domain is finite and a folding function:

$$f = \pi \circ \mathit{foldD}\ \varphi_1\ \varphi_2\ \ldots\ \varphi_k.$$

In the following sections, we use the following fusion theorem:

**Theorem 1 (Fusion).** *If the following equation holds for* $i = 1 \ldots k$,

$$f\ (\phi_i\ (x,\ t_1, \ldots, t_{n_i})) = \psi_i\ (x,\ f\ t_1, \ldots, f\ t_{n_i})$$

*then the following equation holds:*

$$f \circ foldD\ \phi_1\ \ldots\ \phi_k = foldD\ \psi_1\ \ldots\ \psi_k.$$

## 3 Top-Down Accumulative Weight Functions

In this section we define top-down accumulative weight functions on polynomial data types and present a new optimization theorem.

### 3.1 The Top-Down Optimization Theorem

**Definition 2 (Top-Down Accumulative Weight Function).** *A function* $w$ *is* top-down accumulative *if it is defined as follows:*

$$\begin{array}{ll} w & :: D\ (\alpha, Mark) \to Weight \\ w\ x & = w'\ x\ e_0 \\ w' & :: D\ (\alpha, Mark) \to Acc \to Weight \\ w'\ (A_i\ (x, t_1, \ldots, t_{n_i}))\ e & = \phi_i\ (x,\ w'\ t_1\ (\delta_{i1}\ x\ e),\ \ldots,\ w'\ t_{n_i}\ (\delta_{in_i}\ x\ e))\ e \end{array}$$

*where the range of* $\delta_{ij}$ *is finite and* $\phi_i$ $(i = 1, \ldots, k)$ *satisfies the following distributivity condition wrt maximum:*

$$\begin{array}{l} maximum\ \{\phi_i\ (x,\ w_1, \ldots, w_{n_i})\ e \mid w_1 \in S_1, \ldots, w_{n_i} \in S_{n_i}\} = \\ \quad \phi_i\ (x,\ maximum\ S_1, \ldots, maximum\ S_{n_i})\ e \end{array}$$

**Theorem 2 (Top-Down Optimization Theorem).** *If property* $p$ *is finite mutumorphic:*

$$p = \pi \circ foldD\ \rho_1 \ldots\ \rho_k$$

*and weight function* $w$ *is top-down accumulative, MMP specified by*

$$max\ w \circ filter\ p \circ gen_D\ M$$

*has an* $O(|Acc|^d \cdot |C|^d \cdot |M| \cdot n)$ *algorithm described by*

$$opttd\ \phi_1\ \ldots\ \phi_k\ \delta_{11}\ \ldots\ \delta_{kn_k}\ (\lambda(c, e).\ (\pi\ c) \wedge (e = e_0))\ \rho_1\ \ldots\ \rho_k\ M\ Acc$$

*where* $C$ *is the domain of* $\pi$, $M$ *is the list of marks,* $d = maximum\ \{n_i \mid 1 \leq i \leq k\}$, *and* $n$ *is the size of the input data. The definition of the function opttd is given in Fig. 1.*

### 3.2 Proof of the Top Down Optimization Theorem

Here we prove Theorem 2 by showing the correctness and complexity of the function *opttd*.

$$
\begin{array}{l}
\mathit{opttd}\ \phi_1 \ldots \phi_k\ \delta_{11} \ldots \delta_{kn_k}\ \mathit{accept}\ \rho_1 \ldots \rho_k\ M\ \mathit{Acc} = \\
\quad \mathit{third} \circ\ \mathit{max\ second} \circ\ \mathit{filter}\ (\mathit{accept} \circ \mathit{first}) \circ\ \mathit{foldD}\ \zeta_1\ \ldots\ \zeta_k \\
\quad \textbf{where}\ \zeta_i\ (x, t_1, \ldots, t_{n_i}) = \\
\qquad \mathit{eachmax}\ [((\rho_i\ (x^*, c_1, \ldots, c_{n_i}),\ e), \\
\qquad\qquad \phi_i\ (x^*, w_1, \ldots, w_{n_i})\ e, \\
\qquad\qquad A_i\ (x^*, r_1, \ldots, r_{n_i}))\ \mid \\
\qquad\qquad\qquad x^* \leftarrow [\,(x, m) \mid m \leftarrow M], \\
\qquad\qquad\qquad ((c_1, e_1), w_1, r_1) \leftarrow t_1, \cdots, ((c_{n_i}, e_{n_i}), w_{n_i}, r_{n_i}) \leftarrow t_{n_i}, \\
\qquad\qquad\qquad e \leftarrow \mathit{Acc},\ \delta_{i1}\ x^*\ e{=}e_1,\ \ldots,\ \delta_{in_i}\ x^*\ e{=}e_{n_i}]\quad (i{=}1, \ldots, k) \\
\mathit{eachmax}\ xs = \mathit{foldl}\ f\ [\,]\ xs \\
\quad \textbf{where}\ f\ [\,]\ (c, w, \mathit{cand}) = [(c, w, \mathit{cand})] \\
\qquad\quad f\ ((c, w, \mathit{cand}) : \mathit{opts})\ (c', w', \mathit{cand}') = \\
\qquad\qquad \textbf{if}\ c == c'\ \textbf{then if}\ w > w'\ \textbf{then}\ (c, w, \mathit{cand}) : \mathit{opts} \\
\qquad\qquad\qquad\qquad\qquad \textbf{else}\ \mathit{opts} + +[(c', w', \mathit{cand}')] \\
\qquad\qquad \textbf{else}\ (c, w, \mathit{cand}) : f\ \mathit{opts}\ (c', w', \mathit{cand}') \\
\mathit{first}\ (x, \_, \_) = x, \qquad \mathit{second}\ (\_, x, \_) = x, \qquad \mathit{third}\ (\_, \_, x) = x
\end{array}
$$

**Fig. 1.** Optimization function *opttd*

**Correctness.** We show the correctness by transforming the specification into *opttd* as in Fig. 2. In the transformation we use the auxiliary functions $\zeta_i^{'}$ $(i = 1, \ldots, k)$ and an underline notation for simple representation:

$$
\begin{array}{l}
\zeta_i^{'}\ (x, t_1, \ldots, t_{n_i}) = \\
\qquad [\,((\rho_i\ (x^*, c_1, \ldots, c_{n_i}),\ e),\ \phi_i\ (x^*, w_1, \ldots, w_{n_i})\ e,\ A_i\ (x^*, r_1, \ldots, r_{n_i})) \\
\qquad \mid x^* \leftarrow [\,(x, m) \mid m \leftarrow M], \\
\qquad\quad ((c_1, e_1), w_1, r_1) \leftarrow t_1, \ldots, ((c_{n_i}, e_{n_i}), w_{n_i}, r_{n_i}) \leftarrow t_{n_i}, \\
\qquad\quad e \leftarrow \mathit{Acc},\ \delta_{i1}\ x^*\ e = e_1,\ \ldots,\ \delta_{in_i}\ x^*\ e = e_{n_i}] \\
\underline{x} = \lambda((\_, y), \_, \_).\ x == y
\end{array}
$$

The first step is simply unfoldings of $p$ and $gen_D$.
The second step is

$$\forall \epsilon.\ \mathit{foldD}\ \xi_1\ \ldots\ \xi_k = \mathit{map\ third} \circ \mathit{filter}\ \underline{\epsilon} \circ \mathit{foldD}\ \zeta_1^{'}\ \ldots\ \zeta_k^{'},$$

which is proved by induction on the structure of input data. In the case of $A_i\ (x, t_1, \ldots, t_{n_i})$,

$$
\begin{array}{ll}
 & \mathit{RHS} \\
= & \{\ \text{unfolding of } \mathit{foldD}\ \} \\
 & \mathit{map\ third}\ (\mathit{filter}\ \underline{\epsilon} \\
 & [\,((\rho_i\ (x^*, c_1, \ldots, c_{n_i}),\ e),\ \phi_i\ (x^*, w_1, \ldots, w_{n_i})\ e,\ A_i\ (x^*, r_1, \ldots, r_{n_i})) \\
 & \mid x^* \leftarrow [\,(x, m) \mid m \leftarrow M],\ ((c_1, e_1), w_1, r_1) \leftarrow \mathit{foldD}\ \zeta_1^{'}\ \ldots\ \zeta_k^{'}\ t_1, \ldots, \\
 & \quad ((c_{n_i}, e_{n_i}), w_{n_i}, r_{n_i}) \leftarrow \mathit{foldD}\ \zeta_1^{'}\ \ldots\ \zeta_k^{'}\ t_{n_i}, \\
 & \quad e \leftarrow \mathit{Acc},\ \delta_{i1}\ x^*\ e = e_1,\ \ldots,\ \delta_{in_i}\ x^*\ e = e_{n_i}])
\end{array}
$$

$$
\begin{array}{ll}
 & \mathit{max}\ w \circ \mathit{filter}\ p \circ \mathit{gen}_D\ M \\
= & \quad \{\ \text{unfold } p \text{ and } \mathit{gen}_D\ \} \\
 & \mathit{max}\ w \circ \mathit{filter}\ (\pi \circ \mathit{foldD}\ \rho_1 \ldots \rho_k) \circ \mathit{foldD}\ \xi_1 \ldots \xi_k \\
= & \quad \{\ \forall \epsilon.\ \mathit{foldD}\ \xi_1 \ldots \xi_k = \mathit{map}\ \mathit{third} \circ \mathit{filter}\ \underline{\epsilon} \circ \mathit{foldD}\ \zeta_1^{'}\ \ldots\ \zeta_k^{'}\ \} \\
 & \mathit{max}\ w \circ \mathit{filter}\ (\pi \circ \mathit{foldD}\ \rho_1 \ldots \rho_k) \circ \mathit{map}\ \mathit{third} \circ \mathit{filter}\ \underline{e_0} \circ \mathit{foldD}\ \zeta_1^{'}\ \ldots\ \zeta_k^{'} \\
= & \quad \{\ \mathit{filter}\ p \circ \mathit{map}\ f = \mathit{map}\ f \circ \mathit{filter}\ (p \circ f)\ \} \\
 & \mathit{max}\ w \circ \mathit{map}\ \mathit{third} \circ \mathit{filter}\ (\pi \circ \mathit{foldD}\ \rho_1 \ldots \rho_k \circ \mathit{third}) \circ \mathit{filter}\ \underline{e_0} \circ \mathit{foldD}\ \zeta_1^{'}\ \ldots\ \zeta_k^{'} \\
= & \quad \{\ \mathit{max}\ w \circ \mathit{map}\ \mathit{third} \circ \mathit{foldD}\ \zeta_1^{'}\ \ldots\ \zeta_k^{'} = \mathit{third} \circ \mathit{max}\ \mathit{second} \circ \mathit{foldD}\ \zeta_1^{'}\ \ldots\ \zeta_k^{'}\ \} \\
 & \mathit{third} \circ \mathit{max}\ \mathit{second} \circ \mathit{filter}\ (\pi \circ \mathit{foldD}\ \rho_1 \ldots \rho_k \circ \mathit{third}) \circ \mathit{filter}\ \underline{e_0} \circ \mathit{foldD}\ \zeta_1^{'}\ \ldots\ \zeta_k^{'} \\
= & \quad \{\ \mathit{filter}\ p \circ \mathit{filter}\ q = \mathit{filter}\ q \circ \mathit{filter}\ p\ \} \\
 & \mathit{third} \circ \mathit{max}\ \mathit{second} \circ \mathit{filter}\ \underline{e_0} \circ \mathit{filter}\ (\pi \circ \mathit{foldD}\ \rho_1 \ldots \rho_k \circ \mathit{third}) \circ \mathit{foldD}\ \zeta_1^{'}\ \ldots\ \zeta_k^{'} \\
= & \quad \{\mathit{map}\ (\mathit{foldD}\ \rho_1 \ldots \rho_k \circ \mathit{third}) \circ \mathit{foldD}\ \zeta_1^{'}\ \ldots\ \zeta_k^{'} = \\
 & \qquad \mathit{map}\ (\mathit{fst} \circ \mathit{first}) \circ \mathit{foldD}\ \zeta_1^{'}\ \ldots\ \zeta_k^{'}\} \\
 & \mathit{third} \circ \mathit{max}\ \mathit{second} \circ \mathit{filter}\ \underline{e_0} \circ \mathit{filter}\ (\pi \circ \mathit{fst} \circ \mathit{first}) \circ \mathit{foldD}\ \zeta_1^{'}\ \ldots\ \zeta_k^{'} \\
= & \quad \{\ \mathit{max}\ \mathit{second} = \mathit{max}\ \mathit{second} \circ \mathit{eachmax}\ \} \\
 & \mathit{third} \circ \mathit{max}\ \mathit{second} \circ \mathit{eachmax} \circ \mathit{filter}\ \underline{e_0} \circ \mathit{filter}\ (\pi \circ \mathit{fst} \circ \mathit{first}) \circ \mathit{foldD}\ \zeta_1^{'}\ \ldots\ \zeta_k^{'} \\
= & \quad \{\ \mathit{filter}\ p \circ \mathit{filter}\ q = \mathit{filter}\ (p \wedge q)\ \} \\
 & \mathit{third} \circ \mathit{max}\ \mathit{second} \circ \mathit{eachmax} \circ \mathit{filter}\ ((\pi \circ \mathit{fst} \circ \mathit{first}) \wedge \underline{e_0}) \circ \mathit{foldD}\ \zeta_1^{'}\ \ldots\ \zeta_k^{'} \\
= & \quad \{\ \mathit{eachmax} \circ \mathit{filter}\ ((\pi \circ \mathit{fst} \circ \mathit{first}) \wedge \underline{e_0}) = \mathit{filter}\ ((\pi \circ \mathit{fst} \circ \mathit{first}) \wedge \underline{e_0}) \circ \mathit{eachmax}\ \} \\
 & \mathit{third} \circ \mathit{max}\ \mathit{second} \circ \mathit{filter}\ (\pi \circ \mathit{fst} \circ \mathit{first} \wedge \underline{e_0}) \circ \mathit{eachmax} \circ \mathit{foldD}\ \zeta_1^{'}\ \ldots\ \zeta_k^{'} \\
= & \quad \{\ \mathit{eachmax} \circ \mathit{foldD}\ \zeta_1^{'}\ \ldots\ \zeta_k^{'} = \mathit{foldD}\ \zeta_1\ \ldots\ \zeta_k\ \} \\
 & \mathit{third} \circ \mathit{max}\ \mathit{second} \circ \mathit{filter}\ ((\pi \circ \mathit{fst} \circ \mathit{first}) \wedge \underline{e_0}) \circ \mathit{foldD}\ \zeta_1\ \ldots\ \zeta_k \\
= & \quad \{\ \text{fold } \mathit{opttd}\ \} \\
 & \mathit{opttd}\ \phi_1\ \ldots\ \phi_k\ \delta_{11}\ \ldots\ \delta_{kn_k}\ (\lambda(c,e).\ (\pi\ c) \wedge (e == e_0))\ \rho_1\ \ldots\ \rho_k\ M\ \mathit{Acc}
\end{array}
$$

**Fig. 2.** A proof of the top-down optimization theorem

$$
\begin{array}{ll}
= & \quad \{\ \text{distributing } \mathit{filter}\ \} \\
 & \quad \mathit{map}\ \mathit{third} \\
 & \quad [\,((\rho_i\ (x^*, c_1, \ldots, c_{n_i}),\ \epsilon),\ \phi_i\ (x^*, w_1, \ldots, w_{n_i})\ \epsilon,\ A_i\ (x^*, r_1, \ldots, r_{n_i})) \\
 & \quad \mid x^* \leftarrow [\,(x, m) \mid m \leftarrow M], \\
 & \qquad ((c_1, e_1), w_1, r_1) \leftarrow \mathit{filter}\ \underline{\delta_{i1}\ x^*\ \epsilon}\ (\mathit{foldD}\ \zeta_1^{'}\ \ldots\ \zeta_k^{'}\ t_1), \ldots, \\
 & \qquad ((c_{n_i}, e_{n_i}), w_{n_i}, r_{n_i}) \leftarrow \mathit{filter}\ \underline{\delta_{in_i}\ x^*\ \epsilon}\ (\mathit{foldD}\ \zeta_1^{'}\ \ldots\ \zeta_k^{'}\ t_{n_i})\,] \\
= & \quad \{\ \text{distributing } \mathit{map}\ \} \\
 & \quad [\ A_i\ (x^*, r_1, \ldots, r_{n_i}) \\
 & \quad \mid x^* \leftarrow [\,(x, m) \mid m \leftarrow M], \\
 & \qquad ((c_1, e_1), w_1, r_1) \leftarrow \mathit{map}\ \mathit{third}\ (\mathit{filter}\ \underline{\delta_{i1}\ x^*\ \epsilon}\ (\mathit{foldD}\ \zeta_1^{'}\ \ldots\ \zeta_k^{'}\ t_1)), \ldots, \\
 & \qquad ((c_{n_i}, e_{n_i}), w_{n_i}, r_{n_i}) \leftarrow \mathit{map}\ \mathit{third}\ (\mathit{filter}\ \underline{\delta_{in_i}\ x^*\ \epsilon}\ (\mathit{foldD}\ \zeta_1^{'}\ \ldots\ \zeta_k^{'}\ t_{n_i}))\,] \\
= & \quad \{\ \text{induction hypothesis}\ \} \\
 & \quad [\ A_i\ (x^*, r_1, \ldots, r_{n_i}) \\
 & \quad \mid x^* \leftarrow [\,(x, m) \mid m \leftarrow M],\ ((c_1, e_1), w_1, r_1) \leftarrow \mathit{foldD}\ \xi_1\ \ldots\ \xi_k\ t_1, \ldots, \\
 & \qquad ((c_{n_i}, e_{n_i}), w_{n_i}, r_{n_i}) \leftarrow \mathit{foldD}\ \xi_1\ \ldots\ \xi_k\ t_{n_i}\,]
\end{array}
$$

$$= \quad \{ \text{ folding of } \mathit{foldD} \ \}$$
$$\mathit{LHS}.$$

The third step is the commutativity of map and filter [Bir87].

The fourth step is

$$\mathit{max}\ w \circ \mathit{map}\ \mathit{third} \circ \mathit{foldD}\ \zeta_1^{'}\ \ldots\ \zeta_k^{'} = \mathit{third} \circ \mathit{max}\ \mathit{second} \circ \mathit{foldD}\ \zeta_1^{'}\ \ldots\ \zeta_k^{'}.$$

This means that the second element is the weight of the third element, which is proved by induction on the structure of the argument of type $D\ \alpha$.

The fifth step is commutativity of filters.

The sixth step is

$$\mathit{map}\ (\mathit{foldD}\ \rho_1 \ldots \rho_k \circ \mathit{third}) \circ \mathit{foldD}\ \zeta_1^{'}\ \ldots\ \zeta_k^{'} = \mathit{map}\ (\mathit{fst} \circ \mathit{first}) \circ \mathit{foldD}\ \zeta_1^{'}\ \ldots\ \zeta_k^{'}.$$

This equation means the first part of the first element is equal to the value of $\mathit{foldD}\ \rho_1\ \ldots\ \rho_k$ applied to the third element, which is proved by induction on the structure of the argument of type $D\ \alpha$.

The seventh step is

$$\mathit{max}\ \mathit{second} = \mathit{max}\ \mathit{second} \circ \mathit{eachmax}.$$

This holds because *max second* returns the rightmost optimal solution, and *eachmax* returns a list which consists of the rightmost optimal solution for each value of the first element, preserving the order.

The eighth step is an equation concerning filter, which can be proved by induction on the structure of the argument list.

The ninth step is

$$\mathit{eachmax} \circ \mathit{filter}\ ((\pi \circ \mathit{fst} \circ \mathit{first}) \wedge \underline{e_0}) = \mathit{filter}\ ((\pi \circ \mathit{fst} \circ \mathit{first}) \wedge \underline{e_0}) \circ \mathit{eachmax}$$

which means the commutativity between the functions *filter* and *eachmax*. This equation holds because the predicate $((\pi \circ \mathit{fst} \circ \mathit{first}) \wedge \underline{e_0})$ is concerned only with the first elements, and the functions *filter* and *eachmax* preserve the order.

The tenth step is

$$\mathit{eachmax} \circ \mathit{foldD}\ \zeta_1^{'}\ \ldots\ \zeta_k^{'} = \mathit{foldD}\ \zeta_1\ \ldots\ \zeta_k$$

which follows from the fusion theorem (Theorem 1). The prerequisite condition for applying the fusion theorem is that the equations below hold for $i = 1, \ldots, k$:

$$\mathit{eachmax}\ (\zeta_i^{'}\ (x, t_1, \ldots, t_{n_i})) = \zeta_i\ (x, \mathit{eachmax}\ t_1, \ldots, \mathit{eachmax}\ t_{n_i}).$$

Since *eachmax* and $\zeta_i^{'}$ preserve the order, the following equations hold for $i = 1, \ldots, k$:

$$\zeta_i\ (x, t_1, \ldots, t_{n_i}) = \zeta_i\ (x, \mathit{eachmax}\ t_1, \ldots, \mathit{eachmax}\ t_{n_i}).$$

```
msas = third . foldr1 (bmax second) . filter (accept . first) . h
accept ((c1,c2,c3),e) = c1 && e == True
h [] = [((rho1,e), phi1 e, [])| e <- [True,False]]
h (x:xs) = eachmax [((rho2 y c, e), phi2 y w e, y:r)
                   | y <- [(x,True),(x,False)],
                     ((c,e'),w,r) <- h xs,
                     e <- [True,False], delta y e == e']
phi1 e = 0
phi2 y w e = if kind y then (if e then weight y else - weight y) + w
             else w
delta y e = if kind y then not e else e
rho1 = (True, True, True)
rho2 y (c1,c2,c3) = if kind y then (c2,c2,False) else (c1,c3,c3)
```

**Fig. 3.** A linear-time Haskell program for the MSAS problem

Since $\zeta_i = \mathit{eachmax} \circ \zeta_i^{'}$, the prerequisite condition holds.
The eleventh step is simply the folding of *opttd*.

**Complexity.** The complexity of the function *opttd*:

$$\mathit{opttd}\ \phi_1\ \ldots\ \phi_k\ \delta_{11}\ \ldots\ \delta_{kn_k}\ (\lambda(c,e).\ (\pi\ c) \wedge (e = e_0))\ \rho_1\ \ldots\ \rho_k\ M\ \mathit{Acc}$$

is $O(|Acc|^{d+1} \cdot |C|^{d+1} \cdot |M| \cdot n)$ where $C$ is the domain of $\pi$, $M$ is the list of marks, $d = \mathit{maximum}\ \{n_i \mid 1 \leq i \leq k\}$, and $n$ is the size of the input data. The complexity follows from that the complexity of the function

$$\zeta_i\ (x, t_1, \ldots, t_{n_i})$$

is $O(|Acc|^d \cdot |C|^d \cdot |M| \cdot n)$ and it is computed $n$ times. The function $\zeta_i$ firstly generates a list that contains at most $|Acc| \cdot |C| \cdot |M|$ elements. Next, the function *eachmax* reduces it to a list that has at most $|Acc| \cdot |C|$ elements. With a list implementation (as in Fig. 1), this reduction takes $O(|Acc|^{d+1} \cdot |C|^{d+1} \cdot |M|)$ time; however, with an array implementation (as in [SHT01]), it is reduced to $O(|Acc|^d \cdot |C|^d \cdot |M|)$ time. For readability, throughout the paper, we describe algorithms by list implementations.

### 3.3 The Maximum Segment Alternate Sum Problem

Consider the following list problem: find a consecutive sublist from the input list such that the selected sublist has the maximum alternate sum, where the alternate sum is computed by alternately changing the sign. For example, given a list $[-3, 5, 2, 7, 6]$, the sublist $[5, 2, 7]$ gives the maximum alternate sum of $5 + (-2) + 7 = 10$ among all the consecutive sublists (segments) in the input list. We call this the *maximum segment alternate sum problem* (MSAS for short).

The property $p$ and the weight function $w$ are written as follows:

$$
\begin{array}{l}
p = \pi_0 \circ \mathit{foldr}\ \rho\ (\mathit{True}, \mathit{True}, \mathit{True}) \\
\quad \textbf{where}\ \rho\ x\ (r_0, r_1, r_2) = \textbf{if}\ \mathit{kind}\ x\ \textbf{then}\ (r_1, r_1, \mathit{False})\ \textbf{else}\ (r_0, r_2, r_2) \\
\qquad\quad \pi_0\ (r_0, r_1, r_2) = r_0 \\
w\ xs = w'\ xs\ \mathit{True} \\
w'\ [\,]\ e = 0 \\
w'\ (x : xs)\ e = \phi_{cons}\ x\ e\ (w'\ xs\ (\delta\ x\ e)) \\
\phi_{cons}\ x\ e\ r = \textbf{if}\ \mathit{kind}\ x\ \textbf{then} \\
\qquad\qquad\quad (\textbf{if}\ e\ \textbf{then}\ \mathit{weight}\ x\ \textbf{else}\ -\mathit{weight}\ x) + r \\
\qquad\quad \textbf{else}\ r \\
\mathit{kind}\ (x, m) = m \\
\mathit{weight}\ (x, m) = x
\end{array}
$$

where $\phi_{cons}$ satisfies the following distributivity condition:

$$\mathit{maximum}\ \{\phi_{cons}\ x\ e\ w \mid w \in S\} = \phi_{cons}\ x\ e\ (\mathit{maximum}\ S).$$

The function *foldr* is a folding function on lists [Bir98]. Applying Theorem 2 immediately yields the linear algorithm in Fig. 3. Note that the weight function written in the following homomorphic form

$$
\begin{array}{l}
w\ [\,] \qquad\quad = 0 \\
w\ (x : xs) = \textbf{if}\ \mathit{kind}\ x\ \textbf{then}\ \mathit{weight}\ x - w\ xs\ \textbf{else}\ w\ xs
\end{array}
$$

does not meet the prerequisite of the theorems in previous work of MMP.

## 4 Bottom-Up Accumulative Weight Functions

In some cases as shown in Section 4.3, we need weight functions that accumulate in bottom-up way.

### 4.1 The Bottom-Up Optimization Theorem

**Definition 3 (Bottom-up Accumulative Weight Function).** *A weight function $w$ on $D$ is* bottom-up accumulative *if $w$ is defined as follows:*

$$
\begin{array}{lll}
w & :: D\ (\alpha, \mathit{Mark}) \to \mathit{Weight} & \\
w\ (A_i\ (x, t_1, \ldots, t_{n_i})) & = \eta_i\ x\ (w\ t_1) \ldots (w\ t_{n_i})\ (q\ t_1) \ldots (q\ t_{n_i}) & \\
 & \qquad\qquad\qquad\qquad\qquad\qquad (i = 1, \ldots, k) & \\
q & :: D\ (\alpha, \mathit{Mark}) \to \mathit{Acc} & \\
q & = \mathit{foldD}\ \sigma_1 \ldots \sigma_k &
\end{array}
$$

*where Acc is a finite set and $\eta_i$ $(i = 1, \ldots, k)$ satisfies the following distributivity condition:*

$$
\begin{array}{l}
\mathit{maximum}\ \{\eta_i\ x\ w_1 \ldots w_{n_i}\ e_1 \ldots e_{n_i} \mid w_1 \in S_1, \ldots, w_{n_i} \in S_{n_i}\} = \\
\quad \eta_i\ x\ (\mathit{maximum}\ S_1) \ldots (\mathit{maximum}\ S_{n_i})\ e_1 \ldots e_{n_i} \quad (j = 1, \ldots, n_i)
\end{array}
$$

**Theorem 3 (Bottom-up Optimization Theorem).** *If property $p$ is finite mutumorphic:*

$$
\begin{array}{l}
\mathit{optbu}\ \eta_1 \ldots \eta_k\ \sigma_1 \ldots \sigma_k\ \mathit{accept}\ \rho_1 \ldots \rho_k\ M = \\
\quad \mathit{third} \circ \mathit{max\ second} \circ \mathit{filter}\ (\mathit{accept} \circ \mathit{first}) \circ \mathit{foldD}\ \psi_1\ \ldots\ \psi_k \\
\quad \textbf{where}\ \psi_i\ (x, t_1, \ldots, t_{n_i}) = \\
\qquad \mathit{eachmax}\ [\,((\rho_i\ (x^*, c_1, \ldots, c_{n_i}),\ \sigma_i\ (x^*, q_1, \ldots, q_{n_i})), \\
\qquad\qquad \eta_i\ (x^*, (w_1, q_1), \ldots, (w_{n_i}, q_{n_i})), \\
\qquad\qquad A_i\ (x^*, r_1, \ldots, r_{n_i}))\ \mid \\
\qquad\qquad\qquad x^* \leftarrow [\,(x, m)\ \mid\ m \leftarrow M], \\
\qquad\qquad\qquad ((c_1, q_1), w_1, r_1) \leftarrow t_1, \cdots, ((c_{n_i}, q_{n_i}), w_{n_i}, r_{n_i}) \leftarrow t_{n_i}] \\
\qquad (i = 1, \ldots, k)
\end{array}
$$

**Fig. 4.** Optimization function *optbu*

$$p = \pi \circ \mathit{foldD}\ \rho_1 \ldots\ \rho_k$$

*and weight function $w$ is bottom-up accumulative, MMP specified by*

$$\mathit{max}\ w \circ \mathit{filter}\ p \circ \mathit{gen}_D\ M$$

*has an $O(|Acc|^d \cdot |C|^d \cdot |M| \cdot n)$ algorithm described by*

$$\mathit{optbu}\ \eta_1\ \ldots\ \eta_k\ \sigma_1\ \ldots\ \sigma_k\ (\lambda(c, q).\ \pi\ c)\ \rho_1\ \ldots\ \rho_k\ M$$

*where Acc is the range of $q$, $C$ is the domain of $\pi$, $M$ is the list of marks, $d$ = maximum $\{n_i \mid 1 \leq i \leq k\}$, and $n$ is the size of the input data. The definition of the optimization function optbu is given in Fig. 4.*

### 4.2 Proof of the Bottom-Up Optimization Theorem

Here we prove Theorem 3 by showing the correctness and complexity of the function *optbu*.

**Correctness.** We show the correctness by transforming the specification into *optbu* as in Fig. 5. In the transformation we use the auxiliary functions $\psi_i'$ $(i = 1, \ldots, k)$ defined by

$$
\begin{array}{rl}
\psi_i'\ (x, t_1, \ldots, t_{n_i}) = [\,((\rho_i\ (x^*, c_1, \ldots, c_{n_i}), \sigma_i\ (x^*, q_1, \ldots, q_{n_i})), & \\
\qquad \eta_i\ (x^*, (w_1, q_1), \ldots, (w_{n_i}, q_{n_i})), & \\
\qquad A_i\ (x^*, r_1, \ldots, r_{n_i})) & \\
\quad \mid x^* \leftarrow [\,(x, m)\ \mid\ m \in M], & \\
\qquad ((c_1, q_1), w_1, r_1) \leftarrow t_1, \cdots, ((c_{n_i}, q_{n_i}), w_{n_i}, r_{n_i}) \leftarrow t_{n_i}]. &
\end{array}
$$

The transformation is simpler than that in the proof of Theorem 2, so we omit the detail.

**Complexity.** Similarly to *opttd*, the complexity is $O(|Acc|^{d+1} \cdot |C|^{d+1} \cdot |M| \cdot n)$, but $O(|Acc|^d \cdot |C|^d \cdot |M| \cdot n)$ is achieved if we use array implementation.

$$
\begin{array}{ll}
 & \mathit{max}\ w \circ \mathit{filter}\ p \circ \mathit{gen}_D\ M \\
= & \quad \{\ \text{unfold } p \text{ and } \mathit{gen}_D\ \} \\
 & \mathit{max}\ w \circ \mathit{filter}\ (\pi \circ \mathit{foldD}\ \rho_1 \ldots \rho_k) \circ \mathit{foldD}\ \xi_1 \ldots \xi_k \\
= & \quad \{\ \mathit{foldD}\ \xi_1 \ldots \xi_k = \mathit{map}\ \mathit{third} \circ \mathit{foldD}\ \psi_1^{'}\ \ldots\ \psi_k^{'}\ \} \\
 & \mathit{max}\ w \circ \mathit{filter}\ (\pi \circ \mathit{foldD}\ \rho_1 \ldots \rho_k) \circ \mathit{map}\ \mathit{third} \circ \mathit{foldD}\ \psi_1^{'}\ \ldots\ \psi_k^{'} \\
= & \quad \{\ \mathit{filter}\ p \circ \mathit{map}\ f = \mathit{map}\ f \circ \mathit{filter}\ (p \circ f)\ \} \\
 & \mathit{max}\ w \circ \mathit{map}\ \mathit{third} \circ \mathit{filter}\ (\pi \circ \mathit{foldD}\ \rho_1 \ldots \rho_k \circ \mathit{third}) \circ \mathit{foldD}\ \psi_1^{'}\ \ldots\ \psi_k^{'} \\
= & \quad \{\ \mathit{max}\ w \circ \mathit{map}\ \mathit{third} \circ \mathit{foldD}\ \psi_1^{'}\ \ldots\ \psi_k^{'} = \mathit{third} \circ \mathit{max}\ \mathit{second} \circ \mathit{foldD}\ \psi_1^{'}\ \ldots\ \psi_k^{'}\ \} \\
 & \mathit{third} \circ \mathit{max}\ \mathit{second} \circ \mathit{filter}\ (\pi \circ \mathit{foldD}\ \rho_1 \ldots \rho_k \circ \mathit{third}) \circ \mathit{foldD}\ \psi_1^{'}\ \ldots\ \psi_k^{'} \\
= & \quad \{\mathit{map}\ (\mathit{foldD}\ \rho_1 \ldots \rho_k \circ \mathit{third}) \circ \mathit{foldD}\ \psi_1^{'}\ \ldots\ \psi_k^{'} = \\
 & \qquad \mathit{map}\ (\mathit{fst} \circ \mathit{first}) \circ \mathit{foldD}\ \psi_1^{'}\ \ldots\ \psi_k^{'}\} \\
 & \mathit{third} \circ \mathit{max}\ \mathit{second} \circ \mathit{filter}\ (\pi \circ \mathit{fst} \circ \mathit{first}) \circ \mathit{foldD}\ \psi_1^{'}\ \ldots\ \psi_k^{'} \\
= & \quad \{\ \mathit{max}\ \mathit{second} = \mathit{max}\ \mathit{second} \circ \mathit{eachmax}\ \} \\
 & \mathit{third} \circ \mathit{max}\ \mathit{second} \circ \mathit{eachmax} \circ \mathit{filter}\ (\pi \circ \mathit{fst} \circ \mathit{first}) \circ \mathit{foldD}\ \psi_1^{'}\ \ldots\ \psi_k^{'} \\
= & \quad \{\ \mathit{eachmax} \circ \mathit{filter}\ (\pi \circ \mathit{fst} \circ \mathit{first}) = \mathit{filter}\ (\pi \circ \mathit{fst} \circ \mathit{first}) \circ \mathit{eachmax}\ \} \\
 & \mathit{third} \circ \mathit{max}\ \mathit{second} \circ \mathit{filter}\ (\pi \circ \mathit{fst} \circ \mathit{first}) \circ \mathit{eachmax} \circ \mathit{foldD}\ \psi_1^{'}\ \ldots\ \psi_k^{'} \\
= & \quad \{\ \mathit{eachmax} \circ \mathit{foldD}\ \psi_1^{'}\ \ldots\ \psi_k^{'} = \mathit{foldD}\ \psi_1\ \ldots\ \psi_k\ \} \\
 & \mathit{third} \circ \mathit{max}\ \mathit{second} \circ \mathit{filter}\ (\pi \circ \mathit{fst} \circ \mathit{first}) \circ \mathit{foldD}\ \psi_1\ \ldots\ \psi_k \\
= & \quad \{\ \text{fold } \mathit{optbu}\ \} \\
 & \mathit{optbu}\ \eta_1 \ldots \eta_k\ \sigma_1 \ldots \sigma_k\ (\pi \circ \mathit{fst})\ \rho_1 \ldots \rho_k\ M
\end{array}
$$

**Fig. 5.** A proof of the bottom-up optimization theorem

### 4.3 The Fair Bonus Distribution Problem

As an example for the bottom-up accumulative optimization theorem, we consider the *fair bonus distribution problem*. There is some profit $T$ to distribute to people in a company. The company has a hierarchical structure; that is, supervisor relationships form a tree rooted at the president. As a natural requirement, the bonus of a supervisor should be more than that of a subordinate. In order to reduce employee complaints, the sum of the difference in bonus between an employee and his/her immediate supervisor should be minimized.

Fig. 6 shows an optimal distribution for $T = 6$. It is not easy to give an optimal distribution, as there are many possibilities. A naive solution is to generate all the distributions, filter out the invalid distributions, and then select the optimal one. Though this naive solution is exponential, we can reduce it to $O(T^4 n)$ by specifying it as MMP and applying our new theorem.

**Specification.** Before we give the specification, we define the trees as follows:

$$\mathit{RTree}\ \alpha = \mathit{Node}\ \alpha\ [\mathit{RTree}\ \alpha].$$

This data type is called a rose tree and is used to represent a general tree, each node of which can have arbitrarily many children.

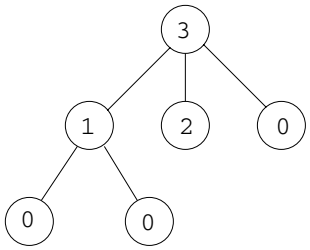


**Fig. 6.** Fair bonus distribution (total = 6)

To specify the problem as MMP we need to define a finite list of marks $M$, property $p$, and weight function $w$.

We use marks to represent the amount of bonus given to each person, so

$$M = [0, 1, \ldots, T].$$

Property $p$ checks whether the sum of the distributed bonuses is $T$ and whether the bonus of each person is more than those of his/her subordinates. Checking the sum can be written as follows:

$$\begin{aligned} &\mathit{csum}\ t &&= (\mathit{bonusSum}\ t = T) \\ &\mathit{bonusSum}\ (\mathit{Node}\ x\ [\,]) &&= \mathit{bonus}\ x \\ &\mathit{bonusSum}\ (\mathit{Node}\ x\ (t : ts)) &&= \mathit{bonusSum}\ t + \mathit{bonusSum}\ (\mathit{Node}\ x\ ts) \end{aligned}$$

Checking to determine whether the bonus of a supervisor is more than those of his/her subordinates can be written as follows:

$$\begin{aligned} &\mathit{more}\ (\mathit{Node}\ x\ [\,]) &&= \mathit{True} \\ &\mathit{more}\ (\mathit{Node}\ x\ (t : ts)) &&= \mathit{bonus}\ x > \mathit{bonus}\ (\mathit{root}\ t)\ \wedge\ \mathit{more}\ t \\ &&&\quad \wedge\ \mathit{more}\ (\mathit{Node}\ x\ ts) \\ &\mathit{root}\ (\mathit{Node}\ x\ ts) &&= x \end{aligned}$$

Using these functions, property $p$ can be defined as follows:

$$p\ t = \mathit{csum}\ t\ \wedge\ \mathit{more}\ t.$$

Weight function $w$ sums up the difference of amount of bonus between an employee and his or her immediate supervisor. In order to minimize the sum of the difference, $w$ returns a negative value.

$$\begin{aligned} &w\ (\mathit{Node}\ x\ [\,]) &&= 0 \\ &w\ (\mathit{Node}\ x\ (t : ts)) &&= \mathit{bonus}\ (\mathit{root}\ t) - \mathit{bonus}\ x + w\ t + w\ (\mathit{Node}\ x\ ts) \end{aligned}$$

Therefore, we can specify the problem as follows:

$$\mathit{fbd} = \mathit{max}\ w \circ \mathit{filter}\ p \circ \mathit{gen}_{RTree}\ M.$$

When total $T$ is not enough, there may be no solution. In such a case the result is "`No solution.`" by the definition of $\mathit{max}\ w$.

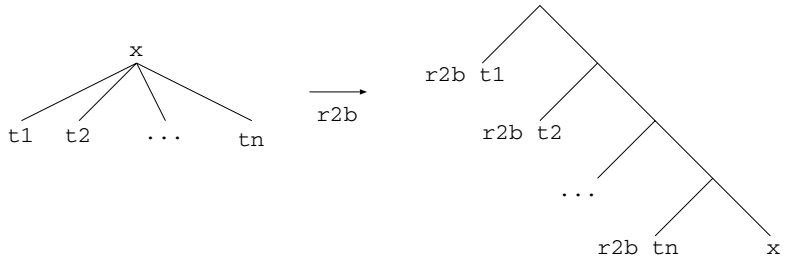


**Fig. 7.** Isomorphism between rose trees and leaf-labeled binary trees

**Derivation.** We have specified the bonus problem as an MMP. Rose tree is not a polynomial data type, so we encode it by leaf-labeled binary tree as in [SHTO00]:

$$\begin{aligned} \mathit{BTree}\ \alpha = {} & \mathit{Tip}\ \alpha \\ \mid {} & \mathit{Bin}\ (\mathit{BTree}\ \alpha)\ (\mathit{BTree}\ \alpha) \end{aligned}$$

Rose trees and leaf-labeled binary trees are isomorphic and we exploit the isomorphism between them, as illustrated in Fig. 7, for converting functions on rose trees into functions on leaf-labeled binary trees. Transformations between the two structures can be implemented in linear time as follows:

$$\begin{aligned} &\mathit{r2b}\ (\mathit{Node}\ x\ [\,]) &&= \mathit{Tip}\ x \\ &\mathit{r2b}\ (\mathit{Node}\ x\ (t : ts)) &&= \mathit{Bin}\ (\mathit{r2b}\ t)\ (\mathit{r2b}\ (\mathit{Node}\ x\ ts)) \\ &\mathit{b2r}\ (\mathit{Tip}\ x) &&= \mathit{Node}\ x\ [\,] \\ &\mathit{b2r}\ (\mathit{Bin}\ t_1\ t_2) &&= \mathbf{let}\ \mathit{Node}\ x\ ts = \mathit{b2r}\ t_2\ \mathbf{in}\ \mathit{Node}\ x\ ((\mathit{b2r}\ t_1) : ts) \end{aligned}$$

Meanwhile, we would like to convert $p :: \mathit{RTree}\ (\alpha, M) \to \mathit{Bool}$ into $p' :: \mathit{BTree}\ (\alpha, M) \to \mathit{Bool}$, which satisfies the following equation:

$$p'\ t = p\ (\mathit{b2r}\ t)$$

By fusion, we get the following:

$$\begin{aligned} &p'\ t &&= \mathit{csum}'\ t\ \wedge\ \mathit{more}'\ t \\ &\mathit{csum}'\ t &&= \mathit{bonusSum}'\ t == T \\ &\mathit{bonusSum}'\ (\mathit{Tip}\ x) &&= \mathit{bonus}\ x \\ &\mathit{bonusSum}'\ (\mathit{Bin}\ t_1\ t_2) &&= \mathit{bonusSum}'\ t_1 + \mathit{bonusSum}'\ t_2 \\ &\mathit{more}'\ (\mathit{Tip}\ x) &&= \mathit{True} \\ &\mathit{more}'\ (\mathit{Bin}\ t_1\ t_2) &&= \mathit{bonus}\ (\mathit{root}'\ t_2) > \mathit{bonus}\ (\mathit{root}'\ t_1) \wedge \\ & && \quad \mathit{more}'\ t_1\ \wedge\ \mathit{more}'\ t_2 \\ &\mathit{root}'\ (\mathit{Tip}\ x) &&= x \\ &\mathit{root}'\ (\mathit{Bin}\ t_1\ t_2) &&= \mathit{root}'\ t_2 \end{aligned}$$

Similarly, we convert $w$ into $w'$, which satisfies $w'\ t = w\ (\mathit{b2r}\ t)$. By fusion, we get the following:

$$\begin{aligned} &w'\ (\mathit{Tip}\ x) &&= 0 \\ &w'\ (\mathit{Bin}\ t_1\ t_2) &&= \mathit{bonus}\ (\mathit{root}'\ t_1) - \mathit{bonus}\ (\mathit{root}'\ t_2) + w'\ t_1 + w'\ t_2 \end{aligned}$$

Using these functions we get the following form:

$$\mathit{fbd} = b2r \circ \mathit{max}\ w' \circ \mathit{filter}\ p' \circ \mathit{gen}_{BTree}\ M \circ r2b.$$

Property $p'$ is defined as mutumorphisms with $\mathit{csum}'$, $\mathit{bonusSum}'$, $\mathit{more}'$, $\mathit{root}'$, but $\mathit{bonusSum}'$ and $\mathit{root}'$ do not have finite range. As for $\mathit{bonusSum}'$, we use the cutting method [SHTO01] by introducing the function $\mathit{cut}$.

$$\begin{aligned}\mathit{csum}'\ t &= \mathit{cut}\ (\mathit{bonusSum}'\ t) == T\\ \mathit{cut}\ s &= \textbf{if}\ s \leq T\ \textbf{then}\ s\ \textbf{else}\ T+1\end{aligned}$$

Let $\mathit{cbs} = \mathit{cut} \circ \mathit{bonusSum}'$, and we get

$$\begin{aligned}\mathit{csum}'\ t &= \mathit{cbs}\ t == T\\ \mathit{cbs}\ (\mathit{Tip}\ x) &= \mathit{cut}\ (\mathit{bonus}\ x)\\ \mathit{cbs}\ (\mathit{Bin}\ t_1\ t_2) &= \mathit{cut}\ (\mathit{cbs}\ t_1 + \mathit{cbs}\ t_2).\end{aligned}$$

As for $\mathit{root}'$, we let $\mathit{br} = \mathit{bonus} \circ \mathit{root}'$. By fusion we get

$$\begin{aligned}\mathit{br}\ (\mathit{Tip}\ x) &= \mathit{bonus}\ x\\ \mathit{br}\ (\mathit{Bin}\ t_1\ t_2) &= \mathit{br}\ t_2.\end{aligned}$$

Using these functions the property $p'$ is described as finite mutumorphisms.

The weight function $w'$ is described using the above function $\mathit{br}$ as follows:

$$\begin{aligned}w'\ (\mathit{Tip}\ x) &= \eta_{\mathit{tip}}\ x\\ w'\ (\mathit{Bin}\ t_1\ t_2) &= \eta_{\mathit{bin}}\ (w'\ t_1)\ (w'\ t_2)\ (\mathit{br}\ t_1)\ (\mathit{br}\ t_2)\\ \eta_{\mathit{tip}}\ x &= 0\\ \eta_{\mathit{bin}}\ w_1\ w_2\ e_1\ e_2 &= e_1 - e_2 + w_1 + w_2\end{aligned}$$

This is bottom-up accumulative, because $\eta_{\mathit{bin}}$ satisfies the monotonicity:

$$w_{11} \leq w_{12} \wedge w_{21} \leq w_{22} \quad \Rightarrow \quad \eta_{\mathit{bin}}\ w_{11}\ w_{21}\ e_1\ e_2 \leq \eta_{\mathit{bin}}\ w_{12}\ w_{22}\ e_1\ e_2$$

and hence satisfies the distributivity condition. By applying Theorem 3, we get an $O(T^6 n)$ algorithm (by an array implementation, it is reduced to $O(T^4 n)$) in Fig. 8, where $n$ is the size of the input tree. When `total = 6`, the expression

```
fbd (Node 'a' [Node 'b' [Node 'c' [], Node 'd' []],
               Node 'e' [], Node 'f' []])
```

computes the following result:

```
Node ('a',3) [Node ('b',1) [Node ('c',0) [],Node ('d',0) []],
              Node ('e',2) [],Node ('f',0) []].
```

```
fbd = b2r . third . foldr1 (bmax second) .
      filter (accept . first) . h . r2b
h (Tip x) = [(((m,True),m),0,Tip (x,m)) | m <- [0..total]]
h (Bin t1 t2) = eachmax [(((cut (c1+c2),q2 > q1 && m1 && m2),q2),
                          q1-q2+w1+w2,Bin r1 r2)
                        | (((c1,m1),q1),w1,r1) <- h t1,
                          (((c2,m2),q2),w2,r2) <- h t2]
accept ((a,b),c) = a == total && b
```

**Fig. 8.** An $O(T^6 n)$ Haskell program for the bonus problem

## 5 Comparison with the Relational Approach

One of the studies that is closely related to our work is derivation by relational calculus [BdM96]. This work showed many optimization problems can be dealt with in a uniform way by relational calculus. Bird showed that MMP can be dealt with by relational calculus [Bir01]. Bird and de Moor gave theorems for deriving efficient greedy, dynamic programming, and thinning algorithms, which cover our optimization theorems. Though they are general, they are not good guides for programmers to write specifications that meet their prerequisite condition. For example, see the thinning theorem [BdM96]:

**Theorem 4.** [BdM96] *If $Q \subseteq R$ and $S$ is monotonic on $Q^\circ$, then*

$$min\ R \circ fold_{\mathsf{F}}(thin\ Q \circ \Lambda(S \circ \mathsf{F}\in)) \quad \subseteq \quad min\ R \circ \Lambda(fold_{\mathsf{F}}\ S).$$

This roughly means that the right side, $min\ R \circ \Lambda(fold_{\mathsf{F}}\ S)$, is the specification, where $S$ is a generating function and $min\ R$ selects the optimal results, and the left side is the derived algorithm. In the example of MMP, $filter\ p \circ gen$ corresponds to $\Lambda(fold_{\mathsf{F}}\ S)$ and $max\ w$ corresponds to $min\ R$. In order to apply this theorem, we have to find $Q$ to satisfy the required conditions with respect to $S$ and $R$; this may be a little burdensome for programmers using the theorem.

Our target is less general, but still includes a useful class of problems called MMP; we provide theorems to automatically derive efficient algorithms with a more friendly interface that guides programmers in writing specifications.

## 6 Conclusions and Future Work

We have presented a new method for deriving efficient algorithms for a class of optimization problems called maximum marking problems. The main contribution of this work is two new powerful optimization theorems, which allow weight functions to be accumulative both in a top-down and bottom-up way. The examples, which cannot be handled by existing approaches, are variants of the maximum segment sum problem and the fair bonus distribution problem. For simplicity, we focused on weight functions and used only finite mutumorphisms

as property descriptions; however, the extension for the property description with accumulators (as in [SHT01]) is straightforward.

Our problem remained, as demonstrated in the fair bonus distribution problem, is that the derived algorithm may have a relatively large constant factor. We expect reduction of the constant factor by using automata compression to eliminate unnecessary states; our current method may produce redundant states due to simple tupling of the property description functions.

Another plan is to apply our new method to more practical real-world problems such as program analysis. Our work [OHS03] solved register allocation without rescheduling as a maximum marking problem. When taking into account the rescheduling of instructions, we will need accumulative information and we expect that the new theorem would play a key role in deriving efficient algorithms for solving these problems.

## References

[BdM96] Richard Bird and Oege de Moor. *Algebra of Programming*. Prentice Hall, 1996.

[Bir87] Richard Bird. An introduction to the theory of lists. In Manfred Broy, editor, *Logic of Programming and Calculi of Discrete Design*, volume F36 of *NATO ASI Series*, pages 5–42. Springer-Verlag, 1987.

[Bir89] Richard Bird. Algebraic identities for program calculation. *The Computer Journal*, 32(2):122–126, 1989.

[Bir98] Richard Bird. *Introduction to Functional Programming using Haskell (second edition)*. Prentice Hall, 1998.

[Bir01] Richard Bird. Maximum marking problems. *Journal of Functional Programming*, 11(4):411–424, 2001.

[BLW87] Marshall W. Bern, Eugene L. Lawler, and A. L. Wong. Linear-time computation of optimal subgraphs of decomposable graphs. *Journal of Algorithms*, 8:216–235, 1987.

[BPT92] Richard B. Borie, R. Gary Parker, and Craig A. Tovey. Automatic generation of linear-time algorithms from predicate calculus descriptions of problems on recursively constructed graph families. *Algorithmica*, 7:555–581, 1992.

[Gri90] D. Gries. The maximum-segment-sum problem. In E. W. Dijkstra, editor, *Formal Development of Programs and Proofs*, pages 33–36. Addison-Wesley, 1990.

[OHS03] Mizuhito Ogawa, Zhenjiang Hu, and Isao Sasano. Iterative-free program analysis. In *Proceedings of the 8th ACM SIGPLAN International Conference on Functional Programming (ICFP'03)*, pages 111–123, Uppsala, Sweden, August 2003. ACM Press.

[PJH99] Simon Peyton Jones and John Hughes, editors. *The Haskell 98 Report*. February 1999. Available from http://www.haskell.org/definition/.

[PP96] Albert Pettrossi and Maurizio Proietti. Rules and strategies for transforming functional and logic programs. *ACM Computing Surveys*, 28(2):360–414, June 1996.

[SHT01] Isao Sasano, Zhenjiang Hu, and Masato Takeichi. Generation of efficient programs for solving maximum multi-marking problems. In Walid Taha, editor, *Semantics, Applications, and Implementation of Program Generation (SAIG'01)*, volume 2196 of *Lecture Notes in Computer Science*, pages 72–91, Firenze, Italy, September 2001. Springer-Verlag.

[SHTO00] Isao Sasano, Zhenjiang Hu, Masato Takeichi, and Mizuhito Ogawa. Make it practical: A generic linear-time algorithm for solving maximum-weightsum problems. In *Proceedings of the 5th ACM SIGPLAN International Conference on Functional Programming (ICFP'00)*, pages 137–149, Montreal, Canada, September 2000. ACM Press.

[SHTO01] Isao Sasano, Zhenjiang Hu, Masato Takeichi, and Mizuhito Ogawa. Solving a class of knapsack problems on recursive data structures (in Japanese). *Computer Software*, 18(2):59–63, 2001.

[SHTO02] Isao Sasano, Zhenjiang Hu, Masato Takeichi, and Mizuhito Ogawa. Derivation of linear algorithm for mining optimized gain association rules. *Computer Software*, 19(4):39–44, 2002.