

修士論文

複製を含む双方向変換と
その応用に関する研究

松田 一孝

指導教員 胡 振江 助教授

平成18年1月27日

東京大学大学院情報理工学系研究科数理情報学専攻

概要

双方向変換は、2つの異なるデータ間の同期を行う技術であり、ソースと呼ばれる元のデータからビューと呼ばれる別のデータを得る順方向の変換と、ビューの変更をソースに書き戻す逆方向の変換から構成される。近年、この双方向変換の枠組みを拡張し、複製を導入することにより、データ内の複製された要素の同期を行えることが示された。それにより、データ内の依存性を表現することができ、様々なアプリケーションの作成に有用であると考えられている。

複製を含まない双方向変換についてはその満たすべき性質に対し十分な議論が行われているにもかかわらず、複製を含む双方向変換については十分な議論が行われていない。たとえば、複製を含まない双方向変換では、変更されたビューと変更が書き戻されたソースのビューが同じであるという性質が変換の組合せについて保存されるが、複製を含む双方向変換については成り立たない。また、双方向変換を利用したアプリケーションもあまりない。

そこで本論文では、複製を含む双方向変換について、その満たすべき性質について議論する。そして、応用例として双方向変換を利用したファイルマネージャ「梅林」の実現を示す。

目次

第1章	はじめに	1
1.1	背景	1
1.2	関連研究	2
1.2.1	ビュー更新の意味論と双方向変換の立場	2
1.2.2	変換記述言語	3
1.3	論文の構成	3
第2章	複製を含まない双方向変換	5
2.1	記法	5
2.2	双方向変換の定義	8
2.3	双方向変換の正当性	10
2.4	基本的な双方向変換	11
2.5	双方向変換の例	20
第3章	複製を含む双方向変換	23
3.1	複製変換	23
3.2	望ましい性質の議論	24
第4章	双方向言語	27
4.1	複数のビューの表現	27
4.1.1	扱うデータ	27
4.1.2	複数のビューを記述する言語の設計と解析	29
4.2	X^\pm 言語と変換システム	34
4.2.1	双方向変換システムの状態	34
4.2.2	操作主導な双方向変換の取り扱い	36
4.2.3	扱うデータ構造	37
4.2.4	X^\pm 言語の定義	40
4.2.5	X^\pm 言語の意味と反射性	41
4.2.6	局所変更保存性に関する議論	52
第5章	双方向変換の応用：ビューを持つ双方向変換を利用したファイルマネージャの実現	53
5.1	「梅林」：ビューを持つ双方向変換を利用したファイルマネージャ	53

5.2	概要	54
5.2.1	機能説明	55
5.2.2	活用例	55
5.3	実装	57
5.3.1	概観	57
5.3.2	データ構造の詳細	58
5.3.3	<i>reflect</i> の実装	60
5.3.4	双方向変換の実装上の問題とその解決法	60
5.4	議論	61
第6章	まとめ	65
6.1	結論	65
6.2	今後の課題	65
6.2.1	双方向変換の理論	65
6.2.2	双方向変換システムとアプリケーション	67
	謝辞	69

第1章 はじめに

1.1 背景

データを抽出，加工し別のデータ形式に変換する．そのような抽出，加工されたデータに対する変更を元のデータに反映することができれば便利である．たとえば，XML データから HTML を生成し，その HTML を WYSIWYG な GUI ツール (WebSphere Studio Homepage Builder[?] など) で編集することにより，元の XML データに反映させることは有用である．

このような元のデータと，元のデータから抽出し加工されたデータとの同期は，双方向変換 [?] を用いることにより実現することができる．双方向変換は，ビュー更新問題 (View Update Problem) [?, ?, ?, ?, ?] に対するアプローチの 1 つであり，元のデータを抽出，加工したビューと呼ばれるデータを得るための順方向変換と，ビューに対する変更をソースに書き戻す逆方向変換の 2 つから構成される．彼らの枠組みでは，双方向変換を様々に組み合わせることで，新たな双方向変換を定義することができる．このような双方向変換の組合せについて，あるいくつかの性質が保存する．たとえば，その性質の 1 つに，変更の行われたビューと，その変更が書き戻された元のデータから得られたビューとが等しいという性質がある．この性質を満たす双方向変換同士をあるやり方で組合せたものも，この性質を満たす．

双方向変換に複製を行う変換を加えることにより，ビュー中の依存性を表現できる [?]．また，複製の導入により，1 つの元となるデータに対し複数のビューを持つことを統一的に定式化できる．このビュー中に依存性を持ち，複数のビューを取り扱えるという双方向変換の性質はアプリケーションの構築において非常に有用であると考えられる．たとえば，HTML により記述される Web サイトの構築を支援するアプリケーションを考えると，リンクによる参照と参照先，個々の Web ページのタイトルとナビゲーションページのリンク文字列の対応関係など，様々な同期関係にある値を取り扱う必要がある．双方向変換はこのような同期関係を構造的に記述するのに有用である．また，音楽ファイルなどに付加された情報から，時代ごとのグルーピングや，演奏者や作曲者ごとのグルーピングなど様々付加情報による整理を行った自由なビューを取り扱うことができ，それらを動的に切り替えたり，同時に扱ったりすることが可能になる．

しかし，アプリケーション構築における，予見される双方向変換の有用性に対し，複製を含む双方向変換の定式化についての議論はあまり進んでいない．たとえば，変更の行われたビューと，その変更が書き戻された元のデータから得られたビューとが等しいという性質は，複製を含む双方向変換においては成り立たない．そのような，複製を含む双方向変換が満たすべき性質および解析についてはあまり行われておらず，また，双方向変換を用いて構築されたアプリケーションもほとんど存在していない．

そこで，本論文では，複製を含む双方向変換の満たすべき性質について議論を行い，また，複

製を含む双方向変換を利用したアプリケーションの実現例として、双方向変換を利用したファイルマネージャの定式化および実現を示し、現状の双方向変換の応用に対する利点および問題点の整理を行う。本論文の貢献は、次のようにまとめることができる。

- 複製を含む双方向変換の満たすべき性質に対し議論を行い、自然な性質を定め、その性質に対し解析を行った。
- 双方向変換をファイルマネージャに応用するにあたって、その問題に特化した定式化を行い、また実現を示した。

1.2 関連研究

1.2.1 ビュー更新の意味論と双方向変換の立場

Bancilhon と Spyratos は、ビュー定義関数 f に対し、得られたビューに対する変更の翻訳として、 x に対し $(f x, g x)$ を割り当てる写像が単射になるような g を考え、その g の戻り値を変更しない (Constant Complement) ようなソースに対する変更に対応させた [?]。これは、すなわちソースに対し f で情報を得ることのできない場所に対する副作用を禁じたものである。また、変更されたビューと翻訳された変更により変更されたソースのビューが同じであるという意味で、ビュー上での副作用もない。彼らの枠組みでは、ビュー上の変更に対応するソースの変更は唯一である。ビュー上の変更をできるだけ許可するために、できるだけ小さい g を構築する必要があるが、関係データベースの場合でも非常にコストがかかる [?, ?]、また、現在においても如何に適切な g を作成するかについての議論がある [?]

Gottlob らは、Consistent View という概念を用いた上で、Constant Complement を緩めたビュー更新の意味論を定めた [?]。彼らの枠組みにおいても、ビュー上の変更に対応するソースの変更は唯一である。そこでは、Constant Complement というソースの上での副作用を禁じるという条件が緩和されている。

Dayal と Bernstein は、さらに緩く、変更されたビューと翻訳された変更により変更されたソースのビューが同じであるという条件を緩めた上で、ビュー更新の意味論を議論した [?]。つまり、彼らの枠組みにおいてビュー上の副作用は許可されている。XML ビューを関係データベース上に実現する場合に彼らの議論が参考にされる場合がある [?]

Foster らは、双方向変換という構成的なアプローチを取った [?]。そこでは、ある「よい性質」を満たす変換を適切に組み合わせるのが「よい性質」を満たす。また、他のビュー更新アプローチと違い、ビュー上の変更をソース上の変更で翻訳するのではなく、ビューの状態とソースの以前の状態を元にソースの新しい状態を求める。双方向変換は、順方向変換であるビュー定義関数と、逆方向変換であるこのビューの状態からソースの新しい状態を求める関数とを同時に記述する。他の [?, ?] などが変更の翻訳を一意に定めるため View Complement などの制約を入れているのに対し、双方向変換では、このような一意的な翻訳は変換の記述により与えられる、そのため、Foster らは制約を満たす変換の合成が制約を満たすことを議論している。彼らはそ

のような制約の1つとして、変更されたビューと、変更が書き戻されたソースのビューは等しくなければならないという制約を扱っている。

Huらは、構造を持つ文書を編集する目的で、文書内の依存性を表現するために双方向変換に複製変換を付け加えたものを議論した[?]。複製変換において、一方に変更が加えられた場合、変更が他方に反映される。この性質を利用して、ビューである文書内の依存性を表現できる。しかし、複製変換の導入は、変更されたビューと変更が書き戻されたソースのビューが等しいという性質を破壊する。この性質に代わる、変換が満たすべき性質については、彼らは十分な議論を行っていない。

1.2.2 変換記述言語

双方向変換においては、ある性質を満たす変換を組み合わせたときの挙動が主に議論されるのに対し、その他のビュー更新問題に対するアプローチやデータ内の同期の表現においては、既存の言語で定義されたビューの性質を議論したり、よい性質を満たすように設計された独自の言語を定めたりする。ここでは、そのような言語について紹介する。

OhoriとTajimaは、オブジェクト指向データベース上で、多相型を含む型システムを持つビュー定義言語を定めた[?]。

Meertensは、ビュー定義関数で得たビューの更新の反映というアプローチではなく、データ中の要素ごと関係を記述し、その関係に従い同期とする言語を定めた[?]。双方向変換などのビュー更新問題に対する多くのアプローチでは、ビューとソースは明確に区別されるが、彼らは言語においては対称に扱われる。

Muらは、単射関数のみを記述できる言語 Inv を定め[?]、その中で双方向変換を議論した[?]。リストが複製された場合に、新しい要素がリストに挿入されたり、リストから要素が削除されたりすると、それぞれの要素の対応付けが難しくなる。彼らは、そのような複製されたリストの要素の対応付けの問題を挿入や削除、修正されたリスト要素にマークを付けをするという方法によって解決した。しかし、彼らの言語で記述された変換の性質に関する議論は充分ではない。たとえば、彼らの言語で定義されたビューでは、ビューに対する変更が書き戻されたソースと、そのソースのビューをまた書き戻したものとが一致しないことがある。

Wadlerは関数型言語において $views$ という概念を提案した[?]。そこでは、データ型を別のデータ型に変換する関数とその逆関数を定義することにより、あるデータ型を別のデータ型のように扱うことができ、パターンマッチなどでそれぞれの構造を使用することができる。たとえば、実部と虚部を与えることで定義されたデータ型を、極座標表現で構成されたものと同じ方法でパターンマッチすることができる。

1.3 論文の構成

本論文の構成を以下に示す。

4 第1章 はじめに

第2章では，Foster らの定式化 [?] に基づき双方向変換の基礎的な説明を行う．第3章では，複製変換 [?] について簡単に説明する．第4章では，簡単な言語を定め，その中で双方向変換の解析を行う．また双方向変換システムについても説明する．第5章では，第4章で述べた双方向変換システムを応用し作成したアプリケーション「梅林」について説明する．第6章で本論文の内容を取りまとめ，結論および今後の課題を示す．

第2章 複製を含まない双方向変換

2.1 記法

まず、本論文を通して使用される記法について述べる。本論文では関数型言語 Haskell[?] に倣った記法を用いる。

関数

関数定義は次のように書く。

$$\text{double } x = x + x$$

これは、引数の値を2倍する関数 *double* の定義である。引数の括弧は省略され、スペースが用いられる。関数適用も同様にスペースを用いて表し、括弧は省略する。また、関数適用が最も結合順位が高い。よって *double 1 + 2* は *double (1 + 2)* ではなく、*(double 1) + 2* を表す。関数合成は $f \circ g$ と書き、

$$(f \circ g) x = f (g x)$$

である。

関数はカーリー化され、関数適用は左結合である。すなわち、 $f a b$ は、 $f (a b)$ でなく $(f a) b$ を表す。例として、2つの値をとりその和を返す、カーリー化された2引数関数 *add* は以下のように定義される。

$$\text{add } x y = x + y$$

ここで、引数の値を1増やした値を返す関数 *inc* は

$$\text{inc} = \text{add } 1$$

のように定義できる。このように *add* は引数を1つとり、「引数を1つとって値を返す関数」を返す関数であると考えられる。

変数 x が型 T であることを $x :: T$ のように書く。値の場合も同様である。また関数 f が A 型の引数を取り B 型の値を返すこととき、 f の型を

$$f :: A \rightarrow B$$

と書く。関数適用が左結合なのに対応し、 \rightarrow は右結合になる。たとえば、整数同士の加算を行う *add* は

$$\text{add} :: \mathbb{Z} \rightarrow \mathbb{Z} \rightarrow \mathbb{Z}$$

という型を持つ。

6 第2章 複製を含まない双方向変換

関数の外延的等価性

関数の等価性を、外延的に以下のように定義する。

定義 2.1 (関数の外延的等価性). ともに $A \rightarrow B$ である関数 f, g が外延的に等価であるとは、

$$\forall x \in A, f x = g x$$

であることをいい、 $f = g$ と書く。 □

たとえば、

$$\text{double}' x = 2 * x$$

は、 double と定義は異なるが、外延的等価性を表す $=$ の下で

$$\text{double}' = \text{double}$$

となる。

全域関数と部分関数

本論文では、定義域の全域で定義された全域関数だけでなく、定義域の一部でしか関数が定義されていない部分関数も扱う。関数 f が値 x について定義されていないとき、未定義を表す記号 \perp を用いて、

$$f x = \perp$$

と書く。また、関係 \sqsubseteq を

$$x \sqsubseteq y \Leftrightarrow (x = \perp \vee x = y)$$

と定義する。関係 $x \sqsubseteq y$ は、 x が定義される場合において y と等しいことを表している。たとえば、自然数 \mathbb{N} 上において引き算 $\text{sub}_{\mathbb{N}} x y$ は、 $x \geq y$ の時のみ定義される。

$$\text{sub}_{\mathbb{N}} x y = \text{if } x \geq y \text{ then } x - y \text{ else } \perp$$

よって、 $\text{add } \perp y = \perp$ であるので、

$$\forall y \in \mathbb{N}, \text{add } (\text{sub}_{\mathbb{N}} x y) y \sqsubseteq x$$

となる。

補題 2.2. 関係 \sqsubseteq は半順序関係 (Partial Order) である。

証明. 反射律, 反対称律, 推移律を満たすことを示す。

反射律 もし, $x = \perp$ なら $x \sqsubseteq x$ であり, $x \neq \perp$ なら $x = x$ より $x \sqsubseteq x$ である. ゆえに反射律

$$x \sqsubseteq x$$

を満たす.

反対称律 もし $x = \perp$ なら $y \sqsubseteq x$ より $y = \perp$ であり, $y = \perp$ なら $x \sqsubseteq y$ より $x = \perp$ である. そうでない場合, \sqsubseteq の定義より $x = y$ がいえる. よって反対称律

$$(x \sqsubseteq y) \wedge (y \sqsubseteq x) \Rightarrow x = y$$

を満たす.

推移律 前提 $(x \sqsubseteq y) \wedge (y \sqsubseteq z)$ を考える. まず, $x = \perp$ のときは明らかに $x \sqsubseteq z$ である. 次に, $x \neq \perp$ のとき $x \sqsubseteq y$ より $x = y$ である. このとき, $y \neq \perp$ かつ $y \sqsubseteq z$ より $y = z$ である. ここから $x = z$ であるので $x \sqsubseteq z$ が成り立つ. よって推移律

$$(x \sqsubseteq y) \wedge (y \sqsubseteq z) \Rightarrow x \sqsubseteq z$$

を満たす. □

また本論文では, 関数の適用が正格であること, すなわち,

$$\forall f, f \perp = \perp$$

を仮定する. また, 正格な評価の下では組 (\dots, \perp, \dots) は \perp と等しいが, 条件分岐に対しては

$$\perp \sqsubseteq \text{if } b \text{ then } s \text{ else } \perp, \quad \perp \sqsubseteq \text{if } b \text{ then } \perp \text{ else } t$$

となる. これは, \perp を通常のプログラム言語における停止しない計算と考えると理解しやすい. つまり, 条件分岐が停止しない計算を含む場合でも, 条件分岐によりその計算が実行されなければ, 条件分岐自体は停止する場合がある.

また, 関数定義は $=$ を用いて定義されるものの, 等価性を表現しているとは限らない. なぜなら, 射影関数

$$\text{first}(a, b) = a$$

について, 正格性より $(a, \perp) = \perp$ となるので

$$\text{first}(a, \perp) = \perp$$

であり,

$$\text{first}(a, b) \sqsubseteq a$$

となる. 一般に正格な関数の定義において左辺 \sqsubseteq 右辺 となる.

また関数 f, g について

$$\forall x, f x \sqsubseteq g x$$

であるとき,

$$f \preceq g$$

と書く.

組

複数の値を (a, b) や (x, y, z) のようにいくつかまとめたものを組と呼ぶ。ある組において、各々の要素の型が同じである必要はない。 $a :: A, b :: B$ の組 (a, b) の型を (A, B) と書く。

リスト

リストは、空リストを表す $[]$ と、要素とリストを取り、要素を先頭に付け加えたリストを作る：という2つのデータ構成子によって構成される均質なデータ構造である。データ構成子は右結合である。簡便のため $1 : 2 : 3 : []$ を $[1, 2, 3]$ と書く。 $[1, (2, 3)]$ などは、1と $(2, 3)$ の型が異なり均質でないので許されない。要素が A 型であるリストの型を $[A]$ と書く。2つのリストは、演算子 $++$ を用いて連結することができる。たとえば、

$$[1, 2, 3] ++ [4, 5] = [1, 2, 3, 4, 5]$$

である。

リストのようにデータ構成子を用いて定義されたデータに対し、

$$\begin{aligned} \text{sum } (x : xs) &= x + \text{sum } xs \\ \text{sum } [] &= 0 \end{aligned}$$

のように、左辺にデータ構成子を使用したパターンマッチを用いることで関数を定義することができる。この関数は、たとえば $[1, 2, 3] = 1 : (2 : (3 : ([])))$ というリストに対し、

$$\begin{aligned} \text{sum } [1, 2, 3] &= \text{sum } (1 : (2 : (3 : []))) \\ &= 1 + \text{sum } (2 : (3 : [])) \\ &= 1 + (2 + \text{sum } (3 : [])) \\ &= 1 + (2 + (3 + \text{sum } [])) \\ &= 1 + (2 + (3 + 0)) \\ &= 6 \end{aligned}$$

のように計算を行う。

2.2 双方向変換の定義

双方向変換は、ビュー更新問題 (View Update Problem) [?, ?, ?, ?, ?] に対するアプローチの1つであり、巨大である場合や目的と異なる形式をしている場合の多いソースと呼ばれる元のデータから、ビューと呼ばれる扱い易く都合の良い形式のデータを得る順方向変換と、ビューの変更をソースへ書き戻す逆方向変換の2つから構成される [?]。これにより、2つの異ったデータ間の同期を行うことができる。

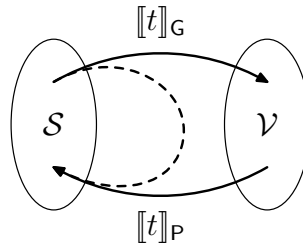


図 2.1. 双方向変換

双方向変換においては、他のビュー更新問題に対するアプローチと異なり、構成的観点から議論されることが多い。すなわち、双方向変換の枠組みにおいて、「よい」双方向変換を組み合わせたものもまた「よい」双方向変換になるかどうか議論される。また、双方向変換は、少数の構成子および基本的な「よい」変換により記述されるため、様々な解析を行いやすい。後の章において、双方向変換を拡張したり、それを利用したアプリケーションを構築したりする際にこの性質が重要になる。

本論文では、双方向変換 f のソースからビューを得る順方向変換としての解釈を $[[f]]_G$ と書き、ビューの変更をソースに書き戻す逆方向変換としての解釈を $[[f]]_P$ と書く。下添字“G”は、順方向変換 $[[f]]_G$ によりビューを得る (get) ことを表し、下添字“P”は、逆方向変換 $[[f]]_P$ によりビューに対する変更を書き戻す (put back) することを表している。順方向変換 $[[f]]_G$ は、ソースを引数に取りビューを返す関数であり、逆方向変換 $[[f]]_P$ は、変更前のソースと変更が加えられたビューを引数に取り、変更が加えられたソースを返す関数である (図 2.1)。これらを定式化すると以下ようになる。

定義 2.3 (双方向変換). 次の 2 つの解釈を持つ f を、ソース S からビュー V への双方向変換という。

$$\begin{aligned} [[f]]_G &:: S \rightarrow V \\ [[f]]_P &:: (S, V) \rightarrow S \end{aligned}$$

但し、 S はソースの集合を表し、 V はビューの集合を表す。 □

また、双方向変換の等価性について以下のように定義する。

定義 2.4 (双方向変換の等価性). 双方向変換 t_1 および t_2 が等価であるとは

$$[[t_1]]_G = [[t_2]]_G \wedge [[t_1]]_P = [[t_2]]_P$$

であることをいい、 $t_1 = t_2$ と書く。また、

$$[[t_1]]_G \preceq [[t_2]]_G \wedge [[t_1]]_P \preceq [[t_2]]_P$$

であるとき、

$$t_1 \preceq t_2$$

と書く。

2.3 双方向変換の正当性

定義 2.3 で定義されたすべての双方向変換が、ビュー更新問題に有用であるわけではない。たとえば、次のような \mathbb{Z} から \mathbb{Z} への双方向変換 f を考えてみよう。

$$\begin{aligned} \llbracket f \rrbracket_G s &= s + 1 \\ \llbracket f \rrbracket_P (s, v) &= v + 1 \end{aligned}$$

この変換は次の 2 つの理由により有用ではない。まず、以下のように、ビューに対し何も変更を行わなくてもソースが変更されてしまう。

$$\llbracket f \rrbracket_P (s, \llbracket f \rrbracket_G s) = \llbracket f \rrbracket_P (s, s + 1) = s + 2 \neq s$$

さらに、以下のように、ビューに変更を行っても、書き戻されたソースのビューは変更されたビューと異なる。

$$\llbracket f \rrbracket_G (\llbracket f \rrbracket_P (s, v)) = \llbracket f \rrbracket_G (v + 1) = v + 2 \neq v$$

このような双方向変換を禁止するため、双方向変換の「よさ」を定式化する必要がある。

これらの性質は以下のように定式化できる [?]。

定義 2.5 (正当な (well-behaved) 双方向変換). 双方向変換 f が、次の 2 つの性質を満たす場合に、 f を正当な双方向変換であるという。

1. 反射性 (Reflexivity)

$$\forall s \in \mathcal{S}, \llbracket f \rrbracket_P (s, \llbracket f \rrbracket_G s) \sqsubseteq s$$

2. 変更保存性 (Update-Preserving)

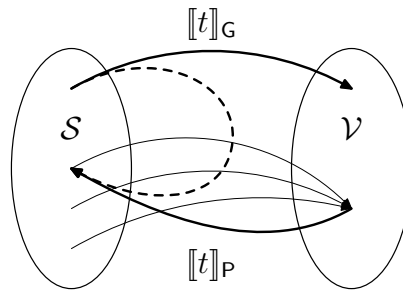
$$\forall s \in \mathcal{S}, \forall v \in \mathcal{V}, \llbracket f \rrbracket_G (\llbracket f \rrbracket_P (s, v)) \sqsubseteq v$$

□

直観的には、反射性はビューが変更されなければソースは変更されないことを意味し、変更保存性はビューが変更されたなら変更されたビューは、変更が書き戻されたソースのビューと同じであることを意味する。

正当な双方向変換 t において、 $\llbracket t \rrbracket_P$ は、 $\llbracket t \rrbracket_G$ の逆関数によく似ている。但し、双方向変換において $\llbracket t \rrbracket_G$ は単射である必要がない。双方向変換では、関数 $\llbracket t \rrbracket_G$ が単射でない場合も、変更前のソース s の情報を利用することにより、関数 $\llbracket t \rrbracket_P (s, v)$ が $\llbracket t \rrbracket_G s' = v$ となるある s' を 1 つ選択する (図 2.2)。例として、以下のように定義される射影変換 fst を考える。

$$\begin{aligned} \llbracket fst \rrbracket_G (s_1, s_2) &= s_1 \\ \llbracket fst \rrbracket_P ((s_1, s_2), v) &= (v, s_2) \end{aligned}$$

図 2.2. $[[t]]_P$ の挙動

明らかにこの $[[fst]]_G$ は単射ではない。しかし fst が正当な双方向変換であることは、

$$[[fst]]_P ((s_1, s_2), [[fst]]_G (s_1, s_2)) = [[fst]]_P ((s_1, s_2), s_1) = (s_1, s_2)$$

および、

$$[[fst]]_G ([[fst]]_P ((s_1, s_2), v)) \subseteq [[fst]]_G (v, s_1) = v.$$

から確認できる。

2.4 基本的な双方向変換

本節では、基本的な双方向変換をいくつか紹介する。

恒等変換

もっとも簡単な双方向変換は恒等変換である。恒等変換 id は順方向において、ソースをそのまま返し、逆方向では変更されたビューをそのまま返す。

$$\begin{aligned} [[id]]_G s &= s \\ [[id]]_P (s, v) &= v \end{aligned}$$

明らかに恒等変換は正当な双方向変換である。

射影変換

前述の fst のような射影変換も、正当な双方向変換である。また同様に以下のように定義される snd も正当な双方向変換になる。

$$\begin{aligned} [[snd]]_G (s_1, s_2) &= s_2 \\ [[snd]]_P ((s_1, s_2), v) &= (s_1, v) \end{aligned}$$

一方向の変換

次の変換

$$\begin{aligned} \llbracket \text{Oneway } f \rrbracket_G s &= f s \\ \llbracket \text{Oneway } f \rrbracket_P (s, v) &= \text{if } f s = v \text{ then } s \text{ else } \perp \end{aligned}$$

は、一方向の変換を表す。すなわち、ビュー上での変更を許さない。

この変換の正当性は、

$$\begin{aligned} \llbracket \text{Oneway } f \rrbracket_P (s, \llbracket \text{Oneway } f \rrbracket_G s) &= \llbracket \text{Oneway } f \rrbracket_P (s, f s) \\ &= s \end{aligned}$$

および、

$$\begin{aligned} \llbracket \text{Oneway } f \rrbracket_G (\llbracket \text{Oneway } f \rrbracket_P (s, v)) &= \llbracket \text{Oneway } f \rrbracket_G (\text{if } f s = v \text{ then } s \text{ else } \perp) \\ &= \{ \text{if の分配則} \} \\ &\quad \text{if } f s = v \text{ then } \llbracket \text{Oneway } f \rrbracket_G s \text{ else } \llbracket \text{Oneway } f \rrbracket_G \perp \\ &= \text{if } f s = v \text{ then } f s \text{ else } f \perp \\ &= \{ \text{正格} \} \\ &\quad \text{if } f s = v \text{ then } f s \text{ else } \perp \\ &\sqsubseteq v \end{aligned}$$

双方向変換の直積

変換 t_1 と変換 t_2 の直積 $t_1 \times t_2$ は、図 2.3 のように変換 t_1 を組の第 1 要素に適用し、変換 t_2 を第 2 要素に適用する。

$$\begin{aligned} \llbracket t_1 \times t_2 \rrbracket_G (s_1, s_2) &= (\llbracket t_1 \rrbracket_G s_1, \llbracket t_2 \rrbracket_G s_2) \\ \llbracket t_1 \times t_2 \rrbracket_P ((s_1, s_2), (v_1, v_2)) &= (\llbracket t_1 \rrbracket_P (s_1, v_1), \llbracket t_2 \rrbracket_P (s_2, v_2)) \end{aligned}$$

双方向変換 $t_1 \times t_2$ は、 t_1 および t_2 が正当な双方向変換であるとき、正当な双方向変換になる。これは、

$$\begin{aligned} \llbracket t_1 \times t_2 \rrbracket_P ((s_1, s_2), \llbracket t_1 \times t_2 \rrbracket_G (s_1, s_2)) &= \llbracket t_1 \times t_2 \rrbracket_P ((s_1, s_2), (\llbracket t_1 \rrbracket_G s_1, \llbracket t_2 \rrbracket_G s_2)) \\ &= (\llbracket t_1 \rrbracket_P (s_1, \llbracket t_1 \rrbracket_G s_1), \llbracket t_2 \rrbracket_P (s_2, \llbracket t_2 \rrbracket_G s_2)) \\ &\sqsubseteq \{ t_1 \text{ と } t_2 \text{ の反射性} \} \\ &\quad (s_1, s_2) \end{aligned}$$

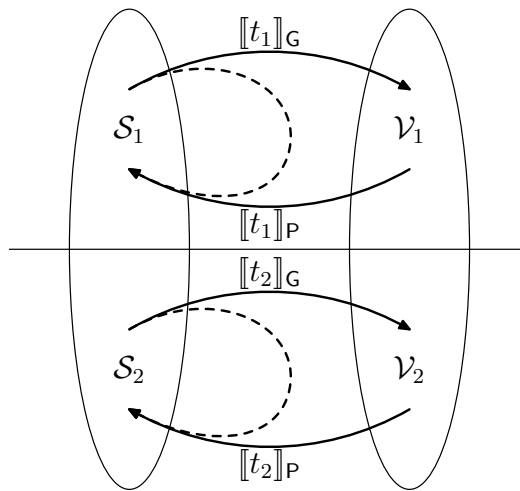


図 2.3. 双方向変換の直積

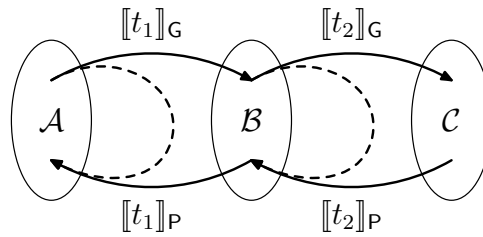


図 2.4. 双方向変換の合成

であり,

$$\begin{aligned}
 [t_1 \times t_2]_G ([t_1 \times t_2]_P ((s_1, s_2), (v_1, v_2))) &= [t_1 \times t_2]_G ([t_1]_P (s_1, v_1), [t_2]_P (s_2, v_2)) \\
 &= ([t_1]_G ([t_1]_P (s_1, v_1)), [t_2]_G ([t_2]_P (s_2, v_2))) \\
 &\sqsubseteq \{ t_1 \text{ と } t_2 \text{ の変更保存性} \} \\
 &\quad (v_1, v_2)
 \end{aligned}$$

であるためである.

双方向変換の合成

双方向変換の合成 $t_1 \hat{\;} t_2$ は, 図 2.4 のように t_1 のビューをソースとみなして t_2 のビューを作成する. これは以下のように定義される.

$$\begin{aligned}
 [t_1 \hat{\;} t_2]_G s &= [t_2]_G ([t_1]_G s) \\
 [t_1 \hat{\;} t_2]_P (s, v) &= [t_1]_P (s, [t_2]_P ([t_1]_G s, v)).
 \end{aligned}$$

双方向変換の合成に対し, 次の性質が成り立つ.

定理 2.6. 双方向変換の反射性および変更保存性は合成に対し閉じている。つまり, t_1 と t_2 が反射的であるとき $t_1 \hat{\;} t_2$ が反射的であり, t_1 と t_2 が変更保存的であるとき $t_1 \hat{\;} t_2$ が変更保存的である。

証明. それぞれの性質に対して証明を行う。

反射性

$$\begin{aligned}
 & [[t_1 \hat{\;} t_2]_{\mathcal{P}} (s, [[t_1 \hat{\;} t_2]_{\mathcal{G}})) \\
 &= [[t_1 \hat{\;} t_2]_{\mathcal{P}} (s, ([[t_2]_{\mathcal{G}} ([[t_1]_{\mathcal{G}} s))]) \\
 &= [[t_1]_{\mathcal{P}} (s, [[t_2]_{\mathcal{P}} ([[t_1]_{\mathcal{G}} s, ([[t_2]_{\mathcal{G}} ([[t_1]_{\mathcal{G}} s))]) \\
 &\sqsubseteq \{ t_2 \text{ の反射性 } \} \\
 &\quad [[t_1]_{\mathcal{P}} (s, [[t_1]_{\mathcal{G}} s) \\
 &\sqsubseteq \{ t_1 \text{ の反射性 } \} \\
 &\quad s
 \end{aligned}$$

よって反射性は合成に対し閉じている。

変更保存性

$$\begin{aligned}
 & [[t_1 \hat{\;} t_2]_{\mathcal{G}} ([[t_1 \hat{\;} t_2]_{\mathcal{P}} (s, v)) \\
 &= [[t_1 \hat{\;} t_2]_{\mathcal{G}} ([[t_1]_{\mathcal{P}} (s, [[t_2]_{\mathcal{P}} ([[t_1]_{\mathcal{G}} s, v))]) \\
 &\quad [[t_2]_{\mathcal{G}} ([[t_1]_{\mathcal{G}} ([[t_1]_{\mathcal{P}} (s, [[t_2]_{\mathcal{P}} ([[t_1]_{\mathcal{G}} s, v))]) \\
 &\sqsubseteq \{ t_1 \text{ の変更保存性 } \} \\
 &\quad [[t_2]_{\mathcal{G}} ([[t_2]_{\mathcal{P}} ([[t_1]_{\mathcal{G}} s, v)) \\
 &\sqsubseteq \{ t_2 \text{ の変更保存性 } \} \\
 &\quad v
 \end{aligned}$$

よって変更保存性は合成に対し閉じている。 □

定理 2.6 より, 任意の正当な双方向変換 t_1 と t_2 の合成 $t_1 \hat{\;} t_2$ が正当な双方向変換になることが示せる。

また, 変換の合成は次のような性質も持つ。

定理 2.7. 変換の合成は結合的である。つまり, 任意の A から B への双方向変換 f , B から C への双方向変換 g , C から D への双方向変換 h について,

$$(f \hat{\;} g) \hat{\;} h = f \hat{\;} (g \hat{\;} h)$$

となる。

証明. まず, 順方向について示す.

$$\begin{aligned} & \llbracket (f \hat{;} g) \hat{;} h \rrbracket_G s \\ &= \llbracket h \rrbracket_G (\llbracket f \hat{;} g \rrbracket_G s) \\ &= \llbracket h \rrbracket_G (\llbracket g \rrbracket_G (\llbracket f \rrbracket_G s)) \end{aligned}$$

および,

$$\begin{aligned} & \llbracket f \hat{;} (g \hat{;} h) \rrbracket_G s \\ &= \llbracket g \hat{;} h \rrbracket_G (\llbracket f \rrbracket_G s) \\ &= \llbracket h \rrbracket_G (\llbracket g \rrbracket_G (\llbracket f \rrbracket_G s)) \end{aligned}$$

より,

$$\llbracket (f \hat{;} g) \hat{;} h \rrbracket_G = \llbracket f \hat{;} (g \hat{;} h) \rrbracket_G$$

となる.

次に, 逆方向について示す.

$$\begin{aligned} & \llbracket (f \hat{;} g) \hat{;} h \rrbracket_P (s, v) \\ &= \llbracket f \hat{;} g \rrbracket_P (s, \llbracket h \rrbracket_P (\llbracket f \hat{;} g \rrbracket_G s, v)) \\ &= \llbracket f \rrbracket_P (s, \llbracket g \rrbracket_P (\llbracket f \rrbracket_G s, \llbracket h \rrbracket_P (\llbracket g \rrbracket_G (\llbracket f \rrbracket_G s), v))) \end{aligned}$$

および,

$$\begin{aligned} & \llbracket f \hat{;} (g \hat{;} h) \rrbracket_P (s, v) \\ &= \llbracket f \rrbracket_P (s, \llbracket g \hat{;} h \rrbracket_P (\llbracket f \rrbracket_G s, v)) \\ &= \llbracket f \rrbracket_P (s, \llbracket g \rrbracket_P (\llbracket f \rrbracket_G s, \llbracket h \rrbracket_P (\llbracket g \rrbracket_G (\llbracket f \rrbracket_G s), v))) \end{aligned}$$

より,

$$\llbracket (f \hat{;} g) \hat{;} h \rrbracket_P = \llbracket f \hat{;} (g \hat{;} h) \rrbracket_P$$

となる.

よって,

$$(f \hat{;} g) \hat{;} h = f \hat{;} (g \hat{;} h)$$

である. □

条件分岐

条件分岐は順方向の変換において, 条件 $pred$ が真の場合に t_1 を利用し変換を行い, 偽の場合に t_2 を利用して変換を行う. 一般の場合の議論は難しいが, 以下のような場合には簡単に正當な双方向変換することができる.

- $\llbracket t_1 \rrbracket_G$ と $\llbracket t_2 \rrbracket_G$ の値域が同じであり, $\llbracket t_1 \rrbracket_P$ の値域で $pred$ が真になり, $\llbracket t_2 \rrbracket_P$ の値域で $pred$ が偽になる場合 .
- $\llbracket t_1 \rrbracket_G$ と $\llbracket t_2 \rrbracket_G$ の値域が排他的であり, $\llbracket t_1 \rrbracket_P$ の値域で $pred$ が真になり, $\llbracket t_2 \rrbracket_P$ の値域で $pred$ が偽になる場合 .

前者の場合の変換を $\text{cond}_s \text{ pred } t_1 t_2$, 後者の場合の変換を $\text{cond}_v \text{ pred}_s \text{ pred}_v t_1 t_2$ と定めることができる .

$$\begin{aligned} \llbracket \text{cond}_s \text{ pred } t_1 t_2 \rrbracket_G s &= \text{if } pred \text{ } s \text{ then } \llbracket t_1 \rrbracket_G s \text{ else } \llbracket t_2 \rrbracket_G s \\ \llbracket \text{cond}_s \text{ pred } t_1 t_2 \rrbracket_P (s, v) &= \text{if } pred \text{ } s \text{ then } \llbracket t_1 \rrbracket_P (s, v) \text{ else } \llbracket t_2 \rrbracket_P (s, v) \\ \llbracket \text{cond}_v \text{ pred}_s \text{ pred}_v t_1 t_2 \rrbracket_G s &= \text{if } pred_s \text{ } s \text{ then } \llbracket t_1 \rrbracket_G s \text{ else } \llbracket t_2 \rrbracket_G s \\ \llbracket \text{cond}_v \text{ pred}_s \text{ pred}_v t_1 t_2 \rrbracket_P (s, v) &= \text{if } pred_v \text{ } v \text{ then } \llbracket t_1 \rrbracket_P (s, v) \text{ else } \llbracket t_2 \rrbracket_P (s, v) \end{aligned}$$

但し, $pred_v$ は $\llbracket t_1 \rrbracket_G$ の値域で真, $\llbracket t_2 \rrbracket_G$ の値域で偽であるとする .

定理 2.8. 双方向変換 t_1 と t_2 が正当な双方向変換であるとき, これらの仮定の上で, $\text{cond}_s \text{ pred } t_1 t_2$ と $\text{cond}_v \text{ pred}_s \text{ pred}_v t_1 t_2$ は正当な双方向変換になる .

証明. まず, $\text{cond}_s \text{ pred}_s t_1 t_2$ について示す .

反射性

$$\begin{aligned} &\llbracket \text{cond}_s \text{ pred}_s t_1 t_2 \rrbracket_P (s, \llbracket \text{cond}_s \text{ pred}_s t_1 t_2 \rrbracket_G s) \\ &= \llbracket \text{cond}_s \text{ pred}_s t_1 t_2 \rrbracket_P (s, \text{if } pred_s \text{ } s \text{ then } \llbracket t_1 \rrbracket_G s \text{ else } \llbracket t_2 \rrbracket_G s) \\ &= \text{if } pred_s \text{ } s \text{ then } \llbracket t_1 \rrbracket_P (s, \llbracket t_1 \rrbracket_G s) \text{ else } \llbracket t_2 \rrbracket_P (s, \llbracket t_2 \rrbracket_G s) \\ &\sqsubseteq \{ t_1 \text{ と } t_2 \text{ の反射性} \} \\ &\quad \text{if } pred_s \text{ } s \text{ then } s \text{ else } s \\ &= s \end{aligned}$$

変更保存性

$$\begin{aligned} &\llbracket \text{cond}_s \text{ pred}_s t_1 t_2 \rrbracket_G (\llbracket \text{cond}_s \text{ pred}_s t_1 t_2 \rrbracket_P (s, v)) \\ &= \llbracket \text{cond}_s \text{ pred}_s t_1 t_2 \rrbracket_G (\text{if } pred_s \text{ } s \text{ then } \llbracket t_1 \rrbracket_P (s, v) \text{ else } \llbracket t_2 \rrbracket_P (s, v)) \\ &= \{ \llbracket t_1 \rrbracket_P \text{ の値域で } pred_s \text{ が真, } \llbracket t_2 \rrbracket_P \text{ の値域で } pred_s \text{ が偽} \} \\ &\quad \text{if } pred_s \text{ } s \text{ then } \llbracket t_1 \rrbracket_G (\llbracket t_1 \rrbracket_P (s, v)) \text{ else } \llbracket t_2 \rrbracket_G (\llbracket t_2 \rrbracket_P (s, v)) \\ &\sqsubseteq \{ t_1 \text{ と } t_2 \text{ の変更保存性} \} \\ &\quad \text{if } pred_s \text{ } s \text{ then } v \text{ else } v \\ &= v \end{aligned}$$

つぎに, $\text{cond}_v \text{ pred}_s \text{ pred}_v t_1 t_2$ について示す .

反射性

$$\begin{aligned}
& \llbracket \text{cond}_v \text{ pred}_s \text{ pred}_v t_1 t_2 \rrbracket_P (s, \llbracket \text{cond}_v \text{ pred}_s \text{ pred}_v t_1 t_2 \rrbracket_G s) \\
&= \llbracket \text{cond}_v \text{ pred}_s \text{ pred}_v t_1 t_2 \rrbracket_P (s, \text{if } \text{pred}_s s \text{ then } \llbracket t_1 \rrbracket_G s \text{ else } \llbracket t_2 \rrbracket_G s) \\
&= \{ \text{pred}_v \text{ は } \llbracket t_1 \rrbracket_G \text{ の値域で真, } \llbracket t_2 \rrbracket_G \text{ の値域で偽} \} \\
&\quad \text{if } \text{pred}_s s \text{ then } \llbracket t_1 \rrbracket_P (s, \llbracket t_1 \rrbracket_G s) \text{ else } \llbracket t_2 \rrbracket_P (s, \llbracket t_2 \rrbracket_G s) \\
&\sqsubseteq \{ t_1 \text{ と } t_2 \text{ の反射性} \} \\
&\quad \text{if } \text{pred}_s s \text{ then } s \text{ else } s \\
&= s
\end{aligned}$$

変更保存性

$$\begin{aligned}
& \llbracket \text{cond}_v \text{ pred}_s \text{ pred}_v t_1 t_2 \rrbracket_G (\llbracket \text{cond}_v \text{ pred}_s \text{ pred}_v t_1 t_2 \rrbracket_P (s, v)) \\
&= \llbracket \text{cond}_v \text{ pred}_s \text{ pred}_v t_1 t_2 \rrbracket_G (\text{if } \text{pred}_v v \text{ then } \llbracket t_1 \rrbracket_P (s, v) \text{ else } \llbracket t_2 \rrbracket_P (s, v)) \\
&= \{ \llbracket t_1 \rrbracket_P \text{ の値域で } \text{pred}_s \text{ が真になり, } \llbracket t_2 \rrbracket_P \text{ の値域で } \text{pred}_s \text{ が偽} \} \\
&\quad \text{if } \text{pred}_v v \text{ then } \llbracket t_1 \rrbracket_G (\llbracket t_1 \rrbracket_P (s, v)) \text{ else } \llbracket t_2 \rrbracket_G (\llbracket t_2 \rrbracket_P (s, v)) \\
&\sqsubseteq \{ t_1 \text{ と } t_2 \text{ の変更保存性} \} \\
&\quad \text{if } \text{pred}_v v \text{ then } v \text{ else } v \\
&= v
\end{aligned}$$

□

再帰

文献 [?] では一般の再帰に対する議論を行っている．ここでは，簡単な例としてリスト上の右畳み込み関数の場合を紹介する．リスト上の右畳み込み関数 *foldr* は

$$\begin{aligned}
\text{foldr } c \ n \ (x : xs) &= c \ x \ (\text{foldr } c \ n \ xs) \\
\text{foldr } c \ n \ [] &= n
\end{aligned}$$

と定義される関数である．ここで関数 *c* の値域かどうかを判定する関数が与えられている場合，以下のような双方向変換 *bifoldr* を定義するとすることができる．

$$\begin{aligned}
\llbracket \text{bifoldr } c \ n \ p \rrbracket_G (s : ss) &= \llbracket c \rrbracket_G (s, (\llbracket \text{bifoldr } c \ n \ p \rrbracket_G ss)) \\
\llbracket \text{bifoldr } c \ n \ p \rrbracket_G [] &= n
\end{aligned}$$

$$\begin{aligned}
& \llbracket \text{bifolldr } c \ n \ p \rrbracket_{\mathcal{P}} ((s : ss), v) = \\
& \quad \text{if } p \ v \ \text{then } s' : \llbracket \text{bifolldr } c \ n \ p \rrbracket_{\mathcal{P}} (ss, v') \\
& \quad \text{else if } v = n \ \text{then } [] \ \text{else } \perp \\
& \quad \quad \text{where } (s', v') = \llbracket c \rrbracket_{\mathcal{P}} ((s, \llbracket \text{bifolldr } c \ n \ p \rrbracket_{\mathcal{G}} ss), v) \\
& \llbracket \text{bifolldr } c \ n \ p \rrbracket_{\mathcal{P}} ((), v) = \\
& \quad \text{if } v = n \ \text{then } [] \\
& \quad \text{else if } p \ v \ \text{then } s' : \llbracket \text{bifolldr } c \ n \ p \rrbracket_{\mathcal{P}} ((), v') \\
& \quad \text{else } \perp \\
& \quad \quad \text{where } (s', v') = \llbracket c \rrbracket_{\mathcal{P}} (\Omega, v)
\end{aligned}$$

ここで, p は $\llbracket c \rrbracket_{\mathcal{G}}$ の値域かどうかを判定する関数であり, Ω は対応するソースがないことを示す記号である. Ω は, Ω かどうか判定できるという意味で \perp ではない.

定理 2.9. $\llbracket \text{bifolldr } c \ n \ p \rrbracket_{\mathcal{P}}$ が停止し, c が正当な双方向変換であるとき, $\text{bifolldr } c \ n \ p$ 正当な双方向変換である. 但し, p は $\llbracket c \rrbracket_{\mathcal{G}}$ の値域かどうか判定する関数であるとする.

証明. それぞれの性質に対し証明する.

反射性 リストの構造に対する帰納法により示す.

まず, $[]$ の場合を示す.

$$\begin{aligned}
& \llbracket \text{bifolldr } c \ n \ p \rrbracket_{\mathcal{P}} ((), \llbracket \text{bifolldr } c \ n \ p \rrbracket_{\mathcal{G}} []) \\
& = \llbracket \text{bifolldr } c \ n \ p \rrbracket_{\mathcal{P}} ((), n) \\
& = []
\end{aligned}$$

つぎに, 入力が ss の場合に成り立つと仮定して, $s : ss$ の場合を示す.

$$\begin{aligned}
& \llbracket \text{bifolldr } c \ n \ p \rrbracket_{\mathcal{P}} ((s : ss), \llbracket \text{bifolldr } c \ n \ p \rrbracket_{\mathcal{G}} (s : ss)) \\
& = \llbracket \text{bifolldr } c \ n \ p \rrbracket_{\mathcal{P}} ((s : ss), \llbracket c \rrbracket_{\mathcal{G}} (s, (\llbracket \text{bifolldr } c \ n \ p \rrbracket_{\mathcal{G}} ss))) \\
& = \{ \text{仮定より, } \llbracket c \rrbracket_{\mathcal{G}} s (\llbracket \text{bifolldr } c \ n \ p \rrbracket_{\mathcal{G}} ss) \text{ に対し } p \text{ は真} \} \\
& \quad s' : \llbracket \text{bifolldr } c \ n \ p \rrbracket_{\mathcal{P}} (ss, v') \\
& \quad \quad \text{where } (s', v') = \llbracket c \rrbracket_{\mathcal{P}} ((s, \llbracket \text{bifolldr } c \ n \ p \rrbracket_{\mathcal{G}} ss), \llbracket c \rrbracket_{\mathcal{G}} (s, (\llbracket \text{bifolldr } c \ n \ p \rrbracket_{\mathcal{G}} ss))) \\
& \sqsubseteq \{ c \text{ の反射性} \} \\
& \quad s' : \llbracket \text{bifolldr } c \ n \ p \rrbracket_{\mathcal{P}} (ss, v') \\
& \quad \quad \text{where } (s', v') = (s, \llbracket \text{bifolldr } c \ n \ p \rrbracket_{\mathcal{G}} ss) \\
& = s : \llbracket \text{bifolldr } c \ n \ p \rrbracket_{\mathcal{P}} (ss, \llbracket \text{bifolldr } c \ n \ p \rrbracket_{\mathcal{G}} ss) \\
& \sqsubseteq \{ \text{帰納法の仮定} \} \\
& \quad s : ss
\end{aligned}$$

変更保存性 まず，ソースが $[]$ の場合を考える． $\llbracket \text{bifldr } c \ n \ p \rrbracket_{\mathcal{P}}$ が停止すると仮定しているの
で， M 回で再帰が止まると仮定し， m 回目の再帰に使われるビューを v_m と置く．

明らかに， $v_M = n$ であり，

$$\begin{aligned} & \llbracket \text{bifldr } c \ n \ p \rrbracket_{\mathcal{G}} (\llbracket \text{bifldr } c \ n \ p \rrbracket_{\mathcal{P}} ([] , v_M)) \\ &= \llbracket \text{bifldr } c \ n \ p \rrbracket_{\mathcal{G}} (\llbracket \text{bifldr } c \ n \ p \rrbracket_{\mathcal{P}} ([] , n)) \\ &= \llbracket \text{bifldr } c \ n \ p \rrbracket_{\mathcal{G}} [] \\ &= n \\ &= v_M \end{aligned}$$

である．

ビューが v_{m+1} のとき，

$$\llbracket \text{bifldr } c \ n \ p \rrbracket_{\mathcal{G}} (\llbracket \text{bifldr } c \ n \ p \rrbracket_{\mathcal{P}} ([] , v_{m+1})) = v_{m+1}$$

が成り立つと仮定する． v_m はそこで再帰が停止しないので， n ではない．

$$\begin{aligned} & \llbracket \text{bifldr } c \ n \ p \rrbracket_{\mathcal{G}} (\llbracket \text{bifldr } c \ n \ p \rrbracket_{\mathcal{P}} ([] , v_m)) \\ &= \{ v_m \neq n \} \\ & \quad \llbracket \text{bifldr } c \ n \ p \rrbracket_{\mathcal{G}} (\text{if } p \ v_m \text{ then } s' : \llbracket \text{bifldr } c \ n \ p \rrbracket_{\mathcal{P}} ([] , v_{m+1}) \text{ else } \perp) \\ & \quad \text{where } (s', v_{m+1}) = \llbracket c \rrbracket_{\mathcal{P}} (\Omega, v_m) \\ & \sqsubseteq \{ \text{正格} \} \\ & \quad \llbracket \text{bifldr } c \ n \ p \rrbracket_{\mathcal{G}} (s' : \llbracket \text{bifldr } c \ n \ p \rrbracket_{\mathcal{P}} ([] , v_{m+1})) \\ & \quad \text{where } (s', v_{m+1}) = \llbracket c \rrbracket_{\mathcal{P}} (\Omega, v_m) \\ &= \llbracket c \rrbracket_{\mathcal{G}} (s', \llbracket \text{bifldr } c \ n \ p \rrbracket_{\mathcal{G}} (\llbracket \text{bifldr } c \ n \ p \rrbracket_{\mathcal{P}} ([] , v_{m+1}))) \\ & \quad \text{where } (s', v_{m+1}) = \llbracket c \rrbracket_{\mathcal{P}} (\Omega, v_m) \\ & \sqsubseteq \{ \text{帰納法の仮定} \} \\ & \quad \llbracket c \rrbracket_{\mathcal{G}} (\llbracket c \rrbracket_{\mathcal{P}} (\Omega, v_m)) \\ & \sqsubseteq \{ c \text{ の変更保存性} \} \\ & \quad v_m \end{aligned}$$

これでソースが $[]$ のときに，

$$\llbracket \text{bifldr } c \ n \ p \rrbracket_{\mathcal{G}} (\llbracket \text{bifldr } c \ n \ p \rrbracket_{\mathcal{P}} ([] , v)) = v$$

であることを示した．次に，ソースが ss のとき成り立つことを仮定し， $s : ss$ のときも成り立

つことを示す .

$$\begin{aligned}
& \llbracket \text{bifldr } c \ n \ p \rrbracket_G (\llbracket \text{bifldr } c \ n \ p \rrbracket_P (s : ss, v)) \\
&= \llbracket \text{bifldr } c \ n \ p \rrbracket_G \left(\begin{array}{l} \text{if } p \ v \ \text{then } s' : \llbracket \text{bifldr } c \ n \ p \rrbracket_P (ss, v') \\ \text{else if } v = n \ \text{then } [] \\ \text{else } \perp \end{array} \right) \\
&\quad \text{where } (s', v') = \llbracket c \rrbracket_P ((s, \llbracket \text{bifldr } c \ n \ p \rrbracket_G ss), v) \\
&= \{ \text{if の分配則} \} \\
&\quad \text{if } p \ v \ \text{then } \llbracket c \rrbracket_G (s', \llbracket \text{bifldr } c \ n \ p \rrbracket_G (\llbracket \text{bifldr } c \ n \ p \rrbracket_P (ss, v'))) \\
&\quad \text{else if } v = n \ \text{then } n \\
&\quad \text{else } \perp \\
&\quad \text{where } (s', v') = \llbracket c \rrbracket_P ((s, \llbracket \text{bifldr } c \ n \ p \rrbracket_G ss), v) \\
&\sqsubseteq \{ \text{帰納法の仮定} \} \\
&\quad \text{if } p \ v \ \text{then } \llbracket c \rrbracket_G (s', v') \\
&\quad \text{else if } v = n \ \text{then } n \\
&\quad \text{else } \perp \\
&\quad \text{where } (s', v') = \llbracket c \rrbracket_P ((s, \llbracket \text{bifldr } c \ n \ p \rrbracket_G ss), v) \\
&\sqsubseteq \{ c \text{ の変更保存性} \} \\
&\quad \text{if } p \ v \ \text{then } v \\
&\quad \text{else if } v = n \ \text{then } n \\
&\quad \text{else } \perp \\
&\sqsubseteq v
\end{aligned}$$

よって帰納法から $\text{bifldr } c \ n \ p$ の変更保存性は示された . □

2.5 双方向変換の例

2つ組の転置

2つ組の転置 trans は以下のように正当な双方向変換にできる .

$$\begin{aligned}
\llbracket \text{trans} \rrbracket_G ((s_{11}, s_{12}), (s_{21}, s_{22})) &= ((s_{11}, s_{21}), (s_{12}, s_{22})) \\
\llbracket \text{trans} \rrbracket_P (s, v) &= \llbracket \text{trans} \rrbracket_G v
\end{aligned}$$

この変換の正当性は以下の性質から自明である .

$$\llbracket \text{trans} \rrbracket_G \circ \llbracket \text{trans} \rrbracket_G = \text{id}$$

最小値

最小値を得る変換 min は以下のように定義される .

$$\begin{aligned}
 min &= \text{cond}_s p \text{ ft } sd \\
 \text{where } p \ x \ y &= x > y \\
 \llbracket ft \rrbracket_G (s_1, s_2) &= s_1 \\
 \llbracket ft \rrbracket_P (s_1, s_2) \ v &= \text{if } v < s_2 \text{ then } (v, s_2) \text{ else } \perp \\
 \llbracket sd \rrbracket_G (s_1, s_2) &= s_2 \\
 \llbracket sd \rrbracket_P (s_1, s_2) \ v &= \text{if } v < s_1 \text{ then } (s_1, v) \text{ else } \perp
 \end{aligned}$$

この変換は順方向変換には2つの数の最小値を返し, 逆変換時には, 変更後の値が最小値でない方の数より小さい場合についてのみ反映される. これは, 変更保存性を保つため, $\text{cond}_s p \ t_1 \ t_2$ が変更前と変更後で条件 p の真偽が変わることを許さないためである.

要素数のカウント

以下の定義により与えられる変換 $count \ c$ を考える .

$$\begin{aligned}
 count \ c &= \text{bifldr } (f \ c) \ 0 \ p \\
 \text{where } \llbracket f \ c \rrbracket_G (s_1, s_2) &= 1 + s_2 \\
 \llbracket f \ c \rrbracket_P ((s_1, s_2), v) &= (s_1, v - 1) \\
 \llbracket f \ c \rrbracket_P (\Omega, v) &= (c, v - 1) \\
 p \ v &= v > 0
 \end{aligned}$$

この変換は, 順方向変換においてリストの要素数を返す. 逆方向変換において, ビューであるカウントが加算されたときには, リストの末尾に要素 c が増加分だけ加えられ, カウントが減算されたときには, リストの末尾から要素を減少分だけ削除する. たとえば, $[1, 2]$ に対し, $\llbracket count \ 0 \rrbracket_G [1, 2] = 2$ となり,

$$\begin{aligned}
 &\llbracket count \ 0 \rrbracket_P ([1, 2], 1) \\
 &= s' : \llbracket count \ 0 \rrbracket_P (ss, v') \\
 &\quad \text{where } (s', v') = \llbracket f \ 0 \rrbracket_P ((1, \llbracket count \ 0 \rrbracket_G [2]), 1) \\
 &= \{ \llbracket f \ 0 \rrbracket_P ((1, -), 1) = (1, 0) \} \\
 &\quad 1 : \llbracket count \ 0 \rrbracket_P ([2], 0) \\
 &= 1 : [] \\
 &= [1]
 \end{aligned}$$

であり,

$$\begin{aligned}
& \llbracket \text{count } 0 \rrbracket_{\mathbf{P}} ([1, 2], 3) \\
&= s' : \llbracket \text{count } 0 \rrbracket_{\mathbf{P}} (ss, v') \\
&\quad \mathbf{where} (s', v') = \llbracket f \ 0 \rrbracket_{\mathbf{P}} ((1, \llbracket \text{count } 0 \rrbracket_{\mathbf{G}}[2]), 3) \\
&= \{ \llbracket f \ 0 \rrbracket_{\mathbf{P}} ((1, -), 3) = (1, 2) \} \\
&\quad 1 : \llbracket \text{count } 0 \rrbracket_{\mathbf{P}} ([2], 2) \\
&= \{ \text{同様に} \} \\
&\quad 1 : 2 : \llbracket \text{count } 0 \rrbracket_{\mathbf{P}} ([], 1) \\
&= 1 : 2 : s' : \llbracket \text{count } 0 \rrbracket_{\mathbf{P}} ([], v') \\
&\quad \mathbf{where} (s', v') = \llbracket f \ 0 \rrbracket_{\mathbf{P}} (\Omega, 1) \\
&= \{ \llbracket f \ 0 \rrbracket_{\mathbf{P}} (\Omega, 1) = (0, 0) \} \\
&\quad 1 : 2 : 0 : \llbracket \text{count } 0 \rrbracket_{\mathbf{P}} ([], 0) \\
&= 1 : 2 : 0 : [] \\
&= [1, 2, 0]
\end{aligned}$$

である.

第3章 複製を含む双方向変換

複製を用いることにより、1つのソースから複数のビューを定義することが可能になる。前章までの双方向変換の枠組みにおいて、複製変換 *copy* を、

$$\begin{aligned} \llbracket copy \rrbracket_G s &= (s, s) \\ \llbracket copy \rrbracket_P (s, (v_1, v_2)) &= \text{if } v_1 = v_2 \text{ then } v_1 \text{ else } \perp \end{aligned}$$

のように定義できるもののこれはあまり有用ではない。なぜなら、上の定義では、複製された値両方が同様に更新されない限り、元のデータに更新を書き戻すことができないからである。このように、前章の枠組みに従って定義された複数のビューは、ユーザにとって非常に使いにくいものになる。ユーザにとって使いやすい複数のビューを取り扱うために、楽観的複製 (Optimistic Replication)[?] を行う枠組みおよび、それを扱うための意味論が必要である。本章では、楽観的複製を行う複製変換およびそれを含んだ場合の意味論に対し議論していく。

3.1 複製変換

上で定義された複製変換 *copy* より楽観的な複製を行う複製変換 $[\delta, ?]$ は、基本的なデータに対し、以下のように定義される。

$$\begin{aligned} \llbracket \delta \rrbracket_G s &= (s, s) \\ \llbracket \delta \rrbracket_P (s, (v_1, v_2)) &= \text{if } s = v_1 \text{ then } v_2 \\ &\quad \text{else if } s = v_2 \text{ then } v_1 \\ &\quad \text{else } \perp \end{aligned}$$

但し、ここで、基本的なデータとは自然数などのように他のデータから構成されることのないデータを意味する。たとえば、組は他のデータ2つから構成されるため基本的なデータではない。関数 $\llbracket \delta \rrbracket_P$ は複製された2つのビューのうち、どちらか一方にのみ更新があった場合に、更新のあった方をソースに反映させる。もし、両方ともに更新があった場合は、その結果両方が同じであった場合はそれがソースに反映される。そうでない場合は、複製されたデータが基本的なデータであるためそれ以上分割して議論することができず、ビューをソースに反映することができない。

組などの基本的なデータでない場合、 δ を用いて以下のようにそのデータ用の複製変換を定義することができる。

$$dup_{pair} = (dup \times dup) \hat{;} trans$$

但し, dup はオーバーロードされた複製変換であり, dup_{pair} はその2つ組に対するものである. 逆方向変換 $\llbracket dup_{pair} \rrbracket_P$ により, 複製されたビューが2つとも変更された場合も, 片方が第1要素のみ変更され, 片方が第2要素のみ変更された場合はソースに反映できる.

この複製変換を用いることにより, ビューの中に依存性を持たせることができる[?]. たとえば, 変換 $\delta \hat{;} (Oneway f \times id)$ において, ビューの第1要素は変更できないものの, 第2要素を変更することで第1要素が自動的に変更される.

3.2 望ましい性質の議論

前節で定義された δ は前章の定義による「正当な双方向変換」ではない. すなわち, 変換 δ に対し, 反射性は

$$\llbracket \delta \rrbracket_P (s, \llbracket \delta \rrbracket_G s) = \llbracket \delta \rrbracket_P (s, (s, s)) = s$$

と成り立つのに対し, 変更保存性は

$$\llbracket \delta \rrbracket_G (\llbracket \delta \rrbracket_P (0, (0, 3))) = \llbracket \delta \rrbracket_G 3 = (3, 3) \neq (0, 3)$$

のように成り立たない. しかし, 変更保存性を要求しない場合,

$$\begin{aligned} \llbracket f \rrbracket_G s &= 2 * s \\ \llbracket f \rrbracket_P (s, v) &= \text{if } v = 2 * s \text{ then } s \text{ else } v/4 \end{aligned}$$

などのように無意味な変換も記述できてしまう. そのため, 変更保存性に代わる概念が必要となる.

これまで, このような概念は, 双方向変換の冪等性 (idempotence), すなわち変換 f が

$$\begin{aligned} \llbracket f \rrbracket_G (\llbracket f \rrbracket_P (s, \llbracket f \rrbracket_G s)) &\sqsubseteq \llbracket f \rrbracket_G s \\ \llbracket f \rrbracket_P (\llbracket f \rrbracket_P (s, v), \llbracket f \rrbracket_G (\llbracket f \rrbracket_P (s, v))) &\sqsubseteq \llbracket f \rrbracket_P (s, v) \end{aligned}$$

を満たすことを複製を含む双方向変換の満たすべき性質であるとして議論されてきた[?, ?]. しかし, 彼らが双方向変換を利用した文書内の依存性解決に主眼を置いているのに対し, 本論文では[?]を拡張し, 複数のビューがある場合の定式化を目指している. そのため, δ に対しても成り立つ反射性は双方向変換が満たすべき性質の1つであると考えられる.

冪等性に対し, 以下の性質が成り立つ.

定理 3.1. 反射的な双方向変換は冪等である.

証明. まず, $\llbracket f \rrbracket_G (\llbracket f \rrbracket_P (s, \llbracket f \rrbracket_G s)) \sqsubseteq \llbracket f \rrbracket_G s$ であることを示す.

$$\begin{aligned} \llbracket f \rrbracket_G (\llbracket f \rrbracket_P (s, \llbracket f \rrbracket_G s)) &\sqsubseteq \{ f \text{ の反射性} \} \\ &\llbracket f \rrbracket_G s \\ &= \text{右辺} \end{aligned}$$

つぎに, $\llbracket f \rrbracket_P (\llbracket f \rrbracket_P (s, v), \llbracket f \rrbracket_G (\llbracket f \rrbracket_P (s, v))) \sqsubseteq \llbracket f \rrbracket_P (s, v)$ を示す .

$$\begin{aligned} \llbracket f \rrbracket_P (\llbracket f \rrbracket_P (s, v), \llbracket f \rrbracket_G (\llbracket f \rrbracket_P (s, v))) &\sqsubseteq \{ f \text{ の反射性} \} \\ &\llbracket f \rrbracket_P (s, v) \\ &= \text{右辺} \end{aligned}$$

ゆえに, 反射的な双方向変換は冪等である . □

そのため, 冪等以外の性質の議論が必要になる . よって次章において, 簡単な言語を定め, その上で満たすべき性質を定式化し, 議論していく .

第4章 双方向言語

4.1 複数のビューの表現

複製を含む双方向変換を使用することにより、複数のビューを取り扱うことができる。ここでは、そのような複数のビューを扱う場合に、どのような言語を用いて複数のビューを構築するのか、そして、どのようなものが「正当な」双方向変換になるかを議論していく。

4.1.1 扱うデータ

変換 δ は 2 つ組を作成する。そのため、変更保存性を複製を含む双方向変換に拡張する場合、2 つ組の各々に対して、変更保存性を議論するのが自然である。しかし、複製を含まない双方向変換と異なり、2 つ組という構造を必然的に含むため、扱うデータに対する議論が必要になる。そこで、本章では、

$$D ::= B \quad \{ \text{基本的なデータ} \} \\ | (D, D) \quad \{ 2 \text{ つ組} \}$$

のように定義されるデータを考える。つまり、ここで扱うデータは、自然数などの基本的なデータであるか、扱うデータ同士の 2 つ組である。但し、2 つ組が無限回入れ子になった構造は許さない。

この上で変更保存性に対する概念として次のようなものが考えられる。

定義 4.1 (局所変更保存性). ソース S からビュー \mathcal{V} への双方向変換 f を考える。このとき、以下のような場合に局所変更保存性を f が満たすと定義する。

- \mathcal{V} が基本的なデータで組ではない場合は、 f が変更保存性を満たす。
- \mathcal{V} が組 $(\mathcal{V}_1, \mathcal{V}_2)$ である場合は、 S から \mathcal{V}_1 への変換 $f \hat{;} fst$ が局所変更保存性を満たし、かつ、 S から \mathcal{V}_2 への変換 $f \hat{;} snd$ が局所変更保存性を満たす。

但し、 fst と snd は射影変換である。 □

これは、複数のビューが存在した場合に、そのそれぞれのみを変更した場合に、そのそれぞれに対し変更保存性が成り立つこと意味している。

局所変更保存性の変更保存性の真の拡張になっていることを示す前に、以下の補題を証明する。

補題 4.2. 複製変換と射影変換の合成は恒等変換と等価である．つまり次が成り立つ．

$$\delta \hat{;} fst = \delta \hat{;} snd = id$$

証明. 対称であるため変換 fst についてのみ示す．

まず，順方向の変換については， $[[\delta \hat{;} fst]]_G s = [[fst]]_G ([[\delta]_G s) = [[fst]]_G (s, s) = s$ により示された．

次に逆方向の変換について，

$$\begin{aligned} [[\delta \hat{;} fst]]_P (s, v) &= [[\delta]]_P (s, [[fst]]_P ([[\delta]_G s, v)) \\ &= [[\delta]]_P (s, [[fst]]_P ((s, s), v)) \\ &= [[\delta]]_P (s, (v, s)) \\ &= \text{if } s = v \text{ then } s \\ &\quad \text{else if } s = s \text{ then } v \\ &\quad \text{else } \perp \\ &= v \\ &= [[id]]_P (s, v) \end{aligned}$$

となる．よって， $\delta \hat{;} fst = id$ である． □

これより，次のことが証明できる．

定理 4.3. 局所変更保存性は変更保存性の真の拡張である．つまり，以下が成り立つ．

1. 双方向変換 f が変更保存性を満たすならば， f は局所変更保存性を満たす．
2. 局所変更保存性を満たし，変更保存性を満たさない双方向変換が存在する．

証明. まず，簡単である後者を示す．後者は， δ が前述のように変更保存性を満たさず，補題により $\delta \hat{;} fst = \delta \hat{;} snd = id$ であるため局所変更保存性を満たすことから示される．

次に，前者である，双方向変換が変更保存性を満たすのならば，局所変更保存性を満たすことを帰納法により示す．

ビューが組でないとき，明らかにすべての変更保存性を満たす双方向変換は局所変更保存性を満たす．

ソースが S でビューが $(\mathcal{V}_1, \mathcal{V}_2)$ の双方向変換 f について， S から \mathcal{V}_1 へのすべての双方向変換と， S から \mathcal{V}_2 へのすべての双方向変換が変更保存性を満たすとき局所変更保存性を満たすことを仮定する．このとき， S から $(\mathcal{V}_1, \mathcal{V}_2)$ への双方向変換 f が変更保存性を満たすという前提条件により， $f \hat{;} fst$ および $f \hat{;} snd$ が変更保存性を満たす．よって，帰納法の仮定より $f \hat{;} fst$ および $f \hat{;} snd$ は局所変更保存性を満たす．ゆえに， S から $(\mathcal{V}_1, \mathcal{V}_2)$ への双方向変換 f は局所変更保存性を満たす．

帰納法により，双方向変換が変更保存性を満たすのならば，局所変更保存性を満たすことが示された． □

4.1.2 複数のビューを記述する言語の設計と解析

この局所変更保存性は変換の合成 $\hat{;}$ に対して保存しない．たとえば，

$$\begin{aligned} \llbracket addl \rrbracket_G (s_1, s_2) &= s_1 + s_2 \\ \llbracket addl \rrbracket_P ((s_1, s_2), v) &= (v - s_2, s_2) \end{aligned}$$

は，変更保存性を満たし， δ は局所変更保存性を満たすものの $\delta \hat{;} addl$ は局所変更保存性を満たさない．このことは，

$$\begin{aligned} &\llbracket \delta \hat{;} addl \rrbracket_G (\llbracket \delta \hat{;} addl \rrbracket_P (1, 5)) \\ &= \llbracket \delta \hat{;} addl \rrbracket_G (\llbracket \delta \rrbracket_P (1, \llbracket addl \rrbracket_P (\llbracket \delta \rrbracket_G 1, 5))) \\ &= \llbracket \delta \hat{;} addl \rrbracket_G (\llbracket \delta \rrbracket_P (1, \llbracket addl \rrbracket_P ((1, 1), 5))) \\ &= \llbracket \delta \hat{;} addl \rrbracket_G (\llbracket \delta \rrbracket_P (1, (4, 1))) \\ &= \llbracket \delta \hat{;} addl \rrbracket_G 4 \\ &= \llbracket addl \rrbracket_G (\llbracket \delta \rrbracket_G 4) \\ &= \llbracket addl \rrbracket_G (4, 4) \\ &= 8 \end{aligned}$$

のように確認できる．そこで，本節では小さな言語を設計し，その上で局所変更保存性に対し解析を行う．

次のような言語を考える．

$$\begin{aligned} f &::= f \times f \quad \{ \text{直積} \} \\ &| \delta \quad \{ \text{複製変換} \} \\ &| fst \quad \{ \text{射影 (第 1 要素)} \} \\ &| snd \quad \{ \text{射影 (第 2 要素)} \} \\ &| f \hat{;} f \quad \{ \text{合成} \} \\ &| t \quad \{ \text{変更保存性を満たす双方向変換} \} \end{aligned}$$

この言語の変換の意味は，それぞれ今まで定義された通りに与えられる．この言語により複数のビューを扱うことができる．たとえば，

$$dup \hat{;} (t_1 \times t_2)$$

により，変換 t_1 が適用されたビューと変換 t_2 が適用されたビューを得ることができる．但し， dup はオーバーロードされた複製変換であり，基本的なデータに対しては $dup = \delta$ ，2 つ組に対しては $dup = (dup \times dup) \hat{;} trans$ と定義される．

この言語において，次の書き換えを考える．

$$\begin{aligned}
 \delta \hat{;} fst &\longrightarrow \text{id} \\
 \delta \hat{;} snd &\longrightarrow \text{id} \\
 f \hat{;} \text{id} &\longrightarrow f \\
 \text{id} \hat{;} f &\longrightarrow f \\
 \text{id} \times \text{id} &\longrightarrow \text{id} \\
 (f_1 \times f_2) \hat{;} (f_3 \times f_4) &\longrightarrow (f_1 \hat{;} f_3) \times (f_2 \hat{;} f_4) \\
 (f_1 \times f_2) \hat{;} fst &\longrightarrow fst \hat{;} f_1 \\
 (f_1 \times f_2) \hat{;} snd &\longrightarrow snd \hat{;} f_2
 \end{aligned}$$

この書き換えについて，以下の性質が成り立つ．

定理 4.4 (停止性). 上の書き換えは停止する．

証明. 左辺と右辺を比較して，項数と \times の数の和が減少していることからいえる． \square

定理 4.5 (書き換え後の変換との関係). 変換 f に対し，上記の構文的な書き換えをほどこす．この書き換えが停止することにより得られた変換 f' とする．変換 f' が δ を含まないなら， f は変更保存性を満たす．

証明. 左辺の変換と右辺の変換の関係を示していく．

以前に示した通り，

$$\begin{aligned}
 \delta \hat{;} fst &= \text{id} \\
 \delta \hat{;} snd &= \text{id}
 \end{aligned}$$

であり，明らかに，

$$\begin{aligned}
 f \hat{;} \text{id} &= f \\
 \text{id} \hat{;} f &= f \\
 \text{id} \times \text{id} &\preceq \text{id}
 \end{aligned}$$

である．

順方向変換について，

$$\begin{aligned}
 & \llbracket (f_1 \times f_2) \hat{;} (f_3 \times f_4) \rrbracket_G (s_1, s_2) \\
 &= \llbracket f_3 \times f_4 \rrbracket_G (\llbracket f_1 \times f_2 \rrbracket_G (s_1, s_2)) \\
 &= \llbracket f_3 \times f_4 \rrbracket_G (\llbracket f_1 \rrbracket_G s_1, \llbracket f_2 \rrbracket_G s_2) \\
 &= (\llbracket f_3 \rrbracket_G (\llbracket f_1 \rrbracket_G s_1), \llbracket f_4 \rrbracket_G (\llbracket f_2 \rrbracket_G s_2))
 \end{aligned}$$

であり,

$$\begin{aligned} & \llbracket (f_1 \hat{;} f_3) \times (f_2 \hat{;} f_4) \rrbracket_G (s_1, s_2) \\ &= (\llbracket f_1 \hat{;} f_3 \rrbracket_G s_1, \llbracket f_2 \hat{;} f_4 \rrbracket_G s_2) \\ &= (\llbracket f_3 \rrbracket_G (\llbracket f_1 \rrbracket_G s_1), \llbracket f_4 \rrbracket_G (\llbracket f_2 \rrbracket_G s_2)) \end{aligned}$$

より,

$$\llbracket (f_1 \times f_2) \hat{;} (f_3 \times f_4) \rrbracket_G = \llbracket (f_1 \hat{;} f_3) \times (f_2 \hat{;} f_4) \rrbracket_G (s_1, s_2)$$

となる. 逆方向変換について,

$$\begin{aligned} & \llbracket (f_1 \times f_2) \hat{;} (f_3 \times f_4) \rrbracket_P ((s_1, s_2), (v_1, v_2)) \\ &= \llbracket f_1 \times f_2 \rrbracket_P ((s_1, s_2), \llbracket f_3 \times f_4 \rrbracket_P (\llbracket f_1 \times f_2 \rrbracket_G (s_1, s_2), (v_1, v_2))) \\ &= \llbracket f_1 \times f_2 \rrbracket_P ((s_1, s_2), \llbracket f_3 \times f_4 \rrbracket_P (\llbracket f_1 \rrbracket_G s_1, \llbracket f_2 \rrbracket_G s_2), (v_1, v_2))) \\ &= \llbracket f_1 \times f_2 \rrbracket_P ((s_1, s_2), (\llbracket f_3 \rrbracket_P (\llbracket f_1 \rrbracket_G s_1, v_1), \llbracket f_4 \rrbracket_P (\llbracket f_2 \rrbracket_G s_2, v_2))) \\ &= (\llbracket f_1 \rrbracket_P (s_1, \llbracket f_3 \rrbracket_P (\llbracket f_1 \rrbracket_G s_1, v_1)), \llbracket f_2 \rrbracket_P (s_2, \llbracket f_4 \rrbracket_P (\llbracket f_2 \rrbracket_G s_2, v_2))) \end{aligned}$$

および,

$$\begin{aligned} & \llbracket (f_1 \hat{;} f_3) \times (f_2 \hat{;} f_4) \rrbracket_P ((s_1, s_2), (v_1, v_2)) \\ &= (\llbracket f_1 \hat{;} f_3 \rrbracket_P (s_1, v_1), \llbracket f_2 \hat{;} f_4 \rrbracket_P (s_2, v_2)) \\ &= (\llbracket f_1 \rrbracket_P (s_1, \llbracket f_3 \rrbracket_P (\llbracket f_1 \rrbracket_G s_1, v_1)), \llbracket f_2 \rrbracket_P (s_2, \llbracket f_4 \rrbracket_P (\llbracket f_2 \rrbracket_G s_2, v_2))) \end{aligned}$$

より,

$$\llbracket (f_1 \times f_2) \hat{;} (f_3 \times f_4) \rrbracket_P = \llbracket (f_1 \hat{;} f_3) \times (f_2 \hat{;} f_4) \rrbracket_P$$

となる. よって,

$$(f_1 \times f_2) \hat{;} (f_3 \times f_4) = (f_1 \hat{;} f_3) \times (f_2 \hat{;} f_4)$$

がいえる.

順方向変換について, $s_2 = \perp$ の場合,

$$\llbracket (f_1 \times f_2) \hat{;} fst \rrbracket_G (s_1, s_2) = \llbracket fst \hat{;} f_1 \rrbracket_G (s_1, s_2) = \perp$$

であり, $s_2 \neq \perp$ の場合,

$$\begin{aligned} \llbracket (f_1 \times f_2) \hat{;} fst \rrbracket_G (s_1, s_2) &= \llbracket fst \rrbracket_G (\llbracket f_1 \times f_2 \rrbracket_G (s_1, s_2)) \\ &= \llbracket fst \rrbracket_G (\llbracket f_1 \rrbracket_G s_1, \llbracket f_2 \rrbracket_G s_2) \\ &\sqsubseteq \{ \text{正格性} \} \\ &\quad \llbracket f_1 \rrbracket_G s_1 \end{aligned}$$

および,

$$\begin{aligned} \llbracket fst \hat{;} f_1 \rrbracket_G (s_1, s_2) &= \llbracket f_1 \rrbracket_G (\llbracket fst \rrbracket_G (s_1, s_2)) \\ &= \{ s_2 \neq \perp \} \\ &\quad \llbracket f_1 \rrbracket_G s_1 \end{aligned}$$

である. よって,

$$\llbracket (f_1 \times f_2) \hat{;} fst \rrbracket_G \preceq \llbracket fst \hat{;} f_1 \rrbracket_G$$

となる. 逆方向変換について,

$$\begin{aligned} \llbracket (f_1 \times f_2) \hat{;} fst \rrbracket_P ((s_1, s_2), v) &= \llbracket f_1 \times f_2 \rrbracket_P ((s_1, s_2), \llbracket fst \rrbracket_P (\llbracket f_1 \times f_2 \rrbracket_G (s_1, s_2), v)) \\ &= \llbracket f_1 \times f_2 \rrbracket_P ((s_1, s_2), \llbracket fst \rrbracket_P (\llbracket f_1 \rrbracket_G s_1, \llbracket f_2 \rrbracket_G s_2), v)) \\ &\sqsubseteq \{ \text{正格性} \} \\ &\quad \llbracket f_1 \times f_2 \rrbracket_P ((s_1, s_2), (v, \llbracket f_2 \rrbracket_G s_2)) \\ &= (\llbracket f_1 \rrbracket_P (s_1, v), \llbracket f_2 \rrbracket_P (s_2, \llbracket f_2 \rrbracket_G s_2)) \\ &\sqsubseteq \{ f_2 \text{ の反射性} \} \\ &\quad (\llbracket f_1 \rrbracket_P (s_1, v), s_2) \end{aligned}$$

および,

$$\begin{aligned} \llbracket fst \hat{;} f_1 \rrbracket_P ((s_1, s_2), v) &= \llbracket fst \rrbracket_P ((s_1, s_2), \llbracket f_1 \rrbracket_P (\llbracket fst \rrbracket_G (s_1, s_2), v)) \\ &= \llbracket fst \rrbracket_P ((s_1, s_2), \llbracket f_1 \rrbracket_P (s_1, v)) \\ &= (\llbracket f_1 \rrbracket_P (s_1, v), s_2) \end{aligned}$$

より,

$$\llbracket (f_1 \times f_2) \hat{;} fst \rrbracket_P \preceq \llbracket fst \hat{;} f_1 \rrbracket_P$$

となる. よって,

$$(f_1 \times f_2) \hat{;} fst \preceq fst \hat{;} f_1$$

がいえる.

これから変換 f を 1 回書き換えるて得られた変換 f' について,

$$f \preceq f'$$

が成り立つことがわかる. ゆえに, f' が変更保存性を満たすなら f も変更保存性を満たす. \square

この書き換えにより簡単に局所変更保存性を確認できる例を示す. 双方向変換

$$f = (\delta \times \text{id}) \hat{;} \delta \hat{;} ((fst \hat{;} fst) \times (snd \times \text{id})) \hat{;} (t_1 \times t_2)$$

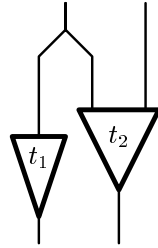


図 4.1. 局所保存性を満たす変換

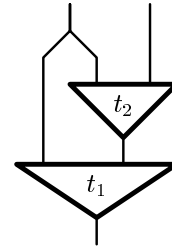


図 4.2. 局所保存性を満たすとは限らない変換

を考える．この f の依存性を表したものが図 4.1 である．図中の上がソースを表し，下がビューを表している．図の示す通り，この変換 f は，ソース (s_1, s_2) に対し， s_1 から t_1 によって作られたビュー v_1 および， (s_1, s_2) から t_2 によって作られたビュー v_2 との組 (v_1, v_2) を作成する．この変換 f に対して，

$$\begin{aligned}
 f \hat{;} fst &= (\delta \times \text{id}) \hat{;} \delta \hat{;} ((fst \hat{;} fst) \times (snd \times \text{id})) \hat{;} (t_1 \times t_2) \hat{;} fst \\
 &\longrightarrow (\delta \times \text{id}) \hat{;} \delta \hat{;} ((fst \hat{;} fst) \times (snd \times \text{id})) \hat{;} fst \hat{;} t_1 \\
 &\longrightarrow (\delta \times \text{id}) \hat{;} \delta \hat{;} fst \hat{;} (fst \hat{;} fst) \hat{;} t_1 \\
 &\longrightarrow (\delta \times \text{id}) \hat{;} (fst \hat{;} fst) \hat{;} t_1 \\
 &\longrightarrow fst \hat{;} \delta \hat{;} fst \hat{;} t_1 \\
 &\longrightarrow fst \hat{;} t_1
 \end{aligned}$$

であり，

$$\begin{aligned}
 f \hat{;} snd &= (\delta \times \text{id}) \hat{;} \delta \hat{;} ((fst \hat{;} fst) \times (snd \times \text{id})) \hat{;} (t_1 \times t_2) \hat{;} snd \\
 &\longrightarrow (\delta \times \text{id}) \hat{;} \delta \hat{;} ((fst \hat{;} fst) \times (snd \times \text{id})) \hat{;} snd \hat{;} t_2 \\
 &\longrightarrow (\delta \times \text{id}) \hat{;} \delta \hat{;} snd \hat{;} (snd \times \text{id}) \hat{;} t_2 \\
 &\longrightarrow (\delta \times \text{id}) \hat{;} (snd \times \text{id}) \hat{;} t_2 \\
 &\longrightarrow ((\delta \hat{;} snd) \times (\text{id} \hat{;} \text{id})) \hat{;} t_2 \\
 &\longrightarrow (\text{id} \times \text{id}) \hat{;} t_2 \\
 &\longrightarrow \text{id} \hat{;} t_2 \\
 &\longrightarrow t_2
 \end{aligned}$$

であるため， f が局所保存性を満たすことが簡単に確認できる．

逆に図 4.2 に表される変換

$$(\delta \times \text{id}) \hat{;} \delta \hat{;} ((fst \hat{;} fst) \times (snd \times \text{id})) \hat{;} (\text{id} \times t_2) \hat{;} t_1$$

はこの方法により確認することはできない．たとえば， $t_1 = t_2 = addl$ のとき，上記の変換は変更保存性を満たさない．

つまり，このことはユーザが一部の变換の変更保存性を保証するだけで，変換の局所変更保存性を健全に判定できることを意味している．また，完全性は成り立たない．たとえば，

$$\begin{aligned} \llbracket add' \rrbracket_G (s_1, s_2) &= \text{if } s_1 = s_2 \text{ then } s_1 + s_2 \text{ else } \perp \\ \llbracket add' \rrbracket_P ((s_1, s_2), v) &= (v/2, v/2) \end{aligned}$$

に対し， $\delta \hat{;} add'$ は，複製変換を含み上の書き換え規則により書き換えられないが，

$$\begin{aligned} \llbracket \delta \hat{;} add' \rrbracket_G (\llbracket \delta \hat{;} add' \rrbracket_P (s, v)) &= \llbracket \delta \hat{;} add' \rrbracket_G (\llbracket \delta \rrbracket_P (s, \llbracket add' \rrbracket_P (\llbracket \delta \rrbracket_G s, v))) \\ &= \llbracket \delta \hat{;} add' \rrbracket_G (\llbracket \delta \rrbracket_P (s, (v/2, v/2))) \\ &= \llbracket \delta \hat{;} add' \rrbracket_G (v/2) \\ &= \llbracket add' \rrbracket_G (\llbracket \delta \rrbracket_G (v/2)) \\ &= \llbracket add' \rrbracket_G (v/2, v/2) \\ &= v/2 + v/2 \\ &= v \end{aligned}$$

となり変更保存性を満たす．

4.2 X^\pm 言語と変換システム

ここでは「梅林」の実装に使用される拡張された X 言語 [?] を説明する．拡張された X 言語は，Hu らが木構造の変換に使用した X 言語 [?] を元に，

- ソースとして子が集合である木を扱う
- 複製変換が複製先として大域的なパスを取る

といった拡張を加えたものである．これを， X^\pm 言語と呼ぶ．複製変換が複製先として大域的なパスをとることができるため，スマートフォルダのような構造が作成可能になっている．

4.2.1 双方向変換システムの状態

ここでは [?] に倣いソースの状態，ビューを得るための双方向変換，ビューの状態の3つ組を，双方向変換システムの状態と定義する．

定義 4.6 (双方向変換システムの状態). 双方向変換システムは3つ組 (s, x, v) を状態として持つ．

- ソース $s \in \mathcal{S}$

- 双方向変換 $x \in \mathcal{X}_{(S, \mathcal{V})}$
- ビュー $v \in \mathcal{V}$

但し, $\mathcal{X}_{(S, \mathcal{V})}$ は, S から \mathcal{V} への反射的な双方向変換の部分集合である. □

システムで記述することのできる双方向変換 $\mathcal{X}_{(S, \mathcal{V})}$ の集合は, 双方向変換記述言語を定めることにより暗黙に与えられる.

また, ソースやビューに変更があった場合に, 順方向変換や逆方向変換によってビューやソースに変更が反映される. このような変更の伝播が停止した状態を定義する.

定義 4.7 (定常な状態). 以下の条件を満たす状態 (s, x, v) を定常な状態という.

$$\begin{aligned} \llbracket x \rrbracket_G s &= v \\ \llbracket x \rrbracket_P (s, v) &= s \end{aligned}$$

□

変更の伝播によりこの定常な状態に達成することが重要になる.

定理 4.8. 定常な状態 (s, x, v) に対し, ソースおよびビューに変更があった場合に以下のように定常な状態を達成できる.

- ソースが s' に変更された場合

$$\begin{aligned} \llbracket x \rrbracket_P (s', \llbracket x \rrbracket_G s') \neq \perp &\Rightarrow (s', x, \llbracket x \rrbracket_G s') \\ \llbracket x \rrbracket_P (s', \llbracket x \rrbracket_G s') = \perp &\Rightarrow (s, x, v) \end{aligned}$$

- ビューが v' に変更された場合

$$\begin{aligned} \llbracket x \rrbracket_P (\llbracket x \rrbracket_P (s, v'), \llbracket x \rrbracket_G (\llbracket x \rrbracket_P (s, v'))) \neq \perp &\Rightarrow (\llbracket x \rrbracket_P (s, v'), x, \llbracket x \rrbracket_G (\llbracket x \rrbracket_P (s, v'))) \\ \llbracket x \rrbracket_P (\llbracket x \rrbracket_P (s, v'), \llbracket x \rrbracket_G (\llbracket x \rrbracket_P (s, v'))) = \perp &\Rightarrow (s, x, v) \end{aligned}$$

証明. 反射的な双方向変換の冪等性から証明できる. □

未定義 \perp は計算が停止しない場合を含むため, \perp と等しいかどうかを計算することはできない. しかし, X^\pm 言語は計算が停止しないような再帰を含まないので, 変換の定義域かどうかを判定する関数を利用したり, 例外を扱ったりすることにより判定することができる.

4.2.2 操作主導な双方向変換の取り扱い

「梅林」で取り扱うディレクトリ木では、ソースの変更は OS のシステムコールやライブラリ関数によって行わなければならない。こういった枠組みでは、直接ソース全体を更新することはできず、特定のノードの追加、削除、内容の変更といった単位でソースの変更を行う必要がある。以下では、そのような操作主導 (Operation-based) の枠組みでの双方向変換について議論していく。

ソースの上では、ファイルの削除や新規ファイルの作成などの、いくつかの原子的な編集操作が定義されている。これを以下のように定式化する。

定義 4.9 (原子編集操作). ソースの上で定義されている $S \rightarrow S$ である関数の集合を $A(S)$ と書き、その元を原子編集操作という。また、この原子編集操作の列で行う編集の集合 $A^*(S)$ を

$$A^*(S) = \{a_k \circ \dots \circ a_1 \mid k \geq 1, \forall j : 1 \leq j \leq k. a_j \in A(S)\}$$

と定義する。 □

ソースは原子編集操作を通してしか変更できないため、ビュー上での操作は必ずソース上で定義された原子編集操作の列に翻訳されなければならない。

定義 4.10 (ビュー上の許容される操作). すべての S から \mathcal{V} への反射的な双方向変換 x について、ビュー上の許容される編集 $A(\mathcal{V})$ と

$$A(\mathcal{V}) = \{a_v \mid \exists e_s \in A^*(S), \llbracket x \rrbracket_P (s, a_v v) = e_s s\}$$

定める。 □

ビュー上の操作がソース上の操作に翻訳されるためには、ビュー上での操作は $A(\mathcal{V})$ の元の列として表されなければならない。 $A(\mathcal{V})$ はビュー上の操作を構成する最小単位ととらえることができるため、 $A(S)$ と同様の表記法を用いる。

我々の目的は、 $A(\mathcal{V})$ の元に対し、 $\mathcal{X}_{(S, \mathcal{V})}$ の各要素について対応する $A^*(S)$ の元を求めることである。しかし、このような編集の翻訳は難しい。なぜなら、 $A(\mathcal{V})$ や $A^*(S)$ の元が関数であり、取り扱いが難しいためである。ここで、 $[?]$ において、リスト間の対応関係などの状態の整合性を保ちつつ編集の反映を行うために利用されたマーク付けの技術を利用する。ビュー上で編集操作が行われたデータに対してマークを付け、それを逆方向の変換によりソースに反映させることにより、編集操作をソースに伝播することを考える。

定義 4.11 (マーク付けによる編集操作の反映). ビュー上での編集操作の集合 $E \subseteq A(\mathcal{V})$ が与えられているとする。任意の $a_s \in A(S)$ および $a_v \in E$ に対し、

$$\begin{aligned} \text{mark}_s a_s &:: S \rightarrow S' \\ \text{mark}_v a_v &:: \mathcal{V} \rightarrow \mathcal{V}' \end{aligned}$$

という関数およびこれらに対し、

$$\begin{aligned} \text{reflect}_v (\text{mark}_v a_v v) &= a_v v \\ \text{reflect}_s (\text{mark}_s a_s s) &= a_s s \end{aligned}$$

である関数 reflect_s と reflect_v を考える．このとき、

$$\text{reflect}_s (\llbracket x \rrbracket'_P (s, \text{mark}_v a_v v)) = \llbracket x \rrbracket_P (s, a_v v)$$

となる $\llbracket x \rrbracket'_P$ および mark_s , mark_v , reflect_s , reflect_v が定義できることをマーク付けによる編集操作が正しく反映できるという．□

この定義において、 S' や \mathcal{V}' は S や \mathcal{V} に対応するマーク付けを含んだデータ構造を表している．通常 $A(\mathcal{V})$ を求めることは容易ではないため、ビュー上で行う編集の集合 E は $A(\mathcal{V})$ の部分集合となっている．ここで、 reflect_s はマーク付けされたソースを元にソース上での編集操作を実行する関数ととらえることができる．変換 x に対して、 $\llbracket x \rrbracket'_P$ および mark_s , mark_v , reflect_s , reflect_v を定義することにより、 $\llbracket x \rrbracket_P$ が定義でき、このマーク付けによる編集操作の反映性を考えなくてよくなるため、今後は $\llbracket x \rrbracket'_P$ および mark_s , mark_v , reflect_s , reflect_v に対して議論を行う．

4.2.3 扱うデータ構造

ディレクトリ木のような子が集合であるデータを扱うため、以下のようなデータ構造を用いる．

$$\text{data Tree} = N (C, M, K, S) \{Tree\}$$

ここで、 C は木構造が保持するデータであり、 M は前述のマーク、 K は子集合の中で一意であるキー、 S はソートに使用される値を示す．簡便のため、マークが付いていないというマークを導入することにより、マーク付きのデータとそうでないデータを同一のデータ構造にて扱う．キー K を C とは別に用いるのは、あるノードに変更があった場合に変換が同じノードを適用されることを保証するのに便利であるからである．ソースが子が集合である木なのに対し、ビューが子がリストある木であるため、ともに子が集合である木で扱えるようにするために、ソートで使用する値 S を導入した．定式化上は、集合のほうがリストより扱いやすいためである．なお実装はリストを用いて行っている．そのため、ソースの等価性について S の値は無視される．

ソースであるディレクトリ木などは、 K や S などを含んでいない．そのため、ソースの木を作成するために K や S を適切に定めなければならない、また、同じディレクトリ木から常に同じビューを作成できることを保証するために、 reflect_s において K や S がディレクトリ木からソース木を得たのと同じやり方で適切に初期化されると仮定する．

たとえば, a という名前のディレクトリ下に, a, b, c というファイルが存在するというディレクトリ構造からソース木を得ると,

$$N((a, Directory, 4096, \dots), Normal, a, a) \left\{ \begin{array}{l} N((a, NormalFile, 125, \dots), Normal, a, a) \emptyset \\ N((b, NormalFile, 498, \dots), Normal, b, b) \emptyset \\ N((c, NormalFile, 267, \dots), Normal, c, c) \emptyset \end{array} \right\}$$

のようになる. C としては, ファイル名, ファイル種類やファイルサイズなど `stat(2)` で習得できるような情報が使用される. キーはあるノードの下で異なっていればよいので, ハードリンクが存在しない場合には, ファイル名を用いることができる. ソートのために使用される値は, ファイル名, ファイルサイズ, 最終アクセス時間などの値が使用される. この値は, ファイルマネージャは木構造の表示にあたって木の子に何らかの順番を与えなければならないために必要である.

また変換 $extract\ k$ を,

$$\begin{aligned} \llbracket extract\ k \rrbracket_G (N - cs) &= \text{if } N(c, m, k, s) t \in cs \text{ then } N(c, m, k, s) t \text{ else } \perp \\ \llbracket extract\ k \rrbracket_P ((N\ d\ cs), t) &= N\ d\ (\{t\} \cup cs) \end{aligned}$$

と定義する.

この木上の編集操作として, 挿入, 削除, 置換を考える. それぞれに対応したマーク付けを以下のように行う.

- 挿入の場合, 新規に挿入されるノードを追加し, そのノードのマークを *Add* にする.
- 削除の場合, 削除されるノードを削除せず, そのノードのマークを *Del* にする.
- 置換の場合, 置換の対象となるノードの保持する値を c に置き換えず, そのノードのマークを $Mod\ c$ にする.

簡便のため, ビューに対し1つの編集操作が行われるたびに逆方向変換, 順方向変換による同期を行うことを仮定する. また, 本節において今後マーク付けされたビューのことを単にビューと呼ぶことにする.

キー K を用いて定義された $extract\ k$ などの変換は, 置換の際に適切に k の値を変更してやらなければ, 置換前と置換後で変換の適用先などが変更されてしまう. たとえば,

$$N(1, Normal, 1, 1) \left\{ \begin{array}{l} N(2, Normal, 2, 2) \emptyset \\ N(3, Normal, 3, 3) \emptyset \end{array} \right\}$$

というソースから変換 $extract\ 2$ により, ビュー $N(2, Normal\ 2, 2) \emptyset$ を得ている場合に, ビューのノード値を2から4へ置換すると, $reflect$ 後の対応するソースは

$$N(1, Normal, 1, 1) \left\{ \begin{array}{l} N(4, Normal, 4, 4) \emptyset \\ N(3, Normal, 3, 3) \emptyset \end{array} \right\}$$

となり, *extract* 2 が順方向変換時に \perp を返す. これは変換が $N(2, Normal, 2, 2) \emptyset$ というノードそのものに適用されていると考え方において, 望ましくない. そのため *reflect* 前の

$$N(1, Normal, 1, 1) \left\{ \begin{array}{l} N(2, Mod\ 4, 2, 2) \emptyset \\ N(3, Normal, 3, 3) \emptyset \end{array} \right\}$$

というソースを利用して, 変換 *extract* 2 のキー 2 を 4 に書き換えてこれを回避する. このように, $\llbracket x \rrbracket'_P(s, v)$ により得られたマーク付きソースを利用し, 順方向変換を通して, すべての置換されたノードを表すキーを置換後のノードに対応するキーに変換する操作を α と書く. つまり, このとき変更されたビューから, ビューの変更が書き戻されたソースのビューを得るための操作を *synchronize* と書くと,

$$\begin{aligned} \text{synchronize}(s, x, v) &= (s'', x', \llbracket x' \rrbracket_G s'') \\ \text{where } x' &= \alpha s' x \\ s' &= \llbracket x \rrbracket'_P(s, v) \\ s'' &= \text{reflect}_s s' \end{aligned}$$

となる. つまりここから, ここで扱う双方向変換システムにおいてソースと変換は結合が高いものになっていることがいえる.

この言語の上で局所変更保存性は [?] に倣い次のように定義される.

定義 4.12. 子が集合である木上の局所変更保存性このデータ構造上の双方向変換 x が局所変更保存性を満たすとは,

- $\forall s \in \mathcal{S}, \forall v \in \mathcal{V}'$,
 $N(c, Add, -, -) \in \text{children } v \wedge \llbracket x \rrbracket_G(\text{reflect}_s \llbracket x \rrbracket'_P(s, v)) \neq \perp$
 $\Rightarrow N(c, -, -, -) \in \text{children}(\llbracket x \rrbracket_G(\text{reflect}_s \llbracket x \rrbracket'_P(s, v)))$
- $\forall s \in \mathcal{S}, \forall v \in \mathcal{V}'$,
 $N(c, Del, k, -) \in \text{children } v \wedge \llbracket x \rrbracket_G(\text{reflect}_s \llbracket x \rrbracket'_P(s, v)) \neq \perp$
 $\Rightarrow N(c, -, k, -) \notin \text{children}(\llbracket x \rrbracket_G(\text{reflect}_s \llbracket x \rrbracket'_P(s, v)))$
- $\forall s \in \mathcal{S}, \forall v \in \mathcal{V}'$,
 $v = N(c, Mod\ c', -, -) \wedge \llbracket x \rrbracket_G(\text{reflect}_s \llbracket x \rrbracket'_P(s, v)) \neq \perp$
 $\Rightarrow v = N(c', -, -, -)$
- $x \hat{;} \text{extract } k$ が局所変更保存性を満たす. □

しかし, 前節の複数のビューのための小さな言語と違ってこれの解析を行うのは, 容易ではない. そのため, この性質に関しては動的に実際に変換を適用してみて検査するとして, 以後反射性について議論していく.

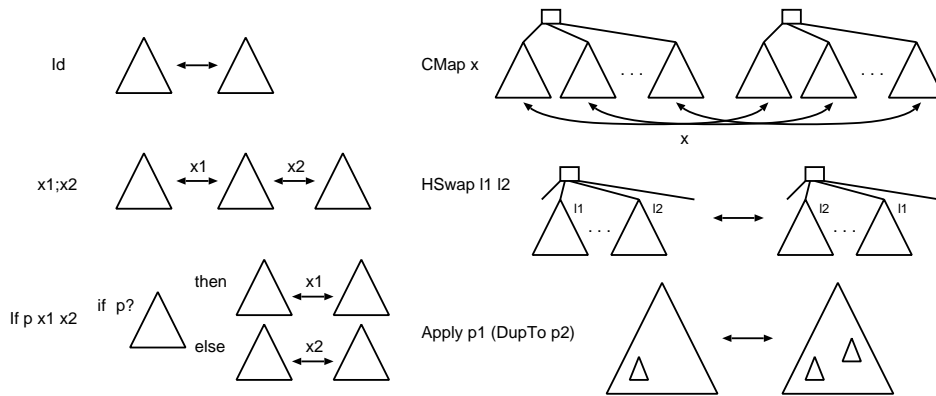


図 4.3. X^\pm で記述される変換の概観

4.2.4 X^\pm 言語の定義

拡張された X 言語である X^\pm を以下のように定義する .

$X ::=$	Apply $[K]$ X'	{ 特定のノードに適用 }
	$X \ ; \ X$	{ 合成 }
$X' ::=$	Id	{ 恒等変換 }
	Sort $(C \rightarrow S)$	{ ソート }
	CMap X'	{ 子すべてに対する適用 }
	Insert <i>Tree</i>	{ 仮想ノードの挿入 }
	If $(Tree \rightarrow Bool)$ $X' X'$	{ 条件分岐 }
	Hide K	{ 隠蔽 }
	HSwap $K K$	{ 並び換え }
	DupTo $[K]$	{ 複製 }

但し, $;$ は右結合であるとする . この言語の挙動を図 4.3 に示す . この言語を用いることにより, たとえば, p で指し示されるディレクトリ直下の, 最近作成されたファイルまたはディレクトリの複製を p' で指し示されるディレクトリに作成するという変換を

$$\text{Apply } p \ (\text{If } \text{isRecentCreated} \ (\text{DupTo } p') \ \text{Id})$$

のように記述することができる .

しかし, 今までの議論だけで, この言語で記述された変換を反射性を満たすものにするのは難しい . たとえば, 変換

$$\text{Apply } [] \ \text{CMap} \ (\text{CMap} \ (\text{DupTo } [])) \ ; \ \text{Apply } [] \ (\text{CMap} \ (\text{CMap} \ (\text{DupTo } [])))$$

などにおいて DupTo の複製先をすべて区別できるようにするのは容易ではない . そこで, $[x]_G$,

$\llbracket x \rrbracket'_P$ に代わる $\llbracket x \rrbracket_G, \llbracket x \rrbracket'_P$ を次のように定義する .

$$\begin{aligned} \llbracket x_1 \hat{\ ; } x_2 \rrbracket_G i s &= \llbracket x_2 \rrbracket_G (i+1) (\llbracket x_1 \rrbracket_G i s) \\ \llbracket x_1 \hat{\ ; } x_2 \rrbracket'_P i (s, v) &= \llbracket x_1 \rrbracket'_P i (s, \llbracket x_2 \rrbracket'_P (i+1) (\llbracket x_1 \rrbracket_G i s, v)) \\ \llbracket \text{Apply } p \ x' \rrbracket_G i s &= \llbracket x' \rrbracket_{\text{Glocal}} i p s \\ \llbracket \text{Apply } p \ x' \rrbracket'_P i (s, v) &= \llbracket x' \rrbracket'_{\text{Plocal}} i p (s, v) \end{aligned}$$

ここで , $\llbracket x \rrbracket_G, \llbracket x \rrbracket'_P$ はカウンタ付きの変換である . カウンタを導入することにより , 変換

$$\text{Apply } [] \text{ CMap (CMap (DupTo } [])) \hat{\ ; } \text{Apply } [] \text{ (CMap (CMap (DupTo } []))$$

における , それぞれの DupTo で挿入されたノードのキーを区別することができるようになる . このカウンタ付きの変換により同期を行うことを考える .

これについての反射性を以下のように定義する .

定義 4.13 (カウンタ付き双方向変換の反射性). カウンタを用いて定義される双方向変換 x が反射性を満たすとは ,

$$\forall i, s, \llbracket x_1 \rrbracket'_P i (s, \llbracket x_1 \rrbracket_G i s) \sqsubseteq s$$

であることをいう . □

このとき , 反射的な x_1, x_2 に対して ,

$$\begin{aligned} &\llbracket x_1 \hat{\ ; } x_2 \rrbracket'_P i (s, \llbracket x_1 \hat{\ ; } x_2 \rrbracket_G i s) \\ &= \llbracket x_1 \hat{\ ; } x_2 \rrbracket'_P i (s, \llbracket x_2 \rrbracket_G (i+1) (\llbracket x_1 \rrbracket_G i s)) \\ &= \llbracket x_1 \rrbracket'_P i (s, \llbracket x_2 \rrbracket'_P (i+1) (\llbracket x_1 \rrbracket_G i s, \llbracket x_2 \rrbracket_G (i+1) (\llbracket x_1 \rrbracket_G i s))) \\ &\sqsubseteq \{x_2 \text{ の反射性} \} \\ &\quad \llbracket x_1 \rrbracket'_P i (s, \llbracket x_1 \rrbracket_G i s) \\ &\sqsubseteq \{x_1 \text{ の反射性} \} \\ & s \end{aligned}$$

と $x_1 \hat{\ ; } x_2$ の反射性を確認できる . また , Apply について ,

$$\forall i, s, \llbracket x \rrbracket'_{\text{Plocal}} i p (s, \llbracket x \rrbracket_{\text{Glocal}} i p s)$$

が成り立てば , Apply の反射性が成り立つ . よって , 今後 $\llbracket x \rrbracket'_{\text{Plocal}}$ および $\llbracket x \rrbracket_{\text{Glocal}}$ の性質を議論していく .

4.2.5 X[±] 言語の意味と反射性

変換の意味を記述するため以下の有用な補助関数を用いる .

- $listify :: \{Tree\} \rightarrow [Tree]$: $listify\ ts$ は木の集合 ts をそれぞれの第4要素を比較することにより、木のリストにする。
- $setify :: [Tree] \rightarrow \{Tree\}$: $setify\ ts$ は木のリスト ts を木の集合にする。
- $label :: C \rightarrow S$: $label\ c$ はディレクトリ木からソース木を構成するために、ノードが含む要素 c からソートに使用する値を生成する。
- $key :: Tree \rightarrow K$: $key\ t$ は木 t からキーを取り出す。

$$key\ (N\ (c, m, k, s)\ f) = k$$

- $fetch :: [K] \rightarrow Tree \rightarrow Tree$: $fetch\ p\ t$ は木 t から、キーの列 p で表される部分木を取り出す。

$$\begin{aligned} fetch\ []\ t &= t \\ fetch\ (k : ks)\ (N\ d\ cs) &= \\ &\quad \text{if } N\ (c, m, k, s)\ cs' \in cs \text{ then } fetch\ ks\ (N\ (c, m, k, s)\ cs) \\ &\quad \text{else } \Omega \end{aligned}$$

- $replace :: [K] \rightarrow Tree \rightarrow Tree \rightarrow Tree$: $replace\ p\ t\ t'$ は木 t の、キーの列 p で表される部分木を t' に置き換えた木を返す。

$$\begin{aligned} replace\ []\ t\ t' &= t' \\ replace\ (k : ks)\ (N\ d\ cs) &= \\ &\quad \text{if } N\ (c, m, k, s)\ cs' \in cs \text{ then} \\ &\quad \quad N\ d\ (\{replace\ ks\ (N\ (c, m, k, s)\ cs')\ t'\} \cup (cs \setminus \{N\ (c, m, k, s)\ cs'})) \\ &\quad \text{else } \perp \end{aligned}$$

- $insert :: Tree \rightarrow [K] \rightarrow Tree \rightarrow Tree$: $insert\ t\ k\ t'$ は木 t の子に t' のキーを k に変えたものを付け加えた木を返す。

$$\begin{aligned} insert\ (N\ d\ cs)\ k\ t' &= N\ d\ (\{N\ (c, m, k, \Omega)\ cs'\} \cup cs) \\ \text{where } N\ (c, m, -, -)\ cs' &= t' \end{aligned}$$

- $remove :: [K] \rightarrow Tree \rightarrow Tree$: $remove\ p\ t$ は木 t のキーの列 p で表される部分木を取り除いた木を返す。

$$\begin{aligned} remove\ [k]\ (N\ d\ cs) &= \\ &\quad \text{if } N\ (c, m, k, s)\ cs' \in cs \text{ then } N\ d\ (cs \setminus \{N\ (c, m, k, s)\ cs'}) \\ &\quad \text{else } \perp \\ remove\ (k : k' : ks)\ (N\ d\ cs) &= \\ &\quad \text{if } N\ (c, m, k, s)\ cs' \in cs \text{ then} \\ &\quad \quad N\ d\ (\{remove\ (k' : ks)\ (N\ (c, m, k, s)\ cs')\} \cup (cs \setminus \{N\ (c, m, k, s)\ cs'})) \\ &\quad \text{else } \perp \end{aligned}$$

恒等変換

恒等変換 Id は、これまでに登場した恒等変換 id と同じように動作する。

$$\begin{aligned} \llbracket \text{Id} \rrbracket_{\text{Glocal}} i p s &= s \\ \llbracket \text{Id} \rrbracket'_{\text{Plocal}} i p (s, v) &= v \end{aligned}$$

反射性は自明である。

ソート

ソートはソートに使用される値を変更するだけである。関数 f は、要素の値からソートに使用される値を生成する関数である。

$$\begin{aligned} \llbracket \text{Sort } f \rrbracket_{\text{Glocal}} i p s &= \\ &\text{if } \text{fetch } p s = \Omega \text{ then } \perp \\ &\text{else } \text{replace } p s \ (N d \ \{N (c, m, k, f c) t \mid N (c, m, k, -) t \in cs\}) \\ &\text{where } (N d cs) = \text{fetch } p s \\ \llbracket \text{Sort } f \rrbracket'_{\text{Plocal}} i p (s, v) &= \\ &\text{if } \text{fetch } p v = \Omega \text{ then } \perp \\ &\text{else } \text{replace } p v \ (N d \ \{N (c, m, k, \text{label } c) t \mid N (c, m, k, -) t \in cs\}) \\ &\text{where } (N d cs) = \text{fetch } p v \end{aligned}$$

ファイルマネージャなどの木の表示が必要なアプリケーションは、この値を利用し木の子に順序を与えることで木の表示を行う。

ソースの等価性について要素のソートに使われる値は無視されるので、反射性は自明である。

子すべてへの適用

子すべてへの適用 $\text{CMap } x'$ は, 図 4.4 のように子のすべてに変換 x' を Apply する .

$$\begin{aligned} \llbracket \text{CMap } x' \rrbracket_{\text{Glocal}} i p s = & \\ \text{if } \text{fetch } p s = \Omega \text{ then } \perp & \\ \text{else } \text{getChildren } lchildren s & \\ \text{where } lchildren & = \text{listify } (\text{children } (\text{fetch } p s)) \\ \text{getChildren } (d : ds) s & = \text{getChildren } ds \ (\llbracket x' \rrbracket_{\text{Glocal}} i (p ++ [\text{key } d]) s) \\ \text{getChildren } [] s & = s \end{aligned}$$

$$\begin{aligned} \llbracket \text{CMap } x' \rrbracket'_{\text{Plocal}} i p s = & \\ \text{if } \text{fetch } p v = \Omega \text{ then } \perp \text{ else} & \\ \text{let} & \\ \text{putInserted } (d : ds) v = \text{if } \text{isAdded } d \text{ then} & \\ \text{putInserted } ds \ (\llbracket x' \rrbracket'_{\text{Plocal}} i p' (\text{remove } p' v, v)) & \\ \text{else } \text{putInserted } ds v & \\ \text{where } p' = p ++ [\text{key } d] & \\ \text{putInserted } [] v = v & \\ \text{in} & \\ \text{if } \text{fetch } p s = \Omega \text{ then} & \\ \text{putInserted } \text{listify } (\text{children } (\text{fetch } p v)) v & \\ \text{else} & \\ \text{putInserted } lchildren_v v' & \\ \text{where } lchildren_s = \text{listify } (\text{children } (\text{fetch } p s)) & \\ \text{putChildren } (d : ds) (s, v) = & \\ \llbracket x' \rrbracket'_{\text{Plocal}} i p' (s, \text{putChildren } ds \ (\llbracket x' \rrbracket_{\text{Glocal}} i p' s, v)) & \\ \text{where } p' = p ++ [\text{key } d] & \\ \text{putChildren } [] (s, v) = v & \\ v' = \text{putChildren } lchildren_s (s, v) & \\ lchildren_v = \text{listify } (\text{children } (\text{fetch } p v')) & \end{aligned}$$

但し, x' が $\text{DupTo } p'$ を含むとき p' は CMap の適用先である p より下の子を指してはならない . 変換 $\text{DupTo } p'$ の p' が p より下の子を指さない場合, DupTo が p の下を変更しないため, 子をどの順番で処理しても変換の結果が変わらない .

変換 x' が反射性を満たすときの $\text{CMap } x'$ の反射性は以下のように確認できる . 但し, $\text{fetch } p s = \Omega$ のときは,

$$\llbracket \text{CMap } x' \rrbracket'_{\text{Plocal}} i p (s, \llbracket \text{CMap } x \rrbracket_{\text{Glocal}} i p s) = \perp$$

となり反射性を満たすため, $\text{fetch } p s \neq \Omega$ の場合を考える . また, マークが付いていない場合

Apply p (CMap x) \simeq Apply $(p ++ k_1) x$; Apply $(p ++ k_2) x$

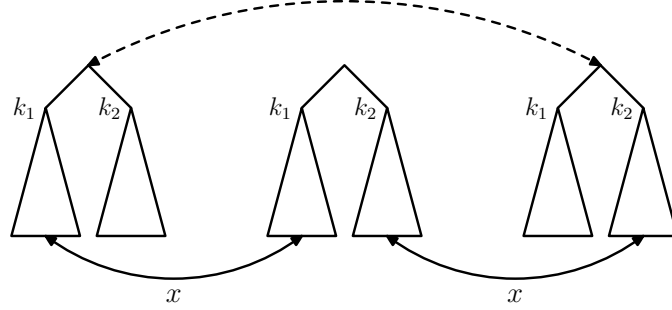


図 4.4. CMap の挙動

$putInsert\ l\ v = v$ であることを利用する .

$$\begin{aligned}
 & \llbracket \text{CMap } x' \rrbracket'_{\text{Plocal}}\ i\ p\ (s, \llbracket \text{CMap } x \rrbracket_{\text{Glocal}}\ i\ p\ s) \\
 &= \llbracket \text{CMap } x' \rrbracket'_{\text{Plocal}}\ i\ p\ (s, \text{getChildren } l\ s) \\
 & \quad \text{where } l = \text{listify } (\text{children } (\text{fetch } p\ s)) \\
 & \quad \text{getChildren } (d : ds)\ s = \text{getChildren } ds\ (\llbracket x' \rrbracket_{\text{Glocal}}\ i\ (p ++ [\text{key } d])\ s) \\
 & \quad \text{getChildren } []\ s = s \\
 &= \text{putChildren } l\ (s, \text{getChildren } l\ s) \\
 & \quad \text{where } l = \text{listify } (\text{children } (\text{fetch } p\ s)) \\
 & \quad \text{getChildren } (d : ds)\ s = \text{getChildren } ds\ (\llbracket x' \rrbracket_{\text{Glocal}}\ i\ (p ++ [\text{key } d])\ s) \\
 & \quad \text{getChildren } []\ s = s \\
 & \quad \text{putChildren } (d : ds)\ (s, v) = \\
 & \quad \quad \llbracket x' \rrbracket'_{\text{Plocal}}\ i\ p'\ (s, \text{putChildren } ds\ (\llbracket x' \rrbracket_{\text{Glocal}}\ i\ p'\ s, v)) \\
 & \quad \quad \quad \text{where } p' = p ++ [\text{key } d] \\
 & \quad \text{putChildren } []\ (s, v) = v
 \end{aligned}$$

このとき , $l = []$ について , $\text{putChildren } []\ (s, \text{getChildren } []\ s) = s$ となる . 次に , $l = ds$ のと

き $putChildren\ l\ (s, getChildren\ l\ s) \sqsubseteq s$ が成り立つと仮定する．すると，

$$\begin{aligned}
& putChildren\ (d : ds)\ (s, getChildren\ (d : ds)\ s) \\
&= \llbracket x' \rrbracket'_{Plocal}\ i\ p'\ (s, putChildren\ ds\ (\llbracket x' \rrbracket_{Glocal}\ i\ p'\ s, getChildren\ (d : ds)\ s)) \\
&\quad \mathbf{where}\ p' = p ++ [key\ d] \\
&= \llbracket x' \rrbracket'_{Plocal}\ i\ p'\ (s, putChildren\ ds\ (\llbracket x' \rrbracket_{Glocal}\ i\ p'\ s, getChildren\ ds\ (\llbracket x' \rrbracket_{Glocal}\ i\ p'\ s))) \\
&\quad \mathbf{where}\ p' = p ++ [key\ d] \\
&\sqsubseteq \{ \text{帰納法の仮定} \} \\
&\quad \llbracket x' \rrbracket'_{Plocal}\ i\ p'\ (s, \llbracket x' \rrbracket_{Glocal}\ i\ p'\ s) \\
&\quad \mathbf{where}\ p' = p ++ [key\ d] \\
&\sqsubseteq \{ x' \text{ の反射性} \} \\
&\quad s
\end{aligned}$$

のように， $l = d : ds$ のときも成り立つ．

よって帰納法により， $putChildren\ l\ (s, getChildren\ l\ s) \sqsubseteq s$ が示せた．

よって，

$$\llbracket CMap\ x' \rrbracket'_{Plocal}\ i\ p\ (s, \llbracket CMap\ x \rrbracket_{Glocal}\ i\ p\ s) \sqsubseteq s$$

が成り立つ．

$CMap\ x'$ されたノードに対し，ノードの挿入が許可されるのは， x' が $Sort$ ， Id および $DupTo$ であるときと，ノードの挿入が許可される変換 x に対して $CMap\ x'$ のときのみである．他は対応するソースがないと逆変換ができないため，ノードの挿入をソースへ伝播することができない．

挿入

挿入は仮想的なノードを挿入する．挿入されたノードに対する編集は許可されない．

$$\begin{aligned}
\llbracket Insert\ t \rrbracket_{Glocal}\ i\ p\ s &= \\
&\quad \mathbf{if}\ fetch\ p\ s = \Omega\ \mathbf{then}\ \perp \\
&\quad \mathbf{else}\ replace\ p\ s\ (insert\ (fetch\ p\ s)\ (i, p)\ t) \\
\llbracket Insert\ t \rrbracket_P\ i\ p\ (s, v) &= \\
&\quad \mathbf{if}\ fetch\ (p ++ [(i, p)])\ v = \Omega\ \mathbf{then}\ v \\
&\quad \mathbf{else\ if}\ containsAnyMark\ (fetch\ (p ++ [(i, p)])\ v)\ \mathbf{then}\ \perp \\
&\quad \mathbf{else}\ remove\ (p ++ [(i, p)])\ v
\end{aligned}$$

但し， t は一切のマークを含んでいない．また $containsAnyMark$ は木がマーク付きノードを持つかどうか調べる関数である．

反射性を満たすことは以下のようにして確認できる． $fetch\ p\ s = \Omega$ のとき，

$$\llbracket \text{Insert } t \rrbracket'_{\text{Plocal}}\ i\ p\ (s, \llbracket \text{Insert } t \rrbracket_{\text{Glocal}}\ i\ p\ s) = \perp$$

より反射性が成り立つ． $fetch\ p\ s \neq \Omega$ のとき

$$\begin{aligned} & \llbracket \text{Insert } t \rrbracket'_{\text{Plocal}}\ i\ p\ (s, \llbracket \text{Insert } t \rrbracket_{\text{Glocal}}\ i\ p\ s) \\ &= \llbracket \text{Insert } t \rrbracket'_{\text{Plocal}}\ i\ p\ (s, \text{replace } p\ s\ (\text{insert } (\text{fetch } p\ s)\ ((i, p))\ t)) \\ &= \text{remove } (p\ ++[(i, p)])\ (s, \text{replace } p\ s\ (\text{insert } (\text{fetch } p\ s)\ ((i, p))\ t)) \\ &= s \end{aligned}$$

とあり反射性が成り立つ．

新規に挿入されるノードのキーが，挿入先の集合の中に含まれていないことを証明するため，次の補題を証明する．

補題 4.14. 順方向変換において，任意 i, p について $\llbracket - \rrbracket_{\text{Glocal}}\ i\ p$ が 2 度呼ばれることはない．

証明. カウンタを用いているため個々の $\llbracket \text{Apply } x' p \rrbracket_{\text{G}}\ i$ には異なる i が渡される．よって， $\llbracket \text{Apply } x' p \rrbracket_{\text{G}}\ i = \llbracket x' \rrbracket_{\text{Glocal}}\ i\ p$ が任意の p について， $\llbracket - \rrbracket_{\text{Glocal}}\ i\ p$ を 2 度呼ばないことを示せばよい．

Id , Sort , Insert , Hide , HSwap , DupTo は $\llbracket - \rrbracket_{\text{Glocal}}\ i\ p$ を呼ばない．次に， $\text{If } x'_1\ x'_2$ は， x'_1 および x'_2 が任意の p について $\llbracket - \rrbracket_{\text{Glocal}}\ i\ p$ を 2 度呼ばない場合， $\text{If } x'_1\ x'_2$ が $\llbracket - \rrbracket_{\text{Glocal}}\ i\ p$ を 2 度呼ぶことがない．また， $\text{CMap } x'$ は x が任意の p について $\llbracket - \rrbracket_{\text{Glocal}}\ i\ p$ を 2 度呼ばない場合， $\llbracket x' \rrbracket_{\text{Glocal}}\ i\ p$ に渡す p はすべて異なり， $\llbracket x' \rrbracket_{\text{Glocal}}\ i\ p$ が p より上のノードを表す p' について $\llbracket - \rrbracket_{\text{Glocal}}\ i\ p'$ を呼ぶことがないため， $\text{CMap } x'$ は任意の p について $\llbracket - \rrbracket_{\text{Glocal}}\ i\ p$ を 2 度呼ばない．これより，帰納的に $\llbracket x' \rrbracket_{\text{Glocal}}\ i\ p$ が任意の p について $\llbracket - \rrbracket_{\text{Glocal}}\ i\ p$ を 2 度呼ばないことが示された．

よって順方向変換において，任意の p に対し $\llbracket - \rrbracket_{\text{Glocal}}\ i\ p$ が 2 度呼ばれることはない． \square

$\llbracket \text{DupTo} \rrbracket_{\text{Glocal}}\ i\ p$ および $\llbracket \text{Insert} \rrbracket_{\text{Glocal}}\ i\ p$ はキーが (i, p) であるノードを挿入するが，補題より順方向変換中で呼びだされる $\llbracket \text{DupTo} \rrbracket_{\text{Glocal}}\ i\ p$ および $\llbracket \text{Insert} \rrbracket_{\text{Glocal}}\ i\ p$ の i, p はすべて異なる．よって最初にソース木を作成するときに，0 以上 i に対し (i, p) というキーのノードを含まないようにしておけば， DupTo および Insert の挿入するノードのキーが他のノードのキーと衝突することはない．

条件分岐

条件分岐はソースを $pred$ で検査することにより, どの変換を使用するかを選択する.

$$\begin{aligned}
 & \llbracket \text{If } pred \ x'_1 \ x'_2 \rrbracket_{Glocal} \ i \ p \ s = \\
 & \quad \text{if } fetch \ p \ s = \Omega \ \text{then } \perp \\
 & \quad \text{else if } pred \ (fetch \ p \ s) \ \text{then} \\
 & \quad \quad \llbracket x'_1 \rrbracket_{Glocal} \ i \ p \ s \\
 & \quad \text{else} \\
 & \quad \quad \llbracket x'_2 \rrbracket_{Glocal} \ i \ p \ s \\
 & \llbracket \text{If } pred \ x'_1 \ x'_2 \rrbracket'_{Plocal} \ i \ p \ (s, v) = \\
 & \quad \text{if } fetch \ p \ s = \Omega \ \text{then } \perp \\
 & \quad \text{else if } pred \ (fetch \ p \ s) \ \text{then} \\
 & \quad \quad \llbracket x'_1 \rrbracket'_{Plocal} \ i \ p \ (s, v) \\
 & \quad \text{else} \\
 & \quad \quad \llbracket x'_2 \rrbracket'_{Plocal} \ i \ p \ (s, v)
 \end{aligned}$$

反射性は x'_1 と x'_2 が反射性を満たすとき, 次のようにして確認できる. まず, $fetch \ p \ s = \Omega$ のとき

$$\llbracket \text{If } pred \ x'_1 \ x'_2 \rrbracket'_{Plocal} \ i \ p \ (s, \llbracket \text{If } pred \ x'_1 \ x'_2 \rrbracket_{Glocal} \ i \ p \ s) = \perp$$

になるので, 反射性を満たす. 次に, $fetch \ p \ s \neq \Omega$ のときは,

$$\begin{aligned}
 & \llbracket \text{If } pred \ x'_1 \ x'_2 \rrbracket'_{Plocal} \ i \ p \ (s, \llbracket \text{If } pred \ x'_1 \ x'_2 \rrbracket_{Glocal} \ i \ p \ s) = \\
 & = \llbracket \text{If } pred \ x'_1 \ x'_2 \rrbracket'_{Plocal} \ i \ p \ \left(\begin{array}{l} \text{if } pred \ (fetch \ p \ s) \ \text{then} \\ \quad \llbracket x'_1 \rrbracket_{Glocal} \ i \ p \ s \\ \text{else} \\ \quad \llbracket x'_2 \rrbracket_{Glocal} \ i \ p \ s \end{array} \right) \\
 & = \text{if } pred \ (fetch \ p \ s) \ \text{then} \\
 & \quad \llbracket x'_1 \rrbracket'_{Plocal} \ i \ p \ (s, \llbracket x'_1 \rrbracket'_{Glocal} \ i \ p \ s) \\
 & \text{else} \\
 & \quad \llbracket x'_2 \rrbracket'_{Plocal} \ i \ p \ (s, \llbracket x'_2 \rrbracket'_{Glocal} \ i \ p \ s) \\
 & \sqsubseteq \{ x'_1 \ \text{と} \ x'_2 \ \text{の反射性} \} \\
 & \quad \text{if } pred \ (fetch \ p \ s) \ \text{then} \\
 & \quad \quad s \\
 & \quad \text{else} \\
 & \quad \quad s \\
 & = s
 \end{aligned}$$

のように確認できる.

複製を含む双方向変換において変更保存性が成り立たないため、変換 $x'_1 \hat{;} x'_2$ について、

$$\llbracket x'_2 \rrbracket'_{\text{Plocal}} (i+1) p (\llbracket x'_1 \rrbracket_{\text{Glocal}} i p s, v)$$

の値と、

$$\llbracket x'_1 \rrbracket_{\text{Glocal}} i p (\llbracket x'_1 \rrbracket'_{\text{Plocal}} i p (s, \llbracket x'_2 \rrbracket'_{\text{Plocal}} (i+1) p (\llbracket x'_1 \rrbracket_{\text{Glocal}} i p s, v)))$$

の値は等しいとは限らない。それゆえ、 $pred$ の真偽が同期後で変更されてしまい、適用される変換が変更される場合がある。そのため、第2章で行ったような条件分岐の変換だけを見て、局所変更保存性などの性質の解析を行うことは難しい。

隠蔽

隠蔽はキーが K であるノードを消去する。

$$\begin{aligned} \llbracket \text{Hide } k \rrbracket_{\text{Glocal}} i p s &= \\ &\text{if } fetch\ p\ s = \Omega \text{ then } \perp \\ &\text{else } remove\ (p\ ++[k])\ s \\ \llbracket \text{Hide } k \rrbracket'_{\text{Plocal}} i p (s, v) &= \\ &\text{if } fetch\ p\ s = \Omega \text{ then } \perp \\ &\text{else } replace\ p\ v\ (Nd\ (\{fetch\ (p\ ++[k])\ s\} \cup cs)) \\ &\text{where } Nd\ cs = fetch\ p\ v \end{aligned}$$

反射性は以下のように確認できる。まず、 $fetch\ p\ s = \Omega$ のとき、

$$\llbracket \text{Hide } k \rrbracket'_{\text{Plocal}} i p (s, \llbracket \text{Hide } k \rrbracket_{\text{Glocal}} i p s) = \perp$$

より反射性を満たす。次に、 $fetch\ p\ s \neq \Omega$ のとき、

$$\begin{aligned} &\llbracket \text{Hide } k \rrbracket'_{\text{Plocal}} i p (s, \llbracket \text{Hide } k \rrbracket_{\text{Glocal}} i p s) \\ &= \llbracket \text{Hide } k \rrbracket'_{\text{Plocal}} i p (s, remove\ (p\ ++[k])\ s) \\ &= replace\ p\ v\ (Nd\ (\{fetch\ (p\ ++[k])\ s\} \cup cs)) \\ &\quad \text{where } Nd\ cs = fetch\ p\ v \\ &\quad v = remove\ (p\ ++[k])\ s \\ &= s \end{aligned}$$

により反射性を満たす。

並び変え

並べ替えは，部分木の k_1 と k_2 の並び順を入れかえる．つまり，ソートに使われる値のみを入れかえることにより実現できる．

$$\begin{aligned} \llbracket \text{HSwap } k_1 \ k_2 \rrbracket_{\text{Glocal}} i \ p \ s = \\ \text{if } \text{fetch } (p \ ++[k_1]) \ s = \Omega \vee \text{fetch } (p \ ++[k_2]) \ s = \Omega \ \text{then } \perp \\ \text{else } \text{replace } (p \ ++[k_2]) \ (\text{replace } (p \ ++[k_1]) \ s \ (N \ (c, m, k, s') \ t)) \ (N \ (c', m', k', s) \ t') \\ \text{where } N \ (c, m, k, s) \ t \quad = \ \text{fetch } (p \ ++[k_1]) \ s \\ N \ (c', m', k', s') \ t' \quad = \ \text{fetch } (p \ ++[k_2]) \ s \\ \llbracket \text{HSwap } k_1 \ k_2 \rrbracket'_{\text{Plocal}} i \ p \ (s, v) = \\ \llbracket \text{HSwap } k_1 \ k_2 \rrbracket_{\text{Glocal}} i \ p \ v \end{aligned}$$

$\text{fetch } (p \ ++[k_1]) \ s = \Omega \vee \text{fetch } (p \ ++[k_2]) \ s \ \text{then } = \Omega \perp$ のときは，

$$\llbracket \text{HSwap } k_1 \ k_2 \rrbracket'_{\text{Plocal}} i \ p \ (s, \llbracket \text{HSwap } k_1 \ k_2 \rrbracket_{\text{Glocal}} i \ p \ s) = \perp$$

より反射性を満たす．よって，それ以外の場合を考える．このとき， $\llbracket \text{HSwap } k_1 \ k_2 \rrbracket_{\text{G}} i \ p$ の，パス $p \ ++[k_1]$ と $p \ ++[k_2]$ で指し示されるノードのソートで使用される値のみを入れ替えるという意味から考えて，この変換は反射性を満たす．

複製

複製変換 $\text{DupTo } p$ は p で指し示されるノードに対し，適用先のノードを複製する．ここで， p は大域的なパスである．

$$\begin{aligned} \llbracket \text{DupTo } p' \rrbracket_{\text{Glocal}} i \ p \ s = \\ \text{if } \text{fetch } p \ s = \Omega \vee \text{fetch } p' \ s = \Omega \ \text{then } \perp \\ \text{else } \text{replace } p' \ (\text{insert } (\text{fetch } p' \ s) \ (i, p) \ (\text{fetch } p \ s)) \\ \llbracket \text{DupTo } p' \rrbracket'_{\text{Plocal}} i \ p \ s = \\ \text{if } \text{fetch } p \ v = \Omega \ \text{then } \perp \\ \text{else if } \text{fetch } (p' \ ++[(i, p)]) \ v = \Omega \ \text{then } v \\ \text{else } \text{replace } p \ v' \ (\text{merge } d \ d') \\ \text{where } d \quad = \ \text{fetch } (p' \ ++[(i, p)]) \ v \\ v' \quad = \ \text{remove } (p' \ ++[(i, p)]) \ v \\ d' \quad = \ \text{fetch } p \ v' \end{aligned}$$

ここで関数 $\text{merge } d \ d'$ は， d および d' に対するマーク付けをマージする関数である．この関数により，一方に対して行われた編集は他方にも行われる．関数 merge は以下のように定義さ

れる .

```

merge d d' =
  if      isDeleted d                then N (c, m, k', s) (mergeC t t')
  else if isDeleted d'              then N (c', m', k', s') (mergeC t t')
  else if isModified d ∧ isNormal d' then N (c, m, k', s) (mergeC t t')
  else if isNormal d ∧ isModified d' then N (c', m', k', s') (mergeC t t')
  else if isModified d ∧ isModified d' ∧ m = m' then N (c, m, k', s) (mergeC t t')
  else if isNormal d ∧ isNormal d'   then N (c, m, k', s) (mergeC t t')
  else ⊥
  where N (c, m, k, s) t = d
        N (c', m', k', s') t' = d'

mergeC t t' =
  setify (mergeCL l l')
  where l = listifyBy label t
        l' = listifyBy label t'

mergeCL (d : ds) (d' : ds') =
  if      isAdded d ∧ isAdded d' ∧ d = d' then d : mergeCL ds ds'
  else if isAdded d ∧ isAdded d' ∧ d ≠ d' then d : d' : mergeCL ds ds'
  else if isAdded d                        then d : mergeCL ds (d' : ds')
  else if isAdded d'                       then d' : mergeCL (d : ds) ds'
  else                                     (merge d d') : mergeCL ds ds'
  where N (c, m, k, s) t = d
        N (c', m', k', s') t' = d'

```

但し , $listifyBy f$ は , 子のソートに使われる値を f により定め直したのちに , $listify$ する . 定義より , $merge d d = d$ となる .

反射性について , $fetch p s = \Omega \vee fetch p s' = \Omega$ のときは ,

$$\llbracket DupTo p' \rrbracket'_{P_{local}} i p (s, \llbracket DupTo p \rrbracket_{G_{local}} i p s) = \perp$$

となるため、それ以外の場合を考える．すると、

$$\begin{aligned}
& \llbracket \text{DupTo } p' \rrbracket'_{\text{Plocal}} i p (s, \llbracket \text{DupTo } p \rrbracket_{\text{Glocal}} i p s) \\
&= \text{replace } p v' (\text{merge } d d') \\
& \quad \mathbf{where} \ v = \text{replace } p' (\text{insert } (\text{fetch } p' s) (i, p) (\text{fetch } p s)) \\
& \quad \quad d = \text{fetch } (p' ++ [(i, p)]) v \\
& \quad \quad v' = \text{remove } (p' ++ [(i, p)]) v \\
& \quad \quad d' = \text{fetch } p v' \\
&= \text{replace } p v' (\text{merge } d d') \\
& \quad \mathbf{where} \ v = \text{replace } p' (\text{insert } (\text{fetch } p' s) (i, p) (\text{fetch } p s)) \\
& \quad \quad d = \text{fetch } p s \\
& \quad \quad v' = s \\
& \quad \quad d' = \text{fetch } p s \\
&= \text{replace } p s (\text{merge } (\text{fetch } p s) (\text{fetch } p s)) \\
&= \text{replace } p s (\text{fetch } p s) \\
&= s
\end{aligned}$$

となるため、反射性を確認できる．

4.2.6 局所変更保存性に関する議論

条件分岐を除いたすべての変換の組み合わせは局所変更保存性を満たす．なぜなら変換の結果が \perp にならない限り、すべての変換はビューで得られる木のマークを消すことがないからである．

複製変換 DupTo に関してこれは自明ではないが、以下の性質が成り立つため簡単に示すことができる．

定理 4.15. 木 d がマークを含まず、 d に編集操作を加えマークを加えることにより得られた木を d' とする．このとき

$$d' = \text{merge } d d'$$

となる．

証明. 関数 merge と mergeCL の性質より自明． □

第5章 双方向変換の応用：ビューを持つ双方向変換を利用したファイルマネージャの実現

前章の双方向変換システムの応用例として著者が中心となって作成した双方向変換を利用したファイルマネージャである「梅林」^[?]の紹介を行う。

5.1 「梅林」：ビューを持つ双方向変換を利用したファイルマネージャ

ファイルの操作において、ショートカット、シンボリックリンクといったファイル別名の存在は、しばしばユーザを混乱させる。初心者にとってファイル実体とそれに対する別名の区別は難しいため、メールへの添付やメディアへのバックアップの際にファイル実体でなく別名を用いてしまうことがある。また、別名と実体の関係は非対称であり、ファイル別名を消してもファイル実体は削除されず、逆にファイル実体を消すと無効なファイル別名が残る。アプリケーションのアンインストールを行う際に、ファイル別名のみを削除し、ファイル実体を削除しないという失敗は少なくない。実際に Microsoft Windows のファイルマネージャであるエクスプローラでは、このような失敗を防ぐため、デスクトップなどに存在するファイル別名を消去する際に、ファイル実体が削除されない旨の確認ダイアログが出る。また、一般にユーザにとって、あるファイル実体を参照しているすべてのファイル別名を列挙するのは容易ではないため、ファイル実体を消去した後に、存在しない実体を指し示している無効なファイル別名が残ったままとなりやすい。

そこで、本章ではファイルへの参照すべてを対称かつ統一的に抽象化して扱うことのできるファイルマネージャ「梅林」を、木上の双方向変換^[?]の技術を用いて実現した。既存のファイルマネージャはファイル実体と別名、つまりファイル名であるファイル参照とファイル参照への参照を用いる。これに対し、ファイルマネージャ「梅林」では、あるファイル参照を双方向変換により複数のファイル参照に変換する方法を用いる。この複数のファイル参照は互いに同期され、1つに加えられた変更は別の同期されたファイル参照に反映される。たとえば、あるファイル参照の名前を変更すると同期されたファイル参照の名前も変更され、あるファイル参照の指す実体を削除すると同期されたファイル参照はすべて削除される。ファイル実体を削除せずに同期されたファイル参照の1つを削除したい場合には、双方向変換によって同期されたファイル参照を「見えなく」することにより、ファイル別名の削除に対応する操作ができる。このように同期されたファイル参照は互いに対称に扱われ、無効なファイル別名を生じない。

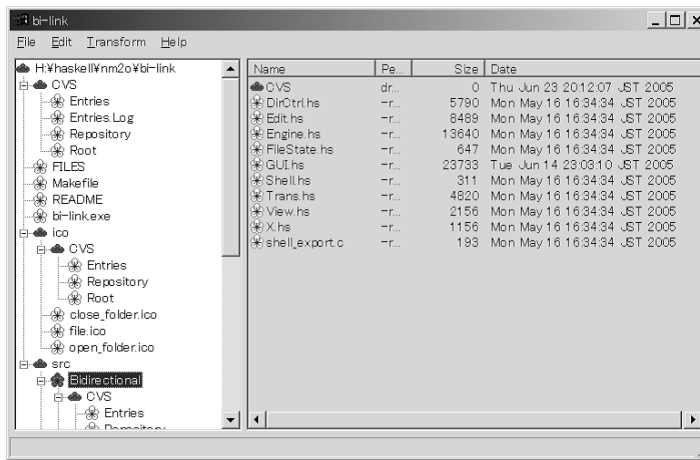


図 5.1. 「梅林」のスクリーンショット

また、既存のファイルマネージャは実際のディレクトリ木の見せ方に関する自由度が低い。ファイルに対する注釈、表示するファイルの順序、特定のファイルの隠蔽などいくつかの基本的な機能は提供されているが、ユーザはそれらを細かにカスタマイズすることはできない。「梅林」では、このような“見せ方”を双方向変換として記述する。そのため、これらの機能を統一的に表現でき、なおかつ見せ方の自由度を高めている。

「梅林」の特長は以下のようなものである。

見せ方の統一的な抽象化 ファイルマネージャは、ファイルへの複数の同期された参照、ソート、お気に入りリストなどの様々な機能を持つ。「梅林」はそのような表示に関する様々な機能を双方向変換を利用して統一的に扱うことができる。

ファイル実体の操作と見せ方の操作 「梅林」では新規作成、削除、名前変更などのディレクトリ木に関する操作とソート順の変更、同期されるファイル参照の追加などの見せ方に対する操作を提供している。これらの操作は GUI を通して対話的に行うことができる。

拡張性 「梅林」では、ファイルへの複数の同期された参照やソートなどの機能を双方向変換を利用して実現している。新たな見せ方に関する機能も、ユーザが双方向変換を用いて記述することにより、ファイル参照の同期関係を保ったまま追加することができる。

5.2 概要

図 5.1 が双方向変換ファイルマネージャ「梅林 (bi-link)」のスクリーンショットである。本節ではこの「梅林」の特徴的な機能および活用例について説明する。

表 5.1. 梅林の提供する操作

ディレクトリ木に対する操作	
Delete	ファイルおよびディレクトリの削除を行う
New File	新規ファイルを作成する
New Directory	新規ディレクトリを作成する
Rename	ファイルおよびディレクトリの名前を変更する

ビューに対する操作	
Hide	指定されたノードを表示されないようにする
Duplicate	複製先を指定し、指定されたノードが2箇所に表示されるようにする
Sort by Name	指定されたディレクトリの内容を名前ですべてソートして表示する
Sort by Size	指定されたディレクトリの内容を容量ですべてソートして表示する
Sort by Time	指定されたディレクトリの内容を更新日時ですべてソートして表示する

5.2.1 機能説明

「梅林」では、ディスク上のディレクトリ木に対し、ある変換によって見せ方を変更したものがビューとして表示され、ユーザはこのビューを通して様々な操作を対話的に実行することができる。ビュー上で実行できる操作には、ディレクトリ木そのものを変更する操作と、ビューのみを変更する操作とがある。ビューのみを変更する操作では、ディレクトリ木の見せ方のみが変更され、ディレクトリ木自体は変更されない。これらの操作によるディレクトリ木の変更や見せ方の変更を元に、新たなビューが生成される。

「梅林」の提供する操作を 5.1 に示す。提供される操作の内、特徴的なのは Duplicate である。Duplicate は同期された2つのファイル参照を作る機能である。Duplicate により同期された複数のファイル参照に対し、一方に加えたディレクトリ木に対する操作は必ずもう一方にも反映される。また、ディレクトリ木に対する操作とビューに対する操作とが異なる例として Hide と Delete が挙げられる。ビューのみに対する操作である Hide では指定されたファイル参照またはディレクトリ参照がビュー上で見えなくなるだけで実体は残り、ディレクトリ木に関する操作である Delete と異なる。

5.2.2 活用例

「梅林」では双方向変換を用いることにより、従来のファイルマネージャの持つ問題を解決できるだけでなく、様々な機能を統一的に扱うことができる。

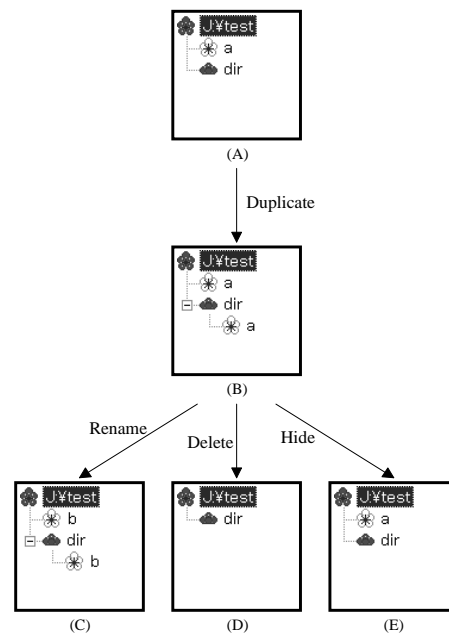


図 5.2. Duplicate により同期された参照

同期されたファイルの追加

Duplicate により同期された 2 つのファイル参照は、ファイル別名と同様の機能を実現でき、また 2 つは同等に扱われる。そのため、実体と別名の取り違いや無効な参照といった煩わしい問題に悩まされることがなくなる。

この様子を図 5.2 に示す。ビュー (A) においてファイル a をディレクトリ dir 以下に、Duplicate を用いて同期されたファイル参照を追加することでビュー (B) が得られる。ビュー上で 2 つの a は同期されており、どちらの a に対する操作も両方に反映される。従って、たとえば Rename により一方の a の名前を b に変更に変更すると、両方の名前が b に変更され (ビュー (C))、Delete により一方の a を削除すると両方の a が削除される (ビュー (D))。一方のみをビュー上から削除するには、削除するノードに Hide を適用すればよい (ビュー (E))。

お気に入りリスト

別名機能は特定のファイルやディレクトリへの簡単なアクセスを可能にする。従って、お気に入りのファイルや使用頻度の高いディレクトリなど、頻繁にアクセスする箇所へのファイルやディレクトリのファイル別名を集めたお気に入りリストは非常に便利である。

「梅林」でお気に入りリストのような機能を実現するには、ホームディレクトリなどのアクセスしやすい場所にディレクトリを作成し、ここに対象ファイルの複製を集めるだけで良い。分類し整理されたリンク先の構造を変えずに好きなファイルを同じ場所に集めることがで

きるだけでなく、複製元のディレクトリ木に対する操作がお気に入りリストの方にも同期される点で従来のファイルマネージャより優れている。

表示のカスタマイズ

「梅林」では従来のファイルマネージャよりも自由度の高い表示のカスタマイズが可能である。たとえば、ビューに対する操作は同期された複数のファイル参照に対してもそれぞれ個別に適用することができるため、あるディレクトリに Duplicate を適用し、一方に Sort by Name を、もう一方には Sort by Time を適用した場合、同じディレクトリの内容が名前ですべてソートされた様子と更新日時ですべてソートされた様子を両方同時に確認することができる。

様々な分類によるファイルの整理

ここまでの活用例の具体的な応用が様々な分類の仕方によるファイルの整理である。たとえば音楽ファイルの整理のように、ジャンル、アーティスト等、様々な要素に基づいてファイルを分類したいことはしばしばある。しかし、木構造を利用した階層構造に基づく整理ではそれらのうち1つの要素によってしか分類を行うことができない。ある分類によってディレクトリ分けしたものと別の分類によってディレクトリ分けしたものの2つのコピーをつくってしまうと、ファイルの名前変更、削除によって生じる両者の内容のくい違いに悩まされることになる。

Duplicate を用いることにより、このような問題も簡単に解決できる。音楽ファイルをジャンル別やアーティスト別などで仕分けされたディレクトリの下に同期されたファイル参照を用意しておけば、削除により、コピーやファイル別名が残ることがない。

5.3 実装

本節では「梅林」の実装の詳細を示す。なお、「梅林」の実装は、関数型言語 Haskell によって行った。

5.3.1 概観

「梅林」では扱うべきデータがディレクトリ木であり、その状態を直接操作することができない。そのため、システムコールを通して、ディレクトリ木と同じ構成の木をソースとして作成する。また、変換の記述には第4章で議論された X^\pm 言語を用いた。この言語は子が集合である木を扱っているものの、適宜ソートを行うことにより子がリストである木を扱うことができるようになる。

「梅林」の構成を図5.3に示す。システムコールによりソースが構築され、そこから順方向変換により得られたビューがGUIコンポーネントに表示される。ユーザがビュー上で編集を加えると、逆方向変換によりソースに対する編集に翻訳され、システムコールを用いてディレクト

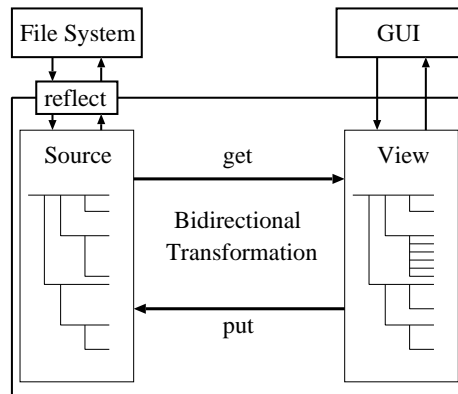


図 5.3. 「梅林」の構成

リ木に反映される。その後、新たにソースが再構築され順方向変換により変換し表示する。“見せ方”が変更された場合は、その新たな変換を用いて、順方向変換によりビューを再構築する。

例として、同期関係にある名前 b のノードに対し、 c に改名する編集を行った場合の挙動を図 5.4 に示す。ここで、図中の b の下に添字されている $\{\text{Mod } c\}$ はマークを表している。

5.3.2 データ構造の詳細

木

「梅林」で用いたソースおよびビューの木は、以下のように定義される。

```

type Source = Tree
type View   = Tree
data Tree = Tree D [Tree]
type D = ( ViewState, Label, Mark )
data ViewState = Concrete FileState
               | Abstract String
data FileState =
  {
    parentpath  :: String,
    filename    :: String,
    filetype    :: FileType,
    size        :: Integer,
    permissions :: Permissions,
    modifiedtime :: ClockTime
  }

```

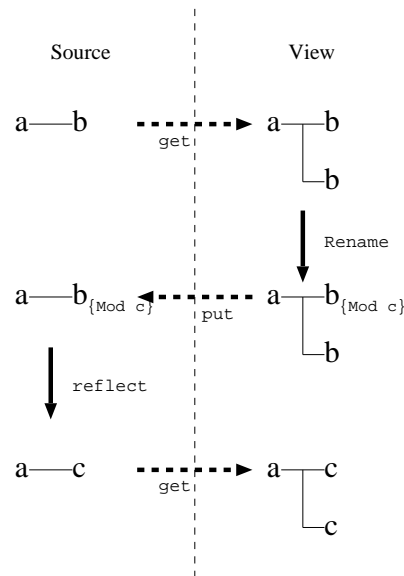


図 5.4. 改名編集の例

表 5.2. 各マークに対応した処理

Add	そのファイル/ディレクトリを新規作成する
Del	そのファイル/ディレクトリを削除する
Mod a	そのファイル/ディレクトリの名前を a に変更する
Open	そのディレクトリ以下のソースをディレクトリ木から再構築する

```
data FileType = File | Dir
```

Mark はマーク情報を表す。簡便のため、マーク情報がノードに含まれている。これにより、マーク付きの木とそうでない木を同一のデータ構造で扱うことができる。FileState はファイルまたはディレクトリの様々な状態を表す。parentpath は親へのパス情報を表し、filename は名前である。size, permissions, modifiedtime はサイズ、パーミッション、更新日時を表す。ここで Label は、 X^\pm 言語における子集合の中のキーに相当する。

ラベル

「梅林」では各ノードに対して以下のようにラベルを定義している。

```
data Label = LName String
           | LAbstract (Int, [Label])
           | LOther
```

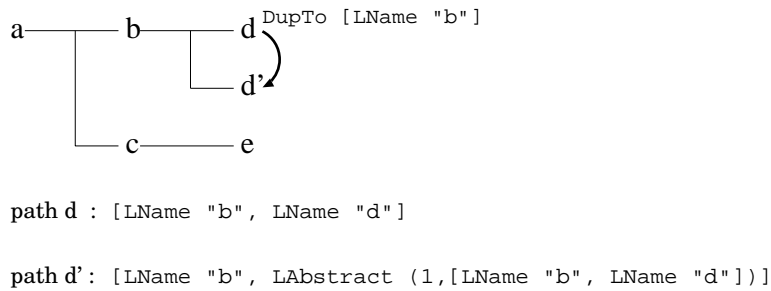


図 5.5. パスの例

LName は実際にディレクトリ木上に存在するノードのラベルであり、ファイル名と同一の文字列が格納される。LAbstract は抽象ノードを表す。抽象ノードは Insert または DupTo により挿入されるノードである。LOther は、編集により挿入される予定のノードに付けられる。挿入されるノードが他のノードと同じラベルを持つこと防ぐためである。

ディレクトリ木において、あるノードの子の名前はすべて異なるため、LName は兄弟間ですべて異なる。また、LOther は決して参照されることがないため、その存在は、あるパスがノードを一意に指し示すことに関して影響を与えない。よって、ここで定義されたラベルを用いて一意識別可能なパスを定義することができる。

「梅林」における具体的なパスの例を図 5.5 に示す。図 5.5 中において d' は d を複製してできたノードである。

5.3.3 reflect の実装

「梅林」では編集操作を行う際、ビュー木の各ノードに編集操作反映のためのマークを付け、それを逆方向変換によりソース木まで伝播させる。マークの実装を以下に示す。

```

type Mark = Maybe NotableMark

data NotableMark = Add
                  | Del
                  | Mod String
                  | Open

```

表 5.2 はマークされたノードに対し、ディレクトリ木上で行われる処理である。Open は、ディレクトリ木からソースを構築する際に使用される。

5.3.4 双方向変換の実装上の問題とその解決法

対象ノードを指定し双方向変換を適用する場合、ノードの指定に起因する問題がある。その問題と、解決法を示す。

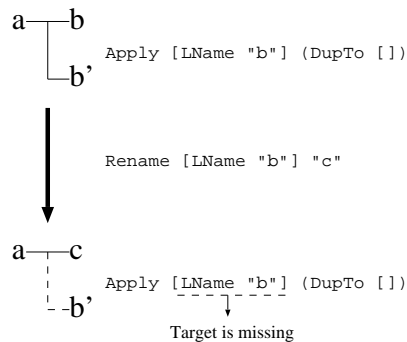


図 5.6. 改名により変換が適用されなくなる例

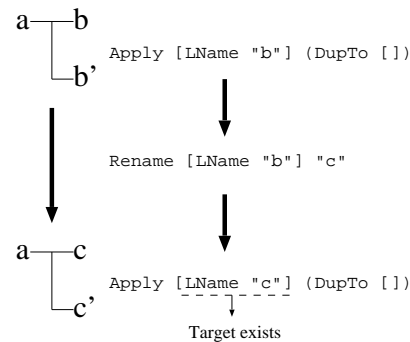


図 5.7. 改名を正しく反映する例

再生成時に生じる問題

ある名前のノードが削除された後に、削除されたものと同じ名前のノードが同じ位置に作成されると、パスが同一になってしまう。このため、削除前にそのノードに対して変換が適用されていた場合、その変換が再度適用されてしまう。しかし、これらのノードは別のものであるため、この挙動は望ましくない。

この問題は、順方向変換時に適用先ノードの存在しない変換を Id に置換することにより解決できる。ノードが削除された時にそのノードに関する変換は存在しなくなるため、再生成されたノードに対して変換が適用されることはなくなる。

5.4 議論

前節まで、双方向変換を利用したファイルマネージャ「梅林」の実現方法に関して説明してきた。本節では、関連研究との比較を通じて、本研究の貢献および残された課題について議論を行う。

関連システム

ここでは、ファイルマネージャ「梅林」と関連の深いシステムについて述べる。

Proxima[?] は汎用エディタであり、「梅林」と同様にビューを持つ。Proxima では両方向の変換をユーザが定義しなければならなかったが、「梅林」は、変換を双方向変換言語で記述することにより、この問題を生じない。

また、BTRON[?] には実身/仮身というファイルシステムがある。仮身はファイル参照に対応する概念であり、実身は必ず仮身を通して参照され、その中に任意の仮身を含むことができ

表 5.3. 双方向変換を利用したシステムと逆ポインタを利用したシステムとの比較

	双方向変換を利用したシステム	逆ポインタを利用したシステム
循環構造	現在では扱えない	扱える
同期関係の保持	変換	ファイルやディレクトリなど
拡張	双方向変換言語で書けば、同期関係を保てる	記述言語は問わないが、同期関係は拡張する側で管理

る。つまり、ファイルシステムそのものがグラフ構造になっている。「梅林」では、循環を含むようなグラフ構造を現在においては扱えないが、非循環有向グラフ構造を、双方向変換を通して扱うことにより、構造化して扱うことができる。

シンボリックリンクを利用した場合、自分の親へリンクを張ることにより、循環構造を作成できる。DupTo を利用した場合は循環構造は不動点として表現されることになるが、双方向変換において、このような不動点の扱いはまだよくわかっていない。しかし、我々は循環構造がファイル管理において有用であると考えないため、これは問題とはならない。

シンボリックリンクの際に、逆ポインタを持たせることでも、別名の非対称性を解決することができる。そこで、双方向変換を利用したシステムと逆ポインタを持つシステムとの比較を行う。主な相違点を、表 5.3 に示す。

逆ポインタを持つシステムでは、個々のファイルやディレクトリといったオブジェクト単位で同期関係の管理を行う。それに対し、双方向変換を利用したシステムは、同期関係を変換がすべて保持し管理を行う。これにより、双方向変換を利用したシステムでは同一のディレクトリ木に対し、異なった変換を適用することにより複数のビューを作成することが可能になる。ユーザが個々の目的に応じたビューを作成できることは、ファイルマネージャとして有用であると考えられる。たとえば、双方向変換を利用したファイルマネージャでは、好みの演奏家を集めたものや好みの年代を集めたものなど、目的別のビューを作成することができ、音楽ファイルを複数人で共有している場合にも、共有している人それぞれに対し固有のビューを作成することができる。

また、逆ポインタを用いたシステムでは、新たな機能や変換を任意の言語で記述できるものの、機能や変換の追加の際に同期関係を適切に管理する必要がある。それに対し、双方向変換を利用したシステムでは、新たな機能や変換を、双方向変換言語により記述することにより、同期関係を保ったまま追加することができる。

編集に関する整理

ファイルマネージャにおいて、ファイルの新規作成や名前変更など編集操作は重要な役割を果たす。DupTo のような複製変換での矛盾を防ぐため、[?] では限られた編集操作のみを考えその上で編集の伝播を考えた。

我々も限られた編集操作に対する伝播を考えてきたが、それは、ビューに対する編集操作をソースに対する編集操作へ翻訳するためである。このような考えかたは、ビュー更新問題へのアプローチとしては自然なものである [?, ?, ?, ?]。第 4 章では、特定の言語を定め、その上で双方向変換を利用した上での編集操作の翻訳に関する整理を行った。

我々は、GUIで対話的にビューを操作することを対象としたため、許される編集操作を自然に限定することができた。それにより、矛盾するような編集操作が同時に起こることを防ぎ、編集操作の伝播の議論を単純化することができた。しかし、複数のファイルマネージャを同時に使用する場合にはこのような状況が仮定できず、互いに矛盾する編集操作が同時に発生し得る。このような場合には、様々な分散ファイルシステムやバージョン管理システム [?, ?] のように、編集のビューへ伝播の際に衝突を解消するポリシーを定める必要がある。

一貫性

文献 [?] では、抽象ノードのラベルとして、挿入される先の子の中での抽象ノードの数を利用していった。しかし、この手法では、CMapなどで子の順番が変換した場合に、抽象ノードのラベルが変換してしまう。このような変換の適用先に対する一貫性を保つことは重要である。しかし、今回は X^\pm 言語においてカウンタを導入したことにより、この問題を解決した。

第6章 まとめ

6.1 結論

複製を含む双方向変換の満たすべき性質として、局所変更保存性を提案し、それに対し実際に小さな言語を定め解析を行った。

また、双方向変換をそれにアプリケーションに特化した定式化を行い、それにより双方向変換を利用したファイルマネージャを実現した。ここでは、複数のファイル別名は複製変換で表現されるため対称であり一方に対する変更は必ずもう一方に反映される。双方向変換を用いることにより、ソート、お気に入りリストなどの機能を統一的に扱うことができ、ユーザはそれらを含んだディレクトリ木のビューを自由に構成することができる。

6.2 今後の課題

6.2.1 双方向変換の理論

極小性

ビューが変更された場合に、ビュー中で変更場所以外で変更される場所は少ないほうが望ましい。変更保存性は、ビューが変更された場合に、変更の書き戻されたソースのビューにおける変更される場所が極小であること表現している。これに対し、局所保存性はこのような極小性に対し何も述べていない。

複製を含む双方向変換における極小性の定式化が求められる。

条件分岐

条件分岐 $\text{cond}_s \text{ pred } t_1 t_2$ について、 $\llbracket t_1 \rrbracket_P$ の値域で pred が真であり、 $\llbracket t_2 \rrbracket_P$ の値域で pred が偽であることを仮定した。しかし、たとえば、 mss は順方向時に最も和が大きくなる部分列を

返す変換であり，

$$\begin{aligned}
\llbracket mss \rrbracket_G (s : ss) &= \\
&\mathbf{if} \ s + \mathit{sum} \ z \geq \mathit{sum} \ y \ \mathbf{then} \ s : z \\
&\mathbf{else} \ y \\
&\quad \mathbf{where} \ (y, z) = (\llbracket mss \rrbracket_G \ ss, \llbracket mis \rrbracket_G \ ss) \\
\llbracket mss \rrbracket_G \ [] &= [] \\
\llbracket mis \rrbracket_G (s : ss) &= \\
&\mathbf{if} \ s + \mathit{sum} \ y < 0 \ \mathbf{then} \ [] \\
&\mathbf{else} \ (s : y) \\
&\quad \mathbf{where} \ y = \llbracket mis \rrbracket_G \ ss \\
\llbracket mis \rrbracket_G \ [] &= []
\end{aligned}$$

と定義されるが，この mss で使用される条件分岐に対してはその条件が成り立たない．実際に，ソースの条件による分岐を用いて，

$$\begin{aligned}
\llbracket mss \rrbracket_P (s : ss) (v : vs) &= \\
&\mathbf{if} \ s + \mathit{sum} \ y \geq \mathit{sum} \ z \ \mathbf{then} \ v : \llbracket mis \rrbracket_P \ as \ vs \\
&\mathbf{else} \ \llbracket mss \rrbracket_P \ ss (v : vs) \\
&\quad \mathbf{where} \ (y, z) = (\llbracket mss \rrbracket_G \ ss, \llbracket mis \rrbracket_G \ ss)
\end{aligned}$$

のように定義してしまうと，

$$\begin{aligned}
\llbracket mss \rrbracket_P [9, -3, 2] [1] &= 1 : \llbracket mis \rrbracket_P [-3, 2] [] \\
&= \{ \text{mis は右畳み込みなので, } \llbracket mis \rrbracket_G [-3, 2] = [] \text{ から, 反射性より} \} \\
&\quad 1 : [-3, 2] \\
&= [1, -3, 2]
\end{aligned}$$

となるが， $\llbracket mss \rrbracket_G [1, -3, 2] = [2]$ となり変更保存性を満たさない．

このような，より一般的な条件分岐をどのようにしたらうまく扱えるかは，まだよくわかっていない．

再帰の構造

リストの右畳み込みのような関数の場合は，その構造自体を双方向変換で定義することができた．しかし，

$$\begin{aligned}
\llbracket para \rrbracket_G f \ n (s : ss) &= f \ s \ ss (\llbracket para \rrbracket_G f \ n \ ss) \\
\llbracket para \rrbracket_G f \ n \ [] &= n
\end{aligned}$$

のような再帰構造を持つ関数だと、 f の逆方向変換で得られた ss と、 $para$ の再帰呼び出しの逆方向変換で得られた ss が矛盾する可能性がある。それ以外の場合でも、その後順方向変換した場合に f の引数の値が変化し、変更保存性を満たさない場合もありうる。

このような右畳み込み以外の再帰の構造を反射性や局所変更保存性などの「よい性質」を保つように取り扱えるかどうかは今後の課題である。

型システム

Foster らは、反射性および変更保存性の合成などに対し閉じているという性質から、型システムに相当するものを双方向変換に導入することができた [?]。

しかし、複製を含む場合、本論文の提案する性質である局所変更保存性は合成に対して閉じていない。それは、変換が持ち運ぶ情報が反射性の成立と局所変更保存性の成立だけでは、変換の合成の性質を議論するに足りないためであると考えられる。どのような情報を持ち運べばこのような合成をうまく行えるかについては、まだわかっていない。

6.2.2 双方向変換システムとアプリケーション

双方向変換の推移性

CVS[?] や Subversion[?] のようなバージョン管理システムに双方向変換システムを応用する場合を考える。その場合、ユーザはしばらくローカルなコピーについて作業したあと、`cvs commit` などをすることにリモートのレポジトリに結果を反映させる。その際に、作業をこまぎれに反映させた結果と、まとめて反映させた結果とが異なっていることは望ましくない。

このような用途に双方向変換システムを応用するためには、

$$\llbracket t \rrbracket_P (\llbracket t \rrbracket_P (s, v'), v'') = \llbracket t \rrbracket_P (s, v)$$

のような性質 [?, ?] が成り立つことが望ましい。

このような推移性を満たさない例として、

$$\begin{aligned} proj &= \text{bifldr } c \ [] \ p \\ \text{where } \llbracket c \rrbracket_G (s_1, s_2) &= s_1 \\ \llbracket c \rrbracket_P ((s_1, s_2), v) &= (v, s_2) \\ \llbracket c \rrbracket_P (\Omega, v) &= (v, 0) \\ p \ x &= x \neq [] \end{aligned}$$

という変換がある。この変換について、

$$\begin{aligned} \llbracket proj \rrbracket_P (\llbracket t \rrbracket_P (\llbracket (1, 2) \rrbracket, []), \llbracket 2 \rrbracket) \\ &= \llbracket proj \rrbracket_P (\llbracket \llbracket \rrbracket, \llbracket 2 \rrbracket) \\ &= \llbracket 2, 0 \rrbracket \end{aligned}$$

に対し,

$$\begin{aligned} \llbracket \text{proj} \rrbracket_P ((1, 2), [2]) \\ = [2, 2] \end{aligned}$$

となり, 推移性を満たさない. これに対し, [?] では型情報によりこれを解決することを提案している. しかし, 彼らはこれに対し十分な議論を行っているわけではなく, また複製を含む双方向変換に関してはさらに議論が少ない.

X^\pm 言語における木の畳み込み

4.2 節で定義された X^\pm 言語に対し木上の畳み込み関数のような変換を導入することができれば, 部分木をすべて走査することができ表現力が広がる.

しかし, 畳み込み関数を導入することで, $\llbracket - \rrbracket_{\text{Global}} i p$ の呼び出しが唯一でなくなり DupTo など複製されたノードを区別するのが難しくなる.

その他のアプリケーションへの応用

本論文では, 双方向変換のファイルマネージャへの応用を示した. その他にも双方向変換の応用先として,

- Web サイト作成支援
- バージョン管理システム
- 汎用エディタ

などが考えられ, そのアプリケーションの性質によって双方向変換に望まれる性質が変わることが考えられる. そのような他のアプリケーションへの応用, および変換の満たすべき性質の議論は今後の課題である.

謝辞

忙しい中，度々研究に関する議論の時間を設けて下さった胡振江助教授に大変深く感謝申し上げます．ありがとうございました．研究に対し大局的な視点から様々な助言を授けてくださった武市正人教授に深謝致します．論文の校正を手伝ってくださった松崎公紀助手に感謝します．

また，本研究に当って様々な助言を下さり，議論に付き合っ下さった箕一彦博士および穆信成博士に心から感謝いたします．

さらに，ホワイトボードを通じて議論しあった研究室の学生の皆様にお礼の言葉を申し上げます．