博士論文

補関数による
プログラムの双方向化に関する研究

松田一孝

指導教員　武市正人　教授

胡振江　　准教授

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，元のデータの一部を抽出し加工する順方向変換と，順方向変換で得られたデータに対する更新を元データに反映する逆方向変換の二つの変換から構成される．ただし，元データへの更新の反映は，たとえば「更新された元データにふたたび変換を適用すると更新を行った変換後のデータと等しい」,「変換後のデータの構築に関係ない部分を変更しない」などの順方向変換に対し何らかの「振る舞いのよさ」を保たなければならない．双方向変換を用いることで，XML 文書の同期や相互変換を行える．さらには，双方向変換はプレゼンテーション指向の文書作成やソフトウェアエンジニアリングにも利用できる．しかし，順方向変換に対し,「振る舞いのよい」逆方向変換を与えるのは難しい．また，仮に逆方向変換が与えられたとしても，その動作が振る舞いのよいものであるかを確認するのも容易ではない．さらには，ある順方向変換に対して双方向に動作する逆方向変換は一つとは限らず，逆方向変換の中には，振る舞いはよいもののほとんどの更新を反映することができず効果的でない逆方向変換も含まれる．よって，順方向変換から，振る舞いがよく効果的な逆方向変換を求める手法が求められている．

近年，木構造データ上の双方向変換の重要性は増してきている．特に木構造データの一種である XML は広く普及しており，様々なアプリケーション間のデータ受け渡し用のデータの事実上標準となっている．それぞれのアプリケーションが異なる形式の XML を要求していることも多いため，XML を様々な形式に変換できると便利である．また，そういった場合に，変換後のデータの上でなされた更新を元のデータに反映することができればさらに便利である．

これまで，順方向変換に対し「補関数」という関数を与えることで，振る舞いのよい逆方向変換を定められることが知られていた．直観的には，補関数とは,「副作用のない」更新の反映を特徴付けるため導入された概念であり，元データのうち順方向変換の結果の構築に関係のない部分を全て抽出する関数である．よって，補関数値を不変に保つことで，変換結果に関係のない部分を変更することなく元データを更新することができる．しかし，数学的な関数としてではなく実行可能なプログラムとして補関数を求める議論は少ない．これまで，関係データベース上の問い合わせについて補関数を問い合わせの形で求める研究はあったものの，木構造データに対する変換プログラムに対する補関数プログラムの導出についての議論はなかった．また，補関数プログラムが仮に求まったとしても，数学的に逆方向変換関数を定義する場合とは違い，逆方向変換プログラムを求めるのは容易ではない．

本論文では，我々は，特定のクラスの関数型言語で記述された木構造データ上の順方向変換プログラムから補関数プログラムを導出し，それに基づき逆方向変換プログラムを自動導出する「プログラムの双方向化」手法を提案する．提案手法は順方向変換の単射性に注目する．順方向変換プログラムは一般には単射ではないために，元データに変換結果の構築に関

係ない部分が含まれる．補関数はこの変換結果の構築に関係ない部分を含まねばならないので，我々は，順方向変換プログラムにおける「プログラムが単射でなくなっている場所」において適切に情報を補うように順方向変換に対する補関数を構成する．

本研究の主な貢献は次の二点である．

一つ目の貢献は，順方向変換記述する言語を適切に制限することにより，プログラムを自動的に双方向化する手法を与えたことである．順方向変換記述言語は treeless で affine と制限されているものの，多くの基本的な変換を記述することができる．また，制限されているおかげでプログラムの単射性を効果的に解析することができる．プログラムの単射性を解析することにより我々は補関数が補うべき情報を適切に知ることができるため，より効果的な逆方向変換を定める補関数を導出することができる．さらに，制限により，得られた補関数プログラムから逆方向変換プログラムを導出するのも容易になっている．

二つ目の貢献は，XML 上の双方向変換を記述できるように一つ目の貢献で述べた双方向化の手法を拡張したことである．確かに，treeless で affine な言語で記述されたプログラムに対する双方向化の手法を，双方向結合子などの他の手法と組み合わせることで広範な順方向変換プログラムに適用することは可能である．しかし，そのようにして得られる逆方向変換はしばしば効果的ではない場合がある．XML の変換においては，XML の要素列の連接や変換結果の複製，そして元データの複数走査を順方向変換が含む場合に，素朴に導出した逆方向変換はあまり効果的でないものになることが多い．これらの問題箇所に対し，我々は，双方向化手法を拡張しまた適切に言語の補助を与えることにより，特定のクラスの XML 変換プログラムに対しても効果的な逆方向変換を自動導出する手法を与えた．

目 次

第 1 章 はじめに 1

1.1 背景 1

1.1.1 双方向変換 1

1.1.2 木構造データに対する変換 4

1.1.3 振る舞いのよい逆方向変換の構成 5

1.2 本論文の目的 6

1.3 我々のアプローチ 6

1.4 論文の構成 8

1.5 基本的な概念と記法 9

第 2 章 関連研究 13

2.1 双方向変換 13

2.2 双方向変換における複製の扱い 17

2.3 双方向変換の定式化 18

第 3 章 補関数に基づく双方向化 21

3.1 双方向変換 22

3.1.1 定義 23

3.1.2 振る舞いのよい双方向変換 24

3.2 補関数に基づく双方向化 26

3.2.1 補関数 27

3.2.2 補関数値不変に基づく逆方向変換 27

3.2.3 補関数の優劣 32

3.3 プログラムの双方向化のために 33

第 4 章 正規木文法と正規生垣文法 35

4.1 木と生垣 35

4.1.1 木 36

4.1.2 生垣 36

4.2 正規木文法と正規生垣文法 37

4.2.1 正規木文法 38

4.2.2 正規生垣文法 . . . 40
4.2.3 性質 . . . 42

第 5 章 プログラムの双方向化 45
5.1 概観 . . . 45
5.1.1 順方向変換 . . . 45
5.1.2 双方向化 . . . 46
5.1.3 更新の反映可能性検査器 . . . 48
5.2 順方向変換記述言語 VDL . . . 48
5.2.1 言語 VDL の構文 . . . 48
5.2.2 言語 VDL の意味 . . . 50
5.3 補関数の導出 . . . 51
5.3.1 素朴な補関数導出 . . . 51
5.3.2 値域の推論 . . . 56
5.3.3 単射性判定 . . . 59
5.3.4 よりよい補関数の導出 . . . 64
5.4 逆方向変換の導出 . . . 68
5.4.1 組化：関数 $\langle f, f^c \rangle$ のプログラムの導出 . . . 68
5.4.2 逆関数の導出：$\langle f, f^c \rangle^{-1}$ の計算 . . . 71
5.4.3 逆方向変換の構成 . . . 73
5.5 更新の反映可能性判定器の導出 . . . 73
5.6 例 . . . 79
5.7 まとめ . . . 81
5.8 問題点 . . . 81

第 6 章 XML 上の順方向変換記述言語 83
6.1 Treeless 制約の緩和 . . . 83
6.1.1 緩和の要請 . . . 83
6.1.2 我々のアプローチ . . . 84
6.2 Affine 制約の緩和 . . . 85
6.2.1 悲観的な解決案 . . . 86
6.2.2 楽観的な解決案 . . . 87
6.2.3 我々のアプローチ . . . 88
6.3 順方向変換記述言語 VDL$^+$ . . . 89
6.3.1 構文と意味 . . . 89
6.3.2 言語 VDL$^+$ の変換記述例 . . . 92
6.4 双方向化における問題 . . . 96

6.4.1 式の値域および関数の単射性の文脈依存性 96
6.4.2 値域の重なりの判定 98

第 7 章 引数の型に基づく関数の特化 99
7.1 関連研究 100
7.2 提案手法のアイデア 101
7.2.1 アイデア 101
7.2.2 素朴な実現の問題点 101
7.3 型に基づく関数の特化 102
7.3.1 パターン特化 105
7.3.2 特化のアルゴリズム 110
7.3.3 提案する特化手法の性質 113
7.3.4 パターンの併合 121
7.4 特化手法の拡張 125
7.5 まとめ 126

第 8 章 XML 上の変換プログラムの双方向化 127
8.1 補関数の導出 127
8.1.1 値域の厳密な推論 128
8.1.2 値域の近似 136
8.1.3 単射性判定 145
8.1.4 補関数導出 148
8.2 逆方向変換導出 156
8.2.1 **let** 束縛の生成とパターンの生成 158
8.2.2 組にした関数の逆関数：$\langle f, f^{c} \rangle^{-1}$ の生成 159
8.3 より詳細な解析による補関数導出 174
8.3.1 返り値の使用に基づく関数の特化 174
8.3.2 近似前の値域を利用した重なりの判定 177
8.3.3 得られた補関数の改良 182

第 9 章 まとめ 185
9.1 結論 185
9.2 今後の課題 185
9.2.1 補関数導出手法 185
9.2.2 順方向変換記述言語 191

謝辞 195

参考文献 197

第1章　はじめに

1.1　背景

1.1.1　双方向変換

あるデータ（**ソース**と呼ぶ）から一部を取り出し加工し別のデータ（**ビュー**と呼ぶ）へと変換する．この時，変換後のデータへの更新を元のデータへと書き戻すことが行えれば便利である．ただし，元データへの更新の反映は，順方向変換に対し，たとえば更新された元データにふたたび変換を適用すると更新を行った変換後のデータと等しいなどの，何らかの整合性を満たさなければならない．**双方向変換**は，この元のデータを加工する順方向変換と，元データと変換後のデータの間の整合性を保ちながら更新の反映をを行う逆方向変換の二つの変換の組である．双方向変換は，たとえば，構造化文書の同期 [FGM+05]，非構造化文書の同期 [BFP+08]，プレゼンテーション指向の文書作成 [HMT04]，ユーザインタフェース作成のための制約解消 [Mee98] などに応用できる．また，双方向変換はデータベースの分野ではビュー更新として研究されている [BS81, DB82, GPZ88, Heg90, LV03]．

双方向変換を例を用いて説明する．ソースとして，研究室のメンバーリストを表す以下の XML 要素を考える．

$$s = \left(\begin{array}{l}\texttt{<members>}\\ \quad\texttt{<student>Metsuda</student>}\\ \quad\texttt{<professor>Hu</professor>}\\ \quad\texttt{<professor>Takeichi</professor>}\\ \texttt{</members>}\end{array}\right)$$

上のメンバーリスト（`<members>`）は学生（`<student>`）と教授（`<professor>`）から構成される．このうち学生（`<student>`）のみを抽出する順方向変換を考える．たとえば上のソース s から学生のみを抽出すると以下となる．

$$v = \left(\begin{array}{l}\texttt{<members>}\\ \quad\texttt{<student>Metsuda</student>}\\ \texttt{</members>}\end{array}\right)$$

この順方向変換は以下の関数 *students* により実現できる．

$$
\begin{aligned}
&\mathit{students}(\texttt{<members>}(x)) &&\mathrel{\hat{=}} \texttt{<members>}(f(x))\\
&\quad f(\varepsilon) &&\mathrel{\hat{=}} \varepsilon\\
&\quad f(\texttt{<student>}(x)\,\texttt{.}\,r) &&\mathrel{\hat{=}} \texttt{<student>}(x)\,\texttt{.}\,f(r)\\
&\quad f(\texttt{<professor>}(x)\,\texttt{.}\,r) &&\mathrel{\hat{=}} f(r)
\end{aligned}
$$

ここで「.」は XML 要素列の連接，ε は空列，そして $\texttt{<tag>}(x)$ は $\texttt{<tag>}x\texttt{</tag>}$ の略記である．順方向変換 *students* に対する逆方向変換は，*students* の結果である学生のみを含むリストに対する変更を，元の研究室のメンバーリストに反映する．たとえば，以下の関数 $\mathit{students}_{\mathrm{B}}$ は，*students* の逆方向変換の一つである．

$$
\begin{aligned}
&\mathit{students}_{\mathrm{B}}(\texttt{<members>}(s), \texttt{<members>}(v)) &&\mathrel{\hat{=}} \texttt{<members>}(f_{\mathrm{B}}(s,v))\\
&\quad f_{\mathrm{B}}(\varepsilon,\varepsilon) &&\mathrel{\hat{=}} \varepsilon\\
&\quad f_{\mathrm{B}}(\texttt{<student>}(x)\,\texttt{.}\,r, \texttt{<student>}(x')\,\texttt{.}\,r') &&\mathrel{\hat{=}} \texttt{<student>}(x')\,\texttt{.}\,f_{\mathrm{B}}(r,r')\\
&\quad f_{\mathrm{B}}(\texttt{<professor>}(x)\,\texttt{.}\,r, r') &&\mathrel{\hat{=}} \texttt{<professor>}(x)\,\texttt{.}\,f_{\mathrm{B}}(r,r')
\end{aligned}
$$

逆方向変換 $\mathit{students}_{\mathrm{B}}$ は更新反映前のソースと更新後のビューを取り，更新を反映した新しいソースを返す．たとえば，ある人が，ビュー上で学生の名前が間違っているのに気付き，学生の名前 Metsuda を Matsuda に更新したとする．この更新によりビュー $v = \mathit{students}(s)$ は以下の v' へと変更される．

$$
v' = \left(\begin{array}{l}
\texttt{<members>}\\
\quad\texttt{<student>Matsuda</student>}\\
\texttt{</members>}
\end{array}\right)
$$

この v から v' への更新は，逆方向変換 $\mathit{students}_{\mathrm{B}}$ により反映され，ソースは以下へと更新される．

$$
\mathit{students}_{\mathrm{B}}(s, v') = \left(\begin{array}{l}
\texttt{<members>}\\
\quad\texttt{<student>Matsuda</student>}\\
\quad\texttt{<professor>Hu</professor>}\\
\quad\texttt{<professor>Takeichi</professor>}\\
\texttt{</members>}
\end{array}\right)
$$

上のソースにおいて，元のソース s 中の Metsuda は，逆方向変換後のソース $\mathit{students}_{\mathrm{B}}(s, v')$ 中では Matsuda へと更新されており，v から v' への更新が反映されたことが確認できる．

　一般には，ある順方向変換に対し逆方向変換は複数存在する．順方向変換に対する逆方向変更は,「ビューの構築に関係のないソースを変更しない」など，**振る舞いのよい**ものであるものが望ましい．また，より多様な更新を反映できるという意味で，より**効果的**な逆方向変換が望ましい．たとえば，以下の二つの関数も *students* の逆方向変換である．

$$
\mathit{students}'_{\mathrm{B}}(s, v) = s \quad \text{if } v = \mathit{students}(s)
$$

$$
\begin{array}{ll}
\mathit{students}''_{\mathrm{B}}(\texttt{<members>}(s), \texttt{<members>}(v)) & \hat{=}\ \texttt{<members>}(g_{\mathrm{B}}(s,v)) \\
\quad g_{\mathrm{B}}(\varepsilon,\varepsilon) & \hat{=}\ \varepsilon \\
\quad g_{\mathrm{B}}(\varepsilon, \texttt{<student>}(x') \centerdot r') & \hat{=}\ \texttt{<students>}(x') \centerdot g_{\mathrm{B}}(\varepsilon, r') \\
\quad g_{\mathrm{B}}(\texttt{<student>}(x) \centerdot r, \varepsilon) & \hat{=}\ \varepsilon \\
\quad g_{\mathrm{B}}(\texttt{<student>}(x) \centerdot r, \texttt{<student>}(x') \centerdot r') & \hat{=}\ \texttt{<student>}(x') \centerdot g_{\mathrm{B}}(r, r') \\
\quad g_{\mathrm{B}}(\texttt{<professor>}(x) \centerdot r, r') & \hat{=}\ \texttt{<professor>}(x) \centerdot g_{\mathrm{B}}(r, r')
\end{array}
$$

これらの逆方向変換は，効果的でないもしくは振る舞いのよくないものである．

逆方向変換 $\mathit{students}'_{\mathrm{B}}$ はあまり効果的でない．これは，関数 $\mathit{students}'_{\mathrm{B}}$ は，$v = \mathit{students}(s)$ であるソースしか定義域に含まないため，ビュー上の意味のある変更をソースに反映することができないためである．

逆方向変換 $\mathit{students}''_{\mathrm{B}}$ は，$\forall s, v.\ \mathit{students}_{\mathrm{B}}(s,v) = s' \Rightarrow \mathit{students}''_{\mathrm{B}}(s,v) = s'$ を満たす．つまり，$\mathit{students}_{\mathrm{B}}$ の反映できる更新を全て反映することができる．そのため，$\mathit{students}''_{\mathrm{B}}$ は $\mathit{students}_{\mathrm{B}}$ より効果的な逆方向変換である．しかし，$\mathit{students}''_{\mathrm{B}}$ は，ビューの構築に関係のないソースを変更してしまうため振る舞いがよいものではない．たとえば，ソースが以下であった場合を考える．

```
<members>
  <student>Metsuda</student>
  <professor>Hu</professor>
  <professor>Takeichi</professor>
</members>
```

このとき，*students* によるビューは以下である．

```
<members>
  <student>Metsuda</student>
</members>
```

また，ソースが

```
<members>
  <professor>Hu</professor>
  <student>Metsuda</student>
  <professor>Takeichi</professor>
</members>
```

であったとして，ビューは上のものとなるので，教授（<professor>）のメンバーリストにおける順序や，学生（<student>）と教授との位置関係はビューの構築には関係がない．

ビューに対する更新として，以下の更新を考える．

```
<members>
  <student>Metsuda</student>
</members>
```

$$\rightarrowtail$$

```
<members>
</members>
```

$$\rightarrowtail$$

```
<members>
  <student>Matsuda</student>
</members>
```

この更新は，`Metsuda` という名前の学生を一度ビューから削除する更新と，`Matsuda` という名前の学生を挿入しなおすという更新の二つの更新により，学生の名前を `Metsuda` から `Matsuda` へと変更する．ここで，それぞれの更新ごとに $students''_{\mathrm{B}}$ により更新を反映するとソースは

```
<members>
  <professor>Hu</professor>
  <professor>Takeichi</professor>
  <student>Matsuda</student>
</members>
```

と更新される．ここで，逆方向変換 $students''_{\mathrm{B}}$ はビューの構築に関係のない「教授と学生のメンバーリストにおける位置関係」を変更してしまっている．そのため，$students''_{\mathrm{B}}$ は振る舞いのよい逆方向変換ではない．ソースにおいて，ビューの構築に関係のない部分が変更されてしまうと，ビューを通してソースにおけるその変更を確認できなくなってしまう．

先程定義した，順方向変換 $students$ に対する逆方向変換 $students_{\mathrm{B}}$ は,「ビューの構築に関係のないソース」を変更することがなく，振る舞いのよいものである．また，逆方向変換 $students_{\mathrm{B}}$ は，抽出した各学生の名前への変更をソースに反映することができ，それなりに多様な更新をソースに反映できるため，振る舞いがよいものの中では効果的である．しかし，一般には，順方向変換に対し，振る舞いのよい効果的な逆方向変換を与えるのは難しい．また，手で逆方向変換を与えたとしても，それが振る舞いのよいものなのかどうかを確認するのも容易ではない．よって，順方向変換から，振る舞いがよく効果的な逆方向変換を求める手法が求められている．

1.1.2 木構造データに対する変換

近年，木構造データ上の双方向変換の重要性は増してきている．特に木構造データの一種である XML は広く普及しており，様々なアプリケーション間のデータ受け渡し用のデータの事実上標準となっている．それぞれのアプリケーションが異なる形式の XML を要求していることも多いため，XML を様々な形式に変換できると便利である．

たとえば，以下の論文様の構造の XML を考える．

```
<chapter>
    <title>はじめに</title>
    <p>あるデータを変換により，別のデータ...</p>
    <p>双方向変換を例を...</p>
</chapter>
<chapter>
    <title>関連研究</title>
    <p>本章では，本論文に関連の深い研究...</p>
    <section>
        <title>双方向変換</title>
        ...
    </section>
</chapter>
...
```

上の XML を次のように XHTML に変換すると，Web ブラウザにより視覚的に XML 内容を確認することができる．

```
<h1>はじめに</h1>
    <p>あるデータを変換により，別のデータ...</p>
    <p>双方向変換を例を...</p>
<h1>関連研究</h1>
    <p>本章では，本論文に関連の深い研究...</p>
    <h2>双方向変換</h2>
...
```

また，そういった場合に，変換後のデータの上でなされた更新を元のデータに反映することができればさらに便利である．

1.1.3 振る舞いのよい逆方向変換の構成

順方向変換に対し**補関数**という関数を与えることで，振る舞いのよい逆方向変換が与えられることが知られている [BS81, Heg90]．直観的には，補関数とは，ソースのうちビューの構築に関係のない部分を全て抽出する関数であり,「副作用のない」更新の反映を特徴付ける．すなわち，関数

$$f :: S \to V$$

に対する補関数

$$g :: S \to V'$$

とは，関数

$$\langle f, g \rangle :: S \to (V \times V')$$

を単射にする関数である．ただし，$\langle f, g\rangle(x) = (f(x), g(x))$ である．よって，補関数値を不変に保つことで，ビューの構築に関係のない部分を変更することなくソースを更新することができる．すなわち，順方向変換 f に対する補関数 g が与えられれば，

$$\mathsf{reflect}_{f,g}(s, v) = \langle f, g\rangle^{-1}(v, g(s))$$

とすることにより，振る舞いのよい逆方向変換を定めることができる．得られる逆方向変換は，補関数の選び方により異なる．たとえば，恒等写像 $\mathrm{id}(x) = x$ は，どの関数に対しても補関数になる．しかし，$\mathsf{reflect}_{f,\mathrm{id}}(s, v)$ は $v = f(s)$ である (s, v) についてしか定義されておらず意味のある更新の反映ができないため，id は補関数として望ましいものではない．一般には，よりソースの値を区別しないという意味で，よりソースの情報を保存しない補関数がより効果的な逆方向変換を定めることが知られている [BS81].

しかし，これまで，順方向変換プログラムから具体的な補関数プログラムを導出し，逆方向変換プログラムを導出する研究はほとんどない [CP84,LLSV01,LV03]．関係データベース上の問い合わせについて補関数を問い合わせの形で求める研究はあった [CP84,LLSV01,LV03] ものの，少くとも我々の知る限り，木構造データに対する変換プログラムに対する補関数プログラムの導出についての議論はなかった．関係データベース上の問い合わせにおいては，ソース中の「ビューの構築に関係のない部分」が組の集合で表現できるため，補関数導出の議論が木構造上に比べて簡単である．ところが，木構造上の変換においては，ソースにおける「ビューの構築に関係のない部分」はソースの構造と直接的な対応をしてるとは限らない．たとえば，以下の *half* において，ソースにおける「ビューの構築に関係のない部分」は，入力となる列の長さの偶奇である．

$$\begin{array}{lcl}
\mathit{half}(\texttt{<z>}(\varepsilon)) & \hat{=} & \texttt{<z>}(\varepsilon) \\
\mathit{half}(\texttt{<s>}(\varepsilon) \mathbin{.} \texttt{<z>}(\varepsilon)) & \hat{=} & \texttt{<z>}(\varepsilon) \\
\mathit{half}(\texttt{<s>}(\varepsilon) \mathbin{.} \texttt{<s>}(\varepsilon) \mathbin{.} r) & \hat{=} & \texttt{<s>}(\varepsilon) \mathbin{.} f(r)
\end{array}$$

1.2　本論文の目的

本論文の目的は，木構造データ上の順方向変換プログラムから，振る舞いがよく，また効果的な逆方向変換プログラムを自動導出する手法を与えることである．すなわち，プログラムの**双方向化**による双方向変換の構成である．また，逆方向変換プログラムの導出にあたって，順方向変換プログラムのどのような性質が，効果的な逆方向変換プログラムを得ることを可能にするのかを明らかにすることを目指す．

1.3　我々のアプローチ

本論文では，我々は，特定のクラスの関数型言語で記述された木構造データ上の順方向変換プログラムから補関数プログラムを導出しそれに基づき逆方向変換プログラムを自動導出

$$
\begin{array}{l}
\textbf{data}\ S \mathrel{\hat{=}} (\texttt{<section>}(\texttt{<title>}(\mathit{String})\,.\,P))^{*} \\
\textbf{data}\ P \mathrel{\hat{=}} (\texttt{<p>}(\mathit{String}))^{*} \\
\\
c2x(\varepsilon) \mathrel{\hat{=}} \varepsilon \\
c2x(\texttt{<chapter>}(\texttt{<title>}(t :: \mathit{String})\,.\,p :: P\,.\,s :: S)\,.\,r) \\
\quad \mathrel{\hat{=}} \texttt{<h1>}(t)\,.\,p\,.\,s2x(s)\,.\,c2x(r) \\
s2x(\varepsilon) \mathrel{\hat{=}} \varepsilon \\
s2x(\texttt{<section>}(\texttt{<title>}(t :: \mathit{String})\,.\,p :: P)\,.\,r) \\
\quad \mathrel{\hat{=}} \texttt{<h2>}(t)\,.\,p\,.\,s2x(r)
\end{array}
$$

図 1.1. 1.1.2 節の変換を記述する関数 $c2x$

する手法を提案する．提案手法は順方向変換の単射性に注目する．順方向変換プログラムは一般には単射ではないために，ソースに変換結果の構築に関係ない部分が含まれる．補関数はこの変換結果の構築に関係ない部分を含まねばならないので，我々は，順方向変換プログラムにおける「プログラムが単射でなくなっている場所」において適切に情報を補うように順方向変換に対する補関数を構成する．

貢献

本論文における我々の貢献は以下の二つにまとめられる．

一つ目の貢献は，**順方向変換記述言語を適切に制限することにより自動的に振る舞いのよい逆方向変換を導出する手法を与えた**，ということである．我々は，affine かつ treeless [Wad90] である一階の関数型言語で記述されたプログラムに対する自動的なプログラムの双方向化手法を提案した．順方向変換言語は，affine（複製がなく，同じ入力を二度以上走査できない）かつ treeless（関数呼出がネストできない）と制限されているものの，*map* 様関数など，多くの基本的な変換を記述することができる．また，制限されているおかげでプログラムの単射性を効果的に解析することができる．本言語で記述された関数に対し，関数が単射であるかないかを厳密に判定することができる．プログラムの単射性を解析することにより我々は補関数が補うべき情報を適切に知ることができるため，本言語で記述された変換については，効果的な逆方向変換を自動的に与えることができる．一般には，補関数のプログラムが得られたのちに，逆方向変換プログラムを導出するのも，プログラム逆計算 [Dij78, Epp85, GK04, GK05, Gri81, NSS05] が必要になり容易ではない．しかし，言語に制限をおいたおかげで，得られた補関数プログラムから逆方向変換プログラムを導出するのも容易になっている．

二つの目の貢献は，**XML 上の変換を効果的に双方向化できるように，前述の双方向化手法を拡張した**，ということである．我々の前述の双方向化手法は，双方向化結合子 [FGM+05, HMT04, MHT04a] の技術などを利用することにより，より広範のプログラムに適用するこ

$$
\begin{array}{l}
\textbf{data}\ T_1 \mathrel{\hat=} (\texttt{<h2>}(\mathit{String})\centerdot P)^* \\
\textbf{data}\ T_2 \mathrel{\hat=} (\texttt{<h1>}(\mathit{String})\centerdot P \centerdot T_1)^* \\
\textbf{data}\ P\ \mathrel{\hat=} (\texttt{<p>}(\mathit{String}))^* \\
c2x_{\mathrm{B}}(s, v) \mathrel{\hat=} c2x^{-1}(v) \\
c2x^{-1}(\varepsilon) \mathrel{\hat=} \varepsilon \\
c2x^{-1}(\texttt{<h1>}(t :: \mathit{String})\centerdot p :: P \centerdot v_1 :: T_1 \centerdot v_2 :: T_2) \\
\quad \texttt{<chapter>}(\texttt{<title>}(t :: \mathit{String})\centerdot p \centerdot s2x^{-1}(v_1))\centerdot c2x^{-1}(v_1) \\
s2x^{-1}(\varepsilon) \mathrel{\hat=} \varepsilon \\
s2x^{-1}(\texttt{<h2>}(t :: \mathit{String})\centerdot p :: P \centerdot v_3 :: T_1) \\
\quad \mathrel{\hat=} \texttt{<section>}(\texttt{<title>}(t)\centerdot p)\centerdot s2x^{-1}(v_3)
\end{array}
$$

図 1.2. 図 1.1 のプログラムの双方向化結果

とができる．ただし，そのようにして得られた逆方向変換はあまり効果的にならない場合がある．XML の変換においては，XML の要素列の連接や変換結果の複製，そして元データの複数走査を順方向変換が含む場合に，素朴な双方向化手法の拡張よりに導出した逆方向変換はあまり効果的でないものになることが多い．これらの問題箇所に対し，我々は，双方向化手法を拡張しまた適切に言語の補助を与えることにより，特定のクラスの XML 変換プログラムに対しても効果的な逆方向変換を自動導出する手法を与えた．具体的には，我々は affine で treeless な順方向変換記述言語を拡張し，XML 要素列の分割・連接のための演算子と多返値関数を加え，双方向化の議論を行なった．たとえば，本章 1.1.2 節の論文様の XML から XHTML 断片への変換は，図 1.1 の $c2x$ のようにこの言語で記述することができる．多返値関数の導入のアイデアは，組化 [HITT97, Chi93] による．複製の含むプログラムの双方向化における問題は，複製により同じ入力が二度以上走査されることによって引き起される．組化はこのデータの複数回走査をなくし多返値関数を導入するプログラム変換である．また，本言語で記述されたプログラムの単射性の解析は決定可能ではないものの，効果的に行うことができる．たとえば，図 1.1 の $c2x$ は単射だと判定され，図 1.2 の逆方向変換 $c2x_{\mathrm{B}}$ が導出される．

また我々は，提案する手法を実装し，手法の有効性を確認した．本論文の内容の実装に関する情報は，`http://www.ipl.t.u-tokyo.ac.jp/~kztk/b18n/` にある．

1.4 論文の構成

本論文の構成は以下の通りである．

第 2 章では，双方向変換分野における研究と本論文の内容との関わりを述べる．また，双方向変換と関連の深い，単射な関数から逆関数を導出するプログラム逆計算，単射な関数とその逆関数を同時に記述する可逆計算，本論文でいう双方向変換の前身であるビュー更新に

ついても簡潔に述べる.

第3章, 第4章は本論文の主要結果を理解するために必要な既存の知識について述べる. 第3章では, 本論文が提案するプログラム変換の基礎となる, 補関数に基づく双方向化 [BS81] に述べる. 第4章では, 木, および XML 文書の表現である生垣について述べる. これらは, 本論文でプログラム変換の対象となる, プログラム言語の入力および出力となる. また, 第4章においては, 木や生垣からなる集合の記述である, 正規木文法や正規生垣文法 [CDG+97] についても述べる.

第5章, 第6章, 第7章, 第8章は本論文の主要成果である. 第5章は, 本論文の貢献の一つである, 補関数に基づく双方向化の基本的なアイデアを, 簡単なプログラム言語を用いて説明する. ここで, 単射性解析が双方向化において, より更新を反映する逆方向変換プログラムを得るために重要であることを示す. 第6, 7, 8章では, 本論文の二つ目の貢献を述べる. 第6章は, 第5章で扱った言語を拡張し, 簡単な XML 変換を記述できるようにする. この際, 順方向変換が複製を含むと双方向化の議論が難しくなることを示し, また, 言語の設計により, 複製の難しさの一部は解決できることも示す. 第7章は, 第6章の言語で記述されたプログラムに対する双方向化のための前処理手法である, 関数の引数の型に対する特化について述べる. 関数を引数の型に基づき特化することにより, より正確に順方向変換の単射性を解析し, より効果的な逆方向変換を導出することが可能になる. 第8章は, 第6章で定めた言語に対し, 双方向化の手法を実際に適用する. 第6章で適切に対象言語を定めたことにより, 第5章の双方向化手法の簡単な拡張により, 第6章の言語で記述された順方向変換も効果的に双方向化できることを示す.

第9章では, 本論文の内容を取りまとめ, 今後の課題を示す.

1.5 基本的な概念と記法

本論文を通して使用する基本的な概念や記法について述べる.

部分関数

本論文では, 定義域の全ての値に対し関数の値が定義されている全域関数 (total function) ではなく, 定義域の一部の値に対し関数の値が定義されていないかもしれない部分関数 (partial function) を扱う. たとえば, 以下の関数 $div :: \mathbb{Z} \times \mathbb{Z} \to \mathbb{Q}$ は部分関数である.

$$div(x, y) = x/y$$

なぜなら, div は第二引数が 0 の時に値が定義されないためである.

部分関数 f に対し, f が入力 x に対し関数の値が定義されている, すなわち $f(x) = v$ となる v が存在する場合に

$$f(x){\downarrow}$$

と書く．また，部分関数 f に対し，f が入力 x に対し関数の値が定義されていないとき，

$$f(x) = \bot$$

と書く．たとえば，$div(2,3)\downarrow$ であり，$div(2,0) = \bot$ である．

本論文では，単に「関数」と書いた場合部分関数を指す．

関数 $f :: X \to Y$ の実際の定義域 $\mathsf{dom}(f)$ を

$$\mathsf{dom}(f) = \{x \mid f(x)\downarrow\}$$

で定める．同様に，関数 $f :: X \to Y$ の実際の値域 $\mathsf{ran}(f)$ を

$$\mathsf{ran}(f) = \{y \mid f(x) = y, y \neq \bot\}$$

定める．たとえば，$\mathsf{dom}(div) = \{(x,y) \mid x \in \mathbb{Z}, y \in \mathbb{Z}, y \neq 0\}$ であり，$\mathsf{ran}(div) = \mathbb{Q}$ である．関数 $f :: X \to Y$ について，常に $\mathsf{dom}(f) \subseteq X$ かつ $\mathsf{ran}(f) \subseteq Y$ であることを仮定する．本論文では，特に断わらない限り,「実際の定義域」を単に定義域と呼び,「実際の値域」を単に値域と呼ぶ．

二つの部分関数 $f :: X \to Y$ および $g :: X \to Y$ について，以下の半順序 $\sqsubseteq$ を定める．

$$f \sqsubseteq g \Leftrightarrow \forall x \in \mathsf{dom}(f).\ f(x) = g(x)$$

直観的には，$f \sqsubseteq g$ は，g が f より広範に定義されていることを表す．

単射関数

後の議論において，我々は関数の単射性に注目し議論する．ここで，関数 f が，単射であるとは，以下を満たすことである．

$$\forall x, y \in \mathsf{dom}(f).\ f(x) = f(y) \Rightarrow x = y$$

単射な関数は唯一の逆関数を持つ．単射な関数 $f :: X \to Y$ に対し，関数 $g :: Y \to X$ が $f :: X \to Y$ の逆関数であるとは，以下を満たすことである．

$$\mathsf{dom}(g) = \mathsf{ran}(f) \wedge \forall x \in \mathsf{dom}(f), y \in \mathsf{dom}(g).\ f(x) = y \Rightarrow g(y) = x$$

単射な関数 f の逆関数を f^{-1} と書く．部分関数を考えているため，関数が全単射でなくてもその逆関数が存在することに注意する．単射な f について以下が成り立つ．

$$\forall x \in \mathsf{dom}(f).\ f^{-1}(f(x)) = x \quad \wedge \quad \forall y \in \mathsf{ran}(f).\ f(f^{-1}(x))$$

組にした関数

関数 $f :: X \to Y$ および $g :: X \to Z$ に対し，f と g を組にした関数 $\langle f, g\rangle$ を以下で定める．

$$\langle f, g\rangle(x) = (f(x), g(x))$$

プログラム

本論文では，次の二種類のプログラム表記を用いる．一つは，

$$f(x) = x - 1 \quad \text{if } x > 0$$

のような $=$ を使用して定義されるものであり，もう一つは，

$$f(\mathsf{S}(x)) \doteq \mathsf{Z}$$

のような $\doteq$ を使用して定義されるものである．前者は，数式の表記に Haskell [Bir98] などのプログラムの記法を借りたものである．それに対し，後者は，本論文で議論するプログラム変換対象のプログラムである．すなわち，$=$ は数式における意味上の等価性を表現し，$\doteq$ はプログラムの構文上のただの記号である．

ベクトル記法

簡便のため，我々は列 $t_1, \ldots, t_n$ に対し，ベクトル記法 $\overrightarrow{t}$ を用いる．ここで，列 $t_1, \ldots, t_n$ の列長 n は，$|\overrightarrow{t}|$ で表現する．すなわち，$\overrightarrow{t} = t_1, \ldots, t_{|\overrightarrow{t}|}$ である．ここで言う列は，XML 要素の列とは異なることに注意する．つまり，ベクトル記法は，数式に中に現れるカンマ区切りの列の省略記法であり，XML 要素列の省略記法ではない．

Don't-Care 記法

我々は，ある値が何であるか気にしない（don't care）であることを明示的に表現するのに，Haskell に倣い「_」を用いる．たとえば，$v = (_, y)$ と書くと，v の第一要素が何であるかには特に言及せず v の第二要素が y であることを表す．また，$f(_) = 1$ と書くと，f は入力が何かによらず 1 を返す関数であることを表す．Don't-Care 記法により，不要な変数名を減らし，式を簡潔に記述することができる．

第2章　関連研究

本章では，本論文に関連の深い研究について述べる．まず，双方向変換分野の研究について 2.1 節で述べる．その後 2.2 節において，双方向変換において取り扱いの難しい要素の一つである複製について，他の研究がどのように取り扱っているのかを述べる．最後に 2.3 において，他の研究がどのように双方向変換を定式化しているのかということについて簡潔に述べる．

2.1　双方向変換

Foster らは，双方向結合子により小さな双方向変換を組み合わせていくことで，双方向変換を構成するアプローチを提案した [FGM+05]．彼らの枠組みにおいては，関数型言語において関数が言語の基本的要素になっているのと同様に，双方向変換が基本的な言語要素となっている．ソース S とビュー V 間の双方向変換は順方向の意味 $S \to V$ と逆方向 $S \times V \to S$ の意味を持つ．ここで，逆方向の意味は，更新反映前のソースと更新後のビューを取り，更新反映後のソースを返す．双方向結合子は，振る舞いのよい[1]双方向変換を結合し，振る舞いのよい双方向変換を作る．結合子としては，変換の合成，条件分岐，再帰などの基本的な言語要素に対応するものがあり，基本的な双方向変換としては，組の射影，木の枝の名前変えなど扱うデータ構造に対応したものがある．たとえば，変換の合成結合子「∘」は以下のように定義される．

$$
\begin{array}{ll}
\text{順方向} & (f \circ g)(s) = f(g(s)) \\
\text{逆方向} & (f \circ g)_{\mathrm{B}}(s, v) = g_{\mathrm{B}}(s, f_{\mathrm{B}}(f(s), v))
\end{array}
$$

結合子を用いた双方向変換構成には，以下のメリットがある．

- 双方向性の保証
- 順方向の意味での，高い記述性
- 高い拡張性

ただし，複製などの一部の変換は，結合子で表現すると効果的な双方向変換が得られない場合がある．また，彼らの枠組みにおいて，基本的な双方向変換の意味はあらかじめ与えられ

[1] 本論文の第 3 章の定義 3.4 とは異なる．

ている．それに対し，我々は，順方向変換のどのような性質がどうような逆方向変換を定めるのかを議論する．彼らの枠組みの保証する振る舞いのよさと，補関数の実現する振る舞いのよさは異なる．ただし，彼らの提案する多くの双方向結合子は，我々の意味でいう振る舞いのよい双方向変換から振る舞いのよい双方向変換を構成する．たとえば，前述の合成結合子「∘」の逆方向の意味は，f^{c} を f の補関数，g^{c} を g の補関数としたときに，$f \circ g$ の補関数を $\langle g^{\mathrm{c}}, f^{\mathrm{c}} \circ g \rangle$ としたものに対応する．

Foster らの結合子による双方向変換構成の枠組み [FGM+05] に対し，いくつかの拡張が提案されている．Hu らは複製結合子を導入し，プレゼンテーション指向エディタの作成において複製結合子を含む双方向変換の有用性を示した [HMT04,HMT08]．彼らの複製結合子はどちらの複製についての更新があった場合にも，その更新を複製元に反映できる．これにより，ビュー上の要素間の同期を達成できる．Mu らは，変更をデータに対するタグ付けで表現することにより Hu ら [HMT04] よりも柔軟な更新反映を行えることを示した [MHT04a,HMT08]．しかし，彼ら枠組みは柔軟ではあるが，振る舞いのよくない双方向変換も記述できる．複製は本研究のテーマとも関連が深いため後に詳細な関連を述べる．Bohannon らは，文字列データ操作用の結合子を与えた [BFP+08]．また，彼ら辞書というデータ構造を用い，キーに基づく逆方向変換を議論した．Foster らは商データを対象にした双方向変換を議論した [FPP08]．これにより，たとえば XML 文書中の空白など，双方向変換において無視したい，ビューやソースに含まれるデータを適切に無視することができる．文献 [FPP08] の枠組みでは，結合子は順方向と逆方向の意味に加え，ソースとビューそれぞれにおいて，どのようなにデータを商データへ対応付けるのかと，商データの値をどうやってデータへ変換するのかという意味を持つ．

Brabrand らは，はデータの XML 表現と文字列表現とを相互に変換する処理系 XSugar を実現した [BMS08]．同じデータの XML 表現と文字列表現の間の相互変換を考えているため，彼らの変換は基本的には全単射である．XSugar では，関係を記述することによって，XML 表現と文字列表現の間の対応を記述する．大雑把に言えば，彼らの関係記述は，構文主導翻訳もしくは同期文法 [AU72] と同様のものである．XSugar は，記述された変換が全単射であるかの検査器も提供している [BGM07]．我々が第 8 章で利用する技術のいくつかは，彼らの単射性検査でも使用されたものである．

双方向変換はデータベースの分野ではビュー更新 [Heg90, LLSV01, LV03, Lan90, BS81, DB82, GPZ88, WR04, BDH04, Mas84] として研究されてきた．彼らの主に，関係データベースから，問い合わせによりビューを構成し，その上の更新を元の関係データベースに反映することを議論している．それに対し，我々は木構造データから木構造データへの変換を議論する．また，補関数という概念は，関係データベースの分野で提案されたものである [BS81]．しかし，補関数に基づくビュー更新の議論は少ない [CP84, LLSV01, LV03, Heg90, Heg04]．

双方向変換は可逆計算 [Bak92,Ben89,Fra99,Lan61,PHW06,Abr05,LD82,MHT04b,PF99, YAG08a, YAG08b, YG07] と関連が深い．可逆計算においては，可逆な計算のみから計算を

記述する．ここで可逆とは，その操作に対し，順方向の意味 f と逆方向の意味 f^{-1} が両方定義されているということである．たとえば，たとえば可逆な条件分岐式は，通常の条件分岐式に加え，どちらの分岐を通って式の値を得たかを知るための，それぞれの分岐に対する互いに素な事後条件を含む [LD82, YG07]．他にも，一般のプログラム言語には，代入や繰り返しなどの可逆でない言語要素が含まれているが，それぞれに対応する可逆な言語要素を定められる．非可逆な計算は，単射にするのに十分な余分な値をともに計算しておくことで可逆な計算に埋め込む [Ben89,Lan61,PF99] ことができることが知られている．本論文で用いる補関数導出法は，この埋め込みの一種であると考えることができる．ただし，埋め込みで使用される余分な値として望ましい性質が，補関数導出による双方向化とプログラムの可逆化で以下のように異なる．

- 補関数導出による双方向化は，導出する逆方向変換が多くの更新を反映できることが望ましい．そのため，補関数はできるだけ情報を持ち運ばないものが望ましい（第 3 章）．

- プログラムを可逆にすることにおいては，オーバヘッドが少ないことが望ましい．そのため，余分な値のデータ量が少ないことや余分な値を計算する手間が少ないことが望ましい．

この二性質は互いに関連はしている．たとえば，少ない情報を持ち運ぶ補関数の返り値のほうが，値を表現するデータ量は本質的には少なくできる．しかし，二性質は等価ではない．たとえば，可逆計算では，関数 f から可逆な関数 f' を構成する手段として，$f'(x) = (f(x), x)$ とする手法が議論される [Ben89, PF99]．ところが，入力をそのまま埋め込むような補関数は，双方向化においては最悪のものである．また，可逆計算の研究は，チューリング機械 [Ben89,Lan61] やラムダ計算 [Abr05]，結合子論理 [PHW06]，フローチャート言語 [LD82] などほとんどが「計算」の上のもので,「プログラム」のようなより抽象度が高いものの上での議論はあまりなされていない [Bak92, PF99, MHT04b, YAG08a, YAG08b, YG07].

また，プログラム逆計算 [Dij78, Epp85, GK04, GK05, Gri81, NSS05] も本論文の手法と関連が深い．プログラム逆計算とは，関数を記述するプログラムからその逆関数を記述するプログラムを得ることである．いくつかの研究 [Dij78, GK05, Gri81, Epp85] では，事後条件を，システムが推定することやユーザが明示的に記述することで決定的な（deterministic）逆関数を得ている．また，いくつかの研究 [GK05,Epp85] は，組化をすることにより多返値関数を得ることで，より決定的な逆関数を導出している．本論文においても，補関数を得たあと逆方向変換を導出するのにプログラム逆計算が用いられる．そのとき，逆計算の前に，順方向変換と得られた補関数を組化するのは，彼らの手法と同様，より効率的に評価できる逆方向変換を得るためである．また，第 6 章において，組化により多返値関数を導入し，効果的に双方向化が行えるようにすることについても，彼らの手法からヒントを得ている．

Kawanaka と Hasoya は異なる形式の XML 文書間の相互変換を議論した [KH06]．彼らも XSugar と同様に関係を記述することにより，相互変換を達成する．彼らの目的は，文書間の相互変換を記述することであり，同期や全単射を記述することではない．そのため，彼らは，出力される文書が目的の形式に適合しているかどうかは検証するが，その相互変換が他にどのような性質を満たすかは議論しない．問題を特定することによって，彼らの枠組みは，一般的な論理プログラム言語で記述された XML 変換 [CF03] と比べ効率よく動作する [KH06]．

Meertens は，制約解消結合子言語を提案した [Mee98]．彼の枠組みにおいて，それぞれ A と B に属する要素間の関係 $R \subseteq A \times B$ についての制約解消器は，左の更新を右に反映させる意味 $A \times B \to B$ と右の更新を左に反映させる意味 $A \times B \to A$ の二つの意味を持つ．また，制約解消器のそれぞれの意味はさまざまな性質を満たすことが要求される．それらの性質を満たした制約解消器を結合子で合成した制約解消器もそれらの性質を満たす．

Xiong らは，グラフ上の多数の要素間の同期を実現するための言語を設計した [XHS$^+$08]．彼らは，複数の要素間の関係の記述することにより同期を達成している．また，いくつかの同期の上で望ましい性質が議論され，彼らの枠組みの上ではそれらの性質が成り立つ．ただし，彼らは，再帰的に定義される関係を扱っていない．

プログラムの双方向化

いくつかの研究は，プログラム双方向化を議論している．

Mu らは HaXML で記述されたプログラムを [MHT06]，Liu らは XQuery で記述されたプログラムを [LHT07]，順方向の意味を保ちつつ双方向結合子で表現されたプログラムに変換する手法を提案した．ここで，変換先に用いた双方向結合子言語は，Mu らは Inv [MHT04a] であり，Liu らは Hu らの結合子言語を Inv のタグ付けの概念を取り入れ XML 変換に特化させたものである．彼らは，変換元のそれぞれの言語要素に対し結合子を割り当てることにより，双方向化を行っている．しかし，彼らは，どのように割り当てたら振る舞いがよいのか，どのように割り当てたら効果的な双方向変換が得られるのかの議論をしておらず，その割り当てはアドホックなものである．

Voigtländer は，多相型を持つ順方向変換に対し，パラメトリシティ[Rey83] を利用して逆方向変換を導出する手法を提案した [Voi09]．リストからリストへの上の順方向変換を挙げて彼らの手法を説明する．順方向変換が $f :: \forall a.\ [a] \to [a]$ という型[2]を持っていたとする．関数 f の結果は，入力のリストに具体的にどんな値が入っているかに依存しない．そのため，同じ「形」のリストを入力として渡す限りにおいて，入力のリストのそれぞれの要素は，その位置に応じて出力のリストの特定の位置へと移動する．そのため，ソースのリスト各要素の「位置」に識別子を振り，それがビューのどの位置にいくのかを覚えておけば，ビュー上

[2] Haskell [Bir98] の記法を用いている．型 $[a]$ は要素が a 型である，リストの型である．

での更新をソース上へと反映することができる．すなわち，彼らは，補関数を導出している．この手法は，リストに限らず，位置を識別子で表現できるデータ構造一般に適用できる．また，識別子の割り振りを工夫することにより，同様の手法を $\forall a.\ \mathsf{Eq}\ a \Rightarrow [a] \rightarrow [a]$ という型の順方向変換や $\forall a.\ \mathsf{Ord}\ a \Rightarrow [a] \rightarrow [a]$ という型の順方向変換に対しても適用できる．彼の手法は，系統的であるものの，得られる逆方向変換はデータの「形」を変更することができない．たとえば，$f :: \forall a.\ [a] \rightarrow [a]$ に対し，彼の手法で得られる逆方向変換を用いて，ソースのリストの長さを変更することはできない．彼とは違い，我々は構造自身の変換を議論している．

2.2 双方向変換における複製の扱い

双方向変換において，複製の取り扱いは難しい．直観的には，複製によりデータが二度以上操作されると，どのタイミングで変更を逆方向変換により書き戻せばよいかわからなくなってしまうためである．

Foster らは文献 [FGM+05] においては，複製についてあまり議論しなかった．これは，彼らの目的が異なる形式の二つのデータの同期であることによる．彼らは，まず，それぞれのデータをソースと見なし，それぞれのデータに対する順方向変換で二つのデータで共有される情報をそれぞれビューへと取り出す．同期のため，この得られるビューはそれぞれ同じ形式をしている．その後，彼らは，得られた同じ形式の二つのビューの上で同期を行う．同期によりビューが変更されるので，その変更されたビューを逆方向変換することにより，異なる形式の二つのデータ間の同期が行える．この目的において，ビューに複製されたデータが含まれることは不自然である．

Hu らは，Foster らの枠組みに複製結合子を導入した [HMT04,HMT08]．彼らは，ビュー上で同期された要素を表現するのに複製を用いた．彼らの複製結合子は，逆方向変換において，複製されたどちらかの要素に変更があった場合にその変更をソースへと伝播する．これにより，ビュー上において，逆方向変換のち順方向変換を行うことで，複製された一方に加えられた変更をもう一方へと伝播することができる．たとえば，これによりファイルマネージャにおける対称なシンボリックリンクが実現できる [松田 05]．Mu らは，Hu らのアプローチをさらに押し進めて，データに変更をタグ付けすることによりさらに柔軟な逆方向変換が実現できることを示した [MHT04a]．第 6 章で述べる *unzip* の例のように，複製を結合子として定めた場合に，その逆方向の意味を逆関数で定義してしまうと，複製結合子を利用して構成された逆方向変換があまり更新を反映できなくなることがある．Hu らの複製結合子のような結合子を用いることでこの問題を解決できる．ところが，第 6 節で述べるように Hu らや Mu らの複製結合子は我々の意味での双方向性を満足しない．

また，Foster らは文献 [FPP08] において，商データ（quotient）を考えることにより一部の Hu らの複製を商データの上での双方向性を保ったまま扱えることを示した．ただし，こ

の手法では *unzip* などの変換に効果的な逆方向変換を定めるのは難しい．

Voigtländer も複製を扱っている [Voi09]．彼の枠組みでは，多相型により，複製された値が走査されないことが保証される．

いくつかの研究は補関数の導出において複製を考慮した導出方法を与えている [LLSV01, LV03]．Laurent ら [LLSV01] や Lechtenbörger と Vossen [LV03] は関係データベース上の関係代数で表現されるクエリに対し補関数の導出手法を与えた．補関数導出の立場において，ある関数 f と別の関数 g が同じデータを操作するということは，そのデータに対し，f が持ち運ぶ情報を g が持ち運ぶ情報の両方が得られることになる．そのため，彼らの枠組みでは，$\langle f, g\rangle$ の補関数は，f の補関数の持ち運ぶ情報と g の補関数の持ち運ぶ情報の共有する情報に基づいて与えられる．大雑把には，彼らの補関数は元の関係データベース（組の集合）の部分集合を返すため，$\langle f, g\rangle$ の補関数は，f と g の補関数をそれぞれ f^{c} と g^{c} とすると，$\langle f, g\rangle^{\mathrm{c}}(x) = f^{\mathrm{c}}(x) \cap g^{\mathrm{c}}(x)$ で与えられる．それに対し，入力が集合の場合と異なり，入力が木や生垣の場合には返り値を合成可能にするのは難しく，本論文ではそのようなアプローチは採用しない．また，一般的には，$\langle f, g\rangle$ の補関数は，f の補関数の持ち運ぶ情報と g の補関数の持ち運ぶ情報の共有する情報に基づいて与えられるとは限らないことに注意する．

2.3　双方向変換の定式化

本論文において，双方向変換とは，あるデータを別のデータに変換する順方向変換と，変換後のデータの上でなされた更新を元のデータ上の反映する逆方向変換の組である．ここで，順方向変換はただの関数である．しかし，逆方向変換の定式化には主に次の二通りある．一つの定式化は，逆方向変換を「更新の反映を行う」と見る状態主導（state-based）の定式化である．すなわち，順方向変換 $f :: S \to V$ に対し，逆方向変換

$$\rho :: S \times V \to S$$

は，更新反映前のソースと更新後のビューを取り，更新反映後のソースを返す．もう一つの定式化は，逆方向変換を「更新の翻訳を行う」と見る操作主導（operation-based）定式化である．すなわち，順方向変換 $f :: S \to V$ に対し，逆方向変換

$$\tau :: U(V) \to U(S)$$

は，ビュー上の更新操作 $u \in U(V)$ をソース上の更新操作 $\tau(u) \in U(S)$ に対応づける．ここで，$U(X)$ は集合 X 上の更新操作の集合であり，それぞれの更新操作 $u \in U(X)$ は，関数としての意味 $[\![u]\!] :: X \to X$ を持つ．

ビュー更新においては，一部 [Heg90,LV03] を除き，操作主導の枠組みにおいて議論されている [BS81,DB82,GPZ88,WR04,BDH04,Mas84]．これは，関係データベースにおいては，管理システムにより，どのようにデータベースの更新を行うのかが規定されているためだと考え

られる．双方向変換の枠組みにおいては，文献 [FGM+05,HMT04,BFP+08,FPP08,BMS08] は状態主導の定式化を採用していて，文献 [MHT04a] は操作主導の定式化を採用している．

一般には，操作主導の双方向変換のほうが，操作に対する情報を逆方向変換が利用できるため，より多様できめの細かい双方向変換を達成できる．たとえば，操作主導のもとでは，リストの先頭要素を削除した後に新たな先頭要素を付け加えるという更新と，リストの先頭要素を別の要素へと変更するという更新は区別される．それに対し，状態主導の元では，このそれぞれの更新は同じ結果（状態）になるため区別されない．また，一般には，操作主導の双方向変換では双方向変換システムが「操作」の情報を手にいれられることを前提においているため，適用範囲が状態主導よりも限定される．状態主導の双方向変換においては，ユーザは任意のアプリケーションを用いてビューを更新できるが，操作主導では，ビューを更新するアプリケーションは双方向変換システムに「どのような操作で更新されたか」の情報を渡さなければならない．

二つの定式化が一致する場合がある．今，ソースとビュー X それぞれにおいて，更新操作の集合 $U(X)$ が任意の X の要素から任意の X の要素への更新を含むとする．すなわち，$\forall x_1, x_2 \in X, \exists u \in U(X).\ [\![u]\!](x_1) = x_2$ であるとする．このとき，もし操作主導の逆方向変換 τ が更新操作の意味にのみ依存する，すなわち，

$$[\![u]\!] = [\![u']\!] \Rightarrow [\![\tau(u)]\!] = [\![\tau(u')]\!]$$

であるならば，同等な反映式の逆方向変換 ρ を

$$\rho(s, v) = [\![\tau(u)]\!](s) \quad \text{ただし } u \text{ は } [\![u]\!](f(s)) = v \text{ となる更新操作}$$

と定められる．逆に状態主導の逆方向変換 ρ から，次により，更新操作の意味のみに依存する操作主導の逆方向変換 τ を与えられる．

$$\tau(u) = u' \quad \text{ただし } u' \text{ は } [\![u']\!](s) = \rho(s, [\![u]\!](v)) \text{ となる更新操作}$$

Gottlob らは，操作主導の逆方向変換 τ が更新操作の意味にのみ依存するための条件を議論した [GPZ86,GPZ88]．この条件は，補関数に基づいて逆方向変換が定義されているという条件よりも弱い．具体的には，彼らの逆方向変換が対応する状態主導の逆方向変換は，第 3 章における許容性と合成可能性を満たすが可逆性を満たすとは限らない．そのため，補関数に基づく双方向考える上では，逆方向変換の関数としての意味のみを議論する場合において，状態主導で議論することと操作主導で議論することに差はない．なお，文献 [FGM+05] で議論される双方向変換は状態主導であるが，彼らの枠組みで定義される双方向変換が合成可能性を満たすとは限らない．これは，Gottlob らが，$U(X)$ が更新操作の結合「$\hat{\circ}$」を含むと仮定しており，その意味と翻訳は以下で与えられることを仮定しているためである．

$$[\![u_1 \mathbin{\hat{\circ}} u_2]\!] = [\![u_1]\!] \circ [\![u_2]\!]$$

$$\tau(u_1 \mathbin{\hat{\circ}} u_2) = \tau(u_1) \mathbin{\hat{\circ}} \tau(u_2)$$

第3章　補関数に基づく双方向化

本章では，補関数に基づく双方向化 [BS81, Heg90, Heg04, CP84, LLSV01, LV03] について述べる．後述する我々のプログラムの双方向化手法は，補関数に基づく双方向化手法に基づく．

補関数に基づく双方向化は「副作用のない」双方向変換を構成する．たとえば，第 1 章においては，研究室のメンバーリスト

$s =$
```
<members>
  <student>Matsuda</student>
  <professor>Hu</professor>
  <professor>Takeichi</professor>
</members>
```

から順方向変換 *students* により学生を抽出し，以下の学生のみのリストを構築していた．

$students(s) =$
```
<members>
  <student>Matsuda</student>
</members>
```

変換元のデータには，変換先のデータの構築に関係のない「情報」が含まれる．たとえば，*students* により以下のどちらのメンバーリストから学生を抽出しても，上のリストを得る．

```
<members>
  <professor>Hu</professor>
  <student>Matsuda</student>
  <professor>Takeichi</professor>
</members>
```

```
<members>
  <student>Matsuda</student>
  <professor>Hu</professor>
</members>
```

よって，これらのデータ間の「違い」は，*students* の変換結果には関係がない．双方向変換を用いることで，研究室のメンバーリストから学生を抽出するだけではなく，抽出された学生のリストを更新することにより研究室のメンバーリストを更新することができる．更新の反映において，副作用のないこと，つまり，*students* の変換結果に関係のない「情報」が更新の反映によって変更されないことが望ましい．順方向変換 *students* に対する補関数は，*students* の変換結果に関係のない「情報」を包含する．たとえば，ある *students* に対する

補関数は，s に対し以下を返す．

```
<members>
  
  <professor>Hu</professor>
  <professor>Takeichi</professor>
</members>
```

上で　は，直観的には，変換 *students* によって抽出された学生 <student>Matsuda</student> に対する更新が書き戻される場所を表現している．補関数は *students* の変換結果に関係ない「情報」を抽出しているため，補関数値を不変にすることで，学生のみからなるリスト上の更新を副作用なく，元のメンバーリスト上に反映することができる．

本章では，まず双方向変換を定義し，我々が議論の対象とする「振る舞いのよい」双方向変換について定義する．その後，補関数に基づく双方向化 [BS81] について述べ，補関数に基づく双方向化が振る舞いのよい双方向変換を定めるのに必要十分であること [BS81]，つまり,「振る舞いのよい」双方向変換と「副作用のない」双方向変換は等価であることを述べる．

3.1　双方向変換

大雑把に言えば，双方向変換は，**ソース**と呼ばれるデータを**ビュー**と呼ばれるデータへ変換する順方向変換とビューの上でなされた更新をソースへと反映する逆方向変換の二つの変換の組である．

双方向変換の形式的な定義を述べる前に，まず，更新について定める．

定義 3.1 (更新)**.** 集合 X 上の**更新**とは，更新前の値 $x_1 \in X$ と更新後の値 $x_2 \in X$ の組 $(x_1, x_2) \in X \times X$ である．

すなわち，我々は任意の値から任意の値への更新を考える．更新を通常の組と区別するために，更新 (x_1, x_2) を $x_1 \rightarrowtail x_2$ と書く．

定義より，集合 X の上の更新の集合は，以下の性質を持つ．

- 全ての $x \in X$ に対し，恒等更新 $x \rightarrowtail x$ が存在する．
- 全ての $x_1, x_2, x_3 \in X$ に対し，更新 $x_1 \rightarrowtail x_2$ と $x_2 \rightarrowtail x_3$ の結合 $x_1 \rightarrowtail x_3$ が存在する．
- 全ての更新 $x_1 \rightarrowtail x_2$ は更新 $x_2 \rightarrowtail x_1$ により取り消すことができる．すなわち，$x_1 \rightarrowtail x_2$ と $x_2 \rightarrowtail x_1$ との結合が恒等更新 $x_1 \rightarrowtail x_1$ になる．

3.1.1 定義

次に，双方向変換の厳密な定義を述べる．まず，双方向変換を構成する二つの変換の一方である順方向変換について定める．

定義 3.2 (順方向変換)**.** ソース S からビュー V への**順方向変換**は，S から V への（全域）関数である．

たとえば，以下の関数 $add :: \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ は自然数の組から自然数への順方向変換である．

$$add(x, y) = x + y$$

特に断わらない限り，我々は順方向変換として全域関数のみを考える．

次に，双方向変換を構成するもう一方の変換である逆方向変換について定める．

定義 3.3 (逆方向変換)**.** 順方向変換 $f :: S \to V$ に対する**逆方向変換** ρ は，以下を満たす関数 $\rho :: S \times V \to S$ である．

$$\forall s \in S, \forall v \in V.\ \rho(s, v){\downarrow} \Rightarrow f(\rho(s, v)) = v$$

逆方向変換 $\rho :: S \times V \to S$ は，更新前のソースと更新後のビューとを引数に取り，更新反映後のソースを返す．たとえば，以下の関数 $add_{\mathrm{B}} :: (\mathbb{N} \times \mathbb{N}) \times \mathbb{N} \to \mathbb{N} \times \mathbb{N}$ は，前述の関数 $add :: \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ に対する逆方向変換の一つである．

$$add_{\mathrm{B}}((s_1, s_2), v) = (s_1, v - s_1) \text{ if } v \geq s_1$$

逆方向変換 add_{B} は，ビューに対する更新をソースの第二要素を変更することにより反映する．たとえば，$add(3, 7) = 10$ であるが，ビュー上で 10 を 6 に変更すると $add_{\mathrm{B}}((3, 7), 6) = (3, 3)$ となり，ソースの第二要素が 7 から 3 へと変更される．ここで，更新後のソースのビューが，$add(3, 3) = 6$ と更新後のビューと等しくなっていることに注意する．

順方向変換 f に対する逆方向変換 ρ は，更新 $f(s) \rightarrowtail v$ を更新 $s \rightarrowtail \rho(s, v)$ へと更新する．ただし，ここで更新が反映されたソース $\rho(s, v)$ に順方向変換を適用したもの $f(\rho(s, v))$ は，更新後のビュー v と等しくなければならない．別の言い方をすると，ソース $s \in S$ とそのビュー $f(s) \in V$ に対し，u を更新 $f(s) \rightarrowtail v$ とおき u' を更新 u の翻訳結果 $s \rightarrowtail \rho(s, v)$ とおくと，以下の図式が可換となる．

$$\begin{array}{ccc} V & \xrightarrow{u} & V \\ {\scriptstyle f}\uparrow & & \uparrow{\scriptstyle f} \\ S & \xrightarrow[u']{} & S \end{array}$$

一般には，逆方向変換は部分関数である．つまり，逆方向変換が反映できない更新が存在する．順方向変換 f に対する逆方向変換 ρ について，$\rho(s, v){\downarrow}$ である場合，逆方向変換 ρ は

更新 $f(s) \rightarrowtail v$ をソースへと反映できる．対し，$\rho(s,v) = \perp$ である場合，逆方向変換 ρ はビュー上の更新 $f(s) \rightarrowtail v$ をソースへ反映することを許可しない．たとえば，前述の add に対する逆方向変換 add_{B} について，もし，ソースが元々$(3,7)$ であった場合，ビューにおける $add(3,7) = 10$ から 2 への更新は許可されない．これは，$add_{\mathrm{B}}((3,7),2) = \perp$ であるためである．また，一般には，ある順方向変換に対する逆方向変換は一意に定まるわけではない．そして，どの逆方向変換を選ぶかによって，双方向変換がどのように振る舞うかが決まる．たとえば，以下の関数 ρ も前述の順方向変換 add の逆方向変換である．

$$\rho((s_1,s_2),v) = (s_1,s_2) \text{ if } add(s_1,s_2) = v$$

この逆方向変換は，恒等更新以外ソースへと反映することができない．すなわち，この逆方向変換を用いても，ビュー上の変更を通してソースを変更することはできない．

3.1.2 振る舞いのよい双方向変換

前述の通り，順方向変換に対する逆方向変換は複数存在する．そこで，順方向変換に対する逆方向変換は，**振る舞いのよい**，つまり更新反映を通してソースとビューの間の何らかの整合性を保つことが望まれる．我々は，この「振る舞いのよさ」の記述として，文献 [BS81] で議論された保守的な**双方向性**の定義を用いる．

双方向性の厳密な定義の前に，我々は簡潔に双方向性について述べる．我々の要求する双方向性は，それぞれ以下に述べる三つの双方向変換に対する要求に対応する．

一つ目の要求は，もしビューが更新されることがなければ，逆方向変換はソースを更新してはならないということである．この要求を説明するために，以下の順方向変換 $fst :: \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ を考える．

$$fst(x,y) = x$$

すなわち，順方向変換 fst は組の第一要素を返す．以下の関数 ρ_1 は，fst の逆方向変換の一つである．

$$\rho_1((x,y),v) = (v,1)$$

逆方向変換 ρ_1 は，あまり望ましいものではない．なぜならば，以下のように，逆方向変換 ρ_1 はビューに全く更新がない場合においても，ソースを変更する場合があるためである．

$$\rho_1((2,3), fst(2,3)) = \rho_1((2,3),2) = (2,1) \neq (2,3)$$

逆方向変換 ρ_1 のこの振る舞いは双方向変換ユーザの直感に反する．

二つ目の要求は，ある値の更新の反映結果は，それまでの更新反映の履歴に依存してはならないということである．この要求を説明するため，以下の順方向変換 $f :: \{\mathsf{a},\mathsf{b},\mathsf{c},\mathsf{d}\} \to$

$\{\mathsf{x},\mathsf{y},\mathsf{z}\}$ と f に対する逆方向変換 ρ_3 を考える．

$$f(x) = \begin{cases} \mathsf{x} & \text{if } x = \mathsf{a} \\ \mathsf{y} & \text{if } x = \mathsf{b} \\ \mathsf{z} & \text{otherwise} \end{cases}$$

$$\begin{aligned} \rho_2(\mathsf{a},\mathsf{y}) &= \mathsf{b} \\ \rho_2(\mathsf{a},\mathsf{z}) &= \mathsf{d} \\ \rho_2(\mathsf{b},\mathsf{z}) &= \mathsf{c} \end{aligned}$$

ここでビュー上の更新 $\mathsf{x} \rightarrowtail \mathsf{z}$ と，更新 $\mathsf{x} \rightarrowtail \mathsf{y}$ と更新 $\mathsf{y} \rightarrowtail \mathsf{z}$ との結合を考える．この二つの更新は，ビュー上においては同じ意味を持つ．しかし，もし更新の結合の反映時に一つの更新毎に反映を行ったとすると，以下のようにそれぞれのビュー上の更新は異なった結果を生じる．

$$\rho_2(\mathsf{a},\mathsf{z}) = \mathsf{d} \neq \rho_2(\rho_2(\mathsf{a},\mathsf{y}),\mathsf{z}) = \rho_2(\mathsf{b},\mathsf{z}) = \mathsf{c}$$

このような ρ_2 の下では，たとえばもし更新がプログラムなどで記述された場合に，ビュー上で同じ意味を持つ更新プログラムが，ソースの上で異なる意味を持つ更新プログラムになりうる．これは，それぞれの更新の記述や取り扱われ方によってそれぞれの更新の反映結果が変わりうるためである．逆に，二つ目の要求が満たされた場合，ビュー上で等しい意味を持つ更新は，記述によらず同じ意味を持つソース上の更新へと翻訳される [GPZ88]．さらには，我々はある一つの更新を細かな更新の結合に分離して考えることができる．ここで，たとえば個々の小さな更新の反映が効率的に実装されていた場合に，大きな更新の反映も効率的に行うことができる．また，ソースを考えることなく，ビュー上のそれぞれの小さな更新を自由にスケージュリングすることもできる [GPZ88]．

三つ目の要求は，双方向変換のユーザは望まぬタイミングで逆方向変換を実行してしまうことがあるため，ソースにアクセスすることなくその望まぬ逆方向変換の実行を取り消すことができるようにするということである．なお，一つ目と二つ目の要求は満たされているが三つめの要求が満たされていない場合，あるビュー v_1 から別のビュー v_2 への更新は反映できるにもかかわらず，ビュー v_2 から v_1 への更新は反映できない場合がある．また，この要求は，ユーザがビューに現れえないソースの情報を更新することを禁止する．

定義 3.4 (双方向性，振る舞いのよさ)**.** 順方向変換 $f :: S \to V$ とその逆方向変換 $\rho :: S \times V \to S$ を考える．順方向変換 f に対し，ρ の**振る舞いがよい**とは，f と ρ が，任意の $s \in S$ と $v, v' \in V$ に対し，以下の三つの双方向性を全て満たすことである．

許容性:	$\rho(s, f(s)) = s$
合成可能性:	$\rho(s,v)\!\downarrow \wedge\, \rho(\rho(s,v),v')\!\downarrow \Rightarrow \rho(\rho(s,v),v') = \rho(s,v')$
可逆性:	$\rho(s,v)\!\downarrow \Rightarrow \rho(\rho(s,v),f(s)) = s$

三つの双方向性は，それぞれ許容性，合成可能性，可逆性に対応する．許容性は，もしビューに変更がなければ逆方向変換はソースを変更することがないことを表す．合成可能性は，更新の翻訳がそれまでの翻訳の履歴によらないことを表す．可逆性は，全ての翻訳された更新はある翻訳された更新により取り消すことができることを表す．

振る舞いのよい逆方向変換は，更新の上の代数構造を保存する．更新の結合を $\circ$ で書くと，集合 X 上の更新は亜群（groupoid）$(X \times X, \circ, _^{-1})$ を成す．ただし，演算 $\circ$ は更新の結合 $x_1 \rightarrowtail x_2 \circ x_2 \rightarrowtail x_3 = x_1 \rightarrowtail x_3$ を表し，演算 $_^{-1}$ は更新の取り消しを行う更新 $(x_1 \rightarrowtail x_2)^{-1} = x_2 \rightarrowtail x_1$ を表す．順方向変換 f に対し振る舞いのよい逆方向変換 ρ の定める翻訳は，ビュー上の更新の亜群 $(V \times V, \circ, _^{-1})$ をソース上の更新の亜群 $(S \times S, \circ, _^{-1})$ に移す準同型写像である．任意のビュー $v = f(s) \in V$ に対し，$v \rightarrowtail v$ は $s \rightarrowtail s$ に移ることが許容性により保証される（単位元の保存）．また，任意のビュー $v = f(s), v', v'' \in V$ に対し，ρ がビュー上の更新 $f(s) \rightarrowtail v'$ と $v' \rightarrowtail v''$ をそれぞれソース上の更新 $s \rightarrowtail s'$ と $s' \rightarrowtail s''$ に移す場合には，ビュー上の更新の結合 $f(s) \rightarrowtail v'' = f(s) \rightarrowtail v' \circ v' \rightarrowtail v''$ をソース上の更新の結合 $s \rightarrowtail s'' = s \rightarrowtail s' \circ s' \rightarrowtail s''$ に移すことが合成可能性により保証される（二項間の演算の保存），さらに，任意のビュー $v = f(s), v' \in V$ に対し，ρ がビュー上の更新 $f(s) \rightarrowtail v$ をソース上の更新 $s \rightarrowtail s'$ に移す場合，ビュー上の更新 $(f(s) \rightarrowtail v)^{-1} = v \rightarrowtail f(s)$ をソース上の更新 $(s \rightarrowtail s')^{-1} = s' \rightarrowtail s$ に移すことが可逆性により保証される（逆元の保存）．

定義 3.4 の定める双方向性は保守的なものである．そのため，逆方向変換の中には，動作が直観的であるものの双方向性を満たさないものがある．Gottlob らは，許容性，合成可能性を満たすものの可逆性を満たさない双方向変換について議論した [GPZ88, GPZ86]．Keller は自然な逆方向変換でありながら，合成可能性や可逆性を満たさないものがあることを指摘した [Kel87]．多くのビュー更新の枠組み [DB82, Mas84, Lan90, BDH04, WR04] も許容性や合成可能性を満たさない．

3.2 補関数に基づく双方向化

双方向化とは，順方向変換から振る舞いのよい逆方向変換を適切に与える手法である．本節では，Bancilhon と Spyratos による，補関数に基づく双方向化について述べる [BS81]．

大雑把に言えば，補関数に基づく双方向化は「副作用のない」更新反映を定式化したものである．ここで「副作用がない」とは，逆方向変換が，ソース内におけるビューの構築に関係のない部分を変更しないことである．この，ソース内におけるビューの構築に関係のない部分を表現したものが**補関数**である．もし，逆方向変換を，補関数の返り値を不変にするように定めたのならば，我々は「副作用のない」逆方向変換を実現することができる．

3.2.1 補関数

定義 3.5 (補関数)**.** 関数 $f :: X \to Y$ に対し，関数 $g :: X \to Z$ が**補関数**であるとは，f と g を組にした関数 $\langle f, g\rangle :: X \to Y \times Z$ が単射であることである．

補関数と順方向変換と組にして単射になることにより，ビューの構築に関係のない部分の情報を補関数が含むことが保証されている．

直観的には，関数 f に対する補関数 g は，関数 f による変換を通して失われたソースの情報を全て保持する．たとえば，前述の順方向変換 add に対し，変換結果 $add(x, y) = 1$ からは変換前のソース (x, y) が $(0, 1)$ であったのか $(1, 0)$ であったのかを知ることができない．関数 f の補関数は，この $(0, 1)$ と $(1, 0)$ のような，f によって同じ値になるソースを区別する．形式的には以下が言える．

定理 3.1 ([BS81])**.** 関数 $g :: X \to Z$ が関数 $f :: X \to Y$ の補関数であるとき，またそのときに限り以下が成り立つ．

$$\forall x, y \in X.\ x \neq y \wedge f(x) = f(y) \Rightarrow g(x) \neq g(y) \qquad \square$$

3.2.2 補関数値不変に基づく逆方向変換

もし，順方向変換 f が単射関数であるならば，逆方向変換は f の逆関数 f^{-1} を用いて $\rho(_, v) = f^{-1}(v)$ で定まる．この逆関数により定まる逆方向変換は振る舞いがよい．しかし，一般には，順方向変換は単射ではない．順方向変換 $f :: S \to V$ に対する補関数 $g :: S \to V'$ を用いることで，単射関数 $\langle f, g\rangle$ を構成できる．ここで，ソース $s \in S$ とそのビュー $v = f(s) \in V$ に対し，更新 $u = f(s) \rightarrowtail v'$ の翻訳 u' を，補関数値を不変にするように，以下の可換図式により定められる．

$$\begin{array}{ccc} V & \xrightarrow{u} & V \\ {\scriptstyle f}\uparrow & & \uparrow{\scriptstyle f} \\ S & \xrightarrow[u']{} & S \end{array} \quad\Longleftrightarrow\quad \begin{array}{ccc} V \times V' & \xrightarrow{u \times \mathrm{id}} & V \times V' \\ {\scriptstyle \langle f,g\rangle}\uparrow & & \uparrow{\scriptstyle \langle f,g\rangle}\ \downarrow{\scriptstyle \langle f,g\rangle^{-1}} \\ S & \xrightarrow[u']{} & S \end{array}$$

形式的には，我々は以下のように補関数値不変の逆方向変換を定める．

定義 3.6 (補関数値不変の逆方向変換)**.** 順方向変換 f とその補関数 g について，補関数値不変の逆方向変換 $\mathsf{reflect}_{f,g}$ を以下で定める．

$$\mathsf{reflect}_{f,g}(s, v) = \langle f, g\rangle^{-1}(v, g(s)) \qquad \text{(RFL)}$$

順方向変換 f とその補関数 g に対し，関数 $\mathsf{reflect}_{f,g}$ は確かに逆方向変換になっている．任意のソース $s \in S$ とビュー $v \in V$ に対し，もし $\mathsf{reflect}_{f,g}(s,v)\!\downarrow$ ならば以下が成り立つことが確認できる．

$$\langle f,g\rangle(\mathsf{reflect}_{f,g}(s,v)) = \langle f,g\rangle(\langle f,g\rangle^{-1})(v,g(s)) = (v,g(s))$$

これは上の可換図式に対応する．また，上の等式は同時に，$\mathsf{reflect}_{f,g}$ が確かに補関数値不変であることを意味する．

定理 3.2 ([BS81])**.** 順方向変換 $f :: S \to V$ とその補関数 $g :: S \to V'$ に対し，補関数値不変の逆方向変換 $\mathsf{reflect}_{f,g}$ は更新の反映を通して補関数値を変更することがない．すなわち

$$\forall s \in S, v \in V.\ \mathsf{reflect}_{f,g}(s,v)\!\downarrow \Rightarrow g(\mathsf{reflect}_{f,g}(s,v)) = g(s)$$

である．

また，逆関数を用いて定められているため，補関数値不変の逆方向変換は振る舞いがよい．

定理 3.3 ([BS81])**.** 順方向変換 $f :: S \to V$ とその補関数 $g :: S \to V'$ に対し，補関数値不変の逆方向変換は振る舞いがよい．

証明. 三つの双方向性（許容性，可逆性，合成可能性）をそれぞれ証明する．

許容性　以下より，許容性が成り立つ．

$$\mathsf{reflect}_{f,g}(s,f(s)) = \langle f,g\rangle^{-1}(f(s),g(s)) = \langle f,g\rangle^{-1}(\langle f,g\rangle(s)) = s$$

可逆性　今，ソース s, s' とビュー v に対し，$\mathsf{reflect}_{f,g}(s,v) = s' \neq \bot$ であったとする．このとき，以下が成り立つ．

$$\begin{array}{lll}
\mathsf{reflect}_{f,g}(s',f(s)) & = & \{\ \text{定義}\ \} \\
 & & \langle f,g\rangle^{-1}(f(s),g(s')) \\
 & = & \{\ \text{定理 3.2 より}\ \mathsf{reflect}_{f,g}(s,v) = s' \Rightarrow g(s') = g(s)\ \} \\
 & & \langle f,g\rangle^{-1}(f(s),g(s)) \\
 & = & \langle f,g\rangle^{-1}(\langle f,g\rangle(s)) \\
 & = & s
\end{array}$$

これにより，可逆性が示された．

合成可能性 今，ソース s, s' とビュー v, v' について，$\mathsf{reflect}_{f,g}(s, v) = s' \neq \bot$ であり $\mathsf{reflect}_{f,g}(s', v')\!\downarrow$ であったとする．このとき，以下が成り立つ．

$$
\begin{array}{rcl}
\mathsf{reflect}_{f,g}(s, v') & = & \{\text{ 定義 }\} \\
 & & \langle f, g\rangle^{-1}(v', g(s)) \\
 & = & \{\text{ 定理 3.2 より } \mathsf{reflect}_{f,g}(s, v) = s' \Rightarrow g(s) = g(s') \ \} \\
 & & \langle f, g\rangle^{-1}(v', g(s')) \\
 & = & \mathsf{reflect}_{f,g}(s', v') = \mathsf{reflect}_{f,g}(\mathsf{reflect}_{f,g}(s, v), v')
\end{array}
$$

これにより合成可能性が示された． □

補関数に基づく双方向化において，補関数の返り値にそのものはあまり重要ではなく，補関数がどの値とどの値を同じ値に移すのか，すなわち，補関数の定める同値関係が重要である．

定義 3.7 (関数値を法とした同値関係)**.** 関数 f に対し，**関数 f の値を法とした同値関係** $\equiv_f$ を以下で定める．

$$x \equiv_f y \Leftrightarrow f(x) = f(y)$$ □

定理 3.4 ([BS81])**.** 関数 g と h はともに関数 f の補関数であるとする．このとき，

$$(\equiv_g) = (\equiv_h) \Leftrightarrow \mathsf{reflect}_{f,g} = \mathsf{reflect}_{f,h}$$

が成り立つ．

定理 3.4 を示すために，以下の補題を用いる．

補題 3.1. 関数 f とその補関数 g について，

$$\forall s, s' \in S.\ s \equiv_g s' \Leftrightarrow \mathsf{reflect}_{f,g}(s, f(s')) = s'$$

が成り立つ．

証明. まず左から右 $\Rightarrow$ を示し，次に右から左 $\Leftarrow$ を示す．

($\Rightarrow$) 今，ソース $s, s' \in S$ に対し，$s \equiv_g s'$ であるとする．すなわち，$g(s) = g(s')$ である．このとき，

$$\mathsf{reflect}_{f,g}(s, f(s')) = \langle f, g\rangle^{-1}(f(s'), g(s)) = \langle f, g\rangle^{-1}(f(s'), g(s')) = s'$$

となる．

($\Leftarrow$)　定理 3.2 より，

$$\mathsf{reflect}_{f,g}(s, f(s')) = s' \Rightarrow g(s) = g(s') \Leftrightarrow s \equiv_g s'$$

となる． □

補題 3.1 から，定理 3.4 は明らかである．
また，補関数定義は以下のように同値関係を用いて書き直すこともできる．

定理 3.5 ([BS81])**.** 関数 $g : X \to Z$ が関数 $f : X \to Y$ の補関数であるとき，またそのときに限り以下が成り立つ．

$$(\equiv_f) \cap (\equiv_g) = (\equiv_{\mathrm{id}})$$

証明. 式 $(\equiv_f) \cap (\equiv_g) = (\equiv_{\mathrm{id}})$ は以下のように書き換えられる．

$$f(x) = f(y) \wedge g(x) = g(y) \Rightarrow x = y$$

これは式 $x \neq y \wedge f(x) = f(y) \Rightarrow g(x) \neq g(y)$ と等価である．定理 3.1 から，$x \neq y \wedge f(x) = f(y) \Rightarrow g(x) \neq g(y)$ であることと，g が f の補関数であることとは等価である． □

前述の通り，補関数値不変の逆方向変換は振る舞いがよい．逆に，振る舞いのよい逆方向変換に対し，順方向変換は，それと等価な逆方向変換を定める補関数を持つ．

定理 3.6 ([BS81])**.** 順方向変換 $f : S \to V$ とその振る舞いのよい逆方向変換 $\rho : S \times V \to S$ に対し，$\rho = \mathsf{reflect}_{f,g}$ となる f の補関数 g が存在する．

証明. 関係 $\sim$ を以下で定める．

$$x \sim y \Leftrightarrow \rho(x, f(y)) = y \neq \bot.$$

我々はまず，$\sim$ が同値関係であることを示す．

反射率

$$\text{許容性} \Leftrightarrow \forall x \in S.\ \rho(x, f(x)) = x \Leftrightarrow \forall x \in S.\ x \sim x$$

対称律

$$\begin{aligned}
\text{可逆性} \quad &\Leftrightarrow \quad \forall x \in S, v \in V.\ \rho(x, v){\downarrow} \Rightarrow \rho(\rho(x, v), f(x)) = x \\
&\Leftrightarrow \quad \forall x, y \in S, v \in V.\ \rho(x, v) = y \neq \bot \Rightarrow \rho(y, f(x)) = x \\
&\Leftrightarrow \quad \{\ \rho \text{ は } f \text{ の逆方向変換}\ \} \\
&\qquad\quad \forall x, y \in S.\ \rho(x, f(y)) = y \neq \bot \Rightarrow \rho(y, f(x)) = x \\
&\Leftrightarrow \quad \forall x, y \in S.\ x \sim y \Rightarrow y \sim x
\end{aligned}$$

推移律

合成可能性

$$\begin{array}{ll} & \\ \Leftrightarrow & \forall x \in S, v, v' \in V.\ \rho(x,v)\downarrow \wedge \rho(\rho(x,v),v')\downarrow \Rightarrow \rho(\rho(x,v),v') = \rho(x,v') \\ \Leftrightarrow & \forall x, y, z \in S, v, v' \in V.\ \rho(x,v) = y \neq \perp \wedge \rho(y,v') = z \neq \perp \Rightarrow \rho(x,v') = z \\ \Leftrightarrow & \{\ \rho \text{ は } f \text{ の逆方向変換 } \} \\ & \forall x, y, z \in S.\ \rho(x,f(y)) = y \neq \perp \wedge \rho(y,f(z)) = z \neq \perp \Rightarrow \rho(x,f(z)) = z \\ \Leftrightarrow & \forall x, y, z \in S.\ x \sim y \wedge y \sim z \Rightarrow x \sim z \end{array}$$

ここで，関数 g を以下で定める．

$$g(s) = [s]_\sim$$

ただし，ここで $[x]_\sim$ は x の関係 $\sim$ による同値類を表す．次に我々は g が f の補関数であること示し, その後 $\rho = \mathsf{reflect}_{f,g}$ を示す.

第一に，我々は g が f の補関数であること，すなわち，

$$f(x) = f(y) \wedge g(x) = g(y) \Rightarrow x = y$$

を示す．今，ソース $x, y \in S$ が，$f(x) = f(y) \wedge g(x) = g(y)$ 満たしているとする．ここで，$g(x) = g(y)$ であるため，$\rho(x, f(y)) = y$ が成り立つ．また，$f(x) = f(y)$ ゆえに，$\rho(x, f(x)) = y$ も成り立つ．許容性より $\rho(x, f(x)) = x$ である．よって，$x = y$ が成り立つ.

第二に，我々は以下を示す.

$$\forall x, y \in S\ .\rho(x, f(y)) = y \Leftrightarrow \mathsf{reflect}_{f,g}(x, f(y)) = y.$$

これは，任意のソース $x, y \in S$ について $\rho(x, f(y)) = y \Leftrightarrow g(x) = g(y)$ が成り立つため，補題 3.1 より簡単に示される．　□

別証．定理 3.6 の別の証明を示す．以下の高階関数 $h : S \to (V \to V)$ を考える．

$$h(x) = h' \text{ ただし } h'(v) = \rho(x, v)$$

定理 3.6 を，関数 h が f の補関数であることと $\rho = \mathsf{reflect}_{f,h}$ であることを示すことにより，

示す．以下が成り立つため，証明の本筋は上の証明と同じである．

$$\begin{array}{rcl} h(x) = h(y) & \Leftrightarrow & \forall z \in S, v \in V.\ \rho(x,v) = z \Leftrightarrow \rho(y,v) = z \\ & \Leftrightarrow & \{\ \rho \text{ は } f \text{ の逆方向変換 } \} \\ & & \forall z \in S.\ \rho(x, f(z)) = z \Leftrightarrow \rho(y, f(z)) = z \\ & \Leftrightarrow & \{ \sim \text{を } a \sim b \Leftrightarrow \rho(a, f(b)) = b \text{ により定める } \} \\ & & \forall z \in S.\ x \sim z \Leftrightarrow y \sim z \\ & \Leftrightarrow & \{ \sim \text{は同値関係 } \} \\ & & x \sim y \\ & \Leftrightarrow & \{\ g \text{ を } g(x) = [x]_\sim \text{ により定める } \} \\ & & g(x) = g(y) \end{array}$$

この証明は，もし ρ が計算可能ならば，f は ρ と等価な逆方向変換を定める計算可能な補関数を持つことも示している．□

3.2.3　補関数の優劣

一般に，関数 f に対する補関数は唯一ではない．たとえば，以下の四つの関数は，どれも自然数同士を加算する $add :: (\mathbb{N} \times \mathbb{N}) \to \mathbb{N}$ の補関数である．

$$\begin{array}{ll} \mathit{fst}(x,y) & = x \\ \mathit{snd}(x,y) & = y \\ \mathit{sub}(x,y) & = x - y \\ \mathit{id}(x,y) & = (x,y) \end{array}$$

これらに対し，補関数の値を不変にする逆方向変換は，それぞれ以下となる．

$$\begin{array}{ll} \mathsf{reflect}_{add,\mathit{fst}}((s_1, _), v) & = (s_1, v - s_1) \text{ if } v \geq s_1 \\ \mathsf{reflect}_{add,\mathit{snd}}((_, s_2), v) & = (v - s_2, s_2) \text{ if } v \geq s_2 \\ \mathsf{reflect}_{add,\mathit{sub}}((s_1, s_2), v) & = \left(\frac{v+(s_1-s_2)}{2}, \frac{v-(s_1-s_2)}{2}\right) \text{ if } v \geq s_1 - s_2 \wedge v = add(s_1, s_2) \bmod 2 \\ \mathsf{reflect}_{add,\mathit{id}_{\mathrm{pair}}}((s_1, s_2), v) & = (s_1, s_2) \text{ if } v = add(s_1, s_2) \end{array}$$

これらの逆方向変換の定義域はそれぞれ異なる．逆方向変換 $\mathsf{reflect}_{add,\mathit{fst}}$ は $\mathsf{reflect}_{add,id}$ より定義域が広く，共通の定義域についての返り値は等しい．これは，直観的には，fst が持ち運ぶ情報は全て id が持ち運んでいるためである．補関数の値を不変にする逆方向変換を通して，補関数が持ち運ぶ情報を変更することはできないことに注意する．また，$\mathsf{reflect}_{add,snd}$ と $\mathsf{reflect}_{add,id}$，$\mathsf{reflect}_{add,sub}$ と $\mathsf{reflect}_{add,id}$ についても同様のことが言える．これに対し，$\mathsf{reflect}_{add,\mathit{fst}}$ と $\mathsf{reflect}_{add,snd}$ と $\mathsf{reflect}_{add,sub}$ は互いに定義域が包含関係になく，また共通の定義域においても返り値が常には一致しない．これは，fst，snd，sub それぞれの持ち運ぶ

情報は，それぞれ組の第一要素，組の第二要素，組の差分というように異なり，比較できないためである．まとめると，より情報を持ち運んでいない補関数がより広い範囲で定義された逆方向変換を定めるため望ましいが，補関数の持ち運ぶ情報は比較できない場合がある．Bancilhon と Spyratos は，この「持ち運ぶ情報」の大小を表現するのために以下の擬順序を導入した [BS81].

定義 3.8 (縮約順序)**.** 定義域の同じ二つの関数 $f :: S \to V$ と $g :: S \to V'$ について，**縮約順序** $f \precsim g$ を

$$f \precsim g \Leftrightarrow (\equiv_g) \subseteq (\equiv_f)$$

と定める． □

この順序について以下が言える．

定理 3.7 ([BS81])**.** 順方向変換 $f :: S \to V$ に対する二つの補関数 $g_1 :: S \to V_1$ と $g_2 :: S \to V_2$ について，

$$g_1 \precsim g_2 \Leftrightarrow \mathsf{reflect}_{f,g_2} \sqsubseteq \mathsf{reflect}_{f,g_1}$$

となる． □

証明. 以下を証明することにより示す．

$$\forall x, y \in S.\ \mathsf{reflect}_{f,g_2}(x, f(y)) = y \Rightarrow \mathsf{reflect}_{f,g_1}(x, f(y)) = y$$

今，ソース $x \in S$ と $y \in S$ が，$\mathsf{reflect}_{f,g_2}(x, f(y)) = y$ を満たすとする．補題 3.1 より $x \equiv_{g_2} y$ が成り立つ．ここで，$g_1 \precsim g_2$ より $(\equiv_{g_2}) \subseteq (\equiv_{g_1})$ が成り立つため $x \equiv_{g_1} y$ が得られる．補題 3.1 より， $x \equiv_{g_1} y$ から $\mathsf{reflect}_{f,g_1}(x, f(y)) = y$ となる． □

定理 3.7 は，定理 3.4 の一般化になっている．

つまり，縮約順序についてより小さい補関数を用いることで，より更新可能性の高い双方向変換が得られる．そのため，より小さい補関数が望ましい．たとえば，$\mathit{fst} \precsim \mathit{id}$ であるため，$\mathsf{reflect}_{\mathit{add},\mathit{id}} \sqsubseteq \mathsf{reflect}_{\mathit{add},\mathit{fst}}$ となっている．また，$\mathit{fst} \not\precsim \mathit{snd}$ かつ $\mathit{snd} \not\precsim \mathit{fst}$ であるため，$\mathsf{reflect}_{\mathit{add},\mathit{fst}} \not\sqsubseteq \mathsf{reflect}_{\mathit{add},\mathit{snd}}$ かつ $\mathsf{reflect}_{\mathit{add},\mathit{snd}} \not\sqsubseteq \mathsf{reflect}_{\mathit{add},\mathit{fst}}$ である．

3.3 プログラムの双方向化のために

これまでの議論から，順方向変換 f に対し，以下の二ステップにより，f の振る舞いのよい逆方向変換を得ることができる．

1. f の補関数 g を導出する
2. $\langle f, g \rangle^{-1}$ の計算

また，より小さい補関数が，より多くの更新を反映できる逆方向変換を定めるため，望ましい．上記二ステップについて，関連データベース上の問い合わせについていくつかの議論がある [LLSV01, LV03] ことを除けば，実際のプログラムに対しそれぞれのステップどのように行えばよいかはまだ明らかではなかった．

第4章　正規木文法と正規生垣文法

第1章で述べた通り，本論文では木構造データ上の双方向変換の構成を議論する．本章では，木構造データについて述べる．木構造データとして，木と生垣について述べる．また，木および生垣の集合の記述である，正規木文法と正規生垣文法 [CDG+97] について述べる．

4.1　木と生垣

本論文では二種類の木構造データについて議論する．図4.1に木と生垣の例を示す．簡単に言えば，木とは，

$$\mathsf{Cons}(\mathsf{S}(\mathsf{Z}()), \mathsf{Cons}(\mathsf{S}(\mathsf{S}(\mathsf{Z}())), \mathsf{Nil}()))$$

のように節点ごとに子の数の定まった木構造であり，生垣は

```
<members>
  <student>Matsuda</student>
  <professor>Hu</professor>
  <professor>Takeichi</professor>
</members>
```

や

```
<members>
  <student>Matsuda</student>
</members>
```

のように節点における子の数が任意の木構造である．

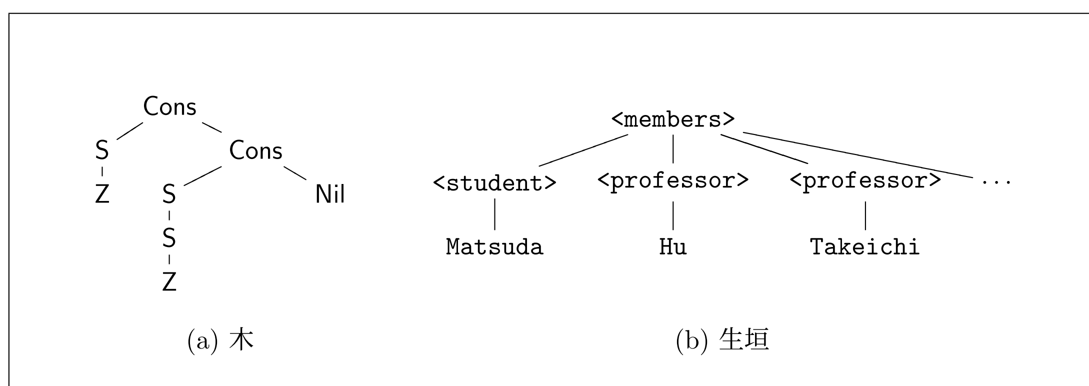

図 4.1. 木と生垣

4.1.1　木

木は，引数の数が定まった（ranked）**構成子**により構成される．たとえば，ペアノ算術における 0 を零引数構成子 Z でそして後者関数 +1 を一引数構成子 S で表現すると，自然数 3 は以下の木で表される．

$$\mathsf{S}(\mathsf{S}(\mathsf{S}(\mathsf{Z})))$$

また，リストの先頭への追加を表現する二引数構成子 Cons とリスト終端を表す一引数構成子 Nil を用いることで，以下のように自然数のリストを木で表すことができる．

$$\mathsf{Cons}(\mathsf{S}(\mathsf{Z}()), \mathsf{Cons}(\mathsf{S}(\mathsf{S}(\mathsf{Z}())), \mathsf{Nil}()))$$

上の自然数のリストは，Haskell の記法でいう $[1, 2]$ である．

形式的には，構成子の集合 Σ から成る**木**の集合 $\mathcal{T}_\Sigma$ は，以下の規則により帰納的に定義される．

- 零引数構成子 $\sigma \in \Sigma$ について，$\sigma() \in \mathcal{T}_\Sigma$ である．
- n 引数構成子 $\sigma \in \Sigma$ について（$n > 0$），木 $t_1, \ldots, t_n \in \mathcal{T}_\Sigma$ に対し，$\sigma(t_1, \ldots, t_n) \in \mathcal{T}_\Sigma$ である．

簡便のため，零引数構成 σ からなる木 $\sigma()$ を単に σ と書く．また，Cons と Nil で構成されるリストは今後の変換記述の例で頻出するため，Haskell の記法 [Bir98] に倣い，Cons(a, x) を $a : x$ と書く，Nil を $[]$ と書く．たとえば，上記自然数のリストは，この記法により以下のように書ける．

$$\mathsf{S}(\mathsf{Z}) : \mathsf{S}(\mathsf{S}(\mathsf{Z})) : []$$

また，さらに Haskell に倣い，リスト $a : b : c : []$ を $[a, b, c]$ と書く．これにより，上の自然数のリストは，以下のように書ける．

$$[\mathsf{S}(\mathsf{Z}), \mathsf{S}(\mathsf{S}(\mathsf{Z}))]$$

文脈 $\mathcal{C}$ はホール $\square_1, \ldots, \square_n$ を含む木である．文脈 $\mathcal{C}$ について，それぞれのホール $\square_1, \ldots, \square_n$ を木 $t_1, \ldots, t_n$ で置き換えて得られる木を $\mathcal{C}[t_1, \ldots, t_n]$ と書く．

4.1.2　生垣

生垣（hedge）もしくは森（forest）は，XML 要素列の表現する．生垣は，木と異なり，引数の数が定まっていない（unranked）構成子により構成される．たとえば，次の XML 要素

において <members> は二つの子を持つ．

```
<members>
  <student>Matsuda</student>
  <professor>Hu</professor>
  <professor>Takeichi</professor>
</members>
```

対し，以下の XML 要素では <members> は一つの子しか持たない．

```
<members>
  <student>Matsuda</student>
</members>
```

形式的には，構成子の集合 Σ の上の**生垣**の集合 $\mathcal{H}_\Sigma$ は，以下の規則により帰納的に定義される．

- 空列 ε は，$\varepsilon \in \mathcal{H}_\Sigma$ である．
- 構成子 $\sigma \in \Sigma$ および生垣 $h \in \mathcal{H}_\Sigma$ について，$\sigma(h) \in \mathcal{H}_\Sigma$ である．
- 生垣 $h_1, \ldots, h_n \in \mathcal{H}_\Sigma$ について，$h_1 \mathbin{\bullet} \ldots \mathbin{\bullet} h_n \in \mathcal{H}_\Sigma$ である．

ここで，演算子 $\bullet$ は生垣同士の連接を表現する．連接演算子 $\bullet$ は結合的であり，ε がその単位元である．すなわち，生垣 $h_1, h_2, h_3, h \in \mathcal{H}_\Sigma$ について

$$h_1 \mathbin{\bullet} (h_2 \mathbin{\bullet} h_3) = (h_1 \mathbin{\bullet} h_2) \mathbin{\bullet} h_3 \qquad h \mathbin{\bullet} \varepsilon = \varepsilon \mathbin{\bullet} h = h$$

となる．簡便のため，$\bullet$ を省略し，$h_1 \mathbin{\bullet} h_2$ を $h_1 h_2$ と書く場合がある．また，生垣 $\sigma(\varepsilon)$ を単に σ と書く場合がある．本論文では，引数の数が定まっていない構成子をラベルと呼ぶ場合がある．

生垣 h の**長さ** $|h|$ を以下により定義する．

$$\begin{aligned} |\varepsilon| &= 0 \\ |\sigma(_) \mathbin{\bullet} h| &= 1 + |h| \end{aligned}$$

生垣上の文脈も，木と同様にして定義される．**文脈** $\mathcal{C}$ は，ホール $\square_1, \ldots, \square_n$ を含む生垣である．文脈 $\mathcal{C}$ について，それぞれのホール $\square_1, \ldots, \square_n$ を生垣 $h_1, \ldots, h_n$ で置き換えて得られる生垣を $\mathcal{C}[h_1, \ldots, h_n]$ と書く．

4.2 正規木文法と正規生垣文法

正規木文法（Regular Tree Grammar）と正規生垣文法（Regular Hedge Grammar）は，それぞれ木の集合と生垣の集合を記述する [CDG+97]．正規木文法と正規生垣文法については，論理演算に対して閉じているなど，様々なよい性質が知られている [CDG+97]．

4.2.1 正規木文法

本論文において，正規木文法は以下により定められる．

定義 4.1 (正規木文法). **正規木文法** G は三つ組 $G = (\Sigma, N, R)$ である．ここで，Σ，N，R は以下の通りである．

- Σ は引数の数が定まった構成子の有限集合.
- N は非終端記号の有限集合.
- R は生成規則の有限集合であり，それぞれ生成規則は以下のいずれかの形式をしている.
  - $A \to B$
  - $A \to \sigma(A_1, \ldots, A_n)$

  ここで，$\sigma \in \Sigma$ は n 引数構成子であり $A, A_1, \ldots, A_n, B \in N$ である． □

通常の正規木文法の定義 [CDG+97] とは異なり，上記定義は開始記号を含まないことに注意する．

我々は，次に非終端記号の意味を定める．

定義 4.2 (生成関係). 正規木文法 $G = (\Sigma, N, R)$ について，**生成関係** $\to_G$ を

$$\mathcal{C}[A] \to_G \mathcal{C}[\sigma(A_1, \ldots, A_n)] \Leftrightarrow A \to \sigma(A_1, \ldots, A_n) \in R.$$

で定める． □

関係 $\to_G$ の反射的推移的閉包 $\xrightarrow{*}_G$ で書く．特に混乱のない場合には，G を省略し，$\to_G$ と $\xrightarrow{*}_G$ をそれぞれ $\to$ および $\xrightarrow{*}$ と書く．

定義 4.3 (言語, 非終端記号の意味). 正規木文法 $G = (\Sigma, N, R)$ について, 非終端記号 $A \in N$ の**言語** $[\![A]\!]_G$ を

$$[\![A]\!]_G = \{t \mid A \xrightarrow{*}_G t, t \in \mathcal{T}_\Sigma\}.$$

で定める． □

正規木文法によって定まる言語を正規木言語と呼ぶ [CDG+97].

正規木文法において，$A \to B$ の形の規則を**単位規則**（unit production）と呼ぶ．正規木文法 $G = (\Sigma, N, R)$ は単位規則を持たない以下の性質を満たす文法 $G' = (\Sigma, N, R')$ に変換することができる．

$$\forall A \in N.\ [\![A]\!]_G = [\![A]\!]_{G'}$$

本論文では特に明示しない限り，正規木文法は単位規則を持たないとする．

例 4.1. Haskell の自然数のリストを表す型

$$\begin{aligned}\textbf{data}\ Nat &= \mathsf{Z} \mid \mathsf{S}(Nat)\\ \textbf{data}\ List &= \mathsf{Nil} \mid \mathsf{Cons}(Nat, List)\end{aligned}$$

は以下の正規木文法 G で表すことができる.

$$\begin{array}{ll} Nat \to \mathsf{Z} & List \to \mathsf{Nil}\\ Nat \to \mathsf{S}(Nat) & List \to \mathsf{Cons}(Nat, List)\end{array}$$

たとえば，Nat について，

$$\begin{array}{l} Nat \to \mathsf{Z}\\ Nat \to \mathsf{S}(Nat) \to \mathsf{S}(\mathsf{Z})\\ Nat \to \mathsf{S}(\mathsf{S}(Nat)) \to \mathsf{S}(\mathsf{S}(\mathsf{Z}))\\ \ldots\end{array}$$

となるため，Nat の言語 $[\![Nat]\!]_G$ は以下となる.

$$[\![Nat]\!]_G = \{\mathsf{Z}, \mathsf{S}(\mathsf{Z}), \mathsf{S}(\mathsf{S}(\mathsf{Z})), \mathsf{S}(\mathsf{S}(\mathsf{S}(\mathsf{Z}))), \mathsf{S}(\mathsf{S}(\mathsf{S}(\mathsf{S}(\mathsf{Z})))), \ldots\}$$

そのため，

$$\begin{array}{l} List \to \mathsf{Nil}\\ List \to \mathsf{Cons}(Nat, List) \to \mathsf{Cons}(Nat, \mathsf{Nil}) \to \cdots\\ List \to \mathsf{Cons}(Nat, List) \to \mathsf{Cons}(Nat, \mathsf{Cons}(Nat, List)) \to \mathsf{Cons}(Nat, \mathsf{Cons}(Nat, \mathsf{Nil})) \to \cdots\\ \cdots\end{array}$$

となり，$List$ の言語は $[\![List]\!]_G$ は以下となる.

$$[\![List]\!]_G = \{[x_1, x_2, \ldots, x_n] \mid x_1, x_2, \ldots, x_n \in [\![Nat]\!]_G, n \in \mathbb{Z}\}$$

すなわち，$List$ の言語は，自然数のリストである.

例 4.2. 以下の正規木文法は上の正規木文法に，非空のリストを表現する非終端記号 $List_+$ に関する生成規則を追加したものである.

$$\begin{array}{lll} Nat \to \mathsf{Z} & List \to \mathsf{Nil} & List_+ \to \mathsf{Cons}(Nat, List)\\ Nat \to \mathsf{S}(Nat) & List \to \mathsf{Cons}(Nat, List) & List_+ \to \mathsf{Cons}(Nat, List_+)\end{array}$$

ここで，

$$List_+ \not\xrightarrow{*} \mathsf{Nil}$$

であるが

$$List_+ \to \mathsf{Cons}(Nat, List), \qquad List \to \mathsf{Cons}(Nat, List_+),$$

であるため，

$$[\![List_+]\!] = [\![List]\!] \setminus \{\mathsf{Nil}\}$$

となる.

文字列上の文法の場合と同じく，曖昧な文法と曖昧でない文法が存在する．

定義 4.4 (曖昧な文法)**.** 正規木文法 $G = (\Sigma, N, R)$ が**曖昧**（ambiguous）であるとは以下を満たすことである．

$$\exists A, B \in N, A \neq B \wedge [\![A]\!]_G \cap [\![B]\!]_G \neq \emptyset$$ □

例 4.1 の正規木文法は曖昧ではないが，例 4.2 の文法は曖昧である．なぜならば，例 4.2 において $\mathit{List}_+ \xrightarrow{*} \mathsf{Cons}(\mathsf{Z}, \mathsf{Nil})$ かつ $\mathit{List} \xrightarrow{*} \mathsf{Cons}(\mathsf{Z}, \mathsf{Nil})$ であるためである．曖昧ではない正規木文法は，決定的な bottom-up 木オートマトン [CDG$^+$97] と等価な概念である．

また，$[\![A]\!]_G = \emptyset$ となる非終端記号を含まない正規生垣文法 G を考えることで議論が簡単になる場合がある．正規生垣文法 G について $[\![A]\!]_G \neq \emptyset$ となる A を**生成的**（generating）であるといい，全ての非終端記号が生成的である場合 G を**既約**（reduced）であるという．

4.2.2 正規生垣文法

正規木文法が木の集合を記述するのに対し，正規生垣文法は生垣の集合を記述する．正規生垣文法は，XML のスキーマのモデルであり，既存のスキーマ記述言語である DTD[1], XML Schema[2] および Relax NG[3] の主要部分を記述することができる [MLMK05].

定義 4.5 (正規生垣文法)**.** 正規生垣文法 G は三つ組 $G = (\Sigma, N, R)$ である．ここで，Σ，N および R は以下の通りである．

- Σ はラベル（引数の数が任意の構成子）の有限集合である．
- N は非終端記号の有限集合である．
- R は生成規則の有限集合であり，各生成規則は以下のいずれか形式をしている．
  - $A \to \varepsilon$
  - $A \to \sigma(A_1) \centerdot A_2$
  - $A \to B$

  ここで，$\sigma \in \Sigma$ であり，$A, A_1, \ldots, A_n, B \in N$ である． □

上の正規生垣文法の定義は，Murata のもの [Mur99] と一見異なるが，表現力は同じである．

我々は，次に非終端記号の意味を定める．

[1] http://www.w3.org/TR/REC-xml/#dt-doctype
[2] http://www.w3.org/XML/Schema
[3] http://relaxng.org/

定義 4.6 (生成関係). 正規生垣文法 $G = (\Sigma, N, R)$ について，**生成関係** $\rightarrow_G$ を

$$\mathcal{C}[A] \rightarrow_G \mathcal{C}[\sigma(A_1)A_2] \Leftrightarrow A \rightarrow \sigma(A_1)A_2 \in R.$$

で定める． □

関係 $\rightarrow_G$ の反射的推移的閉包 $\xrightarrow{*}_G$ で書く．特に混乱のない場合には，$\rightarrow_G$ と $\xrightarrow{*}_G$ を単に $\rightarrow$ および $\xrightarrow{*}$ と書く．

定義 4.7 (言語，非終端記号の意味). 正規生垣文法 $G = (\Sigma, N, R)$ について，非終端記号 $A \in N$ の**言語** $[\![A]\!]_G$ を

$$[\![A]\!]_G = \{t \mid A \xrightarrow{*}_G t, t \in \mathcal{H}_\Sigma\}.$$

で定める． □

正規生垣文法によって定まる言語を正規生垣言語と呼ぶ．

正規生垣文法において，$A \rightarrow B$ の形の規則を単位規則と呼ぶ．正規生垣文法 $G = (\Sigma, N, R)$ は単位規則を持たない以下の性質を満たす文法 $G'(\Sigma, N, R')$ に変換することができる．

$$\forall A \in N.\ [\![A]\!]_G = [\![A]\!]_{G'}$$

本論文では特に明示しない限り，正規生垣文法は単位規則を持たないとする．

また，正規生垣文法は生垣の二分木表現の上の正規木文法と等価である．これを簡潔に示しておく．生垣から，二分木表現への変換を以下の関数 bin で与える．

$$\begin{aligned} bin(\varepsilon) &= \mathsf{Tip} \\ bin(\sigma(x) \centerdot y) &= \mathsf{Node}_\sigma(bin(x), bin(y)) \end{aligned}$$

関数 bin は単射関数である．このとき，以下の関数 $binG$ により，正規生垣文法で等価な二分木上の正規木文法へと変換できる．

$$\begin{aligned} binG(A \rightarrow \varepsilon) &= A \rightarrow \mathsf{Tip} \\ binG(A \rightarrow \sigma(A_1)A_2) &= A \rightarrow \mathsf{Node}_\sigma(A_1, A_2) \\ binG(A \rightarrow B) &= A \rightarrow B \end{aligned}$$

後は，$G' = binG(G), bin(\Box_i) = \Box_i$ とすると，

$$\mathcal{C}[A] \rightarrow_G \mathcal{C}[\sigma(A_1)A_2] \Leftrightarrow bin(\mathcal{C})[A] \rightarrow_{G'} bin(\mathcal{C})[\mathsf{Node}_\sigma(A_1, A_2)]$$

であることから，正規生垣文法は生垣の二分木表現の上の正規木文法と等価であることが確認できる．よって，正規生垣文法も，正規木文法の満たす様々なよい性質を満たす．

4.2.3　性質

正規木文法と正規生垣文法は様々なよい性質を満たす [CDG+97]．このうち，我々が本論文において利用する主な性質を以下に示す．

- 正規木文法は，積に対して閉じている．すなわち，正規木文法 $G_1 = (\Sigma, N_1, R_1)$ と正規木文法 $G_2 = (\Sigma, N_2, R_2)$ に対し，以下の性質を満たす正規木文法 $G' = (\Sigma, N', R')$ を構成できる．

  任意の $A_1 \in N_1$ と $A_2 \in N_2$ に対し，$A' \in N'$ が存在して，$[\![A_1]\!]_{G_1} \cap [\![A_2]\!]_{G_2} = [\![A']\!]_{G'}$ となる．

- 正規木文法 $G = (\Sigma, N, R)$ の $A \in N$ に対し，$[\![A]\!]_G = \emptyset$ かどうかを判定できる．

- 正規木文法 $G = (\Sigma, N, R)$ に対し，曖昧でない正規木文法 $G' = (\Sigma, N', R')$ で，任意の $A \in N$ に対し，$A'_1, \ldots, A'_n \in N'$ が存在して，$[\![A]\!]_G = [\![A'_1]\!]_{G'} \cup \cdots \cup [\![A'_n]\!]_{G'}$ であるものを構成できる．

- 正規生垣言語 S_1, S_2 に対し，以下の条件を満たすかどうかを判定できる [BGM07, BMS08]．

$$\exists h_1, h_3 \in S_1, h_2, h_4 \in S_2.\ h_1 \cdot h_2 = h_3 \cdot h_4 \Rightarrow h_1 = h_3 \wedge h_2 = h_4 \qquad \text{(H-Par)}$$

集合 S_1 と S_2 が，$S_1 \cap S_2 = \emptyset$ であるとき，S_1 と S_2 に**重なりがない**と言う．これまでの議論から，S_1 と S_2 が正規木言語（正規生垣言語）であれば，S_1 と S_2 に重なりがあるかどうかを判定可能である．生垣の集合 S_1 と S_2 が条件 (H-Par) を満たすとき，$S_1 \parallel S_2$ と書き，満たさないとき $S_1 \nparallel S_2$ と書く．また，$S_1 \parallel S_2$ であるとき，**水平方向に重なりがない**と言う．

正規生垣文法 G_1 における A_1 の意味 $[\![A_1]\!]_{G_1}$ と，正規生垣文法 G_2 における A_2 の意味 $[\![A_2]\!]_{G_2}$ に水平方向に重なりがあるかどうかの判定アルゴリズムを示しておく．このアルゴリズムは条件 (H-Par) が次のように書けることを利用する．

$$\exists h_1, h_2, h_3 \in \mathcal{H}_\Sigma.\ (h_1 h_2 \in S_1 \wedge h_3 \in S_2) \wedge (h_1 \in S_1 \wedge h_2 h_3 \in S_2) \wedge |h_2| \geq 1$$

簡単には，以下に示す水平方向の重なり判定アルゴリズムは，$T_1 = [\![A_1]\!]_{G_1}, T_2 = [\![A_2]\!]_{G_2}$ とおくと，以下の処理を実行するものである．

1. $T'_1 = \{x \cdot \mathtt{<\#>}_1 \cdot y \mid x \cdot y \in T_1\}$ を求める．
2. $T'_2 = \{x \cdot \mathtt{<\#>}_2 \cdot y \mid x \cdot y \in T_2\}$ を求める．
3. $T''_1 = \{x \cdot \mathtt{<\#>}_1 \cdot y \mid x \in T_1, y \in T'_2\}$ を求める．
4. $T''_2 = \{x \cdot \mathtt{<\#>}_2 \cdot y \mid x \in T'_1, y \in T_2\}$ を求める．

5. $T''_3 = \{x \mathbin{.} \texttt{<\#>}_1 \mathbin{.} y \mathbin{.} \texttt{<\#>}_2 \mathbin{.} z \mid x, y, z \in \mathcal{H}_\Sigma, |y| \geq 1\}$ を求める．ただし，Σ は T_1 に含まれるラベルと T_2 に含まれるラベルとの和集合である．

6. $T''_1 \cap T''_2 \cap T''_3$ が空かどうか調べる．

アルゴリズム 4.1 (水平方向の重なりの判定).

入力： 正規生垣文法 $G_1 = (\Sigma, N_1, R_1)$ と $G_2 = (\Sigma, N_2, R_2)$，G_1 中の非終端記号 A_1 と G_2 中の非終端記号 A_2

出力： $[\![A_1]\!]_{G_1} \parallel [\![A_2]\!]_{G_2}$ であるかどうか

手続き：

1. Σ に含まれないラベル $\texttt{<\#>}_1, \texttt{<\#>}_2$ を用意する．
2. N_1 と N_2 に含まれない非終端記号 E, Any, X, X', Y, Y' を用意する．
3. 以下の補助手続きを用意する．

$$
\begin{aligned}
&\mathit{insertG}_k(G) = (\Sigma \cup \{\texttt{<\#>}_k\}, (N \times \{0,1,2,3\}) \cup \{E\}, R')\\
&\quad \text{where}\\
&\qquad (\Sigma, N, R) = G\\
&\qquad R' = \{(A,i) \to \sigma((B,i+1))(C,i) \mid A \to \sigma(B)C \in R, i \in \{0,2\}\}\\
&\qquad\qquad \cup \{(A,i) \to \varepsilon \mid A \to \varepsilon \in R, i \in \{1,2,3\}\}\\
&\qquad\qquad \cup \{(A,i) \to \sigma((B,i))(C,i) \mid A \to \sigma(B)C \in R, i \in \{1,3\}\}\\
&\qquad\qquad \cup \{(A,0) \to \texttt{<\#>}_k(E)(B,2) \mid A, B \in N\}\\
&\qquad\qquad \cup \{E \to \varepsilon\}
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{appendWithG}_k(G_1, A_0, G_2) = (\Sigma \cup \{\texttt{<\#>}_k\}, (N_1 \times \{0,1\}) \cup (N_2 \times \{0,1\}) \cup \{E\}, R')\\
&\quad \text{where}\\
&\qquad (\Sigma, N_1, R_1) = G_1\\
&\qquad (\Sigma, N_2, R_2) = G_2\\
&\qquad R' = \{(A,0) \to \sigma((B,1))(C,0) \mid A \to \sigma(B)C \in (R_1 \cup R_2)\}\\
&\qquad\qquad \cup \{(A,1) \to \sigma((B,1))(C,1) \mid A \to \sigma(B)C \in (R_1 \cup R_2)\}\\
&\qquad\qquad \cup \{(A,0) \to \varepsilon \mid A \to \varepsilon \in R_2\}\\
&\qquad\qquad \cup \{(A,0) \to \texttt{<\#>}_k(E)(A_0,0) \mid A \to \varepsilon \in R_1\}\\
&\qquad\qquad \cup \{E \to \varepsilon\}
\end{aligned}
$$

4. $G'_1 := \mathit{insertG}_1(G_1)$.
5. $G'_2 := \mathit{insertG}_2(G_2)$.
6. $G''_1 := \mathit{appendWithG}_1(G_1, (A_2, 0), G'_2)$.
7. $G''_2 := \mathit{appendWithG}_2(G'_1, A_2, G_2)$.
8. 正規生垣文法 $G''_3 = (\Sigma \cup \{\texttt{<\#>}_1, \texttt{<\#>}_2\}, N''_3, R''_3)$ を以下により構成する．
   - $N''_3 = \{Any, X, Y, X', Y', E\}$.

- $R_3'' = \{X \to \sigma(Any)X \mid \sigma \in \Sigma\}$
  $\cup \{X \to \texttt{<\#>}_1(E)X'\} \cup \{X' \to \sigma(Any)Y'\}$
  $\cup \{Y' \to \texttt{<\#>}_2(E)Y\}$
  $\cup \{Y' \to \sigma(Any)Y' \mid \sigma \in \Sigma\}$
  $\cup \{Y \to \sigma(Any)Y \mid \sigma \in \Sigma\} \cup \{Y \to \varepsilon\}$
  $\cup \{Any \to \sigma(Any)Any \mid \sigma \in \Sigma\} \cup \{Any \to \varepsilon\}$.

9. $[\![(A,0)]\!]_{G_1''} \cap [\![((A,0),0)]\!]_{G_2''} \cap [\![X]\!]_{G_3''} = \emptyset$ かどうかを判定する． □

上のアルゴリズムは文献 [Kru08] の正規文字列文法に対する水平方向の重なり判定を元にしている．

第5章　プログラムの双方向化

本章では，木上の変換プログラムに対する双方向化手法を述べる．本章の内容の一部は，文献 [MHN+07] として発表された．

5.1　概観

我々の提案する双方向化の詳細を述べる前に，提案する双方向化の概観を例を用いて説明する．

図 5.1 に提案する双方向化の流れを示す．我々の双方向化は，順方向変換を入力とし，そして逆方向変換と，ビュー上の更新の反映可能性検査器を出力する．

5.1.1　順方向変換

本章の双方向化が対象とする順方向変換は，affine で treeless [Wad90] な一階の関数型言語により記述される．たとえば，二つのリストの連接を行う関数 *append* は，以下のように affine で treeless な言語で記述できる．

$$
\begin{aligned}
&append([], y) &&\hat{=}\ y\\
&append(a : x, y) &&\hat{=}\ a : append(x, y)
\end{aligned}
$$

関数 *append* にみるように，本章で議論する順方向変換記述言語では，Haskell [Bir98] などの通常の関数型言語と同様に，左辺においてパターンマッチにより値を変数に束縛し，右辺

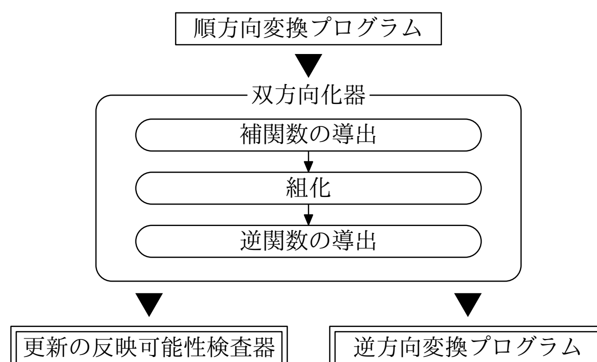

図 5.1. 双方向化の流れ

において式を評価することにより束縛された値を元に値を構成する．言語の詳細な定義は，5.2 節で述べる．

5.1.2 双方向化

入力となる順方向変換プログラムに対し，我々は次の三ステップにより双方向化を行う．

- 補関数 [BS81] の導出
- 元の順方向変換と，導出した補関数の組化 [HITT97, Chi93]
- 逆関数の導出

第 3 章で述べたように，順方向変換 f に対し，補関数 f^{c} を与えることにより，逆方向変換 $f_{\mathrm{B}}(s,v) = \langle f, f^{\mathrm{c}} \rangle^{-1}(v, f^{\mathrm{c}}(s))$ を定めることができる．また，補関数は縮約順序（第 3 章，定義 3.8）において小さいものが，効果的な逆方向変換を導出する上で望ましい．

補関数の導出

提案手法は，順方向変換に対し，変換が単射になるのに十分な情報を補うことにより自動的に補関数を導出する．また，その際において順方向変換の単射性を解析することにより，より小さい補関数を求めることができる．補関数導出手法の詳細は 5.3 において述べる．

たとえば，前述の関数 append に対し，補関数を導出する場合を考える．関数 append は，$\mathit{append}([1,2],[3]) = [1,2,3]$ かつ $\mathit{append}([1,2,3],[]) = [1,2,3]$ となるため単射ではない．関数 append の補関数は，append により同じ値になる入力を区別する．提案手法は，評価中に使用された規則に着目することにより，append の補関数 $\mathit{append}^{\mathrm{c}}$ を導出する．

$$\begin{aligned} \mathit{append}^{\mathrm{c}}([], y) &\mathrel{\hat{=}} \mathsf{B}_1 \\ \mathit{append}^{\mathrm{c}}(a:x, y) &\mathrel{\hat{=}} \mathsf{B}_2(\mathit{append}^{\mathrm{c}}(x,y)) \end{aligned}$$

補関数 $\mathit{append}^{\mathrm{c}}$ は構成子 B_1 と B_2 により，関数 append の評価中に使用された規則を憶える．直観的には，補関数 $\mathit{append}^{\mathrm{c}}$ は第一引数のリストの長さを計算している．たとえば，以下のように，$\mathit{append}^{\mathrm{c}}$ の出力する B_2 の数は第一引数のリストの長さと同じである．

$$\mathit{append}^{\mathrm{c}}([1,2,3],[]) = \underbrace{\mathsf{B}_2(\mathsf{B}_2(\mathsf{B}_2}_{=\ \mathit{length}([1,2,3])}(\mathsf{B}_1)))$$

組化と逆関数導出

順方向変換 f に対する補関数 f^{c} が求まった後，我々は第 3 章の式 (RFL) に従い，f の逆方向変換 f_{B} を

$$f_{\mathrm{B}}(s,v) \mathrel{\hat{=}} \langle f, f^{\mathrm{c}} \rangle^{-1}(v, f^{\mathrm{c}}(s))$$

と構成する．そのため，我々は $\langle f, f^{\mathrm{c}}\rangle^{-1}$ のプログラムを求める．関数 $\langle f, f^{\mathrm{c}}\rangle^{-1}$ のプログラムを求めるのに，我々はまず組化 [HITT97, Chi93] により $\langle f, f^{\mathrm{c}}\rangle$ のプログラムを求め，その後 $\langle f, f^{\mathrm{c}}\rangle^{-1}$ のプログラムを求める．組化することにより，関数 $\langle f, f^{\mathrm{c}}\rangle^{-1}$ の評価が行いやすくする [GK05, Epp85].

たとえば，関数 *append* は補関数 $append^{\mathrm{c}}$ と組化すると以下となる．

$$\begin{aligned}
\langle append, append^{\mathrm{c}}\rangle([], y) &\mathrel{\hat{=}} (y, \mathsf{B}_1)\\
\langle append, append^{\mathrm{c}}\rangle(a : x, y) &\mathrel{\hat{=}} \mathbf{let}\,(s,t) \mathrel{\hat{=}} \langle append, append^{\mathrm{c}}\rangle(x, y)\,\mathbf{in}\ (a : s, \mathsf{B}_2(t))
\end{aligned}$$

そして，組化後，左辺と右辺を入れ替えることにより，以下の組化した関数の逆関数が得られる．

$$\begin{aligned}
\langle append, append^{\mathrm{c}}\rangle^{-1}(y, \mathsf{B}_1) &\mathrel{\hat{=}} ([], y)\\
\langle append, append^{\mathrm{c}}\rangle^{-1}(a : s, \mathsf{B}_2(t)) &\mathrel{\hat{=}} \mathbf{let}\,(x,y) \mathrel{\hat{=}} \langle append, append^{\mathrm{c}}\rangle^{-1}(s, t)\,\mathbf{in}\ (a : x, y)
\end{aligned}$$

上では，決定的な逆関数が得られたが，一般には得られる逆関数は非決定的である．

最後に，第 3 章の式 (RFL) に従い逆方向変換を構成する．たとえば，*append* に対しては以下が得られる．

$$append_{\mathrm{B}}(s, v) \mathrel{\hat{=}} \langle append, append^{\mathrm{c}}\rangle^{-1}(v, append^{\mathrm{c}}(s))$$

上の関数 $\mathsf{reflect}_{append, append^{\mathrm{c}}}$ は以下の関数 ρ と同じ動作を行う．

$$\begin{aligned}
&\rho((x,y),z) = \mathbf{let}\,(s,t) = splitAt(length(x), z)\,\mathbf{in}\ (s,t)\\
&\quad \text{where}\\
&\qquad length([]) &&= 0\\
&\qquad length(a : x) &&= 1 + length(x)\\
&\qquad splitAt(0, x) &&= ([], x)\\
&\qquad splitAt(n+1, a : x) &&= \mathbf{let}\,(s,t) = splitAt(n, x)\,\mathbf{in}\ (a : s, t)
\end{aligned}$$

たとえば，

$$append_{\mathrm{B}}(([1,2,3],[4,5]),[11,12,13,4,5]) = ([11,12,13],[4,5])$$

である．また，

$$append_{\mathrm{B}}(([1,2,3],[4,5]),[1,2,3,4]) = ([1,2,3],[4])$$

となるが，

$$append_{\mathrm{B}}(([1,2,3],[4,5]),[1,2]) = \bot$$

となる．逆方向変換 $append_{\mathrm{B}}$ は，更新後のビューのリストの長さが，ソースの第一要素のリストの長さ以上であるときにしか定義されない．

$$
\begin{array}{llll}
rule & ::= & f(p_1, \ldots, p_n) \doteq e & \text{関数定義規則} \\
p & ::= & \mathsf{C}(p_1, \ldots, p_n) & \text{構成子パターン} \\
 & | & x & \text{変数パターン} \\
e & ::= & \mathsf{C}(e_1, \ldots, e_n) & \text{構成子式} \\
 & | & f(x_1, \ldots, x_n) & \text{関数呼出式} \\
 & | & x & \text{変数式}
\end{array}
$$

ただし，$\mathsf{C} \in \Sigma$ は n 引数構成子，$f \in Q$ は n 引数関数記号，そして x は変数である．

図 5.2. 言語 VDL の構文

5.1.3 更新の反映可能性検査器

上のように，一般には導出された逆方向変換は部分関数になる．そのため，実際に更新の反映を行う前に更新が反映できるかどうかがわかれば便利である．

本章では，更新の反映可能性検査器も提供する．これは，最初のソースが与えられた場合に，ビュー上の反映可能な更新を正規木文法の形で返す．正規木文法については，様々なよい性質が知られている [CDG+97]．また，補関数値不変の逆方向変換の性質により，逆方向変換により更新を反映した後でも，同じ正規木文法により更新の反映可能性判定を行える．

更新の反映可能性検査器の導出については，5.5 節にて詳細を述べる．

5.2 順方向変換記述言語 VDL

本節では，順方向変換記述言語 VDL[1]について述べる．この言語は affine かつ treeless [Wad90] な一階の関数型言語である．

5.2.1 言語 VDL の構文

言語 VDL の構文を図 5.2 に示す．言語 VDL で記述されるプログラム $\mathcal{P}$ は三つ組 $\mathcal{P} = (\Sigma, Q, R)$ であり，それぞれ，構成子の集合 Σ，関数記号の集合 Q，関数定義規則の集合 R である．また，Q と Σ は互いに共通部分を持たない．

プログラムの各規則は以下の形式をしている．

$$f(p_1, \ldots, p_n) \doteq e$$

[1]ビュー定義言語（View Definition Language）．

ここで，$p_1, \ldots, p_n$ はパターン，e は *treeless* 式 [Wad90] である．図 5.2 に示す通り，treeless 式において，関数の引数は変数に限定される．直観的には，treeless 制約のもとでは，関数呼出のネスト，つまり関数の合成が禁止される．関数呼出がネストしないため，プログラムにおいて中間結果となる値（構成子による木）が存在しない（tree-less）．また，我々は各規則において変数出現が *affine* であると制限する．すなわち，右辺において同じ変数は二度以上出現しない．直観的には，affine 制約のもとでは，値の複製や同じ値の複数回走査が禁止される．

同じ関数を定義する各規則についてパターンに重なりがないとする．つまり，入力がマッチするパターンは高々一つである．また，通常の関数型言語と同様に，左辺において同じ変数が複数回出現することなく，右辺に現れる変数は必ず左辺にも現れるとする．

制限はされているものの，言語 VDL では以下に挙げるような例を記述することができる．

例 5.1 (恒等関数)**.** 恒等関数は，もっとも基本的な順方向変換のうちの一つである．恒等関数 id を VDL で記述すると以下となる．

$$id(x) \doteq x$$

例 5.2 (射影関数)**.** 射影関数は，ソースの一部分をビューに抽出する重要な順方向変換である．たとえば，組の第一要素を返す関数 fst および組の第二要素を返す関数 snd は VDL で記述すると以下となる．

$$\begin{aligned} fst(x, y) &\doteq x \\ snd(x, y) &\doteq y \end{aligned}$$

例 5.3 (定数関数)**.** 定数関数は，ソースと関係のない値をビューに含めるもっとも基本的な関数である．たとえば，以下の VDL で記述された関数 nil は，入力によらず空リストを返す．

$$nil(x) \doteq []$$

例 5.4 (自然数の上の再帰関数)**.** 順方向変換は，構成子により再帰的に定義された木を木へと変換する．そのため，再帰関数は順方向変換を定義する上で重要である．たとえば，以下の関数 add は自然数の構造（零 Z，後者関数 S）の上で再帰することにより，二つの自然数を足し合わせる．

$$\begin{aligned} add(\mathsf{Z}, y) &\doteq y \\ add(\mathsf{S}(x), y) &\doteq \mathsf{S}(add(x, y)) \end{aligned}$$

我々は Haskell [Bir98] の記法に従い，大文字から始まる文字列を構成子の名前に用い，小文字から始まる文字列を関数や変数に用いる．

以下の関数 max は二つの自然数のうち大きいほうを返す関数である．関数 add は一つの入力のみを走査するのに対し，関数 max は二つの入力を同時に走査する．

$$\begin{aligned} max(\mathsf{Z}, y) &\doteq y \\ max(\mathsf{S}(x), \mathsf{Z}) &\doteq \mathsf{S}(x) \\ max(\mathsf{S}(x), \mathsf{S}(y)) &\doteq \mathsf{S}(max(x, y)) \end{aligned}$$

$$\frac{e_1 \Downarrow v_1 \quad \cdots \quad e_n \Downarrow v_n}{\mathsf{C}(e_1,\ldots,e_n) \Downarrow \mathsf{C}(v_1,\ldots,v_n)}\textsc{Con}$$

$$\frac{\begin{array}{c} f(p_1,\ldots,p_n) \mathrel{\hat{=}} e \in R \\ \exists\theta.\ f(p_1\theta,\ldots,p_n\theta) = f(v_1,\ldots,v_n) \quad e\theta \Downarrow u \end{array}}{f(v_1,\ldots,v_n) \Downarrow u}\textsc{Fun}$$

ここで，$v_1,\ldots,v_n$ は値，つまり構成子の集合 Σ について $v_1,\ldots,v_n \in \mathcal{T}_\Sigma$ である．

図 5.3. 言語 VDL の意味

例 5.5 (リストや木の上の再帰関数)．先程も述べたように順方向変換は構成子による木を変換する．言語 VDL においては，リストや二分木など，構成子による木で表現できる多様なデータを変換する関数を記述することができる．たとえば，以下の関数 zip は二つのリストを綴じ合わせる．

$$\begin{array}{ll} \mathit{zip}([], y) & \mathrel{\hat{=}} [] \\ \mathit{zip}(a : x, []) & \mathrel{\hat{=}} [] \\ \mathit{zip}(a : x, b : y) & \mathrel{\hat{=}} \mathsf{Pair}(a, b) : \mathit{zip}(x, y) \end{array}$$

また，たとえば，以下の関数 flip は二分木の左右を反転する．

$$\begin{array}{ll} \mathit{flip}(\mathsf{Leaf}) & \mathrel{\hat{=}} \mathsf{Leaf} \\ \mathit{flip}(\mathsf{Node}(n, l, r)) & \mathrel{\hat{=}} \mathsf{Node}(n, \mathit{flip}(r), \mathit{flip}(l)) \end{array}$$

5.2.2　言語 VDL の意味

ここでは言語 VDL の意味を与える．また，今後の議論に使用する記法や関数を導入する．

代入 θ は，$\{x \mid \theta(x) \neq x\}$ が有限集合となるような，変数に対し変数もしくは木を割り当てる写像である．また，式もしくはパターン t 中の変数 x を $\theta(x)$ に置き換えて得られる式もしくはパターンを，$t\theta$ と書く．

代入を導入したことで，パターンに重なりがないことを形式的に述べることことができる．すなわち，VDL において，同じ関数の規則

$$\begin{array}{l} f(p_1,\ldots,p_n) \mathrel{\hat{=}} e \\ f(p'_1,\ldots,p'_n) \mathrel{\hat{=}} e' \end{array}$$

に対し，$(p_1,\ldots,p_n)\theta = (p'_1,\ldots,p'_n)\theta'$ となるような代入 θ と θ は存在しない．

言語 VDL の意味を図 5.3 に示す．ここで，関係 $e \Downarrow v$ は，式 e を評価すると値 v になることを表す．また，図 5.3 の意味に従い，プログラムを P における関数記号 f の意味を

$$[\![f]\!]_P(v_1,\ldots,v_n) = \begin{cases} v & \text{if } f(v_1,\ldots,v_n) \Downarrow v, \\ \bot & \text{otherwise.} \end{cases}$$

で定める．上の式は well-defined ある．これは，パターンに重なりがないため，$f(v_1,\ldots,v_n)\Downarrow v$ となる v は高々一個しか存在しないためである．また，図 5.3 の FUN 規則において，入力がマッチする関数定義規則は高々一個しないため，プログラムは**決定的**（deterministic）である．

今後の議論において，病的な場合を避けるために，プログラムに対しいくつかの仮定を置く．一つは，構成子の集合 Σ は少くとも二つの構成子を含み，内一つは零引数であることである．つまり，順方向変換の入力の集合 $\mathcal{T}_\Sigma$ は二つ以上の元を持つことである．また，プログラムは全ての入力に対して未定義な関数を含まないとする．たとえば，以下の関数 f は，全ての入力に対して未定義である．

$$f(x) \doteq f(x)$$

今後の議論のために，いくつかの記法を導入する．木の列 $t_1,\ldots,t_n$ に現れる全ての変数を集めた集合を $\mathsf{vars}(t_1,\ldots,t_n)$ と書く．たとえば，$\mathsf{vars}(a:x)=\{a,x\}$ であり，$\mathsf{vars}(add(x,y))=\{x,y\}$ である．規則 $r=f(p_1,\ldots,p_n)\doteq e$ について，$\mathsf{lostvars}(r)$ を

$$\mathsf{lostvars}(r)=\mathsf{vars}(p_1,\ldots,p_n)\setminus\mathsf{vars}(e)$$

と定義する．また，式 e の可能な評価結果の集合 $\{v\mid e\theta\Downarrow v,\mathsf{vars}(e\theta)=\emptyset\}$ を式 e の**値域**と呼び，$\mathsf{ran}(e)$ と書く．第 4 章で木や生垣に対し文脈を定めたのと同様に，式やパターンについても文脈を定める．**文脈** $\mathcal{C}$ は，特殊な変数 $\square_1,\ldots,\square_n$ を含むパターン／式であり，$\mathcal{C}$ 中のそれぞれの変数 $\square_i$ をパターン／式 t_i で置き換えて得られるパターン／式を $\mathcal{C}[t_1,\ldots,t_n]$ と書く．

また，1.5 節で述べたように我々はベクトル記法を用いる．たとえば，規則 $f(p_1,\ldots,p_n)\doteq e$ を $f(\overrightarrow{p})\doteq e$ と書く場合がある．

5.3 補関数の導出

ここでは，VDL で記述された順方向変換に対する補関数導出手法について述べる．これまでの関係データベースの上の議論 [CP84, LLSV01, LV03] とは異なり，我々は木上の再帰的な順方向変換プログラムから補関数プログラムを導出する．まず，素朴な補関数導出法を示すことにより，補関数導出における我々の基本的なアイデアを示し，その後，単射性解析を利用することにより，より小さい補関数を導出する手法を示す．

5.3.1 素朴な補関数導出

第 3 章で述べたように，関数 $f::S\to V$ に対する補関数は関数 $g::S\to V'$ は，$\langle f,g\rangle::S\to V\times V'$ を単射にする関数である．もし，f が単射であるならばどんな関数 g も f の

補関数になる．ところが，f が単射でないならば，$f(s_1) = f(s_2)$ となる $s_1, s_2 \in S$ に対し，$g(s_1) \neq g(s_2)$ になるように，g を選ぶ必要がある．

補関数の導出において，我々は関数が単射でなくなっている原因に着目する．たとえば，

$$fst(x, y) \hat{=} x$$

は単射ではない．これは，fst が第二引数右辺において使用しておらず，第二引数の情報が失われているからである．また，たとえば，

$$\begin{aligned} alwaysTrue(\mathsf{True}) &\hat{=} \mathsf{True} \\ alwaysTrue(\mathsf{False}) &\hat{=} \mathsf{True} \end{aligned}$$

も単射ではない．これは，$alwaysTrue$ が，どの規則を使用した場合にも True を返すため，使用した規則の情報が失われているためである．そのため，この二つの情報を補うようにすることで，我々は補関数を導出できる．

素朴な補関数導出アルゴリズム Alg-C を以下に示す．ここで，全ての treeless 式は，構成子からなる部分とそれ以外（関数呼出と変数）に分けることで，構成子のみより構成される文脈 $\mathcal{C}$ により，$\mathcal{C}[f_1(\overrightarrow{x_1}), \ldots, f_n(\overrightarrow{x_n}), \overrightarrow{x}]$ と表すことができることに注意する．

アルゴリズム 5.1 (素朴な補関数導出：Alg-C)**.**

入力： 順方向変換を定義するためのプログラム $\mathcal{P} = (\Sigma, Q, R)$.

出力： 補関数を定義するためのプログラム $\mathcal{P}^{\mathrm{c}}$.

手続き：

1. プログラム中のそれぞれの規則 $r \in R$

$$r = f(\overrightarrow{p}) \hat{=} \mathcal{C}[f_1(\overrightarrow{x_1}), \ldots, f_n(\overrightarrow{x_n}), \overrightarrow{x}]$$

   から以下の規則 r^{c} を作成する．

$$r^{\mathrm{c}} = f^{\mathrm{c}}(\overrightarrow{p}) \hat{=} \mathsf{B}_r(f_1^{\mathrm{c}}(\overrightarrow{x_1}), \ldots, f_n^{\mathrm{c}}(\overrightarrow{x_n}), \overrightarrow{y})$$

   ただし，ここで，$\{\overrightarrow{y}\} = \mathsf{lostvars}(r)$ であり，B_r は各規則を識別するために導入された規則ごとに異なる構成子，そして $f^{\mathrm{c}}, {f_1}^{\mathrm{c}}, \ldots, {f_n}^{\mathrm{c}} \notin Q$ はそれぞれ $f, f_1, \ldots, f_n$ に対応する補関数の関数記号である．

2. プログラム $\mathcal{P}^{\mathrm{c}}$ を以下の通りに構成する．

$$P^{\mathrm{c}} = (\Sigma \cup \{\mathsf{B}_r \mid r \in R\}, \{f^{\mathrm{c}} \mid f \in Q\}, \{r^{\mathrm{c}} \mid r \in R\}) \qquad \square$$

アルゴリズム Alg-C の正しさを示す前に，いくつかの例により Alg-C の動作を確認する．

例 5.6. 以下の関数 fst を考える．

$$fst(x, y) \mathrel{\hat{=}} x$$

前述のように関数 fst は第二引数 y を使用しない．関数 fst に対し，ALG-C は次の関数を導出する．

$$fst^{\mathrm{c}}(x, y) \mathrel{\hat{=}} \mathsf{B}_1(y)$$

ここで，B_1 は，fst の第一規則を識別するためにアルゴリズム ALG-C が導入した構成子である．今後，我々は B_i を第 i 規則に対応する構成子に使用する．上の関数 fst^{c} は，fst が使用しない変数 y を返り値に含めることにより，fst が失った情報を補っている．

例 5.7. 例 5.4 における関数 add を考える．関数 add の定義において，左辺に出現する変数は全て右辺にも出現する．アルゴリズム ALG-C は関数 add に対し以下のプログラムを導出する．

$$\begin{aligned} add^{\mathrm{c}}(\mathsf{Z}, y) &\mathrel{\hat{=}} \mathsf{B}_1 \\ add^{\mathrm{c}}(\mathsf{S}(x), y) &\mathrel{\hat{=}} \mathsf{B}_2(add^{\mathrm{c}}(x, y)) \end{aligned}$$

実質的には，add^{c} は第一引数の情報を返している．

例 5.8. 例 5.4 における関数 max を考える．アルゴリズム ALG-C は関数 max に対し以下のプログラムを導出する．

$$\begin{aligned} max^{\mathrm{c}}(\mathsf{Z}, y) &\mathrel{\hat{=}} \mathsf{B}_1 \\ max^{\mathrm{c}}(\mathsf{S}(x), \mathsf{Z}) &\mathrel{\hat{=}} \mathsf{B}_2 \\ max^{\mathrm{c}}(\mathsf{S}(x), \mathsf{S}(y)) &\mathrel{\hat{=}} \mathsf{B}_3(max^{\mathrm{c}}(x, y)) \end{aligned}$$

この補関数は，縮約順序のもと次の関数 $minle$ と同等，すなわち，$max^{\mathrm{c}} \preceq minle$ かつ $minle \preceq max^{\mathrm{c}}$ である．

$$minle(x, y) = \begin{cases} (x, 1) & \text{if } x \leq y \\ (y, 0) & \text{if } x > y \end{cases}$$

関数 $minle$ は，二つの引数のうちの最小値とともに，第一引数が第二引数以下であるかどうかの真理値を返す．

アルゴリズム ALG-C は正しい，つまり以下が言える．

定理 5.1 (ALG-C の正しさ)**.** プログラム $P = (\Sigma, Q, R)$ に対し，ALG-C によりプログラム $P^{\mathrm{c}} = (\Sigma^{\mathrm{c}}, Q^{\mathrm{c}}, R^{\mathrm{c}})$ が得られたとする．このとき，全ての関数記号 $f \in Q$ に対し $[\![f^{\mathrm{c}}]\!]$ は $[\![f]\!]$ の補関数である．

証明. 式 e を評価するのに使用した FUN 規則の数を $e \Downarrow v$ の証明木に沿って縦に数えたもの最大値，つまり証明木の深さを，#FUNDEPTH(e) と書く．

以下を示すことが必要十分である．

任意の $f \in Q$ および任意の $\overrightarrow{v}, \overrightarrow{v'} \in \mathsf{dom}(f)$ について $\overrightarrow{v} \neq \overrightarrow{v}'$ となるならば,

$$[\![f]\!](\overrightarrow{v}) = [\![f]\!](\overrightarrow{v'}) \Rightarrow [\![f^{\mathrm{c}}]\!](\overrightarrow{v}) \neq [\![f^{\mathrm{c}}]\!](\overrightarrow{v'})$$

となる.

これを，$\#\textsc{FunDepth}(f(\overrightarrow{v})) + \#\textsc{FunDepth}(f(\overrightarrow{v}'))$ の上の帰納法で示す.

元の関数 f の定義域と対応する補関数 f^{c} の定義域とは等しいことに注意する.

基底：$\#\textsc{FunDepth}(f(\overrightarrow{v})) + \#\textsc{FunDepth}(f(\overrightarrow{v'})) = 0$**.**

このとき，規則 $r, r' \in R$

$$r = f(\overrightarrow{p}) \doteq \mathcal{C}[\overrightarrow{x}]$$
$$r' = f(\overrightarrow{p}') \doteq \mathcal{C}'[\overrightarrow{x}']$$

と代入 θ, θ' で

$$\overrightarrow{p}\theta = \overrightarrow{v}, \quad \overrightarrow{p}'\theta' = \overrightarrow{v}', \quad [\![f]\!](\overrightarrow{v}) = [\![f]\!](\overrightarrow{v'})$$

を満たすものが存在する．もし $r \neq r'$ である場合，主張は正しい．なぜなら，構成子 B_r と $\mathsf{B}_{r'}$ は必ず異なるためである．よって，$r = r'$ である場合を考える．ここで $\mathcal{C}[\overrightarrow{x}]\theta = \mathcal{C}[\overrightarrow{x}]\theta'$ より，$x \in \mathsf{vars}(\overrightarrow{p})$ について $\theta(x) = \theta'(x)$ となる．このとき，$\overrightarrow{v} = \overrightarrow{p}\theta$ と $\overrightarrow{v}' = \overrightarrow{p}\theta'$ は異なるにもかかわらず $[\![f]\!](\overrightarrow{v}) = [\![f]\!](\overrightarrow{v}')$ であるため，変数 $z \in \mathsf{lostvars}(r)$ で $\theta(z) \neq \theta'(z)$ となるものが存在する．ここで，f^{c} について $f^{\mathrm{c}}(\overrightarrow{p}) \doteq \mathsf{B}_r(\overrightarrow{y})$ となるものが存在し，定義より $\{\overrightarrow{y}\} = \mathsf{lostvars}(r)$ となる．そのため，$[\![f^{\mathrm{c}}]\!](\overrightarrow{v}) \neq [\![f^{\mathrm{c}}]\!](\overrightarrow{v}')$ が成り立つ.

帰納：$\#\textsc{FunDepth}(f(\overrightarrow{v})) > 0, \#\textsc{FunDepth}(f(\overrightarrow{v}')) = 0$**.**

このとき規則 $r, r' \in R$

$$r = f(\overrightarrow{p}) \doteq \mathcal{C}[f_1(\overrightarrow{x_1}) \ldots f_n(\overrightarrow{x_n}), \overrightarrow{x}] \qquad \text{ただし } n > 0$$
$$r' = f(\overrightarrow{p}') \doteq \mathcal{C}'[\overrightarrow{x}']$$

と代入 θ, θ' で,

$$\overrightarrow{p}\theta = \overrightarrow{v}, \quad \overrightarrow{p'}\theta' = \overrightarrow{v'}, \quad [\![f]\!](\overrightarrow{v}) = [\![f]\!](\overrightarrow{v'})$$

を満たすものが存在する．ここで，$r \neq r'$ より構成子 B_r と $\mathsf{B}_{r'}$ は異なる．そのため，$[\![f^{\mathrm{c}}]\!](\overrightarrow{v}) \neq [\![f^{\mathrm{c}}]\!](\overrightarrow{v'})$ が成り立つ.

帰納： $\#\textsc{FunDepth}(f(\overrightarrow{v})) = 0, \#\textsc{FunDepth}(f(\overrightarrow{v'})) > 0$**.**

上と同様.

帰納： $\#\textsc{FunDepth}(f(\overrightarrow{v})) > 0, \#\textsc{FunDepth}(f(\overrightarrow{v'})) > 0$.

このとき規則 $r, r' \in R$

$$r = f(\overrightarrow{p}) \doteq \mathcal{C}[f_1(\overrightarrow{x_1}), \ldots, f_n(\overrightarrow{x_n}), \overrightarrow{x}]$$
$$r' = f(\overrightarrow{p}') \doteq \mathcal{C}'[f'_1(\overrightarrow{x'_1}), \ldots, f'_{n'}(\overrightarrow{x'_{n'}}), \overrightarrow{x}']$$

と代入 θ, θ' で，

$$\overrightarrow{p}\theta = \overrightarrow{v}, \overrightarrow{p'}\theta' = \overrightarrow{v}', \overrightarrow{v} \neq \overrightarrow{v}', \quad [\![f]\!](\overrightarrow{v}) = \overrightarrow{f}(\overrightarrow{v}')$$

を満たすものが存在する．もし $r \neq r'$ である場合，主張は正しい．なぜなら，構成子 B_r と $\mathsf{B}_{r'}$ とは異なるためである．よって，$r = r'$ である場合を考える．このとき $\overrightarrow{p}\theta \neq \overrightarrow{p}\theta'$ であることから，次の二つの場合を考えればよい．

- $\theta(z) \neq \theta'(z)$ となる $z \in \mathsf{lostvars}(r)$ が存在
- $\theta(z) \neq \theta'(z)$ となる $z \in \{\overrightarrow{x_i}\}$ が存在

ここで，$[\![f]\!](\overrightarrow{v}) = [\![f]\!](\overrightarrow{v'})$ であるため，$\theta(z) \neq \theta'(z)$ のときも $z \in \{\overrightarrow{x}\}$ となりえないことに注意する．一つ目の場合，補関数の規則 r^{c} の定義より $[\![f^{\mathrm{c}}]\!](\overrightarrow{v}) \neq [\![f^{\mathrm{c}}]\!](\overrightarrow{v})$ となる．よって二つ目の場合を考える．このとき，$\overrightarrow{x_i}\theta \neq \overrightarrow{x_i}\theta'$ と $[\![f]\!](\overrightarrow{x_i}\theta) = [\![f]\!](\overrightarrow{x_i}\theta')$ が成り立つ．帰納法の仮定より $[\![{f_i}^{\mathrm{c}}]\!](\overrightarrow{x_i}\theta) \neq [\![{f_i}^{\mathrm{c}}]\!](\overrightarrow{x_i}\theta')$ が言えるため，$[\![f^{\mathrm{c}}]\!](\overrightarrow{v}) \neq [\![f^{\mathrm{c}}]\!](\overrightarrow{v'})$ が成り立つ． □

第 3 章で述べたように，ある関数についての補関数は一般には一つに決まらない．効果的な逆方向変換を定めるためには，ある関数に対する補関数の中で縮約順序においてより小さいものが望ましい．アルゴリズム Alg-C は，いくつかの例については fst のように極小の補関数を導出し，いくつかの例については add や max のように極小ではないものの十分に小さい補関数を導出する．しかし，いくつかの例については，Alg-C はあまり小さくない補関数を導出する．たとえば，以下の関数 neg を考える．

$$\mathit{neg}(\mathsf{True}) \doteq \mathsf{False}$$
$$\mathit{neg}(\mathsf{False}) \doteq \mathsf{True}$$

関数 neg は単射であるので，関数 neg に対する最も小さい補関数は定数関数となる．ところが，アルゴリズム Alg-C は

$$\mathit{neg}^{\mathrm{c}}(\mathsf{True}) \doteq \mathsf{B}_1$$
$$\mathit{neg}^{\mathrm{c}}(\mathsf{False}) \doteq \mathsf{B}_2$$

を返す．関数 $\mathit{neg}^{\mathrm{c}}$ は単射関数であり，縮約順序において最も大きいものである．このことは，アルゴリズム Alg-C が，補う必要のない情報まで補ってしまっていることに起因する．関数 neg の返り値から，どちらの規則を使用したかがわかるため，補関数は neg の規則を識別する必要はない．アルゴリズム Alg-C では単射でない原因に着目して補関数を導出していたが，同様の点に着目することで，より小さい補関数を導出することを考える．

以下，5.3.2 節では，それぞれの式の値域（厳密な型）を推論する手法を述べ，さらに 5.3.3 節ではそれを利用した単射性判定について述べる．その後，5.3.4 節において，よりよい補関数導出アルゴリズムを与える．この 5.3.4 節の補関数導出アルゴリズムは，単射関数以外の場合でも，式の値域を利用することで ALG-C の導出する補関数よりより小さい補関数を導出できる場合がある．

5.3.2　値域の推論

言語 VDL で定義された各関数の値域の推定は，linear top-down tree transducer の値域の推定 [Eng75] と同様に，正規木文法により行える．たとえば，例 5.4 における関数 add について，以下のように関数の引数を「忘れる」ことで，add の値域を厳密に推定できる．

$$\begin{aligned} T_{add} &\to Any \\ T_{add} &\to \mathsf{S}(T_{add}) \end{aligned}$$

ここで，T_{add} は関数 add の値域に対応する非終端記号である．また，Any は，全ての木 $\mathcal{T}_\Sigma$ を生成する非終端記号である．上にあるように，add の値域は全ての木になる．これは，$add(\mathsf{Z}, y) \mathrel{\hat{=}} y$ と，add が第一引数が Z ならば第二引数の値をそのまま返していることによる[2]．

形式的には，図 5.4 に示す導出関係 $\overset{\mathrm{RG}}{\rightsquigarrow}$ を用いて式の値域を推定するための正規木文法を導出する．導出 $e \overset{\mathrm{RG}}{\rightsquigarrow} R$ は，式 e より正規木文法の規則 R が導出されると，読む．また，導出 $f(p_1, \ldots, p_n) \mathrel{\hat{=}} e \overset{\mathrm{RG}}{\dashrightarrow} R$ は，関数定義規則 $f(p_1, \ldots, p_n) \mathrel{\hat{=}} e$ より正規木文法の規則 R が導出される，と読む．プログラム $\mathcal{P} = (\Sigma, Q, R)$ の各規則から正規木文法 $G = (\Sigma, N, R')$ の規則を

$$R' = \bigcup \{R' \mid r \overset{\mathrm{RG}}{\dashrightarrow} R', r \in R\} \cup \{Any \to \mathsf{C}(Any, \ldots, Any) \mid \mathsf{C} \in \Sigma\}$$

により定める．ただし，N は R' から自然に定まるのものとする．図 5.4 において，$\#e$ は，式 e に割り振られたプログラム中で一意な識別子を表す．識別子の割り振りにおいて，異なる位置に出現する同じ式同士を区別する．ただし，言語 VDL において，それらの式の値域は常に一致することに注意する．

図 5.4 による式の値域の推論は厳密である．すなわち以下が言える．

定理 5.2. プログラムに対し，図 5.4 の関係を用いて値域の推論のための正規木文法 G を定めたとする．このとき，任意のプログラム中の式 e について，

$$\mathsf{ran}(e) = [\![T_{\#e}]\!]_G$$

[2]関数 add の値域が自然数の集合ではないのは非直観的に思うかもしれない．これは，VDL が型なし言語であることによる．なお，以下のように定義された add の値域は自然数（Z と S で表現される）の集合になる．

$$\begin{aligned} add(\mathsf{Z}, y) &\mathrel{\hat{=}} idNat(y) & idNat(\mathsf{Z}) &\mathrel{\hat{=}} \mathsf{Z} \\ add(\mathsf{S}(x), y) &\mathrel{\hat{=}} \mathsf{S}(add(x, y)) & idNat(\mathsf{S}(x)) &\mathrel{\hat{=}} \mathsf{S}(idNat(x)) \end{aligned}$$

$$\frac{}{x \overset{\mathrm{RG}}{\rightsquigarrow} \{T_{\#x} \xrightarrow{*} Any\}}\textsc{Var}$$

$$\frac{e_1 \overset{\mathrm{RG}}{\rightsquigarrow} R_1 \quad \cdots \quad e_n \overset{\mathrm{RG}}{\rightsquigarrow} R_n}{\mathsf{C}(e_1, \ldots, e_n) \overset{\mathrm{RG}}{\rightsquigarrow} \{T_{\#\mathsf{C}(e_1,\ldots,e_n)} \to \mathsf{C}(T_{\#e_1}, \ldots, T_{\#e_n})\} \cup R_1 \cup \cdots \cup R_n}\textsc{Con}$$

$$\frac{}{f(x_1, \ldots, x_n) \overset{\mathrm{RG}}{\rightsquigarrow} \{T_{\#f(x_1,\ldots,x_n)} \to S_f\}}\textsc{Fun}$$

$$\frac{e \overset{\mathrm{RG}}{\rightsquigarrow} R}{f(p_1, \ldots, p_n) \mathrel{\hat{=}} e \overset{\mathrm{RG}}{\dashrightarrow} \{T_f \to T_{\#e}\} \cup R}\textsc{FunDef}$$

ただし，$\#e$ はプログラム中で各式 e に割り振られた一意な識別子である．

図 5.4. 言語 VDL から値域の推定のたの正規木文法を導出する導出関係

となる．また，任意のプログラム中の関数 f について，

$$\mathsf{ran}(f) = [\![T_f]\!]_G$$

となる．

証明. 上の定理は linear top-down tree transducer の値域が正規木言語になること [Eng75] と同様に証明できる．

以下を示す．

$$\forall v \in \mathcal{T}_\Sigma.\ (\exists\theta.\ e\theta \Downarrow v \Leftrightarrow T_{\#e} \xrightarrow{*} v)$$

まず，($\Rightarrow$) を，式の構造と評価に関する帰納法により示す．

基底：$e = x$**.**

このとき，$T_{\#e} \to Any$ と規則が e に対応して生成されている．ここで，$[\![Any]\!]_G = \mathcal{T}_\Sigma$ となることより，主張は明らか．

帰納：$e = \mathsf{C}(e_1, \ldots, e_n)$**.**

このとき，$T_{\#e} \to \mathsf{C}(T_{\#e_1}, \ldots, T_{\#e_n})$ となっている．評価の定義より，$e_1\theta \Downarrow v_1, \ldots e_n\theta \Downarrow v_n$ として $v = \mathsf{C}(v_1, \ldots, v_n)$ である．帰納法の仮定より，$i \in \{1, \ldots, n\}$ について $T_{\#e_i} \xrightarrow{*} v_i$ である．よって，生成の定義により，

$$T_{\#e} \to \mathsf{C}(T_{\#e_1}, \ldots, T_{\#e_n}) \xrightarrow{*} \mathsf{C}(v_1, \ldots, v_n)$$

となり，主張は真．

帰納：$e = f(\overrightarrow{x})$.

このとき，$f(\overrightarrow{x}\theta) \Downarrow v$ より，規則 $f(\overrightarrow{p}') \doteq e'$ で，ある代入 η に対し，$\overrightarrow{p}'\eta = \overrightarrow{x}\theta$ となるものが必ず存在している．評価の定義より，$e'\eta \Downarrow v$ となっている．また，G の構成より，G は生成規則 $T_{\#e} \rightarrow T_f$ と生成規則 $T_f \rightarrow T_{\#e'}$ を含む．帰納法の仮定より，$T_{\#e'} \rightarrow v$ である．よって，生成の定義により，

$$T_{\#e} \rightarrow T_f \xrightarrow{*} T_{\#e'} \xrightarrow{*} v$$

となり，主張は真.

次に，($\Leftarrow$) を式の構造と生成に関する帰納法により示す.

基底：$e = x$.

このとき，$\theta = x \mapsto v$ ととれば，$e\theta \Downarrow v$ となり，主張は真.

帰納：$e = \mathsf{C}(e_1, \ldots, e_n)$.

このとき，

$$T_{\#e} \rightarrow \mathsf{C}(T_{\#e_1}, \ldots, T_{\#e_n}) \xrightarrow{*} \mathsf{C}(v_1, \ldots, v_n) = v$$

である．帰納法の仮定より，各 $i \in \{1, \ldots, n\}$ について，$e_i\theta_i \Downarrow v_i$ となる θ_i が存在する．代入 θ を

$$\theta = \theta_1 \circ \cdots \circ \theta_n$$

と定めると，affine 制約より $e\theta = \mathsf{C}(e_1\theta_1, \ldots, e_n\theta_n)$ となる．よって，$e\theta \Downarrow v$ となり，主張は真.

帰納：$e = f(\overrightarrow{x})$.

規則 $f(\overrightarrow{p}') \doteq e'$ により，

$$T_{\#e} \rightarrow T_f \xrightarrow{*} T_{\#e'} \xrightarrow{*} v$$

となったとする．帰納法の仮定より，$e'\eta \Downarrow v$ となる代入 η が存在する．代入 η' を，何らかの木 $t_0 \in \mathcal{T}_\Sigma$ により

$$\eta'(x) = \begin{cases} \eta(x) & \text{if } x \in \mathsf{vars}(e') \\ t_0 & \text{otherwise} \end{cases}$$

と定める．このとき，代入 θ を $\overrightarrow{x}\theta = \overrightarrow{p}\eta'$ によって定めると，$f(\overrightarrow{x}\theta) \Downarrow v$ となり，主張は真. □

$$\frac{\exists f(\overrightarrow{p}) \mathrel{\hat{=}} e \in R.\ \mathsf{vars}(\overrightarrow{p}) \setminus \mathsf{vars}(e) \neq \emptyset}{f \in \mathit{NINJ}_{\mathcal{P}}}\text{NINJ-LostVar}$$

$$\frac{\exists f(\overrightarrow{p_1}) \mathrel{\hat{=}} e_1, f(\overrightarrow{p_2}) \mathrel{\hat{=}} e_2 \in R.\ \overrightarrow{p_1} \neq \overrightarrow{p_2} \wedge \mathsf{ran}(e_1) \cap \mathsf{ran}(e_2) = \emptyset}{f \in \mathit{NINJ}_{\mathcal{P}}}\text{NINJ-Overlap}$$

$$\frac{\exists f(\overrightarrow{p}) \mathrel{\hat{=}} \mathcal{C}[g(\overrightarrow{x})], g \in \mathit{NINJ}_{\mathcal{P}}}{f \in \mathit{NINJ}_{\mathcal{P}}}\text{NINJ-Call}$$

図 5.5. プログラム $\mathcal{P} = (\Sigma, Q, R)$ より，明らかに単射でない関数の集合 $\mathit{NINJ}_{\mathcal{P}}$ を定める関係

5.3.3 単射性判定

関数の単射性判定は双方向化において重要である．もし，順方向変換が単射であると判定できれば，その逆方向変換を逆関数により定められる．この逆関数により定められる逆方向変換は振る舞いがよく最も効果的なものである．また，この逆方向変換は，元の順方向変換に対し補関数として定数関数を用いることで得られるものである．単射な順方向変換に対し，逆関数により定められる逆方向変換が導出されるのは，双方向化の振る舞いとして直観的である．

我々は，5.3.1 節における補関数の素朴な導出において，次の二点に着目した．

- 右辺に出現しない変数
- 使用する規則の識別

これは，上の二点の情報は返り値から得られるとは限らず，関数の単射性を失う原因になりうるためである．同様に，この二点に着目することにより，Vdl で記述された関数の単射性判定を行える．

プログラム $\mathcal{P}$ に対し，図 5.3.3 に定める関係の最小不動点により，明らかに単射でない関数の集合 $\mathit{NINJ}_{\mathcal{P}}$ が求まる．図 5.3.3 において，それぞれの規則は以下を表している．

- 規則 NINJ-LostVar は，右辺に出現しない変数をもつ関数は単射ではないことを表している．たとえば関数 fst

$$\mathit{fst}(x, y) \mathrel{\hat{=}} y$$

  は，変数 y が右辺に出現しないため，単射でない．

- 規則 NINJ-Overlap は， 二つの右辺式の値域に重なりがある関数は単射ではないことを示している．たとえば，関数 $\mathit{AlwaysTrue}$

$$\begin{aligned} \mathit{alwaysTrue}(\mathsf{True}) &\mathrel{\hat{=}} \mathsf{True} \\ \mathit{alwaysTrue}(\mathsf{False}) &\mathrel{\hat{=}} \mathsf{True} \end{aligned}$$

は，第一規則と第二規則の右辺式がともに True と式の値域に重なりがあるため，単射でない．

- 規則 NINJ-Call は，単射でない関数を呼び出している関数は単射ではないことを示している．たとえば，関数 mapfst

$$\begin{array}{ll} \mathit{mapfst}([]) & \hat{=}\ [] \\ \mathit{mapfst}(x:r) & \hat{=}\ \mathit{fsrPair}(x):\mathit{mapfst}(r) \\ \mathit{fstPair}(\mathsf{Pair}(x,y)) & \hat{=}\ x \end{array}$$

は，単射でない関数 $\mathit{fstPair}$ を呼び出しているため単射ではない．

定理 5.3 (単射性判定の健全性)**.** もし $[\![f]\!]_{\mathcal{P}}$ が非単射ならば，$f \in \mathit{NINJ}_{\mathcal{P}}$ である．

証明. $\mathcal{P} = (\Sigma, Q, R)$ とする．我々は以下を示す．

$$\exists \overrightarrow{v}, \overrightarrow{v}', u.\ \overrightarrow{v} \neq \overrightarrow{v}' \wedge f(\overrightarrow{v}) \Downarrow u \wedge f(\overrightarrow{v}') \Downarrow u \Rightarrow f \in \mathit{NINJ}_{\mathcal{P}} \qquad \text{(INJ-Sound)}$$

今，ある $\overrightarrow{v}$，$\overrightarrow{v}'$ および u に対し，$f(\overrightarrow{v}) \Downarrow u$ かつ $f(\overrightarrow{v'}) \Downarrow u$ となったとする．このとき，以下の二つの場合しかない．

場合 1. $f(\overrightarrow{v}) \Downarrow u$ の証明木と $f(\overrightarrow{v}') \Downarrow u$ の証明木が変数への束縛を除いて等しい場合．つまり，二つの証明木がそれぞれの Fun 節点において同じ関数定義規則を用いている場合．

場合 2. それ以外．

場合1について，我々は主張 (INJ-Sound) を $\#\textsc{FunDepth}(f(\overrightarrow{v}))$ についての帰納法により示す．ここで，$\#\textsc{FunDepth}(e)$ は，式 e を評価するのに使用した Fun 規則の数を $e \Downarrow v$ の証明木に沿って縦に数えたもの最大値，つまり証明木の深さを表す．

基底： $\#\textsc{FunDepth}(f(\overrightarrow{v})) = \#\textsc{FunDepth}(f(\overrightarrow{v}')) = 0$**.**

このとき，規則 $r \in R$

$$r = f(\overrightarrow{p}) \mathrel{\hat{=}} \mathcal{C}[\overrightarrow{x}]$$

で，$\overrightarrow{p}\theta = \overrightarrow{v}$ かつ $\overrightarrow{p}\theta' = \overrightarrow{v}'$ となる代入 θ, θ' に対し，$\mathcal{C}[\overrightarrow{x}]\theta = \mathcal{C}[\overrightarrow{x}]\theta'$ となるものが存在する．もし，$\mathsf{lostvars}(r) = \emptyset$ であったとする．すると $\overrightarrow{x}\theta = \overrightarrow{x}\theta'$ より，$\overrightarrow{p}\theta = \overrightarrow{p}\theta'$ が言える．これは $\overrightarrow{v} \neq \overrightarrow{v}'$ に矛盾する．よって $\mathsf{lostvars}(r) \neq \emptyset$ である．これは，NINJ-LostVar の条件であるため，$f \in \mathit{NINJ}_{\mathcal{P}}$ となる．

帰納：$\#\text{FunDepth}(f(\overrightarrow{v})) = \#\text{FunDepth}(f(\overrightarrow{v}')) > 0$**.**

このとき，規則 $r \in R$

$$r = f(\overrightarrow{p}) \hat{=} \mathcal{C}[f_1(\overrightarrow{x_1}), \ldots, f_l(\overrightarrow{x_n}), \overrightarrow{x}]$$

で，ある代入 θ, θ' に対し，$\overrightarrow{p}\theta = \overrightarrow{v}$ かつ $\overrightarrow{p}\theta' = \overrightarrow{v}'$ となるものが存在する．ここで，$i \in \{1, \ldots, n\}$ について，$f_i(\overrightarrow{x_i}\theta)$ および $f_i(\overrightarrow{x_i}\theta')$ の評価結果を w_i と書く．すなわち，$f_i(\overrightarrow{x_i}\theta) \Downarrow w_i$ かつ $f_i(\overrightarrow{x_i}\theta') \Downarrow w_i$ である．これにより，$f(\overrightarrow{v})$ と $f(\overrightarrow{v}')$ の結果は，それぞれ $f(\overrightarrow{v}) \Downarrow \mathcal{C}[w_1, \ldots, w_n, \overrightarrow{x}\theta]$ および $f(\overrightarrow{v}') \Downarrow \mathcal{C}[w_1, \ldots, w_n, \overrightarrow{x}\theta']$ と書ける．まず，もし $\mathsf{lostvars}(r) \neq \emptyset$ ならば，NINJ-LostVar の条件より，$f \in \mathit{NINJ}_{\mathcal{P}}$ となる．次に，$\mathsf{lostvars}(r) = \emptyset$ である場合を考える．ここで，$\overrightarrow{p}\theta \neq \overrightarrow{p}\theta'$ かつ $\overrightarrow{x}\theta = \overrightarrow{x}\theta'$ であるため，$\overrightarrow{x_i}\theta \neq \overrightarrow{x_i}\theta'$ となる i が存在する．ただし，$f_i(\overrightarrow{x_i}\theta)$ と $f_i(\overrightarrow{x_i}\theta')$ の評価結果は一致するため，f_i は非単射である．このとき，帰納法の仮定より，$f_i \in \mathit{NINJ}_{\mathcal{P}}$ である．これは，NINJ-Call の条件であるため，$f \in \mathit{NINJ}_{\mathcal{P}}$ となる．

場合 2 に対し，我々は主張 (INJ-Sound) を $\#\text{FunDepth}(f(\overrightarrow{v})) + \#\text{FunDepth}(f(\overrightarrow{v}'))$ の上の帰納法で示す.

基底：$\#\text{FunDepth}(f(\overrightarrow{v})) + \#\text{FunDepth}(f(\overrightarrow{v}')) = 0$**.**

このとき，二つの異なった規則 $r, r' \in R$

$$r = f(\overrightarrow{p}) \hat{=} \mathcal{C}[\overrightarrow{x}]$$
$$r' = f(\overrightarrow{p}') \hat{=} \mathcal{C}'[\overrightarrow{x}']$$

で，ある代入 θ, θ' に対し，$\overrightarrow{p}\theta = \overrightarrow{v}$ かつ $\overrightarrow{p}'\theta' = \overrightarrow{v}'$ かつ $\overrightarrow{v} \neq \overrightarrow{v}'$ であり，$\mathcal{C}[\overrightarrow{x}]\theta = \mathcal{C}'[\overrightarrow{x'}]\theta'$ となるものが存在する．つまり，異なる f の規則の右辺式 $\mathcal{C}[\overrightarrow{x}]$ と $\mathcal{C}'[\overrightarrow{x}']$ の値域に重なりがある．これは，NINJ-Overlap の条件であるため，$f \in \mathit{NINJ}_{\mathcal{P}}$ となる．

帰納：$\#\text{FunDepth}(f(\overrightarrow{v})) = 0$ **もしくは** $\#\text{FunDepth}(f(\overrightarrow{v}')) = 0$**.**

この場合も使用した二つの規則が異なることから，上と同様に，主張 (INJ-Sound) は証明される．

帰納：$\#\text{FunDepth}(f(\overrightarrow{v})) > 0, \#\text{FunDepth}(f(\overrightarrow{v}')) > 0$**.**

このとき，以下の二つの場合がある．

- 関数 f の二つの異なる規則で，その左辺 $\overrightarrow{p}, \overrightarrow{p}'$ について，ある θ, θ' に対し，$\overrightarrow{p}\theta = \overrightarrow{v}$ かつ $\overrightarrow{p}'\theta' = \overrightarrow{v}'$ となるものが存在する．

- 評価 $f(\overrightarrow{v}) \Downarrow u$ と $f(\overrightarrow{v'}) \Downarrow u$ との証明木において，それぞれの根で同じ規則を使用している．

第一の場合については，二つの規則が異なるため，これまでと同様の議論により主張 (INJ-Sound) を証明できる．よって，第二の場合について考える．このとき，規則 $r \in R$

$$r = f(\overrightarrow{p}) \doteq \mathcal{C}[f_1(\overrightarrow{x_1}), \ldots, f_n(\overrightarrow{x_n}), \overrightarrow{x}]$$

で，ある代入 θ, θ' に対し，$\overrightarrow{v} = \overrightarrow{p}\theta$ かつ $\overrightarrow{v}' = \overrightarrow{p}\theta'$ であり $\overrightarrow{v} \neq \overrightarrow{v}'$ となるものが存在する．このとき，$i \in \{1, \ldots, n\}$ について，$f_i(\overrightarrow{x_i}\theta)$ および $f_i(\overrightarrow{x_i}\theta')$ の評価結果を w_i と書く．すなわち，$f_i(\overrightarrow{x_i}\theta) \Downarrow w_i$ であり $f_i(\overrightarrow{x_i}\theta') \Downarrow w_i$ である．仮定より，どれかの i について，$f_i(\overrightarrow{x_i}\theta)$ と $f_i(\overrightarrow{x_i}\theta')$ の証明木は異なる．これは，また $\overrightarrow{x_i}\theta \neq \overrightarrow{x_i}\theta'$ であることを意味する．すなわち，f_i は単射ではない．このとき，帰納法の仮定により，NINJ-Call から，$f \in \mathit{NINJ}_{\mathcal{P}}$ である． □

定理 5.4 (単射性判定の厳密性)**.** もし $f \in \mathit{NINJ}_{\mathcal{P}}$ であるならば，$[\![f]\!]_{\mathcal{P}}$ は単射ではない．

証明. $\mathcal{P} = (\Sigma, Q, R)$ とする．集合 $\mathit{NINJ}_{\mathcal{P}}$ は図 5.3.3 の関係の最小不動点により定まっているため，図 5.3.3 の関係についての帰納法により示す．

基礎： NINJ-LostVars

このとき，規則 $r \in R$

$$r = f(\overrightarrow{p}) \doteq e$$

で，$\mathsf{lostvars}(r) \neq \emptyset$ となるものが存在する．言語 Vdl に対する仮定より，ある代入 θ が存在して，ある値 u に対し $f(\overrightarrow{p}\theta) \Downarrow u$ となる．これは，e 中に出現するどの関数も，何らかの入力に対しては定義されているためである．ここで，ある互いに異なる木 $t_1, t_2 \in \mathcal{T}_\Sigma$ に対し，θ_1, θ_2 を

$$\theta_1(x) = \begin{cases} t_1 & \text{if } x \in \mathsf{lostvars}(r) \\ \theta(x) & \text{if } x \notin \mathsf{lostvars}(r) \end{cases}$$

$$\theta_2(x) = \begin{cases} t_2 & \text{if } x \in \mathsf{lostvars}(r) \\ \theta(x) & \text{if } x \notin \mathsf{lostvars}(r) \end{cases}$$

と定義すると，$e\theta = e\theta_1 = e\theta_2$ となる．よって，$f(\overrightarrow{p}\theta_1) \Downarrow u$ かつ $f(\overrightarrow{p}\theta_2) \Downarrow u$ である．ところが，$\overrightarrow{p}\theta_1 \neq \overrightarrow{p}\theta_2$ である．よって，f は非単射である．

基礎： NINJ-Overlap

このとき，二つの規則 $r, r' \in R$

$$r = f(\overrightarrow{p}) \mathrel{\hat{=}} e$$
$$r' = f(\overrightarrow{p'}) \mathrel{\hat{=}} e'$$

が存在して，ある η, η' および u に対し $e\eta \Downarrow u$ かつ $e'\eta' \Downarrow u$ となる．このとき，ある木 $t_0 \in \mathcal{T}_\Sigma$ に対し，以下のように，θ_1 と θ_2 を定める．

$$\theta_1(x) = \begin{cases} t_0 & \text{if } x \in \mathsf{lostvars}(r) \\ \eta_1(x) & \text{if } x \notin \mathsf{lostvars}(r) \end{cases}$$

$$\theta_2(x) = \begin{cases} t_0 & \text{if } x \in \mathsf{lostvars}(r') \\ \eta_2(x) & \text{if } x \notin \mathsf{lostvars}(r') \end{cases}$$

すると，$f(\overrightarrow{p}\theta_1) \Downarrow u$ かつ $f(\overrightarrow{p'}\theta_2) \Downarrow u$ である．ところが，$\overrightarrow{p}\theta_1 \neq \{p'\}\theta_2$ である．よって，f は非単射である．

帰納： NINJ-Call

このとき，規則 $r \in R$

$$r = f(\overrightarrow{p}) \mathrel{\hat{=}} \mathcal{C}[g(\overrightarrow{x})]$$

で，$g \in \mathit{NINJ}_\mathcal{P}$ となっているものが存在する．また，言語 Vdl に対する仮定より，ある代入 θ が存在して，ある値 u に対し $f(\overrightarrow{p}\theta) \Downarrow u$ となる．さらに，帰納法の仮定により，g は非単射である．すなわち，ある二つの異なる入力 $\overrightarrow{w_1}$ と $\overrightarrow{w_2}$ に対し，ある値 u が存在し，$g(\overrightarrow{w_1}) \Downarrow u$ かつ $g(\overrightarrow{w_2}) \Downarrow u$ となる．代入 η_1, η_2 を，$\overrightarrow{x}\eta_1 = \overrightarrow{w_1}$ と $\overrightarrow{x}\eta_2 = \overrightarrow{w_2}$ により定める．ここで，代入 $\theta_1\theta_2$ を以下のように定める．

$$\theta_1(x) = \begin{cases} \eta_1(x) & \text{if } x \in \{\overrightarrow{x}\} \\ \theta(x) & \text{otherwise} \end{cases}$$

$$\theta_1(x) = \begin{cases} \eta_2(x) & \text{if } x \in \{\overrightarrow{x}\} \\ \theta(x) & \text{otherwise} \end{cases}$$

すると，ある値 u' に対し，$f(\overrightarrow{p}\theta_1) \Downarrow u'$ かつ $f(\overrightarrow{p}\theta_2) \Downarrow u'$ である．ところが，$\overrightarrow{p}\theta_1 \neq \{p\}\theta_2$ である．よって，f は非単射である． □

5.3.4　よりよい補関数の導出

これまでの議論から，各式についてその値域を厳密に推定することができ，また，厳密に推定した値域より関数単射性を厳密に判定できることがわかった．これからすぐに言えることは，我々は，全ての単射関数に対し，我々は最小の補関数を導出できることである．関数が単射でない場合においても，いくつかの例については式の値域を使用することでより小さい補関数を導出することができる．

たとえば，以下の関数 $half$ を考える．

$$\begin{aligned} half(\mathsf{Z}) &\hat{=} \mathsf{Z} \\ half(\mathsf{S}(\mathsf{Z})) &\hat{=} \mathsf{Z} \\ half(\mathsf{S}(\mathsf{S}(x))) &\hat{=} \mathsf{S}(half(x)) \end{aligned}$$

関数 $half$ は入力の自然数を二で割り，余りを切り捨てる．関数 $half$ に対し，アルゴリズム ALG-C は以下の補関数を導出する．

$$\begin{aligned} half^{\mathrm{c}}(\mathsf{Z}) &\hat{=} \mathsf{B}_1 \\ half^{\mathrm{c}}(\mathsf{S}(\mathsf{Z})) &\hat{=} \mathsf{B}_2 \\ half^{\mathrm{c}}(\mathsf{S}(\mathsf{S}(x))) &\hat{=} \mathsf{B}_3(half^{\mathrm{c}}(x)) \end{aligned}$$

ところが，$half$ において第三規則の右辺式と第一規則の右辺式，第三規則の右辺式と第二規則の右辺式にはそれぞれ重なりがない．そのため，第三規則をいつ何回使用したかという情報は，$half$ の返り値より知ることができる．実際，$half$ の返す S の数と $half^{\mathrm{c}}$ の返す B_3 の数は常に等しい．つまり，B_3 は必要がなく，以下の関数も $half$ の補関数である．

$$\begin{aligned} half^{\mathrm{c}}(\mathsf{Z}) &\hat{=} \mathsf{B}_1 \\ half^{\mathrm{c}}(\mathsf{S}(\mathsf{Z})) &\hat{=} \mathsf{B}_2 \\ half^{\mathrm{c}}(\mathsf{S}(\mathsf{S}(x))) &\hat{=} half^{\mathrm{c}}(x) \end{aligned}$$

また，別の例として自然数の前者を求める関数 $pred$ を考える．

$$\begin{aligned} pred(\mathsf{Z}) &\hat{=} \mathsf{Z} \\ pred(\mathsf{S}(\mathsf{Z})) &\hat{=} \mathsf{Z} \\ pred(\mathsf{S}(\mathsf{S}(x))) &\hat{=} \mathsf{S}(x) \end{aligned}$$

このとき，アルゴリズム ALG-C は，以下の補関数を導出する．

$$\begin{aligned} pred^{\mathrm{c}}(\mathsf{Z}) &\hat{=} \mathsf{B}_1 \\ pred^{\mathrm{c}}(\mathsf{S}(\mathsf{Z})) &\hat{=} \mathsf{B}_2 \\ pred^{\mathrm{c}}(\mathsf{S}(\mathsf{S}(x))) &\hat{=} \mathsf{B}_3 \end{aligned}$$

ところで，上の補関数のうち，B_2 と B_3 は統一し B_2 にしてもよい．なぜなら，第二規則の右辺式と第三規則の右辺式の値域は互いに異なっているためである．つまり，以下も $pred$ の補関数である．

$$\begin{aligned} pred^{\mathrm{c}}(\mathsf{Z}) &\hat{=} \mathsf{B}_1 \\ pred^{\mathrm{c}}(\mathsf{S}(\mathsf{Z})) &\hat{=} \mathsf{B}_2 \\ pred^{\mathrm{c}}(\mathsf{S}(\mathsf{S}(x))) &\hat{=} \mathsf{B}_2 \end{aligned}$$

値域の情報を利用しよりよい補関数を求めるアルゴリズムについて述べる前に，アルゴリズムが利用する規則集合の分割について述べる．プログラム $\mathcal{P} = (\Sigma, Q, R)$ について，規則集合 R を $R = R_1 \uplus R_2 \uplus \cdots \uplus R_k$ という互いに素な集合に分割する．ただし，それぞれの集合 R_i は異なる二つの規則 $r, r' \in R_i$ について以下の三条件全てを満たす．

- r と r' は同じ関数の規則である．
- r と r' に対応する構成子（後述のアルゴリズムで定まる）の引数の個数が同じ．
- r と r' の右辺式の値域が重ならない．

この分割の目的は，順方向変換で関数が使用した規則を構成子により憶える際，同じ構成子を使用可能な規則を定めることにある．そのため，同じ関数の規則について考え，引数の数が同じものしか同じ集合に属さないようにし，同じ構成子を使用しても使用規則が区別できることを保証している．

たとえば，$r_1 = add(\mathsf{Z}, y) \mathrel{\hat{=}} y, r_2 = add(\mathsf{S}(x), y) \mathrel{\hat{=}} \mathsf{S}(add(x, y))$ については，$R = \{r_1\} \uplus \{r_2\}$ という分割しか存在しないが，$r'_1 = neg(\mathsf{True}) \mathrel{\hat{=}} \mathsf{False}, r'_2 = neg(\mathsf{False}) \mathrel{\hat{=}} \mathsf{Ttrue}$ に対しては，$R = \{r'_1\} \uplus \{r'_2\}$ だけでなく $R = \{r'_1, r'_2\}$ という分割が存在する．一般に，より粗い分割に対し Alg-Sc はより小さい補関数を返す．ただし，$r''_1 = f(\mathsf{A}) \mathrel{\hat{=}} \mathsf{True}, r''_2 = f(\mathsf{B}) \mathrel{\hat{=}} \mathsf{False}, r''_3 = f(\mathsf{C}) \mathrel{\hat{=}} \mathsf{False}$ に対する分割 $R = \{r''_1, r''_2\} \uplus \{r''_3\}$ と分割 $R = \{r''_1, r''_3\} \uplus \{r''_2\}$ のように粗さが比較不能な場合もある．

補関数の導出アルゴリズム Alg-C の改良版アルゴリズム Alg-Sc を以下に示す．規則集合の分割には任意性があるが，具体的な分割方法については本研究では触れず，ユーザもしくはシステムが与えものとする．

アルゴリズム 5.2 (補関数の導出（改良版）：Alg-Sc)．
入力： プログラム $\mathcal{P} = (\Sigma, Q, R)$ および分割 $R = R_1 \uplus R_2 \uplus \cdots \uplus R_k$
出力： 補関数を定義するプログラム P^{c}
手続き：

1. それぞれの R_j 中のそれぞれの規則 $r \in R_j$

$$r = f(\overrightarrow{p}) \mathrel{\hat{=}} \mathcal{C}[f_1(\overrightarrow{x_1}), \ldots, f_n(\overrightarrow{x_n}), \overrightarrow{x_0}]$$

について以下の処理を行う．

(a) もし，$f \notin \mathit{NINJ}_{\mathcal{P}}$ ならば，

$$r^{\mathrm{c}} = \mathsf{B}$$

とする．

(b) そうでないならば，以下により規則 r^{c} を構成する．

i. 規則 $r^{\mathrm{c}}_{\mathrm{pre}}$ を以下のように構成する．

$$r^{\mathrm{c}}_{\mathrm{pre}} = f^{\mathrm{c}}(\overrightarrow{p}) \mathrel{\hat{=}} \mathsf{B}_j(g_1(\overrightarrow{y_1}), \ldots, g_m(\overrightarrow{y_m}), \overrightarrow{y_0})$$

ただし，$\{g_i(\overrightarrow{y_i})\}_{i\in\{1,\ldots,m\}}$ は $\{f_i(\overrightarrow{x_i})\}_{i\in\{1,\ldots,n\}}$ から単射な関数の呼び出しを取り除いた後に，それぞれの関数呼び出し $f_i(\overrightarrow{x_i})$ を補関数の呼び出し $f_i{}^{\mathrm{c}}(\overrightarrow{x_i})$ で置き換えたものである．また，$\{\overrightarrow{y_0}\} = \mathsf{lostvars}(r)$ である．

ii. もし，$r^{\mathrm{c}}_{\mathrm{pre}}$ が以下の形式

$$r^{\mathrm{c}}_{\mathrm{pre}} = f^{\mathrm{c}}(\overrightarrow{p}) \mathrel{\hat{=}} \mathsf{B}_j(e)$$

をしていて，r の右辺式の値域が，r 以外の f を定義する任意の規則 $r' \in \mathcal{R}$ の右辺式の値域と重ならないなら，

$$r^{\mathrm{c}} = f^{\mathrm{c}}(\overrightarrow{p}) \mathrel{\hat{=}} e$$

とし，それ以外の場合は

$$r^{\mathrm{c}} = r^{\mathrm{c}}_{\mathrm{pre}}$$

とする．

2. 規則 r^{c} を集めて R^{c} を構成し，プログラム $\mathcal{P}^{\mathrm{c}}$ を得る．　□

定理 5.5 (Alg-Sc の正しさ)**.** プログラム $P = (\Sigma, Q, R)$ に対し，Alg-Sc によりプログラム $P^{\mathrm{c}} = (\Sigma^{\mathrm{c}}, Q^{\mathrm{c}}, R^{\mathrm{c}})$ が得られたとする．このとき，全ての関数記号 $f \in Q$ に対し $[\![f^{\mathrm{c}}]\!]$ は $[\![f]\!]$ の補関数である．

証明の概略. ここでは，証明の概略のみを述べる．これは，Alg-Sc は Alg-C の改良であるため，Alg-Sc の正しさ（定理 5.5）の証明も Alg-C の正しさ（定理 5.1）の証明とほぼ同様であるためである．アルゴリズム Alg-Sc において，我々は以下の改良を行った．

- 単射関数に対応する補関数呼出の削除
- 規則を識別するための構成子の統一
- 規則を識別するための一引数構成子の除去

定理 5.1 の証明において，我々は値 v, v' と規則 r, r'

$$r = f(\overrightarrow{p}) \mathrel{\hat{=}} \mathcal{C}[f_1(\overrightarrow{x_1}), \ldots, f_n(\overrightarrow{x_n}), \overrightarrow{x}]$$
$$r' = f(\overrightarrow{p'}) \mathrel{\hat{=}} \mathcal{C}'[f'_1(\overrightarrow{x'_1}), \ldots, f'_{n'}(\overrightarrow{x'_{n'}}), \overrightarrow{x'}]$$

で，$\overrightarrow{p}\theta = \overrightarrow{v}$ かつ $\overrightarrow{p'}\theta' = \overrightarrow{v'}$ であり $[\![f]\!](\overrightarrow{v}) = [\![f]\!](\overrightarrow{v'})$ となるものについて議論した．つまり，r の r' の右辺式の値域が重なりがあるものについてである．ここで，r の r' については

構成子の統一や除去は行われていない，そのため，r^{c} と r'^{c} は単射関数に対応する補関数の呼出が削除されている以外は ALG-C の導出する規則と同じ形式をしている．

よって，定理 5.1 の証明に少し手を加えることで，アルゴリズム ALG-SC の正しさを証明できる．単射な f について，もし $[\![f]\!](\overrightarrow{u}) = [\![f]\!](\overrightarrow{u'})$ となるならば，$\overrightarrow{u} = \overrightarrow{u'}$ となることに注意する． □

アルゴリズム ALG-SC の効果をいくつかの例により確認する．

例 5.9 (規則集合の分割の役割)**.** 以下の関数 f を考える．

$$\begin{aligned} r_1 &= f(\mathsf{A}_1) \mathrel{\hat{=}} \mathsf{C}_1 \\ r_2 &= f(\mathsf{A}_2) \mathrel{\hat{=}} \mathsf{C}_2 \\ r_3 &= f(\mathsf{A}_3) \mathrel{\hat{=}} \mathsf{C}_1 \end{aligned}$$

ここで，上のプログラムと分割 $R = \{r_1, r_2\} \uplus \{r_3\}$ に対し，アルゴリズム ALG-SC は以下の補関数を返す.

$$\begin{aligned} f^{\mathrm{c}}(\mathsf{A}_1) &\mathrel{\hat{=}} \mathsf{B}_1 \\ f^{\mathrm{c}}(\mathsf{A}_2) &\mathrel{\hat{=}} \mathsf{B}_1 \\ f^{\mathrm{c}}(\mathsf{A}_3) &\mathrel{\hat{=}} \mathsf{B}_2 \end{aligned}$$

対し，もし分割 $R = \{r_1\} \uplus \{r_2, r_3\}$ を用いた場合には，アルゴリズム ALG-SC は以下の補関数を返す.

$$\begin{aligned} f^{\mathrm{c}}(\mathsf{A}_1) &\mathrel{\hat{=}} \mathsf{B}_1 \\ f^{\mathrm{c}}(\mathsf{A}_2) &\mathrel{\hat{=}} \mathsf{B}_2 \\ f^{\mathrm{c}}(\mathsf{A}_3) &\mathrel{\hat{=}} \mathsf{B}_2 \end{aligned}$$

このように，規則の分割が異なれば，導出される補関数も異なる．

例 5.10 (単射関数の補関数)**.** 以下の関数 mapneg を考える．

$$\begin{aligned} &\mathit{mapneg}(a : x) &&\mathrel{\hat{=}} \mathit{neg}(a) : \mathit{mapneg}(x) \\ &\mathit{mapneg}([\,]) &&\mathrel{\hat{=}} [\,] \\ &\mathit{neg}(\mathsf{True}) &&\mathrel{\hat{=}} \mathsf{False} \\ &\mathit{neg}(\mathsf{False}) &&\mathrel{\hat{=}} \mathsf{True} \end{aligned}$$

上の関数 mapneg は単射である．よって，単射性判定により，$\mathit{mapneg} \notin \mathit{NINJ}_{\mathcal{P}}$ となる．アルゴリズム ALG-SC は単射と判定された関数については，補関数として定数関数を返す.

$$\begin{aligned} &\mathit{mapneg}^{\mathrm{c}}(a : x) &&\mathrel{\hat{=}} \mathsf{B} \\ &\mathit{mapneg}^{\mathrm{c}}([\,]) &&\mathrel{\hat{=}} \mathsf{B} \\ &\mathit{neg}^{\mathrm{c}}(\mathsf{True}) &&\mathrel{\hat{=}} \mathsf{B} \\ &\mathit{neg}^{\mathrm{c}}(\mathsf{False}) &&\mathrel{\hat{=}} \mathsf{B} \end{aligned}$$

定数関数は，補関数として最も小さいものである．

例 5.11 (規則識別のための構成子の除去)．例 5.5 における関数 zip を考える．アルゴリズム Alg-Sc は関数 zip について以下を返す．

$$
\begin{aligned}
zip^{c}([], y) &\mathrel{\hat{=}} \mathsf{B}_1(y)\\
zip^{c}(a : x, []) &\mathrel{\hat{=}} \mathsf{B}_2(a, x)\\
zip^{c}(a : x, b : y) &\mathrel{\hat{=}} zip^{c}(x, y)
\end{aligned}
$$

これは，zip の極小の補関数の一つである．上の zip^{c} の定義において，アルゴリズム Alg-Sc により，第三規則を識別するための構成子が取り除かれている．

アルゴリズム Alg-Sc には次の二つの利点がある．一つ目の利点は，導出される補関数が元の関数と同じ形式をしていることである．このため，組化 [HITT97, Chi93] が必ず成功することが保証される．二つ目の利点は，後に更新の反映可能性検査器の導出の議論が示すように，逆方向変換の定義域が取り扱いやすいクラスであることである．

5.4　逆方向変換の導出

順方向変換 f に対し，補関数 f^{c} が得られた後は，第 3 章の式 (RFL) に基づき，補関数不変の逆方向変換 f_{B} を以下のように構成できる．

$$f_{\mathrm{B}}(s, v) \mathrel{\hat{=}} \langle f, f^{c}\rangle^{-1}(v, f^{c}(s))$$

つまり，$\langle f, f^{c}\rangle$ と $\langle f, f^{c}\rangle^{-1}$ が効果的に求まれば，逆方向変換も効果的に求まる．

関数 h が

$$h(x) = (f(x), g(x))$$

の形式で定義されている場合において，h の逆関数を求めるのに f と g を組化 [HITT97, Chi93] することが効果的であることが知られている [Epp85, GK05]．そのため，我々はまず，組化を行い，その後逆関数の導出を行う．これらのステップはともに自動的に行うことができる．

5.4.1　組化：関数 $\langle f, f^{c}\rangle$ のプログラムの導出

関数 fst に対し，Alg-Sc は関数 $fst^{c}(x, y) \mathrel{\hat{=}} y$ を求める．逆方向変換の導出のために求めたいのは，

$$h(x) = (fst(x), fst^{c}(x))$$

となる関数 h の逆関数である．ところが，一般には h の逆関数を直接求めるのは容易ではない．なぜなら，fst も fst^{c} も単射ではないため，これらの関数に対し個別に逆関数を与えることはできないためである．また，逆写像は定義できるものの $fst^{-1}(v))$ の値も $fst^{c-1}(v)$ の値も集合になるため，集合の積を計算し，また得られた積の中から唯一の元を得なければならない．これは集合がともに正規木言語で記述される場合には実行可能であることが知ら

れている [CDG+97]．しかし，言語 VDL では，関数が複数個の引数を取ることを許しているため，関数の値域は正規木言語よりも複雑になる．たとえば，以下の関数 f の定義域は右の子の深さと左の子の深さが同じである木であるが，これは正規木言語ではない．

$$\begin{array}{ll} f(\mathsf{C}(x,y)) & \hat{=}\ g(x,y) \\ g(\mathsf{Z},\mathsf{Z}) & \hat{=}\ \mathsf{Z} \\ g(\mathsf{S}(x),\mathsf{S}(y)) & \hat{=}\ \mathsf{S}(g(x)) \end{array}$$

そのため，我々は組化 [HITT97, Chi93] によりプログラムを変換することで，逆関数の導出のしやすい関数へと変換する．たとえば，fst と $\mathit{fst}^{\mathrm{c}}$ とを組化すると以下を得る．

$$\langle \mathit{fst}, \mathit{fst}^{\mathrm{c}} \rangle (x,y) \ \hat{=}\ (x,y)$$

上の関数 $\langle \mathit{fst}, \mathit{fst}^{\mathrm{c}} \rangle (x,y)$ は単射でない関数を呼び出していないため，その逆関数の導出は組化前と比べ簡単である．

組化は，組化したい関数同士の再帰が同じ形式をしているとき，必ず成功することが知られている．すなわち，それぞれのにおいて対応している関数同士が同じパターンを用い，同じ引数で呼び出されることである．我々は，逆関数の導出を容易にするために関数とその補関数を組化したい．アルゴリズム ALG-SC の導出する補関数は，非単射な関数については，関数とその補関数が必ず同じ形をしている．なお，単射な関数についてはそもそも組化をする必要がないことに注意する．

もう少し複雑な例として，例 5.5 の関数 zip と ALG-SC の導出する例 5.11 の補関数 $\mathit{zip}^{\mathrm{c}}$ を組化し，関数 $\langle \mathit{zip}, \mathit{zip}^{\mathrm{c}} \rangle$ の定義を求めることを考える．関数 zip（$\mathit{zip}^{\mathrm{c}}$）は第三規則おいて zip（$\mathit{zip}^{\mathrm{c}}$）を再帰的に呼んでいる．よって，$\langle \mathit{zip}, \mathit{zip}^{\mathrm{c}} \rangle$ を用いてこれらの関数呼出を書き直すと，$\langle \mathit{zip}, \mathit{zip}^{\mathrm{c}} \rangle$ の返り値の第一要素と第二要素を得るための構文が必要になる．我々は，これを表現するのに **let** 束縛を用いる．すなわち，$\langle \mathit{zip}, \mathit{zip}^{\mathrm{c}} \rangle$ の第三規則を以下のように求める．

$$\begin{array}{c} \mathit{zip}(a:x, b:y) \ \hat{=}\ \mathsf{Pair}(a,b) : \mathit{zip}(x), \quad \mathit{zip}^{\mathrm{c}}(a:x,b:y) \ \hat{=}\ \mathit{zip}^{\mathrm{c}}(x) \\ \downarrow \\ \langle \mathit{zip}, \mathit{zip}^{\mathrm{c}} \rangle (a:x, b:y) \ \hat{=}\ \mathbf{let}\ (s,t) \ \hat{=}\ \langle \mathit{zip}, \mathit{zip}^{\mathrm{c}} \rangle (x)\ \mathbf{in}\ (\mathsf{Pair}(a,b) : \mathit{zip}(x), t) \end{array}$$

関数 zip と補関数 $\mathit{zip}^{\mathrm{c}}$ に組化を適用すると以下が得られる．

$$\begin{array}{ll} \langle \mathit{zip}, \mathit{zip}^{\mathrm{c}} \rangle ([], y) & \hat{=}\ ([], \mathsf{B}_1(y)) \\ \langle \mathit{zip}, \mathit{zip}^{\mathrm{c}} \rangle (a:x, []) & \hat{=}\ ([], \mathsf{B}_2(a,x)) \\ \langle \mathit{zip}, \mathit{zip}^{\mathrm{c}} \rangle (a:x, b:y) & \hat{=}\ \mathbf{let}\ (s,t) \ \hat{=}\ \langle \mathit{zip}, \mathit{zip}^{\mathrm{c}} \rangle (x,y)\ \mathbf{in}\ ((a,b):s, t) \end{array}$$

元の関数と ALG-SC により導出された補関数とを組化すると，組化後の関数定義規則において全ての変数は必ず一回使用されるようになる．また，関数の出力が関数の入力になることがないという意味で，組化後も関数は treeless である．さらに，ALG-SC の定義から，全ての関数について，そのそれぞれの規則の右辺式の値域は互いに異なる．

$$\frac{e \Downarrow v}{\sigma(e) \Downarrow \sigma(v)}\textsc{Con}$$

$$\frac{\begin{array}{c}\exists f(\overrightarrow{p}) \mathrel{\hat{=}} \mathbf{let}\ \{(\overrightarrow{y_i}) \mathrel{\hat{=}} g_i(\overrightarrow{x_i})\}_{i\in\{1,\dots,n\}}\ \mathbf{in}\ (\overrightarrow{q}), \exists\theta.\ \overrightarrow{p}\theta = \overrightarrow{v} \\ g_i(\overrightarrow{x_i}\theta) \Downarrow \overrightarrow{w_i}, \quad \eta := \{x_{ij} \mapsto w_{ij} \mid (x_{i1},\dots,x_{in_i}) = \overrightarrow{x_i}, (w_{i1},\dots,w_{in_i}) = \overrightarrow{w_i}\} \\ \overrightarrow{u} = (\overrightarrow{q}\eta)\theta\end{array}}{f(\overrightarrow{v}) \Downarrow \overrightarrow{u}}\textsc{Fun}$$

図 5.6. **let** を含む Vdl の意味

もう少し形式的に述べると，組化後の関数を表現するために，我々は以下のように言語 Vdl に **let** を追加する．

$$\begin{aligned} rule ::= &\cdots \\ \mid\ & f(p_1,\dots,p_n) \mathrel{\hat{=}} \mathbf{let}\,(y_1,\dots,y_k) \mathrel{\hat{=}} g(x_1,\dots,x_m) \\ & \qquad\qquad\qquad\qquad \cdots \\ & \qquad\qquad\qquad (y'_1,\dots,y'_{k'}) \mathrel{\hat{=}} g'(x'_1,\dots,x'_{m'}) \\ & \qquad\qquad \mathbf{in}\ (e_1,\dots,e_m) \end{aligned}$$

簡便のため，上で $e_1,\dots,e_n$ は関数呼出を含まない，つまりパターンと同じ構造をしているとする．なお，**let** 束縛の左辺で二つ組以外の組が現れているのは，単射な関数やこの形式で記述される関数の逆関数も統一的にこの形式で扱うためである．拡張された Vdl の意味は，元の Vdl の意味の自然な拡張で与えられる．図 5.6 に拡張された Vdl の意味を示す．

組化の詳細なステップや正当性について議論することは我々の論文の目的ではない．参考までに，言語 Vdl で記述された順方向変換と Alg-Sc の導出する補関数に対する組化のアルゴリズムを以下に示す．

アルゴリズム 5.3 (組化).

入力： 順方向変換を記述プログラム P（**let** を含まない）.

出力： 順方向変換とその補関数を組化した関数を記述するプログラム $\mathcal{P}^{\triangle}$.

手続き：

1. 補関数を記述するプログラム $\mathcal{P}^{\mathrm{c}}$ を Alg-Sc により得る．.
2. 非単射な関数 g と，そのそれぞれの規則 r について次を繰り返す.
   (a) 規則 r^{c} を補関数における規則 r に対応する規則であるとする.
   (b) 規則 r と規則 r^{c} を以下の形式であるとする.

$$\begin{aligned} r &= g(\overrightarrow{p}) \mathrel{\hat{=}} \mathcal{C}[\overrightarrow{t}, \overrightarrow{u}, \overrightarrow{x}] \\ r^{\mathrm{c}} &= f^{\mathrm{c}}(\overrightarrow{p}) \mathrel{\hat{=}} \mathcal{C}'[\overrightarrow{t'}, \overrightarrow{x'}] \end{aligned}$$

ここで

- $\overrightarrow{t}$：非単射な関数呼出 $g_1(\overrightarrow{y_1}), \ldots, g_n(\overrightarrow{y_n})$,
- $\overrightarrow{t}'$：補関数の関数呼出 ${g_1}^{\mathrm{c}}(\overrightarrow{y_1}), \ldots, {g_n}^{\mathrm{c}}(\overrightarrow{y_n})$,
- $\overrightarrow{u}$：単射な関数呼出 $f_1(\overrightarrow{z_1}), \ldots, f_m(\overrightarrow{z_m})$.

である．

(c) 新規の変数 $\mathtt{t}_i, \mathtt{t}'_i, \mathtt{u}_j$（$i \in \{1, \ldots, n\}, j \in \{1, \ldots, m\}$）を準備する．

(d) 規則 $r^{\triangle}$ を以下の通りに構成する．

$$\begin{aligned}\langle r, r^{\mathrm{c}}\rangle = \langle g, g^{\mathrm{c}}\rangle(\overrightarrow{p}) \doteq \mathbf{let}\, &\{(\mathtt{t}_i, \mathtt{t}'_i) \doteq \langle g_i, {g_i}^{\mathrm{c}}\rangle(\overrightarrow{y_i})\}_{i\in\{1,\ldots,n\}} \\ &\{\mathtt{u}_j \qquad \doteq f_j(\overrightarrow{z_j})\}_{j\in\{1,\ldots,m\}} \\ \mathbf{in}\ &(\mathcal{C}[\overrightarrow{\mathtt{t}}, \overrightarrow{\mathtt{u}}, \overrightarrow{x}], \mathcal{C}'[\overrightarrow{\mathtt{t}'}, \overrightarrow{x'}])\end{aligned}$$

3. 単射な関数 f と，そのそれぞれの規則 r について，規則 r' を上と同様に以下のように構成する．

$$r' = f(\overrightarrow{p}) \doteq \mathbf{let}\, \{\mathtt{u}_j \doteq f_j(\overrightarrow{z_j})\}_{j\in\{1,\ldots,m\}}\, \mathbf{in}\, \mathcal{C}[\overrightarrow{\mathtt{u}}, \overrightarrow{x}]$$

4. 生成された規則 $r^{\triangle}$ と規則 r' を集め，プログラム $\mathcal{P}^{\triangle}$ を構成する． □

5.4.2 逆関数の導出：$\langle f, f^{\mathrm{c}}\rangle^{-1}$ の計算

前小節の議論より我々は $\langle f, f^{\mathrm{c}}\rangle$ を表現するプログラムを得た．次に，我々は $\langle f, f^{\mathrm{c}}\rangle^{-1}$ を導出する．ここでは，もっとも基本的な逆関数導出手法である，左右の入れ換えによる逆関数の導出を行う．

アルゴリズム 5.4 (組化後の逆関数の導出)**.**

入力： **let** の入ったプログラム $P = (\Sigma, Q, R)$.

出力： **let** の入ったプログラム $P^{-1} = \left(\Sigma, \left\{f^{-1} \mid f \in Q\right\}, R'\right)$.

手続き：プログラム中のそれぞれ規則 $r \in R$

$$\begin{aligned}r = f(\overrightarrow{p}) \doteq \mathbf{let}\, &\{\overrightarrow{y_i} \doteq f_i(\overrightarrow{x_i})\}_{i\in\{1,\ldots,n\}} \\ \mathbf{in}\ &(\overrightarrow{e})\end{aligned}$$

に対し，規則 $r' \in R'$ を以下の通りに構成する．

$$\begin{aligned}r^{-1} = f^{-1}(\overrightarrow{e}) \doteq \mathbf{let}\, &\{\overrightarrow{x_i} \doteq f_i^{-1}(\overrightarrow{y_i})\}_{i\in\{1,\ldots,n\}} \\ \mathbf{in}\ &(\overrightarrow{p})\end{aligned}$$

□

定理 5.6 (正しさ)**.** 上の逆関数導出は正しい．つまり，プログラム $\mathcal{P}$ から上の逆関数導出によりプログラム $\mathcal{P}^{-1}$ が求まったとすると，$\mathcal{P}$ 中の関数記号 f とそれに対応する $\mathcal{P}^{-1}$ 中の関数記号 f^{-1} について，

$$f(\overrightarrow{v}) \Downarrow \overrightarrow{u} \Leftrightarrow f^{-1}(\overrightarrow{u}) \Downarrow \overrightarrow{v}$$

となる．

証明. 対称性より，片方 $(\Rightarrow)$ のみを $\#\text{FunDepth}(f(\overrightarrow{v}))$ についての帰納法により示す．ここで，$\#\text{FunDepth}(e)$ は，式 e を評価するのに使用した Fun 規則の数を $e \Downarrow v$ の証明木に沿って縦に数えたもの最大値，つまり深さを表す．

なお，我々は証明において，左辺に出現する変数は右辺で必ず一回出現することのみを用い，f が単射であることを用いない．

基底：$\#\text{FunDepth}(f(\overrightarrow{v})) = 0$**.**

このとき，規則

$$f(\overrightarrow{p}) \doteq \overrightarrow{q}$$

で，代入 θ に対し $\overrightarrow{p}\theta = \overrightarrow{v}$ となるものが存在する．このとき，$f(\overrightarrow{p}\theta) \Downarrow \overrightarrow{q}\theta$ となる．また，規則

$$f^{-1}(\overrightarrow{q}) \doteq \overrightarrow{p}$$

が存在するため，$f^{-1}(\overrightarrow{q}\theta) \Downarrow \overrightarrow{p}\theta$ となる．

帰納：$\#\text{FunDepth}(f(\overrightarrow{v})) > 1$**.**

このとき，規則

$$\begin{aligned} f(\overrightarrow{p}) \doteq\ & \textbf{let}\ \{\overrightarrow{y_i} \doteq f_i(\overrightarrow{x_i})\}_{i\in\{1,\ldots,n\}} \\ & \textbf{in}\ (\overrightarrow{q}) \end{aligned}$$

で，代入 θ に対し $\overrightarrow{p}\theta = \overrightarrow{v}$ となるものが存在する．ここで，$f(\overrightarrow{p}\theta) \Downarrow \overrightarrow{u}$ より，各 $i \in \{1,\ldots,n\}$ について $\overrightarrow{w_i}$ を $f_i(\overrightarrow{x_i}\theta) \Downarrow \overrightarrow{w_i}$ とおき，代入 η を各 $i \in \{1,\ldots,n\}$ について $\overrightarrow{y_i}\eta = \overrightarrow{w_i}$ となるように定めると，$(\overrightarrow{q}\eta)\theta = \overrightarrow{u}$ となる．さて，このとき対応する規則

$$\begin{aligned} f^{-1}(\overrightarrow{q}) \doteq\ & \textbf{let}\ \{\overrightarrow{x_i} \doteq {f_i}^{-1}(\overrightarrow{y_i})\}_{i\in\{1,\ldots,n\}} \\ & \textbf{in}\ (\overrightarrow{p}) \end{aligned}$$

が存在する．このとき，$\overrightarrow{q}\zeta = \overrightarrow{u}$ となる $\zeta = \theta \circ \eta$ が存在する．各 $i \in \{1,\ldots,n\}$ について，$f_i(\overrightarrow{x_i}\theta) \Downarrow \overrightarrow{y_i}\zeta$ となることから，帰納法の仮定より，${f_i}^{-1}(\overrightarrow{y_i}\zeta) \Downarrow \overrightarrow{x_i}\theta$ となる．ここで，代入 ξ を，各 $i \in \{1,\ldots,n\}$ について $\overrightarrow{x_i}\xi = \overrightarrow{x_i}\theta$ となるように定めると，$\mathsf{vars}(\theta) \cup \mathsf{vars}(\eta) = \mathsf{vars}(\zeta) \cup \mathsf{vars}(\xi)$ より，

$$(\overrightarrow{p}\zeta)\xi = \overrightarrow{p}\theta = \overrightarrow{v}$$

となる．よって，

$$f(\overrightarrow{q}\zeta) \Downarrow (\overrightarrow{p}\zeta)\xi$$

であるため，

$$f(\overrightarrow{q}\zeta) \Downarrow \overrightarrow{v}$$

である．　□

関数記号 f の意味が関数であること，f が決定的に評価されることは異なることに注意する．実際，左右を入れ換えることにより得られる逆関数のプログラムは．決定的なものであるとは限らない．たとえば，単射な関数

$$\begin{aligned} &\mathit{snoc}([],b) &&\hat{=}\ b:[] \\ &\mathit{snoc}(a:x,b) &&\hat{=}\ \mathbf{let}\ t \mathrel{\hat{=}} \mathit{snoc}(x,b)\ \mathbf{in}\ a:t \end{aligned}$$

に対し，左右の入れ替えによって逆関数を作成すると以下を得る．

$$\begin{aligned} &\mathit{snoc}^{-1}(b:[]) &&\hat{=}\ ([],b) \\ &\mathit{snoc}^{-1}(a:t) &&\hat{=}\ \mathbf{let}\ (x,b) \mathrel{\hat{=}} \mathit{snoc}^{-1}(t)\ \mathbf{in}\ (a:x,b) \end{aligned}$$

このプログラムは，左辺のパターン $b:[]$ と $a:t$ に重なり，たとえば入力 $1:[]$ に対しどちらの規則を使用してよいかわからないため，非決定的である．この非決定性は，先読み（look-ahead）[Eng77] を用いることにより，避けることができる．

5.4.3　逆方向変換の構成

第 3 章の式 (RFL) に基づき，もし順方向変換 f が単射であれば，

$$f_{\mathrm{B}}(s,v) \mathrel{\hat{=}} f^{-1}(v)$$

により逆方向変換プログラム f_{B} を構成し，もし順方向変換 f が単射でなければ，

$$f_{\mathrm{B}}(s,v) \mathrel{\hat{=}} \langle f, f^{\mathrm{c}} \rangle^{-1}(v, f^{\mathrm{c}}(s))$$

により逆方向変換プログラム f_{B} を構成する．

定理 5.7 (振る舞いのよい逆方向変換の導出)**.** 上記により得られる逆方向変換は振る舞いがよい．

証明. 定理 3.3 より，第 3 章の式 (RFL) に基づき補関数から構成される逆方向変換は振る舞いはよい．よって，補関数導出の正しさおよび組化された関数の逆関数の導出の正しさが言えれば，主張を示すのに十分である．ところで，定理 5.5 により補関数の導出は正しく，定理 5.6 より逆関数の導出は正しい．よって，主張は真．　□

5.5　更新の反映可能性判定器の導出

更新の反映可能性判定器は，逆方向変換を実行することなしに，与えられた更新がソースに反映できるかどうかを判定する．順方向変換に対し，導出された逆方向変換は以下である．

$$f_{\mathrm{B}}(s,v) \mathrel{\hat{=}} \langle f, f^{\mathrm{c}} \rangle^{-1}(v, f^{\mathrm{c}}(s))$$

そのため，関数 f とその補関数 f^{c} について，最初のソース s が与えられれば，我々は，$(v, f^{\mathrm{c}}(s))$ が $\langle f, f^{\mathrm{c}}\rangle$ の値域に入っているかどうかを判定することで，$f(s)$ から v への更新が反映可能かどうか知ることができる．また，上の逆方向変換は補関数値不変の逆方向変換により定義されたものである．そのため，v への更新が反映可能かどうかは，更新反映を通して変わることがない．よって，更新の反映可能性判定器は，最初のソースについて一度導出してしまえば更新反映を通して使用できるため，実際に逆方向変換を実行するよりも効率が改善できることが期待される．

更新の反映可能性判定器を以下の正規木文法により定める．ただし，第4章に示した定義とは異なり，ここでは，

$$A \to \mathcal{C}[A_1, \ldots, A_n]$$

といった形式の生成規則も許す．この形の生成規則が入ることによって，文法の表現力が変化することはない．なぜならば，この形の規則は適切に非終端記号を導入することによって正規木文法の通常の規則に分解できるためである．

定義 5.1 (更新の反映可能性検査器)**.** プログラム $\mathcal{P} = (\Sigma, Q, _)$ に Alg-Sc を適用することでプログラム $\mathcal{P}^{\mathrm{c}}$ を得たとする．また，これらの上の順方向変換とその補関数について組化を適用し，プログラム $\mathcal{P}^{\triangle}$ を得たとする．順方向変換 f に対する補関数 f^{c} の値域の元 t_0 について，更新の反映可能性検査を行うための正規木文法 $G_{\mathrm{UV}} = (\Sigma, N', R')$ を以下で定める．

- $N' = \{Any\} \cup \{T_f \mid f \in Q\} \cup \{T^t_{\langle f, f^{\mathrm{c}}\rangle} \mid f \in Q, t$ は t_0 の部分木 $\}$
- R' は以下の規則のみを含む．
  - それぞれの $\mathsf{C} \in \Sigma$ に対し，

    $$Any \xrightarrow{*} \mathsf{C}(Any, \ldots, Any) \in R'$$

    である．
  - それぞれの単射な順方向変換の規則

    $$f(\overrightarrow{p}) \mathrel{\hat{=}} \mathcal{C}[f_1(\overrightarrow{x_1}), \ldots, f_n(\overrightarrow{x_n}), \overrightarrow{x}] \in R$$

    に対し，

    $$T_f \to \mathcal{C}[T_{f_1}, \ldots, T_{f_n}, \overrightarrow{Any}] \in R'$$

    である．
  - それぞれの組化された関数の規則

    $$\begin{aligned}\langle g, g^{\mathrm{c}}\rangle(\overrightarrow{p}) \mathrel{\hat{=}} \mathbf{let}\ & \{(t_i, t'_i) \mathrel{\hat{=}} \langle g_i, {g_i}^{\mathrm{c}}\rangle(\overrightarrow{y_i})\}_{i \in \{1,\ldots,n\}} \\ & \{u_j \qquad \mathrel{\hat{=}} f_j(\overrightarrow{z_j})\}_{j \in \{1,\ldots,m\}} \\ \mathbf{in}\ & (\mathcal{C}[\overrightarrow{t}, \overrightarrow{u}, \overrightarrow{x}], \mathcal{C}'[\overrightarrow{t'}, \overrightarrow{x'}])\end{aligned}$$

について，t_0 の部分木 t'' である代入 θ により $t'' = \mathcal{C}'[\overrightarrow{t'}, \overrightarrow{x'}]\theta$ を満たすものについて，

$$T_f^{t''} \to \mathcal{C}[\overrightarrow{T_{\langle g,g^c\rangle}^{t'\theta}}, \overrightarrow{T_f}, \overrightarrow{Any}] \in R'$$

である． □

上の定義において，T_f と Any は，5.3.2 節の T_f と Any と同じ意味を持つ．つまり，Any は Σ から構成される全ての木を表し，T_f は関数 f の値域を表す．また，$T_{\langle f,f^c\rangle}^t \xrightarrow{*} v$ であることは，ある s について $\langle f, f^c\rangle(s) = (v, t)$ となることを表している．

定理 5.8 (反映可能性検査の正しさ)**.** 定義 5.1 の正規木文法 G_{UV} において，$T_{\langle f,f^c\rangle}^{t_0} \xrightarrow{*} v$ であることと，$(v, t_0) \in \mathsf{ran}(\langle f, f^c\rangle)$ であることは同値である．

証明. 以下の主張を示す．

> t_0 の部分木 v' について，$T_{\langle f,f^c\rangle}^{v'} \xrightarrow{*} v$ であるならば，かつそのときに限り，$(v, v') \in \mathsf{ran}(\langle f, f^c\rangle)$ となる．

証明において，単射な関数 f に対し，$[\![T_f]\!]$ が関数 f の値域を示すことを仮定する．

「そのときに限り」. 今，$\langle f, f^c\rangle(\overrightarrow{v_0}) \Downarrow (v, v')$ であったとする．このとき，$T_{\langle f,f^c\rangle}^t \xrightarrow{*} v$ となることを $\#\textsc{FunDepth}(\langle f, f^c\rangle(\overrightarrow{v_0}))$ に関する帰納法により示す．

基底：$\#\textsc{FunDepth}(\langle f, f^c\rangle(\overrightarrow{v_0})) = 0$**.**

このとき，規則

$$\langle f, f^c\rangle(\overrightarrow{p}) \triangleq (\mathcal{C}[\overrightarrow{x}], \mathcal{C}'[\overrightarrow{x'}])$$

で，代入 θ に対し，$\overrightarrow{p}\theta = \overrightarrow{v_0}$ となるものが存在する．また，このとき，$v = \mathcal{C}[\overrightarrow{x}\theta]$ であり，$v' = \mathcal{C}'[\overrightarrow{x'}\theta]$ である．文法 G_{UV} の定義より，G_{UV} は以下の生成規則を含む．

$$T_{\langle f,f^c\rangle}^{v'} \to \mathcal{C}[Any, \ldots, Any]$$

これより，$v = \mathcal{C}[\overrightarrow{x}\theta]$ でありかつ全ての $k \in \{1, \ldots, |\overrightarrow{x}|\}$ に対し $Any \xrightarrow{*} x_k\theta$ であるため，

$$T_{\langle f,f^c\rangle}^{v'} \to \mathcal{C}[Any, \ldots, Any] \xrightarrow{*} v$$

が成り立つ．

帰納：$\#\textsc{FunDepth}(\langle f, f^{\mathrm{c}}\rangle(\overrightarrow{v_0})) > 0$**.**

このとき，規則

$$\begin{array}{lll}\langle f, f^{\mathrm{c}}\rangle(\overrightarrow{p}) \mathrel{\hat{=}} \mathbf{let}\,(t_i, t_i') & \mathrel{\hat{=}} \langle g, g^{\mathrm{c}}\rangle_i(\overrightarrow{y_i}) & i \in \{1, \ldots, n\} \\ \qquad u_j & \mathrel{\hat{=}} f_j(\overrightarrow{z_j}) & j \in \{1, \ldots, m\} \\ \quad \mathbf{in}\ (\mathcal{C}[\overrightarrow{t}, \overrightarrow{u}, \overrightarrow{x}], \mathcal{C}'[\overrightarrow{t'}, \overrightarrow{x'}]) & & \end{array}$$

で，代入 θ と η に対し，

$$\overrightarrow{p}\theta = \overrightarrow{v_0}$$
$$\langle g_i, {g_i}^{\mathrm{c}}\rangle(\overrightarrow{y_i}\theta) \Downarrow (w_i, w_i') \quad i \in \{1, \ldots, n\}$$
$$f_j(\overrightarrow{z_j}\theta) \Downarrow v_j \quad j \in \{1, \ldots, m\}$$
$$(\mathcal{C}[\overrightarrow{t}\eta, \overrightarrow{u}\eta, \overrightarrow{x}\theta], \mathcal{C}'[\overrightarrow{t'}\eta, \overrightarrow{x'}\theta]) = (v, v').$$

となるものが存在する．文法 G_{UV} の定義より，G_{UV} は以下の生成規則を含む．

$$T^{v'}_{\langle f, f^{\mathrm{c}}\rangle} \to \mathcal{C}[\overrightarrow{T^{t'\theta\eta}_{\langle g, g^{\mathrm{c}}\rangle}}, \overrightarrow{T_f}, \overrightarrow{Any}]$$

ここで，$T_{f_j} \xrightarrow{*} v_j = u_j\eta$（$j \in \{1, \ldots, m\}$）かつ $Any \xrightarrow{*} x_k\theta$（$k \in \{1, \ldots, |\overrightarrow{x}|\}$）である．よって，全ての $i \in \{1, \ldots, n\}$ に対し，$\langle g_i, {g_i}^{\mathrm{c}}\rangle(\overrightarrow{y_i}\theta) \Downarrow (w_i, w_i')$ が成り立つことから，帰納法の仮定により，$T^{w_i'}_{\langle g_i, {g_i}^{\mathrm{c}}\rangle} \xrightarrow{*} w_i = t_i\eta$ である．また，$\overrightarrow{t'}\theta\eta = \overrightarrow{t'}\eta = \overrightarrow{w'}$ が成り立つ．よって，

$$T^{v'}_{\langle f, f^{\mathrm{c}}\rangle} \to \mathcal{C}[\overrightarrow{T^{t'\theta\eta}_{\langle g, g^{\mathrm{c}}\rangle}} \xrightarrow{*} \mathcal{C}[\overrightarrow{t}\eta, \overrightarrow{u}\eta, \overrightarrow{x}\theta] = v$$

となる．

「ならば」．　今 $T^{v'}_{\langle f, f^{\mathrm{c}}\rangle} \xrightarrow{*} v$ であったとする．このとき，$\overrightarrow{v_0}$ が存在し $\langle f, f^{\mathrm{c}}\rangle(\overrightarrow{v_0}) \Downarrow (v, v')$ となることを，生成関係 $\xrightarrow{*}$ に関する帰納法により示す．

基底：$\xrightarrow{*} = \to$**.**

このとき，G_{UV} は生成に

$$T^{v'}_{\langle f, f^{\mathrm{c}}\rangle} \to \mathcal{C}[]$$

という生成規則を使用したことが言える．つまり，$v = \mathcal{C}[]$ である．このことから，上の生成規則に対応したプログラムの規則

$$\langle f, f^{\mathrm{c}}\rangle(\overrightarrow{p}) \mathrel{\hat{=}} (\mathcal{C}[], \mathcal{C}'[\overrightarrow{x'}])$$

と代入 θ

$$v' = \mathcal{C}'[\overrightarrow{x'}]\theta.$$

が存在する．よって，$\overrightarrow{v_0}$ を $\overrightarrow{v_0} = \overrightarrow{p}\theta$ として定めると，$\langle f, f^{\mathrm{c}}\rangle(\overrightarrow{v_0}) \Downarrow (v, v')$ となる．

帰納：$\xrightarrow{*} = \rightarrow \circ \xrightarrow{*}$.

このとき

$$T^{v'}_{\langle f, f^c \rangle} \rightarrow \mathcal{C}[\overrightarrow{T^{t'\theta}_{\langle g, g^c \rangle}}, \overrightarrow{T_f}, \overrightarrow{Any}] \xrightarrow{*} v$$

であったとする．このとき上の生成規則に対応するプログラムの規則

$$\begin{array}{l} \langle f, f^c \rangle(\overrightarrow{p}) \mathrel{\hat{=}} \mathbf{let}\,(t_i, t'_i) \mathrel{\hat{=}} \langle g, g^c \rangle_i(\overrightarrow{y_i}) \quad i \in \{1, \ldots, n\} \\ \qquad\qquad\qquad\quad u_j \quad \mathrel{\hat{=}} f_j(\overrightarrow{z_j}) \qquad\quad j \in \{1, \ldots, m\} \\ \qquad\qquad\quad \mathbf{in}\ (\mathcal{C}[\overrightarrow{t}, \overrightarrow{u}, \overrightarrow{x}], \mathcal{C}'[\overrightarrow{t'}, \overrightarrow{x'}]) \end{array}$$

と代入 θ

$$v' = \mathcal{C}'[\overrightarrow{t'}, \overrightarrow{x'}]\theta$$

が存在する．ここで，v は $\mathcal{C}[\overrightarrow{w}, \overrightarrow{v}, \overrightarrow{s}]$ という形式をしていて v' は $\mathcal{C}'[\overrightarrow{w'}, \overrightarrow{s'}]$ という形式をしている．ただし，$\overrightarrow{w'} = \overrightarrow{t'}\theta$, $\overrightarrow{s'} = \overrightarrow{x'}\theta$ であり，全ての $i \in \{1, \ldots, n\}$ に対し $T^{w'_i}_{\langle g_i, g_i{}^c \rangle} \xrightarrow{*} w_i$ であり，また，全ての $j \in \{1, \ldots, m\}$ に対し $T_{f_j} \xrightarrow{*} v_j$ である．このとき，任意の $j \in \{1, \ldots, m\}$ に対し，$T_{f_j} \xrightarrow{*} v_j$ より，代入 η_j が存在して $f_j(\overrightarrow{z_j}\sigma_j) \Downarrow v_j$ となることが言える．また，任意の $i \in \{1, \ldots, n\}$ に対し，帰納法の仮定より，$T^{w'_i}_{\langle g_i, g_i{}^c \rangle} \xrightarrow{*} w_i$ より，代入 σ_j が存在して $\langle g_i, g_i{}^c \rangle(\overrightarrow{y_i}\sigma_i) = (w_i, w'_i)$ となる．このとき，代入 τ を以下で定義する．

$$\tau(x) = \begin{cases} \eta_j(x) & \text{if } x \in \{\overrightarrow{z_j}\} \\ \sigma_i(x) & \text{if } x \in \{\overrightarrow{y_i}\} \\ \theta(x) & \text{if } x \in \{\overrightarrow{x'}\} \\ s_k & \text{if } x = x_k \in \{\overrightarrow{x}\} \end{cases}$$

この τ の定義は，well-defined である．なぜなら，VDL の **let** 束縛は affine であるためである．（実際には affine より強く linear）．よって $[\![\langle f, f^c \rangle]\!](\overrightarrow{p}\tau) = (\mathcal{C}[\overrightarrow{w}, \overrightarrow{v}, \overrightarrow{s}], \mathcal{C}'[\overrightarrow{w'}, \overrightarrow{s'}]) = (v, v')$ が成り立つ． □

いくつかの更新の反映可能性検査器の導出に対し，いくつかの例を示す．以下の例において，反映可能性を表現する正規木文法のうち，必要のない非終端記号は除いてある．

例 5.12. 本章冒頭（5.1 節）の *append* とその補関数 $append^c$ について考える．もし，最初のソースが

$$(s_1, s_2) = ([\mathsf{True}, \mathsf{False}], [])$$

であったとすると，ビューは $append(s_1, s_2) = [\mathsf{True}, \mathsf{False}]$ となる．また，このとき補関数の値は，$append^c(s_1, s_2) = \mathsf{B}_2(\mathsf{B}_2(\mathsf{B}_1))$ となる．

このとき，更新の反映可能性を表現する正規木文法は以下の生成規則を持つ．

$$
\begin{array}{ll}
\mathit{Any} \to \mathsf{True} & \mathit{Any} \to \mathsf{False} \\
\mathit{Any} \to [\,] & \mathit{Any} \to \mathit{Any} : \mathit{Any} \\
\multicolumn{2}{l}{T^{\mathsf{B}_2(\mathsf{B}_2(\mathsf{B}_1))}_{\langle \mathit{append}, \mathit{append}^{\mathrm{c}} \rangle} \to \mathit{Any} : T^{\mathsf{B}_2(\mathsf{B}_1)}_{\langle \mathit{append}, \mathit{append}^{\mathrm{c}} \rangle}} \\
\multicolumn{2}{l}{T^{\mathsf{B}_2(\mathsf{B}_1)}_{\langle \mathit{append}, \mathit{append}^{\mathrm{c}} \rangle} \to \mathit{Any} : T^{\mathsf{B}_1}_{\langle \mathit{append}, \mathit{append}^{\mathrm{c}} \rangle}} \\
\multicolumn{2}{l}{T^{\mathsf{B}_1}_{\langle \mathit{append}, \mathit{append}^{\mathrm{c}} \rangle} \to \mathit{Any}}
\end{array}
$$

ここで，直観的には，非終端記号 $T^{\mathsf{B}_2(\mathsf{B}_2(\mathsf{B}_1))}_{\langle \mathit{append}, \mathit{append}^{\mathrm{c}} \rangle}$ は長さ 2 以上のリストを生成する．よって，更新後のビューは必ず長さ 2 以上のリストでなければならない．

例 5.13. 以下に定義される関数 *filter* を考える．

$$
\begin{array}{ll}
\mathit{filter}([\,]) & \hat{=}\ [\,] \\
\mathit{filter}(\mathsf{A}_1 : x) & \hat{=}\ \mathsf{A}_1 : \mathit{filter}(x) \\
\mathit{filter}(\mathsf{A}_2 : x) & \hat{=}\ \mathsf{A}_2 : \mathit{filter}(x) \\
\mathit{filter}(\mathsf{A}_3 : x) & \hat{=}\ \mathit{filter}(x)
\end{array}
$$

関数 *filter* について，以下の補関数を考える．

$$
\begin{array}{ll}
\mathit{filter}^{\mathrm{c}}([\,]) & \hat{=}\ \mathsf{B}_1 \\
\mathit{filter}^{\mathrm{c}}(\mathsf{A}_1 : x) & \hat{=}\ \mathsf{B}_2(\mathit{filter}^{\mathrm{c}}(x)) \\
\mathit{filter}^{\mathrm{c}}(\mathsf{A}_2 : x) & \hat{=}\ \mathsf{B}_2(\mathit{filter}^{\mathrm{c}}(x)) \\
\mathit{filter}^{\mathrm{c}}(\mathsf{A}_3 : x) & \hat{=}\ \mathsf{B}_3(\mathit{filter}^{\mathrm{c}}(x))
\end{array}
$$

この補関数は Alg-Sc に適切な分割を与えることにより，導出することができる．最初のソースが

$$s = [\mathsf{A}_2, \mathsf{A}_3, \mathsf{A}_1]$$

であったとすると，ビューは $\mathit{filter}(s) = [\mathsf{A}_2, \mathsf{A}_1]$ となる．また，このとき補関数の値は $\mathit{filter}^{\mathrm{c}}(s) = \mathsf{B}_2(\mathsf{B}_3(\mathsf{B}_2(\mathsf{B}_1)))$ となる．

このとき，更新の反映可能性を表現する正規木文法は以下の生成規則を持つ．

$$
\begin{array}{lll}
T^{\mathsf{B}_2(\mathsf{B}_3(\mathsf{B}_2(\mathsf{B}_1)))}_{\langle \mathit{filter}, \mathit{filter}^{\mathrm{c}} \rangle} & \to t : T^{\mathsf{B}_3(\mathsf{B}_2(\mathsf{B}_1))}_{\langle \mathit{filter}, \mathit{filter}^{\mathrm{c}} \rangle} & t \in \{\mathsf{A}_1, \mathsf{A}_2\} \\
T^{\mathsf{B}_3(\mathsf{B}_2(\mathsf{B}_1))}_{\langle \mathit{filter}, \mathit{filter}^{\mathrm{c}} \rangle} & \to T^{\mathsf{B}_2(\mathsf{B}_1)}_{\langle \mathit{filter}, \mathit{filter}^{\mathrm{c}} \rangle} & \\
T^{\mathsf{B}_2(\mathsf{B}_1)}_{\langle \mathit{filter}, \mathit{filter}^{\mathrm{c}} \rangle} & \to t : T^{\mathsf{B}_1}_{\langle \mathit{filter}, \mathit{filter}^{\mathrm{c}} \rangle} & t \in \{\mathsf{A}_1, \mathsf{A}_2\} \\
T^{\mathsf{B}_1}_{\langle \mathit{filter}, \mathit{filter}^{\mathrm{c}} \rangle} & \to [\,] &
\end{array}
$$

ここで，直観的には，非終端記号 $T^{\mathsf{B}_2(\mathsf{B}_3(\mathsf{B}_2(\mathsf{B}_1)))}_{\langle \mathit{filter}, \mathit{filter}^{\mathrm{c}} \rangle}$ は長さが 2 のリストで要素が A_1 か A_2 のどちらかであるものを生成する．よって，ビュー上の更新はリストの長さを変えてはいけない．

5.6 例

本章の双方向化手法の全体像を確認するため，これまでの例よりも少し大きな例を示す．以下の順方向変換 $\mathit{students}$ は，学生と教授のリストから，学生のみを抽出する．

$$\begin{aligned}
&\mathit{students}([]) \mathrel{\hat=} [] \\
&\mathit{students}(\mathsf{Student}(\mathit{name}, \mathit{grade}, \mathit{major}) : r) \\
&\quad \mathrel{\hat=} \mathsf{Students}(\mathit{name}, \mathit{grade}, \mathit{major}) : \mathit{students}(r) \\
&\mathit{students}(\mathsf{Professor}(\mathit{name}, \mathit{position}, \mathit{major}) : r) \\
&\quad \mathrel{\hat=} \mathit{students}(r)
\end{aligned}$$

アルゴリズム ALG-SC により，$\mathit{students}$ に対し，我々は以下の補関数を導出する．

$$\begin{aligned}
&\mathit{students}^{\mathsf{c}}([]) \mathrel{\hat=} \mathsf{B}_1 \\
&\mathit{students}^{\mathsf{c}}(\mathsf{Student}(\mathit{name}, \mathit{grade}, \mathit{major}) : r) \\
&\quad \mathrel{\hat=} \mathsf{B}_2(\mathit{students}^{\mathsf{c}}(r)) \\
&\mathit{students}^{\mathsf{c}}(\mathsf{Professor}(\mathit{name}, \mathit{position}, \mathit{major}) : r) \\
&\quad \mathrel{\hat=} \mathsf{B}_3(\mathit{name}, \mathit{position}, \mathit{major}, \mathit{students}^{\mathsf{c}}(r))
\end{aligned}$$

規則を識別するため，構成子 $\mathsf{B}_1, \mathsf{B}_2, \mathsf{B}_3$ が導入された．また，$\mathit{students}$ の第三規則において変数 $\mathit{name}, \mathit{position}, \mathit{major}$ は右辺に出現しないため，補関数の右辺に出現する．関数 $\mathit{students}$ と補関数 $\mathit{students}^{\mathsf{c}}$ に対し，組化を適用することで以下を得る．

$$\begin{aligned}
&\langle \mathit{students}, \mathit{students}^{\mathsf{c}} \rangle([]) \mathrel{\hat=} ([], \mathsf{B}_1) \\
&\langle \mathit{students}, \mathit{students}^{\mathsf{c}} \rangle(\mathsf{Student}(\mathit{name}, \mathit{grade}, \mathit{major}) : r) \\
&\quad \mathrel{\hat=} \mathbf{let}\ (s, t) \mathrel{\hat=} \langle \mathit{students}, \mathit{students}^{\mathsf{c}} \rangle(r) \\
&\qquad \mathbf{in}\ (\mathsf{Students}(\mathit{name}, \mathit{grade}, \mathit{major}) : s, \mathsf{B}_2(t)) \\
&\langle \mathit{students}, \mathit{students}^{\mathsf{c}} \rangle(\mathsf{Professor}(\mathit{name}, \mathit{position}, \mathit{major}) : r) \\
&\quad \mathrel{\hat=} \mathbf{let}\ (s, t) \mathrel{\hat=} \langle \mathit{students}, \mathit{students}^{\mathsf{c}} \rangle(r) \\
&\qquad \mathbf{in}\ (s, \mathsf{B}_3(\mathit{name}, \mathit{position}, \mathit{major}, t))
\end{aligned}$$

左右を入れ換えることにより，関数 $\langle \mathit{students}, \mathit{students}^{\mathsf{c}} \rangle$ の逆関数は以下の通りに得られる．

$$\begin{aligned}
&\langle \mathit{students}, \mathit{students}^{\mathsf{c}} \rangle^{-1}([], \mathsf{B}_1) \mathrel{\hat=} [] \\
&\langle \mathit{students}, \mathit{students}^{\mathsf{c}} \rangle^{-1}(\mathsf{Students}(\mathit{name}, \mathit{grade}, \mathit{major}) : s, \mathsf{B}_2(t)) \\
&\quad \mathrel{\hat=} \mathbf{let}\ r \mathrel{\hat=} \langle \mathit{students}, \mathit{students}^{\mathsf{c}} \rangle^{-1}(s, t) \\
&\qquad \mathbf{in}\ \mathsf{Student}(\mathit{name}, \mathit{grade}, \mathit{major}) : r \\
&\langle \mathit{students}, \mathit{students}^{\mathsf{c}} \rangle^{-1}(s, \mathsf{B}_3(\mathit{name}, \mathit{position}, \mathit{major}, t)) \\
&\quad \mathrel{\hat=} \mathbf{let}\ r \mathrel{\hat=} \langle \mathit{students}, \mathit{students}^{\mathsf{c}} \rangle(s, t) \\
&\qquad \mathbf{in}\ \mathsf{Professor}(\mathit{name}, \mathit{position}, \mathit{major}) : r
\end{aligned}$$

これらより，以下の逆方向変換 $\mathsf{reflect}_{\mathit{students}, \mathit{students}^{\mathsf{c}}}$ を得られる．

$$\mathit{students}_{\mathrm{B}} \mathrel{\hat=} \langle \mathit{students}, \mathit{students}^{\mathsf{c}} \rangle^{-1}(v, \mathit{students}^{\mathsf{c}}(s))$$

この逆方向変換は以下の関数 ρ と等価な関数である．

$$\begin{aligned}
&\rho([],[]) \mathrel{\hat=} [] \\
&\rho(\mathsf{Student}(name, grade, major) : r, \mathsf{Student}(name', grade', major') : r') \\
&\qquad \mathrel{\hat=} \mathsf{Student}(name', grade', major') : \rho(r, r') \\
&\rho(\mathsf{Professor}(name, position, major) : r, r') \\
&\qquad \mathrel{\hat=} \mathsf{Professor}(name, position, major) : \rho(r, r')
\end{aligned}$$

よって，逆方向変換 $students_{\mathrm{B}}$ により，ビュー上で行った学生の名前の更新や専攻の更新をソースへと反映することができる．

最初のソースが以下であった場合を考える．

$$s = [\mathsf{Students}(\mathsf{David}, \mathsf{M2}, \mathsf{ComputerScience}), \mathsf{Professor}(\mathsf{Emil}, \mathsf{Professor}, \mathsf{Mathematics})]$$

このとき，ビューは

$$students(s) = [\mathsf{Students}(\mathsf{David}, \mathsf{M2}, \mathsf{ComputerScience})]$$

となる．もし，ビューを

$$v' = [\mathsf{Students}(\mathsf{David}, \mathsf{M2}, \mathsf{Mathematics})]$$

と更新した場合，逆方向変換 $students_{\mathrm{B}}$ により反映され，ソースは,

$$\begin{aligned}
&students_{\mathrm{B}}(s, v') \\
&\quad = [\mathsf{Students}(\mathsf{David}, \mathsf{M2}, \mathsf{Mathematics}), \mathsf{Professor}(\mathsf{Emil}, \mathsf{Professor}, \mathsf{Mathematics})]
\end{aligned}$$

となる．ところが，ビューに対し，ビューを $[]$ へ更新するなど，挿入や削除を行うことはできない．

逆方向変換 $students_{\mathrm{B}}$ により反映可能なビューの変更先は，正規木文法で表現される．ビュー v_1, v_2, v_3 を

$$\begin{aligned}
v_1 &= \mathsf{B}_2(\mathsf{B}_3(\mathsf{Emil}, \mathsf{Professor}, \mathsf{Mathematics}, \mathsf{B}_1)) \\
v_2 &= \mathsf{B}_3(\mathsf{Emil}, \mathsf{Professor}, \mathsf{Mathematics}, \mathsf{B}_1) \\
v_3 &= \mathsf{B}_1,
\end{aligned}$$

と置く．このとき，正規木文法

$$\begin{aligned}
&T^{v_1}_{\langle students, students^{\mathrm{c}}\rangle} \to \mathsf{Students}(Any, Any, Any) : T^{v_2}_{\langle students, students^{\mathrm{c}}\rangle} \\
&T^{v_2}_{\langle students, students^{\mathrm{c}}\rangle} \to T^{v_3}_{\langle students, students^{\mathrm{c}}\rangle} \\
&T^{v_3}_{\langle students, students^{\mathrm{c}}\rangle} \to []
\end{aligned}$$

において，$[\![T^{v_1}_{\langle students, students^{\mathrm{c}}\rangle}]\!]$ が，$students_{\mathrm{B}}$ により反映可能なビューの更新先を表す．たとえば，$T^{v_1}_{\langle students, students^{\mathrm{c}}\rangle} \xrightarrow{*} v'$ であるが，$T^{v_1}_{\langle students, students^{\mathrm{c}}\rangle} \stackrel{*}{\not\to} []$ である．

5.7 まとめ

本章では，affine かつ treeless である一階の関数型言語で記述された順方向変換プログラムに対し，プログラムの双方向化，つまり逆方向変換プログラム導出手法を提案した．提案双方向化手法は，補関数 [BS81] の自動生成に基づくものであるため，導出された逆方向変換は振る舞いがよいことが保証される．また，それに加え本章では，順方向変換プログラムの単射性解析を用いることで，縮約順序おいてよりよい補関数を求めることができることを示した．さらに，我々は，更新の反映可能性検査器の導出手法も与えた．反映可能性検査器は正規木文法で与えられるが，正規木文法については様々なよい性質 [CDG+97] が知られている．そのため，逆方向変換を実際に実行するよりも，効率よく反映可能な更新を判定できることが期待される．

5.8 問題点

本章の提案する双方向化の考え方自体は，たとえば，関数呼出のネスト

$$h(x) = f(g(x))$$

に対し，補関数を

$$h^{\mathrm{c}}(x) = (f^{\mathrm{c}}(g(x)), g^{\mathrm{c}}(x))$$

と定める[3]などのすることにより，より広いクラスのプログラムに適用することができる．たとえば，この考え方により，挿入ソート

$$\begin{array}{l} \mathit{ins}(a, b:x) = \begin{cases} a:b:x & \text{if } a < b \\ b:\mathit{ins}(a,x) & \text{if } a \geq b \end{cases} \\ \mathit{sort}([]) \quad\;\; = [] \\ \mathit{sort}(a:x) = \mathit{ins}(a, \mathit{sort}(x)) \end{array}$$

に対し，以下の補関数を得ることができる．

$$\begin{array}{l} \mathit{ins}^{\mathrm{c}}(a, b:x) = \begin{cases} \mathsf{B}_1 & \text{if } a < b \\ \mathsf{B}_2(\mathit{ins}(a,x)) & \text{if } a \geq b \end{cases} \\ \mathit{sort}^{\mathrm{c}}([]) \quad\;\; = [] \\ \mathit{sort}^{\mathrm{c}}(a:x) = \mathit{ins}^{\mathrm{c}}(a, \mathit{sort}(x)) : \mathit{sort}^{\mathrm{c}}(x) \end{array}$$

この補関数により得られる $\mathit{sort}_{\mathrm{B}}$ は以下の挙動をする．

$$\begin{array}{ll} \mathit{sort}_{\mathrm{B}}([1,3,2,4],[11,12,13,14]) & = [11,13,12,14] \\ \mathit{sort}_{\mathrm{B}}([1,3,2,4],[11,12,13]) & = \bot \\ \mathit{sort}_{\mathrm{B}}([1,3,2,4],[11,12,13,14,15]) & = \bot \end{array}$$

[3]結合子により双方向構成 [FGM+05, HMT04] における，双方向変換の合成と等価である．

ただし，単純に考え方を適用するだけでは，効果的な逆方向変換が得られない場合がある．たとえば，関数

$$h(x) = (fst(x), snd(x))$$

は単射であるが，

$$h(x) = (fst \times snd)(dup(x))$$

と見て補関数を導出しても

$$h^{\mathrm{c}}(x) = (\mathrm{const.}, (snd \times fst)(dup(x)))$$

となり，あまり小さなものが得られない．また，たとえば，関数

$$h(x) = append(x, [])$$

に対しても同様にあまり小さな補関数が得られない．

上で小さな補関数が得られない理由は，

$$h(x) = f(g(x))$$

のような関数に対する補関数

$$h^{\mathrm{c}}(x) = (f^{\mathrm{c}}(g(x)), g^{\mathrm{c}}(x))$$

の導出において，f^{c} を定める際に f が g の値域についてどのように振る舞うかを考慮していないためである．たとえば，$(fst \times snd)$ は dup の値域では単射であるし，$append$ も第二引数が [] と固定長の場合には単射である．

以降の章の議論は，上の問題を，生垣の変換に限り部分的に解決するものである．生垣の変換では，変換結果を連接することが多いため，第 6 章で定める言語は $append$ を基本的な言語要素として含んでいる．これにより，$append$ の引数の式の値域を利用して，$append$ が単射である場合を健全に判定することができる（第 8 章）．また，第 6 章は，dup のような関数を直接表現するのではなく，組化後の関数の形式を利用することにより間接的に表現する．これにより，第 6 章の言語で記述されたプログラムは **let** を含むが，第 8 章では，本章の議論はそのようなプログラムに対しても効果的に適用できることを示す．さらに，我々は第 7 章において，上のような h において，g の値域が正規生垣言語で記述される際には，f の，g の値域に特化した定義を求めることができることを示す．

第6章　XML上の順方向変換記述言語

第5章では，affine かつ treeless な一階の関数型言語 VDL で記述されたプログラムの双方向化について議論した．本章では，言語を XML などの生垣の上の変換を記述できるように拡張する．

本章では，どのようにして言語 VDL を拡張したかを述べ，続く第7章では効果的な双方向変換を導出するための前処理手法について述べ，本章の言語の上の双方向化については，第8章で述べる．

本章および第8章において，複製の取り扱いについては文献 [松田 09] にて発表される予定である．

6.1　Treeless 制約の緩和

6.1.1　緩和の要請

XML データは，もちろん構成子からなる木の上でリストを用いて表現することもできる．しかし，そのように XML データを表現してしまうと，affine かつ treeless な言語では記述できる変換は非常に制限されるものになってしまう．たとえば，以下の論文様の構造の XML

```
<chapter>
    <title>はじめに</title>
    <p>あるデータを変換により，別のデータ...</p>
    <p>双方向変換を例を...</p>
</chapter>
<chapter>
    <title>関連研究</title>
    <p>本章では，本論文に関連の深い研究...</p>
    <section>
        <title>双方向変換</title>
    ...
    </section>
</chapter>...
```

を次の XHTML 断片に変換することを考える．

```
<h1>はじめに</h1>
    <p>あるデータを変換により，別のデータ...</p>
    <p>双方向変換を例を...</p>
<h1>関連研究</h1>
    <p>本章では，本論文に関連の深い研究...</p>
    <h2>双方向変換</h2>
...
```

ここで，ソースは章（<chapter>）の列であり，各章は章題（<title>），段落の列（<p>）と節の列（<section>）をこの順で含む．また，各節は節題（<title>）とそれに続く段落の列（<p>）を含む．

この変換は，上記入力を構成子により木により表現した場合には，affine かつ treeless 言語で記述するのは難しくなる．たとえば，章が章題，段落のリスト，節のリスト，を子に持つ三引数構成子 Chapter で表現されているとすると，上の変換を達成するには関数定義規則

$$f(\mathsf{Chapter}(title, paragraphs, secs) : chapters) = \ldots$$

の右辺において

- 段落の列の変換結果
- 節の列の変換結果
- 再帰的な，章の列の変換結果

を一つの列にまとめなければばならない．ところが，treeless 制約のもとではリストの連接 *append* を変換結果に適用することはできない．また，ここで章が章題，段落のリスト，節のリスト，を子に持つ三引数構成子 Chapter で表現されているとしたが，元々ある XML 列をどのように構成子に写像すればよいのかは自明ではない．

6.1.2　我々のアプローチ

我々は，XML 列を構成子による木により扱うのではなく，第 4 章で定めた生垣として扱う．構成子による木の場合とは異なり，生垣において生垣同士の連接は生垣の構成要素であり treeless 言語における「関数」ではない．そのため，生垣の変換においては，上の列に対する三つの変換結果を連接することができる．また，パターンを拡張し正規生垣文法で記述される型情報を用いることで適切に連接演算子を用いて生垣を分割することができる．たと

えば，上に挙げた変換は以下の関数 $c2x$ により表現できる．

$$
\begin{aligned}
&\textbf{data}\ C \mathrel{\hat{=}} (\texttt{<chapter>}(\texttt{<title>}(String)\,.\,P\,.\,S))^* \\
&\textbf{data}\ S \mathrel{\hat{=}} (\texttt{<section>}(\texttt{<title>}(String)\,.\,P))^* \\
&\textbf{data}\ P \mathrel{\hat{=}} (\texttt{<p>}(String))^* \\
&c2x(\varepsilon) \mathrel{\hat{=}} \varepsilon \\
&c2x(\texttt{<chapter>}(\texttt{<title>}(t :: String)\,.\,p :: P\,.\,s :: S)\,.\,r :: C) \\
&\quad \mathrel{\hat{=}} \texttt{<h1>}(t)\,.\,p\,.\,s2x(s)\,.\,c2x(r) \\
&s2x(\varepsilon) \mathrel{\hat{=}} \varepsilon \\
&s2x(\texttt{<section>}(\texttt{<title>}(t :: String)\,.\,p :: P)\,.\,r :: S) \\
&\quad \mathrel{\hat{=}} \texttt{<h2>}(t)\,.\,p\,.\,s2x(r)
\end{aligned}
$$

上の $c2x$ において，演算子「.」は前述の二つの意味で使用されている．右辺に出現する「.」は変換の $c2x$ の結果や $s2x$ の結果を連接により合成するために使用され，左辺に出現する「.」は生垣を複数の部分生垣に分割するのに使用されている．

6.2　Affine 制約の緩和

まず，affine 制約を緩和するにあたって，どのような点が問題になり，素朴に解決しようとするとどのような問題が出るかのについて述べる．以下の，言語 VDL で記述される関数 f と g により定義される，関数 h を考える．

$$h(x) = (f(x), g(x))$$

関数 h は，VDL で記述することができない．なぜなら，上の規則において変数 x が右辺で二度出現しているため，affine 制約に反するためである．ここで，h について，f と g の単射に着目すると，以下の三つの場合がある．

- 関数 f と g がともに単射である．
- 関数 f と g のどちらかが単射である．
- 関数 f と g のどちらもが単射でない．

この一つ目と二つ目の場合については，問題は生じない．一つ目と二つ目の場合には，振る舞いがよく最も効果的な h の逆方向変換を以下のように求められるためである（ここでは，f が単射であったとする）．

$$h_{\mathrm{B}}(s, (v_1, v_2)) = f^{-1}(v_1) \quad \text{if } g(f^{-1}(v_1)) = v_2$$

しかし，三つめの場合について，h の振る舞いがよく効果的な逆方向変換を得るのは難しくなる．これを説明するのに，以下の二つの例を用いる．

$$\begin{aligned}
&\mathit{unzip}(x) = (\mathit{mapFst}(x), \mathit{mapSnd}(x))\\
&\qquad \text{where } \begin{array}{ll}\mathit{mapFst}([]) &= []\\
\mathit{mapFst}(\mathsf{Pair}(a,b):r) &= a : \mathit{mapFst}(x)\\
\mathit{mapSnd}([]) &= []\\
\mathit{mapSnd}(\mathsf{Pair}(a,b):r) &= b : \mathit{mapSnd}(x)\end{array}
\end{aligned}$$

$$\begin{aligned}
&\mathit{divs}(x) = (\mathit{div}_2(x), \mathit{div}_3(x))\\
&\qquad \text{where } \begin{array}{l}\mathit{div}_2(x) = \lfloor x/2 \rfloor\\
\mathit{div}_3(x) = \lfloor x/3 \rfloor\\
\mathit{div}_{2\mathrm{B}}(x,v) = 2 \times v + (x \bmod 2)\\
\mathit{div}_{3\mathrm{B}}(x,v) = 3 \times v + (y \bmod 3)\end{array}
\end{aligned}$$

関数 unzip は綴じ合わされた組のリストを，リストの組へと分解する関数である．たとえば,

$$\mathit{unzip}([\mathsf{Pair}(1,2), \mathsf{Pair}(3,4)]) = ([1,3],[2,4])$$

である．関数 div は，入力となる整数を 2 で割った商と 3 で割った商とを組にしたものである．ただし，割り切れない場合は余りを切り捨てる．たとえば,

$$\mathit{divs}(13) = (6,4)$$

である．

6.2.1　悲観的な解決案

変数の複数回出現がある場合のもっとも簡単な取り扱いは，変数のそれぞれ出現を別の変数として扱い逆方向変換を求めた後，その逆方向変換の結果に対し同じ変数に対応する値が等しいかどうか検査する手法である．すなわち， $\mathit{dup}(x) = (x,x)$ により，h を

$$h = (f \times g) \circ \mathit{dup}$$

と書き換え，$(f \times g)$ と dup を独立に双方向化する手法である．しかし，この解決案はあまり効果的な逆方向変換を与えない．たとえば，この解決案に沿って得られる unzip の逆方向変換は以下である．

$$\mathit{unzip}_{\mathrm{B}}(x,(v_1,v_2)) = \begin{cases} \mathit{mapFst}_{\mathrm{B}}(x,v_1) & \text{if } \mathit{mapFst}_{\mathrm{B}}(x,v_1) = \mathit{mapSnd}_{\mathrm{B}}(x,v_2)\\ \bot & \text{otherwise}\end{cases}$$

しかし，$\mathit{mapFst}_{\mathrm{B}}$ と $\mathit{mapSnd}_{\mathrm{B}}$ の自然な定義の下で，$\mathit{mapFst}_{\mathrm{B}}(x,v_1)$ と $\mathit{mapSnd}_{\mathrm{B}}(x,v_2)$ が等しくなるのは，$(v_1,v_2) = \mathit{unzip}(x)$ となるときのみである．たとえば，以下の $\mathit{mapFst}_{\mathrm{B}}$ と $\mathit{mapSnd}_{\mathrm{B}}$ の動作は自然なものである．

$$\mathit{mapFst}_{\mathrm{B}}([(2,3)],[x]) = [(x,3)] \qquad \mathit{mapSnd}_{\mathrm{B}}([(2,3)],[y]) = [(2,y)]$$

しかし，$\mathit{mapFst}_{\mathrm{B}}([(2,3)],[x]) = \mathit{mapSnd}_{\mathrm{B}}([(2,3)],[y])$ ならば，$x = 2, y = 3$ となる，つまり，$([x],[y]) = \mathit{uznip}([(2,3)])$ である．

商データ（quotient）を用いると，変数の複数回出現に対し，ある一つの変数出現以外に対する更新を無視することができる [FPP08]．これは，*unzip* の場合は，たとえばリストの各要素の第一要素への更新のみを反映することに相当する．これは，*unzip* の効果的な逆方向変換として我々が望むものではない．

6.2.2 楽観的な解決案

Hu らや Mu らは [HMT04, MHT04a]，変数のそれぞれ出現を別の変数として扱い逆方向変換を導出した後，同じの変数に対応する反映結果を併合するアプローチを取った．すなわち，

$$h = (f \times g) \circ \mathit{dup}$$

について，$(f \times g)$ と *dup* を独立に双方向化する際，*dup* の逆方向の変換が柔軟に更新の併合を行う．たとえば，Hu らの手法 [HMT04] に基づく *unzip* の逆方向の変換は以下である．

$$\mathit{unzip}_{\mathrm{B}}(s,(v_1,v_2)) = \begin{cases} \mathit{mapFst}_{\mathrm{B}}(s,v_1) & \text{if } \mathit{mapFst}_{\mathrm{B}}(s,v_1) = \mathit{mapSnd}_{\mathrm{B}}(s,v_2) \\ \mathit{mapSnd}_{\mathrm{B}}(s,v_2) & \text{if } \mathit{mapFst}_{\mathrm{B}}(s,v_1) = x \\ \mathit{mapFst}_{\mathrm{B}}(s,v_1) & \text{if } \mathit{mapSnd}_{\mathrm{B}}(s,v_2) = x \\ \bot & \text{otherwise} \end{cases}$$

ここで，我々は「逆方向変換」ではなく「逆方向の変換」という言葉を用いたのは，$\mathit{unzip}_{\mathrm{B}}$ が我々の言葉でいう逆方向変換ではないためである．この逆方向の変換の下では，*unzip* の各リストの第一要素か第二要素のどちらかに対する更新は反映できるものの，どちらもが同時に更新されている場合は反映できない．Mu らは，更新を値にタグ付けし明示的に表現することにより，もっと柔軟な更新反映を達成している．Mu らの手法 [MHT04a, HMT08] に基づく *unzip* の逆方向の変換は以下である．

$$\mathit{unzip}_{\mathrm{B}}(s,(v_1,v_2)) = \mathit{mergeUpdate}(\mathit{mapFst}_{\mathrm{B}}(s,v_1), \mathit{mapSnd}_{\mathrm{B}}(s,v_2))$$

ここで *mergeUpdate* を適切に定めることにより，上の逆方向の変換 $\mathit{unzip}_{\mathrm{B}}$ は *unzip* の値域については振る舞いがよいものとなる．つまり，$\mathit{unzip}_{\mathrm{B}}$ の定義を限定することにより，振る舞いのよい逆方向変換になる．また，Mu らは実際に *mergeUpdate* の一つの適切な定め方を与えている．

しかし，*divs* に対しは，どちらの手法を用いた場合も，振る舞いのよい逆方向の変換が求まらない．Hu らおよび Mu らの手法に従って得られる *divs* の逆方向の変換は以下である．

$$\mathit{divs}_{\mathrm{B}}(s,(v_1,v_2)) = \begin{cases} \mathit{div}_{2\mathrm{B}}(s,v_1) & \text{if } \mathit{div}_{2\mathrm{B}}(s,v_1) = \mathit{div}_{3\mathrm{B}}(s,v_2) \\ \mathit{div}_{3\mathrm{B}}(s,v_2) & \text{if } \mathit{div}_{2\mathrm{B}}(s,v_1) = x \\ \mathit{div}_{2\mathrm{B}}(s,v_1) & \text{if } \mathit{div}_{3\mathrm{B}}(s,v_2) = x \\ \bot & \text{otherwise} \end{cases}$$

逆方向の変換 $divs_{\rm B}$ の問題点について説明する．ソースが今 12 であったとする．このとき，更新 $(6,4) \rightarrowtail (6,5)$ の後に更新 $(7,5) \rightarrowtail (6,5)$ を適用することを考える．最初の更新の後，

$$divs_{\rm B}(12,(6,5)) = 15$$

であるため，ソースは 15 に更新され，ビューは $(7,5)$ になる．そして，二つ目の更新の後

$$divs_{\rm B}(15,(6,5)) = 13$$

であるため，ソースは 13 に更新され，ビューは $(6,4)$ となる．この結果は問題である．なぜなら，更新の反映を通してソースが 12 から 13 に変更されているにも拘わらず，その変換がビューから観測されないためである．これは,「副作用」，つまりビューの構築に関係ない部分の変更であるため，我々の振る舞いのよさの要求に反する．

6.2.3　我々のアプローチ

既存の研究 [FGM+05, HMT04, MHT04a] は，変数の複数回出現とそれらの変数を走査する関数を分けて取り扱った．つまり，彼らは，$h(x) = (f(x), g(x))$ と定義された関数 h に対して，$h_{\rm B}$ を $f_{\rm B}$ と $g_{\rm B}$ とを組み合わせることで定めた．しかし，このアプローチでは，順方向変換プログラムにおいて変数の複数回出現があった場合に，効果的かつ振る舞いのよい逆方向変換プログラムを定めるのは難しい．それに対し，我々は $h(x) = (f(x), g(x))$ に対し，$h_{\rm B}$ を $f_{\rm B}$ と $g_{\rm B}$ から求めることをしない．我々は，h を変数の複数回出現のない形 h' に書き直して，その上で h' の逆方向変換 $h'_{\rm B}$ を求める．たとえば，以下は，前述の $unzip$ を複数回出現がない形で書いたものである．

$$\begin{array}{lcl} unzip([\,]) & = & ([\,],[\,]) \\ unzip(\mathsf{Pair}(a,b):r) & = & (a:r_1, b:r_2) \text{ where } (r_1,r_2) = unzip(r), \end{array}$$

この変数の複数回出現を含まない $unzip$ の定義は，組化 [HITT97, Chi93] を適用することで得られる．組化は，変数の複数回出現を効果的に削除するが，常に成功するとは限らない．また，成功したとしても議論しやすいプログラムになるとも限らない．そのため，我々は組化を直接導入するのではなく，組化後の，値の組を返す関数（多返値関数 [IHM08]）を考える．

具体的には，我々は第 5 章で組化後のプログラムを表現したのと同様に，**let** を用いる．言語要素 **let** を導入することにより，我々は，組化後の関数を表現できるだけではなく，関数の入力と出力を分離することができる．関数の出力を束縛する変数の複数回使用は問題がないことに注意する．なぜなら，treeless な言語において，それらは再び走査されることなく，順方向変換の出力にそのまま現れるだけであるためである．

$$
\begin{array}{lll}
\mathit{program} & ::= \mathit{prod}_1 \ldots \mathit{prod}_m\ \mathit{rule}_1 \ldots \mathit{rule}_n & \text{(プログラム)} \\
\mathit{prod} & ::= \mathbf{data}\ T \mathrel{\hat{=}} t & \text{(型定義規則)} \\
t & ::= \varepsilon \mid \sigma(t) \mid t_1 \centerdot t_2 \mid T & \text{(型)} \\
\mathit{rule} & ::= f(\overrightarrow{p}) \mathrel{\hat{=}} e & \text{(関数定義規則)} \\
e & ::= \mathbf{let}\ \mathit{bind}_1 \ldots \mathit{bind}_n\ \mathbf{in}\ (\overrightarrow{q}) & \text{(\textbf{let} 式)} \\
p & ::= \varepsilon \mid \sigma(p) \mid p_1 \centerdot p_2 \mid x :: T & \text{(パターン)} \\
q & ::= \varepsilon \mid \sigma(q) \mid q_1 \centerdot q_2 \mid y \mid x & \text{(構成子式)} \\
\mathit{bind} & ::= (\overrightarrow{y}) \mathrel{\hat{=}} f(\overrightarrow{x}) & \text{(\textbf{let} 束縛)}
\end{array}
$$

ただし, $x, \overrightarrow{x}$ と $y, \overrightarrow{y}$ は変数, f は関数名, T は型名, σ はラベルである.

図 6.1. 言語 VDL+ の構文

6.3　順方向変換記述言語 VDL+

これまでの議論から，我々は，連接演算子「▪」と多返値関数のための **let** を加えることにより，言語 VDL を拡張する．本論文では，この VDL を拡張した言語を VDL+ と呼ぶ．言語 VDL で記述された関数の入力と出力は木であるに対し，言語 VDL+ で記述された関数の入力と出力は生垣である.

6.3.1　構文と意味

言語 VDL+ の構文を図 6.1 に示す．言語 VDL+ の式は三つ組 $\mathcal{P} = (G, Q, R)$ である．ここで，G は正規生垣文法であり，プログラム中のパターンに出現する型を記述する．また，Q は関数名の集合であり，R は関数定義規則である.

プログラム $\mathcal{P} = (G, Q, R)$ において G は型定義規則

$$\mathbf{data}\ T \mathrel{\hat{=}} t$$

の集まりにより記述される．これは，型定義規則 $\mathbf{data}\ T \mathrel{\hat{=}} t$ は，G 中の生成規則 $T \to t$ に対応する．簡便のため，プログラムの例を示す場合においては，正規表現や，正規生垣文法の枠を越える文法を用いる場合がある．たとえば，以下の型定義においては，正規表現の一つである Kleene 閉包 _* が使用されていて，型名がラベルの直下や直右以外の位置に出現している.

$$
\begin{array}{l}
\mathbf{data}\ C \mathrel{\hat{=}} (\texttt{<chapter>}(\texttt{<title>}(\mathit{String}) \centerdot P \centerdot S))^* \\
\mathbf{data}\ P \mathrel{\hat{=}} (\texttt{<p>}(\mathit{String}))^* \\
\mathbf{data}\ S \mathrel{\hat{=}} (\texttt{<section>}(\texttt{<title>}(\mathit{String}) \centerdot P))^*
\end{array}
$$

以降，集合の表現としても正規表現を用いる場合がある.

関数を定義する規則は以下の形式をしている．

$$\begin{aligned} f(\overrightarrow{p}) \doteq \mathbf{let}\ (\overrightarrow{y_1}) &\doteq g_i(\overrightarrow{x_1}) \\ &\vdots \\ (\overrightarrow{y_n}) &\doteq g_i(\overrightarrow{x_n}) \\ \mathbf{in}\ (\overrightarrow{q}) & \end{aligned}$$

ここで，$\overrightarrow{p}$ はパターンであり，$\overrightarrow{q}$ はラベル適用と変数使用しか含まない式である．また，我々は規則について以下の二つの制約を課す．

- パターンに現れる変数については複数回使用を禁止する．すなわち，変数 $\overrightarrow{x_1}, \ldots, \overrightarrow{x_n}$ は互いに異なる．
- 関数の出力が関数の入力になることはない．すなわち，変数の集合 $\overrightarrow{x_1}, \ldots, \overrightarrow{x_n}$ と変数の集合 $\overrightarrow{y_1}, \ldots, \overrightarrow{y_n}$ は互いに素である．

つまり，パターンに現れる変数は，以下のいずれかである．

- 規則の右辺（$\overrightarrow{q}$）にも，**let** 束縛の右辺にも現れない．
- 規則の右辺（$\overrightarrow{q}$）で一回現れるものの，**let** 束縛の右辺には現れない．
- 規則の右辺（$\overrightarrow{q}$）には現れないが，**let** 束縛の右辺で一回現れる．

また，**let** 束縛の左辺に現れる変数は，規則の右辺に 0 回以上現れるが，**let** 束縛の右辺には現れない．簡便のため，変数パターンにおいて，型名だけではなく，x :: <a>* のように型を使用することがある．また，全ての Σ 上の生垣を意味する型 Any，すなわち $[\![Any]\!]_G = \mathcal{H}_\Sigma$ となる非終端記号 Any について，変数パターン $x :: Any$ を単に x と書く．

第 4 章，第 5 章と同様に，言語 VDL$^+$ においても文脈の記法を用いる．文脈 $\overrightarrow{\mathcal{C}}$ は，特殊な変数 $\Box_1, \ldots, \Box_n$ を含む，パターン／式の列であり，$\overrightarrow{\mathcal{C}}$ 中のそれぞれの変数 $\Box_i$ を生垣 h_i で置き換えることにより得られるパターン／式の列を $\overrightarrow{\mathcal{C}}(h_1, \ldots, h_n)$ と書く．木の場合と異なり，$\overrightarrow{\mathcal{C}}[h_1, h_2, \ldots, h_n] = \overrightarrow{h}$ となる $h_1, \ldots, h_n$ は，$\overrightarrow{\mathcal{C}}$ と $\overrightarrow{h}$ から一意に定まらない場合があることに注意する．たとえば，$\overrightarrow{\mathcal{C}} = \Box_1 \centerdot \Box_2$ と $h =$ <a> に対し，$\overrightarrow{\mathcal{C}}[$<a>$, \varepsilon] = h$ とも書けるし $\overrightarrow{\mathcal{C}}[\varepsilon,$ <a>$] = h$ とも書ける．

簡便のため，

$$f(\overrightarrow{p}) \doteq \mathbf{let}\ \ldots t \doteq g(\overrightarrow{x}) \ldots\ \mathbf{in}\ \overrightarrow{\mathcal{C}}(t)$$

を

$$f(\overrightarrow{p}) \doteq \mathbf{let}\ \ldots\ \mathbf{in}\ \overrightarrow{\mathcal{C}}(g(\overrightarrow{x}))$$

と書くことがある．また，

$$f(\overrightarrow{p}) \doteq \mathbf{let}\ \mathbf{in}\ \overrightarrow{q}$$

を

$$f(\overrightarrow{p}) \mathrel{\hat{=}} \overrightarrow{q}$$

と書く．

プログラム中で，各式は一意な識別子を持っていると仮定する．式 e の識別子を $\#e$ により表す．規則

$$\begin{aligned} r = f(\overrightarrow{p}) \mathrel{\hat{=}} & \ \mathbf{let}\,\{(\overrightarrow{y_i}) \mathrel{\hat{=}} f_i(\overrightarrow{x_i})\}_{i\in\{1,\ldots,n\}} \\ & \ \mathbf{in}\ (\overrightarrow{q}) \end{aligned}$$

における未使用の変数の集合 $\mathsf{lostvars}(r)$ を

$$\mathsf{lostvars}(r) = \Big(\Big(\mathsf{vars}(\overrightarrow{p}) \setminus \bigcup\{\overrightarrow{x_i}\}\Big) \cup (\bigcup\{\overrightarrow{y_i}\})\Big) \setminus \mathsf{vars}(\overrightarrow{q})$$

で定義する．

パターン $\overrightarrow{p}$ に出現する変数パターンを集めて得られる型環境を $\Gamma_{\overrightarrow{p}}$ と書く．たとえば，$p = (x :: T, \texttt{<t>}(y :: T'))$ について，$\Gamma_p = \{x \mapsto T, y \mapsto T'\}$ となる．代入 θ を，第 5 章の言語 $\mathrm{V{\scriptsize DL}}$ のときと同様に，変数を生垣もしくは元と同じ変数に割り当てる関数であると定める．また，$\overrightarrow{t}\theta$ により，式もしくはパターンに現れる変数 x を $\theta(x)$ で置き換えて得られる式もしくはパターンを表す．ただし，パターンについては，パターン $\overrightarrow{p}$ 中に現れる全ての変数について $\theta(x) \in [\![\Gamma_{\overrightarrow{p}}(x)]\!]_G$ となっている場合に限り，$\overrightarrow{p}\theta$ と書く．パターン p に対し，p に照合する全ての生垣を $[\![p]\!]_{\mathcal{P}} = \{h \mid \exists\theta.\, p\theta = h\}$ で表す．変数パターン $x :: T$ について，$[\![x :: T]\!] = [\![T]\!]_G$ であることに注意する．プログラム $\mathcal{P}$ が文脈から明らかな場合 $[\![p]\!]_G$ を単に $[\![p]\!]$ と書く場合がある．

本論文において，プログラムは決定的であると仮定する．すなわち，パターンマッチングは一意である．形式的には，全てのパターン p について，$p\theta = h = p\eta$ かつ $\mathsf{dom}(\theta) = \mathsf{dom}(\eta) = \mathsf{vars}(p)$ ならば，$\theta = \eta$ であり，同じ関数の異なる二つの規則 $f(\overrightarrow{p}) \mathrel{\hat{=}} \ldots$ と $f(\overrightarrow{p}') \mathrel{\hat{=}} \ldots$ に対し，$\overrightarrow{p}\theta = v = \overrightarrow{p}'\eta$ となる代入 θ と η は存在しない．前者は，全ての連接パターン $p_1 \centerdot p_2$ について，$[\![p_1]\!]_{\mathcal{P}} \parallel [\![p_2]\!]_{\mathcal{P}}$ であれば必要十分である．また，後者は同じ関数の異なる二つの規則 $f(\overrightarrow{p}) \mathrel{\hat{=}} \ldots$ と $f(\overrightarrow{p}') \mathrel{\hat{=}} \ldots$ に対し，$[\![\overrightarrow{p}]\!]_{\mathcal{P}} \cap [\![\overrightarrow{p'}]\!]_{\mathcal{P}'} \mathrel{\hat{=}} e' = \emptyset$ であれば必要十分である．たとえば，以下の関数 f と g はどちらも決定的ではない．

$$\begin{array}{ll} f(x :: \texttt{<a>}^* \centerdot y :: \texttt{<a>}^*) & \mathrel{\hat{=}} \ldots \\ g(x :: \texttt{<a>}) & \mathrel{\hat{=}} \ldots \\ g(x :: \texttt{<a>}^* \centerdot y :: \texttt{<a>}) & \mathrel{\hat{=}} \ldots \end{array}$$

言語 $\mathrm{V{\scriptsize DL}}^+$ は図 6.2 に示す値呼びの意味を持つ．図中で関係 $e \Downarrow v$ は，式 e が値 v に評価されると読む．言語 $\mathrm{V{\scriptsize DL}}$ と同様に，式 e の型環境 Γ における可能な評価結果を，式 e の Γ における**値域**，$\mathsf{ran}_\Gamma(e) = \{v \mid e\theta \Downarrow v, \mathsf{vars}(e\theta) = \emptyset\}$ で定める．また，プログラム $\mathcal{P}$ 中の関数 f の意味を，

$$[\![f]\!]_P(\overrightarrow{v}) = \begin{cases} u & \text{if } f(\overrightarrow{v}) \Downarrow \overrightarrow{u}, \\ \bot & \text{otherwise.} \end{cases}$$

$$\frac{}{\varepsilon \Downarrow \varepsilon}\text{Eps} \quad \frac{e \Downarrow v}{\sigma(e) \Downarrow \sigma(v)}\text{Con} \quad \frac{e_1 \Downarrow v_1 \quad e_2 \Downarrow v_2}{e_1 \centerdot e_2 \Downarrow v_1 \centerdot v_2}\text{Cat}$$

$$\frac{\begin{array}{c}\exists f(\overrightarrow{p}) \hat{=} \textbf{let } \{(\overrightarrow{y_i}) \hat{=} g_i(\overrightarrow{x_i})\}_{i\in\{1,\ldots,n\}} \textbf{ in } (\overrightarrow{q}), \exists\theta.\ \overrightarrow{p}\theta = \overrightarrow{u} \\ g_i(\overrightarrow{x_i}\theta) \Downarrow \overrightarrow{t_i}, \quad \sigma := \{x_{ij} \mapsto t_{ij} \mid (x_{i1},\ldots,x_{in_i}) = \overrightarrow{x_i}, (t_{i1},\ldots,t_{in_i}) = \overrightarrow{t_i}\} \\ \overrightarrow{v} = (\overrightarrow{q}\sigma)\theta\end{array}}{f(\overrightarrow{u}) \Downarrow \overrightarrow{v}}\text{Fun}$$

図 6.2. 言語 VDL$^+$ の操作的意味

で定める．

今後の議論において，病的な場合を避けるために，プログラムに対しいくつかの仮定を置く．一つは，ラベルの集合 Σ は少くとも一つのラベルを含む．つまり，順方向変換の入力の集合 $\mathcal{H}_\Sigma$ は二つ以上の元を持つとする．また，プログラムは全ての入力に対して未定義な関数を含まないとする．たとえば，以下の関数 f は，全ての入力に対して未定義である．

$$f(x) \hat{=} f(x)$$

6.3.2 言語 VDL$^+$ の変換記述例

言語 VDL$^+$ で記述された変換のいくつかの例を示す．以下の例は，全て言語 VDL では記述できなかった例である．

例 6.1 (逆転)**.** 以下の関数 *reverse* は入力の文字列を逆順にした文字列を返す．

$$\begin{array}{ll} reverse(\varepsilon) & \hat{=}\ \varepsilon \\ reverse(x :: Char \centerdot r :: Char^*) & \hat{=}\ reverse(r) \centerdot x \end{array}$$

たとえば，列 `kazutaka` に *reverse* を適用すると，列 `akatuzak` が得られる．

ここで，我々は，全ての文字はラベルとしてエンコードされていると仮定している．

例 6.2 (別の逆転の定義)**.** 以下も *reverseR* も，入力の文字列を逆順にした文字列を出力する．

$$\begin{array}{ll} reverseR(\varepsilon) & \hat{=}\ \varepsilon \\ reverseR(r :: Char^* \centerdot x :: Char) & \hat{=}\ x \centerdot reverseR(r) \end{array}$$

例 6.3 (鏡像)**.** 以下の関数 *mirror* は，入力の文字列と入力の文字列を逆転したものを連接して返す．

$$\begin{array}{ll} mirror(\varepsilon) & \hat{=}\ \varepsilon \\ mirror(x :: Char \centerdot r :: Char^*) & \hat{=}\ x \centerdot mirror(x) \centerdot x \end{array}$$

例 6.4 (完全二分木)**.** 以下の関数 cb は，入力とする自然数から完全二分木を構成する．

$$
\begin{aligned}
cb(\texttt{<z>}) &\mathrel{\hat{=}} \texttt{<tip>} \\
cb(\texttt{<s>} \centerdot r) &\mathrel{\hat{=}} \mathbf{let}\, x \mathrel{\hat{=}} cb(r)\, \mathbf{in}\ \texttt{<bin>}(x \centerdot x)
\end{aligned}
$$

例 6.5 (目次の付与)**.** 関数 toc は，$c2x$ の結果に加え，目次を付与する．

$$
\begin{aligned}
&\mathbf{data}\ P \ \mathrel{\hat{=}} (\texttt{<p>}(String))^{*} \\
&\mathbf{data}\ Sp \mathrel{\hat{=}} (\texttt{<section>}(\texttt{<t>}(String) \centerdot P))^{+}
\end{aligned}
$$

$$
\begin{aligned}
&toc(x) \mathrel{\hat{=}} \mathbf{let}\,(s,t) \mathrel{\hat{=}} f(x)\, \mathbf{in}\ \texttt{<ul>}(s) \centerdot t \\
&f(\varepsilon) \mathrel{\hat{=}} (\varepsilon,\varepsilon) \\
&f(\texttt{<chapter>}(\texttt{<title>}(x) \centerdot p :: P) \centerdot r) \\
&\quad \mathrel{\hat{=}} \mathbf{let}\,(tocR, r') \mathrel{\hat{=}} f(r) \\
&\quad\quad \mathbf{in}\ (\texttt{<li>}(x) \centerdot tocR, \texttt{<h1>}(t) \centerdot p \centerdot r') \\
&f(\texttt{<chapter>}(\texttt{<title>}(x) \centerdot p :: P \centerdot s :: Sp) \centerdot r) \\
&\quad \mathrel{\hat{=}} \mathbf{let}\,(tocS, s') \mathrel{\hat{=}} g(s) \\
&\quad\quad\quad\quad (tocR, r') \mathrel{\hat{=}} f(r) \\
&\quad\quad \mathbf{in}\ (\texttt{<li>}(x) \centerdot \texttt{<ul>}(tocS) \centerdot tocR, \texttt{<h1>}(t) \centerdot p \centerdot s' \centerdot r') \\
&g(\varepsilon) \mathrel{\hat{=}} (\varepsilon,\varepsilon) \\
&g(\texttt{<section>}(\texttt{<title>}(x) \centerdot p :: P) \centerdot r) \\
&\quad \mathrel{\hat{=}} \mathbf{let}\,(tocS, r') \mathrel{\hat{=}} g(s)\, \mathbf{in}\ (\texttt{<li>}(x) \centerdot tocS, \texttt{<h2>}(x) \centerdot p \centerdot r')
\end{aligned}
$$

関数 f の第二規則により，章が節を含まない場合に，子が空の <ul> を生成しないようにしている．たとえば，toc は入力

```
<chapter>
    <title>chapterTitle_1</title>
    <p>para_1</p>
    <p>para_2</p>
    <section>
        <title>sectionTitle_2</title>
        <p>para_3</p>
    </section>
</chapter>
<chapter>
    <title>chapterTitle_3</title>
    <p>para_4</p>
</chapter>
```

に対し，次を返す.

```
<ul>
    <li>$chapterTitle_1$</li>
    <ul>
        <li>$sectionTitle_2$</li>
    </ul>
    <li>$chapterTitle_3$</li>
</ul>
<h1>$chapterTitle_1$</h1>
<p>$para_1$</p>
<p>$para_2$</p>
<h2>$sectionTitle_2$</h2>
<p>$para_3$</p>
<h1>$chapterTitle_3$</h1>
<p>$para_4$</p>
```

例 6.6 (転置). ソースとして，以下のような，それぞれの学生ごとに，数学 `<math>`，物理 `<physics>` および化学 `<chemistry>` の点数を記録したものを考える.

```
<results>
    <result>
        <name>Alice</name>
        <scores>
            <math>72</math><physics>65</physics><chemistry>85</chemistry>
        </scores>
    </result>
    <result>
        <name>Bob</name>
        <scores>
            <math>96</math><physics>85</physics><chemistry>84</chemistry>
        </scores>
    </result>
    <result>
        <name>Charlie</name>
        <scores>
            <math>85</math><physics>97</physics><chemistry>62</chemistry>
        </scores>
    </result>
</results>
```

以下の変換 *result2scores*

```
results2scores(<results>(x))
    ≙ let (ms, ps, cs) ≙ f(r)
      in <scores>( <math_scores>(ms)
                   . <physics_scores>(ps)
                   . <chemistry_scores>(cs) )
f(ε) ≙ ε
f(<result>(<name>(n) . <scores>(<math>(m) . <physics>(p) . <chemistry>(c))) . r)
    ≙ let (ms, ps, cs) ≙ f(r)
      in ( <name>(n) . <score>(m) . ms,
           <name>(n) . <score>(p) . ps,
           <name>(n) . <score>(c) . cs )
```

は上記ソースから以下のビューを構築する．

```
<scores>
    <math_scores>
        <name>Alice</name><score>72</score>
        <name>Bob</name><score>96</score>
        <name>Charlie</name><score>85</score>
    </math_scores>
    <physics_scores>
        <name>Alice</name><score>65</score>
        <name>Bob</name><score>85</score>
        <name>Charlie</name><score>97</score>
    </physics_scores>
    <chemistry_scores>
        <name>Alice</name><score>85</score>
        <name>Bob</name><score>85</score>
        <name>Charlie</name><score>62</score>
    </chemistry_scores>
</scores>
```

例 6.7 (文献の選り分け[1])**.** 以下の変換 *bkref* は，文献リストを `<author>` を持つ書籍と，

[1] 変換 *bkref* は XML Query Use Cases（`http://www.w3.org/TR/xquery-use-cases`）の “XMP”-Q11 を簡略化したものである．元の変換においては，ソースは XML 属性を含んでいるが，言語 VDL+ は XML 属性を考えていない．

<editor> を持つ文集に選り分ける．

$$
\begin{aligned}
&\mathbf{data}\ \mathit{Author} && \doteq \texttt{<author>}(\texttt{<last>}(\mathit{String})\,\mathbf{.}\,\texttt{<first>}(\mathit{String}))\\
&\mathbf{data}\ \mathit{Authors} && \doteq \mathit{Author}\,\mathbf{.}\,\mathit{Author}^*\\
&\mathbf{data}\ \mathit{Editor} && \doteq \texttt{<editor>}(\texttt{<last>}(\mathit{String})\,\mathbf{.}\,\texttt{<first>}(\mathit{String})\,\mathbf{.}\,\texttt{<affiliation>}(\mathit{String}))\\
&\mathbf{data}\ \mathit{Editors} && \doteq \mathit{Editor}\,\mathbf{.}\,\mathit{Editor}^*\\
&\mathbf{data}\ \mathit{Rest} && \doteq (\texttt{<publisher>}(\mathit{String})\,\mathbf{.}\,\texttt{<price>}(\mathit{String}))
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{bkref}(\texttt{<bib>}(r)) \doteq \mathbf{let}\ (x,y) \doteq f(r)\ \mathbf{in}\ \texttt{<bib>}(x\,\mathbf{.}\,y)\\
&f(\varepsilon) \doteq (\varepsilon,\varepsilon)\\
&f(\texttt{<book>}(\texttt{<title>}(t)\,\mathbf{.}\,\mathit{as} :: \mathit{Authors}\,\mathbf{.}\,p :: \mathit{Rest})\,\mathbf{.}\,r)\\
&\qquad \doteq \mathbf{let}\ (x,y) \doteq f(r)\ \mathbf{in}\ (\texttt{<book>}(\texttt{<title>}(t)\,\mathbf{.}\,\mathit{as})\,\mathbf{.}\,x, y)\\
&f(\texttt{<book>}(\texttt{<title>}(t)\,\mathbf{.}\,e :: \mathit{Editors}\,\mathbf{.}\,p :: \mathit{Rest})\,\mathbf{.}\,r)\\
&\qquad \doteq \mathbf{let}\ (x,y) \doteq f(r)\\
&\qquad\qquad\quad a \qquad \doteq \mathit{affil}(e)\\
&\qquad\quad\ \mathbf{in}\ (x, \texttt{<reference>}(\texttt{<title>}(t)\,\mathbf{.}\,a)\,\mathbf{.}\,y)\\
&\mathit{affil}(\varepsilon) \doteq \varepsilon\\
&\mathit{affil}(\texttt{<editor>}(\texttt{<last>}(n)\,\mathbf{.}\,\texttt{<first>}(m)\,\mathbf{.}\,\texttt{<affiliation>}(a))\,\mathbf{.}\,r)\\
&\qquad \doteq \mathbf{let}\ x \doteq \mathit{affil}(r)\ \mathbf{in}\ \texttt{<affilication>}(a)\,\mathbf{.}\,x
\end{aligned}
$$

多返値関数により，このような変換を元データを二度操作することなしに書くことができる．

6.4　双方向化における問題

言語 $\mathrm{V{\small DL}}^+$ で記述された順方向変換から，効果的な逆方向変換を導出するためには，いくつかの問題を解決しなければならない．

6.4.1　式の値域および関数の単射性の文脈依存性

言語 $\mathrm{V{\small DL}}$ とは異なり，言語 $\mathrm{V{\small DL}}^+$ において，関数自体の値域と，その関数を呼出している関数呼出式の値域は異なりうる．たとえば，以下のプログラムにおいて，g の値域は $\texttt{<a>}^*$ であるが，f の右辺における g の関数呼出式の値域は $\texttt{<a>}$ である．

$$
\begin{aligned}
f(x :: \texttt{<a>}) &\doteq g(x)\\
g(x :: \texttt{<a>}^*) &\doteq x
\end{aligned}
$$

また，これにより，呼ばれる関数の単射性が異なる場合もある．たとえば，以下のプログラムにおいて，$\mathit{unifyAddress}$ 自体は単射ではないが，main から呼び出される関数 $\mathit{unifyAddress}$

は単射である．

$$
\begin{aligned}
&\textbf{data}\ T = \texttt{<email>}(\mathit{EmailText}) \mid \texttt{<tel>}(\mathit{TelNumber})\\
&\quad \{\texttt{-}\ [\![\mathit{EmailText}]\!] \text{ と } [\![\mathit{TelNumber}]\!] \text{ は互いに素 } \texttt{-}\}\\
\\
&\mathit{main}(x :: T) \mathrel{\hat{=}} \mathit{unifyAddress}(x)\\
&\mathit{unifyAddress}(\texttt{<email>}(x)) \mathrel{\hat{=}} \texttt{<address>}(x)\\
&\mathit{unifyAddress}(\texttt{<tel>}(x)) \quad\ \mathrel{\hat{=}} \texttt{<address>}(x)
\end{aligned}
$$

効果的な逆方向変換を求めるためには，文脈ごとに異なる関数呼出の値域や単射性をできるだけ正確に把握したい．

この問題は，関数の定義域よりも引数の型のほうが小さい，つまり部分集合になっている場合に起こる．したがって，素朴な解決策は，関数の定義域のよりも引数の型のほうが小さいような関数呼出を禁止することである．しかし，この解決策は以下の二つの理由により望ましくない．

- 実際に変換プログラムを記述する場合に，関数の定義域のほうが引数の型より大きいことはよくある．なぜなら，引数の型に対応した定義域を持つ関数を記述するのは，しばしば，煩雑であり，また関数自体の再利用性を下げるためである．たとえば，<a>, <b>, <c>, <d> を順不同一つずつ含む入力を処理したいとすると，この入力に対応した定義域を持つ関数は，

$$
\left.
\begin{aligned}
f(\texttt{<a>}\centerdot\texttt{<b>}\centerdot\texttt{<c>}\centerdot\texttt{<d>} &\mathrel{\hat{=}} \ldots\\
f(\texttt{<a>}\centerdot\texttt{<b>}\centerdot\texttt{<d>}\centerdot\texttt{<c>} &\mathrel{\hat{=}} \ldots\\
&\ldots\\
f(\texttt{<d>}\centerdot\texttt{<c>}\centerdot\texttt{<b>}\centerdot\texttt{<a>} &\mathrel{\hat{=}} \ldots
\end{aligned}
\right\} 4! = 24 \text{ 個の規則}
$$

  と煩雑なものになるが，もし，処理結果が要素の順序に依存せずまた要素ごとに処理が記述できる場合には，より定義域の広い関数 f'

$$
\begin{aligned}
&\textbf{data}\ T \mathrel{\hat{=}} (\texttt{<a>} \mid \texttt{<b>} \mid \texttt{<c>} \mid \texttt{<d>})\\
&f'(\varepsilon) \qquad\qquad\quad \mathrel{\hat{=}} \varepsilon\\
&f'(a :: T\centerdot x :: T^{*}) \mathrel{\hat{=}} g(a)\centerdot f(x)\\
&g(\texttt{<a>}) \mathrel{\hat{=}} \ldots\\
&g(\texttt{<b>}) \mathrel{\hat{=}} \ldots\\
&g(\texttt{<c>}) \mathrel{\hat{=}} \ldots\\
&g(\texttt{<d>}) \mathrel{\hat{=}} \ldots
\end{aligned}
$$

  により簡潔に変換が記述できる．また，f' は，<a> は複数個含んでもよいなどの入力の仕様に多少の変更があった場合にも f' のプログラムを変換せずに利用できるため，再利用性が高く，仕様の変更に頑健である．

- 我々は，第 5 章の終わりにおいて，

$$h(x) \mathrel{\hat{=}} f(g(x))$$

という形で定義される関数 h の補関数を

$$h^{\mathrm{c}}(x) \mathrel{\hat{=}} (f^{\mathrm{c}}(g(x)), g^{\mathrm{c}}(x))$$

という形で求める場合において，f^{c} を求めるのに，g の値域の情報を用いることを述べた．もし，関数の定義域が引数の型より広くなることを許せば，

$$f'(x :: OutputG) \mathrel{\hat{=}} f(x)$$

のような形式で，f の補関数を求める際において，簡単に g の値域の情報（$OutputG$）を与えることができる．

第7章では，関数を引数に対し特化することにより，プログラムを，関数の値域と関数呼出式の値域が一致するプログラムに変換することでこの問題を解決する．

6.4.2　値域の重なりの判定

言語 VDL$^+$ で記述された関数の値域は，正規生垣文法によりも複雑になる．これにより，言語 VDL では式同士の値域の重なりの判定が決定可能であったの対し，言語 VDL$^+$ においては式同士の値域の重なりの判定が決定不能になる（第8章8.1.1節）．つまり，式同士の値域に重なりがあるかどうかを厳密に判定するアルゴリズムは存在しない．また，プログラムが多返値関数がなく連接演算子を含む場合も，連接演算子がなく多返値関数を含む場合もともに式同士の値域に重なりがあるかどうかは一般には決定不能であることを示すことができる．そのため，我々は式同士の値域の重なりを判定するために，値域を，式同士の値域の重なりが判定可能である集合により近似したり（第8章8.1.2節），また，止まらない場合のある手続きにより値域の重なり探索し適宜探索を打ち切ったり（第8章8.3節）する．

第7章　引数の型に基づく関数の特化

前章の言語 V_{DL}^+ で記述されたプログラム対し，第5章の V_{DL} に対する議論を直接適用したのでは，あまり小さい補関数が得られない場合がある．これは，V_{DL}^+ では，変数パターンが $x :: T$ のように型を含むため，関数呼出式の値域と関数の値域は異なる場合があるためである．例として，次のプログラムを考える．

$$\begin{array}{l}\textbf{data}\ T = \texttt{<email>}(\mathit{EmailText}) \mid \texttt{<tel>}(\mathit{TelNumber}) \\ \quad \{\text{-}\ [\![\mathit{EmailText}]\!]\ \text{と}\ [\![\mathit{TelNumber}]\!]\ \text{は互いに素}\ \text{-}\} \\ \\ \mathit{main}(x :: T) \mathrel{\hat{=}} \mathit{unifyAddress}(x) \\ \mathit{unifyAddress}(\texttt{<email>}(x)) \mathrel{\hat{=}} \texttt{<address>}(x) \\ \mathit{unifyAddress}(\texttt{<tel>}(x)) \quad\ \mathrel{\hat{=}} \texttt{<address>}(x)\end{array}$$

上のプログラムにおいて，$\mathit{unifyAddress}$ 関数の値域は，

$$\mathsf{ran}(\mathit{unifyAddress}) = \{\texttt{<address>}(h) \mid h \in \mathcal{H}_\Sigma\}$$

であるが，main 関数の右辺に出現する式 $\mathit{unifyAddress}(x)$ の値域は

$$\mathsf{ran}_{\{x \to T\}}(\mathit{unifyAddress}(x)) = \{\texttt{<address>}(h) \mid h \in (\mathit{EmailText} \cup \mathit{TelNumber})\}$$

と，$\mathsf{ran}(\mathit{unifyAddress})$ とは異なる．また，$\mathit{unifyAddress}$ はそれ自身は単射関数ではないが，入力が T に制限された関数 $\mathit{unifyAddress}|_T$ は単射となる．

我々の目標の一つは，縮約順序（第3章，定義3.8）においてより小さい補関数を求めることである．よって，関数 f の定義域を T に制限した関数 $f|_T$ に対し，以下を議論したい．

- $f|_T$ の値域，つまり，$\mathsf{ran}(f|_T) = \mathsf{ran}_{\{x \to T\}}(f(x))$ の推定．
- $f|_T$ が単射であるかどうか．

そのため，我々は，関数 f を型 T に特化し $f|_T$ の定義規則を求めることを考える．たとえば，$\mathit{unifyAddress}$ とその関数呼出 $\mathit{unifyAddress}(x)$ における x の型 T に対し，以下の関数定義規則を求める．

$$\begin{array}{l}\mathit{unifyAddress}|_T(\texttt{<email>}(x :: \mathit{EmailText})) \mathrel{\hat{=}} \texttt{<address>}(x) \\ \mathit{unifyAddress}|_T(\texttt{<tel>}(x :: \mathit{TelNumber})) \quad \mathrel{\hat{=}} \texttt{<address>}(x)\end{array}$$

そこで，本章では，正規生垣文法という型 T の構造に基づき，V_{DL}^+ で記述された関数 f から，その定義域を T に制限した関数 $f|_T$ を自動的に求める手法を提案する．本手法は以下の特徴を持つ．

- **提案手法は，関数 f を完全に T に特化する．** すなわち，特化された後では，$\mathsf{dom}(f|_T) = \mathsf{dom}(f) \cap [\![T]\!]$ となる．このことにより，関数呼出と関数の値域を区別せずによくなるため，正確に式の値域を求めることができるようになる．

- **提案手法の停止性が保証できる．** 変換が停止するかどうかは，プログラム変換において重要な関心毎の一つである．提案手法は，全ての関数を完全に特化するにもかかわらず，必ず停止する．

本節は次の通りに構成される．まず 7.1 節において，本章の提案する特化手法の関連研究を述べる．次に 7.2 節において，提案する特化手法のアイデアと素朴な実現における問題について述べる．続く 7.3 節では，言語 $\mathrm{V{\scriptsize DL}}^+$ で記述されたプログラムに対する具体的な特化手法を示し，提案する特化アルゴリズムが満たすいくつかの性質を示す．そして 7.4 節において，本手法の拡張について述べる．最後に 7.5 節において，本章の内容を取り纏める．

本章の内容は，文献 [MHT09] にて発表された．また，本章の内容のプロトタイプ実装が `http://www.ipl.t.u-tokyo.ac.jp/~kztk/sp/` にある．

7.1 関連研究

本章で提案する型に基づく関数の特化手法に関連する研究について述べる．

本章で我々の提案する特化手法は，木変換器の厳密な型推論 [MPS07] において使用された前処理の拡張である．我々の特化手法と彼らの特化手法の主な違いは以下の二点である．一つ目は，我々は x :: <a>* . y :: <b> のような連接パターンを許すことである．二つ目は，我々の特化手法は決定的（deterministic）なプログラムを導出することである．

また，提案する特化手法において，我々は，パターンがある型に制限された場合のそのパターンに出現する変数の型の推論と同様なことを行っている．我々が特化において使用している型推論は，対象言語が制限されているため，既存の手法 [BCF03, Hos03, HP03] よりも正確である．このパターン中の変数の型の推論が正確あることは，特化手法が完全に関数を特化することを保証する上で重要である．

XML 変換において，型に基づくパターンマッチの最適化についての研究がいくつかある [Fri04, LP05]．彼らも，型に基づいて「特化」を行っていると言える．しかし，我々の特化と彼らの特化は多くの点で異なっている．まず，我々の目的は，関数 f と型 T から関数 $f|_T$ の定義を求めることであるのに対し，彼らの目的は，パターンマッチの効率的な評価手法を与えることにある．次に，我々はパターンの関数呼出を特化するがパターンマッチ自体は特化しないのに対し，彼らはパターンマッチ機構をコンパイルことにより特化するが関数呼出に対しては何もしない．また，我々の特化手法において，7.2 節の *reverse* に見るように，再帰関数の同じ関数呼出式を別の型により特化することがあるのに対し，彼らの手法では入力の型は予め与えられ，再帰毎に特化の対象となる型が変わることはない．なお，パ

ターンの表現力は，我々が対象とする $\mathrm{V}_{\mathrm{DL}}^{+}$ よりも，彼らが議論の対象としている言語のほうが表現力が真に大きい．

7.2　提案手法のアイデア

ここでは，提案する型に対する関数の特化手法のアイデアと，単純な特化の実現の問題点について述べる．

7.2.1　アイデア

提案する特化手法の基本的なアイデアは単純である．つまり，以下のように，関数呼出時の引数の型 $T_1, \ldots, T_n$ に対して，関数 f を特化し，$f|_{T_1,\ldots,T_n}$ の定義を生成する．

$$\begin{gathered} f(\ldots, \ldots x :: S \ldots, \ldots) \mathrel{\hat{=}} \ldots g(x) \ldots \\ \downarrow \\ f|_{T_1,\ldots,T_n}(\ldots, \ldots x :: S' \ldots, \ldots) \mathrel{\hat{=}} \ldots g|_{S'}(x) \ldots \end{gathered}$$

ここで，S'_1 は，ある T_i から計算される，関数 f の定義域 $T_1 \times \cdots \times T_n$ に対する値と，$f|_{T_1,\ldots,T_n}$ の値が一致するように定まる型である（7.3 節）．たとえば，関数 *reverse*

$$\begin{aligned} &\mathit{reverse}(\varepsilon) &&\mathrel{\hat{=}} \varepsilon \\ &\mathit{reverse}(a :: \mathit{Elem} \cdot r :: \mathit{Elem}^*) &&\mathrel{\hat{=}} \mathit{reverse}(x) \cdot a \end{aligned}$$

を型 $T = (\texttt{<a>} \cdot \texttt{<b>})^*$ に対して特化すると以下を得る．

$$\begin{aligned} &\mathit{reverse}|_T(\varepsilon) &&\mathrel{\hat{=}} \varepsilon \\ &\mathit{reverse}|_T(a :: \texttt{<a>} \cdot r :: T') &&\mathrel{\hat{=}} \mathit{reverse}|_{T'}(x) \cdot a \\ &\mathit{reverse}|_{T'}(a :: \texttt{<b>} \cdot r :: T) &&\mathrel{\hat{=}} \mathit{reverse}|_T(x) \cdot a \end{aligned}$$

ここで，T' は $T' = \texttt{<b>} \cdot T$ により定まる型である．

この特化の考え方自体は新しいものではない．特化は部分評価 [JGS93] 以外の何物でもないし，また，既に関数の呼出の型を厳密に求めるために用いられている [MPS07]．本章の貢献は，以下の二つの性質を保証するように，特化手法を定めた点にある．

7.2.2　素朴な実現の問題点

特化の停止性

変換の停止性は，プログラム変換の重要な性質の一つである．しかし，$\mathrm{V}_{\mathrm{DL}}^{+}$ に対する特化の停止性の証明はそう簡単ではない．たとえば，関数 f

$$f(\ldots x :: S \ldots) \mathrel{\hat{=}} \ldots g(x) \ldots$$

を型 T に特化するとき，以下の規則が生成されたとする．

$$f|_S(\ldots x :: S' \ldots) \mathrel{\hat{=}} \ldots g|_{S'}(x) \ldots,$$

ここで，新たな型 S' と新たな関数の呼出 $g|_{S'}(x)$ が生成される．よって，$f|_S$ の定義を得るために，関数 g を型 S' に対し特化し，関数 $g|_{S'}$ の定義を得る必要がある．型に基づく特化の処理は，特化により生成された規則に含まれる全ての関数呼出の式 $g|_{S'}(x)$ について，$g|_{S'}$ の規則が全て既に生成されている場合に完了する．しかし，型に基づく特化の処理が止まるのか，それとも，無限に新たな型と新たな関数の呼出を生成しつづけるのかは明らかではない．言語 VDL$^+$ により変数パターンの連接 $x :: U \centerdot y :: V$ により，停止性の保証は難しくなっている．もし，変数パターンや連接パターンがなければ，オートマトンの積構成 [CDG$^+$97] の技術を用いて，文献 [MPS07] と同様に特化の停止性が証明できる．

決定的なプログラムの導出

停止性とは別の重要な性質の一つは，出力されたプログラムが決定的（deterministic）になることである．我々の双方向化の議論，特に単射性の解析は，入力となるプログラムが決定的であることに依存している．また，本章の特化は，双方向化に関連する応用を持つが，それらにおいても都合がよいものではない．型に基づく特化は，一つの関数定義規則から，複数の関数定義規則を生成する場合がある．たとえば，以下の関数 $idAB$

$$idAB(x :: (\texttt{<a>}|\texttt{<b>}) \centerdot y :: (\texttt{<a>}|\texttt{<b>})) \mathrel{\hat{=}} x \centerdot y$$

を型 $T = (\texttt{<a>} \centerdot \texttt{<b>}) \mid (\texttt{<b>} \centerdot \texttt{<a>})$ に対して特化する以下の関数 $idAB|_T$ となる．

$$idAB|_T(x :: \texttt{<a>} \centerdot y :: \texttt{<b>}) \mathrel{\hat{=}} x \centerdot y$$
$$idAB|_T(x :: \texttt{<b>} \centerdot y :: \texttt{<a>}) \mathrel{\hat{=}} x \centerdot y$$

このような場合に，正規生垣文法の操作を慎重にしなければ，非決定的なプログラムを生成しうる．

7.3　型に基づく関数の特化

ここでは，本章で我々が提案する型に基づく関数の特化手法について述べる．本手法は，関数を，関数呼出における引数の型に対して特化する．本手法によって特化されたプログラムにおいて，全ての関数呼出 $f(x_1, \ldots, x_n)$，変数の型を $x_1 :: T_1, \ldots, x_n :: T_n$ とした場合に，f の定義規則 $f(\overrightarrow{p_1}) \mathrel{\hat{=}} e_1, \ldots, f(\overrightarrow{p_n}) \mathrel{\hat{=}} e_n$ において，f の見た目の定義域 $\bigcup_{1 \le i \le n} [\![\overrightarrow{p_i}]\!]$ が $[\![T_1]\!] \times \cdots \times [\![T_n]\!]$ と等しくなる．大雑把に言えば，関数 f を型 T に対し特化する場合において，提案する特化手法は，以下のようにプログラムを変換する．

$$f(\ldots x :: S \ldots) \mathrel{\hat{=}} \ldots g(x) \ldots$$
$$\downarrow$$
$$f|_T(\ldots x :: S' \ldots) \mathrel{\hat{=}} \ldots g|_{S'}(x) \ldots$$

ここで，f のパターン中の各変数パターン $x :: S$ は，型 T に対する**パターン特化**により $x :: S'$ に置き換わる．そして，提案する特化手法は g を S' に対し特化することにより，$g|_{S'}$ の定義規則を求める．

我々の特化手法を具体的な例 *reverse* と *idAB* を用いて説明する．

以下の関数 *reverse* を考える．

$$\begin{aligned}
&\textbf{data}\ S \mathrel{\hat{=}} \texttt{<a>} \mid \texttt{<b>} \\
&\mathit{reverse}(\varepsilon) \mathrel{\hat{=}} \varepsilon \\
&\mathit{reverse}(a :: S \cdot r :: S^*) \mathrel{\hat{=}} \mathit{reverse}(r) \cdot a
\end{aligned}$$

ここで，我々は関数 *reverse* を型 $T = (\texttt{<a>} \cdot \texttt{<b>})^*$ に対し特化する．ここで，T と S および S^* の意味は，以下の正規生垣文法で記述されているとする．

$$\begin{array}{lll}
T \to \texttt{<a>}T' & T \to \varepsilon & T' \to \texttt{<b>}T \\
S \to \texttt{<a>}S' & S \to \texttt{<b>}S' & S' \to \varepsilon \\
S^* \to \texttt{<a>}S^* & S^* \to \texttt{<b>}S^* & S^* \to \varepsilon
\end{array}$$

まず，我々の特化手法は，規則

$$\mathit{reverse}(\varepsilon) \mathrel{\hat{=}} \varepsilon$$

を型 T に対し特化しようとする．ここで，上の規則左辺パターン ε を型 T を特化する．ここで，集合 $[\![T]\!]$ は ε を含む．つまり，$T \xrightarrow{*} \varepsilon$ となる．よって，パターン ε はパターン ε へと特化され，次の規則が生成される．

$$\mathit{reverse}|_T(\varepsilon) \mathrel{\hat{=}} \varepsilon$$

次に，我々の特化手法は，規則

$$\mathit{reverse}(a :: S \cdot r :: S^*) \mathrel{\hat{=}} \mathit{reverse}(r) \cdot a$$

を型 T に対し特化しようとする．ここで，パターン $a :: S \cdot r :: S^*$ を型 T に対し特化することにより $a :: H_1 \cdot r :: H_2$ となったとする．このような生垣の集合 H_1 と H_2 を求めるのに，我々は生垣の集合 $h_1 \in [\![S]\!]$ と $h_2 \in [\![S^*]\!]$ で $h_1 \cdot h_2 \in [\![T]\!]$ を満たすものを考える．そのため，我々は非終端記号 N で $T \xrightarrow{*} h_1 N$ かつ $N \xrightarrow{*} h_2$ であり，さらに $[\![N]\!] \cap [\![S^*]\!] \neq \emptyset$ かつ $\{h \mid T \xrightarrow{*} hN\} \cap [\![S]\!] \neq \emptyset$ となっているものを求める．このとき，$\{h \mid T \xrightarrow{*} hT\} \cap [\![S]\!] = \emptyset$ であるが $[\![T']\!] \cap [\![S^*]\!] = [\![\texttt{<b>} \cdot S^*]\!]$ かつ $\{h \mid T \xrightarrow{*} hT'\} \cap [\![S]\!] = \{\texttt{<a>}\}$ であるため，以下の規則および型 H_1 と型 H_2 が生成される．

$$\begin{aligned}
&\textbf{data}\ H_1 \mathrel{\hat{=}} \texttt{<a>} \\
&\textbf{data}\ H_2 \mathrel{\hat{=}} \texttt{<b>} \cdot S \\
&\mathit{reverse}|_S(a :: H_1 \cdot r :: H_2) \mathrel{\hat{=}} \mathit{reverse}|_{H_2}(r) \cdot a
\end{aligned}$$

ここで，新たに生成された規則の右辺に，新たな関数呼出 $\mathit{reverse}|_{H_2}(r)$ が出現している．ここで，関数 $\mathit{reverse}|_{H_2}$ の規則はまだ生成されていないため，我々の特化手法は，関数 *reverse*

を型 H_2 に対し特化することにより $reverse|_{H_2}$ の規則を生成する．まず，我々の特化手法は，規則

$$reverse(\varepsilon) \mathrel{\hat{=}} \varepsilon$$

を型 H_2 に対し特化しようとする．ここで，パターン ε を型 H_2 に特化する．ところが，$[\![H_2]\!]$ は ε を含まない，つまり，この規則は $[\![H_2]\!]$ に属する入力に対し使われることはないため，特化手法はこの規則に対し何の規則も生成しない．次に，我々の特化手法は，規則

$$reverse(a :: S \centerdot r :: S^*) \mathrel{\hat{=}} reverse(r) \centerdot a$$

を型 H_2 に対して特化しようとする．前述の手続きと同様にして，以下の規則が生成される．

$$reverse|_{H_2}(a :: \mathtt{<b>} \centerdot r :: T) \mathrel{\hat{=}} reverse|_T(r) \centerdot a$$

ここで，新たに生成された規則の右辺式には，関数呼出 $reverse|_T(r)$ が含まれている．しかし，$reverse|_T$ の規則は既に生成されているため，特化が完了する．型 H_1 を式 $H_1 = \mathtt{<a>}$ により簡潔に書き直すと，特化された関数 $reverse|_T$ を定義するプログラムは以下となる．

$$\begin{array}{ll}
\mathbf{data}\ T \mathrel{\hat{=}} (\mathtt{<a>} \centerdot \mathtt{<b>})^* & \\
\mathbf{data}\ S \mathrel{\hat{=}} \mathtt{<a>} \mid \mathtt{<b>} & \\
\mathbf{data}\ H_2 \mathrel{\hat{=}} \mathtt{<b>} \centerdot T & \\
reverse|_T(\varepsilon) & \mathrel{\hat{=}} \varepsilon \\
reverse|_T(a :: \mathtt{<a>} \centerdot r :: H_2) & \mathrel{\hat{=}} reverse|_{H_2}(r) \centerdot a \\
reverse|_{H_2}(a :: \mathtt{<b>} \centerdot r :: T) & \mathrel{\hat{=}} reverse|_T(r) \centerdot a
\end{array}$$

特化手法が，一つの規則から複数の規則を生成する場合がある．関数 $idAB$ を考える．

$$idAB(x :: (\mathtt{<a>}|\mathtt{<b>}) \centerdot y :: (\mathtt{<a>}|\mathtt{<b>})) \mathrel{\hat{=}} x \centerdot y$$

関数 $idAB$ を型 $T = (\mathtt{<a>} \centerdot \mathtt{<b>}) \mid (\mathtt{<b>} \centerdot \mathtt{<a>})$ について特化することを考える．ここで，T の意味は，以下の正規生垣文法により表現される．

$$T \to \mathtt{<a>}U \quad T \to \mathtt{<b>}V \quad U \to \mathtt{<b>}W \quad V \to \mathtt{<a>}W \quad W \to \varepsilon$$

規則

$$idAB(x :: (\mathtt{<a>}|\mathtt{<b>}) \centerdot y :: (\mathtt{<a>}|\mathtt{<b>})) \mathrel{\hat{=}} x \centerdot y$$

を型 T に対して特化するため，まず，我々はパターン $x :: (\mathtt{<a>}|\mathtt{<b>}) \centerdot y :: (\mathtt{<a>}|\mathtt{<b>})$ を型 T に対して特化しようとする．前述の議論と同様，我々は生垣 h_1 と h_2 で，$h_1, h_2 \in [\![\mathtt{<a>}|\mathtt{<b>}]\!]$ かつ $h_1 \centerdot h_2 \in [\![T]\!]$ となっているものと考える．ところが，$reverse$ の場合と異なり，h_2 の選択が h_1 の選択に依存する．もし，h_2 として $h_2 = \mathtt{<a>}$ を選べば，h_1 は $h_1 = \mathtt{<b>}$ となり，もし，h_2 として $h_2 = \mathtt{<b>}$ を選べば，h_1 は $h_1 = \mathtt{<a>}$ となる．別の言い方をすると，集合 $\{h \mid T \xrightarrow{*} hU\}$ と集合 $\{h \mid T \xrightarrow{*} hV\}$ は異なる．ここで，U と V は，$[\![\mathtt{<a>}|\mathtt{<b>}]\!] \cap [\![U]\!] \neq \emptyset$ か

つ $[\![\texttt{<a>}|\texttt{<b>}]\!] \cap [\![V]\!] \neq \emptyset$ 満たし，他の非終端記号は満たさないことに注意する．結果として，$idAB$ の一つの規則に対し，以下の二つの規則が生成される．

$$\begin{aligned} idAB|_T(x :: \texttt{<a>} \centerdot y :: \texttt{<b>}) &\mathrel{\hat{=}} x \centerdot y \\ idAB|_T(x :: \texttt{<b>} \centerdot y :: \texttt{<a>}) &\mathrel{\hat{=}} x \centerdot y \end{aligned}$$

提案する特化手法を，双方向化など，他のプログラム変換や自動化された変換機構において使用するためには，以下の二点を明らかかすること重要である．

- 特化手法が停止するかどうか．
- 特化手法が決定的なプログラムを生成するかどうか．

二つ目を保証するためには，特化手法が一つの規則に対し二つの規則を導出する場合において，慎重な議論が必要になる．

7.3.1 パターン特化

ここでは，パターンの特化手法について述べる．前述の例の通り，我々は関数 f を型 T について特化するのに，関数 f の各規則を型 T に対し特化していた．また，規則 $r = f(p) \mathrel{\hat{=}} e$ の型 T に対する特化において，我々はまずパターン p を T に対し特化し，特化されたパターンに応じて式 e を特化した．

上の *reverse* と $idAB$ の特化において，我々は，非終端記号 A, B に対し，$\{h \mid A \xrightarrow{*} hB\}$ という形式の集合について議論した．この形式の集合は，確かに正規生垣文法になる．なぜなら，$G = (\Sigma, N, R)$ と $B \in N$ に対し，任意の A に対し $\{h \mid A \xrightarrow{*} hB\} = [\![A]\!]_{G'}$ となる G' を，$G' = (\Sigma, N \cup \{A' \mid A \in N\}, R')$ および

$$\begin{aligned} R' = \{B \to \varepsilon\} &\cup \{X \to \sigma(Y)Z \mid X \to \sigma(Y)Z \in R\} \\ &\cup \{X' \to \varepsilon \mid X \to \varepsilon \in R\} \cup \{X' \to \sigma(Y')Z' \mid X \to \sigma(Y)Z \in R\} \end{aligned}$$

により構成できるためである．しかし，このように，連接パターンに対し常に新しい正規生垣文法を作成していたのでは，提案する特化手法の停止性の議論は難しくなる．そこで，我々は**打ち切り正規生垣文法**の概念を導入する．これにより，$\{h \mid A \xrightarrow{*} hB\}$（$A$ の言語を非終端記号 B で打ち切ったもの）のような集合を簡潔かつ新しい正規生垣文法を生成することなしに表現することができる．

定義 7.1 (打ち切り正規生垣文法)**. 打ち切り正規生垣文法** $G\varepsilon E$ は正規生垣文法 $G = (\Sigma, N, R)$ と非終端記号の集合 E（$\subseteq N$）の二つ組である．打ち切り正規生垣文法 $G\varepsilon E$ における $A \in N$ の言語は $[\![A]\!]_{G\varepsilon E} = \{h \mid A \xrightarrow{*} hB, B \in E\}$ で定義される．

全ての正規生垣文法 $G = (\Sigma, N, R)$ は，$F = \{A \mid A \to \varepsilon \in R\}$ とすることで，打ち切り正規生垣文法 $G\varepsilon F$ に変換できる．また，打ち切り正規生垣文法 $G\varepsilon E$ に対し，$G\varepsilon E$ の全ての

非終端記号 A が同じ意味を持つ正規生垣文法 G' を，$G = (\Sigma, N, R)$ とした場合に，生成規則の集合

$$\begin{aligned} R' = &\{B \to \varepsilon \mid B \in N\} \cup \{X \to \sigma(Y\ )Z \mid X \to \sigma(Y)Z \in R\} \\ &\cup \{X' \to \varepsilon \mid X \to \varepsilon \in R\} \cup \{X' \to \sigma(Y')Z' \mid X \to \sigma(Y)Z \in R\} \end{aligned}$$

により，$G' = (\Sigma, N \cup \{A' \mid A \in N\}, R')$ とすることで構成できる．なお，打ち切り正規生垣文法の概念を経由しないものの，この打ち切り正規生垣文法から正規生垣文法への変換と同様な変換が，連接を含むパターンに出現する変数の型の推論において使用されている [Hos03].

正規生垣文法で，$G = (\Sigma, N, R)$ と $G' = (\Sigma, N', R')$ を，積構成 [CDG$^+$97] することにより得られる正規生垣文法を $G \times G'$ と書く．すなわち，

$$\begin{aligned} &G \times G' = (\Sigma, N \times N', R_1 \cup R_2) \\ &\quad \text{ただし } R_1 = \{(A, A') \to \varepsilon \mid A \to \varepsilon \in R, A' \to \varepsilon \in R'\} \\ &\qquad\quad R_2 = \{(A, A') \to \sigma((B, B'))(C, C') \mid A \to \sigma(B)C \in R, A' \to \sigma(B')C' \in R'\} \end{aligned}$$

である．このとき，任意の非終端記号 $A \in N, A' \in N'$ について，$[\![(A, A')]\!]_{G \times G'} = [\![A]\!]_G \cap [\![A']\!]_{G'}$ となる [CDG$^+$97]．ここで，打ち切り正規生垣文法 $G\varepsilon E$ と $G'\varepsilon E'$ に対し，打ち切り正規生垣文法 $(G \times G')\varepsilon(E \times E')$ を考えると，$[\![(A, A')]\!]_{(G \times G'_{)}\varepsilon(E \times E')} = [\![A]\!]_{G\varepsilon E} \cap [\![A']\!]_{G'\varepsilon E'}$ となることに注意する．また，打ち切り正規生垣文法 $(G \times G')\varepsilon(F \times E)$ が，$G = (_, _, R)$ について $F = \{A \mid A \to \varepsilon \in R\}$ である場合に，$G \times G'\varepsilon E'$ と書く．たとえば，関数 *reverse* の特化の例における型 H_1, H_2 は以下を満たす．

$$\begin{aligned} [\![H_1]\!] &= [\![(S, T)]\!]_{G \times G\varepsilon T'} \\ [\![H_2]\!] &= [\![(S^*, T)]\!]_{G \times G}. \end{aligned}$$

今後の議論を簡潔にするために，本章においては以降，型 A を，A_G（$A_{G\varepsilon E}$）と，型 A の意味を定める正規生垣文法 G（打ち切り正規生垣文法 $G\varepsilon E$）を添字して書く．たとえば，この記法により，関数 *reverse* の定義は以下と書ける．

$$\begin{aligned} &reverse(\varepsilon) &&\hat{=}\ \varepsilon \\ &reverse(a :: S_G \centerdot r :: S_G^*) &&\hat{=}\ reverse(r) \centerdot a \end{aligned}$$

ここで，G は T，S および S^* の意味を定めるのに用いた正規生垣文法である．

形式的には，パターン特化は，図 7.1 のパターン特化手続き psp により定義される．ここで，psp$(p; A_{G'\varepsilon E})$ は，型 p と打ち切り正規生垣文法 $G'\varepsilon E$ により定義される型 A を引数に取り，特化されたパターンの集合を返す．手続き psp の各規則は，以下のような振る舞いをする．

1. パターン ε に対し，psp は一つ目の規則により，$[\![A]\!]_{G'\varepsilon E}$ が ε を含む場合にパターン ε を返す．

$$
\begin{array}{lcl}
\mathsf{psp}(\varepsilon;\quad A_{G'\varepsilon E}) & \hat{=} & \{\varepsilon\} \quad \text{if } A \in E \\
\mathsf{psp}(x :: T_G;\ A_{G'\varepsilon E}) & \hat{=} & \{x :: (T, A)_{G\times G'\varepsilon E}\} \quad \text{if } [\![(T, A)]\!]_{G\times G'\varepsilon E} \neq \emptyset \\
\mathsf{psp}(\sigma(p);\quad A_{G'\varepsilon E}) & \hat{=} & \{\sigma(p') \mid p' \in \mathsf{psp}(p; B_{G'}),\ A \to \sigma(B)C \in \mathit{Rules}_{G'},\ C \in E\} \\
\mathsf{psp}(p_1 \centerdot p_2;\ A_{G'\varepsilon E}) & \hat{=} & \{p'_1 \centerdot p'_2 \mid p'_1 \in \mathsf{psp}(p_1; A_{G'\varepsilon\{B\}}),\ p'_2 \in \mathsf{psp}(p_2; B_{G'\varepsilon E}),\ B \in \mathit{NTerms}_{G'}\} \\
\mathsf{psp}(_;\quad A_{G'\varepsilon E}) & \hat{=} & \emptyset
\end{array}
$$

ここで，$G' = (_, \mathit{NTerms}_{G'}, \mathit{Rules}_{G'})$ であるとする．

図 7.1. パターン特化手続き psp

2. パターン $x :: T_G$ に対し，psp は二つ目の規則により新たに生成されたパターンに少くとも一つの生垣がマッチする場合にパターン $x :: (T, A)_{G\times G'\varepsilon E}$ を返す.

3. パターン $\sigma(p)$ に対し，psp は三つ目の規則により，$\bigcup \sigma(p') = [\![A]\!]_{G'\varepsilon E}$ を満たすパターン p' の集合を求める．このようなパターン p' の集合を求めるため, 右辺にて，G' 中のそれぞれの生成規則 $A \to \sigma(B)C$（ただし $C \in E$）に対し，$\mathsf{psp}(p; B_{G'})$ が呼ばれている．ここで，$\mathsf{psp}(p; B_{G'\varepsilon E})$ でなく $\mathsf{psp}(p; B_{G'})$ と，G' が使用されているのは，E が**水平**打ち切り非終端記号の集合を表しているためである．パターン $\sigma(p)$ 中の p は，$\sigma(p)$ とは水平位置が異なる.

4. パターン $p_1 \centerdot p_2$ に対し，psp は四つ目の規則により，$[\![p'_1]\!] \subseteq [\![p_1]\!]$ かつ $[\![p'_2]\!] \subseteq [\![p_2]\!]$ であり $\bigcup [\![p'_1 \centerdot p'_2]\!] = [\![A]\!]_{G'\varepsilon E}$ であるパターン p'_1, p'_2 の集合を求める．このようなパターン p'_1, p'_2 の集合を求めるため, 右辺にて，G' 中のそれぞれの非終端記号 B に対し，型 $A_{G'\varepsilon E}$ が B により $A_{G'\varepsilon\{B\}}$ と $B_{G'\varepsilon E}$ に分割され，$\mathsf{psp}(p_1; A_{G'\varepsilon\{B\}})$ と $\mathsf{psp}(p_2; B_{G'\varepsilon E})$ が呼ばれている.

5. 入力パターンに対し，$[\![A]\!]_{G'\varepsilon E}$ のどの元もマッチしない場合，psp の五つ目の規則が使用される．この規則は，パターン ε とパターン $x :: T_G$ に対してのみ使用されることに注意する.

いくつかの psp の使用例は，後述の例 7.1 と例 7.2 に含まれる.

定理 7.1. パターン特化 $\mathsf{psp}(p; A_{G'\varepsilon E})$ は確かにパターン p を型 $A_{G'\varepsilon N}$ に対して特化する．すなわち,

$$[\![A]\!]_{G'\varepsilon E} \cap [\![p]\!] = \bigcup \{[\![p']\!] \mid p' \in \mathsf{psp}(p; A_{G'\varepsilon E})\}$$

が成り立つ.

証明. まず，以下の主張を，パターン p に関する帰納法により示す.

$$[\![A]\!]_{G'\varepsilon E} \cap [\![p]\!] \subseteq \bigcup \{[\![p']\!] \mid p' \in \mathsf{psp}(p; A_{G'\varepsilon E})\}$$

基底：$p = \varepsilon$．

psp の定義より自明.

基底：$p = x :: T_G$．

psp の定義より自明.

帰納：$p = \sigma(p_1)$．

生垣 h を，$h \in [\![A]\!]_{G'\varepsilon E}$ かつ $h \in [\![p]\!]$ であるとする．このとき，$h = \sigma(h_1)$ であり，$h' \in [\![p_1]\!]$ である．ここで，$h \in [\![A]\!]_{G'\varepsilon E}$ より，少くとも一つの非終端記号 B について，G' は，$C \in E$ である C に対して，生成規則 $A \to \sigma(B)C$ を含み，また，$h_1 \in [\![B]\!]_{G'}$ となっている．ここで，帰納法の仮定より，

$$[\![B]\!]_{G'} \cap [\![p_1]\!] \supseteq \bigcup\{[\![p'_1]\!] \mid p'_1 \in \mathsf{psp}(p_1; B_{G'})\}$$

となる．そのため，ある $p'_1 \in \mathsf{psp}(p_1; B_{G'})$ に対し，$h_1 \in [\![p'_1]\!]$ である．よって，psp の構成より，$\sigma(p') \in \mathsf{psp}(p; A_{G'\varepsilon E})$ となり，$h \in [\![\sigma(p')]\!]$ となるため，主張は成り立つ.

帰納：$p = p_1 \centerdot p_2$．

生垣 h_1 と h_2 を，$h_1 \centerdot h_2 \in [\![A]\!]_{G'\varepsilon E}$ であり，$h_1 \in [\![p_1]\!]$ かつ $h_2 \in [\![p_2]\!]$ であるものとする．ここで，$h_1 \centerdot H_2 \in [\![A]\!]_{G'\varepsilon E}$ より，少くとも一つの非終端記号 B に対し，$A \xrightarrow{*} h_1 B$ となり $B \xrightarrow{*} h_2 C$（$C \in E$）となる．すなわち，$h_1 \in [\![A]\!]_{G'\varepsilon\{B\}}$ かつ $h_2 \in [\![B]\!]_{G'\varepsilon E}$ である．帰納法の仮定より，

$$[\![A]\!]_{G'\varepsilon\{B\}} \cap [\![p_1]\!] \subseteq \bigcup\{[\![p'_1]\!] \mid p'_1 \in \mathsf{psp}(p_1; A_{G'\varepsilon\{B\}})\}$$

かつ

$$[\![B]\!]_{G'\varepsilon E} \cap [\![p_2]\!] \subseteq \bigcup\{[\![p'_2]\!] \mid p'_2 \in \mathsf{psp}(p_2; A_{G'\varepsilon E})\}$$

となる．よって，psp の定義より，パターン $p'_1 \in \mathsf{psp}(p_1; A_{G'\varepsilon\{B\}})$ と $p'_2 \in \mathsf{psp}(p_2; A_{G'\varepsilon E})$ が存在し，$h_1 \in [\![p'_1]\!], h_2 \in [\![p'_2]\!]$ となる．そのため，psp の定義から

$$p'_1 \centerdot p'_2 \in \mathsf{psp}(p_1 \centerdot p_2; A_{G'\varepsilon E}),$$

が言える．これより，主張は真.

次に，以下の主張を，パターン p に関する帰納法により示す.

$$[\![A]\!]_{G'\varepsilon E} \cap [\![p]\!] \supseteq \bigcup\{[\![p']\!] \mid p' \in \mathsf{psp}(p; A_{G'\varepsilon E})\}$$

基底：$p = \varepsilon$.

psp の定義より自明.

基底：$p = x :: T_G$.

psp の定義より自明.

帰納：$p = \sigma(p_1)$.

生垣 h を，$h \in \bigcup\{[\![p']\!] \mid p' \in \mathsf{psp}(p; A_{G'\varepsilon E})\}$ であるものとする．つまり，h は，あるパターン $p' \in \mathsf{psp}(p; A_{G'\varepsilon E})$ について，$h \in [\![p']\!]$ となる．ここで，$h = \sigma(h_1)$ と書けることに注意する．このとき，psp の定義より，非終端記号 B と $C \in E$ が存在して，G' に規則 $A \to \sigma(B)C$ が存在し，$p'_1 \in \mathsf{psp}(p_1; B_{G'})$ について，$h_1 \in [\![p'_1]\!]$ となる．帰納法の仮定より，

$$[\![B]\!]_{G'} \cap [\![p_1]\!] \supseteq \bigcup\{[\![p'_1]\!] \mid p'_1 \in \mathsf{psp}(p_1; B_{G'})\}$$

となる．よって，$h_1 \in [\![B]\!]_{G'}$ かつ $h_1 \in [\![p_1]\!]$ となる．正規生垣文法 G' が規則 $A \to \sigma(B)C$ を持つことと $C \in E$ から $h \in [\![A]\!]_{G'\varepsilon E}$ である．また，$h_1 \in [\![p_1]\!]$ より $h \in [\![p]\!]$ である．よって主張は真.

帰納：$p = p_1 \centerdot p_2$.

生垣 h_1 と h_2 を，$p'_1 \centerdot p'_2 \in \mathsf{psp}(p_1 \centerdot p_2; A_{G'\varepsilon E})$ となるあるパターン p'_1, p'_2 に対し，$h_1 \in [\![p'_1]\!]$ かつ $h_2 \in [\![p'_2]\!]$ となるものとする．このとき psp の定義より，$p'_1 \in \mathsf{psp}(p_1; A_{G'\varepsilon\{B\}})$ かつ $p'_2 \in \mathsf{psp}(p_2; B_{G'\varepsilon E})$ となる．帰納法の仮定より，

$$[\![A]\!]_{G'\varepsilon\{B\}} \cap [\![p_1]\!] \supseteq \bigcup\{[\![p'_1]\!] \mid p'_1 \in \mathsf{psp}(p_1; A_{G'\varepsilon\{B\}})\}$$

かつ

$$[\![B]\!]_{G'\varepsilon E} \cap [\![p_2]\!] \supseteq \bigcup\{[\![p'_2]\!] \mid p'_2 \in \mathsf{psp}(p_2; A_{G'\varepsilon E})\}.$$

となる．よって，$h_1 \in [\![A]\!]_{G'\varepsilon\{B\}} \cap [\![p_1]\!]$ かつ $h_1 \in [\![B]\!]_{G'\varepsilon E} \cap [\![p_2]\!]$ が成り立つ．連接 $\centerdot$ の定義より，$h_1 \centerdot h_2 \in [\![p_1 \centerdot p_2]\!]$ となる．ここで打ち切り正規生垣文法の定義から，$h_1 \centerdot h_2 \in [\![A]\!]_{G'\varepsilon E}$ となる．よって，$h_1 \centerdot h_2 \in [\![A]\!]_{G'\varepsilon E} \cap [\![p_1 \centerdot p_2]\!]$ が成り立つため，主張は真. □

7.3.2　特化のアルゴリズム

特化アルゴリズムにおいて，第 4 章に定義を示した**曖昧性**の概念と**既約**の概念とを用いる．正規生垣文法 $G = (\Sigma, N, R)$ が曖昧でないとは

$$\forall A, B \in N.\ A \neq B \Rightarrow [\![A_G]\!] \cap [\![B_G]\!] = \emptyset \tag{UnAmb}$$

ということであった．また，正規生垣文法 $G = (\Sigma, N, R)$ が既約であるとは

$$\forall A \in N.\ [\![A]\!] \neq \emptyset$$

ということであった．

以下に引数の型に基づく関数の特化アルゴリズムを以下に示す．

アルゴリズム 7.1 (引数の型に基づく関数の特化：Alg-Sp)**.**

入力： プログラム．

出力： 型に基づき特化されたプログラム．

手続き：

1. プログラム中の全ての規則 $h(\overrightarrow{q}) \doteq \mathcal{C}[f(\overrightarrow{x})]$ の全ての関数呼び出し $f(\overrightarrow{x})$ について，$\Gamma_{\overrightarrow{q}}(x_i) = A_{i G'_i \varepsilon E_i}$（$x_i \in \{\overrightarrow{x_i}\}$）とし，ステップ 2–5 を繰り返す．
2. それぞれの，$G'_i \varepsilon E_i$ に対し，$G' \varepsilon E$ に対応する曖昧でなく既約な正規生垣文法 G''_i を作成する．ここで，G''_i 中の非終端記号 $A''_{i1}, \ldots, A''_{in_i}$ について，$\bigcup_{j \in \{1 \ldots n_i\}} [\![A''_{ij}]\!] = [\![A_i]\!]_{G_i \varepsilon E'_i}$ となっている．
3. 元プログラムにおけるそれぞれの f の規則 $f(\overrightarrow{p}) \doteq e$ について，ステップ 3–5 を繰り返す．ここで，$\overrightarrow{p} = (p_1, \ldots, p_m)$ とする．
4. それぞれの特化されたパターン
$$(p'_1, \ldots, p'_m) \in \bigcup_{j \in \{1 \ldots n_1\}} \mathsf{psp}(p_1; A''_{1j\,G''_1}) \times \cdots \times \bigcup_{j \in \{1 \ldots n_m\}} \mathsf{psp}(p_m; A''_{mj\,G''_m})$$
に対し，規則 $f|_{A_{1\,G'_1 \varepsilon E_1}, \ldots, A_{m\,G'_m \varepsilon E_m}}(\overrightarrow{p'}) \doteq e'$ を生成する．ここで，e' は e のうち関数呼出 $g(\overrightarrow{y})$ を $g|_{\overrightarrow{T}}(\overrightarrow{y})$ で置き換えたものである．ただし，$\overrightarrow{T} = \Gamma_{\overrightarrow{p'}}(\overrightarrow{y})$ とする．
5. 本アルゴリズムを全ての新たに生成された関数呼出式に対し適用する．なお，既に $g|_{\overrightarrow{S}}$ で $\overrightarrow{[\![S]\!]} = \overrightarrow{[\![T]\!]}$ を満たす関数が生成されている場合，$g|_{\overrightarrow{S}}$ と $g|_{\overrightarrow{T}}$ は本質的には同じ関数であるため，新たに $g|_{\overrightarrow{T}}$ の規則を生成することはしない．　□

ステップ 3 において，曖昧でない正規生垣文法を作成することにより，特化されたプログラムが決定的であることを保証する．アルゴリズム Alg-Sp の挙動を，本節の冒頭の *reverse* と *idAB* の例を用いて，順を追って説明する．

例 7.1 ($reverse$)**.** 以下のプログラムを考える．

$$main(x :: T) \doteq reverse(x)$$

ここで，$T = (\texttt{<a>} \cdot \texttt{<b>})^*$ である．G を，$reverse$ のプログラムにおける型 T，S と S^* の意味を定義する正規生垣文法であるとする．

1. ステップ 1 に従い，関数 $reverse$ を型 T_G に対して特化することを考える．

2. ステップ 2 に従い，G から，T の意味を定義するための曖昧でない正規生垣文法 G' を作成する．これは，T に関連する生成規則のみを G から集めてくるだけでよい．

3. ステップ 3 に従い，$reverse$ の二つの規則のうち，規則 $reverse(\varepsilon) \doteq \varepsilon$ を型 $T_{G'}$ に対して特化することを考える．

4. ステップ 4 に従い，パターン特化の結果 $\mathsf{psp}(\varepsilon; T_{G'}) = \{\varepsilon\}$ により，規則

$$reverse|_{T_{G'}}(\varepsilon) \doteq \varepsilon$$

が生成される．

5. ステップ 5 では，ステップ 4 で生成された規則は右辺に関数呼出を含まないため何も行わない．そのため，ステップ 3 に戻り，$reverse$ の残った規則を特化する．

6. ステップ 3 に従い，規則 $reverse(a :: S_G \cdot r :: S_G^*) \doteq reverse(r) \cdot a$ を型 $T_{G'}$ に対して特化することを考える．

7. ステップ 4 により，パターン特化の結果

$$\begin{aligned} &\mathsf{psp}(a :: S_G \cdot r :: S_G^*; T_{G'}) \\ &\quad = \{a :: (S,T)_{G \times G' \varepsilon T'} \cdot r :: (S^*, T')_{G \times G'}\}, \end{aligned}$$

より，規則

$$\begin{aligned} &reverse|_{T_{G'}}(a :: (S,T)_{G \times G' \varepsilon T'} \cdot r :: (S^*, T')_{G \times G'}) \\ &\quad \doteq reverse|_{(S^*,T')_{G \times G'}}(r) \cdot a \end{aligned}$$

が生成される．

8. ステップ 5 に従い，ステップ 4 で生成された規則が関数呼出 $reverse|_{(S^*,T')_{G \times G'}}(r)$ を含むため，関数 $reverse$ を型 $(S^*, T')_{G \times G'}$ に対し特化することを考える．

9. ステップ 2 に従い，次の規則により構成される曖昧でない正規生垣文法 G'' を作成する．

$$U \to \texttt{<b>}() \cdot V \qquad V \to \texttt{<a>}() \cdot U \qquad V \to \varepsilon$$

ここで，$[\![U]\!]_{G''} = [\![(S^*, T')]\!]_{G \times G'}$ である．

10. ステップ 3 に従い，*reverse* の二つの規則のうち，規則 $reverse(\varepsilon) \mathrel{\hat{=}} \varepsilon$ を型 $U_{G''}$ に対し特化することを考える．

11. ステップ 4 に従い，パターン特化の結果 $\mathsf{psp}(\varepsilon; U_{G''}) = \emptyset$ より，何の規則も生成しない．よって，ステップ 3 に戻り，*reverse* の残った規則を処理する．

12. ステップ 3 に従い，規則 $reverse(a :: S_G \centerdot r :: S_G^*) \mathrel{\hat{=}} reverse(r) \centerdot a$ を型 $U_{G''}$ に対し特化することを考える．

13. ステップ 4 に従い，パターン特化の結果

$$\begin{aligned} &\mathsf{psp}(a :: S_G \centerdot r :: S_G^*; U_{G''}) \\ &\quad = \{a :: (S, U)_{G \times G'' \varepsilon V} \centerdot r :: (S^*, V)_{G \times G''}\} \end{aligned}$$

より，規則

$$\begin{aligned} &reverse|_{(S^*,T')_{G \times G'}}(a :: (S, U)_{G \times G'' \varepsilon V} \centerdot r :: (S^*, V)_{G \times G''}) \\ &\quad \mathrel{\hat{=}} reverse|_{(S^*,V)_{G \times G''}}(r) \centerdot a \end{aligned}$$

が生成される．ここまでにアルゴリズム Alg-Sp が生成した各型について，以下の等式が成り立つことに注意する．

$$\begin{aligned} [\![(S, T)_{G \times G' \varepsilon T'}]\!] &= \{\texttt{<a>}\} \\ [\![(S^*, T')_{G \times G'}]\!] &= \{\texttt{<b>} \centerdot h \mid h \in [\![T]\!]_G\} \\ [\![(S, U)_{G \times G'' \varepsilon V}]\!] &= \{\texttt{<b>}\} \\ [\![(S^*, V)_{G \times G''}]\!] &= [\![T]\!]_G \end{aligned}$$

14. ステップ 5 において，何の特化も行おうとすることなく，アルゴリズム Alg-Sp は完了する．これは，ステップ 4 で生成された規則の右辺式には関数呼出 $reverse|_{(S^*,V)_{G \times G''}}(r)$ が含まれているが，$[\![(S^*, V)_{G \times G''}]\!] = [\![T]\!]_G$ より関数 $reverse|_{(S^*,V)_{G \times G''}}$ は $reverse|_{T_{G'}}$ に他ならず，すでに関数の規則は生成されているためである．

生成された規則を集めることにより，以下のプログラムを得る．

$$\begin{aligned} &\mathbf{data}\ T \mathrel{\hat{=}} (\texttt{<a>} \centerdot \texttt{<b>})^* \\ &\mathbf{data}\ U \mathrel{\hat{=}} \texttt{<b>} \centerdot V \\ &\mathbf{data}\ V \mathrel{\hat{=}} \texttt{<a>} \centerdot U \mid \varepsilon \\ &reverse|_T(\varepsilon) && \mathrel{\hat{=}} \varepsilon \\ &reverse|TS(a :: \texttt{<a>} \centerdot r :: U) && \mathrel{\hat{=}} reverse|_U(r) \centerdot a \\ &reverse|_U(a :: \texttt{<b>} \centerdot r :: T) && \mathrel{\hat{=}} reverse|_T(r) \centerdot a \end{aligned}$$

ここで $[\![U]\!] = [\![\texttt{<b>} \centerdot T]\!]$ であることに注意してほしい．

例 7.2 ($idAB$)**.** 以下のプログラムを考える．

$$main(x :: T_G) \mathrel{\hat{=}} idAB(x)$$

ここで $idAB$ は次のように定義され，

$$idAB(x :: S_G \cdot y :: S_G) \mathrel{\hat{=}} x \cdot y$$

ここで，正規生垣文法 G は以下の生成規則により構成される．

$$\begin{array}{lllll} S \to \texttt{<a>}S' & S \to \texttt{<b>}S' & S' \to \varepsilon & & \\ T \to \texttt{<a>}U & T \to \texttt{<b>}V & U \to \texttt{<b>}W & V \to \texttt{<a>}W & W \to \varepsilon \end{array}$$

1. ステップ 1 に従い，関数 $idAB$ を型 T_G に特化することを考える．

2. ステップ 2 に従い，正規生垣文法 G より，T の意味を定義するための，G に対応した曖昧でない正規生垣文法 G' を構成する．この場合においては，この G' の構成は，G から T に関連する生成規則を集めるだけで行える．

3. ステップ 3 に従い，$idAB$ の唯一の規則，$idAB(x :: S_G \cdot y :: S_G) \mathrel{\hat{=}} x \cdot y$ を型 $T_{G'}$ に対し特化することを考える．

4. ステップ 4 に従い，パターン特化の結果

$$\begin{aligned} &\mathsf{psp}(x :: S_G \cdot y :: S_G; T_{G'}) \\ &\quad = \left\{ \begin{array}{l} x :: (S,T)_{G \times G' \varepsilon V} \cdot y :: (S,V)_{G \times G'}, \\ x :: (S,T)_{G \times G' \varepsilon U} \cdot y :: (S,U)_{G \times G'} \end{array} \right\}, \end{aligned}$$

   より，二つの規則

$$\begin{array}{l} idAB|_T(x :: (S,T)_{G \times G' \varepsilon V} \cdot y :: (S,V)_{G \times G'}) \mathrel{\hat{=}} x \cdot y \\ idAB|_T(x :: (S,T)_{G \times G' \varepsilon U} \cdot y :: (S,U)_{G \times G'}) \mathrel{\hat{=}} x \cdot y \end{array}$$

   が生成される．

5. ステップ 5 において，Alg-Sp は完了する．なぜなら，ステップ 4 で生成された規則は右辺に関数呼出を含まないためである．

生成された規則を集め，パターンを読み易く書き直すと，以下のプログラムが得られる．

$$\begin{array}{l} idAB|_T(x :: \texttt{<b>} \cdot y :: \texttt{<a>}) \mathrel{\hat{=}} x \cdot y \\ idAB|_T(x :: \texttt{<a>} \cdot y :: \texttt{<b>}) \mathrel{\hat{=}} x \cdot y \end{array}$$

7.3.3 提案する特化手法の性質

ここでは，我々の提案する特化手法である Alg-Sp に関するいくつかの性質を示す．

まず，我々は Alg-Sp の停止性を示す．停止性を示すのにあたり，打ち切り正規生垣文法が重要な役割を果たす．打ち切り正規生垣文法を導入したことにより，我々はパターン特化において，新たな正規生垣文法により表現される新たな型を導入する必要がない．つまり，

パターン特化により現れる型は既存の（打ち切り）正規生垣文法の組み合わせで記述されるものである．

停止性の証明の前に，我々はいくつかの補題を示す．補題 7.1 は，psp が型 $A_{G''\varepsilon E''}$ を導入した場合に，もし psp の入力の型が既存の型を組み合せることにより表現できている場合，G'' の全ての非終端記号の意味も既存の型を組み合せることにより表現できることを示す．補題 7.2 は，もし，$G''\varepsilon E''$ において，G'' の全ての非終端記号の意味が既存の型を組み合せることにより表現できている場合に，$G''\varepsilon E''$ の全ての非終端記号の意味も既存の型の組み合わせにより表現できることを示す．補題 7.3 や 7.4 は，正規生垣文法に対し,「正規生垣文法中の非終端記号の意味が依存の型の組み合わせで表現できている」という性質を変えることなく，打ち切り正規生垣文法から正規生垣文法に変換することと，曖昧な正規生垣文法から曖昧でなく既約な正規生垣文法に変換することができることを示す.

補題の証明において，我々は曖昧でない既約な正規生垣文法 G に対し，$\mathcal{D}_{\mathsf{G}}(x)$，$\mathcal{D}'_{\mathsf{G}}(x)$ といった記法により，$\mathsf{p}_{A/B}(x)$ という形式の原子論理式の上の選言標準形の論理式を表す．ここで，原子論理式 $\mathsf{p}_{A/B}(x)$ は，

$$\mathsf{p}_{A/B}(x) \Leftrightarrow \exists y.\ (x \mathbin{\centerdot} y) \in [\![A]\!]_{\mathsf{G}} \wedge y \in [\![B]\!]_{\mathsf{G}}$$

を表す．このとき，G が既約かつ曖昧でないため，

$$\mathsf{p}_{A/B}(x) \Leftrightarrow A \xrightarrow{*} xB$$

となることに注意する．また，G に対し，$\mathsf{p}_{A/B}$ の形の原子論理式の数は有限であるため，その上の選言標準形の論理式の数も有限であることにも注意する.

補題 7.1. $\mathcal{P} = (G, _, _)$ をプログラム，$G = (_, N, R)$ を正規生垣文法，q をプログラム中に現れるパターンであるとする．このとき，$G' = (_, N', _)$ に対し $\mathsf{psp}(q; A'_{G'})$ は以下の性質を満たす.

$$\begin{array}{l}
\forall T \in N, \exists \mathcal{D}_{\mathsf{G}}.\ (\forall x.\ x \in [\![T']\!]_G \equiv \mathcal{D}_{\mathsf{G}}(x)) \\
\wedge \forall T' \in N', \exists \mathcal{D}'_{\mathsf{G}}.\ (\forall x.\ x \in [\![T']\!]_{G'} \equiv \mathcal{D}'_{\mathsf{G}}(x)) \\
\Rightarrow \begin{pmatrix}
\forall q' \in \mathsf{psp}(q; A'_{G'}), \\
\forall x \in \mathsf{vars}(q'). \\
\text{let } (_, G''\varepsilon E'') = \Gamma_{q'}(x) \text{ with } G'' = (_, N'', _), \\
\forall T'' \in N'', \exists \mathcal{D}''_{\mathsf{G}}.\ (\forall x.\ x \in [\![T'']\!]_{G''} \equiv \mathcal{D}''_{\mathsf{G}}(x))
\end{pmatrix}
\end{array}$$

証明. 手続き psp の定義により，$q' \in \mathsf{psp}(q; A'_{G'})$ 中の $x \in \mathsf{vars}(q')$ に対し，$B_{G''\varepsilon E''} = \Gamma_{q'}(x)$ と書くと，$G'' = (_, N'', _)$ と E'' は，ある $E' \subseteq N'$ について，以下を満たす.

$$G''\varepsilon E'' = G \times G'\varepsilon E' = (G \times G')\varepsilon(F \times E')$$

ここで，$F = \{A \mid A \to \varepsilon \in R\}$ である．よって，$(T, T') \in N''$ ならば $[\![(T, T')]\!]_{G''} = [\![T]\!]_G \cap [\![T']\!]_{G'}$ となる．補題の前提条件より，$x \in [\![T]\!]_G \equiv \mathcal{D}_{\mathsf{G}}(x)$ かつ $x \in [\![T']\!]_{G'} \equiv \mathcal{D}'_{\mathsf{G}}(x)$ となる.

よって，ある $\mathcal{D}''_{\mathsf{G}}(x)$ が存在して，$x \in [\![(T, T')]\!]_{G''})$ である x について以下が成り立つ．

$$
\begin{aligned}
x \in [\![(T, T')]\!]_{G''} &\equiv x \in ([\![T]\!]_G \cap [\![T']\!]_{G'}) \\
&\equiv \mathcal{D}_{\mathsf{G}}(x) \wedge \mathcal{D}'_{\mathsf{G}}(x) \\
&\equiv \mathcal{D}''_{\mathsf{G}}(x)
\end{aligned}
$$

よって，主張は真． □

補題 7.2. 既約な曖昧でない正規生垣文法 $G = (\Sigma, N, R)$ に対し，もし任意の $T \in N$ がある $\mathcal{D}_{\mathsf{G}}(x)$ により $x \in T \equiv \mathcal{D}_{\mathsf{G}}(x)$ と書けるならば，このとき任意の $E \subseteq N$ および任意の $T \in G\varepsilon E$ について，ある $\mathcal{D}'{}_{\mathsf{G}}(x)$ が存在して，$x \in T \equiv \mathcal{D}'_{\mathsf{G}}(x)$ となる．

証明. 正規生垣文法 G が既約かつ曖昧でないため打ち切り正規生垣文法 $G\varepsilon E$ 中の非終端記号 T の意味は，以下の形式で書ける．

$$
\begin{aligned}
[\![T]\!]_{G\varepsilon E} &= \left\{ x \;\middle|\; \bigvee_{S \in E} \exists y.\, T \xrightarrow{*} xS \wedge S \xrightarrow{*} y \right\} \\
&= \left\{ x \;\middle|\; \bigvee_{S \in E} \exists y.\, (x \centerdot y) \in [\![T]\!]_G, y \in [\![S]\!]_G \right\}
\end{aligned}
$$

ここで，G 中の非終端記号 A の意味を表す選言標準形の論理式を $\mathcal{D}^A_{\mathsf{G}}(x)$ と書いたとすると，上の論理式は以下の通りに書き直せる．

$$
[\![T]\!]_{G\varepsilon E} = \left\{ x \;\middle|\; \bigvee_{S \in E} \exists y.\, \mathcal{D}^T_{\mathsf{G}}(x \centerdot y) \wedge \mathcal{D}^S_{\mathsf{G}}(y) \right\}
$$

ここで，$\mathsf{p}_{A/B}(x \centerdot y) = \bigvee_{C \in \mathsf{N}} \mathsf{p}_{A/C}(x) \wedge \mathsf{p}_{C/B}(y)$（ただし，$\mathsf{G} = (_, \mathsf{N}, _)$）であり，$\mathcal{F}_i(z)$ と $\mathcal{F}'_i(z)$（$i \in \{1, \ldots, n\}$）を $\wedge$ と $\neg$ しか含まない，$\mathsf{p}_{A/B}(z)$ という形式の原子論理式の上の論理式であるとする．すると，$\exists$ の分配則（$\exists x.\, p(x) \vee p'(x) \equiv (\exists x.p(x)) \vee (\exists x.p'(x))$）を用いることにより，上の論理式は以下のように整理できる．

$$
[\![T]\!]_{G\varepsilon E} = \left\{ x \;\middle|\; \bigvee_{i} \exists y.\, \mathcal{F}_i(x) \wedge \mathcal{F}'_i(y) \right\}
$$

ここで，論理式 $\mathcal{F}_i(x)$ は存在量化 $\exists y$ により束縛される変数 y を含まないため，上の論理式は以下のように書ける．

$$
[\![T]\!]_{G\varepsilon E} = \left\{ x \;\middle|\; \bigvee_{i} \mathcal{F}_i(x) \wedge \exists y.\, \mathcal{F}'_i(y) \right\}
$$

ここで，$\exists y.\, \mathcal{F}'_i(y)$ は x とは独立に真偽が判定できるため，取り除ける．よって，ある $\mathcal{D}'_{\mathsf{G}}(x)$ が存在して，

$$
[\![T]\!]_{G\varepsilon E} \equiv \mathcal{D}'_{\mathsf{G}}(x)
$$

と書けることが言える．これにより，補題は示された． □

補題 7.3. 次の性質を満たすように，打ち切り正規生垣文法 $G\varepsilon E$（$G = (_, N, _)$）から，正規生垣文法 $G' = (_, N', _)$ を構成できる

$$\forall T \in N, \exists T' \in N'.\ [\![T]\!]_{G\varepsilon E} = [\![T']\!]_{G'} \\ \wedge \begin{pmatrix} \forall T \in N, \exists \mathcal{D}_{\mathsf{G}}.\ x \in [\![T]\!]_{G\varepsilon E} \equiv \mathcal{D}_{\mathsf{G}}(x) \\ \quad \Rightarrow \forall T' \in N', \exists \mathcal{D}'_{\mathsf{G}}.\ x \in [\![T']\!]_{G'} \equiv \mathcal{D}_{\mathsf{G}}(x). \end{pmatrix}$$

証明. 7.3.1 節に示した手法により打ち切り正規生垣文法から，ただ正規生垣文法を構成するだけでよい． □

補題 7.4. 次の性質を満たすように，正規生垣文法 $G = (_, N, _)$ から曖昧でない既約な正規生垣文法 $G' = (_, N', _)$ を構成できる．

$$\forall T \in N, \exists \mathcal{D}_{\mathsf{G}}.\ x \in [\![T]\!]_G \equiv \mathcal{D}_{\mathsf{G}}(x) \\ \quad \Rightarrow \forall T' \in N', \exists \mathcal{D}'_{\mathsf{G}}.\ x \in [\![T']\!]_{G'} \equiv \mathcal{D}'_{\mathsf{G}}(x).$$

証明. 正規生垣文法から曖昧でない既約な正規生垣文法を構成するためには，非決定的有限上昇木オートマトンから決定的上昇木オートマトンに変換するための変換である部分集合構成 [CDG+97] を適用すればよい．なお，部分集合構成によって得られた正規生垣文法に，非終端記号 $\{A\}$ と $\{A, B\}$ が含まれる場合，$\{A\}$ の意味は，$[\![\{A\}]\!] = [\![A]\!]$ ではなく，$[\![\{A\}]\!] = [\![A]\!] \wedge [\![B]\!]^c$ となることに注意する．また，オートマトンの最小化 [CDG+97] も適用することができる．最小化の後，得られる非終端記号 $\{A, B\}$ の意味は，$[\![\{A, B\}]\!] = [\![A]\!] \cup [\![B]\!]$ であることに注意する．

これらの変換により，これまでの補題に議論した選言標準形に $\vee$ と $\neg$ がもたらされることに注意する． □

定理 7.2. アルゴリズム Alg-Sp は停止する．

証明. アルゴリズム Alg-Sp に出現する正規生垣「言語」の数が有限であることにより，証明する．もし，アルゴリズム Alg-Sp に出現する正規生垣言語が有限（k 個とする）ならば，Alg-Sp が生成する関数の数の上限は，プログラムに含まれる関数名の集合を Q とし，M を関数の引数の個数の最大数であるとすると，高々$|Q|k^M$ 個であるため，Alg-Sp は止まる．

$\mathcal{P} = (G, _)$ プログラムであるとする．ここで，$G = (_, _, R)$ である．このとき，G から，既約で曖昧でない正規生垣文法 G を構成する．このとき，G 中の非終端記号 T に対し，G 中の非終端記号 $A_{T,1}, \ldots, A_{T,n_T}$ が存在して

$$[\![A]\!]_G = [\![A_{T,1}]\!]_{\mathsf{G}} \cup \cdots \cup [\![A_{T,n_T}]\!]_{\mathsf{G}}$$

となるので，

$$x \in [\![T]\!]_G \equiv \bigvee_{i \in \{1,\ldots,n\}, C \in F} \mathsf{p}_{A_{T,i}/C}(x) = \mathcal{D}'_{\mathsf{G}}(x)$$

となる．ただし，非終端記号の集合 F は，$\mathsf{G} = (_, _, \mathsf{R})$ として，$F = \{X \mid X \to \varepsilon \in \mathsf{R}\}$ により定義される集合である．

ここで，補題 7.1，補題 7.2，補題 7.3 および補題 7.4 より，psp の適用回数による帰納法から，Alg-Sp 中に現れる全ての型 T について，ある $\mathcal{D}_{\mathsf{G}}(x)$ が存在し $\forall x \in [\![T]\!] \equiv \mathcal{D}_{\mathsf{G}}(x)$ となることが示される．このような選言標準形の論理式の数は有限であるため，Alg-Sp 中に現れる型の意味は（正規生垣言語）は有限である．

また，Alg-Sp において，我々は正規生垣言語同士の等価性が決定可能であるという事実を用いている [CDG+97]. □

上の停止性の証明は同時に Alg-Sp によるプログラムサイズの増加の上界を与えている．プログラム $\mathcal{P} = (G, _, _)$ の Alg-Sp の適用によりサイズは，n を G 中の非終端記号の数とし M を関数の引数の数の最大値とすると，高々$\mathrm{O}((2^{2^{(2^n)^2}})^M)$ 倍にしかならない．これは，m 個の原子論理式の数の上の選言標準形の論理式の数は $\mathrm{O}(2^{2^m})$ であるためである．また，G を G に部分集合構成を適用することにより得ると，G 中の非終端記号の数は最悪 $\mathrm{O}(2^n)$ 個になりうることに注意する．よって，$\mathsf{p}_{A/B}$ の形の原子論理式の数は $m = (2^n)^2$ 個となる．

我々の停止性の証明は過度にまわりくどいものであるように見えるかもしれない．しかし，我々が停止性をこれまで述べたようなやり方で証明したのは，型の内部表現，つまり正規生垣文法，の選択に柔軟性を持たせたかったためである．たとえば，我々は，Alg-Sp のステップ 2 おいて得られた曖昧でない正規生垣文法を最小化 [CDG+97] することもできるし，また psp の返したパターンを意味の変らない範囲で併合することもできる．これらの変換は，Alg-Sp の実行効率に影響を与える．なぜなら，Alg-Sp はステップ 5 において正規生垣言語同士の等価性の判定を含んでいるが，これらの変換は等価性の判定が行われる回数を減らすためである．正規生垣言語同士の等価性の判定は決定可能ではあるが，EXPTIME 完全な問題でもある [CDG+97]．また，パターンの併合は双方向化において重要な役割を果たす．これについては後述する．

次に，Alg-Sp が生成するプログラムが決定的であることを示す．ここで，言語 Vdl+ に対する仮定より，Alg-Sp の入力となるプログラムは決定的であるため，Alg-Sp により一つの関数定義規則より複数の関数定義規則のみを考えればよい．なぜなら，定理 7.1 より，これ以外ではプログラム非決定性は生じえないためである．よって，Alg-Sp 適用後のプログラムが決定的であることを保証するには，パターン p と既約な曖昧でない正規生垣文法 G'' 中の非終端記号 A'' について

$$p', p'' \in \mathsf{psp}(p; A''_{G''}) \Rightarrow [\![p']\!] \cap [\![p'']\!] = \emptyset \qquad \text{(D-PSP)}$$

を満たすことを保証できれば十分である．

補題 7.5. 既約な曖昧でない G'' 中の非終端記号 A'' と，パターン p について式 (D-PSP) が成り立つ．

証明. $G'' = (_, N'', R'')$ とする．まず，次を示す．

$$\begin{aligned}&(\forall A'', B''.\ A'' \neq B'' \Rightarrow [\![A'']\!]_{G''} \cap [\![B'']\!]_{G''} = \emptyset)\\ &\Rightarrow \forall C'' \in N'', (\forall A'', B''.\ A'' \neq B'' \Rightarrow [\![A'']\!]_{G''\varepsilon\{C''\}} \cap [\![B'']\!]_{G''\varepsilon\{C''\}} = \emptyset)\end{aligned}$$

もし，ある h について $h \in [\![A'']\!]_{G''\varepsilon\{C''\}} \cap [\![B'']\!]_{G''\varepsilon\{C''\}}$ であったとすると，G'' が既約なことから $h' \in [\![C'']\!]_{G''}$ となる h' が存在するので，$h \centerdot h' \in [\![A'']\!]_{G''}$ かつ $h \centerdot h' \in [\![B'']\!]_{G''}$ となる．これは，前提条件に矛盾する．

既約な曖昧でない G と $\mathsf{psp}(p; A_G)$ について，$\mathsf{psp}(p; A_G)$ の実行途中に呼び出される psp の再帰呼出について，第二引数は，B_G もしは $B_{G\varepsilon C}$ の形（$B, C \in N$）であることに注意する．上の主張より，これらの G および $G\varepsilon C$ に含まれる非終端記号 A の意味は，A 以外の非終端記号の意味と互いに素である．

これより，式 (D-PSP) が成り立つことをパターン p に対する帰納法により示す．前述の議論により，$\mathsf{psp}(p; A_{G''\varepsilon E''})$ の呼出において，

$$\forall A'', B''.\ A'' \neq B'' \Rightarrow [\![A'']\!]_{G''\varepsilon E''} \cap [\![B'']\!]_{G''\varepsilon E''} = \emptyset \tag{*}$$

であることを仮定してよい．帰納法の基底（$p = \varepsilon, p = x :: T_G$）について，psp は零個または一個のパターンしか返さないため自明である．

帰納：$p = \sigma(p_1)$**.**

このとき

$$\mathsf{psp}(\sigma(p_1); A_{G''\varepsilon E}) = \{\sigma(p_1') \mid p_1' \in \mathsf{psp}(p_1; B_{G''}), A \to \sigma(B)C \in R'', C \in E\}$$

である．ところで，式 (*) より

$$\forall B, D \in N'', D \neq B \Rightarrow \forall q \in \mathsf{psp}(p_1; B_{G''}), q' \in \mathsf{psp}(p_1; D_{G''}).\ [\![q]\!] \cap [\![q']\!] = \emptyset$$

となる．また，帰納法の仮定より，

$$\forall B \in N'', \forall q, q' \in \mathsf{psp}(p_1; B_{G''}) \Rightarrow [\![q]\!] \cap [\![q']\!] = \emptyset$$

となる．よって，

$$\forall q, q' \in \mathsf{psp}(\sigma(p_1); A_{G\varepsilon E}) \Rightarrow [\![q]\!] \cap [\![q']\!] = \emptyset$$

となる．

帰納：$p = p_1 \mathbin{.} p_2$．

このとき

$$\begin{aligned}&\mathsf{psp}(p_1 \mathbin{.} p_2; A_{G''\varepsilon E})\\&\quad = \{p'_1 \mathbin{.} p'_2 \mid p'_1 \in \mathsf{psp}(p_1; A_{G''\varepsilon B}), p'_2 \in \mathsf{psp}(p_1; B_{G''\varepsilon E''}), B \in N''\}\end{aligned}$$

である．ここで，$q, q' \in \mathsf{psp}(p_1 \mathbin{.} p_2; A_{G''\varepsilon E''})$ であるパターン $q = q_1 \mathbin{.} q_2, q' = q'_1 \mathbin{.} q'_2$ について次の二つ場合がありうる．

- ある B について，$q_1, q'_1 \in \mathsf{psp}(p_1; A_{G''\varepsilon B})$ かつ $q_2, q'_2 \in \mathsf{psp}(p_1; B_{G''\varepsilon E''})$ である．
- ある B と C（$B \neq C$）について，$q_1 \in \mathsf{psp}(p_1; A_{G''\varepsilon B})$ かつ $q_2 \in \mathsf{psp}(p_1; B_{G''\varepsilon E''})$ であり，$q'_1 \in \mathsf{psp}(p_1; A_{G''\varepsilon C})$ かつ $q'_2 \in \mathsf{psp}(p_1; C_{G''\varepsilon E''})$ である．

まず前者について考える．帰納法の仮定より，$[\![q_1]\!] \cap [\![q'_1]\!] = \emptyset$ かつ $[\![q_2]\!] \cap [\![q'_2]\!] = \emptyset$ である．仮に，$v \in [\![q_1 \mathbin{.} q_2]\!]$ かつ $v \in [\![q'_1 \mathbin{.} q'_2]\!]$ となる v が存在したとする．このとき，$[\![q_1]\!] \subseteq [\![p_1]\!]$ かつ $[\![q'_1]\!] \subseteq [\![p_1]\!]$ となり，$[\![q_2]\!] \subseteq [\![p_2]\!]$ かつ $[\![q'_2]\!] \subseteq [\![p_2]\!]$ となる．ここで $v \in [\![p_1 \mathbin{.} p_2]\!]$ であるから，言語 $\text{V}\textsc{dl}^+$ におけるパターンマッチの一意性より $v = v_1 \mathbin{.} v_2$ と一意に分解できる．すると $v_2 \in [\![q_2]\!]$ かつ $v_2 \in [\![q_2]\!]$ となり矛盾．よって，$[\![q_1 \mathbin{.} q_2]\!] \cap [\![q'_1 \mathbin{.} q'_2]\!] = \emptyset$ である．

次に後者について考える．式 (*) より $[\![q_2]\!] \cap [\![q'_2]\!] = \emptyset$ である．仮に，$v \in [\![q_1 \mathbin{.} q_2]\!]$ かつ $v \in [\![q'_1 \mathbin{.} q'_2]\!]$ となる v が存在したとする．このとき，$[\![q_1]\!] \subseteq [\![p_1]\!]$ かつ $[\![q'_1]\!] \subseteq [\![p_1]\!]$ となり，$[\![q_2]\!] \subseteq [\![p_2]\!]$ かつ $[\![q'_2]\!] \subseteq [\![p_2]\!]$ となる．ここで $v \in [\![p_1 \mathbin{.} p_2]\!]$ であるから，言語 $\text{V}\textsc{dl}^+$ におけるパターンマッチの一意性より $v = v_1 \mathbin{.} v_2$ と一意に分解できる．すると $v_2 \in [\![q_2]\!]$ かつ $v_2 \in [\![q_2]\!]$ となり矛盾．よって，$[\![q_1 \mathbin{.} q_2]\!] \cap [\![q'_1 \mathbin{.} q'_2]\!] = \emptyset$ である．□

上の補題により，式 (D-PSP) が成り立つため以下が言える．

定理 7.3. アルゴリズム Alg-Sp は決定的なプログラムを返す．□

よって，Alg-Sp において，我々は関数の出力の型を推論していない．これは，$\text{V}\textsc{dl}^+$ の関数の出力がパターンにより分解されることがないためである．つまり，ある関数から呼び出される関数の入力は，必ず，呼び出した関数の入力の一部分になっている．アルゴリズム Alg-Sp の停止性はこの条件に依存している．また，$\text{V}\textsc{dl}^+$ において関数の定義域は，正規生垣言語では表現できない場合があることにも注意する．言語 $\text{V}\textsc{dl}^+$ についても，Alg-Sp が成功するのは，非正規性が関数の再帰構造のみから現れるためである．アルゴリズム Alg-Sp は，関数の再帰構造を変更しない．そのため，元の関数の定義域が正規生垣言語では表現できない場合は，特化された後でも関数の定義域は正規生垣言語では表現できない．たとえば，関数 cf

$$\begin{aligned}&\mathbf{data}\ T \mathrel{\hat{=}} (\texttt{<a>}|\texttt{<b>})^*\\&cf(x :: T) \mathrel{\hat{=}} g(x)\\&g(\varepsilon) \mathrel{\hat{=}} \varepsilon\\&g(\texttt{<a>} \mathbin{.} r \mathbin{.} \texttt{<b>}) \mathrel{\hat{=}} \texttt{<c>} \mathbin{.} g(x) \mathbin{.} \texttt{<d>}\end{aligned}$$

は，以下の関数へと特化される．

$$
\begin{aligned}
&cf(x :: T) \doteq g|_T(x) \\
&g|_T(\varepsilon) && \doteq \varepsilon \\
&g|_T(\texttt{<a>}\,.\,r :: T\,.\,\texttt{<b>}) && \doteq \texttt{<c>}\,.\,g|_T(x)\,.\,\texttt{<d>}
\end{aligned}
$$

関数 g の定義域も関数 $g|_T$ の定義域も，$\{\texttt{<a>}^n\texttt{<b>}^n \mid n \in \mathbb{N}\}$ であり，正規生垣言語では表現できない．

型に基づく関数の特化は，プログラム中変数パターンの型を置き換え，呼び出される関数を特化したものに置き換えることにより，型付けにおける「部分型付け」規則の使用を除去していると見ることができる．

関数を特化することにより，関数の挙動が引数の型には依存せず再帰構造のみから分かるようになる．

定理 7.4 (完全な特化)**.** 特化後のプログラムにおいて，規則 $g(\overrightarrow{p}) \doteq \ldots$ の右辺に出現する関数 $f(\overrightarrow{x})$ に対し，任意の代入 θ について，$\exists\overrightarrow{v}.\ f(\overrightarrow{x}\theta) \Downarrow \overrightarrow{v}$ であるならば $\forall x \in \{\overrightarrow{x}\}.\ \theta(x) \in [\![\Gamma_{\overrightarrow{p}}(x)]\!]$ となる．

証明. アルゴリズム ALG-SP の定義と定理 7.1 により，簡単に示される．　□

定理 7.5 (特化の正しさ)**.** 特化により関数 $f|_{\overrightarrow{T}}$ が f と $\overrightarrow{T}$ より得られたとする．このとき $\overrightarrow{v} \in \overrightarrow{T}, f(\overrightarrow{v}) \Downarrow \overrightarrow{u}$ ならば，そのときに限り $f|_{\overrightarrow{T}}(\overrightarrow{v}) \Downarrow \overrightarrow{u}$ となる．

証明. ここで「そのときに限り」

$$\forall\overrightarrow{u}, \overrightarrow{v}.\ \overrightarrow{v} \in \overrightarrow{T} \wedge f(\overrightarrow{v}) \Downarrow \overrightarrow{u} \Leftarrow f|_{\overrightarrow{T}}(\overrightarrow{v}) \Downarrow \overrightarrow{u}$$

は定理 7.4 と ALG-SP が関数の再帰構造を変えないことより明らかである．よって,「ならば」

$$\forall\overrightarrow{u}, \overrightarrow{v}.\ \overrightarrow{v} \in \overrightarrow{T} \wedge f(\overrightarrow{v}) \Downarrow \overrightarrow{u} \Rightarrow f|_{\overrightarrow{T}}(\overrightarrow{v}) \Downarrow \overrightarrow{u}$$

を評価の証明木における #FUNDEPTH$(f(\overrightarrow{v}))$ の上の帰納法により示す．なお，#FUNDEPTH は第 5 章において，証明に利用したものと同じで，#FUNDEPTH(e) は，式 e を評価するのに使用した FUN の数を $e \Downarrow v$ の証明木に沿って縦に数えたもの最大値，つまり深さを表す．

基底：#FUNDEPTH$(f(\overrightarrow{v})) = 0$**.**

このとき，規則

$$f(\overrightarrow{p}) \doteq \overrightarrow{q}$$

で，ある代入について，$\overrightarrow{p}\theta = \overrightarrow{v}$ となるものが唯一存在する．定理 7.1 より，特化された関数 $f|_{\overrightarrow{T}}$ は,

$$f|_{\overrightarrow{T}}(\overrightarrow{p'}) \doteq \overrightarrow{q}$$

となる規則で $\overrightarrow{p'}\theta = \overrightarrow{v}$ となるものを唯一つ持つ．よって，$f(\overrightarrow{v}) \Downarrow \overrightarrow{q}\theta$ であり，$f|_{\overrightarrow{T}}(\overrightarrow{v}) \Downarrow \overrightarrow{q}\theta$ となる．

帰納：$\#\textsc{FunDepth}(f(\overrightarrow{v})) > 1$.

このとき，規則

$$f(\overrightarrow{p}) \doteq \mathbf{let}\,\{(\overrightarrow{y_i}) \doteq f_i(\overrightarrow{x_i})\}_{i\in\{1,\ldots,n\}} \\ \mathbf{in}\ \overrightarrow{q}$$

で，ある代入について，$\overrightarrow{p}\theta = \overrightarrow{v}$ となるものが唯一存在する．定理 7.1 より，特化された関数 $f|_{\overrightarrow{T}}$ は,

$$f|_{\overrightarrow{T}}(\overrightarrow{p'}) \doteq \mathbf{let}\,\{(\overrightarrow{y_i}) \doteq f_i|_{\overrightarrow{S_i}}(\overrightarrow{x_i})\}_{i\in\{1,\ldots,n\}} \\ \mathbf{in}\ \overrightarrow{q}$$

となる規則で $\overrightarrow{p'}\theta = \overrightarrow{v}$ となるものを唯一つ持つ．ただし，$\overrightarrow{S_i} = \Gamma_{\overrightarrow{p'}}(\overrightarrow{x_i})$ である．ここで，$\overrightarrow{x_i}\theta \in \overrightarrow{S_i}$ であるため，$\overrightarrow{w_i}$ を

$$f_i(\overrightarrow{x_i}\theta) \Downarrow \overrightarrow{w_i}$$

とおくと，帰納法の仮定より，

$$f_i|_{\overrightarrow{S_i}}(\overrightarrow{x_i}\theta) \Downarrow \overrightarrow{w_i}$$

となる．よって，

$$\eta(\overrightarrow{y_i}) = \overrightarrow{w_i} \text{ for all } i \in \{1,\ldots,n\}$$

により代入 η を定義すると，

$$f(\overrightarrow{x_i}\theta) \Downarrow \overrightarrow{q}\theta\eta$$

となり

$$f|_{\overrightarrow{T}}(\overrightarrow{x_i}\theta) \Downarrow \overrightarrow{q}\theta\eta$$

となる．□

定理 7.4 と定理 7.5 は，本章で我々が提案した特化手法の主要定理である．これらの定理により，たとえば，我々は，変数と構成子のみからなる式を除き，式の値域（厳密な型）を求める際に，式中の変数の型を気にする必要がなくなる．たとえば，関数呼出式 $f(x)$ の値域を求めるのに，x の型を知る必要はない．これは，特化後のプログラムにおいて，f が既に x の型に特化されているためである．ただし，変数式 x の値域を求める際には x の型を考える必要がある．

7.3.4　パターンの併合

特化手法を素朴に適用すると不必要に多くの規則を生成しうる．たとえば，関数 f

$$\begin{array}{ll}\mathbf{data}\ A \doteq \texttt{<a>}(B)^* & \\ \mathbf{data}\ B \doteq (\texttt{<b>} \mid \texttt{<c>})^* & \\ f(\varepsilon) & \doteq \varepsilon \\ f(\texttt{<a>}(x :: B) \centerdot y :: A) & \doteq \texttt{<b>}(x) \centerdot f(y)\end{array}$$

に素朴 Alg-Sp を適用すると以下を得る．

$$
\begin{aligned}
&\mathbf{data}\ E \mathrel{\hat{=}} \varepsilon \\
&\mathbf{data}\ S \mathrel{\hat{=}} (\texttt{<b>} \mid \texttt{<c>}) \centerdot (S \mid E) \\
&\mathbf{data}\ T \mathrel{\hat{=}} \texttt{<a>}(S \mid E) \centerdot (T \mid E) \\
&f(\varepsilon) && \mathrel{\hat{=}} \varepsilon \\
&f(\texttt{<a>}(x :: B) \centerdot y :: A) && \mathrel{\hat{=}} \texttt{<b>}(x) \centerdot f|_A(y) \\
&f|_A(\varepsilon) && \mathrel{\hat{=}} \varepsilon \\
&f|_A(\texttt{<a>}(x :: S) \centerdot y :: E) && \mathrel{\hat{=}} \texttt{<b>}(x) \centerdot f|_E(y) \\
&f|_A(\texttt{<a>}(x :: S) \centerdot y :: T) && \mathrel{\hat{=}} \texttt{<b>}(x) \centerdot f|_T(y) \\
&f|_A(\texttt{<a>}(x :: E) \centerdot y :: E) && \mathrel{\hat{=}} \texttt{<b>}(x) \centerdot f|_E(y) \\
&f|_A(\texttt{<a>}(x :: E) \centerdot y :: T) && \mathrel{\hat{=}} \texttt{<b>}(x) \centerdot f|_T(y) \\
&f|_E(\varepsilon) && \mathrel{\hat{=}} \varepsilon \\
&f|_T(\texttt{<a>}(x :: S) \centerdot y :: E) && \mathrel{\hat{=}} \texttt{<b>}(x) \centerdot f|_E(y) \\
&f|_T(\texttt{<a>}(x :: S) \centerdot y :: T) && \mathrel{\hat{=}} \texttt{<b>}(x) \centerdot f|_T(y) \\
&f|_T(\texttt{<a>}(x :: E) \centerdot y :: E) && \mathrel{\hat{=}} \texttt{<b>}(x) \centerdot f|_E(y) \\
&f|_T(\texttt{<a>}(x :: E) \centerdot y :: T) && \mathrel{\hat{=}} \texttt{<b>}(x) \centerdot f|_T(y)
\end{aligned}
$$

となる．これは，A を記述するための曖昧でない正規生垣文法を部分集合構成により得ると以下となるためである．

$$
\begin{aligned}
\{A, B\} &\to \varepsilon \\
\{A\} &\to \texttt{<a>}(\{B\})\{A, B\} \\
\{A\} &\to \texttt{<a>}(\{B\})\{A\} \\
\{A\} &\to \texttt{<a>}(\{A, B\})\{A, B\} \\
\{A\} &\to \texttt{<a>}(\{A, B\})\{A\} \\
\{B\} &\to \texttt{<b>}\{A, B\} \\
\{B\} &\to \texttt{<c>}\{A, B\} \\
\{B\} &\to \texttt{<b>}\{B\} \\
\{B\} &\to \texttt{<c>}\{B\}
\end{aligned}
$$

ただし，特化された関数の定義においては，簡便のため $T = \{A\}, S = \{B\}, E = \{A, B\}$ という型名を用いた．ところが，この f に対しては以下のようなもっと簡潔な特化された関数の定義が存在する．

$$
\begin{aligned}
&f(\varepsilon) && \mathrel{\hat{=}} \varepsilon \\
&f(\texttt{<a>}(x :: B) \centerdot y :: A) && \mathrel{\hat{=}} \texttt{<b>}(x) \centerdot f|_A(y) \\
&f|_A(\varepsilon) && \mathrel{\hat{=}} \varepsilon \\
&f|_A(\texttt{<a>}(x :: B) \centerdot y :: A) && \mathrel{\hat{=}} \texttt{<b>}(x) \centerdot f|_A(y)
\end{aligned}
$$

この関数定義規則の不必要な増加は，双方向化において望ましくない．なぜならば，第 5 章の双方向化手法に従うと，我々は異なる規則に異なる構成子を用いることにより使用した規則を識別するためである．ここで，使用した異なる識別構成子の数が多いほど，求まる補関数は大きくなる．第 5 章では，適切な規則集合の分割を与えることによって，異なる規則に対し，同じ規則識別構成子を使用することができることを示した．しかし，特化において

どの位規則が増加するかが自明ではないため，規則集合の分割を人が与えるのは難しい．確かに規則集合の分割を機械的に与えることは可能であり，実際，機械的な分割構成法として，言語 VDL に対する双方向化のプロトタイプ実装[1] では貪欲に分割を構成する手法を採用している．しかし，第 8 章で詳しく述べるが，VDL$^+$ に対しては，我々は規則の分割による識別構成子の統一を行わない．なぜなら，分割のための条件が VDL$^+$ では VDL よりも複雑になるし，また，変数パターンを使用することで分割による識別構成子の統一と同等のことができるためである．

そこで，我々は ALG-SP のステップ 4 において生成されるパターンを併合することによって，特化の性質を変えることなく生成される規則を減らすことを考える．たとえば，型 A と パターン $p = \texttt{<a>}(x :: B) \centerdot y :: A$ について以下のパターンが生成される．

$$\mathsf{psp}(p; \{A, B\}) \cup \mathsf{psp}(p; \{A\}) = \begin{Bmatrix} \texttt{<a>}(x :: E) \centerdot y :: E \\ \texttt{<a>}(x :: E) \centerdot y :: T \\ \texttt{<a>}(x :: S) \centerdot y :: E \\ \texttt{<a>}(x :: S) \centerdot y :: T \end{Bmatrix}$$

ところが，これらのパターンは併合し，一つのパターン

$$\texttt{<a>}(x :: B) \centerdot y :: A$$

にすることができる．これにより，簡潔な特化された関数を得ることができるようになる．

つまり，我々は ALG-SP のステップ 4 における，一つの引数に対する特化されたパターンの集合の生成

$$\bigcup_{j \in \{1 \dots n_i\}} \mathsf{psp}(p_i; A''_{i\,G''_i})$$

の結果得られるパターンを併合することを考える．ここでは，併合操作と併合可能なパターンの条件を与える．手続き psp の定義より，$p' \in \mathsf{psp}(p; A_G)$ であるならば，p' と p は変数パターンの型を除いて同じ形式をしていることが言える．よって，$p', p'' \in \mathsf{psp}(p; A_G)$ を併合する際に $\Gamma_{p'}$ と $\Gamma_{p''}$ を併合し，併合した型環境に基づき新たなパターンを構成することを考える．

型環境 $\Gamma_{p'}$ と $\Gamma_{p''}$ とを併合した際に，元と意味が変化しないことが，ALG-SP の正しさを保証する上で必要である．すなわち，併合後の型環境を Γ' とすると

$$\begin{pmatrix} \forall \{h_x \mid x \in \mathsf{vars}(p)\} \in (\mathcal{H}_\Sigma)^{|\mathsf{vars}(p)|}. \\ \quad (\forall x.\ h_x \in \Gamma_{p'}(x)) \vee (\forall x.\ h_x \in \Gamma_{p''}(x)) \Leftrightarrow (\forall x.\ h_x \in [\![\Gamma'(x)]\!]) \end{pmatrix} \qquad \text{(Merge-Env)}$$

となることである．たとえば，$\Gamma_{p'} = \{x \mapsto \texttt{<a>}, y \mapsto \texttt{<b>}\}$ と $\Gamma_{p''} = \{x \mapsto \texttt{<a>}, y \mapsto \texttt{<c>}\}$ は併合できて，

$$\Gamma' = \{x \mapsto \texttt{<a>}, y \mapsto (\texttt{<b>}|\texttt{<c>})\}$$

[1] `http://www.ipl.t.u-tokyo.ac.jp/~kztk/bidirectionalization/`

となる．ところが，$\Gamma_{p'} = \{x \mapsto \mathsf{<a>}, y \mapsto \mathsf{<b>}\}$ と $\Gamma_{p''} = \{x \mapsto \mathsf{<d>}, y \mapsto \mathsf{<c>}\}$ は併合できない．これは，式 (Merge-Env) を満たす Γ' が存在しないためである．式 (Merge-Env) を満たさない場合，パターンを併合する前と併合した後でパターンにマッチする生垣の集合が異なってしまう．

型環境の併合については，以下の十分条件[2]が与えられる．もし，

$$\exists! x \in \mathsf{vars}(p).\ \forall y \in \mathsf{vars}(p).\ x \neq y \Rightarrow [\![\Gamma_{p'}(y)]\!] = [\![\Gamma_{p''}(y)]\!] \qquad \text{(Mergeable)}$$

ならば，

$$\Gamma'(x) = \begin{cases} \Gamma_{p'}(x) \mid \Gamma_{p''}(x) & \forall y \in \mathsf{vars}(p).\ x \neq y \Rightarrow \Gamma_{p'}(y) = \Gamma_{p''}(y) \\ \Gamma_{p'}(x) & \text{otherwise} \end{cases}$$

により型環境の併合が行える．

型環境の併合が成功すれば，p 中の $x :: T$ を $x :: \Gamma'(x)$ で置き換えることにより，併合されたパターン p''' が得られる．ここで，p 中の $x :: T$ を，$x :: \Gamma_{p'}(x)$ で置き換えたものが p' であり，$x :: \Gamma_{p''}(x)$ で置き換えたものが p'' であることに注意する．式 (Merge-Env) により，

$$[\![p''']\!] = [\![p']\!] \cup [\![p'']\!]$$

であることが保証される．

例として前述のパターンの集合

$$\begin{aligned} p_1 &= \mathsf{<a>}(x :: E) \centerdot y :: E \\ p_2 &= \mathsf{<a>}(x :: E) \centerdot y :: T \\ p_3 &= \mathsf{<a>}(x :: S) \centerdot y :: E \\ p_4 &= \mathsf{<a>}(x :: S) \centerdot y :: T \end{aligned}$$

を併合することを考える．なお，それぞれ，以下の型環境と対応づいている．

$$\begin{aligned} \Gamma_{p_1} &= \{x \mapsto E, y \mapsto E\} \\ \Gamma_{p_2} &= \{x \mapsto E, y \mapsto T\} \\ \Gamma_{p_3} &= \{x \mapsto S, y \mapsto E\} \\ \Gamma_{p_4} &= \{x \mapsto S, y \mapsto T\} \end{aligned}$$

---

[2]必要十分条件を考える必要はない．なぜなら，必要十分条件は，

$$\begin{aligned} & (\exists x \in \mathsf{vars}(p).\ [\![\Gamma_{p'}(x)]\!] = [\![\Gamma_{p''}(y)]\!] = \emptyset) \\ \vee\ & (\forall x \in \mathsf{vars}(p).\ [\![\Gamma_{p'}(x)]\!] \subseteq [\![\Gamma_{p''}(x)]\!]) \\ \vee\ & (\forall x \in \mathsf{vars}(p).\ [\![\Gamma_{p'}(x)]\!] \supseteq [\![\Gamma_{p''}(x)]\!]) \\ \vee\ & (Mergeable) \end{aligned}$$

であるが，上三行の条件は，アルゴリズム ALG-SP 中で psp の返すパターンの集合もしくはそれを併合して得られるパターンの集合に含まれるパターン p', p'' については，成り立たないためである．手続き psp の第一規則および第二規則の条件式により，一行目の場合が起こらないことが保証される．また，$\mathsf{psp}(p; A_G)$ が ALG-SP 中で呼ばれる場合，G は曖昧でないことから，psp の返すそれぞれのパターン p について，$[\![p]\!]$ は互いに素である．そのため，二行目と三行目が起こらない．

条件 (Mergeable) より，Γ_{p_1} と Γ_{p_2}，Γ_{p_1} と Γ_{p_3}，Γ_{p_2} と Γ_{p_4}，Γ_{p_3} と Γ_{p_4} が併合できる．ここで，Γ_{p_1} と Γ_{p_2}，Γ_{p_3} と Γ_{p_4} とを併合したとすると，以下が得られる．

$$\Gamma_{12} = \{x \mapsto E, y \mapsto A\}$$
$$\Gamma_{34} = \{x \mapsto S, y \mapsto A\}$$

併合において，Γ_{p_1} と Γ_{p_2} の併合と Γ_{p_1} と Γ_{p_3} との併合を同時に行えないことに注意する．これは，特化後のプログラムの決定性を保証するためである．型環境 Γ_{12} と Γ_{34} は，条件 (Mergeable) を満たすため，併合可能であり，併合すると型環境

$$\Gamma_{1234} = \{x \mapsto B, y \mapsto A\}$$

を得る．型環境 Γ_{1234} からパターンを構成すると

$$\texttt{<a>}(x :: B) \centerdot y :: A$$

を得る．これまで述べた通り，パターンの併合を Alg-Sp に導入することで，プログラム

$$\begin{array}{ll} \textbf{data}\ A \mathrel{\hat{=}} \texttt{<a>}(B)^* & \\ \textbf{data}\ B \mathrel{\hat{=}} (\texttt{<b>} \mid \texttt{<c>})^* & \\ f(\varepsilon) & \mathrel{\hat{=}} \varepsilon \\ f(\texttt{<a>}(x :: B) \centerdot y :: A) & \mathrel{\hat{=}} \texttt{<b>}(x) \centerdot f(y) \end{array}$$

から，簡潔な特化されたプログラム

$$\begin{array}{ll} f(\varepsilon) & \mathrel{\hat{=}} \varepsilon \\ f(\texttt{<a>}(x :: B) \centerdot y :: A) & \mathrel{\hat{=}} \texttt{<b>}(x) \centerdot f|_A(y) \\ f|_A(\varepsilon) & \mathrel{\hat{=}} \varepsilon \\ f|_A(\texttt{<a>}(x :: B) \centerdot y :: A) & \mathrel{\hat{=}} \texttt{<b>}(x) \centerdot f|_A(y) \end{array}$$

を得ることができる．

なお，パターンの併合は停止性を壊さない．これは，パターンの併合が導入する型は，既存の型の選言で記述されるものであるためである．

7.4 特化手法の拡張

本章では，我々は言語 Vdl$^+$ で記述されたプログラムの特化について議論してきた．これは，後の双方向化のためである．提案する特化アルゴリズム Alg-Sp 自体は，Vdl$^+$ よりも広いクラスのプログラムに対し適用することができる．これは，特化手法の停止性（定理 7.2）や正しさ（定理 7.5）は，関数の出力がパターンマッチによって分解されることはないというプログラムの性質のみに依存しているためである．そのため，たとえばプログラムが，macro forest transducer [PS04] のような，累積変数を含む場合も提案する特化手法は停止し，正しく動く．ただし，プログラム累積変数を含む場合，定理 7.4 には少々変更が必要に

なる．特化された後においても，関数の引数のうち累積変数部分については型を考慮する必要がある．

提案する特化手法は XML 属性の操作を含むプログラムも扱うことができる．これは，どの要素がどの XML 属性かという情報は，正規生垣文法に埋め込むことにより扱えるためである．しかし，単純に XML 属性に関する制約を正規生垣文法で扱ってしまうと，正規生垣文法の非終端記号の数が爆発する．なぜなら，n 個の異なる XML 属性を扱うためには，2^n 個の非終端記号が必要なためである．これは，XML 属性は同じ属性が重複してはならないがどの順番で現れてもよいため，どの XML 属性が既に現れたかを区別しなければならないためである．XML 属性に対する特化を効率よく行うためには，レコード型 [BP99] の取り扱いに関する議論が応用できることが期待できる．

なお，本論文は，累積変数および XML 属性を含むプログラムの双方向化の議論はしない．第 9 章で述べるように，これらは今後の課題である．

7.5　まとめ

本章では，我々は VDL$^+$ で記述されたプログラムに対する，引数の型に基づく関数の特化手法を与えた．本章の提案する特化手法は，必ず停止し，決定的なプログラムを返す．特化により，関数の呼出式の値域と関数の値域が一致するようなる．そのため，関数の単射性や可能な評価結果が，関数が使用される文脈，つまり引数の型，によって変わることがなくなる．これにより，より多くの関数の単射性を効果的に解析でき，よりよい補関数を求めるようになることが期待される．

第8章　XML上の変換プログラムの双方向化

本章では，第6章の言語 $\textsc{Vdl}^+$ で記述されたプログラムに対する双方向化について述べる．双方向化の基本的なアイデアは第5章のものと同様である．すなわち，順方向変換がどこで単射性を失っているのかを調べ，順方向変換を単射にするの十分な情報を返すように補関数を導出し，導出された補関数に基づき逆方向変換を定める．

8.1　補関数の導出

第5章では，我々は以下の情報を補関数に含めるようにすることで補関数を構成した.

- 使用されない変数.
- 評価が使用する規則.

言語 $\textsc{Vdl}^+$ についても同様に，我々は補関数をこれらの情報を含めるようにし構成する．たとえば，関数

$$fst(x,y) \mathrel{\hat{=}} x$$

に対し，使用されない変数 y の情報を補うことにより，以下の補関数が得られる．

$$fst^{\mathrm{c}}(x,y) \mathrel{\hat{=}} y$$

また，たとえば，関数

$$\begin{aligned} add(\texttt{<z>}) &\mathrel{\hat{=}} y \\ add(\texttt{<s>} \centerdot x, y) &\mathrel{\hat{=}} \texttt{<s>} \centerdot add(x,y) \end{aligned}$$

に対し，評価中に使用される規則を識別子により憶えることで，以下の補関数が得られる．

$$\begin{aligned} add^{\mathrm{c}}(\texttt{<z>}) &\mathrel{\hat{=}} \mathsf{B}_1(y) \\ add^{\mathrm{c}}(\texttt{<s>} \centerdot x, y) &\mathrel{\hat{=}} \mathsf{B}_2(add^{\mathrm{c}}(x,y)) \end{aligned}$$

しかし，言語 $\textsc{Vdl}$ の場合と異なり，言語 $\textsc{Vdl}^+$ で記述されたプログラムに対しては，上の二つの情報を補うだけでは不十分な場合がある．たとえば，以下の関数 g_1 と g_2 を考える．

$$\begin{aligned} &\mathbf{data}\ T \mathrel{\hat{=}} \texttt{<a>}^* \\ &\mathbf{data}\ S \mathrel{\hat{=}} \texttt{<b>}^* \\ &g_1(x :: T, y :: T) \mathrel{\hat{=}} x \centerdot y \\ &g_2(x :: S, y :: T) \mathrel{\hat{=}} x \centerdot y \end{aligned}$$

関数 g_1, g_2 両方に対し，上の二つの情報を補う必要はない．なぜなら，g_1, g_2 の左辺に出現する変数は全て右辺式に出現し，また，g_1, g_2 は一つの規則から成るためである．ところが，g_2 が単射であるに対し，g_1 は以下のように単射ではない．

$$g_1(\texttt{<a><a>}, \texttt{<a>}) = \texttt{<a><a><a>}$$
$$g_1(\texttt{<a>}, \texttt{<a><a>}) = \texttt{<a><a><a>}$$

したがって，補関数を作成するためには，連接に対しても情報を補う必要がある．言語 $\mathrm{V{\scriptstyle DL}}^+$ で記述された関数に対し，我々は，上記二つ情報に加え連接演算の第一引数の生垣の長さの情報を補うことによって，補関数を導出する．たとえば，g_1 に対し，本章で提案する補関数導出手法は以下の補関数を導出する．

$${g_1}^{\mathrm{c}}(x :: T, y :: T) \mathrel{\hat{=}} \textsc{length}(x)$$

第 5 章では，関数の単射性を解析することで，より小さい補関数を導出できることを示した．これは，言語 $\mathrm{V{\scriptstyle DL}}^+$ についても同様である．言語 $\mathrm{V{\scriptstyle DL}}^+$ で記述された関数の単射性を判定するためには，連接を考慮する必要がある．たとえば，上の g_2 は g_1 とは異なり単射である．なぜなら，g_2 の返り値 <b> . . . <b><a> . . . <a> は，$x :: S$ に属する部分と $y :: T$ に属する部分に常に一意に分解できるためである．つまり，g_2 の右辺において，連接の演算の第一引数の式の値域 $[\![S]\!]$ と第二引数の式の値域 $[\![T]\!]$ の上で，連接演算「$\centerdot$」が単射となっている．

本節では，まず 8.1.1 節において，言語 $\mathrm{V{\scriptstyle DL}}^+$ で記述されたプログラム中の式の値域の，厳密な推論手法を与える．しかし，得られる厳密な値域はあまり双方向化の議論において扱いやすいものではない．たとえば，言語 $\mathrm{V{\scriptstyle DL}}^+$ の二つの式の値域に重なりがあるかどうかは決定不能である．よって，式の値域を直接利用すると，単射性解析を行うことができない．よって，8.1.2 節において，値域を扱いやすいように近似する手法について述べる．その後，8.1.3 節において近似された値域を用いて，健全に単射性判定が行えることを示す．そして，8.1.4 節で補関数導出法に述べる．

8.1.1　値域の厳密な推論

ここでは，関数の値域を厳密に推論する手法を与える．すなわち，

$$f(\ldots x :: S \ldots y :: T \ldots) \mathrel{\hat{=}} \ldots g(x, y) \ldots$$

の中の $g(x, y)$ について，集合

$$\{v \mid g(x, y) \Downarrow v, x \in [\![S]\!], y \in [\![T]\!]\}$$

の記述を与える．プログラムに対し，第 7 章に示した関数の引数の型に基づく特化を行うことにより，関数の値域と関数呼出式の値域を一致させられる．関数呼出式の値域がわかれば，

あとは他の式の値域は簡単に求めることができる．よって，関数の値域を厳密に推論できれば，式の値域を厳密に求めることができる．

第 5 章に示したように，言語 VDL で記述されたプログラムに対しては，正規木文法により式の値域を表現することができる．言語 VDL と比べ，言語 VDL$^+$ は以下の言語要素を含むため，式の値域は正規生垣言語よりも複雑になる．

- 連接.
- 多返値関数.

たとえば，関数

$$\begin{aligned} &\mathit{wrapCD}(\varepsilon) &&\doteq \varepsilon \\ &\mathit{wrapCD}(\texttt{<a>}(x)) &&\doteq \texttt{<c>}\,.\,\mathit{wrapCD}(x)\,.\,\texttt{<d>} \end{aligned}$$

の値域は $\{\texttt{<c>}^n\texttt{<d>}^n \mid n \geq 0\}$ と正規生垣言語にならない．また，たとえば，関数

$$\begin{aligned} &\mathit{snocAB}(x) &&\doteq \mathbf{let}\,(s,t) \doteq g(x)\,\mathbf{in}\,\texttt{<first>}(s)\,.\,\texttt{<second>}(t) \\ &g(\texttt{<z>}) &&\doteq (\texttt{<a>}, \texttt{<b>}) \\ &g(\texttt{<s>}\,.\,x) &&\doteq \mathbf{let}\,(s,t) \doteq g(x)\,\mathbf{in}\,(\texttt{<c>}\,.\,t, \texttt{<c>}\,.\,s) \end{aligned}$$

の値域も正規生垣文法にならず，また依存性が含まれる．関数 snocAB の返り値は <first> と <second> の子供に同数の <c> を含む．また，関数 snocAB の返り値は，<first> 要素の末端が <a> ならば必ず <second> 要素の末端は <b> であり，<first> 要素の末端が <b> ならば必ず <second> 要素の末端は <a> である．

我々は，構文主導翻訳 [AU72] において議論された「同期」の概念を含む文脈自由生垣文法を用いることにより，関数の値域を表現する．言語 VDL で記述された関数については，第 5 章に示したように，[MPS07] と同様に引数の情報を「忘れる」ことにより関数の値域を表現していた．言語 VDL$^+$ に対しては，同じ多返値関数呼出の返り値であるという「同期」情報を除き引数の情報を忘れることで，関数の値域を求められる．たとえば，上の snocAB の値域は以下の文法の非終端記号 T により表現される．

$$\begin{aligned} &T_{\mathit{snocAB}} &&\to \texttt{<first>}(\pi_1 T_g)\,.\,\texttt{<second>}(\pi_2 T_g) \\ &T_g &&\to (\texttt{<a>}, \texttt{<b>}) \\ &T_g &&\to (\texttt{<c>}\,.\,\pi_2 T_g^{(1)}, \texttt{<c>}\,.\,\pi_1 T_g^{(1)}) \end{aligned}$$

上の二行目は，$g(\texttt{<s>}\,.\,x)$ の返り値は，同じ $g(x)$ の呼出の返り値の第二要素の頭に <c> 要素を連接したものと第一要素の頭に <c> 要素を連接したものの組であることを表している．

定義 8.1 (文脈自由多生垣文法)．**文脈自由多生垣文法** G は，三つ組 $G = (\Sigma, N, R)$ である．ここで，Σ，N，R は以下の通りである．

- Σ は構成子の集合.
- N は返り値の数が関連付けられた非終端記号の集合.

- R は生成規則の集合であり，それぞれの生成規則は以下の形式をしている．

$$A \to (\mathcal{C}_1[\pi_{j_{11}} A_{11}^{(i_{11})}, \ldots, \pi_{j_{1n_1}} A_{1n_1}^{(i_{1n_1})}], \ldots, \mathcal{C}_m[\pi_{j_{m1}} A_{m1}^{(i_{m1})}, \ldots, \pi_{j_{mn_m}} A_{mn_m}^{(i_{1n_m})}])$$

ここで，$\mathcal{C}_1, \ldots, \mathcal{C}_m$ は Σ 上の文脈，$A_{11}, \ldots, A_{mn_m} \in N$ であり，$i_{11}, \ldots, i_{mn_m} \in \mathbb{N}$ は同期のための識別子であり，$j_{11}, \ldots, j_{mn_m} \in \mathbb{N}$ は射影のための識別子である．また，右の組の要素数 m は非終端記号 A に関連付けられた返り値の数である．さらに，$\pi_i B$ と書いた場合，B の返り値の数を l として $i \leq l$ であるとする． □

上の定義は，複数のテキスト間の翻訳の枠組みである Multitext Grammar [Mel03] を拡張した文法である，Generalized Multitext Grammar [MSW04] に近い．文脈自由多生垣文法 G において，全ての非終端記号の返り値の数が一で生成規則の右辺の同期識別子が互いに異なるとき，文法 G を文脈自由生垣文法と呼ぶ．文脈自由生垣文法は，context-free sequence-tree automata [OST03] と同じものだが，生垣の二分木表現の上の文脈自由木文法 [CDG⁺97] より表現力が小さい．

簡便のため，生成規則において同期識別子が右辺で一意である場合，同期識別子を省略する場合がある．また，返り値の数が 1 である非終端記号 A について，$\pi_1 A^{(i)}$ を単に $A^{(i)}$ と書く．

文脈自由多生垣文法に対し生成関係を定める．同じ同期識別子を持つ非終端記号は，同時に展開されなければならない．

定義 8.2 (生成関係)**.** 文脈自由多生垣文法 $G = (\Sigma, N, R)$ に対し，生成関係 $\to_G$ を以下で定める．

$$\overrightarrow{\mathcal{C}}[\pi_{k_1} A^{(i)}, \ldots \pi_{k_l} A^{(i)}] \to_G \overrightarrow{\mathcal{C}} \begin{bmatrix} \mathcal{C}_{k_1}[\pi_{j_{k_1 1}} A_{k_1 1}^{(i, i_{k_1 1}))}, \ldots, \pi_{j_{k_1 n_1}} A_{k_1 n_1}^{(i, i_{k_1 n_1})}], \\ \ldots, \\ \mathcal{C}_{k_l}[\pi_{j_{k_m 1}} A_{k_l 1}^{(i, i_{k_l 1})}, \ldots, \pi_{j_{k_l n_1}} A_{k_l n_1}^{(i, i_{k_l n_1})}] \end{bmatrix}$$

$$\Updownarrow$$

$$A \to (\mathcal{C}_1[\pi_{j_{11}} A_{11}^{(i_{11})}, \ldots, \pi_{j_{1n_1}} A_{1n_1}^{(i_{1n_1})}], \ldots, \mathcal{C}_m[\pi_{j_{m1}} A_{m1}^{(i_{m1})}, \ldots, \pi_{j_{mn_m}} A_{mn_m}^{(i_{1n_m})}]) \in R$$

ただし，$\overrightarrow{\mathcal{C}}$ は，$\pi_k A^{(i)}$ という形式の非終端記号を含まないとする．

定義 8.3 (言語，非終端記号の意味)**.** 文脈自由多生垣文法 $G = (\Sigma, N, R)$ に対し，n 返値の非終端記号 $A \in N$ の言語を以下で与える．

$$[\![A]\!]_G = \{w \mid (d_1, \ldots, d_n) \xrightarrow{*}_G w, A \to (d_1, \ldots, d_n) \in R\}$$ □

例 8.1 (Mirror)**.** 第 6 章の例 6.3 の関数 *mirror* を考える．このとき，*mirror* の値域は以下の文法の T により与えられる．

$$\begin{array}{lll} T_{mirror} & \to & \varepsilon \\ T_{mirror} & \to & Char^{(1)} \centerdot T_{mirror} \centerdot Char^{(1)} \\ Char & \to & \mathtt{a} \mid \mathtt{b} \mid \mathtt{c} \mid \ldots \end{array}$$

たとえば，

$$\begin{array}{ll} T^{()}_{mirror} & \to (Char^{(1)} \centerdot T_{mirror} \centerdot Char^{(1)}) \to (\mathtt{a} \centerdot T \centerdot \mathtt{a}) \\ & \to (\texttt{<a>} \centerdot Char^{(1,1)} \centerdot T_{mirror} \centerdot Char^{(1,1)} \centerdot \texttt{<a>}) \to (\mathtt{a} \centerdot \mathtt{b} \centerdot T \centerdot \mathtt{b} \centerdot \mathtt{a}) \to \mathtt{a} \centerdot \mathtt{b} \centerdot \mathtt{b} \centerdot \mathtt{a} \end{array}$$

となる．

例 8.2 (目次の付与)**.** 第 6 章の例 6.5 の関数 *toc* を考える．このとき，*toc* の返り値の集合は以下の文法の T により与えられる．

$$\begin{array}{lll} T_{toc} & \to & (\texttt{<ul>}(\pi_1 T_f^{(1)}), \pi_2 T_f^{(1)}) \\ T_f & \to & (\varepsilon, \varepsilon) \\ T_f & \to & (\texttt{<li>}(String^{(1)}) \centerdot \texttt{<ul>}(\pi_1 G^{(2)}) \centerdot \pi_1 T_f^{(3)}, \texttt{<h1>}(String^{(1)}) \centerdot P \centerdot \pi_2 T_g^{(2)} \centerdot \pi_2 T_f^{(3)}) \\ T_g & \to & (\varepsilon, \varepsilon) \\ T_g & \to & (\texttt{<li>}(String^{(1)}) \centerdot \pi_1 T_g^{(2)}, \texttt{<h2>}(String^{(1)}) \centerdot P \centerdot \pi_2 T_g^{(2)}) \\ P & \to & \texttt{<p>}(String) \end{array}$$

形式的には，以下のアルゴリズムにより，関数の値域を記述する文脈自由多生垣文法を得られらる．

アルゴリズム 8.1 (関数の値域の厳密な推論)**.**

入力：プログラム $\mathcal{P} = (G = (\Sigma, N, R), R')$

出力：文脈自由多生垣文法 $G'' = (\Sigma'', N'', R'')$

手続き：

1. $\Sigma'' := \Sigma$.
2. $N'' := N \cup \{T_f \mid f(\ldots) \mathrel{\hat{=}} \cdots \in R'\}$.
3. $R'' := R$.
4. それぞれの規則 $r' \in R'$

$$\begin{array}{l} r' = f(\overrightarrow{p}) \mathrel{\hat{=}} \textbf{let}\, \{(\overrightarrow{y_i}) \mathrel{\hat{=}} f_i(\overrightarrow{x_i})\}_{i \in \{1,\ldots,n\}} \\ \qquad\qquad\quad \textbf{in}\ (\overrightarrow{\mathcal{C}}[y_{11}, \ldots, y_{1|\overrightarrow{y}|}, \ldots, y_{n1}, \ldots, y_{n|\overrightarrow{y_n}|}, z_1, \ldots, z_{|\overrightarrow{z}|}]) \end{array}$$

について以下を行う．

(a) $R'' := R'' \cup \{r''\}$．ここで，

$$r'' = T_f \to (\overrightarrow{\mathcal{C}}[\pi_1 T_{f_1}^{(1)}, \ldots, \pi_{|\overrightarrow{y_i}|} T_{f_i}^{(1)}, \ldots, \pi_1 T_{f_n}^{(n)}, \ldots, \pi_{|\overrightarrow{y_n}|} T_{f_n}^{(n)}, S_1, \ldots, S_n]$$

である．ただし，$(S_1, \ldots, S_n) = (\Gamma_{\overrightarrow{p}}(z_1), \ldots, \Gamma_{\overrightarrow{p}}(z_n))$ とする．

5. $G'' = (\Sigma'', N'', R'')$. □

定理 8.1. プログラム $\mathcal{P}$ について，関数の値域の推論アルゴリズムにより文法 G が得られたとする．このとき，

$$\exists\theta.\ f(\overrightarrow{x}\theta) \Downarrow \overrightarrow{v} \Rightarrow (\pi_1 T_f, \ldots, \pi_{|\overrightarrow{v}|} T_f) \xrightarrow{*} \overrightarrow{v}$$

が成り立つ．

証明. 式の評価に関する帰納法により示す．今 $f(\overrightarrow{x}\theta)$ について，$f(\overrightarrow{x}\theta) \Downarrow v$ となったとする．このとき，規則

$$f(\overrightarrow{p}) \mathrel{\hat{=}} \mathbf{let}\,\{(\overrightarrow{y_i}) \mathrel{\hat{=}} f_i(\overrightarrow{x_i})\}_{i\in\{1,\ldots,n\}}\,\mathbf{in}\ (\overrightarrow{\mathcal{C}}\,[\overrightarrow{y_1}, \ldots \overrightarrow{y_n}, \overrightarrow{z}])$$

で，ある η について $\overrightarrow{x}\theta = \overrightarrow{p}\eta$ となるものが存在する．このとき，プログラムの意味より，$\overrightarrow{u_1}, \ldots \overrightarrow{u_n}, \overrightarrow{w}$ を

$$\begin{aligned} &\forall i \in \{1, \ldots, n\}.\ f_i(\overrightarrow{x_i}\eta) \Downarrow \overrightarrow{u_i} \\ &\overrightarrow{w} = \overrightarrow{z}\eta \end{aligned}$$

とおくと，

$$\overrightarrow{v} = \overrightarrow{\mathcal{C}}\,[\overrightarrow{u_1}, \ldots \overrightarrow{u_n}, \overrightarrow{w}]$$

と書ける．

一方 G は規則

$$T_f \to (\overrightarrow{\mathcal{C}}\,[\pi_1 T_{f_1}^{(1)}, \ldots, \pi_{|\overrightarrow{y_i}|} T_{f_i}^{(1)}, \ldots, \pi_1 T_{f_n}^{(n)}, \ldots, \pi_{|\overrightarrow{y_n}|} T_{f_n}^{(n)}, \overrightarrow{S}\,])$$

を持つ．ここで，

$$\overrightarrow{S} = (\Gamma_{\overrightarrow{p}}(z_1), \ldots, \Gamma_{\overrightarrow{p}}(z_{|\overrightarrow{z}|}))$$

である．パターンへの代入の定義より，

$$\forall i \in \{1, \ldots, |\overrightarrow{z}|\}.\ w_i \in [\![\Gamma_{\overrightarrow{p}}(z_i)]\!]$$

となる．よって，

$$\overrightarrow{S} \xrightarrow{*} \overrightarrow{w}$$

となる．帰納法の仮定より，それぞれの $i \in \{1, \ldots, n\}$ について

$$(\pi_1 T_{f_i}^{()}, \ldots, \pi_{\overrightarrow{y_i}} T_{f_i}^{()}) \xrightarrow{*} \overrightarrow{u_i}$$

である．よって，G は文脈自由なので

$$(\pi_1 T_f^{()}, \ldots, \pi_{|\overrightarrow{v}|} T_f^{()}) \xrightarrow{*} \mathcal{C}[\overrightarrow{u_1}, \ldots \overrightarrow{u_n}, \overrightarrow{w}] = \overrightarrow{v}$$

となる．

上で，$n = 0$ のときが，帰納法の基底に対応する． □

定理 8.2. プログラム $\mathcal{P}$ について，関数の値域の推論アルゴリズムにより文法 G が得られたとする．このとき，

$$\exists\theta.\ f(\overrightarrow{x}\theta) \Downarrow \overrightarrow{v} \Leftarrow (\pi_1 T_f, \ldots, \pi_{|\overrightarrow{v}|} T_f) \xrightarrow{*} \overrightarrow{v}$$

が成り立つ．

証明. 生成に関する帰納法により示す．

プログラムの規則

$$f(\overrightarrow{p}) \mathrel{\hat{=}} \mathbf{let}\ \{(\overrightarrow{y_i}) \mathrel{\hat{=}} f_i(\overrightarrow{x_i})\}_{i \in \{1,\ldots,n\}}\ \mathbf{in}\ (\overrightarrow{\mathcal{C}}[\overrightarrow{y_1}, \ldots \overrightarrow{y_n}, \overrightarrow{z}])$$

について考える．

今，G の規則

$$T_f \to (\overrightarrow{\mathcal{C}}[\pi_1 T^{(1)}_{f_1}, \ldots, \pi_{|\overrightarrow{y_i}|} T^{(1)}_{f_i}, \ldots, \pi_1 T^{(n)}_{f_n}, \ldots, \pi_{|\overrightarrow{y_n}|} T^{(n)}_{f_n}, \overrightarrow{S}])$$

について，

$$(\pi_1 T^{()}_f, \ldots, \pi_{|\overrightarrow{v}|} T^{()}_f) \to (\overrightarrow{\mathcal{C}}[\pi_1 T^{(1)}_{f_1}, \ldots, \pi_{|\overrightarrow{y_i}|} T^{(1)}_{f_i}, \ldots, \pi_1 T^{(n)}_{f_n}, \ldots, \pi_{|\overrightarrow{y_n}|} T^{(n)}_{f_n}, \overrightarrow{S}]) \xrightarrow{*} \overrightarrow{v}$$

となったとする．ここで，

$$\overrightarrow{S} = (\Gamma_{\overrightarrow{p}}(z_1), \ldots, \Gamma_{\overrightarrow{p}}(z_{|\overrightarrow{z}|}))$$

である．このとき，文法 G が文脈自由であることから，

$$\begin{array}{l} \forall i \in \{1, \ldots, n\}.\ (\pi_1 T^{(i)}_{f_i}, \ldots, \pi_{|\overrightarrow{y_i}|} T^{(i)}_{f_i}) \xrightarrow{*} \overrightarrow{u_i} \\ \overrightarrow{S} \xrightarrow{*} \overrightarrow{w} \end{array}$$

となる $\overrightarrow{u_1}, \ldots \overrightarrow{u_n}$ と $\overrightarrow{w}$ により，

$$\overrightarrow{v} = \overrightarrow{\mathcal{C}}[\overrightarrow{u_1}, \ldots \overrightarrow{u_n}, \overrightarrow{w}]$$

と書ける．

帰納法の仮定から，

$$\forall i \in \{1 \ldots n\}.\ f_i(\overrightarrow{x_i}\eta) \Downarrow \overrightarrow{u_i}$$

となるような η が存在する．また，

$$\overrightarrow{w}\zeta = \overrightarrow{v}$$

となる ζ が存在する．プログラムが既に特化されていることから，

$$\forall x \in \{x \mid \eta(x) \neq x\}.\ x\eta \in [\![\Gamma_{\overrightarrow{p}}(x)]\!]$$

となる．ここで，$\overrightarrow{x}\theta = \overrightarrow{p}\eta\zeta$ とすると

$$f(x\theta) \Downarrow \overrightarrow{\mathcal{C}}[\overrightarrow{u_1}, \ldots \overrightarrow{u_n}, \overrightarrow{w}] = \overrightarrow{v}$$

となる．

帰納法の基底は，上で $(\pi_1 T_f, \ldots, \pi_{|\overrightarrow{v}|} T_f) \to (\overrightarrow{\mathcal{C}}[]) = \overrightarrow{v}$ となるときである． □

定理 8.1 と定理 8.2 から，前述の関数の値域の推論アルゴリズムは厳密であると言える．

関数の値域が求まったので，これにより，式の値域を厳密に求めることができる．しかし，式の厳密な値域をそのまま双方向化に利用することは難しい．これは，文脈自由多生垣文法により記述された二つの値域に重なりがあるかどうかが一般に決定不能であるためである．第 5 章では，二つの規則の右辺式の厳密な型に重なりがあるかを調べることにより単射性を解析し，より小さい補関数を導出していた．

文脈自由多生垣文法のもとで記述される二つの値域に重なりがあるかないが一般には決定不能であることを確認しておく．なお，二つの文脈自由文法についてその生成する文字列に重なりがあるかないかが決定不能であることはよく知られた事実である [HMU06] ため，文脈自由多生垣文法の上で重なり判定であるかないかを示すことには新規性はない．文脈自由多生垣文法の上で重なり判定を確認する目的は，言語 $\mathrm{V{\scriptsize DL}}^+$ の性質を確認するためである．我々は，文脈自由多生垣文法の上で重なり判定が決定不能であることを，決定不能問題の一つであるポストの対応問題（Post's Correspondence Problem）[Pos46] に帰着することにより示す．

問題 (ポストの対応問題)．文字列の集合 $\alpha_1, \ldots, \alpha_n$ と $\beta_1, \ldots, \beta_n$ が与えられているとする．このとき，列

$$i_1, \ldots, i_m \in \{1, \ldots, n\}, 0 < m < \infty$$

で，

$$\alpha_{i_1} \ldots \alpha_{i_m} = \beta_{i_1} \ldots \beta_{i_m}$$

とできるものがあるかどうか判定せよ．

たとえば

$$\begin{pmatrix} \alpha_1 & \alpha_2 & \alpha_3 \\ \beta_1 & \beta_2 & \beta_3 \end{pmatrix} = \begin{pmatrix} \mathtt{100} & \mathtt{0} & \mathtt{1} \\ \mathtt{1} & \mathtt{100} & \mathtt{00} \end{pmatrix}$$

に対しては，列

$$\overrightarrow{i} = (1, 3, 1, 1, 3, 2, 2)$$

により，

$$\alpha_{i_1}\alpha_{i_2}\alpha_{i_3}\alpha_{i_4}\alpha_{i_5}\alpha_{i_6}\alpha_{i_7} = \mathtt{1001100100100}$$
$$\beta_{i_1}\beta_{i_2}\beta_{i_3}\beta_{i_4}\beta_{i_5}\beta_{i_6}\beta_{i_7} = \mathtt{1001100100100}$$

とできる．また，

$$\begin{pmatrix} \alpha_1 & \alpha_2 & \alpha_3 \\ \beta_1 & \beta_2 & \beta_3 \end{pmatrix} = \begin{pmatrix} \mathtt{1} & \mathtt{0} & \mathtt{1} \\ \mathtt{10} & \mathtt{100} & \mathtt{00} \end{pmatrix}$$

に対しては，どんな列 $i_1, \ldots, i_m$ を持ってきても，

$$\alpha_{i_1} \ldots \alpha_{i_m} = \beta_{i_1} \ldots \beta_{i_m}$$

とすることはできない．

ポストの対応問題は決定不能である [Pos46]．よって，文脈自由多生垣文法の重なり判定を解くことでポストの対応問題を解けるように，ポストの対応問題を文脈自由多生垣文法の重なり判定に埋めこむことができれば，文脈自由多生垣文法の重なり判定の決定不能性を示すことができる．

これには，二つの手法がある．一つは

$$
\begin{array}{ll}
T_\alpha \to (\alpha_1 \centerdot T_\alpha \centerdot \mathsf{l}_1) & T_\beta \to (\beta_1 \centerdot T_\beta \centerdot \mathsf{l}_1) \\
T_\alpha \to (\alpha_2 \centerdot T_\alpha \centerdot \mathsf{l}_2) & T_\beta \to (\beta_2 \centerdot T_\beta \centerdot \mathsf{l}_2) \\
\quad \cdots & \quad \cdots \\
T_\alpha \to \varepsilon & T_\beta \to \varepsilon
\end{array}
$$

と連接に埋めこむ方法である．そして，もう一つは，

$$
\begin{array}{ll}
T_\alpha \to (\mathit{labelify}(\alpha_1)[\pi_1 T_\alpha^{(1)}], \mathsf{l}_1(\pi_2 T_\alpha^{(1)})) & T_\beta \to (\mathit{laberify}(\beta_1)[\pi_1 T_\beta^{(1)}], \mathsf{l}_1(\pi_2 T_\beta^{(1)})) \\
T_\alpha \to (\mathit{labelify}(\alpha_2)[\pi_1 T_\alpha^{(1)}], \mathsf{l}_2(\pi_2 T_\alpha^{(1)})) & T_\beta \to (\mathit{laberify}(\beta_2)[\pi_1 T_\beta^{(1)}], \mathsf{l}_2(\pi_2 T_\beta^{(1)})) \\
\quad \cdots & \quad \cdots \\
T_\alpha \to (\varepsilon, \varepsilon) & T_\beta \to (\varepsilon, \varepsilon) \\
\mathit{laberify}(\varepsilon) \qquad\qquad\quad = \square_1 & \\
\mathit{laberify}(a :: \mathit{Char} \centerdot y) = \mathsf{L}_a(\mathit{laberify}(y)) &
\end{array}
$$

と同期に埋めこむ方法 [Fül94] である．たたし，l_i や L_a は添字により異なるラベルであるとする．どちらの場合にも，$[\![T_\alpha]\!] \cap [\![T_\beta]\!] = \emptyset$ かどうか判定することによって，ポストの対応問題が解けてしまうため，文脈自由多生垣文法の重なり判定問題の決定不能を結論づけられる．

よって，これまでの議論から以下が言える．

定理 8.3. 言語 $\text{V}\textsc{dl}^+$ で記述されたプログラムについて，二つの式の値域に重なりがあるかどうかは一般には決定不能である． □

上記定理は

- プログラムが多返値関数を含むが，連接を含まないとき
- プログラムが連接を含むが，多返値関数を含まないとき

のいずれの場合も同様である．なお，プログラムが多返値関数も連接も含まない場合，プログラムは式の型を表現する文脈自由多生垣文法は正規生垣文法になるので，式の値域の重なり判定は決定可能である．また，上の定理は，言語 $\text{V}\textsc{dl}^+$ で記述されたプログラムにおける二つの式の値域に重なりがあるかどうか判定するためには，

- 右辺における連接
- 多返値関数

のどちらにも何らかの対策が必要であることを示唆している．

また，言語 $\mathrm{V{\scriptsize DL}}^{+}$ で記述されたプログラムに対し，厳密な単射性判定はできないことも同様にして確認できる．

直接利用することはできないにも関わらず，式の値域を厳密に求めたのは，重なりの判定を効果的に行うために，何を近似する必要があるのかを明らかにするためである．

8.1.2　値域の近似

効果的に重なりの判定を行うために，我々は関数呼出式の値域を正規生垣言語により近似したのち，他の式の値域を近似された関数呼出式の値域に従い近似する．前述のように，文脈自由多生垣文法における以下の二つが，式の値域の重なり判定を行う際に問題になる．

- 多返値関数に相当する同期．
- 連接に相当する文脈自由性．

よって，我々はそれぞれに対応して文法を近似する．

我々は以下の二ステップにより型の近似を行う．

1. 同期の近似
    - 文脈自由多生垣文法を文脈自由生垣文法で近似する．
2. 文脈自由性の近似
    (a) Morhi と Nederhof の近似手法 [MN01] の変種により，文脈自由生垣文法を強正規文脈自由生垣文法で近似する．
    (b) 強正規文脈自由生垣文法を等価な正規生垣文法に変換する．

文脈自由多生垣文法の文脈自由生垣文法による近似

文脈自由多生垣文法をその文法を近似する文脈自由生垣文法への変換は非常に単純であり，文脈自由多生垣文法の各規則

$$A \to (\mathcal{C}_1[\pi_{j_{11}} A_{11}^{(i_{11})}, \ldots, \pi_{j_{1n_1}} A_{1n_1}^{(i_{1n_1})}], \ldots, \mathcal{C}_m[\pi_{j_{m1}} A_{m1}^{(i_{m1})}, \ldots, \pi_{j_{mn_m}} A_{mn_m}^{(i_{1n_m})}])$$

を文脈自由生垣文法の規則

$$\begin{aligned} \pi_1 A &\to \mathcal{C}_1[\pi_{j_{11}} A_{11}, \ldots, \pi_{j_{1n_1}} A_{1n_1}] \\ &\vdots \\ \pi_m A &\to \mathcal{C}_m[\pi_{j_{m1}} A_{m1}, \ldots, \pi_{j_{mn_m}} A_{mn_m}] \end{aligned}$$

へと変換することで達成できる．

たとえば，例 8.2 の文法を文脈自由生垣文法で近似すると以下となる．

$$
\begin{array}{lll}
\pi_1 T_{toc} & \rightarrow & \texttt{<ul>}(\pi_1 T_f) \\
\pi_2 T_{toc} & \rightarrow & \pi_2 T_f \\
\pi_1 T_f & \rightarrow & \varepsilon \\
\pi_1 T_f & \rightarrow & \texttt{<li>}(String) \centerdot \texttt{<ul>}(\pi_1 T_g) \centerdot \pi_1 T_f \\
\pi_2 T_f & \rightarrow & \varepsilon \\
\pi_2 T_f & \rightarrow & \texttt{<h1>}(String) \centerdot P \centerdot \pi_2 T_g \centerdot \pi_2 T_f \\
\pi_1 T_g & \rightarrow & \varepsilon \\
\pi_1 T_g & \rightarrow & \texttt{<li>}(String) \centerdot \pi_1 T_g \\
\pi_2 T_g & \rightarrow & \varepsilon \\
\pi_2 T_g & \rightarrow & \texttt{<h2>}(String) \centerdot P \centerdot \pi_2 T_g \\
P & \rightarrow & \texttt{<p>}(String)
\end{array}
$$

定理 8.4. 文脈自由多生垣文法 G を近似して文脈自由文法 G' が得られたとする．このとき，G 中の n 返り値非終端記号 T について，

$$(\pi_1 T^{()}, \ldots, \pi_n T^{()}) \xrightarrow{*}_G (v_1, \ldots, v_n) \Rightarrow \forall i \in \{1, \ldots, n\}.\ \pi_i T \xrightarrow{*}_{G'} v_i$$

となる．

証明. $\rightarrow_G \subseteq \rightarrow_{G'}$ であることにより明らか． □

Mohri と Nederhof の近似

次に得られた文脈自由生垣文法を正規生垣文法で近似することを考える．我々は，Mohri と Nederhof の近似手法 [MN01] の生垣版を使用することによりこれを行う．Mohri と Nederhof の近似手法 [MN01] は，元々は，文字列上の文脈自由文法を強正規（strongly-regular）文脈自由文法で近似する手法である．強正規な文法は，簡単かつ効果的に正規文法に変換できる [MN01].

まず，強正規性を定義しておく．

定理 8.5 (強正規文脈自由生垣文法)**.** 文脈自由文法が**強正規**であるとは，全ての生成規則 $A \rightarrow h$ について，もし，$h = \mathcal{C}[B]$ な B について $B \xrightarrow{*} \mathcal{C}'[A]$ となるならば，h 中で B は連接の最右に位置することである．

非終端記号 A と B について，$A \xrightarrow{*} \mathcal{C}[B]$ であり，$B \xrightarrow{*} \mathcal{C}'[A]$ であるとき，A と B は**相互再帰的に定義されている**と言う．この概念を用いて，文脈自由生垣文法が強正規であるとは，全ての非終端記号 A に対し相互再帰的に定義される B が存在するなら，任意の $B \rightarrow h$ という形式の規則について，h に A 含まれるならば A は連接の最右に位置する，と言い換えることができる．たとえば，文法

$$A \rightarrow \mathtt{a}BA \qquad B \rightarrow \varepsilon \qquad B \rightarrow \mathtt{b}B$$

は強正規である．なぜなら，B は連接の最右に位置してはいないが，$B \xrightarrow{*} \mathcal{C}[A]$ とはならないためである．対し，文法

$$A \to \mathtt{a}B\mathtt{b} \qquad B \to A \qquad B \to \mathtt{b}B$$

は強正規ではない．これは，A と B は相互再帰的に定義されているにもかかわらず，B は A の規則において連接の最右に出現していないためである．

Mohri と Nederhof の近似手法は，強正規でない原因となっている規則に着目することで，強正規でない文法を強正規な文法で近似する．もし，A と B が相互再帰的に定義されていると，規則 $A \to \mathcal{C}[B \,\centerdot\, h]$ を含む文法は強正規ではない．彼らの手法は，元の規則 $A \to \mathcal{C}[B \,\centerdot\, h]$ を $A \to \mathcal{C}[B]$ に置き換え，h の部分を生成するために，新たな非終端記号 B^{R} を導入し規則 $B^{\mathrm{R}} \to h$ を追加する．また，このとき $B \xrightarrow{*} vB^{\mathrm{R}}$ と元の非終端が生成する生垣 v を生成した後 B^{R} を生成するように，B の規則を置き換える．たとえば，以下の文脈自由文法を考える．

$$A \to \mathtt{a}A\mathtt{b} \qquad A \to \varepsilon$$

彼らの近似手法により得られる文法は以下である．

$$A \to \mathtt{a}A \qquad A \to A^{\mathrm{R}} \qquad A^{\mathrm{R}} \to \mathtt{b}A^{\mathrm{R}}$$

元の文法において $[\![A]\!] = \{\mathtt{a}^n\mathtt{b}^n \mid n \in \mathbb{N}\}$ であったのが，近似後 $[\![A]\!] = \{\mathtt{a}^n\mathtt{b}^m \mid n, m \in \mathbb{N}\} = [\![\mathtt{a}^*\mathtt{b}^*]\!]$ となっている．別の例として，以下の文脈自由生垣文法を考える．

$$A \to \mathtt{<a>}(\mathtt{<b>}A\mathtt{<c>}) \,\centerdot\, \mathtt{<d>} \qquad A \to \varepsilon$$

生垣版の彼らの手法は以下を返す．

$$A \to \mathtt{<a>}(\mathtt{<b>}A) \,\centerdot\, \mathtt{<d>} \,\centerdot\, A^{\mathrm{R}} \qquad A \to A^{\mathrm{R}} \qquad A^{\mathrm{R}} \to \varepsilon \qquad A^{\mathrm{R}} \to \mathtt{<c>}$$

ここで，$A^{\mathrm{R}} \to \mathtt{<c>}A^{\mathrm{R}}$ でないのは，<c> は元の A の規則の最右に出現しているわけではないためである．

具体的な近似アルゴリズムを以下に示す．

アルゴリズム 8.2 (文脈自由生垣文法の強正規文脈自由生垣文法による近似)**.**

入力： 文脈自由生垣文法 G.

出力： 強正規文脈自由生垣文法 G'.

手続き：

1. 文法 G 中のそれぞれの非終端記号 A について，以下を繰り返す．
2. もし，A について，規則 $A \to h$ で
   - 非空な生垣 h' と A と相互再帰的に定義されている非終端記号 B について，$h = \mathcal{C}[Bh']$ と書ける．

ものが存在しないのなら，G 中の $A \to h$ という形の規則を，G' へ追加する．

3. 文法 G 中の全ての非終端記号 A について，$A^{\mathrm{R}} \to \varepsilon$ という規則を G' へ追加する．
4. 文法 G 中の全ての生成規則 $A \to h$ のうちステップ 2 の条件が成り立たないものについて, 以下のステップ 5–7 を繰り返す.
5. もし，h を $h = \mathcal{C}[B_1h_1, \ldots B_mh_m] \centerdot C_1g_1 \centerdot \ldots \centerdot C_ng_n$ と，条件
   - $\mathcal{C}$ 中でホールの右に非空な生垣が来ることはない.
   - $B_1, \ldots, B_m, C_1, \ldots, C_n$ はそれぞれ A と相互再帰的に定義されている.
   - $\mathcal{C}, h_1, \ldots, h_m, g_1, \ldots, g_1$ は，A と相互再帰的に定義される非終端記号を含まない.

   を満たすように書くことができるならば，生成規則

$$
\begin{array}{l}
A \to \mathcal{C}[B_1, \ldots, B_m]C_1 \\
C_1^{\mathrm{R}} \to g_1C_2 \quad \ldots \quad C_i^{\mathrm{R}} \to g_iC_{i+1} \quad \ldots \quad C_{n-1}^{\mathrm{R}} \to g_{n-1}C_n \quad C_n^{\mathrm{R}} \to g_nA^{\mathrm{R}}
\end{array}
$$

   を G' へ追加し，ステップ 7 へ進む.
6. もし，h を $h = \mathcal{C}[B_1h_1, \ldots, B_mh_m]$ と，条件
   - $\mathcal{C}$ 中でホールの右に非空な生垣が来ることはない.
   - $B_1, \ldots, B_m$ はそれぞれ A と相互再帰的に定義されている.
   - $\mathcal{C}, h_1, \ldots, h_m$ は，A と相互再帰的に定義される非終端記号を含まない.

   を満たすように書くことができるならば，生成規則

$$A \to \mathcal{C}[B_1, \ldots, B_m]A^{\mathrm{R}}$$

   を G' に追加し，ステップ 7 へ進む.
7. ステップ 5 もしくは 6 における，それぞれの B_i と h_i（$i \in \{1, \ldots, m\}$）について，以下の補助手続きを適用する.

補助手続き：（入力：B, h）：

1'. 文脈 $\mathcal{C}'$ を以下の条件を満たすように定める.
   - $h = \mathcal{C}[D_1f_1, \ldots, D_lf_l]$.
   - $\mathcal{C}'$ でホールの右に非空な生垣が来ることはない.
   - $D_1, \ldots, D_l$ は，それぞれ B と相互再帰的に定義されている.
   - $\mathcal{C}, f_1, \ldots, f_l$ は，B と相互再帰的に定義される非終端記号を含まない.

2'. 規則

$$B^{\mathrm{R}} \to \mathcal{C}[D_1^{\mathrm{L}}, \ldots, D_l^{\mathrm{L}}]$$

   を G' に追加する.

3'. この補助手続きをそれぞれの D_i と f_i（$i \in \{1,\ldots,l\}$）について適用する． □

なお，上のアルゴリズムにおける補助手続きは，元の文字列版の Mohri と Nederhof の近似手法 [MN01] には含まれないものである．この補助手続きは，前述の生垣版の近似の例において，生垣 <c> は元の A の規則の最右に出現しているわけではないため，$A^{\mathrm{R}} \to$ <c>A^{R} ではなく $A^{\mathrm{R}} \to$ <c> が追加されていることに相当する．

補題 8.1. 上のアルゴリズムにより，文脈自由生垣文法 G から強正規文脈自由生垣文法 G' が得られたとする．このとき，全ての G 中の非終端記号 A について，もし，A がステップ 2 の条件を満たすならば，

$$A \xrightarrow{*}_G v \Rightarrow A \xrightarrow{*}_{G'} v$$

そうでないなら

$$A \xrightarrow{*}_G v \Rightarrow A \xrightarrow{*}_{G'} vA^{\mathrm{R}}$$

となる．

証明. 帰納法により示す.

基底：$A \to_G v$.

このとき，

$$A \to v$$

という規則が G 中に存在する．このとき，

$$A \to v$$

という規則か

$$A \to vA^{\mathrm{R}}$$

という規則が G' 中に存在する．ここで，$A^{\mathrm{R}} \to \varepsilon$ という規則が G' 中に存在するので，

$$A \xrightarrow{*}_{G'} v$$

となる．

帰納：$A \to_G h \xrightarrow{*}_G v$.

A がステップ 2 の条件を満たす場合は明らかでなので，そうでない場合を考える．
まず，h がステップ 5 の条件を満たすように，

$$h = \mathcal{C}[B_1 h_1, \ldots B_m h_m] \centerdot C_1 g_1 \centerdot \ldots \centerdot C_n g_n$$

と書くことができた場合を考える．このとき，生垣 $w_1, \ldots, w_m, v_1, \ldots, v_n$ が存在して，$v = \mathcal{C}[w_1, \ldots, w_m] \cdot v_1 \cdot \ldots \cdot v_n$ かつ，各 $i \in \{1, \ldots, m\}$ について $B_i h_i \xrightarrow{*} w_i$ であり各 $j \in \{1, \ldots, n\}$ について $C_j g_j \xrightarrow{*} v_j$ である．ここで，$C_j \xrightarrow{*}_G x_j$ かつ $g_j \xrightarrow{*}_G y_j$ な x_j と y_j について，$v_j = x_j \cdot y_j$ と書ける．このとき，G' は規則

$$\begin{array}{l} A \to \mathcal{C}'[B_1, \ldots, B_m] C_1 \\ C_1^{\mathrm{R}} \to g_1 C_2 \quad \ldots \quad C_j^{\mathrm{R}} \to g_j C_{j+1} \quad \ldots \quad C_{n-1}^{\mathrm{R}} \to g_{n-1} C_n \quad C_n^{\mathrm{R}} \to g_n A^{\mathrm{R}} \end{array}$$

を含む．帰納法の仮定より，$g_j \xrightarrow{*}_{G'} y_j$ としてよい．これは，仮に $g_j \xrightarrow{*}_{G'} y_j X^{\mathrm{R}}$ となり X^{R} が残っても，$X^{\mathrm{R}} \to \varepsilon$ という規則が G' に含まれるためである．これにより，帰納法の仮定 $C_i \xrightarrow{*} x_i C_j^{\mathrm{R}}$ より

$$\begin{aligned} C_1 &\xrightarrow{*}_{G'} x_1 C_i^{\mathrm{R}} \xrightarrow{*} x_1 g_1 C_2 \\ &\xrightarrow{*}_{G'} x_1 y_1 x_2 C_2^{\mathrm{R}} \\ &\xrightarrow{*}_{G'} x_1 y_1 \ldots x_{n-1} y_{n-1} x_n C_n^{\mathrm{R}} \\ &\xrightarrow{*}_{G'} x_1 y_1 \ldots x_{n-1} y_{n-1} x_n g_n A^{\mathrm{R}} \\ &\xrightarrow{*}_{G'} x_1 y_1 \ldots x_{n-1} y_{n-1} x_n y_n A^{\mathrm{R}} \\ &= v_1 \ldots v_n A^{\mathrm{R}} \end{aligned}$$

となる．

あとは，

$$B_i \xrightarrow{*} w_i \qquad (i \in \{1, \ldots, m\})$$

を示せばよい．ここで，$B_i \xrightarrow{*}_G z_i$, $h_i \xrightarrow{*}_G u_i$ となる z_i と u_i について，$w_i = z_i \cdot u_i$ と書ける．帰納法の仮定より，$B_i \xrightarrow{*}_{G'} z_i B_i^{\mathrm{R}}$ がいえるので，$B_i^{\mathrm{R}} \xrightarrow{*}_{G'} u_i$ を示せばよい．これを示すのに，h_i の構造に対する帰納法を用いる．ここで，$h_i = \mathcal{C}_i[D_{i1} f_{i1}, \ldots, D_{il} f_{il}]$ と書けたとする．もし，$l = 0$ ならば，生成規則

$$B_i^{\mathrm{R}} \to \mathcal{C}_i[]$$

を G' が持つ．関係 $(\xrightarrow{*}_G)$ に対する帰納法の仮定より，$\mathcal{C}_i[] \xrightarrow{*}_G u_i$ より，$\mathcal{C}_i[] \xrightarrow{*}_{G'} u_i$ としてよい．もし，$l = 0$ ならば，生成規則

$$B_i^{\mathrm{R}} \to \mathcal{C}_i[D_{i1}, \ldots, D_{il}]$$

を持つ．このとき，h_i の構造に対する帰納法の仮定より，

$$f_{ik} \xrightarrow{*}_{G'} s_{ik} \Rightarrow D_{ik}^{\mathrm{R}} \to_{G'} s_{ik}$$

そして，関係 $(\xrightarrow{*}_G)$ に対する帰納法の仮定より

$$D_{ik} \xrightarrow{*}_{G'} t_{ik} \Rightarrow D_1 \xrightarrow{*} t_{ik} D_{il}^{\mathrm{R}}$$

となる．よって，

$$B_i^{\mathrm{R}} \xrightarrow{*}_{G'} u_i$$

である．

ここまでの議論より，

$$A \xrightarrow{*}_{G'} \mathcal{C}[w_1, \ldots, w_m]v_1 \ldots v_n \xrightarrow{*}_{G'} vA^{\mathrm{R}}$$

が言える．これにより，h がステップ 5 の条件を満たす場合には，主張は真．

あとは h がステップ 6 の条件を満たすように，

$$h = \mathcal{C}[B_1 h_1, \ldots B_m h_m]$$

と書くことができた場合であるが，これも上と同様にして示すことができる．□

定理 8.6. 上のアルゴリズムにより，文脈自由生垣文法 G から強正規文脈自由生垣文法 G' が得られたとする．このとき，全ての G 中の非終端記号 A について，

$$A \xrightarrow{*}_{G} v \Rightarrow A \xrightarrow{*}_{G'} v$$

となる．

証明. 補題 8.1 より明らか．□

なお，アルゴリズムの定義より明らかであるが，元の文法の全ての生成規則がステップ 2 の条件を満たすときには，アルゴリズムは基本的には元の文法を変更しない．正確には，$A^{\mathrm{R}} \to \varepsilon$ という規則が追加されているものの，A^{R} は我々が興味のある非終端記号から生成されることがない．ステップ 2 の条件は，元のプログラムにおいて，相互再帰的に定義される関数の関数呼出が，常に連接の右端にあることに相当する．たとえば，第 6 章における関数 $c2x$ のプログラムは，特化されており，多返値関数を含まず，相互再帰的に定義される関数を含まないため，近似後の文法も関数呼出式の値域を厳密に記述している．

強正規文脈自由生垣文法の正規生垣文法への変換

最後に強正規文脈自由生垣文法を等価な正規生垣文法に変換することで，文脈自由多生垣文法の正規生垣文法により近似は完了する．この変換は，treeless 関数の融合変換（fusion）[Wad90] と同様の手法で行える．また，同様の変換は，XDuce においても，ユーザの記述した型を内部表現に変換するために用いられている [Hos03]．

強正規文脈自由生垣文法の正規生垣文法への変換は，正規生垣文法の生成規則を（有向）グラフと見るとわかりやすい．すなわち，$A \to \sigma(B)C$ という生成規則を，$\sigma_{\mathrm{sibling}}$ というラベルを持つ枝で節点 A と C がリンクされていて，$\sigma_{\mathrm{children}}$ というラベルを持つ枝で節点 A

と B がリンクされている，と見る．また，$A \to \varepsilon$ という規則を持つ節点とそうでない節点を区別する．正規生垣文法には対応するグラフがあるので，強正規文脈自由生垣文法の正規生垣文法への変換は，強正規文脈自由生垣文法の各非終端記号をグラフにおきかえ生成規則に従い接合し正規生垣文法に対応するグラフを構成していく．下のアルゴリズムにおいて，未出の非終端記号の導入は節点の導入に対応する．また辞書は，ある節点 X に対応するグラフを作成するのにあたって，そのグラフの作成中に X に対応するグラフを再び生成することなく X への枝を生成するために使用される．

アルゴリズム 8.3 (強正規文脈自由生垣文法の正規生垣文法への変換).

入力： 強正規文脈自由生垣文法 $G = (\Sigma, N, R)$.

出力： 正規生垣文法 $G' = (\Sigma, N', R')$.

手続き：

1. $R' := \emptyset$.
2. $N' := N$.
3. 辞書 Dic を用意する．始めは Dic は空.
4. G 中のそれぞれの規則 $A \to h$ について，以下で定まる手続き $f(h, A)$ を適用する．以下で X, Y は未出の非終端記号を表すとする．

$$
\begin{array}{l}
f(h,A) = \\
\quad \text{if } h = \varepsilon \text{ then} \\
\quad\quad R' := R' \cup \{A \to \varepsilon\} \\
\quad \text{if } h = B \text{ then} \\
\quad\quad R' := R' \cup \{A \to B\} \\
\quad \text{if } h = \sigma(h') \text{ then} \\
\quad\quad f(\sigma(h') \centerdot \varepsilon, A) \\
\quad \text{if } h = \sigma(h_1) \centerdot h_2 \text{ then} \\
\quad\quad f(h_1, X); \\
\quad\quad f(h_2, Y); \\
\quad\quad N' := N' \cup \{X, Y\}; \\
\quad\quad R' := R' \cup \{A \to \sigma(X)Y\} \\
\quad \text{if } h = B \centerdot h_2 \text{ then} \\
\quad\quad \text{if } B \centerdot h_2 \text{ を } Dic \text{ が含む.} \\
\quad\quad\quad N' := N' \cup Dic[B \centerdot h_2]; R' := R' \cup \{A \to D[B \centerdot h_2]\} \\
\quad\quad \text{else} \\
\quad\quad\quad Dic[B \centerdot h_2] := X; \\
\quad\quad\quad R' := R' \cup \{A \to X\}; \\
\quad\quad\quad \text{for each } h \text{ of } \ (B \to h) \in R \text{ do} \\
\quad\quad\quad\quad f(h \centerdot h_2, X)
\end{array}
$$

5. $G' = (\Sigma, N', R')$ とする． □

上の手続きの動作を説明するのに以下の強正規文脈自由生垣文法を用いる．

$$A \to B\mathtt{b} \qquad B \to \varepsilon \qquad B \to \mathtt{a}B$$

まず，最初の状態では，$Dic = \emptyset$，$N' = N$，$R' = \emptyset$ である．ここで規則 $A \to B\mathtt{b}$ について手続き呼び出し $f(B\mathtt{b}, A)$ が実行される．生垣 $B\mathtt{b}$ は辞書 Dic に登録されていないので，未出の識別子 X を導入し，$Dic = \{B\mathtt{b} \to X\}$ とし $R' = \{A \to X\}$ とする．そして，B を展開し，$f(\mathtt{b}, X)$ および $f(\mathtt{a}B\mathtt{b}, X)$ を実行する．どちらから実行しても結果は同等であるが，仮に最初のものから実行したとする．手続き呼び出し $f(\mathtt{b}, X)$ の結果，規則 $X \to \mathtt{b}$ が追加され，規則は $R' = \{A \to X, X \to \mathtt{b}\}$ となる．手続き呼び出し $f(\mathtt{a}B\mathtt{b}, X)$ を実行を考える．ここで，未出の識別子 Y を導入し，生成規則 $X \to \mathtt{a}Y$ を追加し，$f(B\mathtt{b}, Y)$ を実行する．ここで，$B\mathtt{b}$ は辞書 Dic に登録されているため，生成規則 $Y \to X$ を追加する．これで，$A \to B\mathtt{b}$ に対する規則生成は終了したので，$B \to \varepsilon$ と $B \to \mathtt{a}B$ のそれぞれに対し，$f(\varepsilon, B)$，$f(\mathtt{a}B, B)$ を実行する．最終的には，以下の文法が得られる．

$$A \to X \quad X \to \mathtt{b} \quad X \to \mathtt{a}Y \quad Y \to X \quad B \to \varepsilon \quad B \to \mathtt{a}Z \quad Z \to B$$

ここまでの議論から，各関数呼出式の近似された値域を正規生垣文法により記述することができる．残りの式の値域の近似は，規則

$$f(\overrightarrow{p}) \doteq \mathbf{let}\, \{(\overrightarrow{y_i}) \doteq f_i(\overrightarrow{x_i})\}_{i \in \{1,\ldots,n\}}\, \mathbf{in}\, (\overrightarrow{q})$$

中の式 q'（$\overrightarrow{q} = \overrightarrow{\mathcal{C}}[q']$）の近似された値域を，関数の近似された値域を表現する型環境

$$\Gamma := \Gamma_{\overrightarrow{p}} \cup \{y_{ij} \mapsto \pi_j T_{f_i}, i \in \{1,\ldots,n\}, j \in \{1,\ldots,|\overrightarrow{y_i}|\}\}$$

より，

$$\mathsf{ran}_\Gamma(q)$$

で与えることにより，関数呼出式の近似された値域から自然に定めることができる．返り値の数 n 個の関数 f の値域と近似された値域に対し，

$$\mathsf{ran}(f) \subseteq [\![\pi_1 T_f]\!] \times \cdots \times [\![\pi_n T_f]\!]$$

となることに注意する．また，これは式の値域について同様である．等号が成立する十分条件は，プログラムが次の二つをともに満たすことである．

- 多返値関数を含まない．
- 相互再帰的に定義される関数の関数呼出が必ず連接の右端に現れる．

8.1.3 単射性判定

ここでは，式の近似された値域に基づき，順方向変換の単射性を健全に判定することができることを述べる．単射だと判定された関数については，もっとも小さい補関数を得ることができ，もっとも効果的な逆方向変換を得ることができる．そのため，単射性の判定は効果的な逆方向変換を導出するために重要である．

集合 $S_1 \times S_2$ の上で，演算子「$\centerdot$」が単射となる条件を考える．もし，演算子「$\centerdot$」が単射でない場合，またそのときに限り

$$\exists (h_1, h_2), (h_3, h_4) \in (S_1 \times S_2).\ h_1 \centerdot h_2 = h_3 \centerdot h_4 \wedge h_1 \neq h_3 \wedge h_2 \neq h_3$$

が成り立つ．すなわち，$S_1 \parallel S_2$ あれば，$S_1 \times S_2$ の上で演算子「$\centerdot$」が単射であり，そうでなければ，演算子「$\centerdot$」が単射ではない．

よって，以下の三箇所に注目することにより関数の単射性判定を行うことができる．

- 未使用の変数
- 異なる規則の右辺式の値域の重なり
- 連接演算子の引数となる二つの式の値域の水平方向の重なり

集合 S_1 と T_1（$S_1 \subseteq T_1$）および S_2 と T_2（$S_2 \subseteq T_2$）について，以下が成り立つ．

$$S_1 \cap S_2 \neq \emptyset \Rightarrow T_1 \cap T_2 \neq \emptyset$$
$$S_1 \nparallel S_2 \Rightarrow T_1 \nparallel T_2$$

このため，近似された値域を用いても，我々は非単射な関数を必ず非単射を判定する手続きを構成することができる．

プログラム $\mathcal{P} = (_, Q, _)$ に対し，図 8.1 に定める関係の最小不動点により，$\mathcal{P}$ 中の単射な関数を全て含む集合 $\mathit{NINJ}_{\mathcal{P}}$ を定める．直観的には，図 8.1 の関係は，上の三個所のいずれかを含む関数は非単射であることを示している．明らかに単射な関数の集合 $\mathit{INJ}_{\mathcal{P}}$ は，$\mathit{NINJ}_{\mathcal{P}}$ より $\mathit{INJ}_{\mathcal{P}} = Q \setminus \mathit{NINJ}_{\mathcal{P}}$ として定めることができる．図 8.1 の上の三つの規則は，第 5 章の図 5.3.3 の三つの規則と同様である．

定理 8.7 (単射性判定の健全性)**.** 集合 $\mathit{INJ}_{\mathcal{P}}$ に含まれる関数は全て単射である．

証明. 対偶，つまり，非単射な関数は全て $\mathit{NINJ}_{\mathcal{P}}$ に含まれることを示す．具体的には，

$$\exists \overrightarrow{v}, \overrightarrow{v}', u.\ \overrightarrow{v} \neq \overrightarrow{v'} \wedge f(\overrightarrow{v}) \Downarrow \overrightarrow{u} \wedge f(\overrightarrow{v}') \Downarrow \overrightarrow{u} \Rightarrow f \in \mathit{NINJ}_{\mathcal{P}} \qquad \text{(INJ-Sound}^{+}\text{)}$$

を示す．

今，ある $\overrightarrow{v}$，$\overrightarrow{v}'$ および u に対し，$f(\overrightarrow{v}) \Downarrow u$ かつ $f(\overrightarrow{v'}) \Downarrow u$ となったとする．このとき，以下の二つの場合しかない．

$$\frac{\exists f(\overrightarrow{p}) \doteq \overrightarrow{e} \in R.\ \mathsf{lostvars}(f(\overrightarrow{p}) \doteq \overrightarrow{e}) \neq \emptyset}{f \in \mathit{NINJ}_{\mathcal{P}}}\text{NINJ-LostVar}$$

$$\frac{\begin{array}{c}\exists f(\overrightarrow{p}) \doteq \mathbf{let}\ \{(\overrightarrow{y_i}) \doteq f_i(\overrightarrow{x_i})\}_{i\in\{1,\ldots,n\}}\ \mathbf{in}\ (\overrightarrow{q}) \in R,\\ \exists f(\overrightarrow{p}') \doteq \mathbf{let}\ \{(\overrightarrow{y_i}') \doteq f'_i(\overrightarrow{x_i}')\}_{i\in\{1,\ldots,n'\}}\ \mathbf{in}\ (\overrightarrow{q}') \in R.\\ \overrightarrow{p} \neq \overrightarrow{p}' \quad \mathsf{ran}_\Gamma(\overrightarrow{q}) \cap \mathsf{ran}_{\Gamma'}(\overrightarrow{q}') \neq \emptyset\end{array}}{f \in \mathit{NINJ}_{\mathcal{P}}}\text{NINJ-Overlap}$$

ただし，$\begin{cases}\Gamma := \Gamma_{\overrightarrow{p}} \cup \{y_{ij} \mapsto \pi_j T_{f_i}, i \in \{1,\ldots,n\}, j \in \{1,\ldots,|\overrightarrow{y_i}|\}\},\\ \Gamma' := \Gamma_{\overrightarrow{p}'} \cup \{y'_{ij} \mapsto \pi_j T'_{f'_i}, i \in \{1,\ldots,n'\}, j \in \{1,\ldots,|\overrightarrow{y_i}'|\}\}.\end{cases}$

$$\frac{\exists f(\overrightarrow{p}) \doteq \mathbf{let}\ \{(\overrightarrow{y_i}) \doteq f_i(\overrightarrow{x_i})\}_{i\in\{1,\ldots,n\}}\ \mathbf{in}\ (\overrightarrow{q}) \in R, \exists i.\ f_i \in \mathit{NINJ}_{\mathcal{P}}}{f \in \mathit{NINJ}_{\mathcal{P}}}\text{NINJ-Call}$$

$$\frac{\begin{array}{c}\exists f(\overrightarrow{p}) \doteq \mathbf{let}\ \{(\overrightarrow{y_i}) \doteq f_i(\overrightarrow{x_i})\}_{i\in\{1,\ldots,n\}}\ \mathbf{in}\ (\overrightarrow{C}[q_1 \centerdot q_2]) \in R.\\ \mathsf{ran}_\Gamma(q_1) \not\parallel \mathsf{ran}_\Gamma(q_1)\end{array}}{f \in \mathit{NINJ}_{\mathcal{P}}}\text{NINJ-HOverlap}$$

ただし，$\Gamma := \Gamma_{\overrightarrow{p}} \cup \{y_{ij} \mapsto \pi_j T_{f_i}, i \in \{1,\ldots,n\}, j \in \{1,\ldots,|\overrightarrow{y_i}|\}\}$.

図 8.1. プログラム $\mathcal{P} = (G, Q, R)$ より，非単射な関数を全て含む集合 $\mathit{NINJ}_{\mathcal{P}}$ を求める関係

場合 1． $f(\overrightarrow{v}) \Downarrow u$ の証明木と $f(\overrightarrow{v'}) \Downarrow u$ の証明木が変数への束縛を除いて等しい場合．つまり，二つの証明木がそれぞれの Fun 節点において同じ規則を用いている場合.

場合 2. それ以外.

場合 1 について，我々は主張 (INJ-Sound$^+$) を #FunDepth$(f(\overrightarrow{v}))$ についての帰納法により示す．ここで，#FunDepth(e) は，式 e を評価するのに使用した Fun の数を $e \Downarrow v$ の証明木に沿って縦に数えたもの最大値，つまり深さを表す.

基底：$f(\overrightarrow{v}) = f(\overrightarrow{v}') = 0$．

このとき，**let** 束縛を持たない規則

$$r = f(\overrightarrow{p}) \doteq (\overrightarrow{q})$$

で，$\overrightarrow{p}\theta = \overrightarrow{v}, \overrightarrow{p}\theta' = \overrightarrow{v}'$ となる代入 θ, θ' に対し，$\overrightarrow{q}\theta = \overrightarrow{u}$ かつ $\overrightarrow{q}\theta' = \overrightarrow{u}$ となるものが存在する．もし，$\mathsf{lostvars}(r) = \emptyset$ であったとする．すると $\mathsf{vars}(q) = \mathsf{vars}(p)$ かつ $\overrightarrow{q}\theta = \overrightarrow{q}\theta'$ より，$\overrightarrow{p}\theta = \overrightarrow{p}\theta'$ が言える．これは $\overrightarrow{v} \neq \overrightarrow{v}'$ に矛盾する．よって，$\mathsf{lostvars}(r) = \mathsf{vars}(\overrightarrow{p}) \backslash \mathsf{vars}(\overrightarrow{q}) \neq \emptyset$ である．これは，NINJ-LostVar の条件であるため，$f \in \mathit{NINJ}_{\mathcal{P}}$ となる.

帰納：$\#\textsc{FunDepth}(f(\overrightarrow{v})) = \#\textsc{FunDepth}(f(\overrightarrow{v'})) > 0$．

このとき，規則

$$r = f(\overrightarrow{p}) \mathrel{\hat{=}} \mathbf{let}\ \{(\overrightarrow{y_i}) \mathrel{\hat{=}} f_i(\overrightarrow{x_i})\}_{i \in \{1,\ldots,n\}} \ \mathbf{in}\ (\overrightarrow{q})$$

で，$\overrightarrow{p}\theta = \overrightarrow{v}, \overrightarrow{p}\theta' = \overrightarrow{v}'$ となる代入 θ, θ' に対し，$f(\overrightarrow{p}\theta) \Downarrow \overrightarrow{u}$ かつ $f(\overrightarrow{p}\theta') \Downarrow \overrightarrow{u}$ となるものが存在する．ここで，$\overrightarrow{q}$ が非単射な連接を含まないと仮定してよい．なぜなら，$\overrightarrow{q}$ が非単射な連接を含む場合 NINJ-HOverlap の条件より，$f \in \mathit{NINJ}_{\mathcal{P}}$ となるためである．ここで，$\overrightarrow{q} = \overrightarrow{\mathcal{C}}[\overrightarrow{y_1}, \ldots, \overrightarrow{y_n}, \overrightarrow{x}]$ とおく．このとき，$\overrightarrow{q}$ の連接は単射であるため，$\overrightarrow{u} = \overrightarrow{\mathcal{C}}[\overrightarrow{w_1}, \ldots, \overrightarrow{w_n}, \overrightarrow{x}\theta] = \overrightarrow{\mathcal{C}}[\overrightarrow{w_1}, \ldots, \overrightarrow{w_n}, \overrightarrow{x}\theta']$ と一意に書ける．このとき，$\overrightarrow{w_i}$（$i \in \{1, \ldots, n\}$）は $f_i(\overrightarrow{x_i}\theta)$ および $f_i(\overrightarrow{x_i}\theta')$ の評価結果である．すなわち，$f_i(\overrightarrow{x_i}\theta) \Downarrow \overrightarrow{w_i}$ であり $f_i(\overrightarrow{x_i}\theta') \Downarrow \overrightarrow{w_i}$ である．もし $\mathsf{lostvars}(r) \neq \emptyset$ ならば，NINJ-LostVar の条件より，$f \in \mathit{NINJ}_{\mathcal{P}}$ となる．よって，$\mathsf{lostvars}(r) = \emptyset$ である場合を考える．ここで，$\overrightarrow{p}\theta \neq \overrightarrow{p}\theta'$ かつ $\overrightarrow{x}\theta = \overrightarrow{x}\theta'$ であるため，$\overrightarrow{x_i}\theta \neq \overrightarrow{x_i}\theta'$ となる i が存在する．ただし，$f_i(\overrightarrow{x_i}\theta)$ と $f_i(\overrightarrow{x_i}\theta')$ の評価結果は一致するため，f_i は非単射である．このとき，帰納法の仮定より，$f_i \in \mathit{NINJ}_{\mathcal{P}}$ である．これは，NINJ-Call の条件であるため，$f \in \mathit{NINJ}_{\mathcal{P}}$ となる．

場合 2 に対し，我々は主張 (INJ-Sound$^+$) を $\#\textsc{FunDepth}(f(\overrightarrow{v})) + \#\textsc{FunDepth}(f(\overrightarrow{'v}))$ についての帰納法で示す．

基底：$f(\overrightarrow{v}) + f(\overrightarrow{v}') = 0$．

このとき，**let** 束縛を持たない二つの規則

$$r = f(\overrightarrow{p}) \mathrel{\hat{=}} (\overrightarrow{q})$$
$$r' = f(\overrightarrow{p}') \mathrel{\hat{=}} (\overrightarrow{q}')$$

で，$\overrightarrow{p}\theta = \overrightarrow{v}, \overrightarrow{p}'\theta' = \overrightarrow{v}'$ となる代入 θ, θ' に対し，$\overrightarrow{q}\theta = \overrightarrow{u}$ かつ $\overrightarrow{q}'\theta' = \overrightarrow{u}$ となるものが存在する．これは，$\mathsf{ran}_{\Gamma_{\overrightarrow{p}}}(\overrightarrow{q}) \cap \mathsf{ran}_{\Gamma_{\overrightarrow{p}'}}(\overrightarrow{q}') \neq \emptyset$ であることに他ならないため，NINJ-Overlap の条件より，$f \in \mathit{NINJ}_{\mathcal{P}}$ となる．

帰納：$\#\textsc{FunDepth}(f(\overrightarrow{v})) > 0, \#\textsc{FunDepth}(f(\overrightarrow{v}')) = 0$ **もしくは** $\#\textsc{FunDepth}(f(\overrightarrow{v})) = 0, \#\textsc{FunDepth}(f(\overrightarrow{v}')) > 0$．

この場合も使用した二つの規則が異なることから，上と同様に，主張 (INJ-Sound$^+$) は証明される．

帰納：$\#\textsc{FunDepth}(f(\overrightarrow{v})) > 0, \#\textsc{FunDepth}(f(\overrightarrow{v'})) > 0$．

このとき，以下の二つの場合がある．

- 関数 f の二つの異なる規則で，その左辺 $\overrightarrow{p}, \overrightarrow{p}'$ について，ある θ, θ' に対し，$\overrightarrow{p}\theta = \overrightarrow{v}$ かつ $\overrightarrow{p'}\theta' = \overrightarrow{v'}$ となるものが存在する．
- 評価 $f(\overrightarrow{v}) \Downarrow \overrightarrow{u}$ と $f(\overrightarrow{v'}) \Downarrow \overrightarrow{u}$ との証明木において，それぞれの根で同じ規則を使用している．

第一の場合については，二つの規則が異なるため，これまでと同様の議論により主張 (INJ-Sound$^+$) を証明できる．よって，第二の場合について考える．このとき，規則

$$\begin{aligned} r = f(\overrightarrow{p}) \doteq\ & \mathbf{let}\,\{(\overrightarrow{y_i}) \doteq f_i(\overrightarrow{x_i})\}_{i\in\{1,\ldots,n\}} \\ & \mathbf{in}\ (\overrightarrow{q}) \end{aligned}$$

で，$\overrightarrow{p}\theta = \overrightarrow{v}, \overrightarrow{p}\theta' = \overrightarrow{v}'$ となる代入 θ, θ' に対し，$f(\overrightarrow{p}\theta) \Downarrow \overrightarrow{u}$ かつ $f(\overrightarrow{p}\theta') \Downarrow \overrightarrow{u}$ となるものが存在する．ここで，$\overrightarrow{q}$ が非単射な連接を含まないと仮定してよい．なぜなら，$\overrightarrow{q}$ が非単射な連接を含む場合 NINJ-H$\textsc{Overlap}$ の条件より，$f \in \mathit{NINJ}_{\mathcal{P}}$ となるためである．ここで，$\overrightarrow{q} = \overrightarrow{C}[\overrightarrow{y_1}, \ldots, \overrightarrow{y_n}, \overrightarrow{x}]$ とおく．このとき，$\overrightarrow{q}$ の連接は単射であるため，$\overrightarrow{u} = \overrightarrow{C}[\overrightarrow{w_1}, \ldots, \overrightarrow{w_n}, \overrightarrow{x}\theta] = \overrightarrow{C}[\overrightarrow{w_1}, \ldots, \overrightarrow{w_n}, \overrightarrow{x}\theta']$ と一意に書ける．このとき，$\overrightarrow{w_i}$ $(i \in \{1, \ldots, n\})$ は $f_i(\overrightarrow{x_i}\theta)$ および $f_i(\overrightarrow{x_i}\theta')$ の評価結果である．すなわち，$f_i(\overrightarrow{x_i}\theta) \Downarrow \overrightarrow{w_i}$ であり $f_i(\overrightarrow{x_i}\theta') \Downarrow \overrightarrow{w_i}$ である．仮定より，どれかの i について，$f_i(\overrightarrow{x_i}\theta)$ と $f_i(\overrightarrow{x_i}\theta')$ の証明木は異なる．これは，また $\overrightarrow{x_i}\theta \neq \overrightarrow{x_i}\theta'$ であることを意味する．すなわち，f_i は単射ではない．このとき，帰納法の仮定により，NINJ-C$\textsc{all}$ から，$f \in \mathit{NINJ}_{\mathcal{P}}$ である．□

近似した値域を用いているため単射性判定は完全ではない．つまり，単射であるが $\mathit{INJ}_{\mathcal{P}}$ に含まれない関数が存在しうる．しかし，定理 8.7 単射性判定は健全ではあるため，$\mathit{INJ}_{\mathcal{P}}$ に含まれる関数は，全て単射である．第 5 と同様の議論により，特化されたプログラム $\mathcal{P}$ に対し，もし，全ての式に対し式の近似された値域と式の値域が一致するならば，$\mathit{INJ}_{\mathcal{P}}$ が全ての単射関数を含むことが言える．

8.1.4　補関数導出

言語 V$\textsc{dl}^+$ で記述された関数に対する補関数導出は，第 5 章のアルゴリズム A$\textsc{lg}$-S$\textsc{c}$ とほぼ同様である．ただし，言語 V$\textsc{dl}^+$ のプログラムは次の関数 g のように非単射な連接を含む場合がある．

$$\begin{aligned} &\mathbf{data}\ A \doteq \texttt{<a>}^* \\ &\mathbf{data}\ B \doteq \texttt{<b>}^* \\ &f(x :: A, y :: B) \doteq x \centerdot y \\ &g(x :: A, y :: A) \doteq x \centerdot y \end{aligned}$$

関数 g のような非単射な連接を含む関数に対しては

$$g^{\mathrm{c}}(x :: A, y :: A) \mathrel{\hat{=}} \textsc{length}(x)$$

と非単射な連接の第一引数の長さを補関数に含めることにより，補関数を作成する．ただし，関数は複数個の非単射な連接を含みうるため．式の識別子を用い，どの LENGTH がどの連接に対応するのかを区別する．たとえば,

$$\begin{array}{l} \textbf{data}\ A \mathrel{\hat{=}} \texttt{<a>}^* \\ g(x :: A, y :: A, z :: A, w :: A) \mathrel{\hat{=}} \texttt{<a>}(x \centerdot y) \centerdot \texttt{<b>}(z \centerdot w) \end{array}$$

に対し，もし，$\#(x \centerdot y) = 1, \#(z \centerdot w) = 2$ とすると，本章の補関数導出は以下の補関数を求める．

$$g^{\mathrm{c}}(x :: A, y :: A, z :: A, w :: A) \mathrel{\hat{=}} \mathsf{B}_1(\textsc{length}_1(x), \textsc{length}_2(z))$$

を求める．簡便のためこれ以降，$e_1 \centerdot e_2$ の識別子が k である場合，$e_1 \centerdot_{(k)} e_2$ と書く．また，連接演算は結合的であるが，結合順により導出される補関数が異なりうる．たとえば，以下の f と g は，連接の結合を除いて等価な関数である．

$$\begin{array}{l} \textbf{data}\ A \mathrel{\hat{=}} \texttt{<a>}^*; \textbf{data}\ B \mathrel{\hat{=}} \texttt{<b>}^* \\ f(x :: A, y :: B, z :: B) \mathrel{\hat{=}} (x \centerdot y) \centerdot_{(1)} z \\ g(x :: A, y :: B, z :: B) \mathrel{\hat{=}} x \centerdot (y \centerdot_{(2)} z) \end{array}$$

しかし，連接を第一引数の長さの情報を補うようにして得られる補関数は以下のように異なる．

$$\begin{array}{l} f^{\mathrm{c}}(x :: A, y :: B, z :: B) \mathrel{\hat{=}} \mathsf{B}_1(\textsc{length}_1(x \centerdot y)) \\ g^{\mathrm{c}}(x :: A, y :: B, z :: B) \mathrel{\hat{=}} \mathsf{B}_2(\textsc{length}_2(x)) \end{array}$$

本論文において，補関数導出においては「$\centerdot$」は右結合演算子であると扱う．つまり,

$$e_1 \centerdot e_2 \centerdot e_3$$

と書くと

$$e_1 \centerdot (e_2 \centerdot e_3)$$

を意味する．

第 5 章のアルゴリズム ALG-SC と異なり，言語 VDL$^+$ で記述されたプログラムに対する本章の補関数の導出アルゴリズムは，規則の集合の分割をしない．これは，変数パターンを利用することにより，規則集合の分割と同等の補関数が得られる場合が多いためである．たとえば，以下の関数 pred を考える．

$$\begin{array}{ll} r_1 = \mathit{pred}(\texttt{<z>}) & \mathrel{\hat{=}} \texttt{<z>} \\ r_2 = \mathit{pred}(\texttt{<s>} \centerdot \texttt{<z>}) & \mathrel{\hat{=}} \texttt{<z>} \\ r_3 = \mathit{pred}(\texttt{<s>} \centerdot \texttt{<s>} \centerdot r) & \mathrel{\hat{=}} \texttt{<s>} \centerdot r \end{array}$$

第 5 章の ALG-SC と同様に，規則集合の分割 $\{r_1\} \uplus \{r_2, r_3\}$ を利用し補関数を導出すると，補関数

$$\begin{aligned} pred^{\mathrm{c}}(\texttt{<z>}) &\mathrel{\hat{=}} \mathsf{B}_1 \\ pred^{\mathrm{c}}(\texttt{<s>.<z>}) &\mathrel{\hat{=}} \mathsf{B}_2 \\ pred^{\mathrm{c}}(\texttt{<s>.<s>.}r) &\mathrel{\hat{=}} \mathsf{B}_2 \end{aligned}$$

が得られる．それに対し，以下のように定義された $pred$ を考える．

$$\begin{aligned} pred(\texttt{<z>}) &\mathrel{\hat{=}} \texttt{<z>} \\ pred(x :: (\mathsf{s}^{*}\texttt{.<s>.<z>})) &\mathrel{\hat{=}} pred'(x) \\ pred'(\texttt{<s>.<z>}) &\mathrel{\hat{=}} \texttt{<z>} \\ pred'(\texttt{<s>.<s>.}r) &\mathrel{\hat{=}} \texttt{<s>.}r \end{aligned}$$

ここで，関数 $pred'$ は単射だと判定することができるため，その情報を利用することで，規則集合を用いることなしに上と同等な補関数

$$\begin{aligned} pred^{\mathrm{c}}(\texttt{<z>}) &\mathrel{\hat{=}} \mathsf{B}_1 \\ pred^{\mathrm{c}}(x :: (\texttt{<s>}^{*}\texttt{.<s>.<z>})) &\mathrel{\hat{=}} \mathsf{B}_2 \end{aligned}$$

が得られる．

変数パターンを利用し別の関数定義を与えることにより，規則集合の分割により得られる補関数と同等な補関数が，得られるとは限らない．しかし，我々は，第 5 章の ALG-SC をいくつかの例に適用した結果以下の経験則を得たため，言語 VDL^{+} に対する補関数導出においては規則集合の分割を考えない．

- 規則集合の分割が有効な例はあまりない．それに対し，構成子の取り外しのほうが zip などのように有効に働く例が多い．
- 関数 $pred$ のように，規則集合の分割が有効な例に対しては，変数パターンを用いて関数定義を書き直すことで同等な補関数を得られる．
- ユーザにとって規則集合の分割の与え方が明確でない．

言語 VDL^{+} で記述された関数に対する補関数導出アルゴリズム ALG-C^{+} を以下に示す．導出される補関数の返り値は，木でも生垣でもなく,「生垣を葉として含む木」の列であることに注意する．

アルゴリズム 8.4 (補関数導出：ALG-C^{+})**.**

入力： 言語 VDL^{+} のプログラム $\mathcal{P}$

出力： 言語 VDL^{+} のプログラム $\mathcal{P}^{\mathrm{c}}$

手続き：

1. $\mathcal{P}$ 中のそれぞれの規則 r

$$r = f(\overrightarrow{p}) \mathrel{\hat{=}} \textbf{let}\,\{(\overrightarrow{y_i}) \mathrel{\hat{=}} f_i(\overrightarrow{x_i})\}_{i \in \{1,\ldots,n\}} \\ \textbf{in}\ (\overrightarrow{q})$$

   について，以下により補関数の規則 r^{c} を構成する．

(a) 各 $\overrightarrow{y_i}$ について，$\overrightarrow{y_i} = (y_{i1}, \ldots, y_{i|\overrightarrow{y_i}|})$ とおく．

(b) $\Gamma := \Gamma_{\overrightarrow{p}} \cup \{y_{ij} \mapsto \pi_j T_{f_i} \mid i \in \{1, \ldots, n\}, j \in \{1, \ldots, |\overrightarrow{y_i}|\}\}$ とする．

(c) $I := \{i \mid f_i \in \mathit{INJ}_{\mathcal{P}}\}$ とする．

(d) $Y := \bigcup_{i \in \{1,\ldots,n\}, i \neq I} \{\overrightarrow{y_i}\}$ とする．

(e) $V := \mathsf{lostvars}(r)$ とする．

(f) $\overrightarrow{q}' := (\mathsf{B}_{r1}(\overrightarrow{q_1{}^{\mathrm{c}}}, V), \ldots, \mathsf{B}_{rm}(\overrightarrow{q_m{}^{\mathrm{c}}}, V))$ とする．ただし，$\overrightarrow{q} = (q_1, \ldots, q_m)$ であり，以下の関係 $\overset{\mathrm{c}}{\leadsto}$ により $\Gamma, Y, q_i \overset{\mathrm{c}}{\leadsto} \overrightarrow{q_i{}^{\mathrm{c}}}$ であるとする．

$$\frac{}{\Gamma, Y, \varepsilon \overset{\mathrm{c}}{\leadsto} ()}\textsc{Eps} \quad \frac{x \notin Y}{\Gamma, Y, x \overset{\mathrm{c}}{\leadsto} ()}\textsc{Var-Inj} \quad \frac{y \in Y}{\Gamma, Y, y \overset{\mathrm{c}}{\leadsto} y^{\mathrm{c}}}\textsc{Var-NInj}$$

$$\frac{\Gamma, Y, e \overset{\mathrm{c}}{\leadsto} \overrightarrow{e^{\mathrm{c}}}}{\Gamma, Y, \sigma(e) \overset{\mathrm{c}}{\leadsto} \overrightarrow{e^{\mathrm{c}}}}\textsc{Con}$$

$$\frac{\mathsf{ran}_{\Gamma}(e_1) \nparallel \mathsf{ran}_{\Gamma}(e_2) \quad \Gamma, Y, e_1 \overset{\mathrm{c}}{\leadsto} \overrightarrow{e_1{}^{\mathrm{c}}} \quad \Gamma, Y, e_2 \overset{\mathrm{c}}{\leadsto} \overrightarrow{e_2{}^{\mathrm{c}}}}{\Gamma, Y, e_1 \centerdot_{(i)} e_2 \overset{\mathrm{c}}{\leadsto} \textsc{length}_i(e), \overrightarrow{e_1{}^{\mathrm{c}}}, \overrightarrow{e_2{}^{\mathrm{c}}}}\textsc{Cat}$$

$$\frac{\mathsf{ran}_{\Gamma}(e_1) \parallel \mathsf{ran}_{\Gamma}(e_2) \quad \Gamma, Y, e_1 \overset{\mathrm{c}}{\leadsto} \overrightarrow{e_1{}^{\mathrm{c}}} \quad \Gamma, Y, e_2 \overset{\mathrm{c}}{\leadsto} \overrightarrow{e_2{}^{\mathrm{c}}}}{\Gamma, Y, e_1 \centerdot e_2 \overset{\mathrm{c}}{\leadsto} \overrightarrow{e_1{}^{\mathrm{c}}}, \overrightarrow{e_2{}^{\mathrm{c}}}}\textsc{ICat}$$

(g) もし，$|\overrightarrow{q_1{}^{\mathrm{c}}}, V| = 1, \ldots, |\overrightarrow{q_m{}^{\mathrm{c}}}, V| = 1$，すなわち，$\mathsf{B}_{r1}, \ldots, \mathsf{B}_{rm}$ の引数の数が一であり，r の右辺式の近似された値域が，f の他の規則の右辺式の近似された値域と重なりがなければ，$\overrightarrow{q}' := ((\overrightarrow{q_1{}^{\mathrm{c}}}, V), \ldots, (\overrightarrow{q_m{}^{\mathrm{c}}}, V))$ とおきなおす．

(h) 規則 r^{c} を以下で得る．

$$\begin{aligned} r^{\mathrm{c}} = f^{\mathrm{c}}(\overrightarrow{p}) \mathrel{\hat{=}} \mathbf{let}\, & \{(\overrightarrow{y_i}^{\mathrm{c}}) \mathrel{\hat{=}} f_i{}^{\mathrm{c}}(\overrightarrow{x_i})\}_{i \in \{1,\ldots,n\}, i \neq I} \\ & \{(\overrightarrow{y_i}) \mathrel{\hat{=}} f_i(\overrightarrow{x_i})\}_{i \in \{1,\ldots,n\}} \\ \mathbf{in}\, & (\overrightarrow{q}') \end{aligned}$$

ただし，$\overrightarrow{y_i}^{\mathrm{c}} = (y_{i1}{}^{\mathrm{c}}, \ldots, y_{i|\overrightarrow{y_i}|}{}^{\mathrm{c}})$ とする．

2. 規則 r^{c} を集めてプログラム $\mathcal{P}^{\mathrm{c}}$ を構成する．　□

木の場合と異なり，生垣の場合は，$\mathcal{C}[x_1, x_2]\theta = h$ となる代入が一意には定まらない．たとえば，$\mathcal{C} = \square_1 \centerdot \square_2, h =$ <a> である場合に，$\mathcal{C}[$<a>$, \varepsilon] = \mathcal{C}[\varepsilon,$ <a>$] = h$ となる．アルゴリズム Alg-C$^+$ 正しさの証明において，この問題を回避するため以下の補題を用いる．補題 8.2 は，直感的には，もし，このような代入が複数個存在した場合には，それぞれ代入に対し補関数の返り値が異なることを示している．もう少し正確に言うと，型環境 Γ の元で e は非単射と判定される連接を含んでいる場合があるため，ある生垣 h に対し $e\theta = h$ となる代入 θ が複数個存在する場合があるが，変数集合 Y に対し $\Gamma, Y, e^{\mathrm{c}}$ となる e^{c} について，そのようなそれぞれの代入に対し e^{c} の取る値は互いに異なる．

補題 8.2. $\Gamma, Y, e \overset{\mathrm{c}}{\leadsto} \overrightarrow{e^{\mathrm{c}}}$ となったとする．このとき，$e\theta = e\theta'$ かつ $\forall x.\ \theta(x), \theta'(x) \in \Gamma(x)$ であったならば，$\exists z \in \mathsf{vars}(e).\ \theta(z) \neq \theta'(z) \Rightarrow \forall \eta, \eta'.\ \overrightarrow{e^{\mathrm{c}}}\theta\eta \neq \overrightarrow{e^{\mathrm{c}}}\theta'\eta'$ となる．

証明. 式 e の構造に関する帰納法により示す.

基底：$e = \varepsilon$ **もしくは** $e = x$**.**

条件 $e\theta = e\theta'$ のもとで，$\exists z \in \mathsf{vars}(e).\ \theta(z) \neq \theta'(z)$ となることはない．よって主張は真.

帰納：$e = \sigma(e')$**.**

帰納法の仮定より，直截.

帰納：$e = e_1 \centerdot e_2$**.**

まず，$\mathsf{ran}_\Gamma(e_1) \parallel \mathsf{ran}_\Gamma(e_2)$ である場合を考える．このとき，

$$(e_1 \centerdot e_2)\theta = (e_1 \centerdot e_2)\theta' \Rightarrow e_1\theta = e_1\theta' \wedge e_2\theta = e_2\theta'$$

となる．帰納法の仮定より，

$$\begin{array}{l} \forall \eta_1, \eta_1'.\ \overrightarrow{e_1}^{\mathrm{c}}\theta\eta_1 \neq \overrightarrow{e_1}^{\mathrm{c}}\theta'\eta_1', \\ \forall \eta_2, \eta_2'.\ \overrightarrow{e_2}^{\mathrm{c}}\theta\eta_2 \neq \overrightarrow{e_2}^{\mathrm{c}}\theta'\eta_2' \end{array}$$

となる．今，仮に

$$(\overrightarrow{e_1}^{\mathrm{c}}, \overrightarrow{e_2}^{\mathrm{c}})\theta\eta = (\overrightarrow{e_1}^{\mathrm{c}}, \overrightarrow{e_2}^{\mathrm{c}})\theta'\eta'$$

となったとする．このとき

$$\overrightarrow{e_1}^{\mathrm{c}}\theta\eta = \overrightarrow{e_1}^{\mathrm{c}}\theta'\eta' \wedge \overrightarrow{e_2}^{\mathrm{c}}\theta\eta = \overrightarrow{e_2}^{\mathrm{c}}\theta'\eta'$$

となり，帰納法の仮定に矛盾．よって，

$$\forall \eta, \eta'.\ (\overrightarrow{e_1}^{\mathrm{c}}, \overrightarrow{e_2}^{\mathrm{c}})\theta\eta \neq (\overrightarrow{e_1}^{\mathrm{c}}, \overrightarrow{e_2}^{\mathrm{c}})\theta'\eta'$$

により，主張は真.

次に，$\mathsf{ran}_\Gamma(e_1) \nparallel \mathsf{ran}_\Gamma(e_2)$ である場合を考える．このとき，

$$(e_1 \centerdot e_2)\theta = (e_1 \centerdot e_2)\theta' \wedge e_1\theta \neq e_1\theta' \wedge e_2\theta \neq e_2\theta'$$

となりうる．このとき，必ず $\textsc{length}(e_1\theta) \neq \textsc{length}(e_1\theta')$ である．よって，

$$\forall \eta, \eta'.\ (\textsc{length}(e_1), \overrightarrow{e_1}^{\mathrm{c}}, \overrightarrow{e_2}^{\mathrm{c}})\theta\eta \neq (\textsc{length}(e_1), \overrightarrow{e_1}^{\mathrm{c}}, \overrightarrow{e_2}^{\mathrm{c}})\theta'\eta'$$

により，主張は真. □

アルゴリズム $\textsc{Alg-C}^+$ は正しい．つまり，以下が成り立つ.

定理 8.8 (ALG-C$^+$ の正しさ)**.** プログラム $P=(G,Q,R)$ に対し，ALG-C$^+$ によりプログラム $P^{\mathrm{c}}=(G^{\mathrm{c}},Q^{\mathrm{c}},R^{\mathrm{c}})$ が得られたとする．このとき，全ての関数記号 $f\in Q$ に対し $[\![f^{\mathrm{c}}]\!]$ は $[\![f]\!]$ の補関数である．

証明. 式 e を評価するのに使用した FUN の数を $e\Downarrow v$ の証明木に沿って縦に数えたもの最大値，つまり深さを，$\#\textsc{FunDepth}(e)$ と書く．

以下を示すことが必要十分である．

> 任意の $f\in Q$ および任意の $\overrightarrow{v},\overrightarrow{v}'\in\mathsf{dom}(f)$ について $\overrightarrow{v}\neq\overrightarrow{v}'$ となるならば，
> $$[\![f]\!](\overrightarrow{v})=[\![f]\!](\overrightarrow{v'})\Rightarrow[\![f^{\mathrm{c}}]\!](\overrightarrow{v})\neq[\![f^{\mathrm{c}}]\!](\overrightarrow{v}')$$
> となる．

これを，$\#\textsc{FunDepth}(f(\overrightarrow{v}))+\#\textsc{FunDepth}(f(\overrightarrow{v}'))$ の上の帰納法で示す．

元の関数 f の定義域と対応する補関数 f^{c} の定義域とは等しいことに注意する．

基底：$\#\textsc{FunDepth}(f(\overrightarrow{v}))+\#\textsc{FunDepth}(f(\overrightarrow{v}'))=0$**.**

このとき，規則 $r,r'\in R$

$$r=f(\overrightarrow{p})\hat{=}(\overrightarrow{q})$$
$$r'=f(\overrightarrow{p'})\hat{=}(\overrightarrow{q}')$$

と代入 θ,θ' で

$$\overrightarrow{p}\theta=\overrightarrow{v},\quad \overrightarrow{p}'\theta'=\overrightarrow{v}',\quad [\![f]\!](\overrightarrow{v})=[\![f]\!](\overrightarrow{v'})$$

を満たすもの存在する．もし，$r\neq r'$ であったとすると，r^{c} と r'^{c} の右辺式の構成子が必ず異なるため，主張は正しい．なお，この場合 r と r' の右辺式の値域に重なりがあるため，右辺式 r^{c} と r'^{c} の右辺式の構成子は取り除かれることがないことに注意する．よって，$r=r'$ である場合を考える．ここで，$\overrightarrow{q}\theta=\overrightarrow{q}\theta'$ である．もし $\theta(x)\neq\theta'(x)$ となる x が存在した仮定する．このとき，$x\in\mathsf{vars}(\overrightarrow{q})$ ならば，$\overrightarrow{q}\theta\neq\overrightarrow{q}\theta'$ より補題 8.2 から補関数値が必ず異なり主張は真．このとき，$x\in\mathsf{lostvars}(r)$ ならば，補関数の右辺は $\mathsf{lostvars}(r)$ を含むため補関数値が異なり主張は真．よって，$\forall x\in\mathsf{vars}(\overrightarrow{q}).\,\theta(x)=\theta'(x)$ となる場合を考える．このとき，$\overrightarrow{v}=\overrightarrow{p}\theta$ と $\overrightarrow{v}'=\overrightarrow{p}\theta'$ は異なるにもかかわらず $[\![f]\!](\overrightarrow{v})=[\![f]\!](\overrightarrow{v}')$ であるため，変数 $z\in\mathsf{lostvars}(r)$ で $\theta(z)\neq\theta'(z)$ となるものが存在する．補関数の右辺式は，$\mathsf{lostvars}(r)$ を含むため，$[\![f^{\mathrm{c}}]\!](\overrightarrow{v})\neq[\![f^{\mathrm{c}}]\!](\overrightarrow{v}')$ が成り立つ．

帰納：$\#\textsc{FunDepth}(f(\overrightarrow{v}))>0,\#\textsc{FunDepth}(f(\overrightarrow{v}'))=0$ **もしくは** $\#\textsc{FunDepth}(f(\overrightarrow{v}))=0,\#\textsc{FunDepth}(f(\overrightarrow{v}'))>0$**.**

このとき，必ず評価 $f(\overrightarrow{v})\Downarrow\overrightarrow{u}$ と $f(\overrightarrow{v}')\Downarrow\overrightarrow{u}$ の証明木は根で必ず異なる規則を使用している．あとは，上で $r\neq r'$ の場合と同様である．

帰納： $\#\text{FunDepth}(f(\overrightarrow{v})) > 0, \#\text{FunDepth}(f(\overrightarrow{v}')) > 0$．

　このとき

$$r = f(\overrightarrow{p}) \mathrel{\hat{=}} \textbf{let}\,\{(\overrightarrow{y_i}) \mathrel{\hat{=}} f_i(\overrightarrow{x_i})\}_{i\in\{1,\ldots,n\}} \quad \textbf{in}\ (\overrightarrow{q})$$

$$r' = f(\overrightarrow{p}') \mathrel{\hat{=}} \textbf{let}\,\{(\overrightarrow{y_i}') \mathrel{\hat{=}} f'_i(\overrightarrow{x_i}')\}_{i\in\{1,\ldots,n'\}} \quad \textbf{in}\ (\overrightarrow{q}')$$

と代入 θ, θ' で

$$\overrightarrow{p}\theta = \overrightarrow{v}, \quad \overrightarrow{p}'\theta' = \overrightarrow{v}', \quad [\![f]\!](\overrightarrow{v}) = [\![f]\!](\overrightarrow{v'})$$

を満たすもの存在する．もし $r \neq r'$ である場合，これまでの議論と同様に主張は正しい．よって，$r = r'$ である場合を考える．このとき $\overrightarrow{p}\theta \neq \overrightarrow{p}\theta'$ であることから，次の三つの場合を考えればよい．

- $\theta(z) \neq \theta'(z)$ となる z が存在し，$z \in \mathsf{lostvars}(r)$
- $\theta(z) \neq \theta'(z)$ となる z が存在し，$z \in \{\overrightarrow{x_i}\}$
- $\theta(z) \neq \theta'(z)$ となる z が存在し，$z \in (\mathsf{vars}(\overrightarrow{q}) \cap \mathsf{vars}(\overrightarrow{p}))$

一つ目の場合，補関数の規則 r^{c} の定義より $[\![f^{\mathrm{c}}]\!](\overrightarrow{v}) \neq [\![f^{\mathrm{c}}]\!](\overrightarrow{v})$ となる．よって二つ目の場合を考える．このとき，$[\![f_i]\!](\overrightarrow{x_i}\theta) \neq [\![f_i]\!](\overrightarrow{x_i}\theta')$ であったとすると，補題 8.2 より主張は真．よって，$[\![f_i]\!](\overrightarrow{x_i}\theta) = [\![f_i]\!](\overrightarrow{x_i}\theta')$ が成り立つ場合を考える．ここで $\overrightarrow{x_i}\theta \neq \overrightarrow{x_i}\theta'$ となるため，帰納法の仮定より $[\![{f_i}^{\mathrm{c}}]\!](\overrightarrow{x_i}\theta) \neq [\![{f_i}^{\mathrm{c}}]\!](\overrightarrow{x_i}\theta')$ が言える．よって，$[\![f^{\mathrm{c}}]\!](\overrightarrow{v}) \neq [\![f^{\mathrm{c}}]\!](\overrightarrow{v'})$ が成り立つ．三つ目の場合は，補題 8.2 より主張は真．

□

　いくつかの例により，Alg-C^{+} の効果を確認する．

例 8.3 (三人目以降の著者[1])．以下の関数 *simplyfyAuthors* は，文献リストから `<author>` を持つ書籍のみを抜き出しその上で，本の著者が三人以上の場合に三人目以降を省略し，

[1]変換 *simplyfyAuthors* は XML Query Use Cases（`http://www.w3.org/TR/xquery-use-cases`）の “XMP”-Q6 を簡略化したものである．元の変換のソースは XML 属性を含んでいるが，関数 *simplyfyAuthors* は XML 属性を考慮していない．

<et-al> に置き換える．

$$
\begin{array}{l}
\textbf{data}\ \mathit{Author}\ \ \hat{=}\ \texttt{<author>}(\texttt{<last>}(\mathit{String}) \centerdot \texttt{<first>}(\mathit{String}))\\
\textbf{data}\ \mathit{Authors} \hat{=}\ \mathit{Author} \centerdot \mathit{Author}^*\\
\textbf{data}\ \mathit{Editor}\ \ \hat{=}\ \texttt{<editor>}(\texttt{<last>}(\mathit{String}) \centerdot \texttt{<first>}(\mathit{String}) \centerdot \texttt{<affiliation>}(\mathit{String}))\\
\textbf{data}\ \mathit{Editors}\ \hat{=}\ \mathit{Editor} \centerdot \mathit{Editor}^*\\
\textbf{data}\ \mathit{Rest}\ \ \ \hat{=}\ (\texttt{<publisher>}(\mathit{String}) \centerdot \texttt{<price>}(\mathit{String}))\\
\mathit{simplfyAuthors}(\texttt{<bib>}(r)) \hat{=} \textbf{let}\ x \hat{=} f(r)\ \textbf{in}\ \texttt{<bib>}(x)\\
f(\varepsilon) \hat{=} \varepsilon\\
f(\texttt{<book>}(\texttt{<title>}(t) \centerdot \mathit{a1} :: \mathit{Author} \centerdot \mathit{a2} :: \mathit{Author} \centerdot \mathit{as} :: \mathit{Authors} \centerdot p :: \mathit{Rest}) \centerdot r)\\
\quad \hat{=} \textbf{let}\ x \hat{=} f(r)\ \textbf{in}\ \texttt{<book>}(\texttt{<title>}(t) \centerdot \mathit{a1} \centerdot \mathit{a2} \centerdot \texttt{<et-al>}) \centerdot x\\
f(\texttt{<book>}(\texttt{<title>}(t) \centerdot \mathit{as} :: (\mathit{Author} \mid \mathit{Author} \centerdot \mathit{Author}) \centerdot p :: \mathit{Rest}) \centerdot r)\\
\quad \hat{=} \textbf{let}\ x \hat{=} f(r)\ \textbf{in}\ \texttt{<book>}(\texttt{<title>}(t) \centerdot \mathit{as}) \centerdot x\\
f(\texttt{<book>}(\texttt{<title>}(t) \centerdot e :: \mathit{Editors} \centerdot p :: \mathit{Rest}) \centerdot r) \hat{=} \textbf{let}\ x \hat{=} f(r)\ \textbf{in}\ x
\end{array}
$$

アルゴリズム $\text{A}\textsc{lg}\text{-C}^+$ は，上の *simplifyAuthors* に対し，以下の補関数 $\mathit{simplifyAuthors}^{\mathrm{c}}$ を返す．

$$
\begin{array}{l}
\mathit{simplfyAuthors}^{\mathrm{c}}(\texttt{<bib>}(r)) \hat{=} \textbf{let}\ x^{\mathrm{c}} \hat{=} f^{\mathrm{c}}(r)\ \textbf{in}\ x^{\mathrm{c}}\\
f^{\mathrm{c}}(\varepsilon) \hat{=} \mathsf{B21}\\
f^{\mathrm{c}}(\texttt{<book>}(\texttt{<title>}(t) \centerdot \mathit{a1} :: \mathit{Author} \centerdot \mathit{a2} :: \mathit{Author} \centerdot \mathit{as} :: \mathit{Authors} \centerdot p :: \mathit{Rest}) \centerdot r)\\
\quad \hat{=} \textbf{let}\ x^{\mathrm{c}} \hat{=} f^{\mathrm{c}}(r)\ \textbf{in}\ \mathsf{B}_{31}(x^{\mathrm{c}}, \mathit{as}, p)\\
f(\texttt{<book>}(\texttt{<title>}(t) \centerdot \mathit{as} :: (\mathit{Author} \mid \mathit{Author} \centerdot \mathit{Author}) \centerdot p :: \mathit{Rest}) \centerdot r)\\
\quad \hat{=} \textbf{let}\ x^{\mathrm{c}} \hat{=} f^{\mathrm{c}}(r)\ \textbf{in}\ \mathsf{B}_{41}(x^{\mathrm{c}}, p)\\
f(\texttt{<book>}(\texttt{<title>}(t) \centerdot e :: \mathit{Editors} \centerdot p :: \mathit{Rest}) \centerdot r)\\
\quad \hat{=} \textbf{let}\ x^{\mathrm{c}} \hat{=} f^{\mathrm{c}}(r)\ \textbf{in}\ \mathsf{B}_{51}(x^{\mathrm{c}}, t, e, p)
\end{array}
$$

上の *simplifyAuthors* プログラムは非単射と判定される連接を含まないことに注意する．

例 8.4 (題目の抽出)．以下の関数 *titles* は，論文様の構造の生垣から各章の題目と各節の題目を抜き出す．

$$
\begin{array}{l}
\textbf{data}\ S\ \hat{=}\ (\texttt{<section>}(\texttt{<title>}(\mathit{String}) \centerdot P))^*\\
\textbf{data}\ P\ \hat{=}\ (\texttt{<p>}(\mathit{String}))^*\\
\mathit{titles}(\varepsilon) \hat{=} \varepsilon\\
\mathit{titles}(\texttt{<chapter>}(\texttt{<title>}(t) \centerdot p :: P \centerdot s :: S) \centerdot r)\\
\quad \hat{=} \textbf{let}\ x \hat{=} \mathit{stitles}(s)\\
\qquad\qquad y \hat{=} \mathit{titles}(r)\\
\qquad \textbf{in}\ \texttt{<title>}(t) \centerdot (x \centerdot_{(1)} y)\\
\mathit{stitles}(\varepsilon) \hat{=} \varepsilon\\
\mathit{stitles}(\texttt{<section>}(\texttt{<title>}(t) \centerdot p :: P) \centerdot r)\\
\quad \hat{=} \textbf{let}\ x \hat{=} \mathit{stitles}(r)\ \textbf{in}\ \texttt{<title>}(t) \centerdot x
\end{array}
$$

この変換は，*titles* の二つ目の規則の右辺に非単射と判定される連接 $x \centerdot_{(1)} y$ を含む．しかし，その他の連接は単射であると判定される．

アルゴリズム $\textsc{Alg-C}^+$ は，上の *titles* に対し，以下の補関数 $stitles^{\mathrm{c}}$ を返す．

$$\begin{aligned}
&titles^{\mathrm{c}}(\varepsilon) \mathrel{\hat{=}} \mathsf{B}_{11}\\
&titles^{\mathrm{c}}(\texttt{<chapter>}(\texttt{<title>}(t) \centerdot p :: P \centerdot s :: S) \centerdot r)\\
&\quad \mathrel{\hat{=}} \mathbf{let}\, x \;\mathrel{\hat{=}} stitles(s)\\
&\qquad\quad x^{\mathrm{c}} \mathrel{\hat{=}} stitles^{\mathrm{c}}(s)\\
&\qquad\quad y^{\mathrm{c}} \mathrel{\hat{=}} titles^{\mathrm{c}}(r)\\
&\qquad \mathbf{in}\ \mathsf{B}_{21}(\textsc{length}_1(x)), x^{\mathrm{c}}, y^{\mathrm{c}}, p)\\
&stitles^{\mathrm{c}}(\varepsilon) \mathrel{\hat{=}} \mathsf{B}_{31}\\
&stitles^{\mathrm{c}}(\texttt{<section>}(\texttt{<title>}(t) \centerdot p :: P) \centerdot r)\\
&\quad \mathrel{\hat{=}} \mathbf{let}\, x^{\mathrm{c}} \mathrel{\hat{=}} stitles^{\mathrm{c}}(r)\, \mathbf{in}\ \mathsf{B}_{31}(x^{\mathrm{c}}, p)
\end{aligned}$$

連接 $x \centerdot_{(1)} y$ は非単射と判定されたので，補関数において，第一引数の長さが保存されている．また，非単射と判定された連接の第一引数 x を計算のに必要な順方向変換の関数 *stitles* は，補関数においても呼ばれている．

例 8.5 (文献リストの選り分け)．第 6 章の例 6.7 の変換 *bkref* を考える．アルゴリズム $\textsc{Alg-C}^+$ は，*bkref* に対し，以下の補関数 $bkref^{\mathrm{c}}$ を返す．

$$\begin{aligned}
&bkref^{\mathrm{c}}(\texttt{<bib>}(r)) \mathrel{\hat{=}} \mathbf{let}\,(x^{\mathrm{c}}, y^{\mathrm{c}}) \mathrel{\hat{=}} f^{\mathrm{c}}(r)\, \mathbf{in}\ \mathsf{B}_{11}(x^{\mathrm{c}}, y^{\mathrm{c}})\\
&f^{\mathrm{c}}(\varepsilon) \mathrel{\hat{=}} (\mathsf{B}_{21}, \mathsf{B}_{22})\\
&f^{\mathrm{c}}(\texttt{<book>}(\texttt{<title>}(t) \centerdot as :: Authors \centerdot p :: Rest) \centerdot r)\\
&\quad \mathrel{\hat{=}} \mathbf{let}\,(x^{\mathrm{c}}, y^{\mathrm{c}}) \mathrel{\hat{=}} f^{\mathrm{c}}(r)\, \mathbf{in}\ (\mathsf{B}_{31}(x^{\mathrm{c}}, p), \mathsf{B}_{32}(y^{\mathrm{c}}, p))\\
&f^{\mathrm{c}}(\texttt{<book>}(\texttt{<title>}(t) \centerdot e :: Editors \centerdot p :: Rest) \centerdot r)\\
&\quad \mathrel{\hat{=}} \mathbf{let}\,(x^{\mathrm{c}}, y^{\mathrm{c}}) \mathrel{\hat{=}} f^{\mathrm{c}}(r)\\
&\qquad\qquad a^{\mathrm{c}} \qquad \mathrel{\hat{=}} affil^{\mathrm{c}}(e)\\
&\qquad \mathbf{in}\ (\mathsf{B}_{41}(x^{\mathrm{c}}, p), \mathsf{B}_{42}(y^{\mathrm{c}}, a^{\mathrm{c}}, p))\\
&affil^{\mathrm{c}}(\varepsilon) \mathrel{\hat{=}} \mathsf{B}_{51}\\
&affil^{\mathrm{c}}(\texttt{<editor>}(\texttt{<last>}(n) \centerdot \texttt{<first>}(m) \centerdot \texttt{<affiliation>}(a)) \centerdot r)\\
&\quad \mathrel{\hat{=}} \mathbf{let}\, x^{\mathrm{c}} \mathrel{\hat{=}} affil^{\mathrm{c}}(r)\, \mathbf{in}\ \mathsf{B}_{61}(x^{\mathrm{c}}, n, m)
\end{aligned}$$

直感的には，補関数の返り値の第 j 要素は，元の関数の返り値の第 j 要素の計算において失われた情報を表している．

8.2　逆方向変換導出

これまでの議論から，順方向変換 f について，その補関数 f^{c} が得られた．よって，それらを組にした関数の逆関数である $\langle f, f^{\mathrm{c}}\rangle^{-1}$ を求めることができれば，第 3 章の式 (RFL) により，

$$f_{\mathrm{B}}(s, v) \mathrel{\hat{=}} \langle f, f^{\mathrm{c}}\rangle^{-1}(v, f^{\mathrm{c}}(s))$$

とすることで，逆方向変換プログラムが得られる．ところが，言語 $\textsc{Vdl}^+$ で記述された順方向変換とそれに対し $\textsc{Alg-C}^+$ の導出する補関数については，組化したあとただ左右を入

れかえる手法ではうまく $\langle f, f^c \rangle^{-1}$ を導出できない．これは，補関数は LENGTH 関数を含んでいることにより LENGTH の分だけ元の関数の補関数で再帰構造が異なるため，組化自身も単純ではなくなり，また仮に組化が成功したとしても関数呼出のネストにより効果的な逆計算も難しくなるためである．また，非単射と判定される生垣の連接を含むプログラムの左右を入れ換えることで得られるパターンはパターンマッチが一意ではなくなり，同じパターンと同じ入力に対し，複数の束縛を生じうる．

組化が単純でなくなる原因は LENGTH にある．よって，我々は LENGTH を特別扱いしつつ組化された関数の逆関数を導出する．関数 LENGTH に対し，以下の性質を満たす SPLITAT が定義できる．

$$\textsc{splitAt}(\textsc{length}(x), x \centerdot y) = (x, y)$$

我々はこの SPLITAT を逆関数の導出に利用する．また，変数パターンを利用することにより，我々は決定的な逆関数を導出する．たとえば，以下の非単射と判断される連接を含む関数

$$\begin{aligned}
&\textbf{data}\ A \mathrel{\hat{=}} \texttt{<a>}^* \\
&f(x :: A, y :: A, z :: A) \mathrel{\hat{=}} \textbf{let}\ s \mathrel{\hat{=}} g(x) \\
&\qquad\qquad\qquad\qquad\qquad\quad\ \ t \mathrel{\hat{=}} g(y) \\
&\qquad\qquad\qquad\qquad\qquad \textbf{in}\ s \centerdot_{(1)} (t \centerdot_{(2)} z) \\
&g(x :: A) \mathrel{\hat{=}} x
\end{aligned}$$

と，ALG-C$^+$ が導出したその補関数

$$\begin{aligned}
f^c(x, y, z) \mathrel{\hat{=}}\ &\textbf{let}\ s \mathrel{\hat{=}} g(x) \\
&\qquad t \mathrel{\hat{=}} g(y) \\
&\textbf{in}\ \mathsf{B}_{11}(\textsc{length}_1(s), \textsc{length}_2(t))
\end{aligned}$$

から以下の $\langle f, f^c \rangle^{-1}$ のプログラムが得られる．

$$\begin{aligned}
\langle f, f^c \rangle^{-1}(v_1, \mathsf{B}_{11}(l_1, l_2)) \mathrel{\hat{=}}\ &\textbf{let}\ (s, v_2) \mathrel{\hat{=}} \textsc{splitAt}(l_1, v_1) \\
&\qquad (t, z :: A) \mathrel{\hat{=}} \textsc{splitAt}(l_2, v_2) \\
&\qquad x :: A \mathrel{\hat{=}} g^{-1}(s) \\
&\qquad y :: A \mathrel{\hat{=}} g^{-1}(t) \\
&\textbf{in}\ (x, y, z)
\end{aligned}$$

ここで，v_k は識別子 k を持つ非単射な連接の結果を束縛するための変数であり，l_k はその連接の第一引数の長さを束縛するための変数である．関数 SPLITAT(l, v) は，Haskell における `splitAt` 関数とほぼ同じ関数である．Haskell の `splitAt` と異なり，SPLITAT(l, v) は v の長さが l より小さい場合に未定義である．

上の $\langle f, f^c \rangle^{-1}$ のような組にした関数の逆関数を表現するのに，言語 VDL$^+$ を少し拡張する．まず，我々は，**let** 束縛において束縛の左辺に現れた変数が束縛の右辺に現れることを許す．また，**let** 束縛の左辺において，変数のみではなくパターンの出現を許す．

8.2.1　let 束縛の生成とパターンの生成

上のような決定的な $\langle f, f^c\rangle^{-1}$ の定義を求めるのに，我々は，適切に **let** 束縛を生成し，また，決定的になるよう関数のパターンを導出しなければならない．我々は，これをパターン導出関係 $\overset{\mathrm{p}}{\leadsto}$ と **let** 束縛導入関係 $\overset{\mathrm{let}}{\leadsto}$ により実現する．パターン導出関係 $\overset{\mathrm{p}}{\leadsto}$ は，式をパターンに変換した場合に，式に非単射と判断される連接が含まれているとパターンとして有効なものが得られないため，式中の非単射と判断される連接の部分を変数で置き換えパターンを作成する．このとき，非単射と判断される連接に対応する変数を分解するための $\textsc{splitAt}$ を導入するのが **let** 束縛導入関係 $\overset{\mathrm{let}}{\leadsto}$ である．ただし，非単射と判断される連接は，連接されるそれぞれの式にまた非単射と判断される連接を含んでいる可能性があるので，関係 $\overset{\mathrm{let}}{\leadsto}$ は，関係 $\overset{\mathrm{p}}{\leadsto}$ を使用する．形式的には，パターン導出関係 $\overset{\mathrm{p}}{\leadsto}$ と **let** 束縛導入関係 $\overset{\mathrm{let}}{\leadsto}$ は以下の規則により定義される．ここで，関係 $\Gamma, e \overset{\mathrm{p}}{\leadsto} p$ は，型環境 Γ と e から，e の結果をパターンマッチするためのパターン p が生成される，と読む．そして，関係 $\Gamma, V, e \overset{\mathrm{let}}{\leadsto} L$ は，型環境 Γ と長さを束縛する変数の集合 V から，非単射と判定された連接を分解するための **let** 束縛 L を生成する，と読む．

$$\frac{}{\Gamma, \varepsilon \overset{\mathrm{p}}{\leadsto} \varepsilon}\textsc{Eps} \quad \frac{}{\Gamma, x \overset{\mathrm{p}}{\leadsto} x :: \Gamma(x)}\textsc{Var} \quad \frac{\Gamma, e \overset{\mathrm{p}}{\leadsto} p}{\Gamma, \sigma(e) \overset{\mathrm{p}}{\leadsto} \sigma(p)}\textsc{Con}$$

$$\frac{\mathsf{ran}_\Gamma(e_1) \nparallel \mathsf{ran}_\Gamma(e_2) \quad \Gamma, e_1 \overset{\mathrm{p}}{\leadsto} p_1 \quad \Gamma, e_2 \overset{\mathrm{p}}{\leadsto} p_2}{\Gamma, e_1 \centerdot e_2 \overset{\mathrm{p}}{\leadsto} p_1 \centerdot p_2}\textsc{Cat}$$

$$\frac{\mathsf{ran}_\Gamma(e_1) \parallel \mathsf{ran}_\Gamma(e_2)}{\Gamma, e_1 \centerdot_{(k)} e_2 \overset{\mathrm{p}}{\leadsto} v_k :: T_k}\textsc{ICat}$$

$$\frac{}{\Gamma, V, \varepsilon \overset{\mathrm{let}}{\leadsto} \emptyset}\textsc{Eps} \quad \frac{}{\Gamma, V, x \overset{\mathrm{let}}{\leadsto} \emptyset}\textsc{Var} \quad \frac{\Gamma, V, e \overset{\mathrm{let}}{\leadsto} L}{\Gamma, V, \sigma(e) \overset{\mathrm{let}}{\leadsto} L}\textsc{Con}$$

$$\frac{\mathsf{ran}_\Gamma(e_1) \nparallel \mathsf{ran}_\Gamma(e_2) \quad \Gamma, V, e_1 \overset{\mathrm{let}}{\leadsto} L_1 \quad \Gamma, V, e_2 \overset{\mathrm{let}}{\leadsto} L_2}{\Gamma, V, e_1 \centerdot e_2 \overset{\mathrm{let}}{\leadsto} L_1 \cup L_2}\textsc{Cat}$$

$$\frac{\begin{array}{c}\mathsf{ran}_\Gamma(e_1) \parallel \mathsf{ran}_\Gamma(e_2) \quad l_k \in V \\ \Gamma, V, e_1 \overset{\mathrm{let}}{\leadsto} L_1 \quad \Gamma, V, e_2 \overset{\mathrm{let}}{\leadsto} L_2 \quad \Gamma, e_1 \overset{\mathrm{p}}{\leadsto} p_1 \quad \Gamma, e_2 \overset{\mathrm{p}}{\leadsto} p_2\end{array}}{\Gamma, V, e_1 \centerdot_{(k)} e_2 \overset{\mathrm{let}}{\leadsto} \{(p_1, p_2) \triangleq \textsc{splitAt}(l_k, v_k)\} \cup L_1 \cup L_2}\textsc{ICat}$$

組にした関数の逆関数においては，補関数の返り値もパターンにより分解しなければならない．そのためのパターン導出関係が $\overset{\mathrm{cp}}{\leadsto}$ である．

$$\frac{}{x \overset{\mathrm{cp}}{\leadsto} x}\textsc{C-Var} \quad \frac{}{\textsc{length}_k(e) \overset{\mathrm{cp}}{\leadsto} l_k}\textsc{C-Len} \quad \frac{e_1 \overset{\mathrm{cp}}{\leadsto} p_1 \quad \ldots \quad e_n \overset{\mathrm{cp}}{\leadsto} p_n}{\mathsf{C}(e_1, \ldots, e_n) \overset{\mathrm{cp}}{\leadsto} \mathsf{C}(p_1, \ldots, e_n)}\textsc{C-Con}$$

また，パターンから式を作成する関係 $\overset{\mathrm{e}}{\rightsquigarrow}$ も以下の通りに定める．関係 $\overset{\mathrm{e}}{\rightsquigarrow}$ は，パターン中のそれぞれの変数パターン $x :: T$ を変数式 x に置き換える．

$$\frac{}{\varepsilon \overset{\mathrm{e}}{\rightsquigarrow} \varepsilon}\textsc{Eps} \quad \frac{}{x :: T \overset{\mathrm{e}}{\rightsquigarrow} x}\textsc{Var} \quad \frac{p \overset{\mathrm{e}}{\rightsquigarrow} e}{\sigma(p) \overset{\mathrm{e}}{\rightsquigarrow} \sigma(e)}\textsc{Con} \quad \frac{p_1 \overset{\mathrm{e}}{\rightsquigarrow} e_1 \quad p_2 \overset{\mathrm{e}}{\rightsquigarrow} e_2}{p_1 \centerdot p_2 \overset{\mathrm{e}}{\rightsquigarrow} e_1 \centerdot e_2}\textsc{Cat}$$

今後の利便性のため，我々は以下の通りにパターン導出を列に拡張する．

$$\begin{gathered}
\Gamma, (q_1, \ldots, q_n) \overset{\mathrm{p}}{\rightsquigarrow} (p_1, \ldots, p_n) \Leftrightarrow \forall i \in \{1, \ldots, n\}.\, \Gamma, q_i \overset{\mathrm{p}}{\rightsquigarrow} p_i \\
\Gamma, V, (q_1, \ldots, q_n) \overset{\mathrm{let}}{\rightsquigarrow} L_1 \cup \cdots \cup L_n \Leftrightarrow \forall i \in \{1, \ldots, n\}.\, \Gamma, V, q_i \overset{\mathrm{p}}{\rightsquigarrow} L_i \\
(q_1, \ldots, q_n) \overset{\mathrm{cp}}{\rightsquigarrow} (p_1, \ldots, p_n) \Leftrightarrow \forall i \in \{1, \ldots, n\}.\, q_i \overset{\mathrm{p}}{\rightsquigarrow} p_i \\
(p_1, \ldots, p_n) \overset{\mathrm{e}}{\rightsquigarrow} (q_1, \ldots, q_n) \Leftrightarrow \forall i \in \{1, \ldots, n\}.\, p_i \overset{\mathrm{e}}{\rightsquigarrow} q_i
\end{gathered}$$

パターン導出について以下の補題が成り立つ．

補題 8.3. $\Gamma, p \overset{\mathrm{p}}{\rightsquigarrow} p' \Rightarrow \mathsf{ran}_\Gamma(p) = [\![p']\!]$

証明の概略. 値域の近似の際に，我々は関数呼出式の値域を近似し，その他の式の値域を関数呼出式の近似された値域より求めたことによる． □

8.2.2 組にした関数の逆関数：$\langle f, f^{\mathrm{c}} \rangle^{-1}$ の生成

順方向変換 f と f に対し Alg-C$^+$ が導出する f^{c} に対し，以下のアルゴリズムにより，組にした関数の逆関数 $\langle f, f^{\mathrm{c}} \rangle^{-1}$ を生成する．

アルゴリズム 8.5 (組にした関数の逆関数の生成).
入力： 言語 Vdl$^+$ のプログラム $\mathcal{P}$.
出力： プログラム $\mathcal{P}''$.
手続き：

1. プログラム $\mathcal{P}$ に Alg-C$^+$ を適用することで，$\mathcal{P}^{\mathrm{c}}$ を得る．
2. プログラム $\mathcal{P}$ のそれぞれの規則

$$\begin{aligned} r = f(\overrightarrow{p}) \doteq \mathbf{let}\, &\{(\overrightarrow{y_i}) \doteq f_i(\overrightarrow{x_i})\}_{i \in \{1,\ldots,n\}} \\ \mathbf{in}\, &(\overrightarrow{q}) \end{aligned}$$

に対し以下を行う．

(a) f_i の返り値の近似された型を記述するための型環境 Γ を

$$\Gamma = \Gamma_{\overrightarrow{p}} \cup \{y_{ij} \mapsto \pi_j T_{f_i} \mid i \in \{1, \ldots, n\}, j \in \{1, \ldots, |\overrightarrow{y_i}|\}\}$$

により定める．ただし，$\overrightarrow{y_i} = (y_{i1}, \ldots, y_{i|\overrightarrow{y_i}|})$ であるとする．

(b) もし，$f \in INJ_{\mathcal{P}}$ である場合，

$$\overrightarrow{p} \overset{\mathrm{e}}{\rightsquigarrow} \overrightarrow{p}'' \qquad \Gamma, \overrightarrow{q} \overset{\mathrm{p}}{\rightsquigarrow} \overrightarrow{q}''$$

により，逆関数のパターンと式を準備し，規則 r'' を

$$\begin{aligned} r'' = f^{-1}(\overrightarrow{q}'') \mathrel{\hat{=}} & \ \mathbf{let}\ \{(\overrightarrow{x_i} :: \overrightarrow{T_i}) \mathrel{\hat{=}} {f_i}^{-1}(\overrightarrow{y_i})\}_{i\in\{1,\ldots,n\}} \\ & \ \mathbf{in}\ \overrightarrow{p}'' \end{aligned}$$

と構成する．ただし，$\overrightarrow{x_i} = (x_{i1}, \ldots, x_{i|\overrightarrow{x_i}|})$ とした場合に，

$$\overrightarrow{T_i} = (T_{i1}, \ldots, T_{i|\overrightarrow{x_i}|}) = (\Gamma(x_{i1}), \ldots, \Gamma(x_{i|\overrightarrow{x_i}|}))$$

とする．

(c) もし，$f \notin INJ_{\mathcal{P}}$ である場合，対応している補関数の規則

$$\begin{aligned} r^{\mathrm{c}} = f^{\mathrm{c}}(\overrightarrow{p}) \mathrel{\hat{=}} & \ \mathbf{let}\ \{(\overrightarrow{y_i}^{\mathrm{c}}) \mathrel{\hat{=}} {f_i}^{\mathrm{c}}(\overrightarrow{x_i})\}_{i\in\{1,\ldots,n\}, i\neq I} \\ & \qquad \{(\overrightarrow{y_i}) \mathrel{\hat{=}} f_i(\overrightarrow{x_i})\}_{i\in\{1,\ldots,n\}} \\ & \ \mathbf{in}\ (\overrightarrow{q}') \end{aligned}$$

を考える．ただし，$I = \{i \mid f_i \in INJ_{\mathcal{P}}\}$ である．ここで，

$$\overrightarrow{p} \overset{\mathrm{e}}{\rightsquigarrow} \overrightarrow{p}'' \qquad \Gamma, \overrightarrow{q} \overset{\mathrm{p}}{\rightsquigarrow} \overrightarrow{q}''_1 \qquad \overrightarrow{q}' \overset{\mathrm{cp}}{\rightsquigarrow} \overrightarrow{q}''_2 \qquad \Gamma, \mathsf{vars}(\overrightarrow{q}''_2), \overrightarrow{q} \overset{\mathrm{let}}{\rightsquigarrow} L$$

により，逆関数のパターンと式を準備し，規則 r'' を以下で構成する．

$$\begin{aligned} r'' = \langle f, f^{\mathrm{c}}\rangle^{-1}(\overrightarrow{q}''_1, \overrightarrow{q}''_2) \mathrel{\hat{=}} & \ \mathbf{let}\ \{(\overrightarrow{x_i} :: \overrightarrow{T_i}) \mathrel{\hat{=}} {f_i}^{-1}(\overrightarrow{y_i})\}_{i\in\{1,\ldots,n\}, i\in I} \\ & \qquad \{(\overrightarrow{x_i} :: \overrightarrow{T_i}) \mathrel{\hat{=}} \langle f_i, {f_i}^{\mathrm{c}}\rangle^{-1}(\overrightarrow{y_i}, \overrightarrow{y_i}^{\mathrm{c}})\}_{i\in\{1,\ldots,n\}, i\notin I} \\ & \qquad L \\ & \ \mathbf{in}\ (\overrightarrow{p}'') \end{aligned}$$

ただし，$\overrightarrow{x_i} = (x_{i1}, \ldots, x_{i|\overrightarrow{x_i}|})$ とした場合に，

$$\overrightarrow{T_i} = (T_{i1}, \ldots, T_{i|\overrightarrow{x_i}|}) = (\Gamma(x_{i1}), \ldots, \Gamma(x_{i|\overrightarrow{x_i}|}))$$

とする．ここで，$\overrightarrow{y_i}^{\mathrm{c}} = ({y_{i1}}^{\mathrm{c}}, \ldots, {y_{i|\overrightarrow{y_i}|}}^{\mathrm{c}})$ である．

3. 規則 r'' を集めて，プログラム $\mathcal{P}''$ を構成する．　□

上記アルゴリズムは，もし関数 f が単射である場合には，f の定義の左右を入れ替えているだけであることに注意する．

ステップ 2-(b)，ステップ 2-(c) において，関数 f_i が第 7 章の手法により引数に対して特化されている場合には，**let** 式の左辺は $(\overrightarrow{x_i} :: \overrightarrow{T_i})$ ではなく，単に $\overrightarrow{x_i}$ とすることができる．これは，特化された関数の定義域は，呼び出されるときに与えられる引数の型と比べて，小さいかまたは等しいためである．

逆関数の導出の後，$f \in \mathit{INJ}_{\mathcal{P}}$ である場合は，

$$f_{\mathrm{B}}(_, v) \mathrel{\hat{=}} f^{-1}(v)$$

$f \in \mathit{NINJ}_{\mathcal{P}}$ である場合は

$$f_{\mathrm{B}}(s, v) \mathrel{\hat{=}} \langle f, f^{\mathrm{c}} \rangle^{-1}(v, f^{\mathrm{c}}(s))$$

とすることにより，逆方向変換 f_{B} を求めることができる．

定理 8.9. 上記で得られる逆方向変換 f_{B} は振る舞いがよい．

証明. 定理 3.3 より，第 3 章の式 (RFL) に基づき補関数から構成される逆方向変換は振る舞いはよい．よって，補関数導出の正しさおよび組化された関数の逆関数の導出の正しさが言えれば，主張を示すのに十分である．ところで，定理 8.8 により補関数の導出は正しいことは示された．逆関数の導出の正しさについては，後に定理 8.10 を示す． □

上記アルゴリズムにより導出される逆方向変換の例をいくつか示す．

例 8.6 (近似された型の役割). 第 6 章の $\mathit{c2x}$ を考える．単射性判定により，$\mathit{c2x}$ は単射であると判定される．よって，逆方向変換 $\mathit{c2x}_{\mathrm{B}}$ は以下のように逆関数を用いて定義される．

$$\begin{aligned}
&\mathbf{data}\ T_1 \mathrel{\hat{=}} (\texttt{<h2>}(\mathit{String}) \centerdot P)^* \\
&\mathbf{data}\ T_2 \mathrel{\hat{=}} (\texttt{<h1>}(\mathit{String}) \centerdot P \centerdot T_1)^* \\
&\mathbf{data}\ P\ \ \mathrel{\hat{=}} (\texttt{<p>}(\mathit{String}))^*
\end{aligned}$$

$$\begin{aligned}
&\mathit{c2x}_{\mathrm{B}}(s, v) \mathrel{\hat{=}} \mathit{c2x}^{-1}(v) \\
&\mathit{c2x}^{-1}(\varepsilon) \mathrel{\hat{=}} \varepsilon \\
&\mathit{c2x}^{-1}(\texttt{<h1>}(t :: \mathit{String}) \centerdot p :: P \centerdot v_1 :: T_1 \centerdot v_2 :: T_2) \\
&\quad \texttt{<chapter>}(\texttt{<title>}(t :: \mathit{String}) \centerdot p \centerdot \mathit{s2x}^{-1}(v_1)) \centerdot \mathit{c2x}^{-1}(v_1) \\
&\mathit{s2x}^{-1}(\varepsilon) \mathrel{\hat{=}} \varepsilon \\
&\mathit{s2x}^{-1}(\texttt{<h2>}(t :: \mathit{String}) \centerdot p :: P \centerdot v_3 :: T_1) \\
&\quad \mathrel{\hat{=}} \texttt{<section>}(\texttt{<title>}(t) \centerdot p) \centerdot \mathit{s2x}^{-1}(v_3)
\end{aligned}$$

ここで，T_1 と T_2 はそれぞれ g と f の近似された値域を表現する型である．近似された値域の上で単射性が保証されているため，補題 8.3 により左右の入れ換えにより得られる逆関数 $\mathit{c2x}^{-1}$ の決定性も保証される．

その他，いくつかの例（第 6 章の例 6.5 の toc や例 6.6 の $\mathit{results2scores}$ など）は，近似された値域の上での単射性判定により単射だと判定される．このとき，逆方向変換は逆関数で定義されるため，得られる逆方向変換により，順方向変換の値域の上の任意の更新をソースへ反映することができる．

例 8.7 (非単射な変換に対する逆方向変換導出). 第 6 章の例 6.7 の bkref を考える．この関数 bkref に対し ALG-C$^+$ の導出する補関数は，例 8.5 の $\mathit{bkref}^{\mathrm{c}}$ である．

これらより得られる逆方向変換は以下である．

$$
\begin{aligned}
&\mathbf{data}\ \mathit{Author} &&\mathrel{\hat{=}} \texttt{<author>}(\texttt{<last>}(\mathit{String})\,\textbf{.}\,\texttt{<first>}(\mathit{String}))\\
&\mathbf{data}\ \mathit{Authors} &&\mathrel{\hat{=}} \mathit{Author}\,\textbf{.}\,\mathit{Author}^*\\
&\mathbf{data}\ \mathit{Editor} &&\mathrel{\hat{=}} \texttt{<editor>}(\texttt{<last>}(\mathit{String})\,\textbf{.}\,\texttt{<first>}(\mathit{String})\,\textbf{.}\,\texttt{<affiliation>}(\mathit{String}))\\
&\mathbf{data}\ \mathit{Editors} &&\mathrel{\hat{=}} \mathit{Editor}\,\textbf{.}\,\mathit{Editor}^*\\
&\mathbf{data}\ \mathit{Affils} &&\mathrel{\hat{=}} \texttt{<affiliation>}(\mathit{String})\,\textbf{.}\,(\varepsilon \mid \mathit{Affils})\\
&\mathbf{data}\ \mathit{Books} &&\mathrel{\hat{=}} (\texttt{<book>}(\texttt{<title>}(\mathit{String})\,\textbf{.}\,\mathit{Authors}))^*\\
&\mathbf{data}\ \mathit{Refs} &&\mathrel{\hat{=}} (\texttt{<reference>}(\texttt{<title>}(\mathit{String}\,\textbf{.}\,\mathit{Affils})))^*
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{bkref}_{\mathrm{B}}(s,v) \mathrel{\hat{=}} \langle \mathit{bkref}, \mathit{bkref}^{\mathrm{c}}\rangle^{-1}(v, \mathit{bkref}^{\mathrm{c}}(s))\\
&\langle \mathit{bkref}, \mathit{bkref}^{\mathrm{c}}\rangle^{-1}(\texttt{<bib>}(x :: \mathit{Books}\,\textbf{.}\,y :: \mathit{Refs}), \mathsf{B}_{11}(x^{\mathrm{c}}, y^{\mathrm{c}}))\\
&\qquad \mathrel{\hat{=}} \mathbf{let}\ r \mathrel{\hat{=}} \langle f, f^{\mathrm{c}}\rangle^{-1}(x, y, x^{\mathrm{c}}, y^{\mathrm{c}})\ \mathbf{in}\ \texttt{<bib>}(r)\\
&\langle f, f^{\mathrm{c}}\rangle^{-1}(\varepsilon, \varepsilon, \mathsf{B}_{21}, \mathsf{B}_{22}) \mathrel{\hat{=}} \varepsilon\\
&\langle f, f^{\mathrm{c}}\rangle^{-1}(\texttt{<book>}(\texttt{<title>}(t)\,\textbf{.}\,as)\,\textbf{.}\,x, y, \mathsf{B}_{31}(x^{\mathrm{c}}, p), \mathsf{B}_{32}(y^{\mathrm{c}}, p))\\
&\qquad \mathrel{\hat{=}} \mathbf{let}\ r \mathrel{\hat{=}} \langle f, f^{\mathrm{c}}\rangle^{-1}(x, y, x^{\mathrm{c}}, y^{\mathrm{c}})\ \mathbf{in}\ \texttt{<book>}(\texttt{<title>}(t)\,\textbf{.}\,as\,\textbf{.}\,p)\,\textbf{.}\,r\\
&\langle f, f^{\mathrm{c}}\rangle^{-1}(x, \texttt{<reference>}(\texttt{<title>}(t)\,\textbf{.}\,a)\,\textbf{.}\,y, \mathsf{B}_{41}(x^{\mathrm{c}}, p), \mathsf{B}_{42}(y^{\mathrm{c}}, a^{\mathrm{c}}, p))\\
&\qquad \mathrel{\hat{=}} \mathbf{let}\ r \qquad\qquad\ \mathrel{\hat{=}} \langle f, f^{\mathrm{c}}\rangle^{-1}(x, y, x^{\mathrm{c}}, y^{\mathrm{c}})\\
&\qquad\qquad\quad e :: \mathit{Editors} \mathrel{\hat{=}} \langle \mathit{affil}, \mathit{affil}^{\mathrm{c}}\rangle^{-1}(a, a^{\mathrm{c}})\\
&\qquad\quad\ \mathbf{in}\ \texttt{<book>}(\texttt{<title>}(t)\,\textbf{.}\,e\,\textbf{.}\,p)\,\textbf{.}\,r\\
&\langle \mathit{affil}, \mathit{affil}^{\mathrm{c}}\rangle^{-1}(\varepsilon, \mathsf{B}_{51}) \mathrel{\hat{=}} \varepsilon\\
&\langle \mathit{affil}, \mathit{affil}^{\mathrm{c}}\rangle^{-1}(\texttt{<affilication>}(a)\,\textbf{.}\,x, \mathsf{B}_{61}(x^{\mathrm{c}}, n, m))\\
&\qquad \mathrel{\hat{=}} \mathbf{let}\ r \mathrel{\hat{=}} \langle \mathit{affil}, \mathit{affil}^{\mathrm{c}}\rangle^{-1}(x, x^{\mathrm{c}})\\
&\qquad\quad\ \mathbf{in}\ \texttt{<editor>}(\texttt{<last>}(n)\,\textbf{.}\,\texttt{<first>}(m)\,\textbf{.}\,\texttt{<affiliation>}(a))\,\textbf{.}\,r
\end{aligned}
$$

この逆方向変換 $\mathit{bkref}_{\mathrm{B}}$ を用いることで，得られた文献リストのうち，文献名（<title>），著者（<author>），所属（<affiliation>）を自由に変更することができる．特に，一つの書籍の著者については名前の変更だけではなく挿入や削除も可能である．しかし，書籍（<book>）や参考文献（<reference>）を削除したり挿入したりすることはできない．これは，*bkref* が文献の出版社（<publisher>）や価格（<price>）の情報を失っているため，文献の削除や挿入を許すと，文献とその出版者と価格との対応が失われてしまうためである．

導出される逆方向変換があまり効果的なものにならない例もいくつか存在する．以下に典型的な例を三つ挙げる．

例 8.8 (文脈自由性). 以下の変換 *tree2paren* を考える．

$$
\begin{aligned}
&\mathbf{data}\ B \mathrel{\hat{=}} \texttt{<bin>}(B\,\textbf{.}\,B) \mid \varepsilon\\
&\mathit{tree2paren}(\varepsilon) \mathrel{\hat{=}} \varepsilon\\
&\mathit{tree2paren}(\texttt{<bin>}(x :: B\,\textbf{.}\,y :: B))\\
&\qquad \mathrel{\hat{=}} \mathbf{let}\ s \mathrel{\hat{=}} \mathit{tree2paren}(x)\\
&\qquad\qquad\quad\ \ t \mathrel{\hat{=}} \mathit{tree2paren}(y)\\
&\qquad\quad\ \mathbf{in}\ \texttt{<L>}\,\textbf{.}\,s\,\textbf{.}_{(1)}\,t\,\textbf{.}\,\texttt{<R>}
\end{aligned}
$$

上の変換 *tree2paren* は，二分木を開き括弧 `<L>` と閉じ括弧 `<R>` でエンコードする．たとえば，

$$\texttt{<bin>}(\texttt{<bin>}(\texttt{<bin>})\,\texttt{.}\,\texttt{<bin>})$$

に対し，*tree2paren* は以下を出力する．

$$\texttt{<L>.<L>.<L>.<R>.<R>.<L>.<R>.<R>}$$

関数 *tree2paren* は単射である．しかし，本章の単射性判定は *tree2paren* を単射と判定しない．これは，*tree2paren* の値域を記述する文法

$$T_{\mathit{tree2paren}} \to \varepsilon \quad T_{\mathit{tree2paren}} \to \texttt{<L>}T_{\mathit{tree2paren}}T_{\mathit{tree2paren}}\texttt{<R>}$$

を近似して得られる強正規文脈自由生垣文法

$$\begin{array}{lll} T_{\mathit{tree2paren}} \to T^{\mathrm{R}}_{\mathit{tree2paren}} & T_{\mathit{tree2paren}} \to \texttt{<L>}T_{\mathit{tree2paren}} & \\ T^{\mathrm{R}}_{\mathit{tree2paren}} \to T_{\mathit{tree2paren}} & T^{\mathrm{R}}_{\mathit{tree2paren}} \to \texttt{<R>}T^{\mathrm{R}}_{\mathit{tree2paren}} & T^{\mathrm{R}}_{\mathit{tree2paren}} \to \varepsilon \end{array}$$

により，*tree2paren* の近似された値域は正規表現 $(\texttt{<L>}|\texttt{<R>})^*$ と同じものとなり，本来は単射である連接 $s\,\texttt{.}\,t$ が非単射であると判定されるためである．

よって，ALG-C$^+$ の返す補関数

$$\begin{array}{l} \mathit{tree2paren}^{\mathrm{c}}(\varepsilon) \mathrel{\hat{=}} \mathsf{B}_{11} \\ \mathit{tree2paren}^{\mathrm{c}}(\texttt{<bin>}(x :: B\,\texttt{.}\,y :: B)) \\ \quad \mathrel{\hat{=}} \mathbf{let}\; s \mathrel{\hat{=}} \mathit{tree2paren}(x) \\ \qquad\quad s^{\mathrm{c}} \mathrel{\hat{=}} \mathit{tree2paren}^{\mathrm{c}}(x) \\ \qquad\quad t^{\mathrm{c}} \mathrel{\hat{=}} \mathit{tree2paren}^{\mathrm{c}}(y) \\ \qquad \mathbf{in}\; \mathsf{B}_{21}(\textsc{length}_1(s), s^{\mathrm{c}}, t^{\mathrm{c}}) \end{array}$$

に基づき逆方向変換を得ると以下となる．

$$\begin{array}{l} \mathit{tree2paren}_{\mathrm{B}}(s, v) \mathrel{\hat{=}} \langle \mathit{tree2paren}, \mathit{tree2paren}^{\mathrm{c}} \rangle^{-1}(v, \mathit{tree2paren}^{\mathrm{c}}(s)) \\ \langle \mathit{tree2paren}, \mathit{tree2paren}^{\mathrm{c}} \rangle^{-1}(\varepsilon, \mathsf{B}_{11}) \mathrel{\hat{=}} \varepsilon \\ \langle \mathit{tree2paren}, \mathit{tree2paren}^{\mathrm{c}} \rangle^{-1}(\texttt{<L>}\,\texttt{.}\,v_1\,\texttt{.}\,\texttt{<R>}, \mathsf{B}_{21}(l_1, s^{\mathrm{c}}, t^{\mathrm{c}})) \\ \quad \mathrel{\hat{=}} \mathbf{let}\; (s, t) \mathrel{\hat{=}} \textsc{splitAt}(l_1, v_1) \\ \qquad\quad x \mathrel{\hat{=}} \langle \mathit{tree2paren}, \mathit{tree2paren}^{\mathrm{c}} \rangle^{-1}(s, s^{\mathrm{c}}) \\ \qquad\quad y \mathrel{\hat{=}} \langle \mathit{tree2paren}, \mathit{tree2paren}^{\mathrm{c}} \rangle^{-1}(t, t^{\mathrm{c}}) \\ \qquad \mathbf{in}\; \texttt{<bin>}(x\,\texttt{.}\,y) \end{array}$$

しかし，この逆方向変換を用いては恒等更新以外の更新が反映できない．

我々は，上の *tree2paren* に対し効果的な逆方向変換が求まらないことを本章の補関数導出手法の致命的な問題であるとは考えない．なぜなら，我々が対象として考えているのは，木構造から木構造への変換であって，木構造から文字列の変換ではない．また，ビューが生垣であるにもかかわらず，木構造そのものをもってデータ構造を表現するのではなく，*tree2paren* のビューのように括弧のようなものを用いて構造を表現するのは不自然である．

例 8.9 (同期情報の影響)**.** 以下の変換 *halfeven* を考える．

$$
\begin{aligned}
&\mathit{halfeven}(x) \mathrel{\hat{=}} \textbf{let}\,(s,t) \mathrel{\hat{=}} f(x)\,\textbf{in}\ \texttt{<half>}(s)\,\texttt{.}\,\texttt{<isEven>}(t)\\
&f(\texttt{<z>}) \qquad\ \mathrel{\hat{=}} (\texttt{<z>}, \texttt{<true>})\\
&f(\texttt{<s>}\,\texttt{.}\,x) \mathrel{\hat{=}} \textbf{let}\,(s,t) \mathrel{\hat{=}} g(x)\,\textbf{in}\ (s,t)\\
&g(\texttt{<z>}) \qquad\ \mathrel{\hat{=}} (\texttt{<z>}, \texttt{<false>})\\
&g(\texttt{<s>}\,\texttt{.}\,x) \mathrel{\hat{=}} \textbf{let}\,(s,t) \mathrel{\hat{=}} f(x)\,\textbf{in}\ (\texttt{<s>}\,\texttt{.}\,s,t)
\end{aligned}
$$

関数 *halfeven* は入力の自然数を x とすると，$\lfloor x/2 \rfloor$ と x の偶奇を返す．よって，*halfeven* は単射である．しかし，本章の式の近似された値域の上での単射性判定は *halfeven* を単射であると判定しない．これは，f の第二規則の右辺式の近似された値域が $(\texttt{<s>}^{*}\texttt{.<z>}, (\texttt{<true>}|\texttt{<false>}))$ となるため，f の第一規則の右辺式の近似された値域 $\{(\texttt{<z>}, \texttt{<false>})\}$ と重なりがあるためである．同期情報を失っているため，関数 f の返り値において，第一要素が $\texttt{<z>}$ なら第二要素は $\texttt{<true>}$ であることは近似された値域の上ではわからないことに注意する．よって，Alg-C^{+} の返す補関数

$$
\begin{aligned}
&\mathit{halfeven}^{\mathrm{c}}(x) \mathrel{\hat{=}} \textbf{let}\,(s^{\mathrm{c}},t^{\mathrm{c}}) \mathrel{\hat{=}} f^{\mathrm{c}}(x)\,\textbf{in}\ \mathsf{B}_{11}(s^{\mathrm{c}},t^{\mathrm{c}})\\
&f^{\mathrm{c}}(\texttt{<z>}) \qquad \mathrel{\hat{=}} (\mathsf{B}_{21}, \mathsf{B}_{22})\\
&f^{\mathrm{c}}(\texttt{<s>}\,\texttt{.}\,x) \mathrel{\hat{=}} \textbf{let}\,(s^{\mathrm{c}},t^{\mathrm{c}}) \mathrel{\hat{=}} g^{\mathrm{c}}(x)\,\textbf{in}\ (\mathsf{B}_{31}(s^{\mathrm{c}}), \mathsf{B}_{32}(t^{\mathrm{c}}))\\
&g^{\mathrm{c}}(\texttt{<z>}) \qquad \mathrel{\hat{=}} (\mathsf{B}_{41}, \mathsf{B}_{42})\\
&g^{\mathrm{c}}(\texttt{<s>}\,\texttt{.}\,x) \mathrel{\hat{=}} \textbf{let}\,(s^{\mathrm{c}},t^{\mathrm{c}}) \mathrel{\hat{=}} f^{\mathrm{c}}(x)\,\textbf{in}\ (s^{\mathrm{c}},t^{\mathrm{c}})
\end{aligned}
$$

に基づき逆方向変換を得ると以下となる．

$$
\begin{aligned}
&\mathit{halfeven}_{\mathrm{B}}(s,v) \mathrel{\hat{=}} \langle f, f^{\mathrm{c}}\rangle^{-1}(v, f^{\mathrm{c}}(s))\\
&\langle \mathit{halfeven}, \mathit{halfeven}^{\mathrm{c}}\rangle^{-1}(\texttt{<half>}(s)\,\texttt{.}\,\texttt{<isEven>}(t), \mathsf{B}_{11}(s^{\mathrm{c}},t^{\mathrm{c}}))\\
&\qquad \mathrel{\hat{=}} \textbf{let}\,x \mathrel{\hat{=}} \langle f, f^{\mathrm{c}}\rangle^{-1}(s,t,s^{\mathrm{c}},t^{\mathrm{c}})\,\textbf{in}\ x\\
&\langle f, f^{\mathrm{c}}\rangle^{-1}(\texttt{<z>}, \mathsf{true}, \mathsf{B}_{21}, \mathsf{B}_{22}) \quad \mathrel{\hat{=}} \texttt{<z>}\\
&\langle f, f^{\mathrm{c}}\rangle^{-1}(s,t,\mathsf{B}_{31}(s^{\mathrm{c}}), \mathsf{B}_{32}(t^{\mathrm{c}})) \mathrel{\hat{=}} \textbf{let}\,x \mathrel{\hat{=}} \langle g, g^{\mathrm{c}}\rangle^{-1}(s,t,s^{\mathrm{c}},t^{\mathrm{c}})\,\textbf{in}\ \texttt{<s>}\,\texttt{.}\,x\\
&\langle g, g^{\mathrm{c}}\rangle^{-1}(\texttt{<z>}, \mathsf{false}, \mathsf{B}_{41}, \mathsf{B}_{42}) \quad \mathrel{\hat{=}} \texttt{<z>}\\
&\langle g, g^{\mathrm{c}}\rangle^{-1}(\texttt{<s>}\,\texttt{.}\,s,t,s^{\mathrm{c}},t^{\mathrm{c}}) \qquad \mathrel{\hat{=}} \textbf{let}\,x \mathrel{\hat{=}} \langle f, f^{\mathrm{c}}\rangle^{-1}(s,t,s^{\mathrm{c}},t^{\mathrm{c}})\,\textbf{in}\ \texttt{<s>}\,\texttt{.}\,x
\end{aligned}
$$

補関数 $\mathit{halfeven}^{\mathrm{c}}$ は入力の自然数 x の $\lceil x/2 \rceil$（$\mathsf{B}_{3_}$ の数）と偶奇（$\mathsf{B}_{2_}$ か $\mathsf{B}_{3_}$）を返すため単射である．上の逆方向変換は $\mathit{halfeven}_{\mathrm{B}}$ は，恒等更新しか反映できないため，あまり効果的なものではない．

なお，以下のように定義された等価な関数 $\mathit{halfeven}'$ については，近似された値域の上の単射性判定により $\mathit{halfeven}'$ の単射性を判定することができる．

$$
\begin{aligned}
&\mathit{halfeven}'(x) \mathrel{\hat{=}} \textbf{let}\,(s,t) \mathrel{\hat{=}} f(x)\,\textbf{in}\ \texttt{<half>}(s)\,\texttt{.}\,\texttt{<isEven>}(t)\\
&f(\texttt{<z>}) \qquad\qquad \mathrel{\hat{=}} (\texttt{<z>}, \texttt{<true>})\\
&f(\texttt{<s>}\,\texttt{.}\,\texttt{<z>}) \qquad \mathrel{\hat{=}} (\texttt{<z>}, \texttt{<false>})\\
&f(\texttt{<s>}\,\texttt{.}\,\texttt{<s>}\,\texttt{.}\,x) \mathrel{\hat{=}} \textbf{let}\,(s,t) \mathrel{\hat{=}} f(x)\,\textbf{in}\ (\texttt{<s>}\,\texttt{.}\,s,t)
\end{aligned}
$$

これは，f の右辺式の近似された値域に明らかに重なりがないためである．

例 8.10 (必要のない多返値). 以下の変換 thalf を考える.

$$
\begin{aligned}
&\mathit{tupledHalf}(x) \doteq \mathbf{let}\,(s,t) \doteq f(x)\,\mathbf{in}\; s\\
&f(\text{<z>}) \qquad\;\; \doteq (\text{<z>},\text{<z>})\\
&f(\text{<s>}\,\text{.}\,x) \doteq \mathbf{let}\,(s,t) \doteq f(x)\,\mathbf{in}\;(t,\text{<s>}\,\text{.}\,s)
\end{aligned}
$$

この関数 $\mathit{tupledHalf}$ は入力の自然数 x に対し，$\lfloor x/2 \rfloor$ を計算する．ここで，f は単射であり，また本章の単射性判定により，f は単射と判定することができる．よって $\textsc{Alg-C}^{+}$ の導出する補関数

$$\mathit{tupledHalf}^{\mathrm{c}}(x) \doteq \mathbf{let}\,(s,t) \doteq f(x)\,\mathbf{in}\; t$$

によって定義される逆方向変換は以下である．

$$
\begin{aligned}
&\mathit{tupledHalf}_{\mathrm{B}}(s,v) \doteq \langle \mathit{tupledHalf}, \mathit{tupledHalf}^{\mathrm{c}}\rangle^{-1}(v, \mathit{tupledHalf}^{\mathrm{c}}(s))\\
&\langle \mathit{tupledHalf}, \mathit{tupledHalf}^{\mathrm{c}}\rangle^{-1}(s,t) \doteq \mathbf{let}\, x \doteq f^{-1}(s,t)\,\mathbf{in}\; x\\
&f(\text{<z>}) \qquad\;\; \doteq (\text{<z>},\text{<z>})\\
&f(\text{<s>}\,\text{.}\,x) \doteq \mathbf{let}\,(s,t) \doteq f(x)\,\mathbf{in}\;(t,\text{<s>}\,\text{.}\,s)\\
&f^{-1}(\text{<z>},\text{<z>}) \doteq \text{<z>}\\
&f^{-1}(t,\text{<s>}\,\text{.}\,s) \doteq \mathbf{let}\, x \doteq f^{-1}(s,t)\,\mathbf{in}\;(\text{<s>}\,\text{.}\,x)
\end{aligned}
$$

補関数 $\mathit{tupledHalf}^{\mathrm{c}}$ は，入力の自然数 x に対し，$\lceil x/2 \rceil$ を計算する．補関数 $\mathit{tupledHalf}^{\mathrm{c}}$ から得られる逆方向変換 $\mathit{tupledHalf}_{\mathrm{B}}$ はあまり効果的なものではない．逆方向変換 $\mathit{tupledHalf}_{\mathrm{B}}$ のもとで反映可能な更新は，恒等更新を除けば，ソースの奇数ならビューの値を 1 増やす更新，ソースが偶数ならビューの値を 1 減らすか更新の二つのみである．

なお，上の $\mathit{tupledHalf}$ と等価な関数 half

$$
\begin{aligned}
&\mathit{half}(\text{<z>}) \qquad\qquad\; \doteq \text{<z>}\\
&\mathit{half}(\text{<s>}\,\text{.}\,\text{<z>}) \qquad \doteq \text{<z>}\\
&\mathit{half}(\text{<s>}\,\text{.}\,\text{<s>}\,\text{.}\,x) \doteq \mathbf{let}\, t \doteq \mathit{half}(x)\,\mathbf{in}\;\text{<s>}\,\text{.}\,t
\end{aligned}
$$

については，もっと小さい以下の補関数が得られる．

$$
\begin{aligned}
&\mathit{half}^{\mathrm{c}}(\text{<z>}) \qquad\quad\;\; \doteq \mathsf{B}_{11}\\
&\mathit{half}^{\mathrm{c}}(\text{<s>}\,\text{.}\,\text{<z>}) \;\doteq \mathsf{B}_{21}\\
&\mathit{half}^{\mathrm{c}}(\text{<s>}\,\text{.}\,\text{<s>.}) \doteq \mathbf{let}\, t^{\mathrm{c}} \doteq \mathit{half}^{\mathrm{c}}(x)\,\mathbf{in}\; t^{\mathrm{c}}
\end{aligned}
$$

直観的には，$\mathit{half}^{\mathrm{c}}$ は入力の数の偶奇を判定している．補関数 $\mathit{half}^{\mathrm{c}}$ を用いて定義される逆関数は以下となる．

$$
\begin{aligned}
&\mathit{half}_{\mathrm{B}}(s,v) \doteq \langle \mathit{half}, \mathit{half}^{\mathrm{c}}\rangle^{-1}(v, \mathit{half}^{\mathrm{c}}(s))\\
&\langle \mathit{half}, \mathit{half}^{\mathrm{c}}\rangle^{-1}(\text{<z>},\mathsf{B}_{11}) \;\doteq \text{<z>}\\
&\langle \mathit{half}, \mathit{half}^{\mathrm{c}}\rangle^{-1}(\text{<z>},\mathsf{B}_{21}) \;\doteq \text{<z>}\\
&\langle \mathit{half}, \mathit{half}^{\mathrm{c}}\rangle^{-1}(\text{<s>}\,\text{.}\,t,t^{\mathrm{c}}) \doteq \mathbf{let}\, x \doteq \langle \mathit{half}, \mathit{half}^{\mathrm{c}}\rangle^{-1}(t,t^{\mathrm{c}})\,\mathbf{in}\;\text{<s>}\,\text{.}\,x
\end{aligned}
$$

逆方向変換 $\mathit{half}_{\mathrm{B}}$ は，half のビューの上の任意の更新をソースに反映することができる．そのため，$\mathit{half}^{\mathrm{c}}$ は，half の補関数として極小のものである．

$$\frac{}{\varepsilon \Downarrow \varepsilon}\textsc{Eps} \quad \frac{e \Downarrow v}{\sigma(e) \Downarrow \sigma(v)}\textsc{Con} \quad \frac{e_1 \Downarrow v_1 \quad e_2 \Downarrow v_2}{e_1 \centerdot e_2 \Downarrow v_1 \centerdot v_2}\textsc{Cat}$$

$$\frac{\exists f(\overrightarrow{p}) \mathrel{\hat{=}} \mathbf{let}\, L\, \mathbf{in}\ (\overrightarrow{q}) \quad \exists\theta.\ \overrightarrow{p}\theta = \overrightarrow{u} \qquad L, \theta \Downarrow\!\!\Downarrow \sigma \quad \overrightarrow{v} = \overrightarrow{q}\sigma}{f(\overrightarrow{u}) \Downarrow \overrightarrow{v}}\textsc{Fun}$$

$$\frac{\{\overrightarrow{x}\} \subseteq \mathsf{dom}(\sigma), f(\overrightarrow{x}\sigma) \Downarrow \overrightarrow{v}, \qquad \exists\eta.\ \overrightarrow{p}\eta = \overrightarrow{u}, \mathsf{dom}(\eta) = \mathsf{vars}(\overrightarrow{p}) \quad L, (\eta \circ \sigma) \Downarrow\!\!\Downarrow \sigma'}{((\overrightarrow{p}) \mathrel{\hat{=}} f(\overrightarrow{x})) \cup L, \sigma \Downarrow\!\!\Downarrow \sigma'}\textsc{Bind1}$$

$$\frac{}{\emptyset, \sigma \Downarrow\!\!\Downarrow \sigma}\textsc{Bind2}$$

図 8.2. 拡張された言語 $\textsc{Vdl}^{+}$ の操作的意味

逆関数の生成の正しさ

逆関数の生成アルゴリズムが正しいことを証明する．逆関数の生成アルゴリズムの出力するプログラムは，第 6 章で定義したプログラムよりも拡張されていて，**let** 束縛に評価順序の依存性がある．よって，逆関数の生成アルゴリズムが正しさの証明のために，アルゴリズムが出力するプログラムの形式的な意味を図 8.2 に定めておく．

正しさの証明のために，以下の補題を用意する．直観的には，以下の補題は導出された逆関数において，補関数に含まれる生垣の長さの情報を用いて連接を適切に分解し，入力を $\Gamma, q \overset{\mathrm{p}}{\rightsquigarrow} q_1''$ となる q 程度まで分解できることを表現している．

補題 8.4. 変数の集合 Y と式 q について

$$\Gamma, Y, q \overset{\mathrm{c}}{\rightsquigarrow} \overrightarrow{q^{\mathrm{c}}} \quad \Gamma, q \overset{\mathrm{p}}{\rightsquigarrow} q_1'' \quad \overrightarrow{q^{\mathrm{c}}} \overset{\mathrm{cp}}{\rightsquigarrow} \overrightarrow{q}''_2 \quad \Gamma, \mathsf{vars}(\overrightarrow{q^{\mathrm{c}}}), \overrightarrow{q} \overset{\mathrm{let}}{\rightsquigarrow} L$$

となっているとする．このとき，

$$\exists\eta.\ (q_1'', \overrightarrow{q}''_2)\eta{\downarrow} \wedge \mathsf{vars}((q_1'', \overrightarrow{q}''_2)\eta) = \emptyset \wedge L, \eta \Downarrow\!\!\Downarrow \eta' \Rightarrow (q, \overrightarrow{q}')\eta' = (q_1'', \overrightarrow{q}''_2)\eta$$

となる．

証明. 式 q の構造に関する帰納法により示す.

基底：$q = \varepsilon$.

このとき，$q_1'' = \varepsilon$ かつ $\overrightarrow{q^{\mathrm{c}}} = \overrightarrow{q}''_2 = ()$ となるため，自明.

基底：$q = y \in Y$．

このとき，$q_1'' = y :: \Gamma(y)$ かつ $\overrightarrow{q^{\mathrm{c}}} = \overrightarrow{q}''_2 = y^{\mathrm{c}}$ であり，$L = \emptyset$ となるため，自明.

基底：$q = y \in Y$．

このとき，$q_1'' = y :: \Gamma(y)$ かつ $\overrightarrow{q^{\mathrm{c}}} = \overrightarrow{q}''_2 = ()$ であり，$L = \emptyset$ となるため，自明.

帰納：$q = \sigma(r)$．

このとき，$q_1'' = \sigma(r_1'')$ かつ $\overrightarrow{q^{\mathrm{c}}} = \overrightarrow{r^{\mathrm{c}}}$，そして $\overrightarrow{q}''_2 = \overrightarrow{r''}_2$ となる．ただし，

$$\Gamma, Y, r \overset{\mathrm{c}}{\leadsto} \overrightarrow{r^{\mathrm{c}}} \quad \Gamma, r \overset{\mathrm{p}}{\leadsto} r_1'' \quad \overrightarrow{r^{\mathrm{c}}} \overset{\mathrm{cp}}{\leadsto} \overrightarrow{r}''_2 \quad \Gamma, \mathsf{vars}(\overrightarrow{r^{\mathrm{c}}}), \overrightarrow{r} \overset{\mathrm{let}}{\leadsto} L$$

となる．今，$(q_1'', \overrightarrow{q}''_2)\eta{\downarrow}$ かつ $\mathsf{vars}((q_1'', \overrightarrow{q}''_2)\eta) = \emptyset$ となり，

$$L, \eta \downdownarrows \eta'$$

となったとする．帰納法の仮定より，

$$(r, \overrightarrow{r^{\mathrm{c}}})\eta' = (r_1'', \overrightarrow{r}''_2)\eta$$

となる．よって，

$$(\sigma(r), \overrightarrow{r^{\mathrm{c}}})\eta' = (\sigma(r_1''), \overrightarrow{r}''_2)\eta$$

よって，主張は真.

帰納：$q = r \centerdot s$．

まず，$\mathsf{ran}_\Gamma(r) \parallel \mathsf{ran}_\Gamma(s)$ である場合を考える．このとき，$q_1'' = r_1'' \centerdot s_1''$ かつ $\overrightarrow{q^{\mathrm{c}}} = \overrightarrow{r^{\mathrm{c}}}, \overrightarrow{s^{\mathrm{c}}}$，そして，$\overrightarrow{q}''_2 = \overrightarrow{r''_2}, \overrightarrow{s''_2}$ かつ $L = L_1 \cup L_2$ となる．ただし，

$$\begin{array}{llll}
\Gamma, Y, r \overset{\mathrm{c}}{\leadsto} \overrightarrow{r^{\mathrm{c}}} & \Gamma, r \overset{\mathrm{p}}{\leadsto} r_1'' & \overrightarrow{r^{\mathrm{c}}} \overset{\mathrm{cp}}{\leadsto} \overrightarrow{r}''_2 & \Gamma, \mathsf{vars}(\overrightarrow{r^{\mathrm{c}}}), \overrightarrow{r} \overset{\mathrm{let}}{\leadsto} L_1 \\
\Gamma, Y, s \overset{\mathrm{c}}{\leadsto} \overrightarrow{s^{\mathrm{c}}} & \Gamma, s \overset{\mathrm{p}}{\leadsto} s_1'' & \overrightarrow{s^{\mathrm{c}}} \overset{\mathrm{cp}}{\leadsto} \overrightarrow{s}''_2 & \Gamma, \mathsf{vars}(\overrightarrow{s^{\mathrm{c}}}), \overrightarrow{s} \overset{\mathrm{let}}{\leadsto} L_2
\end{array}$$

である．今，$(q_1'', \overrightarrow{q}''_2)\eta{\downarrow}$ かつ $\mathsf{vars}((q_1'', \overrightarrow{q}''_2)\eta) = \emptyset$ となり，

$$L, \eta \downdownarrows \eta'$$

となったとする．ここで，$L_1, \eta \downdownarrows \zeta$ および $L_2, \eta \downdownarrows \xi$ とし，L に含まれる束縛規則の左辺に登場する変数全てからなる集合を $\mathsf{leftvars}(L)$ と書いたとすると，$\mathsf{leftvars}(L_1) \cap \mathsf{leftvars}(L_2) = \emptyset$ であるため，

$$r\eta' = r\zeta \wedge s\eta' = s\xi$$

となる．帰納法の仮定より，

$$(r, \overrightarrow{r^{\mathrm{c}}})\zeta = (r''_1, \overrightarrow{r}''_2)\eta$$

および

$$(s, \overrightarrow{s^{\mathrm{c}}})\xi = (s''_1, \overrightarrow{s}''_2)\eta$$

となる．よって，

$$(r \centerdot s, \overrightarrow{r^{\mathrm{c}}}, \overrightarrow{s^{\mathrm{c}}})\eta' = (r''_1 \centerdot s''_1, \overrightarrow{r}''_2, \overrightarrow{s}''_2)\eta$$

となり，主張は真．

次に，$\mathsf{ran}_\Gamma(r) \not\parallel \mathsf{ran}_\Gamma(s)$ である場合を考える．このとき，$q''_1 = v_k$ となり，かつ $\overrightarrow{q^{\mathrm{c}}} = \textsc{length}_k(r), \overrightarrow{r^{\mathrm{c}}}, \overrightarrow{s^{\mathrm{c}}}$ となる．そして，$\overrightarrow{q}''_2 = l_k, \overrightarrow{r''_2}, \overrightarrow{s''_2}$ かつ $L = \{(r''_1, s''_1) \hat{=} \textsc{splitAt}(l_k, v_k)\} \cup L_1 \cup L_2$ となる．ただし，

$$\begin{array}{llll} \Gamma, Y, r \overset{\mathrm{c}}{\rightsquigarrow} \overrightarrow{r^{\mathrm{c}}} & \Gamma, r \overset{\mathrm{p}}{\rightsquigarrow} r''_1 & \overrightarrow{r^{\mathrm{c}}} \overset{\mathrm{cp}}{\rightsquigarrow} \overrightarrow{r}''_2 & \Gamma, \mathsf{vars}(\overrightarrow{r^{\mathrm{c}}}), \overrightarrow{r} \overset{\mathrm{let}}{\rightsquigarrow} L_1 \\ \Gamma, Y, s \overset{\mathrm{c}}{\rightsquigarrow} \overrightarrow{s^{\mathrm{c}}} & \Gamma, s \overset{\mathrm{p}}{\rightsquigarrow} s''_1 & \overrightarrow{s^{\mathrm{c}}} \overset{\mathrm{cp}}{\rightsquigarrow} \overrightarrow{s}''_2 & \Gamma, \mathsf{vars}(\overrightarrow{s^{\mathrm{c}}}), \overrightarrow{s} \overset{\mathrm{let}}{\rightsquigarrow} L_2 \end{array}$$

である．今，$(q''_1, \overrightarrow{q}''_2)\eta\downarrow$ かつ $\mathsf{vars}((q''_1, \overrightarrow{q}''_2)\eta) = \emptyset$ となり，

$$L, \eta \downdownarrows \eta'$$

となったとする．ここで，$L = \{(r''_1, s''_1) \hat{=} \textsc{splitAt}(l_k, v_k)\} \cup L_1 \cup L_2$ であるために，

$$(r''_1, s''_2)\theta = \textsc{splitAt}(l_k\eta, v_k\eta)$$

と定められる θ により，

$$L_1 \cup L_2, \theta \circ \eta \downdownarrows \eta'$$

となったとする．ここで，$L_1, \eta \downdownarrows \zeta$ および $L_2, \eta \downdownarrows \xi$ とし，L に含まれる束縛規則の左辺に登場する変数全てからなる集合を $\mathsf{leftvars}(L)$ と書いたとすると，$\mathsf{leftvars}(L_1) \cap \mathsf{leftvars}(L_2) = \emptyset$ であるため，

$$r\eta' = r\zeta \wedge s\eta' = s\xi$$

となる．帰納法の仮定より，

$$(r, \overrightarrow{r^{\mathrm{c}}})\zeta = (r''_1, \overrightarrow{r}''_2)\theta$$

および

$$(s, \overrightarrow{s^{\mathrm{c}}})\xi = (s''_1, \overrightarrow{s}''_2)\theta$$

となる．よって，

$$(r \centerdot s, \overrightarrow{r^{\mathrm{c}}}, \overrightarrow{s^{\mathrm{c}}})\eta' = (r''_1 \centerdot s''_1, \overrightarrow{r}''_2, \overrightarrow{s}''_2)\theta$$

となる．関数 $\textsc{splitAt}$ の性質により，

$$(r''_1 \centerdot s''_1)\theta = v_k\eta$$

かつ

$$\textsc{length}(r''_1\theta) = l_k\eta$$

となる．よって，

$$(r \centerdot s, \overrightarrow{r^{\mathrm{c}}}, \overrightarrow{s^{\mathrm{c}}})\eta' = (r''_1 \centerdot s''_1, \overrightarrow{r}''_2, \overrightarrow{s}''_2)\theta = (v_k, l_k, \overrightarrow{r}''_2, \overrightarrow{s}''_2)\eta$$

となり，主張は真. □

系 8.1. 互いに素な変数の集合 Y, V と木上の文脈 $\mathcal{C}_i$ および式の列 $\overrightarrow{q}$ について

$$\begin{aligned}&\Gamma, Y, q_i \overset{\mathrm{c}}{\leadsto} \overrightarrow{q_i}^{\mathrm{c}},\\ &\overrightarrow{q}' = (\mathcal{C}_1[\overrightarrow{q_i}^{\mathrm{c}}, V], \ldots, \mathcal{C}_m[\overrightarrow{q_m}^{\mathrm{c}}, V])\end{aligned}$$

かつ

$$\Gamma, Y, \overrightarrow{q} \overset{\mathrm{c}}{\leadsto} \overrightarrow{q}' \quad \Gamma, \overrightarrow{q} \overset{\mathrm{p}}{\leadsto} \overrightarrow{q}''_1 \quad \overrightarrow{q}' \overset{\mathrm{cp}}{\leadsto} \overrightarrow{q}''_2 \quad \Gamma, \mathsf{vars}(\overrightarrow{q}''_2), \overrightarrow{q} \overset{\mathrm{let}}{\leadsto} L$$

となっているとする．このとき

$$\exists\eta.\ (\overrightarrow{q}''_1, \overrightarrow{q}''_2)\eta{\downarrow} \wedge \mathsf{vars}((\overrightarrow{q}''_1, \overrightarrow{q}''_2)\eta) = \emptyset \wedge L, \eta \Downarrow \eta' \Rightarrow (\overrightarrow{q}, \overrightarrow{q}')\eta' = (\overrightarrow{q}''_1, \overrightarrow{q}''_2)\eta$$

となる．

定理 8.10. 本節の逆関数生成アルゴリズムは正しい．

証明. 以下を示せばよい．

- $f \in \mathit{INJ}_{\mathcal{P}}$ に対し，$f(\overrightarrow{v}) \Downarrow \overrightarrow{u} \Leftrightarrow f^{-1}(\overrightarrow{u}) \Downarrow \overrightarrow{v}$ である．
- $f \notin \mathit{INJ}_{\mathcal{P}}$ に対し，$f(\overrightarrow{v}) \Downarrow \overrightarrow{u}, f^{\mathrm{c}}(\overrightarrow{v}) \Downarrow \overrightarrow{u}' \Leftrightarrow \langle f, f^{\mathrm{c}}\rangle^{-1}(\overrightarrow{u}) \Downarrow \overrightarrow{v}$ である．

このうち，前者は第 5 章の定理 5.6 と同様に証明できる．これは，$f \in \mathit{INJ}_{\mathcal{P}}$ に対し，我々の逆関数の導出は左右入れ換えているだけであり，また f は単射と判定される連接に含まないため，**let** 束縛の評価順序に依存性がないためである．よって，後者のみを議論する．

式 e を評価するのに使用した Fun の数を $e \Downarrow v$ の証明木に沿って縦に数えたもの最大値，つまり深さを，#FunDepth(e) と書く．

まず，$(\Rightarrow)$ を，#FunDepth$(f(\overrightarrow{v}))$ に関する帰納法により示す．

基底：#FunDepth$(f(\overrightarrow{v})) = 0$．

このとき，規則

$$f(\overrightarrow{p}) \mathrel{\hat{=}} \overrightarrow{q}$$

で，代入 θ に対し，$\overrightarrow{p}\theta = \overrightarrow{v}$ となるものが存在する．また，対応する補関数規則は以下の形をしている．

$$f^{\mathrm{c}}(\overrightarrow{p}) \mathrel{\hat{=}} \overrightarrow{q}'$$

このとき，対応する逆関数の規則

$$\langle f, f^{\mathrm{c}}\rangle^{-1}(\overrightarrow{q}''_1, \overrightarrow{q}''_2) \mathrel{\hat{=}} \mathbf{let}\, L \,\mathbf{in}\ \overrightarrow{p}''$$

が存在する．ただし，

$$\overrightarrow{p} \overset{\mathrm{e}}{\rightsquigarrow} \overrightarrow{p}'' \quad \Gamma_{\overrightarrow{p}}, \overrightarrow{q} \overset{\mathrm{p}}{\rightsquigarrow} \overrightarrow{q}''_1, \quad \overrightarrow{q}' \overset{\mathrm{cp}}{\rightsquigarrow} \overrightarrow{q}''_2 \quad \Gamma_{\overrightarrow{p}}, \mathsf{vars}(\overrightarrow{q}''_2), \overrightarrow{q} \overset{\mathrm{let}}{\rightsquigarrow} L$$

である．このとき，$(\overrightarrow{q}''_1, \overrightarrow{q}''_2)\eta = (\overrightarrow{q}, \overrightarrow{q}')\theta$ となる η が存在する．系 8.1 より，

$$L, \eta \Downarrow \eta'$$

ならば $\overrightarrow{q}''\eta = \overrightarrow{q}\theta$ となるため，$\langle f, f^{\mathrm{c}}\rangle^{-1}(\overrightarrow{q}''\eta, \overrightarrow{q}\eta) \Downarrow \overrightarrow{p}\theta$ となり，主張は真．

帰納：$\#\textsc{FunDepth}(f(\overrightarrow{v})) > 0$．

このとき，規則

$$\begin{aligned} r = f(\overrightarrow{p}) \mathrel{\hat{=}} &\mathbf{let}\, \{(\overrightarrow{y_i}) \mathrel{\hat{=}} f_i(\overrightarrow{x_i})\}_{i\in\{1,\ldots,n\}} \\ &\mathbf{in}\ (\overrightarrow{q}) \end{aligned}$$

で，代入 θ に対し，$\overrightarrow{p}\theta = \overrightarrow{v}$ となり $f(\overrightarrow{p}\theta) \Downarrow \overrightarrow{u}$ となるものが存在する．ここで，

$$\begin{aligned} \Gamma &= \Gamma_{\overrightarrow{p}} \cup \{y_{ij} \mapsto \pi_j T_{f_i} \mid i \in \{1, \ldots, n\}\} \\ I &= \{i \mid f_i \in \mathit{INJ}_{\mathcal{P}}\} \\ Y &= \{y_{ij} \mid i \in I\} \end{aligned}$$

である．すると，上の規則に対応した以下の形式の補関数の規則が存在する．

$$\begin{aligned} r^{\mathrm{c}} = f(\overrightarrow{p}) \mathrel{\hat{=}} &\mathbf{let}\, \{(\overrightarrow{y_i^{\mathrm{c}}}) \mathrel{\hat{=}} f_i^{\mathrm{c}}(\overrightarrow{x_i})\}_{i\in\{1,\ldots,n\}, i\in I} \\ &\qquad \{(\overrightarrow{y_i}) \mathrel{\hat{=}} f_i(\overrightarrow{x_i})\}_{i\in\{1,\ldots,n\}} \\ &\mathbf{in}\ \overrightarrow{q}' \end{aligned}$$

ここで，

$$\overrightarrow{q}' = (\mathcal{C}_1[\overrightarrow{q_1^{\mathrm{c}}}, V], \ldots, \mathcal{C}_m[\overrightarrow{q_m^{\mathrm{c}}, V}])$$

である．ただし，$V = \mathsf{lostvars}(r)$ かつ $\Gamma, Y, q_i \overset{\mathrm{c}}{\rightsquigarrow} \overrightarrow{q_i^{\mathrm{c}}}$ であり，

$$\mathcal{C}_i = \begin{cases} \Box_1 & \text{if } \forall i.\ |(\overrightarrow{q_i^{\mathrm{c}}}, V)| = 1, \\ & \quad \wedge\ \mathsf{ran}_\Gamma(\overrightarrow{q}) \text{ が他の } f \text{ の規則の右辺式の近似された値域と互いに素} \\ \mathsf{B}_{ri}(\Box_1) & \text{otherwise} \end{cases}$$

である．このとき，次の形式の対応する逆関数の規則が存在する．

$$\begin{aligned} \langle f, f^{\mathrm{c}}\rangle^{-1}(\overrightarrow{q}''_1, \overrightarrow{q}''_2) \mathrel{\hat{=}} &\mathbf{let}\, \{(\overrightarrow{x_i} :: \overrightarrow{T_i}) \mathrel{\hat{=}} f_i^{-1}(\overrightarrow{y_i})\}_{i\in\{1,\ldots,n\}, i\in I} \\ &\qquad \{(\overrightarrow{x_i} :: \overrightarrow{T_i}) \mathrel{\hat{=}} \langle f_i, f_i^{\mathrm{c}}\rangle^{-1}(\overrightarrow{y_i}, \overrightarrow{y_i^{\mathrm{c}}})\}_{i\in\{1,\ldots,n\}, i\notin I} \\ &\qquad L \\ &\mathbf{in}\ (\overrightarrow{p}'') \end{aligned}$$

ただし，

$$\overrightarrow{p} \overset{\mathrm{e}}{\rightsquigarrow} \overrightarrow{p}'' \qquad \Gamma, \overrightarrow{q} \overset{\mathrm{p}}{\rightsquigarrow} \overrightarrow{q}''_1 \qquad \overrightarrow{q}' \overset{\mathrm{cp}}{\rightsquigarrow} \overrightarrow{q}''_2 \qquad \Gamma, \mathsf{vars}(\overrightarrow{q}''_2), \overrightarrow{q} \overset{\mathrm{let}}{\rightsquigarrow} L$$

であり

$$\overrightarrow{T_i} = (T_{i1}, \ldots, T_{i|\overrightarrow{x_i}|}) = (\Gamma(x_{i1}), \ldots, \Gamma(x_{i|\overrightarrow{x_i}|}))$$

である．

ここで，補題 8.3 により $(\overrightarrow{q}''_1\eta, \overrightarrow{q}''_2\eta) = (\overrightarrow{u}, \overrightarrow{u}')$ となる η が存在する．系 8.1 より，

$$L, \eta \Downarrow\!\!\Downarrow \eta' \Rightarrow (\overrightarrow{q}, \overrightarrow{q}')\eta' = (\overrightarrow{q}''_1, \overrightarrow{q}''_2)\eta$$

となる．ここで，$i \in I$ についてであり，

$$f_i(\overrightarrow{x_i}\theta) \Downarrow \overrightarrow{y_i}\eta'$$

$i \notin I$ について

$$f_i(\overrightarrow{x_i}\theta) \Downarrow \overrightarrow{y_i}\eta' \wedge {f_i}^{\mathrm{c}}(\overrightarrow{x_i}\theta) \Downarrow \overrightarrow{{y_i}^{\mathrm{c}}}\eta'$$

である．よって，$i \in I$ について

$${f_i}^{-1}(\overrightarrow{y_i}\eta') \Downarrow \overrightarrow{x_i}\theta$$

かつ帰納法の仮定より，$i \notin I$ について

$$\langle f_i, {f_i}^{\mathrm{c}} \rangle^{-1}(\overrightarrow{y_i}\eta', \overrightarrow{{y_i}^{\mathrm{c}}}\eta') \Downarrow \overrightarrow{x_i}\theta$$

となる．よって $\langle f, f^{\mathrm{c}} \rangle^{-1}(\overrightarrow{q}''_1\eta, \overrightarrow{q}''_2\eta) \Downarrow \overrightarrow{p}\theta$ となり，主張は真．

次に，$(\Leftarrow)$ を，$\#\textsc{FunDepth}(\langle f, f^{\mathrm{c}} \rangle^{-1}(\overrightarrow{u}, \overrightarrow{u}'))$ に関する帰納法により示す．

基底：$\#\textsc{FunDepth}(\langle f, f^{\mathrm{c}} \rangle^{-1}(\overrightarrow{u}, \overrightarrow{u}')) = 0$**.**

このとき，逆関数の規則

$$\langle f, f^{\mathrm{c}} \rangle^{-1}(\overrightarrow{q}''_1, \overrightarrow{q}''_2) \mathrel{\hat{=}} \mathbf{let}\, L\, \mathbf{in}\ \overrightarrow{p}''$$

で，代入 η により $(\overrightarrow{q}''_1, \overrightarrow{q}''_2)\eta = (\overrightarrow{u}, \overrightarrow{u}')$ となるものが存在する．このとき，対応する順方向変換の規則

$$f(\overrightarrow{p}) \mathrel{\hat{=}} \overrightarrow{q}$$

と，対応する補関数規則

$$f^{\mathrm{c}}(\overrightarrow{p}) \mathrel{\hat{=}} \overrightarrow{q}'$$

が存在する．ただし，

$$\overrightarrow{p} \overset{\mathrm{e}}{\rightsquigarrow} \overrightarrow{p}'' \quad \Gamma_{\overrightarrow{p}}, \overrightarrow{q} \overset{\mathrm{p}}{\rightsquigarrow} \overrightarrow{q}''_1, \quad \overrightarrow{q}' \overset{\mathrm{cp}}{\rightsquigarrow} \overrightarrow{q}''_2 \quad \Gamma_{\overrightarrow{p}}, \mathsf{vars}(\overrightarrow{q}''_2), \overrightarrow{q} \overset{\mathrm{let}}{\rightsquigarrow} L$$

である．系 8.1 より，

$$L, \eta \Downarrow\!\!\Downarrow \eta' \Rightarrow (\overrightarrow{q}, \overrightarrow{q}')\eta' = (\overrightarrow{q}''_1, \overrightarrow{q}''_2)\eta$$

である．このとき，

$$f(\overrightarrow{p}\eta') \Downarrow \overrightarrow{q}\eta' \wedge f^{\mathrm{c}}(\overrightarrow{p}\eta') \Downarrow \overrightarrow{q}\eta'$$

となるので，主張は真．

帰納：$\#\textsc{FunDepth}(\langle f, f^{\mathrm{c}}\rangle^{-1}(\overrightarrow{u}, \overrightarrow{u}')) > 0$**．**

このとき，次の形式の逆関数の規則が存在する．

$$\begin{aligned}\langle f, f^{\mathrm{c}}\rangle^{-1}(\overrightarrow{q}''_1, \overrightarrow{q}''_2) \mathrel{\hat{=}} \mathbf{let}\ & \{(\overrightarrow{x_i} :: \overrightarrow{T_i}) \mathrel{\hat{=}} {f_i}^{-1}(\overrightarrow{y_i})\}_{i\in\{1,\ldots,n\}, i\in I}\\ & \{(\overrightarrow{x_i} :: \overrightarrow{T_i}) \mathrel{\hat{=}} \langle f_i, {f_i}^{\mathrm{c}}\rangle^{-1}(\overrightarrow{y_i}, \overrightarrow{{y_i}^{\mathrm{c}}})\}_{i\in\{1,\ldots,n\}, i\notin I}\\ & L\\ \mathbf{in}\ & (\overrightarrow{p}'')\end{aligned}$$

で，代入 η により $(\overrightarrow{q}''_1, \overrightarrow{q}''_2)\eta = (\overrightarrow{u}, \overrightarrow{u}')$ かつ $\langle f, f^{\mathrm{c}}\rangle^{-1}(\overrightarrow{q}''_1\eta, \overrightarrow{q}''_2\eta) \Downarrow \overrightarrow{v}$ となるものが存在する．このとき，対応する順方向変換の規則

$$\begin{aligned}r = f(\overrightarrow{p}) \mathrel{\hat{=}} \mathbf{let}\ & \{(\overrightarrow{y_i}) \mathrel{\hat{=}} f_i(\overrightarrow{x_i})\}_{i\in\{1,\ldots,n\}}\\ \mathbf{in}\ & (\overrightarrow{q})\end{aligned}$$

および，

$$\Gamma = \Gamma_{\overrightarrow{p}} \cup \{y_{ij} \mapsto \pi_j T_{f_i} \mid i \in \{1, \ldots, n\}\}$$
$$I = \{i \mid f_i \in \mathit{INJ}_{\mathcal{P}}\}$$
$$Y = \{y_{ij} \mid i \in I\}$$

として，

$$\begin{aligned}r^{\mathrm{c}} = f(\overrightarrow{p}) \mathrel{\hat{=}} \mathbf{let}\ & \{(\overrightarrow{{y_i}^{\mathrm{c}}}) \mathrel{\hat{=}} {f_i}^{\mathrm{c}}(\overrightarrow{x_i})\}_{i\in\{1,\ldots,n\}, i\in I}\\ & \{(\overrightarrow{y_i}) \mathrel{\hat{=}} f_i(\overrightarrow{x_i})\}_{i\in\{1,\ldots,n\}}\\ \mathbf{in}\ & \overrightarrow{q}'\end{aligned}$$

となる補関数の規則が存在する．ここで，

$$\overrightarrow{q}' = (\mathcal{C}_1[\overrightarrow{{q_1}^{\mathrm{c}}}, V], \ldots, \mathcal{C}_m[\overrightarrow{q_m}, \overrightarrow{V}])$$

である．ただし，$V = \mathsf{lostvars}(r)$ かつ $\Gamma, Y, q_i \stackrel{\mathrm{c}}{\rightsquigarrow} \overrightarrow{{q_i}^{\mathrm{c}}}$ であり，

$$\mathcal{C}_i = \begin{cases} \square_1 & \text{if } \forall i.\ |(\overrightarrow{{q_i}^{\mathrm{c}}}, V)| = 1, \\ & \quad \wedge\ \mathsf{ran}_{\Gamma}(\overrightarrow{q}) \text{ が他の } f \text{ の規則の右辺式の近似された値域と互いに素} \\ \mathsf{B}_{ri}(\square_1) & \text{otherwise} \end{cases}$$

である．ただし，

$$\overrightarrow{p} \overset{\mathrm{e}}{\rightsquigarrow} \overrightarrow{p}'' \qquad \Gamma, \overrightarrow{q} \overset{\mathrm{p}}{\rightsquigarrow} \overrightarrow{q}''_1 \qquad \overrightarrow{q}' \overset{\mathrm{cp}}{\rightsquigarrow} \overrightarrow{q}''_2 \qquad \Gamma, \mathsf{vars}(\overrightarrow{q}''_2), \overrightarrow{q} \overset{\mathrm{let}}{\rightsquigarrow} L$$

であり

$$\overrightarrow{T_i} = (T_{i1}, \ldots, T_{i|\overrightarrow{x_i}|}) = (\Gamma(x_{i1}), \ldots, \Gamma(x_{i|\overrightarrow{x_i}|}))$$

である．

系 8.1 より，

$$L, \eta \Downarrow \eta' \Rightarrow (\overrightarrow{q}, \overrightarrow{q}')\eta' = (\overrightarrow{q}''_1, \overrightarrow{q}''_2)\eta$$

となる．ここで，$i \in I$ について，

$$f_i{}^{-1}(\overrightarrow{y_i}\eta') \Downarrow \overrightarrow{w_i}$$

$i \notin I$ について，

$$\langle f_i, f_i{}^{\mathrm{c}}\rangle^{-1}(\overrightarrow{y_i}\eta', \overrightarrow{y_i{}^{\mathrm{c}}}\eta') \Downarrow \overrightarrow{w_i}$$

となっている．よって，代入 θ を

$$\overrightarrow{x_i}\theta = \overrightarrow{w_i}$$

とおくと，$\overrightarrow{p}''\theta = \overrightarrow{v}$ となる．このとき，

$$\overrightarrow{x_i}\theta \subseteq [\![\overrightarrow{T_i}]\!]$$

となっているので，$\overrightarrow{p}\theta = \overrightarrow{v}$ でもある．ここで，$i \in I$ について

$$f_i(\overrightarrow{x_i}\theta) \Downarrow \overrightarrow{y_i}\eta'$$

であり，また，帰納法の仮定により $i \notin I$ について

$$f_i(\overrightarrow{x_i}\theta) \Downarrow \overrightarrow{y_i}\eta' \wedge f_i{}^{\mathrm{c}}(\overrightarrow{x_i}\theta) \Downarrow \overrightarrow{y_i{}^{\mathrm{c}}}\eta'$$

である．よって，$f(\overrightarrow{v}) \Downarrow (\overrightarrow{q}\eta', \overrightarrow{q}'\eta')$ となる．ここで，$(\overrightarrow{q}\eta', \overrightarrow{q}'\eta') = (\overrightarrow{q}''_1, \overrightarrow{q}''_2)\eta = (\overrightarrow{u}, \overrightarrow{u}')$ であったので，$f(\overrightarrow{v}) \Downarrow (\overrightarrow{u}, \overrightarrow{u}')$ となり，主張は真． □

定理 8.11. 本節の逆関数導出アルゴリズムは決定的なプログラムを生成する．

証明の概略．逆関数導出の構成より，元の順方向変換の規則の右辺式の近似された値域と対応する補関数の規則の右辺式の値域の組と，アルゴリズムが生成する対応する規則のパターンにマッチする全ての入力の集合が常に等しくなることによる． □

8.3　より詳細な解析による補関数導出

我々は，補関数導出アルゴリズム $\textsc{Alg-C}^+$ が言語 $\textsc{Vdl}^+$ で記述された順方向変換に対して不十分であるとは考えてはいない．しかし，$\textsc{Alg-C}^+$ があまり小さい補関数を返せない例は前述の例 8.8 や例 8.9 および例 8.10 のようにいくつかあるため，それらの例に対し，どのようにすればより小さい補関数を返すことができるかを議論しておく．

8.3.1　返り値の使用に基づく関数の特化

例 8.10 の示すように，多返値関数の返り値のある一つの要素と別の一つの要素は依存性が強い．つまり，情報を共有することが多いため，多返値関数の返り値の一部のみが補関数に保存されてしまうのを避けたい．ここでは，多返値関数の返り値の使用に基づく関数の特化について述べる．ここで述べる特化手法は，関数から，実際に使用される返り値のみを計算する関数を作成する．

手法を説明するのに，例 8.10 の $\mathit{tupledHalf}$ を用いる．

$$
\begin{aligned}
&\mathit{tupledHalf}(x) &&\mathrel{\hat{=}} \mathbf{let}\,(s,t) \mathrel{\hat{=}} f(x)\,\mathbf{in}\;s\\
&f(\text{<z>}) &&\mathrel{\hat{=}} (\text{<z>},\text{<z>})\\
&f(\text{<s>}\mathbin{\boldsymbol{.}}x) &&\mathrel{\hat{=}} \mathbf{let}\,(s,t) \mathrel{\hat{=}} f(x)\,\mathbf{in}\;(t,\text{<s>}\mathbin{\boldsymbol{.}}s)
\end{aligned}
$$

関数 $\mathit{tupledHalf}$ の $\mathbf{let}$ 束縛において，多返値関数の f の返り値の第一要素しか使用されていないことがわかる．よって，まず，f から，元の関数の返り値の第一要素のみ計算する関数 $f_{\{1\}}$ を作成することを考える．そのため，f のそれぞれの規則を，f の返り値の第一要素のみを計算するように変換する．まず，規則 $f(\text{<z>}) \mathrel{\hat{=}} (\text{<z>},\text{<z>})$ より，規則

$$f_{\{1\}}(\text{<z>}) \mathrel{\hat{=}} \text{<z>}$$

を生成する．次，規則 $f(\text{<s>}\mathbin{\boldsymbol{.}}x) \mathrel{\hat{=}} \mathbf{let}\,(s,t) \mathrel{\hat{=}} f(x)\,\mathbf{in}\;(t,\text{<s>}\mathbin{\boldsymbol{.}}s)$ より，規則

$$f_{\{1\}}(\text{<z>}) \mathrel{\hat{=}} \mathbf{let}\,t \mathrel{\hat{=}} f_{\{2\}}(x)\,\mathbf{in}\;t$$

を生成する．ここで，関数呼出の $f(x)$ の返り値の内，$f_{\{1\}}$ の値の計算には第二要素しか使用されていないため，f の返り値の第二要素のみを計算する関数 $f_{\{2\}}(x)$ が $\mathbf{let}$ 束縛にて呼ばれている．関数 $f_{\{2\}}$ の定義はまだ生成されていないため，$f_{\{2\}}$ を定義する規則を生成することを考える．まず，規則 $f(\text{<z>}) \mathrel{\hat{=}} (\text{<z>},\text{<z>})$ より，規則

$$f_{\{2\}}(\text{<z>}) \mathrel{\hat{=}} \text{<z>}$$

を生成する．次，規則 $f(\text{<s>}\mathbin{\boldsymbol{.}}x) \mathrel{\hat{=}} \mathbf{let}\,(s,t) \mathrel{\hat{=}} f(x)\,\mathbf{in}\;(t,\text{<s>}\mathbin{\boldsymbol{.}}s)$ より，規則

$$f_{\{2\}}(\text{<s>}\mathbin{\boldsymbol{.}}x) \mathrel{\hat{=}} \mathbf{let}\,s \mathrel{\hat{=}} f_{\{1\}}(x)\,\mathbf{in}\;\text{<s>}\mathbin{\boldsymbol{.}}s$$

を生成する．ここで，関数呼出の $f(x)$ の返り値の内，$f_{\{2\}}$ の値の計算には第一要素しか使用されていないため，f の返り値の第一要素のみを計算する関数 $f_{\{1\}}(x)$ が **let** 束縛にて呼ばれている．ここで，$f_{\{1\}}$ を定義する規則は全て生成されているため，返り値の使用に基づく関数の特化は停止する．最終的に得られる関数の定義は以下となる．

$$\begin{array}{ll} \mathit{tupledHalf}(x) & \hat{=}\ \mathbf{let}\, s \mathrel{\hat{=}} f_{\{1\}}(x)\, \mathbf{in}\ s \\ f_{\{1\}}(\texttt{<z>}) & \hat{=}\ \texttt{<z>} \\ f_{\{1\}}(\texttt{<s>}\,\texttt{.}\, x) & \hat{=}\ \mathbf{let}\, t \mathrel{\hat{=}} f_{\{2\}}(x)\, \mathbf{in}\ t \\ f_{\{2\}}(\texttt{<z>}) & \hat{=}\ \texttt{<z>} \\ f_{\{2\}}(\texttt{<s>}\,\texttt{.}\, x) & \hat{=}\ \mathbf{let}\, s \mathrel{\hat{=}} f_{\{1\}}(x)\, \mathbf{in}\ \texttt{<s>}\,\texttt{.}\, s \end{array}$$

上の関数に対し ALG-C$^+$ は入力の偶奇を返す補関数は導出できないものの，入力 x に対し $\lceil x/2 \rceil$ を返す補関数を導出できるようになる．後の 8.3.3 節に，上の関数に対し入力の偶奇を返す関数の導出法の議論がある．

アルゴリズム 8.6 (返り値の使用に基づく関数の特化).

入力： プログラム $\mathcal{P} = (G, Q, R)$ および順方向変換を定める関数記号 $\mathit{entry} \in Q$（n 個組を返す）

出力： プログラム P'

手続き：

1. $\mathit{Req} := \{(r, \{1, \ldots, n\}) \mid r \in R, r$ は entry の規則 $\}$.
2. $R' := \emptyset$，$\mathit{Done} := \emptyset$.
3. 以下を $\mathit{Req} \setminus \mathit{Done}$ が空になるまで繰り返す.
   (a) $(r, I) \in \mathit{Req} \setminus \mathit{Done}$，$\mathit{Done} := \mathit{Done} \cup \{(r, I)\}$．ただし，
   $$r = f(\overrightarrow{p}) \mathrel{\hat{=}} \mathbf{let}\, \{(\overrightarrow{y_i}) \mathrel{\hat{=}} f_i(\overrightarrow{x_i})\}_{i \in \{1,\ldots,n\}} \ \mathbf{in}\ \overrightarrow{q}$$
   である．
   (b) 以下のようにして関数 f_I の新しい規則 r' を定める．
   $$r' = f_I(\overrightarrow{p}) \mathrel{\hat{=}} \mathbf{let}\, \{(\overrightarrow{y_i}') \mathrel{\hat{=}} f_{iI_i}(\overrightarrow{x_i})\}_{i \in \{1,\ldots,n\}} \ \mathbf{in}\ \overrightarrow{q}'$$
   ただし，
   $$\overrightarrow{q}' = q_{m_1} \ldots q_{m_l} \text{ where } (m_1, \ldots, m_l) = \text{sort } I$$
   である．また，各 $\overrightarrow{y_i}'$ および各 I_i は，$\overrightarrow{y_i}$ を
   $$(y_{i1}, \ldots, y_{ij}, \ldots y_{i|\overrightarrow{y_i}|}) = \overrightarrow{y_i}$$
   と書くとき
   $$I_i = \{j \mid y_{ij} \in \mathsf{vars}(\overrightarrow{q'})\}$$

および

$$\overrightarrow{y_i}' = y_{ik_1} \dots y_{ik_l}$$
$$\text{where } (k_1, \dots, k_l) = \mathrm{sort}(I_i)$$

で定める．ここで，sort は自然数の集合を昇順に整列し自然数の列を得る関数である．

(c) $R' := R' \cup \{r'\}$.

(d) $\mathit{Req} := \mathit{Req} \cup \bigcup_{i\in\{1,\dots,n\}} \{(r'', I_i) \mid r''$ は f_i の規則$\}$.

4. 規則集合 R' からプログラム P' を構成する． □

上のアルゴリズムは停止する．なぜなら，返り値の数が n 個で関数 f に対し，特化された先の関数の候補は，$\{f_I \mid I \subset \{1,\dots,n\}\}$ と有限であり，上のアルゴリズムが同じ関数の規則を二度以上生成することがないためである．これは，同時に返り値の数の最大値を n とすると特化後プログラムサイズの増加の上界が 2^n であることを意味する．しかし，このアルゴリズムがプログラムを変更するのは，一部の返り値が使用されていない関数呼出がある場合のみである．これまで本論文に出現したプログラムは，*tupledHalf* を除いて，関数呼出の返り値は全て使用されているため，このアルゴリズムを適用する必要はない．

なお，この変換を行っても関数の値域の近似結果は変わることがない．なぜなら，*half* の返り値を記述するための正規生垣文法では，以下のように，そもそも多返値関数の返り値は別個に扱われるためである．

$$\begin{array}{ll} T_{\mathit{tupledHalf}} \to \pi_1 T_f & \\ \pi_1 T_f \to \texttt{<z>} & \pi_2 T_f \to \texttt{<z>} \\ \pi_1 T_f \to \pi_2 T_f & \pi_2 T_f \to \texttt{<s>}\pi_1 T_f \end{array}$$

それに対し，単射性判定はより正確になる．これは $\mathsf{lostvars}(r)$ の扱いによる．たとえば，以下の関数 g は，単射だと判定されない．

$$\begin{array}{ll} g(x) \mathrel{\hat=} \textbf{let}\,(s,t) \mathrel{\hat=} \mathit{dupNat}(x)\,\textbf{in}\ s & \\ \mathit{dupNat}(\texttt{<z>}) & \mathrel{\hat=} (\texttt{<z>}, \texttt{<z>}) \\ \mathit{dupNat}(\texttt{<s>}\,.\,x) & \mathrel{\hat=} \textbf{let}\,(s,t) \mathrel{\hat=} \mathit{dupNat}(x)\,\textbf{in}\ (\texttt{<s>}\,.\,s, \texttt{<s>}\,.\,t) \end{array}$$

なぜなら，g の右辺において変数 t が使用されていない，つまり $\mathsf{lostvars}(g(x) \mathrel{\hat=} \textbf{let}\,(s,t) \mathrel{\hat=} \mathit{dupNat}(x)\,\textbf{in}\ s) = \{t\} \neq \emptyset$ であるためである．ところが，返り値の使用に基づく特化を適用すると，g は以下となる．

$$\begin{array}{ll} g_{\{1\}}(x) \mathrel{\hat=} \textbf{let}\,s \mathrel{\hat=} \mathit{dupNat}_{\{1\}}(x)\,\textbf{in}\ s & \\ \mathit{dupNat}_{\{1\}}(\texttt{<z>}) & \mathrel{\hat=} \texttt{<z>} \\ \mathit{dupNat}_{\{1\}}(\texttt{<s>}\,.\,x) & \mathrel{\hat=} \textbf{let}\,s \mathrel{\hat=} \mathit{dupNat}_{\{1\}}(x)\,\textbf{in}\ \texttt{<s>}\,.\,s \end{array}$$

この関数 $g_{\{1\}}$ の定義は未使用の変数を含まない．そのため関数 $g_{\{1\}}$ は本章の単射性判定により単射であると判定される．

8.3.2 近似前の値域を利用した重なりの判定

本節冒頭では，同期機構付き文脈自由生垣文法により，式の値域を厳密に求めた．ここでは，厳密な値域を利用し値域同士の重なりを判定について述べる．

前述のように一般には厳密な型の上での重なり判定は決定不能である．そのため，何らかの手法により重なりを探索し，もし重なりが見つかるか重なりが存在しないことがわかれば，停止し，そうでなければ，適当に探索を打ち切ることを考える．

例として，例 8.9 の *halfeven* を考える．関数 *halfeven* の単射性が判定されなかったのは，関数 g が単射であると判定されなかったためである．よって，g の各規則の右辺式の値域の厳密な推論結果を考える．まず，f と g の値域を厳密に記述する文法は以下である．

$$
\begin{aligned}
T_f &\to (\texttt{<z>}, \texttt{<true>}) \\
T_f &\to (\texttt{<s>}\,\mathbf{.}\,\pi_1 T_g^{(1)}, \pi_2 T_g^{(1)}) \\
T_g &\to (\texttt{<z>}, \texttt{<false>}) \\
T_g &\to (\pi_1 T_f^{(1)}, \pi_2 T_f^{(1)})
\end{aligned}
$$

ここで，T_g の一つ目と二つ目の生成規則が，それぞれ g の一つ目と二つ目の定義規則の右辺式の値域に対応している．もし，これらの値域に重なりがあるならば，規則により非終端記号を展開していった場合に <z> と $\pi_1 T_f^{(1)}$ が，<false> と $\pi_2 T_f^{(1)}$ が等しくなるはずである．これらをまとめて，

$$
\texttt{<z>} \stackrel{?}{=} \pi_1 T_f^{(1)} \qquad \texttt{<false>} \stackrel{?}{=} \pi_2 T_f^{(1)}
$$

表す．ここで，各 $\stackrel{?}{=}$ の右辺には非終端記号が含まれているので，等式が本当に成立するかどうかを調べるのにこれらの非終端記号を展開する．ただし，上添字である同期識別子が同じ非終端記号は同時に展開しなければならない．ここで，T_f は二つの生成規則を持っているので展開すると，

$$
\texttt{<z>} \stackrel{?}{=} \texttt{<z>} \qquad \texttt{<false>} \stackrel{?}{=} \texttt{<true>}
$$

と

$$
\texttt{<z>} \stackrel{?}{=} \texttt{<s>}\,\mathbf{.}\,\pi_1 T_g^{(1)} \qquad \texttt{<false>} \stackrel{?}{=} \pi_2 T_g^{(1)}
$$

となる．ところが，どちらの等式も成立しえないことがわかる．なぜなら，上の等式においては $\texttt{<false>} \neq \texttt{<true>}$ であるし，$\pi_1 T_g^{(1)}$ が何に展開されたとしても $\texttt{<z>} \neq \texttt{<s>}\,\mathbf{.}\,\pi_1 T_g^{(1)}$ であるためである．よって，g の各規則の右辺式の値域に重なりがないことがわかり，g の単射性が結論づけられる．

重なりがある例として，以下の関数 *add* を考える．

$$
\begin{aligned}
&\mathbf{data}\ \mathit{Nat} \mathrel{\hat{=}} \texttt{<s>}\,\mathbf{.}\,\mathit{Nat} \mid \texttt{<z>} \\
&\mathit{add}(\texttt{z}, y :: \mathit{Nat}) \quad\ \mathrel{\hat{=}} y \\
&\mathit{add}(\texttt{s}\,\mathbf{.}\,x, y :: \mathit{Nat}) \mathrel{\hat{=}} \mathbf{let}\, t \mathrel{\hat{=}} \mathit{add}(x, y)\, \mathbf{in}\ \texttt{<s>}\,\mathbf{.}\,t
\end{aligned}
$$

関数 add の第一規則と第二規則の右辺式の値域には重なりがあるので，これを前述の手法で見つけることを考える．まず，add の値域を厳密に記述する文法は以下である．

$$\begin{array}{lcl} Nat & \to & \texttt{<z>} \\ Nat & \to & \texttt{<s>} \cdot Nat \\ T_{add} & \to & Nat \\ T_{add} & \to & \texttt{<s>} \cdot T_{add} \end{array}$$

先程の議論と同様に，

$$Nat \stackrel{?}{=} \texttt{<s>} \cdot T_{add}$$

を考える．このとき，上式の展開の仕方は，左辺を展開する場合と右辺を展開する場合がそれぞれ二通りずつあわせて四通りある．ここでは，とりあえず左辺を展開するとする．すると，

$$\texttt{<z>} \stackrel{?}{=} \texttt{<s>} \cdot T_{add}$$

と

$$\texttt{<s>} \cdot Nat \stackrel{?}{=} \texttt{<s>} \cdot T_{add}$$

が得られる．前者の等式は明らかに満たされないので，後者の等式について考える．ここで，等式が成立するかどうかに先頭の <s> の部分は関係ないので取り除くと，

$$Nat \stackrel{?}{=} T_{add}$$

を得る．このときも，左辺を展開する場合と右辺を展開する場合それぞれ二通りずつあわせて四通りある．ここで，とりあえず左辺を展開したとする．すると

$$\texttt{<z>} \stackrel{?}{=} T_{add}$$

と

$$\texttt{<s>} \cdot Nat \stackrel{?}{=} T_{add}$$

が得られる．このうち，前者について考える．非終端記号 T_{add} を展開することにより，

$$\texttt{<z>} \stackrel{?}{=} Nat$$

と

$$\texttt{<z>} \stackrel{?}{=} \texttt{<s>} \cdot T_{add}$$

が得られる．このうち，前者について考え，非終端記号 Nat を展開すると，

$$\texttt{<z>} \stackrel{?}{=} \texttt{<z>}$$

と

$$\texttt{<z>} \stackrel{?}{=} \texttt{<s>} \cdot Nat$$

が得られる．前者の等式は成り立つため，$\texttt{<z>} \stackrel{?}{=} Nat$，$\texttt{<z>} \stackrel{?}{=} T_{add}$，$Nat \stackrel{?}{=} T_{add}$，$\texttt{<s>.}Nat \stackrel{?}{=} \texttt{<s>.}T_{add}$ および $Nat \stackrel{?}{=} \texttt{<s>.}T_{add}$ も成立させることができる．よって，我々は add の二つの規則の右辺式の値域に重なりがあったと結論づけられる．これは，これまでの操作が

$$Nat \to \texttt{<s>.}Nat \to \texttt{<s>.<z>}$$

という左辺の展開と

$$\texttt{<s>.}T_{add} \to \texttt{<s>.}Nat \to \texttt{<s>.<z>}$$

という右辺の展開に対応づけられるためである．

文脈自由多生垣文法による式の値域の厳密な記述に基づき，重なりの判定を行っても図 8.1 の単射性判定および補関数導出アルゴリズム ALG-C$^+$ の正しさには影響はない．なぜなら，これらの正しさは重なりの判定の完全性つまり，値域に重なりがあった場合には重なり判定は重なり必ず見つけ出すことにのみ依存しているためである．また，8.2.2 節での逆方向変換の導出も適用できる．逆方向変換の導出においては，近似された式の値域の性質を利用しているものの，それは水平方向の重なりを調べることにおいてのみであるためである．よって，値域の重なりをより正確に解析しても問題はない．しかし，得られる逆方向変換は，入力に対しマッチするパターンが複数個ある場合が起こりうるため，決定的なプログラムであると限らない．ただし，パターンを一つに定めると，パターンマッチにより生じる変数の束縛は一つに定まる．

And/Or 探索による重なりの判定

これまでに述べた探索の操作を，節点が無限個のグラフ上の And/Or 探索により定式化する．値域の重なりを調べたいプログラムの厳密な値域を記述する文脈自由多生垣文法を G とおく．

And/Or 探索をするために，And 節点と Or 節点，それぞれの節点からの枝そして根について定めておく．

まず，節点は等式候補の集合 E とラベル $l \in \{\textsc{And}, \textsc{Or}\}$ の組である．各等式候補（$\in E$）は

$$\mathcal{C}[\pi_{j_1}T_1^{(i_1)}, \ldots, \pi_{j_n}T_n^{(i_n)}] \stackrel{?}{=} \mathcal{C}'[\pi_{j'_1}T_1^{(i'_1)}, \ldots, \pi_{j'_{n'}}T_{n'}^{(i'_{n'})}]$$

もしくは

$$\textsc{Fail}$$

という形式をしている．なお，同期識別子の同値性を保存する名前換えにより等しくなる二つの節点は同一であるとする．たとえば，節点 $(\{\texttt{<z>} \stackrel{?}{=} \pi_1 T_g^{(1)}, \texttt{<false>} \stackrel{?}{=} \pi_2 T_g^{(1)}\}, \textsc{And})$ と $(\{\texttt{<z>} \stackrel{?}{=} \pi_1 T_g^{(2)}, \texttt{<false>} \stackrel{?}{=} \pi_2 T_g^{(2)}\}, \textsc{And})$ とは同一であると見なす．それに対し，節点 $(\{\texttt{<z>} \stackrel{?}{=} \pi_1 T_g^{(1)}, \texttt{<false>} \stackrel{?}{=} \pi_2 T_g^{(1)}\}, \textsc{And})$ と $(\{\texttt{<z>} \stackrel{?}{=} \pi_1 T_g^{(1)}, \texttt{<false>} \stackrel{?}{=} \pi_2 T_g^{(2)}\}, \textsc{And})$ とは区別される．

次に，Or 節点から And 節点への枝について述べる．節点 $(\{s_1 \overset{?}{=} t_1, \ldots, s_n \overset{?}{=} t_n\}, \mathrm{Or})$ は，節点の集合

$$\{(\mathsf{trunc}(\{s_1 \overset{?}{=} t'_1, \ldots, s_n \overset{?}{=} t'_n\}), \textsc{And}) \mid (t_1, \ldots, t_n) \to_G (t'_1, \ldots, t'_n), \exists i.\ t_i = \pi_j T^{(k)}\}$$
$$\cup \{(\mathsf{trunc}(\{s'_1 \overset{?}{=} t_1, \ldots, s'_n \overset{?}{=} t_n\}), \textsc{And}) \mid (s_1, \ldots, s_n) \to_G (s'_1, \ldots, s'_n), \exists i.\ s_i = \pi_j T^{(k)}\}$$

のそれぞれの節点で，等式候補の集合の中に $\textsc{Fail}$ を含まないもの，に対して枝を持つ．ただし，trunc は共通の先頭を取り除く操作であり，

$$\begin{aligned}
&\mathsf{trunc}(\{s_i \overset{?}{=} t_i\}_{i\in\{1,\ldots,n\}}) = \textstyle\bigcup_{i\in\{1,\ldots,n\}} tr(s_i \overset{?}{=} t_i)\\
&tr(\sigma(s_1)\,\textbf{.}\,s_2 \overset{?}{=} \sigma(t_1)\,\textbf{.}\,t_2) = tr(s_1 \overset{?}{=} t_1) \cup tr(s_2 \overset{?}{=} t_2)\\
&tr(\sigma'(s_1)\,\textbf{.}\,s_2 \overset{?}{=} \sigma(t_1)\,\textbf{.}\,t_2) = \textsc{Fail}\\
&tr(\sigma(s_1)\,\textbf{.}\,s_2 \overset{?}{=} \varepsilon) = \textsc{Fail}\\
&tr(\varepsilon \overset{?}{=} \sigma(t_1)\,\textbf{.}\,t_2) = \textsc{Fail}\\
&tr(x \overset{?}{=} \pi_i T^{(j)}) = \{x \overset{?}{=} \pi_i T^{(j)}\}\\
&tr(\pi_i T^{(j)} \overset{?}{=} x) = \{\pi_i T^{(j)} \overset{?}{=} x\}
\end{aligned}$$

である．上で，$\exists i.\ t_i = \pi_j T^{(k)}$ や $\exists i.\ s_i = \pi_j T^{(k)}$ を要求しているのは，等式候補の右辺もしくは左辺のみを展開しつづけることを防ぐためである．

等式候補の集合は，等式が成立するかどうかを独立に調べられる集合に分解できる場合がある．たとえば，$\{T_{add} \overset{?}{=} \texttt{<z>}, Nat \overset{?}{=} \texttt{<s>}\,\textbf{.}\,\texttt{<s>}\,\textbf{.}\,T_{add}\}$ の二つの等式候補は，独立に等式が成立するか調べられる．それに対し，$\{\texttt{<z>} \overset{?}{=} \pi_1 T_g^{(1)}, \texttt{<false>} \overset{?}{=} \pi_2 T_g^{(1)}\}$ の二つの等式候補は，独立に等式が成立するかどうかは調べられない．これは，同期識別子により，それぞれの要素の右辺の非終端記号は，同じ生成規則で同時に展開されなければならないためである．And 節点から Or 節点への枝は，この独立に調べられる集合への分解に対応する．まず，分解のための同値関係 $\overset{\mathrm{share}}{\equiv}$ を，次の関係 S

$$\begin{aligned}
&(s \overset{?}{=} t)\ S\ (s' \overset{?}{=} t')\\
&\quad\Leftrightarrow \left(\begin{aligned}&\exists T^{(i)}, \mathcal{C}, \mathcal{C}'.\\ &(s = \mathcal{C}[\pi_T^{(i)}] \land s' = \mathcal{C}'[\pi_T^{(i)}]) \lor (t = \mathcal{C}[\pi_T^{(i)}] \land t' = \mathcal{C}'[\pi_T^{(i)}])\end{aligned}\right)
\end{aligned}$$

により，$\overset{\mathrm{share}}{\equiv} = S^*$ と定める．ここで R^* は関係 R の反射的推移的閉包である．直観的には，$(s \overset{?}{=} t)\ S\ (s' \overset{?}{=} t')$ は，s と s' もしくは t と t' に同時に展開しなければならない非終端記号を含むことを意味する．これにより，And 節点の持つ枝を次のように定める．節点 $(\{s_1 \overset{?}{=} t_1, \ldots, s_n \overset{?}{=} t_n\}, \mathrm{And})$ は，節点の集合

$$\{([s_i \overset{?}{=} t_i]_{\overset{\mathrm{share}}{\equiv}}, \textsc{Or}) \mid i \in \{1, \ldots, n\}\}$$

の各節点へと枝を持つ．

根の与え方について述べる．右辺式の重なりを調べたい規則を r_1 および r_2 とすると，それぞれ，対応する生成規則 r'_1, r'_2 を同期機構付き文脈自由生垣文法 G の中に持つ．

$$r'_1 = T_f \to (s_1, \ldots, s_n)$$
$$r'_2 = T_f \to (t_1, \ldots, t_n)$$

このとき，根 n_0 を，$E = \mathsf{trunc}(\{s_i \stackrel{?}{=} t_i\} \mid i \in \{1, \ldots, n\})$ として，

$$n_0 = \begin{cases} (E, \textsc{And}) & \text{if } \textsc{Fail} \notin E \\ (\emptyset, \textsc{Or}) & \text{otherwise} \end{cases}$$

とする．

定義 8.4 (And/Or 探索)**.** 節点から真理値への写像を ϕ を次で与える．

- $\phi((E, \textsc{Or}))$ は，$(E, \textsc{Or})$ の子 n で $\phi(n) = \mathsf{True}$ なものが存在すれば $\phi((E, \textsc{Or})) = \mathsf{True}$, $(E, \textsc{Or})$ の全ての子 n が $\phi(n) = \mathsf{False}$ ならば $\phi((E, \textsc{Or})) = \mathsf{False}$.
- $\phi((E, \textsc{And}))$ は，$(E, \textsc{And})$ の全ての子 n が $\phi(n) = \mathsf{True}$ ならば $\phi((E, \textsc{And})) = \mathsf{True}$, $(E, \textsc{And})$ の子 n で $\phi(n) = \mathsf{False}$ なものが存在すれば $\phi((E, \textsc{And})) = \mathsf{False}$.
- $\phi(n)$ の値は，最小不動点により決定する．

このとき，$\phi(n_0)$ の値が And/Or 探索の結果である．

定義より，節点 $(\emptyset, \textsc{And})$ および $(\emptyset, \textsc{Or})$ は子を持たないため，

$$\phi((\emptyset, \textsc{And})) = \mathsf{True} \qquad \phi((\emptyset, \textsc{Or})) = \mathsf{False}$$

となることに注意する．上の定義において，一般には $\phi(n_0)$ の真偽は決定不能であることに注意する．これは，$\textsc{Vdl}^+$ 上の式の値域の重なりの判定が決定不能であることと等価である．また，$\phi(n)$ の真偽を最小不動点で定義しているのは，我々は有限サイズの生垣が共通の値域に含まれているかどうかを判定しているためである．

例 8.11 (*halfeven*)**.** 前述の *halfeven* の f について，f のそれぞれの規則の右辺式の値域に重なりがあるかないかを判定するのに用いる And/Or 探索グラフを以下に示す．

$$\longrightarrow \boxed{\{\texttt{<z>} \stackrel{?}{=} \pi_1 T_g^{(1)}, \texttt{<false>} \stackrel{?}{=} \pi_2 T_g^{(1)}\}} \longrightarrow \boxed{\{\texttt{<z>} \stackrel{?}{=} \pi_1 T_g^{(1)}, \texttt{<false>} \stackrel{?}{=} \pi_2 T_g^{(1)}\}}$$

ここで，角丸の四角が And 節点，四角が Or 節点である．規則を展開した後の trunc が Fail を返すため，Or 節点は子をもたない，つまり，False である．よって，根も False である．よって，f のそれぞれの規則の右辺式の値域に重なりはない．

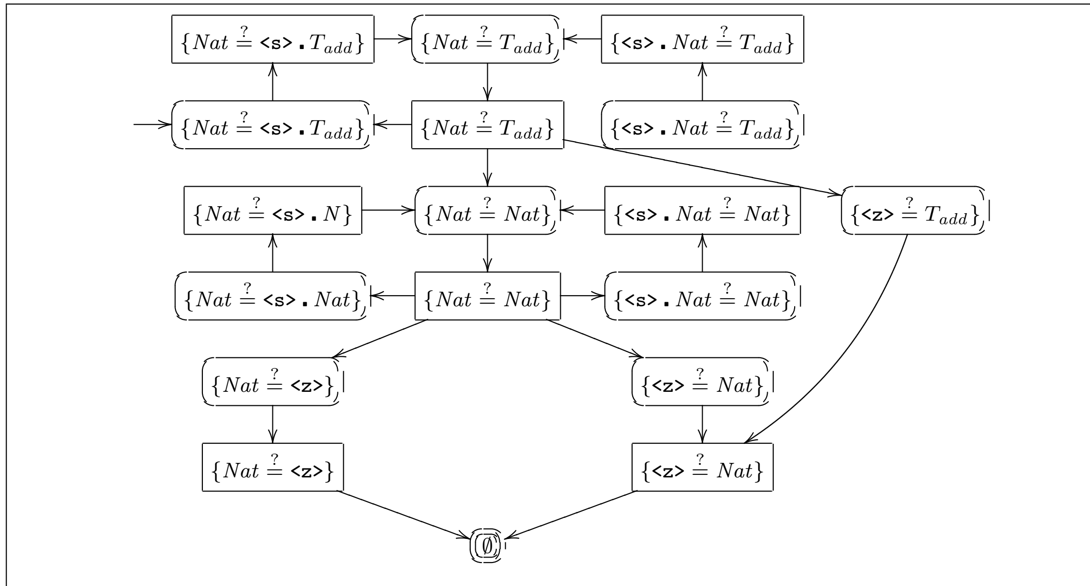

図 8.3. add の第一規則と第二規則の右辺式の重なりを判定するための And/Or 探索グラフ：角丸の四角が And 節点，四角が Or 節点，角丸の二重四角は $(\emptyset, \mathsf{And})$.

例 8.12 (add)**.** 前述の add について，And/Or 探索に用いるグラフを図 8.3 に示す．ここで，角丸の四角が And 節点，四角が Or 節点である．また，角丸の二重四角は $(\emptyset, \mathsf{And})$，つまり True が割り振られる節点である．図において And 節点は一つしか子がないので，この場合，根から True となる節点に経路があるかないかが，第一規則と第二規則の右辺式に重なりがあるかないかに対応する．

なお，And 節点と Or 節点の子の定義より，G が正規生垣文法のときは必ず探索グラフが有限になる．つまり，G が正規生垣文法程度なら And/Or 探索は止って正しい答えを返す．しかし，G が文脈自由多生垣文法のときは探索グラフは無限になり，And/Or 探索は止まるとは限らない．このような場合には，

- 根からの距離
- 等式候補の集合のサイズ
- 等式候補に現れる左辺もしくは右辺の生垣の最大サイズ

に上限を設けることで探索を打ち切ることが考えられる．上のどれも $G = (\Sigma, N, R)$ が正規生垣文法の場合には上限が存在し，それぞれ，$2|N||R|$，1，1 である．

8.3.3　得られた補関数の改良

言語 VDL$^+$ で記述されたプログラムが非単射と判定される連接を含まない場合，ALG-C$^+$ は LENGTH を導入することがない．その場合元の関数と導出された補関数の再帰構造が同

じになるため，簡単に組化を行うことができる．このとき，組化された関数の単射性を確認しながら構成子を取り除くことにより，より小さい補関数が得られる場合がある．導出された補関数から構成子を取り除くことによる単射性への影響は，辺式の値域の重なりがあるかないかが変わるのみである．よって，And/Or 探索による重なりの判定を，補関数の返り値にも適用できるように少し修正するだけで組化された関数の単射性の判定に利用することができる．逆に近似された値域に基づいて単射性の判定を行ったのでは，組化後に新たに構成子を取り除けることはない．これは，そのような構成子は $\textsc{Alg-C}^{+}$ が既に取り除いているためである．

例として，*tupledHalf* を返り値の使用に対する関数の特化後に得られる次の関数を考える．

$$
\begin{aligned}
&\mathit{tupledHalf}_{\{1,2\}}(x) \mathrel{\hat{=}} \mathbf{let}\, s \mathrel{\hat{=}} f_{\{1\}}(x)\, \mathbf{in}\ s\\
&f_{\{1\}}(\mathsf{<z>}) \quad\ \mathrel{\hat{=}} \mathsf{<z>}\\
&f_{\{1\}}(\mathsf{<s>}\,.\,x) \mathrel{\hat{=}} \mathbf{let}\, t \mathrel{\hat{=}} f_{\{2\}}(x)\, \mathbf{in}\ t\\
&f_{\{2\}}(\mathsf{<z>}) \quad\ \mathrel{\hat{=}} \mathsf{<z>}\\
&f_{\{2\}}(\mathsf{<s>}\,.\,x) \mathrel{\hat{=}} \mathbf{let}\, s \mathrel{\hat{=}} f_{\{1\}}(x)\, \mathbf{in}\ \mathsf{<s>}\,.\,s
\end{aligned}
$$

上の関数 $\mathit{tupledHalf}_{\{1,2\}}$ に対し，アルゴリズム $\textsc{Alg-C}^{+}$ は以下の補関数を導出する．

$$
\begin{aligned}
&\mathit{tupledHalf}^{\mathrm{c}}_{\{1,2\}}(x) \mathrel{\hat{=}} \mathbf{let}\, s^{\mathrm{c}} \mathrel{\hat{=}} {f_{\{1\}}}^{\mathrm{c}}(x)\, \mathbf{in}\ s^{\mathrm{c}}\\
&{f_{\{1\}}}^{\mathrm{c}}(\mathsf{<z>}) \quad\ \mathrel{\hat{=}} \mathsf{B}_{21}\\
&{f_{\{1\}}}^{\mathrm{c}}(\mathsf{<s>}\,.\,x) \mathrel{\hat{=}} \mathbf{let}\, t^{\mathrm{c}} \mathrel{\hat{=}} {f_{\{2\}}}^{\mathrm{c}}(x)\, \mathbf{in}\ \mathsf{B}_{31}(t^{\mathrm{c}})\\
&{f_{\{2\}}}^{\mathrm{c}}(\mathsf{<z>}) \quad\ \mathrel{\hat{=}} \mathsf{B}_{41}\\
&{f_{\{2\}}}^{\mathrm{c}}(\mathsf{<s>}\,.\,x) \mathrel{\hat{=}} \mathbf{let}\, s^{\mathrm{c}} \mathrel{\hat{=}} {f_{\{1\}}}^{\mathrm{c}}(x)\, \mathbf{in}\ s^{\mathrm{c}}
\end{aligned}
$$

これらの関数を組化すると以下を得る．

$$
\begin{aligned}
&\langle \mathit{tupledHalf}_{\{1,2\}}, {\mathit{tupledHalf}_{\{1,2\}}}^{\mathrm{c}}\rangle(x) \mathrel{\hat{=}} \mathbf{let}\,(s, s^{\mathrm{c}}) \mathrel{\hat{=}} \langle f_{\{1\}}, {f_{\{1\}}}^{\mathrm{c}}\rangle(x)\, \mathbf{in}\ (s, s^{\mathrm{c}})\\
&\langle f_{\{1\}}, {f_{\{1\}}}^{\mathrm{c}}\rangle(\mathsf{<z>}) \quad\ \mathrel{\hat{=}} (\mathsf{<z>}, \mathsf{B}_{21})\\
&\langle f_{\{1\}}, {f_{\{1\}}}^{\mathrm{c}}\rangle(\mathsf{<s>}\,.\,x) \mathrel{\hat{=}} \mathbf{let}\,(t, t^{\mathrm{c}}) \mathrel{\hat{=}} \langle f_{\{2\}}, {f_{\{2\}}}^{\mathrm{c}}\rangle(x)\, \mathbf{in}\ (t, \mathsf{B}_{31}(t^{\mathrm{c}}))\\
&\langle f_{\{2\}}, {f_{\{2\}}}^{\mathrm{c}}\rangle(\mathsf{<z>}) \quad\ \mathrel{\hat{=}} (\mathsf{<z>}, \mathsf{B}_{41})\\
&\langle f_{\{2\}}, {f_{\{2\}}}^{\mathrm{c}}\rangle(\mathsf{<s>}\,.\,x) \mathrel{\hat{=}} \mathbf{let}\,(s, s^{\mathrm{c}}) \mathrel{\hat{=}} \langle f_{\{1\}}, {f_{\{1\}}}^{\mathrm{c}}\rangle(x)\, \mathbf{in}\ (\mathsf{<s>}\,.\,s, s^{\mathrm{c}})
\end{aligned}
$$

このとき，B_{31} は一引数構成子なのだが，取り除いた後も $\langle \mathit{tupledHalf}_{\{1,2\}}, {\mathit{tupledHalf}_{\{1,2\}}}^{\mathrm{c}}\rangle$ が単射であるかもしれない．もし，取り除いた後も $\langle \mathit{tupledHalf}_{\{1,2\}}, {\mathit{tupledHalf}_{\{1,2\}}}^{\mathrm{c}}\rangle$ が単射であるのなら，取り除いたほうが補関数が小さくなる．実際，取り除くことにより得られる以下の関数は *halfeven* と等価な関数である．

$$
\begin{aligned}
&\langle \mathit{tupledHalf}_{\{1,2\}}, {\mathit{tupledHalf}_{\{1,2\}}}^{\mathrm{c}}\rangle(x) \mathrel{\hat{=}} \mathbf{let}\,(s, s^{\mathrm{c}}) \mathrel{\hat{=}} \langle f_{\{1\}}, {f_{\{1\}}}^{\mathrm{c}}\rangle(x)\, \mathbf{in}\ (s, s^{\mathrm{c}})\\
&\langle f_{\{1\}}, {f_{\{1\}}}^{\mathrm{c}}\rangle(\mathsf{<z>}) \quad\ \mathrel{\hat{=}} (\mathsf{<z>}, \mathsf{B}_{21})\\
&\langle f_{\{1\}}, {f_{\{1\}}}^{\mathrm{c}}\rangle(\mathsf{<s>}\,.\,x) \mathrel{\hat{=}} \mathbf{let}\,(t, t^{\mathrm{c}}) \mathrel{\hat{=}} \langle f_{\{2\}}, {f_{\{2\}}}^{\mathrm{c}}\rangle(x)\, \mathbf{in}\ (t, t^{\mathrm{c}})\\
&\langle f_{\{2\}}, {f_{\{2\}}}^{\mathrm{c}}\rangle(\mathsf{<z>}) \quad\ \mathrel{\hat{=}} (\mathsf{<z>}, \mathsf{B}_{41})\\
&\langle f_{\{2\}}, {f_{\{2\}}}^{\mathrm{c}}\rangle(\mathsf{<s>}\,.\,x) \mathrel{\hat{=}} \mathbf{let}\,(s, s^{\mathrm{c}}) \mathrel{\hat{=}} \langle f_{\{1\}}, {f_{\{1\}}}^{\mathrm{c}}\rangle(x)\, \mathbf{in}\ (\mathsf{<s>}\,.\,s, s^{\mathrm{c}})
\end{aligned}
$$

よって，And/Or 探索により値域の重なり判定を用いることで，上の構成子を取り除いた関数の単射性を判定することができる．

順方向変換を $entry$ であるとし，ALG-C$^+$ により得られる補関数を $entry^{\mathrm{c}}$ とすると，以下のアルゴリズムにより $entry^{\mathrm{c}}$ をより小さなものに変換できる場合がある．

アルゴリズム 8.7 (補関数の改良).

入力：組化された関数 $\langle entry, entry^{\mathrm{c}}\rangle$ を定義するプログラム

出力：組化された関数 $\langle entry, entry^{\mathrm{c}}\rangle$ を定義するプログラム

手続き：

プログラムの全ての規則について以下を行う．

- 規則 r について，もし，ALG-C$^+$ が導入した構成子が全て一引数構成子であり，かつ，それら全ての構成子をはずした規則 r' を作成した場合に r を r' に置き換えても $\langle entry, entry^{\mathrm{c}}\rangle$ の単射性が And/Or 探索による値域の重なり判定により確認できるのなら，r を r' で置き換える．
- そうでないなら，規則 r は維持する．□

上で，$entry$ の単射性を確認するには，$entry$ を定義するプログラムについて，全ての関数の互いに異なる規則の右辺式の値域に重なりがないことを，And/Or 探索による値域の重なり判定により確認すればよい．

第9章　まとめ

9.1　結論

本論文では，我々はプログラムの双方向化，すなわち順方向変換プログラムから双方向変換プログラムを自動構成する手法について議論した．本論文で提案した双方向化手法は，補関数 [BS81] の導出に基づく．これにより，本論文の手法の導出する双方向変換は常に振る舞いがよいことが保証される．我々は，特定のプログラム言語で記述された順方向変換プログラムに対し，補関数プログラムを自動的に導出する手法を与え，その後組化や逆関数の導出など既存のプログラム変換を適用することにより，逆方向変換プログラムを導出し，双方向変換プログラムを構成した．本論文の言語 VDL や VDL$^+$ は制限されているため，それらの言語で記述されたプログラムに対し，効果的な双方向変換を得るためのプログラム解析を効果的に行うことができる．しかし，いくつかの本論文の例に示すように，制限されていながらも言語 VDL や VDL$^+$ では木構造データの上の基本的な変換を記述できる．さらに，我々はいくつかの例を通して本手法の有効性を確認した．

9.2　今後の課題

我々の今後の課題を以下に示す．

9.2.1　補関数導出手法

多様な補関数

本論文の提案する補関数導出手法は，元の関数と同じ再帰構造を持つ補関数を導出する．しかし，元の関数と同じ再帰構造を持つ補関数が全て本論文の補関数導出手法により導出できるわけではない．第 3 章において，二つの自然数を加算する関数 $add :: (\mathbb{N} \times \mathbb{N}) \to \mathbb{N}$ の補関数の一つが二つの自然数の差を求める関数 $sub :: (\mathbb{N} \times \mathbb{N}) \to \mathbb{Z}$ であることを示した．

しかし，どのようにすれば add から sub を導出できるかは明らかでない．たとえば，次のように定義された add を考える．

$$
\begin{array}{ll}
add(\mathsf{Z}, y) & \hat{=}\ y \\
add(\mathsf{S}(x), \mathsf{Z}) & \hat{=}\ \mathsf{S}(x) \\
add(\mathsf{S}(x), \mathsf{S}(y)) & \hat{=}\ \mathsf{S}(\mathsf{S}(add(x)))
\end{array}
$$

以下の関数 sub' は，add の補関数であり sub と補関数として等価な関数である．

$$\begin{aligned} &sub'(\mathsf{Z}, y) &&\hat{=} \mathsf{B}_1(y) \\ &sub'(\mathsf{S}(x), \mathsf{Z}) &&\hat{=} \mathsf{B}_2(x) \\ &sub'(\mathsf{S}(x), \mathsf{S}(y)) &&\hat{=} sub'(x) \end{aligned}$$

関数 sub' は，add と同じ再帰構造を持っている．しかし，我々は現在，add から上の f と同等な補関数を得ることができていない．また，8.3.2 節の And/Or 探索を用いても $\langle add, sub' \rangle$ の単射性の確認に失敗する．

どのように $add :: (\mathbb{N} \times \mathbb{N}) \to \mathbb{N}$ を記述し，どのように単射性を解析すれば $sub :: (\mathbb{N} \times \mathbb{N}) \to \mathbb{Z}$ と同等な関数が得られるかがわかれば，より効果的かつ直観的な逆方向変換導出が実現できることが期待される．

連接の補関数

第 8 章においては，連接演算を含む関数の補関数を作成するのに，連接演算子の第一引数の生垣の長さの情報を用いた．しかし，連接演算子の引数の型によっては，第一引数の生垣の長さを用いる方法よりも，直観的な逆方向変換を達成する補関数を構成できる場合がある．

たとえば，以下の関数 f における非単射な連接 $x \centerdot y$ を考える．

$$\begin{aligned} &\mathbf{data}\ T \mathrel{\hat{=}} (\texttt{<a>} \centerdot \texttt{<b>}^*)^* \\ &f(x :: T, y :: T) \mathrel{\hat{=}} x \centerdot y \end{aligned}$$

この非単射な連接に対して，第一引数の生垣の長さを補う補関数

$$f^{\mathrm{c}}(x :: T, y :: T) \mathrel{\hat{=}} \textsc{length}(x)$$

から定まる逆方向変換は以下の関数と等価でものである．

$$\begin{aligned} &f_{\mathrm{B}}((x :: T, y :: T), v) \\ &\quad = \mathbf{let}\ (v_1 :: T, v_2 :: T) = \textsc{splitAt}(\textsc{length}(x), v)\ \mathbf{in}\ (v_1, v_2) \end{aligned}$$

逆方向変換 f_{B} は，ソースの組の第一要素の生垣の長さを変更することができない．そのため，たとえば以下の更新の反映は失敗する．

$$f_{\mathrm{B}}((\texttt{<a>} \centerdot \texttt{<b>} \centerdot \texttt{<b>}, \texttt{<a>}), \texttt{<a>}) = \bot$$

ところが，以下の関数 $f^{\mathrm{c}\prime}$ も f の補関数となる．

$$\begin{aligned} &f^{\mathrm{c}\prime}(x :: T, y :: T) &&\hat{=} lengthA(x) \\ &lengthA(\varepsilon) &&\hat{=} 0 \\ &lengthA(\texttt{<a>} \centerdot r) &&\hat{=} 1 + lengthA(r) \\ &lengthA(\texttt{<b>} \centerdot r) &&\hat{=} lengthA(r) \end{aligned}$$

補関数 $f^{\mathrm{c}\prime}$ を利用して得た逆方向変換は以下の関数と等価なものである．

$$
\begin{aligned}
& f'_{\mathrm{B}}((x :: T, y :: T), v) \\
& \qquad \mathrel{\hat{=}} \mathbf{let}\ (v_1 :: T, v_2 :: T) \mathrel{\hat{=}} \mathit{splitByA}(\mathit{lengthA}(x), v)\ \mathbf{in}\ v_1 \mathbin{\centerdot} v_2 \\
& \mathit{splitByA}(\varepsilon, 0) && \mathrel{\hat{=}} (\varepsilon, \varepsilon) \\
& \mathit{splitByA}(\texttt{<a>} \mathbin{\centerdot} r, 0) && \mathrel{\hat{=}} (\varepsilon, \texttt{<a>} \mathbin{\centerdot} r) \\
& \mathit{splitByA}(\texttt{<b>} \mathbin{\centerdot} r, 0) && \mathrel{\hat{=}} \mathbf{let}\ (s, t) \mathrel{\hat{=}} \mathit{splitByA}(r, 0)\ \mathbf{in}\ (\texttt{<b>} \mathbin{\centerdot} s, t) \\
& \mathit{splitByA}(\texttt{<a>} \mathbin{\centerdot} r, n+1) && \mathrel{\hat{=}} \mathbf{let}\ (s, t) \mathrel{\hat{=}} \mathit{splitByA}(r, n)\ \mathbf{in}\ (\texttt{<a>} \mathbin{\centerdot} s, t) \\
& \mathit{splitByA}(\texttt{<b>} \mathbin{\centerdot} r, n+1) && \mathrel{\hat{=}} \mathbf{let}\ (s, t) \mathrel{\hat{=}} \mathit{splitByA}(r, n+1)\ \mathbf{in}\ (\texttt{<b>} \mathbin{\centerdot} s, t)
\end{aligned}
$$

逆方向変換 f'_{B} は，第一引数中の <a> の数を変えることを許さないが，<b> の自由な挿入および削除を許す．そのため，たとえば以下の更新の反映は成功する．

$$f'_{\mathrm{B}}((\texttt{<a>} \mathbin{\centerdot} \texttt{<b>} \mathbin{\centerdot} \texttt{<b>}, \texttt{<a>}), \texttt{<a>}) = (\texttt{<a>}, \varepsilon)$$

この $f^{\mathrm{c}\prime}$ は f^{c} とは縮約順序（第 3 章，定義 3.8）では比較不能である．なぜなら

$$
\begin{aligned}
& \mathit{lengthA}(\texttt{<a>} \mathbin{\centerdot} \texttt{<b>}) = \mathit{lengthA}(\texttt{<a>}) && \textsc{length}(\texttt{<a>} \mathbin{\centerdot} \texttt{<b>}) \neq \textsc{length}(\texttt{<a>}) \\
& \mathit{lengthA}(\texttt{<a>} \mathbin{\centerdot} \texttt{<b>}) \neq \mathit{lengthA}(\texttt{<a>} \mathbin{\centerdot} \texttt{<a>}) && \textsc{length}(\texttt{<a>} \mathbin{\centerdot} \texttt{<b>}) = \textsc{length}(\texttt{<a>} \mathbin{\centerdot} \texttt{<a>})
\end{aligned}
$$

より，

$$
\begin{aligned}
& \exists s_1, s_2, s'_1, s'_2 \in [\![T]\!].\ f^{\mathrm{c}}(s_1, s_2) = f^{\mathrm{c}}(s'_1, s'_2) \wedge f^{\mathrm{c}\prime}(s_1, s_2) \neq f^{\mathrm{c}\prime}(s'_1, s'_2) \\
& \exists s_1, s_2, s'_1, s'_2 \in [\![T]\!].\ f^{\mathrm{c}}(s_1, s_2) \neq f^{\mathrm{c}}(s'_1, s'_2) \wedge f^{\mathrm{c}\prime}(s_1, s_2) = f^{\mathrm{c}\prime}(s'_1, s'_2)
\end{aligned}
$$

となるためである．

もう少し具体的な例として以下の関数 $\mathit{c2xf}$ を考える．

$$
\begin{aligned}
& \mathbf{data}\ S \mathrel{\hat{=}} (\texttt{<section>}(\texttt{<title>}(\mathit{String}) \mathbin{\centerdot} P))^{*} \\
& \mathbf{data}\ P \mathrel{\hat{=}} (\texttt{<p>}(\mathit{String}))^{*}
\end{aligned}
$$

$$
\begin{aligned}
& \mathit{c2xf}(\varepsilon) \mathrel{\hat{=}} \varepsilon \\
& \mathit{c2xf}(\texttt{<chapter>}(\texttt{<title>}(t :: \mathit{String}) \mathbin{\centerdot} p :: P \mathbin{\centerdot} s :: S) \mathbin{\centerdot} r :: C) \\
& \quad \mathrel{\hat{=}} \texttt{<h1>}(t) \mathbin{\centerdot} p \mathbin{\centerdot} \mathit{s2xf}(s) \mathbin{\centerdot} \mathit{c2xf}(r) \\
& \mathit{s2xf}(\varepsilon) \mathrel{\hat{=}} \varepsilon \\
& \mathit{s2xf}(\texttt{<section>}(\texttt{<title>}(t :: \mathit{String}) \mathbin{\centerdot} p :: P) \mathbin{\centerdot} r) \\
& \quad \mathrel{\hat{=}} \texttt{<h1>}(t) \mathbin{\centerdot} p \mathbin{\centerdot} \mathit{s2xf}(r)
\end{aligned}
$$

この関数は第 1 章，第 6 章に登場した $\mathit{c2x}$ とは少し異なり，節題を <h2> 要素ではなく <h1> 要素へと変換する．ここで，第二規則の $\mathit{s2xf}(s) \mathbin{\centerdot} \mathit{c2xf}(r)$ の部分に現れる連接が単射ではない．なぜならば，$\mathit{s2xf}(s)$ の値域と $\mathit{c2xf}(r)$ の値域はともに以下の集合 U であるためである．

$$U = (\texttt{<h1>}(\mathit{String}) \mathbin{\centerdot} (\texttt{<p>}(\mathit{String}))^{*})^{*}$$

上の型 U は先程の型 T と同様の形式をしている．もし，非単射な連接に対し第一引数の生垣の長さを保存することで補関数を作成した場合，その補関数より得られる逆方向変換は以

下の関数と等価な関数になる．

$$
\begin{aligned}
&\textbf{data}\ S \mathrel{\hat{=}} (\texttt{<section>}(\texttt{<title>}(\mathit{String}) \cdot P))^{*}\\
&\textbf{data}\ P \mathrel{\hat{=}} (\texttt{<p>}(\mathit{String}))^{*}
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{c2xf}_{\mathrm{B}}(\varepsilon, \varepsilon) \mathrel{\hat{=}} \varepsilon\\
&\mathit{c2xf}_{\mathrm{B}}(\texttt{<chapter>}(\texttt{<title>}(t :: \mathit{String}) \cdot p :: P \cdot s :: S) \cdot r :: C, \texttt{<h1>}(t') \cdot p' \cdot v)\\
&\quad \mathrel{\hat{=}} \textbf{let}\ (v_1, v_2) \mathrel{\hat{=}} \textsc{splitAt}(\textsc{length}(\mathit{s2xf}(s), v))\\
&\qquad \textbf{in}\ \texttt{<chapter>}(\texttt{<title>}(t') \cdot p' \cdot \mathit{s2xf}_{\mathrm{B}}(s, v_1)) \cdot \mathit{c2xf}_{\mathrm{B}}(r, v_2)\\
&\mathit{s2xf}_{\mathrm{B}}(_, \varepsilon) \mathrel{\hat{=}} \varepsilon\\
&\mathit{s2xf}_{\mathrm{B}}(_, \texttt{<h1>}(t) \cdot p \cdot v)\\
&\quad \mathrel{\hat{=}} \texttt{<section>}(\texttt{<title>}(t :: \mathit{String}) \cdot p :: P) \cdot \mathit{s2xf}_{\mathrm{B}}(_, v)
\end{aligned}
$$

逆方向変換 $\mathit{c2xf}_{\mathrm{B}}$ を用いると，ビュー上で章題に対応する <h1> から次の章題に対応する <h1> までの生垣の長さを変更できない．これは，ソースにおいて <chapter> の数を変更できないことに対応する．ビュー上で章題に対応する <h1> から次の章題に対応する <h1> の間には，設題に対応する <h1> を含むことに注意する．このとき，ビュー上で <p> を対応して減らし節題に対応する <h1> を挿入することで，ソースにおいて章の直下の段落を減らし節を挿入することができる．ここで挿入された節題がどの章に属するかは，挿入した位置により決まる．先程の *lengthA* と同様の考え方を用いて得た補関数により得られる逆方向変換は，以下の関数 $\mathit{c2xf}'_{\mathrm{B}}$ と等価なものである（*splitByH1* と *lenghtH1* の定義は省略）．

$$
\begin{aligned}
&\textbf{data}\ S \mathrel{\hat{=}} (\texttt{<section>}(\texttt{<title>}(\mathit{String}) \cdot P))^{*}\\
&\textbf{data}\ P \mathrel{\hat{=}} (\texttt{<p>}(\mathit{String}))^{*}
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{c2xf}'_{\mathrm{B}}(\varepsilon, \varepsilon) \mathrel{\hat{=}} \varepsilon\\
&\mathit{c2xf}'_{\mathrm{B}}(\texttt{<chapter>}(\texttt{<title>}(t :: \mathit{String}) \cdot p :: P \cdot s :: S) \cdot r :: C, \texttt{<h1>}(t') \cdot p' \cdot v)\\
&\quad \mathrel{\hat{=}} \textbf{let}\ (v_1, v_2) \mathrel{\hat{=}} \mathit{splitByH1}(\mathit{lengthH1}(\mathit{s2xf}(s), v))\\
&\qquad \textbf{in}\ \texttt{<chapter>}(\texttt{<title>}(t') \cdot p' \cdot \mathit{s2xf}_{\mathrm{B}}(s, v_1)) \cdot \mathit{c2xf}'_{\mathrm{B}}(r, v_2)\\
&\mathit{s2xf}_{\mathrm{B}}(_, \varepsilon) \mathrel{\hat{=}} \varepsilon\\
&\mathit{s2xf}_{\mathrm{B}}(_, \texttt{<h1>}(t) \cdot p \cdot v)\\
&\quad \mathrel{\hat{=}} \texttt{<section>}(\texttt{<title>}(t :: \mathit{String}) \cdot p :: P) \cdot \mathit{s2xf}_{\mathrm{B}}(_, v)
\end{aligned}
$$

これを用いると，章題に対応する <h1> から別の章題に対応する <h1> までにある <h1> の数を変更することはできなくなるが，段落 <p> を自由に挿入し削除することが可能になる．挿入された段落が章直下の段落になるか節下の段落になるかは，ビュー上で直前の <h1> は章題であるか節題であるかによって決まる．

現在，後者の *lengthA* や *lengthH1* のような関数を使用した補関数を，どのようにすれば連接演算子の引数の型の構造より導出できるかはまだわかっていない．

連接の単射性判定

第 8 章において，我々は，それぞれの式の値域の厳密な記述の上で重なりの判定を行うことでより小さい補関数が導出できる場合があることを示した．しかし，水平方向の重なりの

判定，すなわち連接演算が単射でないかどうかの判定に，式の厳密な型を効果的に利用する手法はわかっていない．たとえば，第 8 章の例 8.8 に加え，以下の関数における連接演算の単射性も，正規生垣文法により近似した型の上では判定できない．

$$twice(x) \triangleq x \cdot x$$

$$\begin{array}{ll} exp(x) & \triangleq \textbf{let}\,(s,t) \triangleq f(x)\,\textbf{in}\ s \cdot t \\ f(\varepsilon) & \triangleq (\texttt{<s>}, \texttt{<s>}) \\ f(\texttt{<s>} \cdot r) & \triangleq \textbf{let}\,(s,t) \triangleq f(x)\,\textbf{in}\ (s \cdot t, s \cdot t) \end{array}$$

水平方向の重なりの判定において，どのように式の値域の厳密な記述を利用してよいかは明らかでない．第 4 章で述べた通り，我々は，集合 T_1 と T_2 が水平方向に重なりがないかどうかの判定を以下の通りに行っている．

1. $T_1' = \{x \cdot \texttt{<\#>}_1 \cdot y \mid x \cdot y \in T_1\}$ を求める．
2. $T_2' = \{x \cdot \texttt{<\#>}_2 \cdot y \mid x \cdot y \in T_2\}$ を求める．
3. $T_1'' = \{x \cdot \texttt{<\#>}_1 \cdot y \mid x \in T_1, y \in T_2'\}$ を求める．
4. $T_2'' = \{x \cdot \texttt{<\#>}_2 \cdot y \mid x \in T_1', y \in T_2\}$ を求める．
5. $T_3'' = \{x \cdot \texttt{<\#>}_1 \cdot y \cdot \texttt{<\#>}_2 \cdot z \mid x, y, z \in \mathcal{H}_\Sigma, |y| \geq 1\}$ を求める．ただし，Σ は T_1 に含まれるラベルと T_2 に含まれるラベルとの和集合である．
6. $T_1'' \cap T_2'' \cap T_3''$ が空かどうか調べる．

上の判定は，T_1 と T_2 が正規生垣言語である場合は，$T_1', T_2', T_1'', T_2'', T_3''$ がそれぞれ正規生垣言語となるため効果的に行うことができる．しかし，ステップ 2 とステップ 3 を文脈自由多生垣文法で表現される T_1, T_2 に対しどのように行えばよいかは明らかではない．また，仮に水平方向の重なりが効果的に判定でき連接演算の単射性を効果的に判定できたとしても，逆方向変換においてどのようにその連接結果を分解してよいかわかっていない．

複製の合成可能な取り扱い

第 6 章では，treeless な言語において変数の複数回出現には以下の二種類があることを述べた．

- 入力の複数回走査（入力の複製）
- 変換結果の複製（出力の複製）

この内，前者が双方向化において問題となる．そのため，本論文では，前者の変数の複数回出現を陽に扱うことをせず，多返値関数を用いることにより前者を含む変換の一部を表現した（第6章）．すなわち，関数

$$h(x) = (f(x), g(x)),$$

のような入力の複数回走査を含む順方向変換を双方向化するのに，f と g を別個に双方向化しその結果を利用するのではなく，組化 [HITT97, Chi93] により得られる f と g を使用しない形で記述した h の定義に対し双方向化を行った．それに対し，上のような h に対し，f と g の双方向化結果をうまく組み合わせたり，f と g の双方向化を工夫したりすることで，振る舞いがよく効果的な h の逆方向変換を求めることが考えられる．しかし，どのようにして f と g の逆方向変換を組み合わせ h の振る舞いがよく効果的な逆方向変換を得ればよいかはまだわかっていない．つまり，複製に対する，モジュール性の意味で合成可能な取り扱いはまだ明らかではない．

補関数に基づく双方向化において，複製に対する合成可能な取り扱いがどのように難しいかについて以下に述べる．

先程の関数 $h = \langle f, g \rangle$ について，Hu らの複製 [HMT04] のように，ビューの第一要素に対する更新を f^{c} の定める逆方向変換により反映し，第二要素に対する更新を g^{c} の定める逆方向変換により反映するのは，双方向変換の合成として自然である．もし，このような合成された逆方向変換に対応する h の補関数が存在したとすると，$h^{\mathrm{c}} \preceq f^{\mathrm{c}}$ かつ $h^{\mathrm{c}} \preceq g^{\mathrm{c}}$ を満たす．なぜなら，f^{c} の定める逆方向変換によりソースが s_1 から s_2 に更新可能である場合，h^{c} の定める逆方向変換でも s_1 から s_2 に更新可能であるためである（第3章，定理3.6証明，定理3.7）．ところが，関数 $h = \langle f, g \rangle$ について，f と g それぞれの補関数 f^{c} と g^{c} が得られていたとしても，h の補関数 h^{c} で，$h^{\mathrm{c}} \preceq f^{\mathrm{c}}$ と $h^{\mathrm{c}} \preceq g^{\mathrm{c}}$ となるものが存在するとは限らない．たとえば，第6章の $divs(x) = (div_2(x), div_3(x))$ の例について，div_2 の補関数を「入力を2で割った余り」，div_3 の補関数を「入力を3で割った余り」にとると，どちらの補関数と比べてもより小さいか等しい $divs$ の補関数は存在しない．なぜなら，任意の自然数 x と y に対し，

$$\exists z.\ x = z \bmod 2 \wedge z = y \bmod 3$$

となるため「入力を2で割った余り」よりも「入力を3で割った余り」よりも小さいか等しい関数は任意の入力について値の等しい定数関数なるが，定数関数は $divs$ の補関数ではないためである．

関数 $h = \langle f, g \rangle$ に対し，$h^{\mathrm{c}} \preceq f^{\mathrm{c}}$ かつ $h^{\mathrm{c}} \preceq g^{\mathrm{c}}$ となる h の補関数 h^{c} が存在することは以下と等価である．

$$(\equiv_h) \cap ((\equiv_{f^{\mathrm{c}}}) \cup (\equiv_{g^{\mathrm{c}}}))^* = (\equiv_{\mathrm{id}})$$

なお，$((\equiv_{f^{\mathrm{c}}}) \cup (\equiv_{g^{\mathrm{c}}}))^*$ は，$(\equiv_{f^{\mathrm{c}}})$ と $(\equiv_{g^{\mathrm{c}}})$ とを含む最小の同値関係である．ここで，

$$((\equiv_{f^{\mathrm{c}}}) \cup (\equiv_{g^{\mathrm{c}}}))^* = ((\equiv_{f^{\mathrm{c}}}) \circ (\equiv_{g^{\mathrm{c}}}))^*$$

であることに注意する．今，f_{B} と g_{B} をそれぞれ f^{c} と g^{c} によって定まる f と g の逆方向変換とする．第 3 章の定理 3.1 により，任意の $f_{\mathrm{B}} = \mathsf{reflect}_{f,f^{\mathrm{c}}}$ について，

$$\forall s, s'.\ \big((\exists v.\ f_{\mathrm{B}}(s,v) = s') \Leftrightarrow (s \equiv_{f^{\mathrm{c}}} s')\big)$$

が成り立つ．すなわち，$s\ ((\equiv_{f^{\mathrm{c}}}) \circ (\equiv_{g^{\mathrm{c}}}))^*\ s'$ であるということは，f_{B} および g_{B} を交互に有限回数繰り返し適用することによって s が s' に更新されることを意味する．

定理 9.1. 関数 f と g について，それぞれの補関数 f^{c} と g^{c} が定まり，対応したそれぞれの逆方向変換 $f_{\mathrm{B}} = \mathsf{reflect}_{f,f^{\mathrm{c}}}$ と $g_{\mathrm{B}} = \mathsf{reflect}_{g,g^{\mathrm{c}}}$ が求まっているとする．このとき，$h = \langle f, g\rangle$ の Hu らの複製により定義された逆方向変換[1]が振る舞いがよいのは，以下が成り立つ場合でありかつそのときに限る．

$$(\equiv_h) \cap ((\equiv_{f^{\mathrm{c}}}) \circ (\equiv_{g^{\mathrm{c}}}))^* = (\equiv_{\mathrm{id}}) \qquad \square$$

ところが，我々の知る限り，$((\equiv_{f^{\mathrm{c}}}) \circ (\equiv_{g^{\mathrm{c}}}))^*$ に対応する関数，すなわち

$$(\equiv_k) = ((\equiv_{f^{\mathrm{c}}}) \circ (\equiv_{g^{\mathrm{c}}}))^*$$

を満たす関数 k の構成方法はわかっていない．もしも上の k が得られると，$\langle h, k\rangle$ の単射性解析を行うことで，Hu らの複製により定義される逆方向変換の振る舞いがよいかどうかを判定できるかもしれない．

9.2.2 順方向変換記述言語

累積変数

本論文では，我々は macro forest transducers [PS04] にあるような累積変数を含む順方向変換の双方向化を議論していない．本論文で用いた手法のうち，たとえば，本論文の第 7 章の手法や，第 8 章の関数の値域の厳密な推定とその近似などは，累積変数を含む順方向変換についても適用できる．これらの手法は，関数の出力がパターンマッチにより分解されないことに基づいているためである．ところが，素朴な拡張により得られる正規生垣言語の上での単射性解析やそれに基づく補関数導出は，あまり効果的なものにはならない場合がある．

たとえば，以下の $\mathit{reverseAcc}$ は，入力の文字列を逆転する関数であり，累積変数を用いて定義されている．

$$\begin{aligned} \mathit{reverseAcc}(x) &\mathrel{\hat{=}} \mathit{rev}(x, \varepsilon) \\ \mathit{rev}(\varepsilon, y) &\mathrel{\hat{=}} y \\ \mathit{rev}(a :: \mathit{Char} \centerdot x, y) &\mathrel{\hat{=}} \mathit{rev}(x, a \centerdot y) \end{aligned}$$

関数 $\mathit{reverseAcc}$ は明らかに単射である．ところが，素朴に rev の第一規則と第二規則の右辺式の型を正規言語で近似すると，それらの型の間には重なりが出てしまう．なぜなら，第

[1] Hu らの言う逆方向変換は，第 3 節における逆方向変換ではないことに注意する．ただし，彼らの逆方向変換の定義域を順方向変換の値域に制限すれば同様の議論ができる．

一規則は入力となる y をそのまま返しているため，どのような値も rev の第一規則の右辺式の評価結果になりうるためである．

累積変数を使って定義された関数について，値域を厳密に表現する文法も以下のように累積変数を持つ．

$$\begin{array}{ll} T_{reverseAcc} & \to T_{rev}(\varepsilon) \\ T_{rev}(y) & \to y \\ T_{rev}(y) & \to T_{rev}(Char \centerdot y) \end{array}$$

これは，直観的には，累積変数を持つ関数は，累積変数に対しどのような計算が行われるかにより結果が変わるためである．また，値域を厳密に表現する文法が累積変数を持つことは，linear macro tree transducer の値域が文脈自由木文法 [CDG+97] で表現できる [MPS07] ことと同様である．この値域を記述する文法から，rev の第一規則と第二規則の右辺式の値域に重なりがあるかないかを議論するのは簡単ではない．

集合および辞書

本論文では，XML 文書を生垣として扱い，生垣の上の変換の双方向化を議論した．生垣においては要素の並び順が意味を持つ．たとえば，生垣 <a>.<b> と生垣 <b>.<a> は異なる．しかし，XML 文書中には並び順が意味を持たない XML 要素列が含まれる場合も多く，それを考慮することにより効果的な逆方向変換を得られることがある．

たとえば，ある述語 p に対して定まる以下の順方向変換 $filter$ を考える．

$$\begin{array}{lll} \textbf{data}\ T \mathrel{\hat{=}} \ldots & & \text{-- } x \in [\![P_1]\!] \Leftrightarrow \textsc{length}(x) = 1 \wedge p(x) \\ \textbf{data}\ F \mathrel{\hat{=}} \ldots & & \text{-- } x \in [\![P_2]\!] \Leftrightarrow \textsc{length}(x) = 1 \wedge \neg p(x) \\ \mathit{filter}(\varepsilon) & \mathrel{\hat{=}} \varepsilon & \\ \mathit{filter}(a :: T \centerdot x) & \mathrel{\hat{=}} \textbf{let}\, y \mathrel{\hat{=}} \mathit{filter}(x)\, \textbf{in}\ a \centerdot y & \\ \mathit{filter}(a :: F \centerdot x) & \mathrel{\hat{=}} \textbf{let}\, y \mathrel{\hat{=}} \mathit{filter}(x)\, \textbf{in}\ y & \end{array}$$

本論文の手法により求まる補関数は以下である．

$$\begin{array}{ll} \mathit{filter}^{\mathrm{c}}(\varepsilon) & \mathrel{\hat{=}} \mathsf{B}_1 \\ \mathit{filter}^{\mathrm{c}}(a :: T \centerdot x) & \mathrel{\hat{=}} \textbf{let}\, y^{\mathrm{c}} \mathrel{\hat{=}} \mathit{filter}^{\mathrm{c}}(x)\, \textbf{in}\ \mathsf{B}_2(y^{\mathrm{c}}) \\ \mathit{filter}^{\mathrm{c}}(a :: F \centerdot x) & \mathrel{\hat{=}} \textbf{let}\, y^{\mathrm{c}} \mathrel{\hat{=}} \mathit{filter}^{\mathrm{c}}(x)\, \textbf{in}\ \mathsf{B}_3(y^{\mathrm{c}}, a) \end{array}$$

直観的には，構成子 $\mathsf{B}_1, \mathsf{B}_2, \mathsf{B}_3$ は $filter$ により抽出されない要素の元の列における位置を表現している．これは，逆方向変換により $filter$ で抽出される要素の位置，すなわち順序を保存するためである．ところで，並び順に意味を持たない列については，順序を保存する必要はない．よって，以下の $filter^{\mathrm{c}\prime}$ のようなより小さい関数が補関数となる．

$$\begin{array}{ll} \mathit{filter}^{\mathrm{c}\prime}(\varepsilon) & \mathrel{\hat{=}} \varepsilon \\ \mathit{filter}^{\mathrm{c}\prime}(a :: T \centerdot x) & \mathrel{\hat{=}} \textbf{let}\, y^{\mathrm{c}} \mathrel{\hat{=}} \mathit{filter}^{\mathrm{c}\prime}(x)\, \textbf{in}\ y^{\mathrm{c}} \\ \mathit{filter}^{\mathrm{c}\prime}(a :: F \centerdot x) & \mathrel{\hat{=}} \textbf{let}\, y^{\mathrm{c}} \mathrel{\hat{=}} \mathit{filter}^{\mathrm{c}\prime}(x)\, \textbf{in}\ a \centerdot y^{\mathrm{c}} \end{array}$$

正規化子（canonizer）[FPP08] を用いることで，順方向変換および逆方向変換の際に，見た目は異なるが同じ意味を持つデータを統一することができる．たとえば，入力の並び順に意味を持たない場合には，正規化子は入力を整列する．たしかに，正規化子は実際の問題に双方向変換を適用する際には有用である．しかし，効果的な双方向変換を得るためには，正規化後のデータ構造をどう選ぶか，そしてそのデータ上の変換の双方向化をどうするかは別に議論する必要がある．たとえば，*filter* の場合，入力が整列されているからといって，$\mathit{filter}^{c\prime}$ を得るのは容易ではない．

補関数 $\mathit{filter}^{c\prime}$ を得る有効な方法の一つは，集合を明示的に扱う方法である．すなわち，集合というデータを陽に扱い，*filter* を集合に対する基本的な演算して取り扱う方法である．集合に対しどのような基本的な演算を提供すれば，多くの並び順が意味を持たない列上の変換が記述できるか検討する必要がある．また，集合の各要素は，キーと呼ばれる集合内でその要素を一意に差し示す部分データを持つことがよくある．キーを持つ集合は，各要素がキーにより一意に指し示されていると考えると，辞書であるとも考えられる．双方向変換における辞書の取り扱いは文献 [BFP+08] で議論されてはいるものの，補関数に基づく双方向化に応用できるかはわかっていない．集合というデータを陽に扱う際において，関係データベースは組の集合に近いため，我々は関係データベース上で補関数導出の議論 [LLSV01, LV03] が応用できるのではないかと考えている．

補関数に基づく双方向化と決定的な逆関数導出に関する予想

本論文を通してプログラムの双方向化を議論するにあたって，我々は以下の予想を得た．

予想． 関数に対する単射性解析手法は，その自然な補関数導出法を与え，またもし単射であったならばその決定的な逆関数の導出手法を与える．

たとえば，第 5 章において，我々は式の値域を正規木文法で表現することにより，明らかに単射でない場所で失った情報を補うことで補関数を構成した．また，単射であった場合に得られる逆関数は，先読み付き決定的 top-down tree transducers [Eng77] の同様の形式で構成することにより，決定的に評価できる．また，第 8 では，同様の手法により決定的な逆方向変換を導出している．

また，Glück と Kawabe の逆関数導出の際に LR 構文解析器を使用する手法 [GK04] も同様であるといえる．彼らは，プログラムをコマンド列を記述する文脈自由文法であると見なし，逆プログラムの決定性を LR 構文解析器を用いて解析することにより，決定的な逆プログラムを得ている．

この予想に基づき，我々は，LR 構文解析器のような文脈自由文法に対する構文解析を利用したアプローチにより，累積変数を含む順方向変換プログラムに対し効果的な双方向変換を導出できるのではないかと考えている．

謝辞

まず最初に，修士課程時代の私の指導教員であり，また博士課程における私の指導教員である胡振江准教授に大変深く感謝申し上げます．胡准教授の情熱的なコメントに私は研究において度々励まされました．そして，博士課程における私のもう一人の指導教員である武市正人教授に深謝致します．武市教授は大局的な視点から思慮深い様々な助言を授けてくださいました．また，胡准教授も武市教授もお忙しい中私のために研究に関する議論の時間を設けて下さいました．本当にありがとうございました．

また，電気通信大学の中野圭介助教および群馬大学の浜名誠助教には，研究にあたり多大な協力と助言を頂きました．特に第5章の内容は彼等の協力なしには達成できませんでした．また，中野助教と浜名助教には研究の契機やヒントになった様々な興味深い文献を紹介していただきました．大変感謝しております．

さらに，海外で勉強する機会を設けて下さいましたコペンハーゲン大学の Robert Glück 准教授に深く感謝致します．また，コペンハーゲンにおいて，議論に付き合って頂き，また慣れぬ海外生活を助けて下さった Johan Gade さんに感謝致します．コペンハーゲンでの三週間は短い間でしたが，本研究の着想において多大なインスピレーションを得ることができました．

時にはホワイトボードの前で真剣に，時には酒とともに気楽に，研究に関し遅くまで議論につきあってくれた森畑明昌さんと江本健斗さんに感謝致します．有益な TeX 技術を教えて下さり，計算機環境のカスタマイズや大学生活において様々な助言をくださった松崎公紀助教に感謝致します．

最後になりますが，研究において様々な刺激をあたえてくれた研究室の学生と，金銭面で私の研究をサポートしてくださった両親に感謝致します．

参考文献

[Abr05] Samson Abramsky. A structural approach to reversible computation. *Theoretical Computer Science*, Vol. 347, No. 3, pp. 441–464, 2005.

[AU72] Alfred V. Aho and Jeffrey D. Ullman. *The theory of parsing, translation, and compiling*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1972.

[Bak92] Henry G. Baker. Nreversal of fortune - the thermodynamics of garbage collection. In *IWMM '92: Proceedings of the International Workshop on Memory Management*, pp. 507–524, London, UK, 1992. Springer-Verlag.

[BCF03] Véronique Benzaken, Giuseppe Castagna, and Alain Frisch. CDuce: an XML-centric general-purpose language. In *ICFP '03: Proceedings of the 2003 ACM SIGPLAN international conference on Functional programming*, pp. 51–63, 2003.

[BDH04] Vanessa P. Braganholo, Susan B. Davidson, and Carlos A. Heuser. From XML view updates to relational view updates: old solutions to a new problem. In *VLDB '04: International Conference on Very Large Data Bases*, pp. 276–287. Morgan Kaufmann, 2004.

[Ben89] Charles H. Bennett. Time/space trade-offs for reversible computation. *SIAM Journal on Computing*, Vol. 18, No. 4, pp. 766–776, 1989.

[BFP+08] Aaron Bohannon, J. Nathan Foster, Benjamin C. Pierce, Alexandre Pilkiewicz, and Alan Schmitt. Boomerang: resourceful lenses for string data. In *POPL '08: Proceedings of the 35th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pp. 407–419, New York, NY, USA, 2008. ACM.

[BGM07] Claus Brabrand, Robert Giegerich, and Anders Møller. Analyzing ambiguity of context-free grammars. In *Proceedings of 12th International Conference on Implementation and Application of Automata, CIAA '07*, Vol. 4783 of *LNCS*. Springer-Verlag, July 2007.

[Bir98] Richard S. Bird. *Introduction to Functional Programming Using Haskell.* Prentice Hall, 1998.

[BMS08] Claus Brabrand, Anders Møller, and Michael I. Schwartzbach. Dual syntax for XML languages. *Information Systems*, Vol. 33, No. 4, June 2008. Earlier version in Proc. 10th International Workshop on Database Programming Languages, DBPL '05, Springer-Verlag LNCS vol. 3774.

[BP99] Peter Bunneman and Benjamine C. Pierce. Union types for seminstrcutred data. Technical Report MS-CIS-99-09, University of Pennsylvania Department of Computer and Information Science, 1999.

[BS81] François Bancilhon and Nicolas Spyratos. Update semantics of relational views. *ACM Transactions of Database Systems*, Vol. 6, No. 4, pp. 557–575, 1981.

[CDG+97] Hubert Comon, Max Dauchet, Remi Gilleron, Florent Jacquemard, Denis Lugiez, Sophie Tison, and Marc Tommasi. Tree automata techniques and applications. Available on: `http://www.grappa.univ-lille3.fr/tata`, 1997.

[CF03] Jorge Coelho and Mário Florido. Type-based XML processing in logic programming. In *PADL '03: Proceedings of the 5th International Symposium on Practical Aspects of Declarative Languages*, pp. 273–285, London, UK, 2003. Springer-Verlag.

[Chi93] Wei-Ngan Chin. Towards an automated tupling strategy. In *PEPM '93: Proceedings of the 1993 ACM SIGPLAN symposium on Partial evaluation and semantics-based program manipulation*, pp. 119–132, New York, NY, USA, 1993. ACM.

[CP84] Stavros S. Cosmadakis and Christos H. Papadimitriou. Updates of relational views. *Journal of the ACM*, Vol. 31, No. 4, pp. 742–760, 1984.

[DB82] Umeshwar Dayal and Philip A. Bernstein. On the correct translation of update operations on relational views. *ACM Transactions of Database Systems*, Vol. 7, No. 3, pp. 381–416, 1982.

[Dij78] Edsger W. Dijkstra. Program inversion. In *Program Construction*, Vol. 69 of *LNCS*, pp. 54–57. Springer, 1978.

[Eng75] Joost Engelfriet. Bottom-up and top-down tree transformations — a comparison. *Mathematical Systems Theory*, Vol. 9, No. 3, pp. 198–231, 1975.

[Eng77] Joost Engelfriet. Top-down tree transducers with regular look-ahead. *Mathematical Systems Theory*, Vol. 10, pp. 289–303, 1977.

[Epp85] David Eppstein. A heuristic approach to program inversion. In *International Joint Conference on Artificial Intelligence (IJCAI-85)*, pp. 219–221, 1985.

[FGM+05] J. Nathan Foster, Michael B. Greenwald, Jonathan T. Moore, Benjamin C. Pierce, and Alan Schmitt. Combinators for bi-directional tree transformations: a linguistic approach to the view update problem. In *POPL '05: Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pp. 233–246, New York, NY, USA, 2005. ACM Press.

[FPP08] J. Nathan Foster, Alexandre Pilkiewicz, and Benjamin C. Pierce. Quotient lenses. In *ICFP '08: Proceeding of the 13th ACM SIGPLAN international conference on Functional programming*, pp. 383–396, New York, NY, USA, 2008. ACM.

[Fra99] Michael P. Frank. *Reversibility for Efficient Computing*. PhD thesis, CISE Department, University of Florida, 1999.

[Fri04] Alain Frisch. Regular tree language recognition with static information. In *Exploring New Frontiers of Theoretical Informatics, IFIP 18th World Computer Congress, TC1 3rd International Conference on Theoretical Computer Science (TCS2004)*, pp. 661–674, 2004.

[Fül94] Zoltán Fülöp. Undecidable properties of deterministic top-down tree transducers. *Theoretical Computer Science*, Vol. 134, No. 2, pp. 311–328, 1994.

[GK04] Robert Glück and Masahiko Kawabe. Derivation of deterministic inverse programs based on LR parsing. In *FLOPS '04: The Seventh International Symposium on Functional and Logic Programming*, pp. 291–306, 2004.

[GK05] Robert Glück and Masahiko Kawabe. Revisiting an automatic program inverter for lisp. *SIGPLAN Notices*, Vol. 40, No. 5, pp. 8–17, 2005.

[GPZ86] Georg Gottlob, Paolo Paolini, and Roberto Zicari. Properties and update semantics of consistent views. Technical Report UCB/CSD-86-258, EECS Department, University of California, Berkeley, 1986.

[GPZ88] Georg Gottlob, Paolo Paolini, and Roberto Zicari. Properties and update semantics of consistent views. *ACM Transactions of Database Systems*, Vol. 13, No. 4, pp. 486–524, 1988.

[Gri81] David Gries. *The Science of Programming*, chapter Chapter 21 Inverting Programs. Springer, 1981.

[Heg90] Stephen J. Hegner. Foundations of canonical update support for closed database views. In *ICDT '90: Proceedings of the Third International Conference on Database Theory*, pp. 422–436, London, UK, 1990. Springer-Verlag.

[Heg04] Stephen J. Hegner. An order-based theory of updates for closed database views. *Annals of Mathematics and Artificial Intelligence*, Vol. 40, No. 1-2, pp. 63–125, 2004.

[HITT97] Zhenjiang Hu, Hideya Iwasaki, Masato Takeichi, and Akihiko Takano. Tupling calculation eliminates multiple data traversals. In *ICFP '97: Proceedings of the second ACM SIGPLAN International Conference on Functional Programming*, pp. 164–175, New York, NY, USA, 1997. ACM Press.

[HMT04] Zhenjiang Hu, Shin-Cheng Mu, and Masato Takeichi. A programmable editor for developing structured documents based on bidirectional transformations. In *PEPM '04: Proceedings of the 2004 ACM SIGPLAN Symposium on Partial Evaluation and Semantics-based Program Manipulation*, pp. 178–189, New York, NY, USA, 2004. ACM Press.

[HMT08] Zhenjiang Hu, Shin-Cheng Mu, and Masato Takeichi. A programmable editor for developing structured documents based on bidirectional transformations. *Higher-Order and Symbolic Computation*, Vol. 21, No. 1-2, pp. 89–118, 2008.

[HMU06] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*, chapter 7. Addison-Wesley Longman Publishing Co., Inc., 2006.

[Hos03] Haruo Hosoya. Regular expression pattern matching — a simpler design. Technical Report 1397, RIMS, Kyoto University, 2003.

[HP03] Haruo Hosoya and Benjamin C. Pierce. XDuce: A statically typed XML processing language. *ACM Transactions on Internet Technology*, Vol. 3, No. 2, pp. 117–148, 2003.

[IHM08] Kazuhiro Inaba, Haruo Hosoya, and Sebastian Maneth. Multi-return macro tree transducers. In *Proceedings of 13th International Conference on Implementation and Application of Automata, CIAA '08*, pp. 102–111, 2008.

[JGS93] Neil D. Jones, Carsten K. Gomard, and Peter Sestoft. *Partial Evaluation and Automatic Program Generation*. Prentice Hall International, June 1993.

[Kel87] Arthur M. Keller. Comments on Bancilhon and Spyratos' "Update semantics and relational views". *ACM Transactions on Database Systems*, Vol. 12, No. 3, pp. 521–523, 1987.

[KH06] Shinya Kawanaka and Haruo Hosoya. bixid: a bidirectional transformation language for XML. In *ICFP '06: Proceedings of the eleventh ACM SIGPLAN international conference on Functional programming*, pp. 201–214, New York, NY, USA, 2006. ACM.

[Kru08] Michael Kruse. Ambiguity detection for context-free grammars in eli, 5 2008. Bachelor 's Thesis in Faculty of Computer Science, Electrical Engineering and Mathematics, Universtäte Paderborn.

[Lan61] Rolf Landauer. Irreversibility and heat generation in the computing process. *IBM Journal of Research and Development*, Vol. 5, No. 3, pp. 183–191, July 1961.

[Lan90] Rom Langerak. View updates in relational databases with an independent scheme. *ACM Transactions on Database Systems*, Vol. 15, No. 1, pp. 40–66, 1990.

[LD82] Chris Lutz and Howard Derby. Janus: A time-reversible language. Caltech class project, California Institute of Technology, 1982. `http://www.cise.ufl.edu/~mpf/rc/janus.html`.

[LHT07] Dongxi Liu, Zhenjiang Hu, and Masato Takeichi. Bidirectional interpretation of xquery. In *PEPM '07: Proceedings of the 2007 ACM SIGPLAN symposium on Partial evaluation and semantics-based program manipulation*, pp. 21–30, New York, NY, USA, 2007. ACM.

[LLSV01] Dominique Laurent, Jens Lechtenbörger, Nicolas Spyratos, and Gottfried Vossen. Monotonic complements for independent data warehouses. *The VLDB Journal*, Vol. 10, No. 4, pp. 295–315, 2001.

[LP05] Michael Y. Levin and Benjamin C. Pierce. Type-based optimization for regular patterns. In *DBPL '05: Proceedings of 10th International Symposium on Database Programming Languages*, pp. 184–198, 2005.

[LV03] Jens Lechtenbörger and Gottfried Vossen. On the computation of relational view complements. *ACM Transactions of Database Systems*, Vol. 28, No. 2, pp. 175–208, 2003.

[Mas84] Yoshifumi Masunaga. A relational database view update translation mechanism. In *VLDB '84: Proceedings of the 10th International Conference on Very Large Data Bases*, pp. 309–320, San Francisco, CA, USA, 1984. Morgan Kaufmann Publishers Inc.

[Mee98] Lambert Meertens. Designing constraint maintainers for user interaction. `http://www.cwi.nl/~lambert`, June 1998.

[Mel03] I. Dan Melamed. Multitext grammars and synchronous parsers. In *NAACL '03: Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, pp. 79–86, Morristown, NJ, USA, 2003. Association for Computational Linguistics.

[MHN+07] Kazutaka Matsuda, Zhenjiang Hu, Keisuke Nakano, Makoto Hamana, and Masato Takeichi. Bidirectionalization transformation based on automatic derivation of view complement functions. In *ICFP '07: Proceedings of the 2007 ACM SIGPLAN International Conference on Functional Programming*, pp. 47–58, New York, NY, USA, 2007. ACM.

[MHT04a] Shin-Cheng Mu, Zhenjiang Hu, and Masato Takeichi. An algebraic approach to bidirectional updating. In *APLAS '04: Second ASIAN Symposium on Programming Languages and Systems*, pp. 2–18. Springer Verlag, 2004.

[MHT04b] Shin-Cheng Mu, Zhenjiang Hu, and Masato Takeichi. An injective language for reversible computation. In *MPC 2004: Mathematics of Program Construction 7th International Conference*, pp. 289–313, London, UK, 2004. Springer-Verlag.

[MHT06] Shin-Cheng Mu, Zhenjiang Hu, and Masato Takeichi. Bidirectionalizing tree transformation languages: a case study. コンピュータ・ソフトウェア, Vol. 23, No. 2, pp. 129–141, 2006.

[MHT09] Kazutaka Matsuda, Zhenjiang Hu, and Masato Takeichi. Type-based specialization of XML transformations. In *PEPM '09: the 2009 ACM SIGPLAN symposium on Partial evaluation and semantics-based program manipulation*, 2009.

[MLMK05] Makoto Murata, Dongwon Lee, Murali Mani, and Kohsuke Kawaguchi. Taxonomy of XML schema languages using formal language theory. *ACM Transactions on Internet Technology*, Vol. 5, No. 4, pp. 660–704, 2005.

[MN01] Mehryar Mohri and Mark-Jan Nederhof. Regular approximation of context-free grammars through transformation. In Jean-Claude Junqua and Gertjan van Noord, editors, *Robusteness in Language and Speech Technology.* Kluwer Academic Publishers, The Netherlands, 2001.

[MPS07] Sebastian Maneth, Thomas Perst, and Helmut Seidl. Exact XML type checking in polynomial time. In *ICDT '07: Proceedings of the 11th International Conference of Database Theory*, Vol. 4353 of *LNCS*, pp. 254–268, 2007.

[MSW04] I. Dan Melamed, Giorgio Satta, and Benjamin Wellington. Generalized multitext grammars. In *Proceedings of the 42nd Annual Conference of the Association for Computational Linguistics (ACL)*, pp. 661–668, 2004.

[Mur99] Murata. Hedge automata: a formal model for XML schemata, 1999. Available on: `http://www.xml.gr.jp/relax/hedge_nice.html`.

[NSS05] Naoki Nishida, Masahiko Sakai, and Toshiki Sakabe. Partial inversion of constructor term rewriting systems. In *RTA '05: Rewriting Techniques and Applications*, pp. 264–278, 2005.

[OST03] Hitoshi Ohsaki, Hiroyuki Seki, and Toshinori Takai. Recognizing boolean closed A-tree languages with membership conditional rewriting mechanism. In *Rewriting Techniques and Applications*, Vol. 2706 of *LNCS*, pp. 483–498, 2003.

[PF99] Kalyan Perumalla and Richard Fujimoto. Source-code transformations for efficient reversibility. Technical Report GIT-CC-99-21, College of Computing, Georgia Institute of Technology, 09 1999.

[PHW06] Alessandra Di Pierro, Chris Hankin, and Herbert Wiklicky. On reversible combinatory logic. *Electronic Notes in Theoretical Computer Science*, Vol. 135, No. 3, pp. 25–35, 2006.

[Pos46] Emil L. Post. A variant of a recursively unsolvable problem. *Bulletin of the American Mathematical Society*, Vol. 52, No. 4, pp. 264–268, 1946.

[PS04] Thomas Perst and Helmut Seidl. Macro forest transducers. *Information Processsing Letters*, Vol. 89, No. 3, pp. 141–149, 2004.

[Rey83] John C. Reynolds. Types, abstraction and parametric polymorphism. In *Information Processing 83, Proceedings of the IFIP 9th World Computer Congress*, pp. 513–523, 1983.

[Voi09] Voigtländer. Bidirectionalization for free! In *POPL '09: Proceedings of the 35th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, 2009.

[Wad90] Philip Wadler. Deforestation: Transforming programs to eliminate trees. *Theoretical Computer Science*, Vol. 73, No. 2, pp. 231–248, 1990.

[WR04] Ling Wang and Elke A. Rundensteiner. On the updatability of XML views published over relational data. In *ER 2004: International Conference on Conceptual Modeling*, pp. 795–809, 2004.

[XHS+08] Yingfei Xiong, Zhenjiang Hu, Hui Song, Masato Takeichi, Haiyan Zhao, and Hong Mei. On-site synchronizers for multi-view applications. In *Proceedings of the 25th JSSST Conference*, pp. No. 3C–2, 9 2008.

[YAG08a] Tetsuo Yokoyama, Holger Bock Axelsen, and Robert Glück. Principles of a reversible programming language. In *Proceedings of the 5th Conference on Computing Frontiers*, pp. 43–54, 2008.

[YAG08b] Tetsuo Yokoyama, Holger Bock Axelsen, and Robert Glück. Reversible flowchart languages and the structured reversible program theorem. In *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Proceedings, Part II - Track B: Logic, Semantics, and Theory of Programming & Track C: Security and Cryptography Foundations*, pp. 258–270, 2008.

[YG07] Tetsuo Yokoyama and Robert Glück. A reversible programming language and its invertible self-interpreter. In *Proceedings of the 2007 ACM SIGPLAN Workshop on Partial Evaluation and Semantics-based Program Manipulation*, pp. 144–153, 2007.

[松田 05] 松田一孝, 大川徳之, 野村芳明, 森田直幸, 筧一彦, 胡振江, 武市正人. 木上の双方向変換を利用したファイルマネージャの実現. 情報処理学会論文誌：トランザクション「プログラミング」, 2005.

[松田 09] 松田一孝, 胡振江, 中野圭介, 浜名誠, 武市正人. 補関数の生成による複製機能付きプログラムの自動双方向化. コンピュータ・ソフトウェア, 2009. 採録決定済.