



Title	階層関係と部分構造を持つ対象の表現と計算の定式化に関する研究
Author(s)	川口, 雄一
Degree Grantor	北海道大学
Degree Name	博士(工学)
Dissertation Number	甲第5117号
Issue Date	2000-03-24
DOI	https://doi.org/10.11501/3168681
Doc URL	https://hdl.handle.net/2115/51642
Type	doctoral thesis
File Information	000000353874.pdf



階層関係と部分構造を持つ対象の
表現と計算の定式化に関する研究

川口 雄一

学位論文

階層関係と部分構造を持つ対象の
表現と計算の定式化に関する研究

情報システム工学専攻

複雑系工学講座 表現系分野

川口 雄一

目次

要旨	8
第1章 はじめに	11
1.1 本研究の目的	11
1.2 本論文の概要	11
1.3 本章のまとめ	12
第2章 動機	13
2.1 計算機を用いた問題の解決	13
2.1.1 プログラムとアルゴリズム	13
2.1.2 アルゴリズムの生成	14
2.2 表現と計算と正当性	15
2.2.1 問題の表現	15
2.2.2 計算とその正当性	16
2.2.3 等価変換パラダイム	17
2.3 表現と計算のための座標系	18
2.3.1 概論	18
2.3.2 特殊化システムに基づく表現の定式化	19
2.4 確定節を利用した表現	20
2.5 等価変換に基づく計算の利点	21
2.6 定式化の対象の具体例	22
2.7 本研究の成果と意義	24

	3
2.8 本章のまとめ	25
第3章 関連する研究との比較	26
3.1 Typed Prolog	26
3.1.1 Typed Prolog の概要	26
3.1.2 Typed Prolog の表現と計算の理論	28
3.2 HiLog	28
3.2.1 HiLog の概要	28
3.2.2 HiLog の表現と計算の理論	30
3.3 λ Prolog	30
3.3.1 λ Prolog の概要	30
3.3.2 λ Prolog の表現と計算の理論	31
3.4 LOGIN	32
3.4.1 LOGIN の概要	32
3.4.2 LOGIN の表現と計算の理論	34
3.5 Typed Feature Structure	34
3.5.1 TFS の概要	34
3.5.2 TFS の表現と計算の理論	35
3.6 本章のまとめ	36
第4章 特殊化システムに基づく意味論	37
4.1 はじめに	37
4.2 特殊化システム	37
4.3 宣言的記述と意味	38
4.3.1 宣言的記述	38
4.3.2 宣言的記述の意味	39
4.4 特殊化システムの意義	40
4.5 本章のまとめ	41

	4
第5章 等価変換に基づく計算	42
5.1 等価変換	42
5.2 Unfold 変換	42
5.2.1 Unfold 変換の例	43
5.2.2 Unfold 変換の形式的な定義	44
5.3 本章のまとめ	45
第6章 階層関係の表現	46
6.1 はじめに	46
6.2 クラスとインスタンスの統合	46
6.3 列で表現されるクラス	48
6.4 列で表現されるクラスの階層関係	49
6.5 本章のまとめ	50
第7章 部分構造の表現	51
7.1 部分構造	51
7.2 省略記法	52
7.3 原子対象: $\mathcal{A}$	53
7.4 本章のまとめ	54
第8章 構造制約の導入	55
8.1 構造制約	55
8.2 基底対象: $\mathcal{G}$	57
8.3 本章のまとめ	58
第9章 述語名と定数名の統合	59
9.1 原子論理式と原子対象の対応	59
9.2 クラス対象の導入	59
9.2.1 変数を用いた原子対象の参照	59
9.2.2 変数を用いたクラスの参照の問題	61

	5
9.2.3 クラスのための変数の導入	63
9.3 原子対象 (2): $\mathcal{A}$	64
9.4 構造制約の記法	64
9.5 例: append の表現	65
9.6 本章のまとめ	66
第 10 章 対象に対する操作	68
10.1 はじめに	68
10.2 原子対象の単一化	68
10.2.1 両方の部分構造の長さが 0 である例	69
10.2.2 片方のみ部分構造の長さが 1 以上である例	69
10.2.3 両方の部分構造の長さが 1 以上である例	69
10.3 基礎的な特殊化: S, μ	70
10.4 本章のまとめ	70
第 11 章 LHS の表現の定式化	72
11.1 はじめに	72
11.2 階層関係	73
11.3 クラス対象	77
11.4 原子対象	77
11.5 基底対象	78
11.6 特殊化と原子対象への適用	79
11.7 特殊化システムの条件	81
11.8 具体例	82
11.9 本章のまとめ	84
第 12 章 LHS の計算の定式化	85
12.1 はじめに	85
12.2 等価変換に基づく表現の計算	85

	6
12.3 原子対象の単一化	85
12.4 単一化手続き	86
12.5 定理と証明	90
12.6 構造制約の充足	92
12.7 本章のまとめ	94
第 13 章 高階表現	95
13.1 はじめに	95
13.2 2 種類の高階表現	95
13.3 原子論理式を対象変数で参照する	98
13.4 述語を対象変数で参照する	99
13.5 述語をクラスで参照する	100
13.6 本章のまとめ	100
第 14 章 型推論	101
14.1 はじめに	101
14.2 宣言的記述の変換	102
14.2.1 定義と命題	102
14.2.2 頭部から体部へ: SPSHtoB	106
14.2.3 体部から頭部へ: SPSBtoH	107
14.3 型推論を適用する例	107
14.4 本章のまとめ	109
第 15 章 等価変換の例	110
15.1 はじめに	110
15.2 定数の階層関係	110
15.3 原子対象と基底対象	111
15.4 式変形の規則	112
15.5 一次方程式のための表現	113

15.6 等価変換の例	114
15.7 本章のまとめ	116
第16章 むすび	118
謝辞	118
参考文献	119

要旨

本研究の目的

本研究の目的は、階層関係と部分構造を持つ対象の表現と計算を定式化することである。ここで言う定式化とは、その対象を構成するための基礎となる構成要素を見つけ出し、これらを基礎となる理論の上で表現し、そして定式化の対象を構築し直すことである。本研究では、階層関係と部分構造を持つ対象の表現と計算を定式化の対象とした。

等価変換パラダイム

従来、問題を計算機によって解く場合には、人間が、その問題の解法を、アルゴリズムとして直接的に記述する方法がとられてきた。しかし、人間の直観に基礎を置く限り、アルゴリズムの品質を一定に保つのは難しい。アルゴリズムの品質には、正当性と効率という2つの側面がある。アルゴリズムは、その品質を保証する理論に基づいて、自動的に生成されるべきであるというのが、本研究の出発点である。

本研究では、問題の表現と計算を定式化するための枠組として、等価変換パラダイムを採用した。等価変換パラダイムは、問題をその解法と独立に表現するための系、および、表現された問題とその答を結び付ける写像の存在を仮定する。等価変換パラダイムは、この仮定の上で、計算の正当性を広く保証する。等価変換パラダイムにおいて、アルゴリズムとは、表現された問題に対する等価変換手続きとその適用を組にしたものである。等価変換パラダイムにおいて、問題の表現とその計算は、等価変換という概念だけで、非常に緩く結び付けられている。このため、正当性を保証できる利用可能なアルゴリズムの範囲は広い。正当性を保証する範囲が広いということは、その範囲の中で、効率の良い計算を発見できる可能性が高いということを意味する。

特殊化システム

等価変換パラダイムは、問題を表現するための系と、表現された問題とその答を結び付けるための写像の存在を、抽象的に仮定している。このため、問題が具体的に定められた場合には、その問題の中で扱う対象のために、具体的な表現系と写像を定める必要がある。本研究では、具体的な表現系として、特殊化システムを採用した。特殊化システムは、問題に現れる対象とその操作の組からなる。この対象を用いて確定節を構成し、確定節の集合を問題の表現とする。論理型パラダイムでは、述語とその引数である項からなる原子論理式を利用して確定節を構成する。これに対して、特殊化システムでは、確定節を構成する対象の形を具体的に固定せずに、その存在を抽象的に仮定する。この点で、論理型パラダイムにおける確定節と、特殊化システムにおける確定節は異なる。また、特殊化システムは、対象に対する操作も具体的に固定せず、ある条件の下でその存在を抽象的に仮定する。特殊化システムは、この仮定の上で、問題の表現とその答を結び付ける写像を提供する。

本研究の成果

本研究の重要な成果は以下の2つである。

1. 階層関係と部分構造を持つ対象のための表現系として、具体的な特殊化システムを考案したこと。

特殊化システムは、対象を表現するための表現系を、具体的に構築するための枠組である。特殊化システムそのものは、対象の具体的な形に依らず、抽象的に定義されている。対象のための表現系を具体的に示すためには、特殊化システムが抽象的に述べている部分を、具体化する必要がある。

2. 階層関係と部分構造を持つ対象を等価変換するために、単一化手続きを考案したこと。

等価変換とは、表現に対する変換であり、表現と結び付いた答を保存するものである。本論文では、基礎となる等価変換として Unfold 変換を採用した。Unfold 変換は、対象の単一化を内部で利用する。このために、本研究では、対

象の単一化手続きを考案し，また，この健全性と停止性を証明した．

表現

本研究では，階層関係と部分構造を持つ対象を，原子対象と呼ばれる形式により表現した．原子対象を表現するためには，定数の集合とその上の階層関係，および，変数の集合が必要である．本論文では，これらが与えられた場合に，どのような特殊化システムを作るべきかを述べた．本研究が目標としている対象のうちで，この原子対象だけでは表現できないような対象が存在する．このため新たに，クラス対象という形式を加え，原子対象を拡張した．

計算

特殊化システムは，原子対象への操作も含む．本研究では，等価変換の手続きとして Unfold 変換を採用した．このため，原子対象の単一化手続きが必要となる．本論文では，原子対象を単一化するために必要となる，原子対象への基本操作を具体的に定めた．そして，これを用いて単一化手続きを定め，その手続きが単一化であること (健全性)，および，有限の時間内に終了すること (停止性) を証明した．

まとめ

本研究は，階層関係と部分構造を持つ対象の表現と計算を，等価変換パラダイムの上に定式化した．そのために，本論文では，階層関係と部分構造を持つ対象のための表現系として特殊化システムを具体的に構築し，この上の表現に対する等価変換を示した．具体的に構築された表現と計算のための枠組を *LHS* と呼ぶ．*LHS* の応用として，高階表現の例，型推論の例，具体的な等価変換の例を示した．

第1章 はじめに

1.1 本研究の目的

本研究は，階層関係と部分構造を持つ対象の表現と計算を定式化することを目的とする．対象の表現と計算は，人工知能の基礎となる技術の1つである．世の中に存在する様々な問題領域においては，扱う対象を階層関係の中で捉えることにより，直観にかなった自然な理解が可能になる場合が多い．また，対象の間の階層関係だけではなく，対象自身の部分構造を表現することも必要である．

本論文で言う定式化とは，定式化する対象を，形式的に表現することである．形式的に表現するとは，対象を構成するための基礎となる構成要素を発見し，これらを一般的な表現系の上で表現し，これらを用いて定式化の対象を構築し直すことである．本研究では，階層関係と部分構造を持つ対象の表現と計算を定式化する．したがって，対象を構成するための基礎となる構成要素には，少なくとも，階層関係と部分構造がある．

1.2 本論文の概要

本研究では，問題を表現し計算するための枠組として，等価変換パラダイムを採用している．等価変換パラダイムでは，与えられた問題を宣言的に表現し，表現された記述を等価変換を利用して計算する．等価変換パラダイムは，問題を表現するための表現系を固定しない．また，計算のための等価変換手続きも固定しない．

等価変換パラダイムに基づいて，ある対象を定式化するためには，対象を表現するための表現系を具体的に示し，計算のための等価変換手続きを具体的に示す

必要がある。

本研究の重要な成果は以下の2つである。

1. 階層関係と部分構造を持つ対象のための具体的な表現系を考案したこと。
2. 階層関係と部分構造を持つ対象のための単一化手続きを考案したこと。

階層関係と部分構造を持つ対象のための具体的な表現系として、本研究では、特殊化システムを採用する。特殊化システムは、対象を表現するための表現系を、具体的に構築するための枠組である。特殊化システムそのものは、対象の具体的な形に依らず、抽象的に定義されている。対象のための表現系を具体的に示すためには、特殊化システムが抽象的に述べている部分を、具体化する必要がある。

等価変換とは、表現に対する変換であり、表現の意味を保存するものである。本論文では、基礎となる等価変換として Unfold 変換を採用している。このために、本研究では、対象の単一化手続きを考案する。

本論文の範囲を超えるが、実は、Unfold 変換は、等価変換として、効率良く計算を進めることができない場合があることがわかっている [12, 10]。Unfold 変換に代わる、効率の良い、等価変換規則とその適用の制御方法を発見することは、本研究に残された課題である。

1.3 本章のまとめ

結論として、本研究は、階層関係と部分構造を持つ対象の表現と計算を、特殊化システムと等価変換に基づいて、定式化する。特殊化システムと等価変換に基づく表現と計算のための枠組は、これまでに、計算機科学における様々な概念を定式化することに成功している。この枠組は計算機科学の分野における様々な概念を定式化するための大域的な座標系のようなものを与えると、期待している。本研究は、この期待を実証するための、1つの具体例としても捉えることができる。

第2章 動機

2.1 計算機を用いた問題の解決

2.1.1 プログラムとアルゴリズム

計算機を用いて問題を解決する場合には、アルゴリズムが必要である。アルゴリズムとは、計算の対象を操作するための手続きの集まりと、その手続きを問題に適用する方法の制御との組である。多くの場合、アルゴリズムは、人間が熟慮を重ねた結果として、考案される。考案されたアルゴリズムは、具体的にはプログラムとして記述される。プログラムを記述するためには、何らかのプログラミング言語を利用する。

例えば Fortran 言語は、数値計算を処理するアルゴリズムを書くためのプログラム言語である。Fortran 言語のプログラマは、数値計算を処理するアルゴリズムをプログラムとして記述する。このプログラムを計算機に与えると、これが実行され、答が求められる。

記号を処理するプログラムを書くためのプログラミング言語である Lisp の場合も、同様である。Lisp のプログラマは、記号を処理するアルゴリズムを、プログラムとして記述する。このプログラムを計算機に与えると、これが実行され、答が求められる。

以上をまとめると、プログラムとはアルゴリズムをプログラム言語を利用して書き下したものであるということ、および、アルゴリズムとは対象を操作する手続きの集まりと、その適用の制御との組である、ということが言える。

本論文の中では、プログラム言語を特に仮定せず、計算を抽象的に示す言葉としてアルゴリズムを使う。特定のプログラム言語を強く意識して計算を具体的に

示す場合には、プログラムという言葉を使う。

2.1.2 アルゴリズムの生成

2.1.1 節で述べたような、問題を解決するために、人間が、アルゴリズムを直接的に記述する方法には、問題がある。それは、アルゴリズムの品質を一定に保つことが難しいということである。

ソフトウェアが、問題解決のための一般的な道具となるためには、例えば工業製品のように、一定の理論のもとで、一定の品質のものとして考案されるべきである。つまり、アルゴリズムは、人間ではなく、理論により自動的に作られるべきものである。

その理由の1つは、アルゴリズムを人間の熟慮によって考案、または発見(例えば、[57, 56])する方法には限界があるからである。天才的な閃きを持つプログラマであれば、問題を効率良く解決できるアルゴリズムを考案できるかも知れない。しかし、ソフトウェアが、そのような、つまり音楽や美術のような、人間の不安定な感性に基づいて考案されるものであるとするならば、問題解決のための道具として、一般的に利用することは難しい。

ここでは、アルゴリズムの品質の1つの側面として、アルゴリズムの効率に注目した。アルゴリズムの品質には、もう1つ、正当性という重要な側面がある。

問題を解決するために、直接的にアルゴリズムを記述する場合には、そのアルゴリズムが正当であることを証明するのは難しい。例えば方程式を計算機を利用して解く場合を考える。この場合には、両辺に同じ値を加えるなど、その正当性が既に証明されている、いくつかの基本操作が既に存在する。あるアルゴリズムが、これらの基本操作のみを組み合わせで記述されている場合には、そのアルゴリズムも正当であることを証明するのは難しくない。しかし、正当性が証明されている基礎的な手続きが用意されていない対象(例えば、自然言語など)を扱うアルゴリズムの場合には、直感的にも理論的にも、そのアルゴリズムの正当性を証明するのは難しい。

以上をまとめると、次のようになる。

1. アルゴリズムの品質とは、効率と正当性である。
2. アルゴリズムを人間が直接的に記述する方法では、アルゴリズムの品質を保証するのは難しい。

これを受けて、本節では、次のように結論する；アルゴリズムは、その品質を保証する理論に基づき、自動的に生成されるべきである。

2.2 表現と計算と正当性

2.2.1 問題の表現

本研究では、ソフトウェアを利用して問題を解決する際には、問題の表現と、計算の2つを分離して扱う、という立場をとる。計算の正当性を保証するための既存の理論の多くも、問題の表現と計算を分離するという考え方を採用している(例えば, [18, 47])。

与えられる問題は、自然言語などで述べられ、あいまいさを含むことが多い。または、解きたい問題は、頭の中に漠然とだけ存在することもある。このような問題を、あいまいさを含むことなく、形式的に記述したものが、問題の表現である。問題を正しく解くためには、問題をあいまいさなく表現する必要がある。そもそも、あいまいな問題が与えられても、解法や導かれた解が正しいかどうかは、誰にも判断できない。

問題が具体的に記述されて初めて、その解法、すなわちアルゴリズムを考案することができる。したがって、問題の表現は、問題を解くためのアルゴリズムを考案する前に、これとは独立して記述される。

このように、本論文では、問題を解くために、まず問題をアルゴリズムとは独立に表現するという立場を取る。これは、従来のプログラミングパラダイムで取られてきた立場、すなわち、問題を解くために、直接的にプログラムを記述する

というものとは大きく異なる。

2.2.2 計算とその正当性

問題の表現 d と、問題の答 a とは、ある写像 f により対応付けられる。すなわち、

$$a = f(d) \quad (2.1)$$

が成り立つと仮定する。本研究では、式 (2.1) が成立する範囲のみを扱う。写像 f の定義域を D 、値域を R とする。

写像 f は D の全体に対して定義されている。問題の表現 $d \in D$ の形によっては、 $f(d)$ を求めるのが難しい場合や、時間が非常に長くなる場合もあり得る。このような場合には、 $f(d)$ を計算するのは、現実的ではない。

ここで、 D の部分集合 D' に対して、写像 $g: D' \rightarrow R$ が存在して、 $f(d') = g(d')$ が成り立つと仮定する。そして写像 g は、 $g(d')$ を計算するのが、 $f(d')$ を計算するよりも、易しいとする。この場合、問題の答を求めるためには、写像 f ではなく、写像 g を使うのが現実的である。

以上を仮定した場合、問題の表現 $d \in D$ が与えられた場合の解法とは、次のようになる。

1. $d \in D$ に対して、ある変換手続き r を適用し、 $d' \in D'$ を得る。
2. $a = g(d')$ により、問題の答 a を求める。

この解法が、正当な解法であるためには、

$$a = f(d) = f(d') = g(d')$$

を満たしていれば充分である。 $a = f(d)$ と $f(d') = g(d')$ は、仮定から常に成り立つ。したがって、上に述べた解法が正当であるための充分条件は $f(d) = f(d')$ である。この条件は、最初の問題から導かれる答 $a \in R$ は、変換手続き r を $d \in D$ に適用し $d' \in D'$ を得た後でも保存されているということを意味する。

手続き r のように, $f(d) = f(d')$ を満たす変換手続きを, 等価変換と言う. 一般的には, 変換手続きの1度の適用で $d \in D$ から $d' \in D'$ を求めるのは難しい. そこで, 表現 d に対して等価変換手続きを繰り返し適用し, 写像 g を使うことのできる領域 $D' \rightarrow R$ へ変換してゆく. 本論文における計算とは, 問題の表現に対して等価変換を繰り返し適用することである.

最初に与えられた問題の表現 $d \in D$ に対して,

$$d \xrightarrow{r_1} d_1 \xrightarrow{r_2} d_2 \cdots \xrightarrow{r_n} d_n \in D'$$

のように等価変換 r_i ($i = 1, \dots, n$) を順番に適用し, 最終的に, $d_n \in D'$ を得ることが, 問題を解くことである. $d_n \in D'$ が導かれたならば, $a = g(d_n)$ により, 答を導くことができる.

全ての操作 r_i ($i = 1, \dots, n$) は等価変換であるので,

$$a = f(d) = f(d_1) = f(d_2) = \cdots = f(d_n) = g(d_n)$$

が保証される. したがって, 問題の答が a であるとして, $a = f(d) = g(d_n)$ が成り立つ. これは, 解法の正当性を保証する. つまり, この等価変換の列の適用によって, 問題の表現 $d \in D$ から表現が $d_n \in D'$ と変化しても, 導かれる答 a は保存されることを保証する.

2.2.3 等価変換パラダイム

等価変換パラダイムは, 問題を表現するための表現系を, 具体的に固定していない. 等価変換パラダイムが仮定しているのは, 問題の答 a と問題の表現 d が, ある写像 f により $a = f(d)$ と対応付けられることだけである.

この写像 $f: D \rightarrow R$ について, 等価変換パラダイムは, D の部分集合 D' と, その上の写像 $g: D' \rightarrow R$ の存在, および, 元 $d' \in D'$ に対して, $f(d')$ を求めるよりも, $g(d')$ の方が易しいということを仮定している. そして, $d \in D$ に等価変換手続きを適用して $d' \in D'$ を求めることが, 等価変換パラダイムにおける計算である. これは, 問題と計算と答を, 非常に緩く結び付けている.

例えば、論理型パラダイムでは、問題を確定節で表現し、SLD 導出に基づいて計算を実行する。SLD 導出の場合には、節と原子論理式を選択する順番にしか自由度がない。したがって、論理型パラダイムでは、問題の表現と計算(アルゴリズム)は強く結び付いている。このことは、例えば、計算する方法として、SLD 導出よりも良い計算があったとしても、それを利用できないことを意味する。論理型パラダイムは、SLD に基づく計算に対しては、その正当性を保証できる。しかし、今述べたように、アルゴリズムに関して、正当性を保証できる範囲は狭い。

これに対して、等価変換パラダイムでは、問題の表現と計算は、等価変換という条件のみで結び付いている。このため、計算には任意の等価変換を利用でき、それゆえに、より良い計算を選び出せる可能性が高まる。その上、等価変換を利用している限り、その解法の正当性は保証される。このように、等価変換パラダイムは、アルゴリズムに関して、正当性を保証できる範囲が広い。

ところで、アルゴリズムの品質には、正当性の他に、効率という側面もあった。効率の良いアルゴリズムを自動的に生成するというのは、本研究を導く長期的な目標である。等価変換パラダイムは、広い範囲から、より良いアルゴリズムを選ぶ可能性がある。効率の良いアルゴリズムを自動的に生成するという研究は、現在、進行中である。

2.3 表現と計算のための座標系

2.3.1 概論

本研究で採用した表現と計算のための枠組である等価変換パラダイムは、問題を $d \in D$ で表現し、問題の答 $a = f(d)$ を等価に保ちながら、 $d \in D$ を $d' \in D'$ へ変換し、最終的に、問題の答 a を $a = g(d')$ によって得る、というものであった。この時点では、 $d \in D$ を記述するための具体的な表現系は固定されていない。

実際に問題を表現し、計算を進めるためには、表現系を具体的に定め、表現の形に対応した等価変換手続きとその制御の組(アルゴリズム)を利用する必要がある。

る。2.4節でも述べるが、本研究では、問題を表現するために、確定節の集合を利用する。この確定節は、論理型パラダイムという確定節と比べて、広い範囲のデータ構造を利用できるように、拡張したものである。

確定節で表現可能な問題の範囲は、等価変換パラダイム全てを覆うわけではない。すなわち、確定節で表現可能な問題の範囲は、等価変換パラダイムで扱える問題の範囲の一部分に過ぎない。しかし、本研究では、確定節の集合を問題の表現として採用する。なぜなら、確定節は人間が古くから、対象を宣言的に表現するために利用してきた表現系であり、實際上、有用な問題の範囲を多く含んでいるからである。

2.3.2 特殊化システムに基づく表現の定式化

2.6節で詳しく述べるが、本研究では、型に代表されるような階層関係を持つ対象の定式化を目標の1つとしている。階層関係を確定節の中に取り込むための、本研究が採用している以外の、代表的な理論については、3章で詳しく述べる。

本研究では、階層関係と部分構造を確定節の中に取り込むための基礎となる枠組として、特殊化システムを採用する。特殊化システムについては4章で詳しく述べる。本節では、特殊化システムについて、簡単に述べる。

特殊化システムは $\langle \mathcal{A}, G, S, \mu \rangle$ という4つ組の上に、問題を確定節の集合として表現する。 $\mathcal{A}$ は、表現されるべき対象の集合である。 S は、対象に対する操作の集合である。 $\mathcal{A}$ の元を利用して問題を表現し、 S の元を利用して、表現を等価変換し、答を導く。ここで、 μ は S の元を $\mathcal{A}$ の元に適用する方法(写像)である。 G は問題の答に直接的に関係する対象の集合である。

特殊化システムは、対象を表現する場合に $\mathcal{A}$, G , S , μ という4つの座標を持つ。これは、ある意味で、現象を把握する場合に、自然科学が従来から取ってきた方法を踏襲している。例えば、物理学に代表されるような自然科学の場合には、最初に時間、空間などの基本的な座標系が定められており、この上に、様々な物理現象を捉える。これに対して、従来のソフトウェアの理論は、ある概念を、例え

ば述語や項や型などの基礎的と思われる構成要素に分解し、これらを用いて、もとの概念を再構築するという方法をとってきた。そのため、捉えるべき概念毎に、基礎となる構成要素は異なり、全体を把握することは難しい。

特殊化システムは、そのような従来の手法を採用せず、基礎となる座標系を最初に定めて、その上に、対象となる概念を把握し表現する。この方法は、特殊化システムが表現できる概念の範囲が明確になるという点で、優れている。また、現在までの研究で、重要であると考えられている、ソフトウェア上で利用される、たくさんの概念を、特殊化システムの上に捉えることができることがわかっている。例えば論理型パラダイムは、等価変換パラダイムの具体例であると言える [12, 8, 7, 9, 11, 6, 5, 39, 40, 37, 35, 34, 33, 32, 31, 29, 30, 28, 63, 62, 59]。このことは、特殊化システムを適用可能な範囲が広いことを、実証的に証明している。

2.4 確定節を利用した表現

本節では、特殊化システムの上における確定節について述べる。

$H, B_1, \dots, B_n$ ($0 \leq n$) を $\mathcal{A}$ の元とする場合、

$$H \leftarrow B_1, \dots, B_n.$$

の形をした式を確定節と呼ぶ [23]。ここで H をこの確定節の頭部 (ヘッド) という。 $B_1, \dots, B_n$ をこの確定節の体部 (ボディ) と言う。確定節で $n = 0$ 、すなわち、体部が空であるものは、単位節と呼ばれることがある。

例えば、論理型パラダイムにおいては、個々の $\mathcal{A}$ の元は、 $p(t_1, \dots, t_n)$ の形をしている。ここで p は述語、 t_i ($i = 1, \dots, n$) は項と呼ばれる。 $p(t_1, \dots, t_n)$ 全体は、アトムと呼ばれる。本論文でも、この呼称をそのまま採用する。述語 p を持つアトムを、 p アトムと呼ぶ。頭部が p アトムである節を p 節と呼ぶ。 p 節を宣言するための確定節の集合を、 p プログラムと呼ぶことがある。

確定節は、対象を宣言的に記述する場合に用いられる。本論文では、問題を表現するものとして、確定節の集合を利用する。

確定節の集合を利用する計算のパラダイムに、論理型パラダイム [49, 50, 23] がある。論理型パラダイムでは、確定節の集合で問題を表現し、SLD 導出を計算の基礎とする。SLD 導出の適用の制御は、例えば論理型パラダイムの具体例である Prolog などでは、バックトラックやカットによって行われる。すなわち、確定節の集合を記述した時点で、計算のアルゴリズムは SLD 導出を利用をする狭い範囲に限定されてしまう。

本論文で採用している等価変換パラダイムでは、確定節の集合自身は、どんなアルゴリズムも決定しない。確定節の集合、すなわち、問題の表現には、ただ、答が対応するのみである。この点で、論理型パラダイムと、等価変換パラダイムは異なる。

また、確定節の集合を、問題の答に結び付ける方法においても、論理型パラダイムと、等価変換パラダイムでは異なる。論理型パラダイムでは、確定節の集合に対してモデルや解釈を用意し、論理的帰結を利用して、問題とその答を対応付ける。等価変換パラダイムの場合には、問題とその答を対応付けるために、モデルや解釈や論理的帰結の概念は必要ない。

2.5 等価変換に基づく計算の利点

アルゴリズムを構築するための従来のパラダイムは、計算の基本となる方法が狭い範囲に固定されており、これに基づいたアルゴリズムをプログラムとして記述するものであった。

例えば、手続き型パラダイム [19, 14] は、代入と制御構造(接続, 分岐, 反復)を計算の方法とし、これらがうまく働くように、問題を記述する。手続き型パラダイムの場合には、問題を解くために、アルゴリズムを直接記述する。したがって、問題の表現はもともとなく、アルゴリズムだけがある。

これに対して、論理型と関数型のパラダイムは、問題の表現を、その計算とは独立に記述することを目指した。しかし、問題の表現が十分に宣言的であるとは言い難い。

論理型パラダイム [49, 50, 23] では, SLD 導出を計算の方法とし, SLD 導出をうまく働かせるように, 確定節の集合をプログラムとして記述する. 関数型パラダイム [52, 15] では, 関数の適用 (またはラムダ計算) を計算の方法とし, 関数の適用がうまく働くように問題を記述する.

このように, どちらも問題の表現と計算が強く結び付いている. このため利用可能なアルゴリズムの範囲は狭い. したがって, 充分に良い計算を選ぶことが難しい場合が少なくない.

これらのパラダイムと比較して, 等価変換パラダイムでは, 問題の答を等価に保つような計算の方法, すなわち等価変換を, 広い範囲から利用可能である. この場合, いくつかの基本的な等価変換を最初に用意しておき, これらを表現に適用するための制御を合せたものがアルゴリズムとなる.

等価変換パラダイムでは, 計算の基本となる方法に対して, 等価変換であることのみを要請している. このため, 問題の表現に対応して, より良い計算を広い範囲から選択でき, 結果として良いアルゴリズムを導き易い. 例えば, 等価変換の向きを大域的な尺度として用意しておけば, 適用できる等価変換を自動的に選択可能になる可能性がある. これは, 問題の表現に対して, これを計算するためのアルゴリズムを, 自動的に導くことを可能にする.

2.6 定式化の対象の具体例

本論文は, 階層関係と部分構造を持つ知識の表現と計算を定式化することを目指とする.

階層関係と部分構造を持つ例としては, 図 2.1 に示すようなプログラムが挙げられる. これは, 従来の論理型パラダイムにおいて, 型付きのプログラムとして把握されているものである. 図 2.1 の記述は, 大きく 3 つの部分に分かれる. 1 つ目は, 1 ~ 5 行目であり, 型の階層構造の定義である. 2 つ目は, 6, 7 行目であり, 原子論理式や関数を取り得る引数に関して, 型の観点から正しい構造を定義したものである. 3 つ目は, 8 ~ 10 行目であり, 型の情報を含むプログラムである.


```
any → data | list;  
data → num | char;  
  
list ⇒ nil, cons;  
num ⇒ 0, 1, 2, ...;  
char ⇒ a, b, c, ...;  
  
append(list, list, list).  
cons(data, list).  
  
append(nil, X, X).  
  
append(cons(A:char, X), Y, cons(A, Z)) :-  
    append(X, Y, Z).
```

図 2.1: 型の付いたプログラムの例

型の階層関係の定義では、型にはany, data, num, char, listがあることが示されている。また、型の階層関係として、 $\text{any} \rightarrow \text{data}$, $\text{any} \rightarrow \text{list}$, $\text{data} \rightarrow \text{num}$, $\text{data} \rightarrow \text{char}$ があることが示されている。|を利用した記法は $\rightarrow$ の左辺が等しい階層関係を、まとめて記述するための、省略記法である。次にlist型にはnil, consという具体的なデータがあり、num型には0, 1, ...という具体的なデータがあり、char型にはa, b, ...という具体的なデータがあることが示されている。ここに至って、プログラムは、型とデータの2つの存在を扱うことが分かる。なお、consのように、引数を伴って具体的なデータを構成するものは、関数と呼ばれることがある。

原子論理式や関数の構造の定義では、appendは引数を3つ取り、それぞれ、順

番にlist型, list型, list型であることを示している. consは引数を2つ取り, それぞれ, 順番にdata型, list型であることを示している. 型には階層関係が定められており, 例えばlist型の引数が指定されている場所には, list, cons, nilを置くことができる.

プログラムの部分には, 確定節が2つある. 1つ目の`append(nil, X, X)`では, `append`という述語に引数`nil, X, X`が順番に与えられている. 論理型パラダイムは, 一般的に, `append`が位置する場所に現れる記号に対して, 述語という名前を与える. 述語は, 型やデータの階層関係に現れることがない. このことは, 論理型パラダイムにおいて, 述語は, 計算の対象とならないことを意味する. 次の確定節`append(cons(A:char, X), Y, cons(A, Z)) :- append(X, Y, Z).`では, `cons`が使われている. 論理型パラダイムは, 一般的に, `cons`のように, 述語ではないが, 引数を伴ってデータを構成することのできる記号に対して, 関数という名前を与える. `cons`の第1引数である`A:char`は, 変数`A`は`char`型の値のみを参照するというを示している.

2.7 本研究の成果と意義

図2.1に示したような, いわゆる型付きのプログラムに対する, 従来の定式化の例を3章で考察する. 図2.1には, 述語, 定数, 変数, 型, 関数, 項, 引数, 原子論理式というような概念が存在する.

本論文では, これらの概念を特殊化システムの上に把握し, $\langle \mathcal{A}, \mathcal{G}, S, \mu \rangle$ の要素として定式化する. この定式化により, 階層関係と部分構造を持つ対象を扱う問題を宣言的に表現し, 等価変換パラダイムの上で扱うことが可能になる. これは, 本研究の重要な成果の1つである.

本研究のもう1つの成果は, 階層関係と部分構造を持つ対象の単一化手続きを考案したことである. 本研究では, 計算の基礎となる等価変換として, `Unfold`変換を採用する. `Unfold`変換は其中で, 対象の単一化を利用する. したがって, `Unfold`変換を利用するためには, 対象の単一化手続きを, 具体的に示す必要がある.

3章に示すが、階層関係ト部分構造を持つ対象を扱う理論は、従来から存在している。本研究の意義は、このような対象を、等価変換パラダイムと特殊化システムという、より一般的な枠組の上で定式化したことにある。

従来の定式化のように、その対象を把握するためだけの定式化では、それ以外の対象を、同様の枠組の上で定式化するのは無理がある。したがって、枠組を拡張したり変更したりする必要がある。しかし、広い、つまり、大域的な視点のないままの拡張や変更は、それが正しい拡張や正しい変更であるかどうかを判断することは困難である。このような定式化は、本当の意味で、定式化と言えるかどうか、疑問がある。

本研究で言う定式化とは、対象を、等価変換パラダイムと特殊化システムの上に把握することである。対象をこれらの広い枠組の上で把握することが、本当の意味の定式化であると、本研究では考えている。

2.8 本章のまとめ

本研究では、階層関係と部分構造を持つ対象の表現と計算を、等価変換パラダイムと特殊化システムの上に定式化する。従来の定式化とは異なり、より一般的な枠組の上に対象を定式化することに、本研究の意義がある。

なお、等価変換パラダイムが計算機を利用して問題を解決する場合に、広い範囲で有効であることを実証する研究の1つとしても、本研究は位置付けられる。

第3章 関連する研究との比較

3.1 Typed Prolog

Typed Prolog [48] は、型に関する宣言と、型のないプログラミング言語である Prolog とを、合わせた言語である。

3.1.1 Typed Prolog の概要

例えば、図 2.1 に示したような、型の情報を持つ append 節のための表現は、Typed Prolog では以下の図 3.1 のように表わされる。

TYPE DECLARATIONS の部分には、Type constructors と、Predicate types to be reconstructed の2つの部分がある。Type constructors の部分では、表現の中で利用できる型と、その階層関係の宣言をしている。ここでは `list(T)` 型は `[](nil)` 型または `[](cons)` 型の親であることが宣言されている。ここで `list(T)` の `T` は、変数(型変数)であり、`list` は引数を取り、型を構成することが示されている。

Predicate types to be reconstructed の部分には、表現の中で使用する述語の引数に関し、型の宣言がなされる。今回はこれを省略している。

末尾の PREDICATE DEFINITIONS の部分には、Prolog の節の集合がある。この部分には型の情報は現れない。Typed Prolog では、このように、型に関する宣言の部分と確定節の集合の部分は独立している。

Typed Prolog は、Predicate types to be reconstructed の部分が完全に記述された入力に対しては、「型検査」(Type Checking) のアルゴリズムを適用し、引数の型の宣言と、確定節の集合の部分に実際に使用されている引数の型と


```
% TYPE DECLARATIONS

% Type constructors
type list(T) --> '[]' ; [ T | list(T) ].

% Predicate types to be reconstructed
% -----

% PREDICATE DEFINITIONS

append([], L, L).
append([X|L1], L2, [X|L3]) :- append(L1, L2, L3)..

% -----
```

図 3.1: Typed Prolog の append

の整合性を検査する。Predicate types to be reconstructed の部分がまったくないか、不完全に記述されている場合には、Typed Prolog は、「型推論」(Type Reconstruction) のアルゴリズムを入力に適用し、Predicate types to be reconstructed の部分の完全な記述を与える。

Typed Prolog は、型検査や型推論のアルゴリズムとして、関数型言語である ML [51] で採用されたアルゴリズムを、採用している。

図 3.1 では Predicate types to be reconstructed の部分の記述はない。したがって、この入力を Typed Prolog が受け取ると、型推論が行われ、

```
append(list(T), list(T), list(T))
```

が出力される。この出力は、Predicate types to be reconstructed の部分に記述されるべき、述語の引数の型を宣言したものである。

3.1.2 Typed Prolog の表現と計算の理論

Typed Prolog の確定節は, Prolog の確定節である. 確定節の部分には, 型の情報は現れない. プログラムの実行は, Prolog と同様の SLD 導出によって行われる.

表現と意味を基礎付ける理論として, Typed Prolog は, 通常の Prolog が採用している Herbrand Interpretation に対して型に関する拡張を施した, Typed Herbrand Interpretation を採用している. このように, 型を扱うために, 従来の理論に型を導入し, 拡張するという理論の構築方法は, 本研究の採用している方法とは異なる. 本研究では, 始めに特殊化システムという座標系を規定し, この上で, 扱うべき概念を把握する. Typed Prolog の方法を採用する場合には, 新しい概念が現れる度に, 基礎となる理論は拡張されるが, 拡張すべきかどうかの議論がそこにはない. 本研究の方法では, 扱うべき概念が, 特殊化システムとしてどう表現できるかを考えるので, 基礎となる理論は不変である. 特殊化システムでは, 絶対に表現できない概念が現れて初めて, 特殊化システムは拡張される.

特殊化システムについては, 4 章で詳しく述べる.

3.2 HiLog

HiLog [17, 64, 43] は, 論理型言語 Prolog で高階な表現を扱うために研究された言語である. 型を述語とみなすことにより, 型を引数として受け渡す表現が可能である.

3.2.1 HiLog の概要

図 2.1 に示した, 型の次いた append プログラムは, HiLog においては, 以下の図 3.2 のように表わされる.

HiLog の特徴は, Prolog の原子論理式に, 型変数を加えたことである. 図 3.2 の, 例えば 1 行目にある `list( $\tau$ )([])` の τ が, 型変数である. HiLog では, 型は `list( $\tau$ )([])` のように, 述語で表現される. このとき型変数は, 例えばこの


```

list( $\tau$ )([]).
list( $\tau$ )([X|L])  $\leftarrow$   $\tau$ (X), list( $\tau$ )(L).

append( $\tau$ )([], L, L)  $\leftarrow$  list( $\tau$ )(L).
append( $\tau$ )([X|L1], L2, [X|L3])  $\leftarrow$ 
     $\tau$ (X), append( $\tau$ )(L1, L2, L3).

```

図 3.2: HiLog の append

場合では、リストの要素の型を示す。このことは、2行目の

```
list( $\tau$ )([X|L])  $\leftarrow$   $\tau$ (X), list( $\tau$ )(L).
```

において、ボディの1つ目の原子論理式が $\tau(X)$ となっていることからわかる。

HiLog は、この型変数に関して、head-only type completion という制限を付け、構文としては高階に見える HiLog の意味論を、一階の意味論で構築することに成功している。この head-only type completion とは、体部にある型変数は、体部の原子論理式が利用される時点では、必ず、具体化されている必要があるという制限である。

図 3.2 のプログラムに対して、例えば

```
 $\leftarrow$  append(int)([0,1], [a], L)
```

のような問い合わせを入力することを考える。この場合、型変数 τ はintに具体化されるので、head-only type completion の条件は満足する。append 節が呼び出される度に、int(X) が呼び出され、リストの要素が、整数であるかどうか、検査される。これは Typed Prolog における型検査である。このとき、int 節は、プログラム中の別な部分に記述されていなければならない。

結局、この入力に対しては、最後に

```
append(int)([], [a], L)  $\leftarrow$  list(int)([a]).
```


の節が使われ、`list(int)([a])`の型検査に失敗し、問い合わせ全体でも失敗する。

3.2.2 HiLogの表現と計算の理論

HiLogは文献[17]で述べられているように、高階な表現を1階の意味論により基礎付けるという目標を持つ。図3.2にも示したが、型変数の部分を利用することにより、高階な表現が可能となる。HiLogは、プログラム中に現れる記号から、定数、関数、そして述語という区別を取り去ることにより、1階の意味論の構築に成功している。この考え方は、6章以降で述べるように、本研究でも採用している。

しかし、HiLogは、結局、解釈やモデルを基礎とする表現の理論を基礎に置いた。それ故に、HiLogの計算は、通常の述語計算(Predicate Calculus)に基礎を置いた。

結論として、HiLogは、表現の能力を1階述語論理に比べて向上させることに成功したが、計算に関しては、Prologと同程度のものを持つ。

3.3 λProlog

λProlog[53, 54, 55]は、一階述語論理の理論に基礎を置く論理型言語Prologと、関数型言語の基礎理論である型付きラムダ計算[26, 58, 60]を合わせた言語である。

3.3.1 λPrologの概要

図2.1に示したappendプログラムは、λPrologにおいては、図3.3のように表わされる。

λPrologでは、述語も、真理値を返す関数として、型が付く。図3.3の例では、appendには

```
(list A) -> (list A) -> (list A) -> o
```



```
kind int type.  
kind list type -> type.  
  
type nil (list A).  
type :: A -> (list A) -> (list A).  
type append  
      (list A) -> (list A) -> (list A) -> o.  
  
append nil L L.  
(append (X::L1) L2 (X::L3)) :- (append L1 L2 L3).
```

図 3.3: λProlog の append

という型が付けられている。この意味は (list A) 型の引数を 3 つ取り、真理値を返す、である。真理値は「o」型として表わされる。->は、関数をカーリー化 [27] し、全てを 1 引数の関数と単純化してみなすために利用している。A は型変数であり、list は、引数を 1 つ取ることにより、色々なリストのための型を表わすことができる。

λProlog における型は、型付きラムダ計算によって基礎付けられている。型付きラムダ計算には、サブタイプという、型の階層関係を扱う概念がある。しかし、サブタイプを扱う場合の問題も指摘されており [58]、λProlog でも型の階層関係表現しない。

例えば、nil は (list A) 型の、引数を持たない、関数として、型が付けられている。

3.3.2 λProlog の表現と計算の理論

λProlog は、一階述語論理に高階な表現を導入することを目的として考案された。

λ Prolog は、論理式の部分の表現と、関数の部分の表現の2つの表現を持つ。論理式の部分の表現は、一階述語論理に基礎を置く。関数の部分の表現は、型付きラムダ計算に基礎を置く。

計算は、基本的には確定節の SLD 導出により進む。関数が記述されている部分は、ラムダ計算(リダクション)により、計算が進められる。例えば、原子論理式を単一化する場合には、関数を全て正規形まで計算を進めておいてから、単一化の計算が進められる。

λ Prolog の考え方は、論理型言語では関数の計算をできないので、ラムダ計算を導入することにより、これを可能にするというものである。この方法を採用する場合には、ある概念を扱う必要が生じる度に、基礎となる理論を拡張する必要がある。この立場は、本研究の立場とは、全く逆である。本研究では、2.3 節でも述べたように、始めに基礎となる座標系を固定し、その上に、扱うべき概念を把握する。基礎となる座標系、すなわち特殊化システムの上で、その概念を把握することが、本研究で言うところの定式化である。 λ Prolog は、必要な概念を、形式的な記述体系の上で記述することを定式化としているように見える。この考え方は、本研究の立場とは相違する。

3.4 LOGIN

LOGIN [3]、またはその後継である LIFE [1] は、論理型言語において、階層関係を扱うために研究された。LIFE は LOGIN に関数型の計算を可能とする拡張を施した言語である。

3.4.1 LOGIN の概要

図 2.1 の append プログラムは、LOGIN においては、図 3.4 のように表わされる。なお、以下のプログラムは LIFE の実装である wild_life [2] の表記法を用いた記述である。


```

{ cons(tail=>list), nil } <| list.

append(one=>X:nil, two=>L:list, three=>L).
append(one=>cons(head=>A,
                    tail=>X:list),
        two=>Y:list,
        three=>cons(head=>A,
                    tail=>Z:list)) :-
append(one=>X, two=>Y, three=>Z).

```

図 3.4: LOGIN の append

LOGIN では、階層関係と部分構造を持つ対象を表現するために、 ψ -項という構造を考案した。 ψ -項は、一般的には以下のような形をしている:

$$X : \text{sort}(f_1 \Rightarrow \psi_1, \dots, f_n \Rightarrow \psi_n), 0 \leq n$$

ここで各 ψ_i ($i = 1, \dots, n$) は、再帰的に ψ -項である。 sort はソート (sort) と呼ばれる。ソートはプログラム中では `append` や `cons` が対応する。各 f_i ($i = 1, \dots, n$) は属性 (attribute) ラベル (LIFE ではフィーチャ) と呼ばれ、プログラム中では `one` や `tail` が対応する。属性ラベルは省略することができる。省略した場合には、自動的に整数 (1, 2, ...) が割り当てられる。

ψ -項の階層関係は、ソートの階層関係によって決まる。ソートの階層関係は、図 3.4 の 1 行目にあるように、`<|` によって宣言される。ソートの階層関係を定義する時点で、ソートが持つ属性ラベルに制限を付けることもできる。図 3.4 のプログラムでは、`list` の下にある `cons` は、`tail` という属性ラベルを持ち、その属性ラベルには `list` のソートを持つ ψ -項が付くことを示している。

3.4.2 LOGINの表現と計算の理論

LOGINは、階層関係と部分構造を表現するために、例えば λ Prologのように従来からある原子論理式を拡張するのではなく、 ψ -項という、独自の構造を考案した。LOGINにおける表現は、 ψ -項に基づいて原子論理式を構成し、確定節の集合として示される。

確定節の集合として表わされたプログラムは、Prologと同様にSLD導出によって計算が進む。したがって、計算の効率はSLD導出の効率に左右される。これは、より良い計算を発見し易い枠組とは言えない。

LOGINの ψ -項の考え方は、色々な研究に影響を与えている。例えば次3.5節で述べるTFSや、DOT [61]などは、LOGINの影響を受けている。DOTは ψ -項の持つ階層関係を表現する能力に着目した、表現の方法である。しかし、計算に関しては、文献 [61]の範囲外となっている。

3.5 Typed Feature Structure

Typed Feature Structure(TFS, 型付き素性構造) [16, 45, 46]は、複雑な階層関係を持つ知識を表現するために研究された。Typed Unification Grammar [20]のような、自然言語処理への応用がある。

3.5.1 TFSの概要

図3.5は、図2.1に示したappendプログラムをTFSの記法により表現したものである。実際には、TFSでは多くの場合、2次元的な表現を用いる。しかし、本論文では、他の言語との比較のし易さから、文献 [21]に示されている、1次元的な表記法を採用する。

LOGINが、全ての知識を ψ -項により表現しているのと同様に、TFSでは、全ての知識を型付き素性構造 (Typed Feature Structure, Typed FS, TFS) により表現する。

図3.5において、APPEND, ENDなどは型 (Type) と呼ばれる。図3.5では、型

$$\text{LIST} = \text{END} \vee \text{CONS}[\text{FIRST:WORD}, \text{REST:LIST}].$$

$$\begin{aligned} \text{APPEND} = & [\text{F:END}, \text{B:}\boxed{\text{L}}\text{LIST}, \text{w:}\boxed{\text{L}}] \vee \\ & [\text{F:CONS}[\text{FIRST:}\boxed{\text{X}}\text{REST:}\boxed{\text{L1}}], \\ & \text{B:}\boxed{\text{L2}}\text{LIST}, \\ & \text{w:CONS}[\text{FIRST:}\boxed{\text{X}}\text{REST:}\boxed{\text{L3}}]] \\ & :- \text{APPEND}[\text{F:}\boxed{\text{L1}}, \text{B:}\boxed{\text{L2}}, \text{w:}\boxed{\text{L3}}]. \end{aligned}$$

図 3.5: TFS のappend

の階層関係は1行目に示されている。

図 3.5 中に現れている, FIRST, REST, F, B, W などの記号は素性 (Feature) と呼ばれる。素性の後にある '[' と ']' でくくられた部分で, 対象の部分構造を表現する。TFS の素性は LOGIN の属性ラベルと同じ働きをし, 構造の中に, 再び型付き素性構造を含む場合に利用する。素性の後には $\boxed{\text{L}}$ などの, タグと呼ばれる記号が付くことがある。これは, 型付き素性構造を参照し, 同じタグの付いた型付き素性構造は, 同じものであることを示す。

3.5.2 TFS の表現と計算の理論

TFS は, 最初は, 確定節を表現することを目標としていたわけではなく, 階層関係やネットワーク化された知識を, 一般的に表現するために考案された。TFS を確定節に応用した場合には, 階層関係と部分構造を持つ対象を確定節で表現することができる。

TFS の表現のための理論は, 型付き素性構造である。型付き素性構造を確定節の記述, すなわち論理型プログラミングに応用した場合には, 確定節の表現と計算の理論は, 論理型パラダイムの理論にしたがう。

3.6 本章のまとめ

本章では、本研究と関連する、階層関係と部分構造を扱うための、従来の理論について、本研究との比較を混じえて、考察した。結論として、従来の多くの理論は、計算の方法をSLD導出のような、単一の方法に固定していると言える。このことは、より良い計算を導き出すという観点からは問題がある。本研究では、任意の等価変換を計算の方法として利用できる。したがって、より良い計算を導き出せる可能性を残している。

また、従来の理論の多くは、型とその階層関係を扱うにあたり、これらが今までの理論の中でどう把握されるべきなのかという視点を持たない。このように、何か今までとは異なる概念を扱う度に、その概念を新たな拡張として理論に組み込むという理論化の方法は、全体を統一的に把握するための視点を欠く。

本研究では、表現の基礎となる枠組として、等価変換パラダイムを採用している。何かある概念を扱う場合には、特殊化システムの上でそれを表現し、等価変換によりこれを計算する。扱いたい概念が変化しても、等価変換パラダイム自体は全く変化しない。等価変換パラダイムはたくさんの概念を統一的な枠組の中に定式化する、一般的な枠組である。

第4章 特殊化システムに基づく意味論

4.1 はじめに

特殊化システム [6] は，問題を表現する場合の基礎となる枠組である．特殊化システム上で記述された問題の表現は，その意味を対応付けられる．本章では，特殊化システムの定義と，その上で定義された表現の定義を一般的に示す．

4.2 特殊化システム

本節では，特殊化システムの定義を一般的に示す．

特殊化システムは，集合 $\mathcal{A}$, $\mathcal{G}$, S と，写像 $\mu: S \rightarrow \text{partial_map}(\mathcal{A})$ からなる4つ組 $\langle \mathcal{A}, \mathcal{G}, S, \mu \rangle$ で，定義 1 に示す条件を満足するものである．

ここで $\text{partial_map}(\mathcal{A})$ は $\mathcal{A}$ 上の部分写像の集合を意味する．

定義 1 (特殊化システムの条件)

1. $\forall s_1, s_2, \exists s \in S: \mu(s) = \mu(s_1) \circ \mu(s_2)$
2. $\exists s \in S, \forall a \in \mathcal{A}: \mu(s)(a) = a$
3. $\mathcal{G} \subset \mathcal{A}$

□

ここで， $\mu(s_1) \circ \mu(s_2)$ は $\mu(s_1)$ と $\mu(s_2)$ の合成関数を意味する．

集合 $\mathcal{A}$ の元を原子対象という．集合 $\mathcal{G}$ の元を基底対象という．集合 S の元を特殊化という．条件 (2) で示されている特殊化 s を単位特殊化という．

特殊化システムは，このように，具体的な表現と計算の対象を想定せず，一般的に定義されている．このことは，特殊化システムが広い範囲の対象を定式化できる可能性を持つことを示している．

4.3 宣言的記述と意味

4.3.1 宣言的記述

等価変換パラダイムが要請している，問題の宣言的な表現を，宣言的記述と呼ぶ．特殊化システムにおける宣言的記述は，以下のように定義される．

定義 2 (宣言的記述) 特殊化システム Γ を $\langle \mathcal{A}, G, S, \mu \rangle$ とする． Γ 上の宣言的記述を以下のように定義する．

1. Γ 上の確定節とは， $H, B_1, \dots, B_n \in \mathcal{A}$ ($0 \leq n$) から構成される

$$H \leftarrow B_1, \dots, B_n.$$

の形をした式である． H を確定節の頭部 (ヘッド)， $B_1, \dots, B_n$ を確定節の体部 (ボディ) とよぶ．

なお，確定節の末尾にある「 $.$ 」は，文脈に応じて，付ける場合と付けない場合がある．

2. Γ 上の宣言的記述とは， Γ 上の確定節の任意の集合である．

□

以降では，あいまいさを引き起す恐れのない場合には，確定節を単に節と呼ぶ．特殊化構造 Γ 上の宣言的記述全体の集合を $\mathcal{D}(\Gamma)$ と表わす．以下に，節に対する，いくつかの記法を示す．

定義 3 (確定節に対する記法) 特殊化システム $\Gamma = \langle \mathcal{A}, G, S, \mu \rangle$ が与えられていると仮定する．

1. Γ 上の節 C に対し, $head(C)$ で C の頭部を, $body(C)$ で C の体部に現れる原子対象全体の集合を表現する.
2. $d \in \mathcal{D}(\Gamma)$ に対し, $Gclause(d)$ で d に属する節の基礎例全体の集合を表現する.
 ここで, 節 $H \leftarrow B_1, \dots, B_n$ の基礎例とは, $H', B'_1, \dots, B'_n \in \mathcal{G}$ から作られる節 $H' \leftarrow B'_1, \dots, B'_n$ であり, かつ $\exists \theta \in \mathcal{S}, H' = H\theta, B'_1 = B_1\theta, \dots, B'_n = B_n\theta$ を満たすものをいう. なお, ある節からその節の基礎例を得ることを, その節の基礎化という.

□

4.3.2 宣言的記述の意味

等価変換パラダイムの一般的な考え方は 2.2.2 に述べたように, 問題の表現 d と問題の答 a を写像 f によって $a = f(d)$ と対応付けるというものであった. この時点では, d のための具体的な表現系は定められていない.

本研究では, 具体的な表現系として特殊化システムを採用した. 特殊化システムは, 問題を確定節の集合として表現する. これを宣言的記述と呼ぶ. 特殊化システムでは, 宣言的記述と問題の答を結び付けるための写像 f として,

$$f = \pi \circ \mathcal{M}$$

を採用する. すなわち, 表現と答を対応付ける写像 f を, π と $\mathcal{M}$ の2つの写像に分ける.

$\mathcal{M}$ は, 問題の具体的な形に依らず, $\langle \mathcal{A}, \mathcal{G}, \mathcal{S}, \mu \rangle$ の上であらかじめ定義される. π は, 任意の写像である.

定義 4 (宣言的記述の意味) 特殊化システム $\Gamma = \langle \mathcal{A}, \mathcal{G}, \mathcal{S}, \mu \rangle$ が与えられているとする. $d \in \mathcal{D}(\Gamma)$ に対する意味 $\mathcal{M}(d)$ を

$$\mathcal{M}(d) = \bigcup_{n=0}^{\infty} T_d^n(\phi)$$

と定義する. なお,

$$\begin{aligned} T_d: 2\mathcal{G} &\rightarrow 2\mathcal{G} \\ T_d(x) &= \{head(C) \mid body(C) \subset x, C \in Gclause(d)\} \\ T_d^1(x) &= T_d(x) \\ T_d^n(x) &= T_d^{n-1}(T_d(x)), \quad (2 \leq n) \end{aligned}$$

である. □

以上が特殊化システム上の表現とその意味の一般的な定義である.

4.4 特殊化システムの意義

従来, あるソフトウェア上の概念を定式化する場合には, その概念がどういう部品から構成されているのかをまず研究する. そして, その部品の組み合わせとして, 定式化したいソフトウェア上の概念を再構築する.

この方法に依れば, 定式化したい概念毎に, 部品は存在することになる. 複数の概念を定式化する場合には, それぞれの概念を部品化し, 再構築することに成功しても, お互いの関係を定式化することはできなかった.

従来のソフトウェアの分野では, 個々の概念を定式化した時点で, その概念を把握することに成功したと考えられてきた. しかし, 実際には, 科学の様々な分野, 例えば物理学に代表されるような自然科学の分野では, かつてはそうであったかも知れないが, 現在は, そのような方法は取られていない. これらの分野では, 世界を記述するための統一的な座標系をあらかじめ定め, 個々の現象は, この座標系の上で表現され, 把握される. この座標系の上で把握できない現象が発見されて初めて, その座標系を変更する. 従来のソフトウェアの分野には, この統一的な座標系が欠如している.

特殊化システムは, ソフトウェアにおける, 座標系のようなものを与えるシステムとなり得る可能性がある. 現在, 様々なソフトウェア上における概念が特殊化システムの上で再構築されている. このことは, 特殊化システムが, ソフトウェ

アの分野において、ソフトウェア一般を表現するための座標系となり得ることを実証的に証明している。

本研究は、クラスと部分構造を対象領域とする知識を、特殊化システムの上で定式化する。これは、従来から提案されてきた個々の定式化とは異なり、全体を統一的に把握する可能性を持つ、一般的な座標系の上での定式化である。本研究はこの点で価値がある。

4.5 本章のまとめ

等価変換を利用して計算を進めるという考え方自体は、問題を表現するための表現系を具体的に定めることなく、理論化できる。しかし、具体的に問題を表現し、これを等価変換により計算するためには、問題の表現、および、これと答を結び付ける写像を具体的に定める必要がある。

本研究では、階層関係と部分構造を持つ対象を扱う問題を、具体的に表現することを目標とする。このための具体的な表現系として、特殊化システムを採用する。

特殊化システムは、問題を表現し、表現をその意味と対応させるための具体的な表現系である。これは、4つ組 $\langle \mathcal{A}, \mathcal{G}, \mathcal{S}, \mu \rangle$ の存在を仮定することにより、その上における確定節を定める。問題の表現 d は確定節の集合であり、これは、その意味 $\mathcal{M}(d)$ と対応する。さらに、問題に応じて写像 π を定め、問題の表現 d とその答 a とは

$$a = \pi(\mathcal{M}(d))$$

により対応付けられる。

本研究では、階層関係と部分構造を持つ対象のための特殊化システム $\langle \mathcal{A}, \mathcal{G}, \mathcal{S}, \mu \rangle$ を、具体的に定める。これが、本研究で言うところの、定式化である。

第5章 等価変換に基づく計算

5.1 等価変換

等価変換の理論は、具体的な表現系を仮定せずに、構築できる。その内容は2.2.2節に述べた。繰り返すと、等価変換とは、問題の宣言的記述 d が与えられた場合に、その意味 $\mathcal{M}(d)$ を保存する、 d に対する変換操作のことである。本研究では、全ての計算を等価変換の組み合わせとして表現する。この方法の利点は、問題の解法の正当性を保証できることと、より良い計算を発見する可能性が高いということである。

なお、具体的な計算を進めるためには、表現系と共に等価変換も具体的に定めなければならない。次節では、本論文で採用した具体的な等価変換として、Unfold変換について述べる。

5.2 Unfold 変換

具体的な等価変換として、本論文では Unfold(展開)変換 [42] を採用する。Unfold変換は、論理型プログラミング言語の研究から生まれた。論理型プログラミング言語では、通常は SLD 導出を利用して計算を進める。しかし、計算を効率的に進められないような場合に、Unfold 変換と、これと対をなす、Fold(畳み込み)変換とを利用することによって、効率の良い計算ができる場合があることが発見された。

本節では、文献 [42](淵など, 1987) に示してある Unfold 変換手続きの定義とその例を示す。Fold 変換は、これ単独では等価変換ではないので、本論文では触れない。Unfold 変換が等価変換であることを示すのは、本論文の範囲外である。Unfold 変換は、原子論理式の単一化を利用する。本研究で扱う対象に対する具体的な単

一化アルゴリズムは 12.4 節に示す。

5.2.1 Unfold 変換の例

本節では、文献 [42] で述べられている、具体的な Unfold 変換の例を示す。この例では、プログラムを表現するために、確定節の集合を用いている。

始めに、図 5.1 に示す確定節の集合、すなわち、プログラムが与えられている。それぞれの節には C_1 から C_4 まで、番号が付けられている。

```
C1: subseq([], X) ← .  
C2: subseq([A|X], [A|Y]) ← subseq(X, Y).  
C3: subseq([B|X], [A|Y]) ← subseq([B|X], Y).  
C4: common_subseq(X, Y, Z)  
      ← subseq(X, Y), subseq(X, Z).
```

図 5.1: 文献 [42] の例題

このプログラムは、共通な部分列 (common_subseq) を求めるためのものである。このプログラムに対して、例えば

```
common_subseq([a,c], [b,a,c], [a,b,c,d])
```

という問い合わせを与えると、Prolog の処理系は SLD 導出を繰り返して、最後に「yes」を返す。しかし、この導出の途中では、無駄な導出がたくさんなされるので、これを省くために Fold/Unfold 変換を利用する。

Unfold 変換は、直感的に言うと、ボディに現れる原子論理式を、これと単一化可能なヘッドを持つ節のボディで置き換える。

例えば C_4 節のボディの 1 番目に現れる原子論理式 $\text{subseq}(X, Y)$ を Unfold 変換により、置き換えることを考える。これには C_1, C_2, C_3 の 3 つの節を適用できる。したがって、Unfold 変換によって、 C_4 は以下の C_5, C_6, C_7 で置き換えられる。

$C_5: \text{common_subseq}([], Y, Z) \leftarrow \text{subseq}([], Z).$

$C_6: \text{common_subseq}([A|X], [A|Y], Z)$
 $\leftarrow \text{subseq}(X, Y), \text{subseq}([A|X], Z).$

$C_7: \text{common_subseq}([B|X], [A|Y], Z)$
 $\leftarrow \text{subseq}([B|X], Y), \text{subseq}([B|X], Z).$

これ以上の説明は、本論文の範囲を越えるので省略するが、この Unfold 変換により得られたプログラムは、等価に変換されている。

5.2.2 Unfold 変換の形式的な定義

手続き 1 に示す手続きは、文献 [42] からの引用である。

手続き 1 (Unfold 変換) $C \in P$ を $A \leftarrow B, L$ の形の節とする。 $C' \in P$ を $B' \leftarrow L'$ の形の節とする。 B と B' が単一化可能で、その最汎単一化子 (mgu, the most general unifier) を θ とする。

このとき、節 $A\theta \leftarrow L'\theta, L\theta$ を、「節 C の原子論理式 B に、節 C' を適用した結果」と呼ぶ。節の適用を考えると、2つの節は変数を共有しないよう、必要ならば、変数の名前替えを前もって行うものとする。

確定節の集合 P が与えられている。 P 中の任意の確定節 C と、その右辺の原子論理式 A を選び、 A と単一化可能な左辺 (頭部) を持つ P 中の全ての節を $C_1, \dots, C_k$ とし、各 C_j ($j = 1, \dots, k$) を C の A に適用した結果を C'_j とする。このとき、 P の C を $C'_1, \dots, C'_k$ で置き換えたものを P' とする。

P を P' に変換する操作を、Unfold(展開) 変換と言う。 □

このように、Unfold 変換は、その中で、原子論理式の単一化を利用している。したがって、本論文では、階層関係と部分構造を持つ対象のための具体的な単一化手続きを示す必要がある。対象の単一化手続きは、直観的な説明を 12.3 節で述べ、形式的な定義を 12.4 節で示す。

5.3 本章のまとめ

等価変換を利用して計算を進めるという考え方自体は、問題を変換するための表現系を具体的に固定することなく、抽象的に理論化できる。4章において、問題を変換するための表現系を、特殊化システムに固定した。すると、この上の表現に対する意味が対応付けられ、この意味を固定する、等価変換を具体的に定めることができる。

本研究では、等価変換として Unfold 変換を採用する。Unfold 変換が利用する、対象の単一化手続きについては、10章と12章で述べる。

第6章 階層関係の表現

6.1 はじめに

本論文で言うところの定式化とは，階層関係と部分構造を持つ対象のための特殊化システムを具体的に定めることである．特殊化システムを具体的に定めるということは，その上における，問題の表現が定まるということである．問題の表現から，その意味が定まる．表現の意味が定まると，問題の表現に対する変換の操作が，等価変換かどうかを判定できる．本研究では，等価変換を計算の方法として採用している．したがって，問題の意味は，問題を解くために利用可能な計算の範囲を定める．このように，特殊化システムを具体的に決めると，全てが具体的に定まる．

本章以降では，階層関係と部分構造を持つ対象のための特殊化システムを4章にしたがい，具体的に定めてゆく．なお，本論文で構築する具体的な特殊化システムに基く，階層関係と部分構造を持つ対象の表現と計算のための枠組を *LHS* [36] (*Land for Hierarchy and Substructure*) と呼ぶ．

6.2 クラスとインスタンスの統合

世の中に存在する様々な問題領域においては，扱う対象を階層関係の中で捉えることにより，直観にかなった自然な理解が可能になる場合が多い．それを知識表現に生かすために，例えば，論理型言語の *LOGIN* [3] や *PAL* [4] などでは，*Prolog* の拡張として，クラス階層を扱う機構を導入している．

例として図 6.1 に示すクラス階層を考える．なお，以降ではクラスの階層はすべて木構造をなしていることを仮定する．図 6.1 において，*human*, *male*, *female*

は「クラス」と呼ばれ, socrates, adam, eve は「インスタンス」と呼ばれる。クラスはインスタンスの集合に対応するものとみなすことができる。図 6.1 ではクラス同士の関係を示すために記号 $\rightarrow$ を用い, クラスとインスタンスの関係を示すために記号 $\Rightarrow$ を用い, 2つの関係を区別している。

```
human  →  male | female
male   ⇒  socrates ; adam
female ⇒  eve
```

図 6.1: クラス階層の例

階層関係を基礎とした対象の表現について考える。例えば変数 X が『男』や『人』というクラスに属する計算対象を表わすことを, LOGIN では $X:\text{male}$, $X:\text{human}$ のように, 変数にクラスを付けて表記する。クラス male はクラス human よりも狭い範囲を表わす集合であり, したがって $X:\text{male}$ は $X:\text{human}$ より詳細に X を表現している。対象が具体的なインスタンスの場合には, 変数を付けず, インスタンスのみで対象を表わす。

この表記を使えば, 図 6.1 のクラス階層のもとでの「人間 $\rightarrow$ 女 $\rightarrow$ イブ」という対象の詳細化は, 以下の式 (6.1) のようになる。

$$X:\text{human} \rightarrow X:\text{female} \rightarrow \text{eve} \quad (6.1)$$

式 (6.1) では クラスとインスタンスという 2 種類の概念を用いて対象をとらえていた。そこでは, eve はインスタンスとされ, female などのクラスと区別されている。これはクラスをインスタンスからなる集合とするモデルに基づく。しかし, 例えば, インスタンスをそれ自身だけからなる単元集合に対応させれば, インスタンスもクラスのように集合として扱うことが可能である。

また, 次の観点からすれば, 集合の基本要素としてのインスタンスという概念を持ち出す必然性は必ずしも高くはないと考えられる。たとえば, 「イブ」という

名前の女性は実際には世界中に何人も存在し、「楽園に住むイブ」や「18歳のイブ」、さらには「楽園に住む18歳のイブ」、…など、いくらでも情報を詳細化することが可能である。したがってインスタンスの概念を用いないモデル化は、現実世界をより忠実に表現しようとする観点から見れば、有力な案と考えられる。

これらの観察に基づいて、本論文では、階層に現れるクラスとインスタンスの区別をなくし、改めてすべてクラスとして扱う。この結果、図 6.1 に示した階層の定義は、クラスとインスタンスの区別がなくなるので、図 6.2 のように改められる。また、すべての計算対象を変数とクラスの組で表記する。したがって式 (6.1) の記述は式 (6.2) のように改められる。

$$X:\text{human} \rightarrow X:\text{female} \rightarrow X:\text{eve} \quad (6.2)$$

human	→	male female
male	→	socrates adam
female	→	eve

図 6.2: 全てをクラスとしたクラス階層

従来は当然と考えられていた概念を、特殊化システムの上で表現するにあたり、本研究では、このように、本質的に必要な概念は何かを、検討する。今回は、インスタンスという概念が無くとも、知識の表現に影響は無いと考え、インスタンスとクラスを統合し、両者を改めてクラスと呼ぶことにした。

6.3 列で表現されるクラス

6.2 節では、クラスは、定数により表現されていた。

ある人が結婚して息子を一人持ったとする。『第一子が息子である人』を表わすためには human-having-a-son のような新たなクラスを作成する必要がある。

しかしこれを進めて行くと、『二人の息子を持つ人』『第一子に息子，第二子に娘を持つ人』など様々な組合せ一つ一つに対してクラス名を導入する必要がある。

このような組合せによるクラス名の増加を抑制するために，クラスに構造を持たせることを考える。図 6.2 のクラス階層を引き続き仮定する。例えば『第一子が息子である人』というクラスは (human male) のように書く。これは，列の第一要素 human により人であることを，第二要素 male により第一子は男であることを表現している。同様に，『第一子が息子で第二子が娘である人』というクラスは (human male female) となる。

以降ではこのようにクラスに列の構造を与える。列を構成する要素の個数を「クラスの長さ」と呼ぶ。対象は，クラスの新しい定義を用いて，変数とクラスの組で表現する。たとえば， $x: (\text{human male})$ がその例である。

クラスと対象の新しい定義を以下にまとめる。

- 今までクラスと呼んでいた human, male などの名前は以降では「定数」と呼ぶ。
- 定数には木構造の階層関係が与えられる。
- クラスは定数の 0 個以上の列である。
- 対象は，変数とクラスの組である。

6.4 列で表現されるクラスの階層関係

次に，列で表現されたクラス間の階層関係を決定することを考える。図 6.2 に示したような定数の階層関係があらかじめ与えられているものとする。

前に示した『第一子が息子である人』というクラス (human male) に加えて，“第二子が娘である”という情報を持つクラスは (human male female) と表現される。クラス (human male female) はクラス (human male) よりも狭い範囲を表わしており，クラス階層としては下に位置する。このように列で表現されたクラスの階層関係は，定数の階層関係から自然に導かれる。

図 6.2 に示した定数の階層関係のもとで、クラスの階層関係の例を図 6.3 に示す。 $\succeq$ は右辺の方がより狭い (具体的な) クラスであり、クラス階層の下位 (子孫) に位置することを示す。 $??$ は上下関係を決定できないことを示す。

$()$	$\succeq$	(human)
(female)	$\succeq$	(eve)
(eve)	$\succeq$	(eve human)
(eve human)	$\succeq$	(eve female)
(female human)	$??$	(human female)
(human female)	$??$	(eve)

図 6.3: 列によるクラス階層関係の例

6.5 本章のまとめ

本章では、クラスを直感的に定めた。図 2.1 に例示したように、本研究で扱うべき対象の 1 つに「型」があった。本研究では、型よりも広い範囲の概念と対応する「クラス」という言葉を採用した。

従来の多くの理論ではクラスは、特殊化され、最後にはインスタンスに変化する。しかし、本研究では、インスタンスの概念はなくても問題はないと考え、クラスとインスタンスを統合し、改めてクラスと呼ぶことにする。

第7章 部分構造の表現

7.1 部分構造

これまでは、対象を変数とクラスの組で表現してきたが、それだけでは対象の部分構造を表現することができない。本節では、対象の部分構造を表現することを考える。

部分構造を持つ対象の例として図 7.1 に示す BNF で定義される『数式』を取りあげる。この BNF は 2 種類のことを定義している。図 7.1 の 4 行目以外の式は階層関係を定義している。しかし、4 行目の “c-expr ::= expr opr expr” という記述は他と異なり c-expr (複合式, complex expression の意味) の部分構造を定義していると考えられる。すなわちこの記述により、c-expr は数式、オペレータ、数式がこの順番で並んだ部分構造を持つということを定義している。

```

expr  ::=  num | var | c-expr
num   ::=  0 | 1 | 2 | ...
var   ::=  A | B | C | ...
c-expr ::=  expr opr expr
opr   ::=  + | - | × | ÷

```

図 7.1: BNF による式の定義

このようにクラスに加えて、部分構造を持つような計算対象を表現する方法を考える。例えば “1+3” という数式は (c-expr) クラスであり、部分構造として +,

1, 3 をこの順序で持つ。+ は (+) クラスであり部分構造を持たない。1 や 3 も同様である。この式を計算対象として以下の式 (7.1) のように表現する。

$$X: [(c\text{-}expr) \ E1: [(1)] \ R: [(+)] \ E2: [(3)]] \quad (7.1)$$

一般に部分構造を持つ計算対象を表現するために、クラスと部分構造の組を “[” と “] ” でくくり、前に変数と “ : ” を付ける。この変数を「対象変数」と呼ぶ。また、+ や 1 などのように部分構造を持たない計算対象も $D: [(+)]$, $E: [(1)]$ のように表現し、表記を統一する。

以上の例では長さ 1 のクラスのみしか出現していないが、長さが 2 以上のクラスももちろん可能である。例えば対象 $D: [(+ \text{ num})]$ は「整数の加算演算子」である D を意味する。これを式 (7.1) の $D: [(+)]$ と置き換えれば、長さが 2 以上のクラスの対象を部分構造として持つ式ができる。

もう少し複雑な数式として “ $1+3 \div a$ ” を考える。この式は原子対象として式 (7.2) のように表現できる。

$$\begin{aligned} Y: [(c\text{-}expr) \\ X: [(c\text{-}expr) \ E1: [(1)] \ R: [(+)] \ E2: [(3)]] \\ Q: [(\div)] \\ V: [(a)]] \end{aligned} \quad (7.2)$$

なお図 7.1 に示した BNF に依る式の定義では、演算子の優先順位について、考慮していない。したがって、式 (7.2) に示した原子対象も、演算子の優先順位を考慮しない表現になっている。

7.2 省略記法

原子対象の記述を簡単にするために、以降では 1 つの計算対象の中で、同一の対象変数には同一のクラスと部分構造が付くことを仮定する。この仮定のもとで、「1 つの計算対象の中で、同一の対象変数の 2 回目以降の出現にはクラスと部分構造を付けなくてもよい」という省略記法を用いることにする。

例えば以下のような計算対象を考える。

$$X: [(c\text{-}expr) \ E: [(num)] \ R: [(*)] \ E: [(num)]]$$

これは次のような省略記法で表わすことができる。

$$X: [(c\text{-}expr) \ E: [(num)] \ D: [(*)] \ E]$$

またさらに、特に必要のない場合は対象変数を示さないことがある。このとき、上式は更に簡略化されて以下のように表わされる。

$$[(c\text{-}expr) \ E: [(num)] \ [(*)] \ E]$$

7.3 原子対象: $\mathcal{A}$

これまで述べて来たように、階層関係と部分構造を持つ知識は、原子対象によって表現できる。これまでの説明を総合すると、原子対象とは次のようなものである。

- 原子対象はクラスを持つ。クラスは、原子対象どうしの階層関係を定める。
- 原子対象は部分構造を持つ。部分構造とは、原子対象の0個以上の列である。部分構造に含まれている原子対象の個数を、部分構造の長さと言う。部分構造に含まれている原子対象を、元の原子対象の部分対象と言う。部分対象が再び長さ0以上の部分構造を持つ場合もあり、これは再帰的に繰り返される。
- 原子対象は変数を持つ。同じ変数が付いた原子対象は、同じクラスと同じ部分構造を持つ。

実は、異なる原子対象に同じ変数を付けても、理論的には問題ない。そのような原子対象を含む確定節は、計算が失敗するので、単に、計算の途中で消えてしまうだけである。しかし、本論文では、簡単のために、同じ変数が付いた原子対象は、同じクラスと同じ部分構造を持つ、という性質を仮定して論を進める。

7.4 本章のまとめ

本章では，クラス階層と部分構造を対象領域とする知識を，原子対象として表現する方法を直感的に示した．原子対象の形式的な定義は，11章に示す．

第8章 構造制約の導入

8.1 構造制約

図 7.1 に示した BNF から, (c-expr) クラスの原子対象の部分構造は, それぞれ (expr), (op), (expr) クラスの原子対象であるという制約を持つことがわかる. 例えば, (c-expr) クラスの数式を表わす原子対象

$$X: [(c\text{-}expr) \ E1: [(1)] \ R: [(+)] \ E2: [(3)]]$$

は, クラス (1), クラス (+), (3) の部分構造を持つ. それぞれのクラスは (op), (expr), (expr) のサブクラスであり, その制約を満足している.

本節では, 原子対象が持つ部分構造に関するこのような制約について考える. この制約はクラスごとに与えられるものとする. この制約を「構造制約」と呼ぶ.

「(c-expr) クラスの原子対象は (op), (expr), (expr) クラスの原子対象をこの順序で部分構造として持つ」という構造制約を式 (8.1) のように表現する.

$$(c\text{-}expr) == (expr) \oplus (op) \oplus (expr) \quad (8.1)$$

== の右辺の並びは, 順に, 被演算項 1, 演算子, 被演算項 2 を部分構造として書くことを示している. これは BNF の記法である $c\text{-}expr ::= expr \ op \ expr$ にはほぼ対応しているが, BNF では実際の式 (“1+2” など) の文字 (1, +, 2 など) の並ぶ順序までを表現しているのに対し, 上記の構造制約の順序は原子対象の記述順は指定しているが, 実際の式の記述自体には言及していない.

次に, より複雑な構造制約の表現方法を導入する. 『整数を要素とする長さ 1 以上のリスト』のクラスの原子対象は, 以下のような制約を持つと考えられる.

- car 部と cdr 部をこの順序で持つ.

- car 部の原子対象は「整数」である.
- cdr 部の原子対象は「整数を要素とするリスト」である.

図 8.1 に示す定数の階層関係を考える.

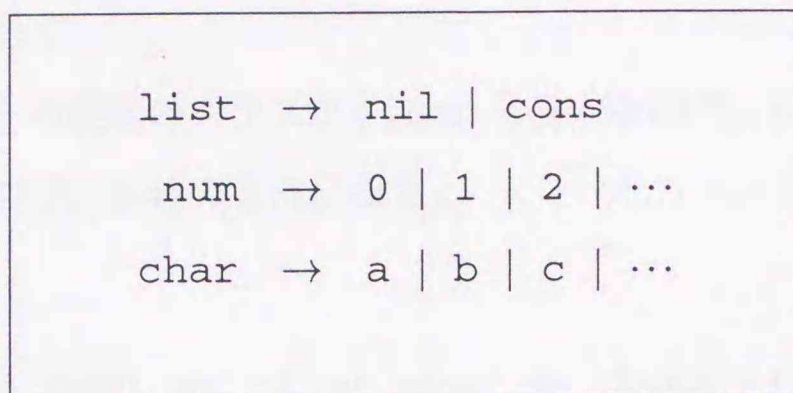


図 8.1: リストを含む定数の階層関係

図 8.1 における,

$$\text{list} \rightarrow \text{nil} \mid \text{cons}$$

の記述は, リストを「長さ 0 のリスト (nil クラス)」と「長さ 1 以上のリスト (cons クラス)」に分類していることを表わす.『整数を要素とする長さ 1 以上のリスト』のクラスを (cons num) で, また,『整数を要素とするリスト』のクラスを (list num) で表す. そのとき, 構造制約は式 (8.2) のように記述できる.

$$(\text{cons num}) == (\text{num}) \oplus (\text{list num}) \quad (8.2)$$

他にも,『文字を要素とする長さ 1 以上のリスト』を表わすクラス (cons char), 『0 のみを要素とする長さ 1 以上のリスト』を表わすクラス (cons 0) などのクラスを持つ原子対象のための構造制約を同様に記述できる. それらは以下の式 (8.3) のように, 変数 *t を用いて, まとめて記述できる.

$$(\text{cons } *t) == (*t) \oplus (\text{list } *t) \quad (8.3)$$

ここで *t は変数であり, 任意の定数, 例えば char や 0 など, を表わす.

さらに『整数のリストを要素として持つ長さ1以上のリスト』を表わすクラスを考える。Prologの記法を用いれば $[[1, 0], [0, 1, 0]]$ や $[[1], [1, 1], [1, 1, 1]]$ などがこのクラスの対象である。このクラスを (cons list num) と表わす。同様に、『整数のリストを要素として持つリスト』のクラスを (list list num) と表わす。

この場合、リストの要素のクラスが (list num) なので、 (cons list num) に対する構造制約は式(8.4)のようになる。

$$(\text{cons list num}) == (\text{list num}) \oplus (\text{list list num}) \quad (8.4)$$

このような場合の構造制約をもまとめて記述するために、ここではLispのS式表現(ドット記法)を流用し、以下の式(8.5)のように記述する。

$$(\text{cons} . *t) == *t \oplus (\text{list} . *t) \quad (8.5)$$

ここで変数 $*t$ は定数の任意のリストを表わしている。

式(8.5)において $*t = ()$ とすれば、式(8.6)を得る。これは (cons) クラスの対象が、 $()$ クラスの対象(すなわち任意の対象)と (list) クラスの対象からなることを表わしている。

$$(\text{cons}) == () \oplus (\text{list}) \quad (8.6)$$

構造制約を記述する変数には $*t$ などの“*”から始まる文字列を用いることにする。この変数は、原子対象を表わすための変数とは異なる。原子対象のための変数と構造制約のための変数を区別すべきときには、前者を対象変数と呼び、後者を構造変数と呼ぶ。

8.2 基底対象: G

問題の答は、問題の表現からある簡単な操作により求められる。4章で述べたように、問題の表現の意味は、特殊化システムにおける基底対象の部分集合として

得られる。したがって、問題の答を得るためには、基底対象の集合が定まっている必要がある。

通常、変数と代入のある領域では、変数のない対象を基底対象と呼ぶ。LHSでも変数はあるが、代入はない。そうではなく、クラスを階層関係に従い特殊化することで計算が進む。LHSの場合には、階層関係の底(比較せよ., 定義6)にある定数のみを持つクラスだけからなる原子対象が、基底対象の候補となる。

LHSの場合には、更に、正しい部分構造の形を、構造制約の宣言により指定していた。

まとめると、LHSの場合では、階層関係の底にある定数のみを持つクラスだけからなる原子対象のうち、構造制約を満足するものが基底対象となる。この形式的な定義は定義20に示す。

8.3 本章のまとめ

本章では、構造制約をLHSに導入した。構造制約とは、例えば論理パラダイムでは、型宣言のように呼ばれていた概念である。特殊化シスムの上で、構造制約は、基底対象を定める働きをする。構造制約を満足しなければ、基底対象となれない。特殊化システムの上で、表現の意味は、基底対象の部分集合となる。したがって、構造制約を満足しない原子対象は、表現の意味の要素となることができず、したがって、問題の答となり得ない。これは、対象の正しい構造を宣言するという、構造制約の直感的な意義と合致している。

第9章 述語名と定数名の統合

9.1 原子論理式と原子対象の対応

原子対象を利用して確定節を表わす方法について述べる.

通常の場合, 原子論理式は “ $\text{pred}(\text{arg}, \dots)$ ” というように表現される. このとき pred は述語であり $\text{arg}, \dots$ は引数である. これに対し, 本論文における原子対象は “ $X: [(\text{const } \dots) \text{ obj } \dots]$ ” という形をしている. この2つを比較すると, 述語 “ pred ” とクラス “ $(\text{const } \dots)$ ” が, 引数 “ $\text{arg}, \dots$ ” と構造 “ $\text{obj } \dots$ ” がそれぞれ対応していることがわかる. したがって, 原子論理式 $\text{pred}(\text{arg}, \dots)$ を原子対象として表現する場合, 対応関係は以下のようにすればよい.

$$\text{pred}(\text{arg}, \dots) \leftrightarrow X: [(\text{pred}) \text{ arg } \dots]$$

これを実際に可能とするためには, 述語を定数の階層の中に統合する必要がある. 例えば, 図 2.1 に示した, `append` プログラムを LHS 上で表現するためには, 図 9.1 に示すような, 定数の階層関係となる.

図 9.1 では, `append` は定数として, $\text{any} \rightarrow \text{pred} \rightarrow \text{append}$ という階層関係の中に組み込まれている.

9.2 クラス対象の導入

9.2.1 変数を用いた原子対象の参照

変数を利用して, 原子対象を参照することにより, 2つの原子対象が等しいことを表現できる. 例を用いて説明する.


```

any → pred | data | list
pred → append
data → num | char
list → nil | cons
num → 0, 1, 2, ...
char → a, b, c, ...

```

図 9.1: append プログラムを表現するための定数の階層関係

例えば「単位元との演算は、消去して構わない」という規則を表現したい。

まず、演算に関する単位元のクラスを (unit) で表わす場合を考える。単位元は、演算子によって変化する。例えば、加算に関する単位元を、

```
[(unit) R:[(+)]]
```

として表わすことにする。

つぎに、数式を処理するための規則を、ここでは、

```
[(rule) E1[(expr)] E2:[(expr)]]
```

のように表現することにする。これは、E1 で参照される原子対象により表現される式を E2 の式へ置き換えても良いということを表わす。

単位元との演算を表現する原子対象は次のようになる。

```
[(c-expr) [(expr)] R:[(opr)] [(unit) R]]
```

この原子対象は、省略記法を用いて表わされている。この場合、式の演算子と、単位元を指定する演算子の両方が、変数 R により繋がれ、等しいものであることを表わしている。

それでは、以上を示した原子対象を組み合わせて、「単位元との演算は、消去し

て構わない」という規則は

$$[(\text{rule}) [(\text{c-expr}) E:[(\text{expr})] R:[(\text{opr})] [(\text{unit}) R]] \\ E] \leftarrow .$$

のように表現できる。前述のように演算子を変数Rで参照され、また、演算子Rに関して、単位元とともに彼演算子となっている式も変数Eにより参照され、それぞれ、同じ原子対象であることを表現している。

このように、2つの原子対象が等しいことを表現するためには、原子対象の変数を利用して、等しい原子対象を参照すればよい、ということがわかる。

9.2.2 変数を用いたクラスの参照の問題

それでは次に、列の全ての要素の合計を求める計算を表現してみる。これには、(sum) クラスの演算を使うことにする。もしも合計を求めるべき列(リスト)の長さが0であるならば、その合計は0である。もしもその長さが1であるならば、その合計は、列の要素と等しい。もしもその長さが1以上であれば、前から2つずつ、順番に加算((add) クラス)を繰り返せば、合計を求めることができる。これを式(9.1)のような表現で表わす。

$$\begin{aligned} &[(\text{sum}) [(\text{nil})] [(0)]] . \\ &[(\text{sum}) [(\text{cons}) A:[(\text{num})] [(\text{nil})]] A] . \\ &[(\text{sum}) [(\text{cons}) A:[(\text{num})] \\ &\quad [(\text{cons}) B:[(\text{num})] R:[(\text{list})]]] \\ &S] \leftarrow [(\text{add}) A B C], [(\text{sum}) [(\text{cons}) C R] Z] . \end{aligned} \tag{9.1}$$

ここで原子対象[(add) A B C]は、AとBを加算(add)した結果がCであるという原子論理式を表現している。

同様の考え方に依り、列の全ての要素の積((prod) クラス)を求める計算を表現することを考える。このためには、式(9.1)において、全ての(sum)を(prod)に置

き換え, 全ての (add) を (mul)(乗算)に置き換えればよい. したがって, 式 (9.2) を得る.

```
[(prod) [(nil)] [(1)]] .
[(prod) [(cons) A:[(num)] [(nil)]] A] .
[(prod) [(cons) A:[(num)]
          [(cons) B:[(num)] R:[(list)]]]
  S] ← [(mul) A B C], [(prod) [(cons) C R] Z] .
```

(9.2)

式 (9.1) と式 (9.2) に示した表現は, その構成がよく似ているので, この2つを1つの表現にまとめることを考える. 例えば式 (9.3) のようにまとめてみる.

```
[(slo) F:[(func)]
  [(nil)] [(unit) F] .
[(slo) [(func)]
  [(cons) A:[(num)] [(nil)]] A] .
[(slo) F:[(func)]
  [(cons) A:[(num)]
    [(cons) B:[(num)] R:[(list)]]]
  S] ← [F] A B C], [(slo) F [(cons) C R] Z] .
```

(9.3)

全体をまとめる原子対象には (slo)(Sum-Like-Operation) というクラスを付けた.

(slo) クラスの原子対象は, 基礎となる演算子を示すために, 式 (9.1) や, 式 (9.2) に比べて, 部分構造の個数を1つ増やしている. ここで, 基礎となる演算子には, 例えば合計 ((sum) クラス) であれば (add) クラスの原子対象が対応する. 式 (9.3) の中では, 変数 F によって参照される原子対象が, 基礎となる演算子を示している.

列の長さが0であった場合には, 各演算子に対応する単位元を, 計算の結果とするために, 9.2.1 節で説明した (unit) クラスを利用している.

ここで **F** で示した変数に注目する. 原子対象の構成から言って, 本来この部分にはクラスが配置されるべきである. しかし, 今回は, [(add)] や [(mul)] な

どの、基礎となる演算子を配置しなくてはならない。演算子は (func) クラスの原子対象として一般的に表現される。そこで、変数Fを用いて、基礎となる演算子である原子対象を参照し、配置している。これは、原子対象の定義に反するので、問題がある。

9.2.3 クラスのための変数の導入

9.2.2 節で説明した問題は、結局、クラスのみを参照する仕組みの必要性を示す。原子対象は、変数によって参照することができる。クラスを参照するためには、同様に、クラスのための変数を導入する必要がある。

例えば、今までは、単に (list) のように表わしていたクラスに、変数 α を付け、 $\alpha: (list)$ のように表わすことを考える。これにより、原子対象のように、クラスも変数を利用して参照することができるようになった。以降では、変数とクラスの組をクラス対象と呼ぶ。

原子対象のための変数と、クラス対象のための変数には区別はない。計算の途中で、原子対象とクラス対象を、同じ変数で参照しようとした場合には、計算を進めることができなくなり、問題の表現の中から取り除かれる。しかし、特に変数を区別する必要が生じた場合には、原子対象のための変数を対象変数、クラス対象のための変数をクラス変数と呼び、両者を区別する。

クラス対象を利用すると、式 (9.3) は、以下に示す式 (9.4) のように改められる。今までは、原子対象の部分構造には原子対象のみを配置することが許されていた。以降では、原子対象の部分構造には、原子対象とクラス対象の両方を配置することを許す。


```

[(slo) α: (func)
  [(nil)] [(unit) α].
[(slo) (func)
  [(cons) A: [(num)] [(nil)]] A].
[(slo) α: (func)
  [(cons) A: [(num)]
    [(cons) B: [(num)] R: [(list)]]]
  S] ← [⌊α⌋ A B C], [(slo) α [(cons) C R] Z].

```

(9.4)

式(9.4)では、9.2.2節で述べた問題が解決されている。

なお、以降では、7.2節で述べた、原子対象に対する省略記法を、クラス対象に対しても、適用することがある。すなわち、クラス変数やクラスを省略する場合がある。

9.3 原子対象 (2): $\mathcal{A}$

7.3節では、原子対象を直感的に定めた。本節では、この定義のうち、部分構造に関する部分を、以下のように改める。

- 原子対象は部分構造を持つ。部分構造とは、原子対象またはクラス対象の0個以上の列である。部分構造に含まれている原子対象とクラス対象の個数を、部分構造の長さと言う。部分構造に含まれている原子対象やクラス対象を、元の原子対象の部分対象と言う。部分対象が原子対象である場合には、再び長さ0以上の部分構造を持つ場合もある。これは再帰的に繰り返される。

9.4 構造制約の記法

8章では、原子対象の部分構造を定めるために、構造制約を導入した。しかし、ここでの構造制約は、クラス対象を導入する前の、部分構造には原子対象のみが

許されている場合のものであった。本節では、式 (9.4) のように、クラス対象も含めて部分構造として存在し得る場合の構造制約について述べる。

原子対象のみを部分構造に許していたときには、部分構造として存在し得る原子対象のクラスを、構造制約として記述していた。この場合、部分構造になるのは、クラス対象なのか原子対象なのか、区別が付かない。そこで次のように、構造制約の記法を改める。

- 部分構造中の部分対象がクラス対象の場合には、(add) のように、そのクラスのみを書く。
- 部分構造中の部分対象が原子対象の場合には、[(add)] のように、クラスのみを持つ原子対象を書く。

このように、構造制約の記法を決めると、式 (9.4) に現れる原子対象の構造制約は、次の式 (9.5) のように表現できる。

$$\left\{ \begin{array}{l} (\text{slo}) == (\text{func}) \oplus [(\text{list})] [(\text{data})] \\ (\text{func}) == [(\text{data})] \oplus [(\text{data})] \oplus [(\text{data})] \\ (\text{cons} \ . \ *t) == [*t] \oplus [(\text{list} \ . *t)] \\ (\text{unit}) == (\text{func}) \end{array} \right. \quad (9.5)$$

なお、式 (9.5) の第3式では、8.1 節で述べた構造変数を利用して、構造制約を記述している。

9.5 例: **append** の表現

これまでの議論を総合して、図 2.1 に対応する、LHS による **append** の表現は、図 9.2 のようになる。

定数の階層関係と構造制約は特殊化システムを作る。この特殊化システムにより、確定節の集合の意味が定まる。この確定節の集合に対して等価変換を繰り返し適用することが、計算である。

9.6 本章のまとめ

本章では、2つのことを述べた。

1つは、確定節を構成するための原子論理式を原子対象を用いて表現するためには、論理パラダイムで言うところの述語にあたる記号を、定数の階層関係に統合する必要があるということである。結果として、述語という概念は、原子対象のクラスとして表現された。

もう1つは、クラスだけを原子対象や確定節の中で参照する必要がある、そのためには、クラスに変数を付加する必要があるということである。変数を付加されたクラスを、クラス対象と呼んだ。

述語はクラス対象により表現され、クラス対象は変数を利用して、参照することができる。このことは、LHSにおいて、いわゆる高階な表現を可能にする。LHSの高階な表現への応用は、13章で詳しく述べる。


```
-- 定数と階層関係の定義
any → pred | data | list
pred → append
data → num | char
list → nil | cons
num → 0, 1, 2, ...
char → a, b, c, ...

-- 構造制約の宣言
(cons . *t) == [*t] ⊕ [(list . *t)]
(append . *t) == [(list . *t)] ⊕
    [(list . *t)] ⊕ [(list . *t)]

-- 確定節の集合
[(append) [(nil)] X:[()] X] ← .
[(append) [(cons) A:[()] X:[()]]
    Y:[()]]
    [(cons) A:[()] Z:[()]]] ← [(append) X Y Z].
```

図 9.2: append の表現

第10章 対象に対する操作

10.1 はじめに

6章から9章において、原子対象を利用した、階層関係と部分構造を持つ知識を表現する方法を述べた。本章では、この原子対象を利用した計算について述べる。今まで述べてきたように、本研究が基礎とする枠組では、計算とは等価変換を適用することである。本論文では LHS のための基礎的な等価変換として、Unfold 変換 [42] を利用する。Unfold 変換は、原子論理式の単一化を利用する。このため、原子対象の単一化のための、特殊化を定め、これを用いて単一化を行う。

10.2 原子対象の単一化

直観的にいえば、2つの原子対象を単一化することは、各原子対象が表現しているものの共通部分を求めることに相当する。原子対象を単一化するためには、以下の手順で計算を行う。

- 2つのクラスの最汎共通子孫を求め、これを、単一化後の原子対象のクラスとする。
- 原子対象の各部分対象毎に単一化を行う。部分対象がクラス対象の場合は、最汎共通子孫のクラスを求め、新しいクラス対象を得る。

最汎共通子孫を求めるための手続きは、手続き 2 に示した。

10.2.1 両方の部分構造の長さが0である例

例えば、以下に示す2つの原子対象aとbを単一化することを考える。なお、図9.1に示した、定数の階層関係を仮定する。

```
a = A:[(data)]
```

```
b = B:[(char)]
```

aもbも、部分構造の長さは0である。aのクラス(data)とbのクラス(char)の最汎共通子孫は(char)である。したがって、aとbは、単一化の結果、どちらも

```
c = C:[(char)]
```

へと特殊化される。ここで、単一化の結果として得られた原子対象の変数がAとBからCへ変化している。

10.2.2 片方のみ部分構造の長さが1以上である例

本節でも図9.1に示した、定数の階層関係を仮定する。以下に示す2つの原子対象aとbを単一化することを考える。

```
a = A:[(list data)]
```

```
b = B:[(cons) S:[(num)] T:[(nil)]]
```

この場合には、まずクラス(list data)と(cons)の最汎共通子孫を求め、(cons data)が導かれる。次に、部分構造を再帰的に単一化する場合に、aの部分対象は0個なので、単にbの部分構造をaの部分構造へ複写する。結果として

```
c = C:[(cons data) S:[(num)] T:[(nil)]]
```

を得る。

10.2.3 両方の部分構造の長さが1以上である例

本節でも図9.1に示した、定数の階層関係を仮定する。以下に示す2つの原子対象aとbを単一化することを考える。


```
a = A:[(cons) X:[()] Y:[(nil)]]
```

```
b = B:[(cons) S:[(num)] T:[(list)]]
```

この場合には、部分構造の単一化が再帰的に行われ、結果として

```
c = C:[(cons) C1:[(num)] C2:[(nil)]]
```

を得る。

10.3 基礎的な特殊化: S, μ

これまで例示してきたように、単一化を行うには、基本的な特殊化として、

- 変数の付け替え
- クラスの特殊化
- 部分対象の追加

の3つがあれば充分であることがわかる。複数の基本的な特殊化を原子対象に適用するためには、

- 基本的な特殊化を原子対象に適用する方法 $-\mu$
- 基本的な特殊化を組み合わせて、新しい特殊化を作る方法

を定める必要がある。これらも含めて特殊化の形式的な定義は 11.6 節に示す。

10.4 本章のまとめ

本章では、原子対象の単一化について述べた。本研究で採用した等価変換である Unfold 変換は、対象の単一化を必要とする。定義 1 に示した Unfold 変換は、具

体的な単一化を固定せずに述べられていた。そのため、Unfold 変換を利用するためには対象の単一化を具体的に定める必要があった。

LHS の原子対象には、階層関係と部分構造がある。このため、階層関係にしたがって、単一化する際には原子対象のクラスを特殊化する。また、部分構造については、その長さを合わせる必要があり、さらに、再帰的な単一化も必要である。

単一化の形式的な定義は 12 章に述べる。

第11章 LHSの表現の定式化

11.1 はじめに

6章から10章では、直感的にLHSを示した、本章では、LHSに対する特殊化システム $\langle \mathcal{A}, \mathcal{G}, S, \mu \rangle$ を、4章に基づき、具体的に定める。特殊化システムは、問題の表現と、その意味を定めるために必要である。

LHSでは、問題の表現は、定数の順序集合、変数の集合、構造制約と共に記述される。以降では、与えられた定数の順序集合を $(\mathbf{C}, \rightarrow)$ 、変数の集合を $\mathbf{V}$ で表わす。順序 $\rightarrow$ は、定数の順序対 $(a, b) \in \mathbf{C} \times \mathbf{C}$ の集合である。 $(a, b) \in \rightarrow$ であることを、 $a \rightarrow b$ と表わし、 a は b の親であると言う。逆に、 b は a の子であるとも言う。構造制約の集合を T 、問題の表現(確定節の集合)を d と書く。なお、 $\mathbf{C}$ と $\mathbf{V}$ は互いに排反であること、および、順序 $\rightarrow$ は木構造をなすことを仮定する。分り易さのために、 $\mathbf{C}$ の元には英小文字 $(a, b, \dots)$ を用い、 $\mathbf{V}$ の元は英大文字 $(X, Y, \dots)$ を用いて表現する。

集合 $(\mathbf{C}, \rightarrow)$ 、 $\mathbf{V}$ 、 T は特殊化システム $\langle \mathcal{A}, \mathcal{G}, S, \mu \rangle$ を定める。そして、この特殊化システム $\langle \mathcal{A}, \mathcal{G}, S, \mu \rangle$ は、問題の表現 d の意味 $\mathcal{M}(d)$ を定める。この流れは次のようになる:

1. 集合 $(\mathbf{C}, \rightarrow)$ 、 $\mathbf{V}$ を用いて、原子対象の集合 $\mathcal{A}$ を定める。
2. 原子対象に対する基本的な操作として S の元(特殊化)を定める。このとき同時に、特殊化を原子対象に作用させる方法 μ も定める。
3. $\mathcal{A}$ の元のうち、部分構造まで含めて、クラスを構成するすべての定数が、順序 $\rightarrow$ に関して、葉に位置するものを $\mathcal{G}_0$ の元とする。
4. $\mathcal{G}_0$ のうち構造制約(T の元)を充足しているものを基底対象($\mathcal{G}$ の元)とする。

5. 特殊化システム $\langle \mathcal{A}, \mathcal{G}, \mathcal{S}, \mu \rangle$ の上で, 問題の表現 d の意味 $\mathcal{M}(d)$ を得る.

このように構築した4つ組 $\langle \mathcal{A}, \mathcal{G}, \mathcal{S}, \mu \rangle$ が特殊化システムの条件(定義1)を満たすことは後で述べる. 以降では, この流れに基づいて, 言語 LHS のための特殊化システムを構築する.

11.2 階層関係

始めに, 定数の集合 $\mathbf{C}$ と, $\mathbf{C}$ 上の順序関係 $\rightarrow$, 変数の集合 $\mathbf{V}$ が与えられる. 以下の3つの条件を仮定する.

条件 CI: 任意の定数 $a \in \mathbf{C}$ に対して, もしも, ある定数 $x \in \mathbf{C}$ が, $x \rightarrow a$ を満足するならば, そのような x は唯一に存在する.

条件 CII: 任意の定数 $a \in \mathbf{C}$ に対して, もしも, いくつかの定数 $x_1, \dots, x_n \in \mathbf{C}$ ($1 \leq n$) が $a \rightarrow x_1 \rightarrow \dots \rightarrow x_n \rightarrow a$ を満足するならば, $a = x_1 = \dots = x_n$ が成り立つ.

条件 CIII: $\mathbf{C} \cap \mathbf{V} = \emptyset$.

定義 5 ($\mathbf{C}$ 上の子孫関係) 定数 $a, b \in \mathbf{C}$ と $\mathbf{C}$ 上の関係 $\rightarrow$ に関して, a が b の子孫であることを, $b \xrightarrow{*} a$ と表わし, 次の条件を満たすことと定義する.

- $b = a$, または
- $\exists b' \in \mathbf{C}, \text{ s.t. } b \rightarrow b', b' \xrightarrow{*} a$

□

もしも $a \xrightarrow{*} b$ が成り立つならば, 列 $a \rightarrow x_1 \rightarrow \dots \rightarrow x_n = b$ ($0 \leq n \leq \infty$) が存在しなくてはならない. 特に, $n \leq \infty$ の条件は, 定義5からは導かれないが, 本論文では, これを仮定する.

定義6(子孫関係の底) 定数 $a \in \mathbf{C}$ に関し, もしも定数 $x \in \mathbf{C}$ で $a \xrightarrow{*} x$ を満足するものが存在しないならば, a は子孫関係の底にあると言う. $\square$

命題1 任意の $a, b \in \mathbf{C}$ に関して, もしも $a \xrightarrow{*} b$ と $b \xrightarrow{*} a$ が成り立つならば, $a = b$ が成り立つ.

証明.

$a \xrightarrow{*} b$ に依り, いくつかの定数 $x_1, \dots, x_n \in \mathbf{C}$ で $a \rightarrow x_1 \rightarrow \dots \rightarrow x_n = b$ ($1 \leq n$) を満たすものが存在する. 逆に, $b \xrightarrow{*} a$ に依り, いくつかの定数 $y_1, \dots, y_m \in \mathbf{C}$ で $b \rightarrow y_1 \rightarrow \dots \rightarrow y_m = a$ ($1 \leq m$) を満たすものが存在する. したがって, 列 $a \rightarrow x_1 \rightarrow \dots \rightarrow x_n = b \rightarrow y_1 \rightarrow \dots \rightarrow y_m = a$ が存在する. 条件CIIに依り, この場合には, $a = x_1 = \dots = x_n = b = y_1 = \dots = y_m$ でなくてはならない.

故に, $a = b$ が成り立つ. $\square$

定義7(クラス) クラスとは, 0個以上の定数の列である. クラスの長さとは, そのクラスに含まれている定数の個数である. 本論文では, 全てのクラスの集合を $\mathbf{C}^*$ で表わす. $\square$

例えば $\mathbf{C} = \{\text{list}, \text{nil}, \text{num}\}$ とすれば, 以下は $\mathbf{C}^*$ の要素の例である.

$()$, (list) , (list num) , (nil)

定義8($\mathbf{C}^*$ 上の子孫関係) 2つのクラス $s = (s_1, \dots, s_m)$, $t = (t_1, \dots, t_n)$ ($0 \leq n, m$) に関して, s が t の子孫であることを, $t \succeq s$ と表わし, 次の条件を満たすことと定義する.

- $n \leq m$, かつ
- $t_i \xrightarrow{*} s_i$, $i = 1, \dots, n$

$\square$

定義 9 (最汎共通子孫 MGCD) 2つのクラス $s, t \in \mathbf{C}^*$ に関して, クラス $u \in \mathbf{C}^*$ で, 次に示す条件を満たすものを, s と t の最汎共通子孫 (Most General Common Descendant, MGCD) と呼び, $\Upsilon(s, t) = u$ と表わす.

- $s \succeq u, t \succeq u$
- $\forall u' \in \mathbf{C}^*$ に関して, $u' \succeq s, u' \succeq t: u \succeq u'$

□

$\forall s, t \in \mathbf{C}^*$ に関して, $\Upsilon(s, t) = \Upsilon(t, s)$ が, 成り立つ. これは定義より自明である. 2つのクラスの最汎共通子孫は, 常に存在するとは限らない. クラス s, t の最汎共通子孫が存在しない場合には, $\Upsilon(s, t) = \perp$ と表わす.

命題 2 もしも 2つのクラスの最汎共通子孫が存在するならば, それは唯一に存在する.

証明.

2つのクラス $s, t \in \mathbf{C}^*$ に関して, $u = \Upsilon(s, t) \neq \perp$ かつ $v = \Upsilon(s, t) \neq \perp$ が成り立つと仮定する. 定義 9 に依り, $u \succeq v$ かつ $v \succeq u$ が成り立つ. したがって, u の長さと v の長さは等しい. これを n とする. ここで $u = (u_1, \dots, u_n), v = (v_1, \dots, v_n)$ とすると, $u \succeq v$ かつ $v \succeq u$ より $u_i \xrightarrow{*} v_i$ かつ $v_i \xrightarrow{*} u_i$ ($i = 1, \dots, n$) が成り立つ. 命題 1 に依り, 全ての $i = 1, \dots, n$ に対して $u_i = v_i$ が成り立つ. 故に $u = v$ が成り立つ. □

2つのクラス $s = (s_1, \dots, s_m)$ と $t = (t_1, \dots, t_n)$ ($0 \leq n \leq m$) に関して, 以下に示す手続きを考える.

手続き 2 1. 個々の $i = 1, \dots, n$ に対して,

- もしも $s_i \xrightarrow{*} t_i$ であるならば, $u_i := t_i$ と代入する.
- もしも $t_i \xrightarrow{*} s_i$ であるならば, $u_i := s_i$ と代入する.
- もしも上のどちらでもないならば, $u = \perp$ と代入し, この手続きを失敗として終了する.

2. 個々の $i = n+1, \dots, m$ に対して, $u_i := s_i$ と代入する.
3. $u := (u_1, \dots, u_m)$ とする.
4. この手続きを成功として終了する.

命題 3 では, 手続き 2 で使用した記法をそのまま利用する.

命題 3 もしも手続き 2 が成功として終了したならば, $u = Y(s, t)$ が成り立つ.

証明.

[最汎子孫] あるクラス $v = (v_1, \dots, v_k) \in \mathbf{C}^*$ に対して, $s \succeq v$ かつ $t \succeq v$ が成り立つと仮定する. もしも $k < m$ であれば, $v \succeq t$ を満たさない. したがって, 常に $m \leq k$ である. 全ての $i = 1, \dots, n$ に対して, $s_i \xrightarrow{*} v_i$ かつ $t_i \xrightarrow{*} v_i$ が成り立つ. 全ての $i = 1, \dots, n$ に対して, もしも $s_i \xrightarrow{*} t_i$ であれば, $u_i = t_i$ が成り立ち, したがって $u_i \xrightarrow{*} v_i$ である. 逆に $t_i \xrightarrow{*} s_i$ であれば, $u_i = s_i$ が成り立ち, したがって $u_i \xrightarrow{*} v_i$ である. このように, 全ての $i = 1, \dots, n$ に対して, 常に $u_i \xrightarrow{*} v_i$ が成り立つ. 全ての $i = n+1, \dots, m$ に対して, $u_i = s_i$ であるので, したがって $u_i \xrightarrow{*} v_i$ である. このように, 全ての $i = 1, \dots, m$ に対して, $u_i \xrightarrow{*} v_i$ が成り立つ. したがって, $u \succeq v$ が成り立つ. 故に, u は最汎な子孫である.

[共通子孫] クラス u の長さは m である. クラス s の長さは m であり, クラス t の長さ n は m 以下である ($m \leq n$). 全ての $i = 1, \dots, n$ に対して, もしも $s_i \xrightarrow{*} t_i$ であるならば, $u_i = t_i$ である. このとき $t_i \xrightarrow{*} u_i$ が成り立つ. 逆に, もしも $t_i \xrightarrow{*} s_i$ であるならば, $u_i = s_i$ であり, したがって $t_i \xrightarrow{*} u_i$ が成り立つ.

このように, $t \succeq u$ が成り立つ. 同様に, 全ての $i = 1, \dots, n$ に対して $s_i \xrightarrow{*} u_i$ が成り立つ. 全ての $i = n+1, \dots, m$ に対して, $s_i = u_i$ である. このとき $s_i \xrightarrow{*} u_i$ が成り立つ. このように, $s \succeq u$ が成り立つ. 故に, u は s と t の共通な子孫である. $\square$

命題 2 と, 命題 3 は, 手続き 2 が成功として終了した場合には, 2 つのクラスの最汎共通子孫が, 唯一に, 得られることを, 保証する.

もう 1 つ, 後で利用するために, 最汎共通子孫に関係のある有用な補題を示す. この補題の中では, 手続き 2 で利用した記号や記法を, そのまま利用する.

命題4 2つのクラス $s, t \in \mathbf{C}^*$ に対して, $u = \Upsilon(s, t) \neq \perp$ を仮定する. このとき, $\Upsilon(s, u) = \Upsilon(t, u) = u$ が成り立つ.

証明.

命題3に依り, u は手続き2により得ることができる. 仮定 $u = \Upsilon(s, t) \neq \perp$ に依り, 全ての $i = 1, \dots, n$ に対して, $s_i \xrightarrow{*} t_i$ または, $t_i \xrightarrow{*} s_i$ が成り立つ.

$\Upsilon(s, u)$ の長さは m である. なぜならば, s の長さは n であり, u の長さは m であり, さらに $n \leq m$ だからである. $\Upsilon(s, u)$ の長さは, s と u の長さのうちの, 長い方になる. 全ての $i = 1, \dots, n$ に対して, もしも $s_i \xrightarrow{*} t_i$ であるならば, $u_i = t_i$ が成り立ち, $s_i \xrightarrow{*} u_i$ を得る. そうではなく, もしも $t_i \xrightarrow{*} s_i$ であるならば, $u_i = s_i$ が成り立ち, $s_i \xrightarrow{*} u_i$ を得る. したがって, 全ての $i = 1, \dots, n$ に対して, $s_i \xrightarrow{*} u_i$ が常に成り立つ.

このとき, 手続き2に依り $u = \Upsilon(s, u)$ を得る.

同様にして, $u = \Upsilon(t, u)$ を得る. □

11.3 クラス対象

クラスに変数を付加し, クラスを, 計算の途中で受け渡しできるような仕組みを提供する. 変数を付加されたクラスをクラス対象 [41] と言う.

定義10 (クラス対象) クラス対象 c とは $c = (\alpha, cl)$ なる2つ組である. このとき $\alpha \in \mathbf{V}$, $cl \in \mathbf{C}^*$ である. C を, 全てのクラス対象の集合とする. □

11.4 原子対象

原子対象 ($\mathcal{A}$ の元) を以下のように帰納的に定義する.

定義11 (原子対象) 原子対象 $a \in \mathcal{A}$ は $a = \langle X, c, sl \rangle$ なる3つ組である. ここで X , c , sl はそれぞれ以下のように定められる.

1. $X \in \mathbf{V}$
2. $c \in C$
3. $sl = (A_1, \dots, A_m), A_i \in \mathcal{A}, i = 1, \dots, m$

□

なお, (2) の c のクラス, すなわち $c = (\alpha, cl)$ の場合の cl を原子対象 a のクラスという. (3) の sl を原子対象 a の部分構造という. sl に含まれている原子対象とクラス対象のそれぞれを, 部分対象という.

クラス対象の変数と原子対象の変数には, 理論的には区別はない. しかし, 敢えて区別した方が理解が明瞭になると判断した場合には, 前者をクラス変数と呼び, 後者を対象変数と呼ぶことにする.

11.5 基底対象

基底対象の集合 G は, $\mathbf{C}$ 上の順序 $\rightarrow$ と構造制約の集合 T の 2 つに基づいて定まる. まず原子対象の集合 $\mathcal{A}$ の元のうち, クラスを構成する定数がすべて, 順序 $\rightarrow$ に関し底にあるものの集合として G_0 を得る. この G_0 の元のうち, T が与える構造制約を充足しているものの集合として, 基底対象の集合 G を得る.

原子対象は部分構造として原子対象を持つので, 1 つの原子対象は, 複数の構造制約から制約を受ける. $g \in G_0$ が T から受ける構造制約全てを充足することを, g は T を充足するといい, 以下のように帰納的に定義する.

定義 12 (構造制約充足の条件) $g = \langle X, c, (g_1, \dots, g_n) \rangle \in G_0$ ($0 \leq n$) とする.

1. 構造制約 $(C == d_1 \oplus \dots \oplus d_m) \in T$ ($0 \leq m$) が存在し, $c = (\alpha, (a_1, \dots, a_x)), C = (b_1, \dots, b_y)$ とすると, $a_1 = b_1, C \succeq (a_1, \dots, a_x)$ を満たす.
2. (部分構造の個数) $n = m$
3. (部分構造のクラス) 各部分構造 g_i ($i = 1, \dots, m$) について, 以下が成り立つ.
 - (a) もしも $d_i \in \mathbf{C}^*$ であるならば, $g_i = (\alpha_i, c_i) \in C$, かつ $d_i \succeq c_i$ が成り立つ.

(b) もしも $d_i = [e], e \in \mathbf{C}^*$ であるならば, $g_i = \langle X_i, (\alpha_i, c_i), sl_i \rangle \in \mathcal{A}$, かつ $e \succeq c_i$ が成り立つ.

なお, 構造制約の中に配置されている, 構造変数があれば, それは, 原子対象との対応により, 具体化される.

4. (部分構造に対する構造制約) 各部分対象 g_i ($i = 1, \dots, n$) が, もしも原子対象であるならば, g_i は構造制約を充足する.

□

11.6 特殊化と原子対象への適用

本節では LHS の特殊化システムにおける特殊化 (S の元) を定義する. 特殊化は原子対象に対する基本的な操作を提供する. 原子対象を特殊化する向きは, 子孫関係の向きである. 基本特殊化は原子対象を子孫関係の向きにしたがって変化させる. この基本特殊化の組合せとして特殊化を定義する.

以下に示す 13 つの操作を基本特殊化として導入する.

- 「変数」 $\rightarrow$ 「変数」
- 「変数」 $\triangleright$ 「クラス」
- 「変数」 $\Rightarrow$ 「原子対象」

原子対象を M とし, 基本特殊化を θ とした場合の, θ の M への適用は $\mu(\theta)(M)$ となるが, 簡単のため, これを $M[\theta]$ と表わす.

定義 13 (「変数」 $\rightarrow$ 「変数」) $x \rightarrow y$ は変数の名前 $x \in \mathbf{V}$ を $y \in \mathbf{V}$ に付け替える.

以下に例を示す.

$$\langle X, c, (s_1, \dots, s_n) \rangle [X \rightarrow Y] = \langle Y, c, (s_1[X \rightarrow Y], \dots, s_n[X \rightarrow Y]) \rangle$$

$$\langle X, c, (s_1, \dots, s_n) \rangle [Z \rightarrow Y] = \langle X, c, (s_1[Z \rightarrow Y], \dots, s_n[Z \rightarrow Y]) \rangle$$

$$(\alpha, c)[\alpha \rightarrow \beta] = (\beta, c)$$

□

このように、基本特殊化は、部分構造に対して、再帰的に適用が繰り返される。この繰り返しが終了するかどうかは、この時点では、問わない。

定義 14 (「変数」 $\triangleright$ 「クラス」) $\alpha \triangleright c$ は $\alpha \in \mathbf{V}$ で示されるクラス対象のクラスを、新しいクラス c に変える。 α で示されるクラス対象が基本特殊化を適用する前に持っていたクラスを d とすれば、 $d \succeq c$ が成り立たなくてはならない (適用可能性)。以下に例を示す。

$$\begin{aligned} \langle X, (\alpha, p), (s_1, \dots, s_n) \rangle [\alpha \triangleright q] &= \langle X, (\alpha, q), (s_1[\alpha \triangleright q], \dots, s_n[\alpha \triangleright q]) \rangle \\ \langle X, (\beta, p), (s_1, \dots, s_n) \rangle [\alpha \triangleright q] &= \langle X, (\beta, p), (s_1[\alpha \triangleright q], \dots, s_n[\alpha \triangleright q]) \rangle \end{aligned}$$

□

定義 15 (「変数」 $\Rightarrow$ 「原子対象」) $x \Rightarrow m$ は $x \in \mathbf{V}$ で示される原子対象に $m \in \mathcal{A}$ を部分構造として追加する。 m は基本特殊化を適用する前の部分構造の列の末尾に付け加えられる。以下に例を示す。

$$\begin{aligned} \langle X, c, (s_1, \dots, s_n) \rangle [X \Rightarrow a] &= \langle X, c, (s_1[X \Rightarrow a], \dots, s_n[X \Rightarrow a], a) \rangle \\ \langle X, c, (s_1, \dots, s_n) \rangle [Z \Rightarrow a] &= \langle X, c, (s_1[X \Rightarrow a], \dots, s_n[X \Rightarrow a]) \rangle \end{aligned}$$

□

基本特殊化のうち $\triangleright$ には、適用可能性の条件が与えられている。この条件を満たさない場合、基本特殊化 $\triangleright$ を適用することはできない。基本特殊化 $\Rightarrow$ の適用は、原子対象に構造制約を満足させなくする可能性がある。この時点では、これをエラーとしないが、11.5 節で述べるように、構造制約を満足しない原子対象は、基底対象とはなれず、計算の途中で消滅する。

定義 16 (特殊化) 基本特殊化の 0 個以上の列を特殊化とする。例えば $\theta_1, \dots, \theta_n$ ($n \geq 0$) をそれぞれ基本特殊化とした場合、これらから構成される特殊化を以下のように表現する。

$$[\theta_1, \dots, \theta_n] \in \mathcal{S}$$

なお $n = 0$ の場合、 $[]$ は単位特殊化である。

□

定義 17 (特殊化の原子対象への適用) $M \in \mathcal{A}$, $\theta \in S$ とした場合の, θ の M への適用は $\mu(\theta)(M)$ と表現される. これを簡単のため $M\theta$ と表現することにする. 特殊化の原子対象への適用を, 以下のように定義する.

$$\begin{aligned} M[] &= M \\ M[\theta_1, \dots, \theta_n] &= (M[\theta_1])[\theta_2, \dots, \theta_n], (n \geq 1) \end{aligned}$$

□

$s = [\theta_1, \dots, \theta_n] \in S$, $M \in \mathcal{A}$ とした場合, すべての基本特殊化 θ_i ($i = 1, \dots, n$) が M に適用可能でなければ, $\mu(s)$ を M に対して適用することはできない. このように μ は, 定義域を S とし, 値域を $\mathcal{A}$ 上の部分写像の集合としている.

定義 18 (特殊化の節への適用) $C = (H \leftarrow B_1, \dots, B_n)$ を節, θ を特殊化とする. このとき C への θ の適用 (これも $C\theta$ と書く) を以下のように定義する.

$$C\theta = (H\theta \leftarrow B_1\theta, \dots, B_n\theta.)$$

なお, θ を C に適用するためには, 原子対象 $H, B_1, \dots, B_n$ のすべてに対して, θ は適用可能性を満足している必要がある. □

11.7 特殊化システムの条件

本節では, 上のように定めた $\langle \mathcal{A}, G, S, \mu \rangle$ が, 定義 1 の条件を満たしており, LHS のための特殊化システムをなしていることを示す.

まず $\mathcal{A}$ には, 集合であるという以外特に条件は付いておらず, $\mathcal{A}$ はこれを満たす.

$\mathcal{A}$ の元のうち, 順序 $\rightarrow$ に関し「葉」の位置にあるものの集合として G_0 を得たので, $\mathcal{A} \supset G_0$ は明らかである. この G_0 の元のうち, 構造制約 T を充足しているものの集合が G である. したがって $G_0 \supset G$ であり, 結果として $\mathcal{A} \supset G$, すなわち特殊化システムの条件 (3) である $G \subset \mathcal{A}$ を満たす.

単位特殊化 $[] \in S$ は特殊化システムの条件 (2) を満たす.

μ は定義 17 の後で述べたように $\mu: S \rightarrow \text{partial_map}(\mathcal{A})$ となっている.

一般的に, 2つの特殊化 $s_1 = [\theta_1, \dots, \theta_n]$, $s_2 = [\sigma_1, \dots, \sigma_m] \in S$ に対して, $s = [\sigma_1, \dots, \sigma_m, \theta_1, \dots, \theta_n]$ とおくと, $\forall M \in \mathcal{A}$ に対し,

$$\begin{aligned}
 & (\mu(s_1) \circ \mu(s_2))(M) \\
 &= \mu(s_1)(\mu(s_2)(M)) \\
 &= \mu(s_1)(Ms_2) \\
 &= \mu(s_1)((M[\sigma_1]) \cdots [\sigma_m]) \\
 &= ((M[\sigma_1]) \cdots [\sigma_m])s_1 \\
 &= (((M[\sigma_1]) \cdots [\sigma_m])[\theta_1]) \cdots [\theta_n] \\
 &= M[\sigma_1, \dots, \sigma_m, \theta_1, \dots, \theta_n] \\
 &= \mu(s)(M)
 \end{aligned}$$

が, s_1 または s_2 が M に適用可能でない場合も含めて成り立つ. したがって, $\mu(s) = \mu(s_1) \circ \mu(s_2)$ となる (条件 (1)).

このように上で求めた $\langle \mathcal{A}, \mathcal{G}, S, \mu \rangle$ は定義 1 にある特殊化システムの条件 (1)~(3) をすべて満たしており, LHS のための特殊化システムをなしている.

11.8 具体例

11.2 節から 11.6 節で示したように, $(\mathbf{C}, \rightarrow), \mathbf{V}$ により $\langle \mathcal{A}, \mathcal{G}_0, S, \mu \rangle$ が定まる. そして, 構造制約の集合 T により $\mathcal{G}_0$ から基底対象の集合 $\mathcal{G}$ が定まり, 言語 LHS の特殊化システム $\langle \mathcal{A}, \mathcal{G}, S, \mu \rangle$ が定まる.

本節では, LHS で表現する対象の具体例として, 図 2.1 に示した `append` プログラムを再び考える. これを含む表現に対する特殊化システム $\langle \mathcal{A}, \mathcal{G}, S, \mu \rangle$ は 11 章で定めた. 基底対象の集合 $\mathcal{G}$ を定めると, 特殊化システム上の表現 d に対して写像 T_d が定まり, 結果として d の意味である $\mathcal{M}(d)$ が定まる.

本章では基底対象を具体的に示し, 意味の具体例とする.

図 2.1 の場合, まず, 定数の集合 $\mathbf{C} = \{\text{list}, \text{cons}, \text{nil}, \text{append}\}$, 定数の階層関係 $\rightarrow = \{(\text{list}, \text{nil}), (\text{list}, \text{cons})\}$, 変数の集合 $\mathbf{V} = \{A, \dots, Z\}$ から, $\mathcal{A}, S, \mu$ が定まる.

階層関係 $\rightarrow$ に関して, 底の位置に存在する定数は $\text{cons}, \text{nil}, \text{append}$ である.

G_0 の元は A の元のうち、これらの定数のみからなるクラスを持つものである。 G_0 の元の例を次に示す。

$$\begin{aligned} &\langle X, (nil), () \rangle \\ &\langle A, (nil), (\langle X, (append), () \rangle) \rangle \\ &\langle M, (cons), (\\ &\quad \langle A, (), () \rangle, \langle K, (nil), () \rangle, \langle L, (nil), () \rangle) \rangle \\ &\langle B, (append), (\\ &\quad \langle S, (nil), () \rangle, \langle T, (nil), () \rangle, \\ &\quad \langle U, (nil), () \rangle) \rangle \end{aligned}$$

これら G_0 の元のうち、構造制約 T を充足している元の集合として、基底対象の集合 G が定まる。図 9.2 では、省略記法も含めて、次のような構造制約が書かれていた。

- (nil) クラスの基底対象は、部分構造を持たない。
- $(cons)$ クラスの基底対象は、部分構造を 2 つ持つ。このクラスを構成する 2 番目以降の定数の列を τ とすると、部分構造の 1 つ目はクラス (τ) の子孫のクラスの基底対象、2 つ目はクラス $(list\tau)$ の子孫のクラスの基底対象である。
- $(append)$ クラスの基底対象は、部分構造を 3 つ持つ。すべて $(list)$ クラスの子孫のクラスの基底対象である。

集合 G_0 の元のうちで、これらの構造制約を充足している元の例を以下に示す。

$$\begin{aligned} &\langle X, (nil), () \rangle \\ &\langle M, (cons), (\\ &\quad \langle A, (), () \rangle, \langle K, (nil), () \rangle) \rangle \\ &\langle B, (append), (\\ &\quad \langle S, (nil), () \rangle, \langle T, (nil), () \rangle, \langle T, (nil), () \rangle) \rangle \end{aligned}$$

このようにして図 9.2 から基底対象が導かれ、集合 G が定まる。

11.9 本章のまとめ

本章は、本論文の中心である。等価変換パラダイムは、表現とその意味を具体的に固定せずに、理論化されている。それゆえに、等価変換パラダイムを、ある具体的な問題に適用するためには、表現系を固定し、その上における表現の意味を定める必要がある。

特殊化システムは、問題を表現するために、確定節を利用する表現系である。本研究では、この特殊化システムを具体的な表現系として採用した。特殊化システムは $\langle \mathcal{A}, G, S, \mu \rangle$ で表わされ、この上に、確定節を定める。特殊化システムにおいて、確定節の集合が問題の表現である。確定節の集合と、その意味を対応させる方法は、やはり $\langle \mathcal{A}, G, S, \mu \rangle$ の上で定義されている。特殊化システムという枠組自身は、具体的な $\langle \mathcal{A}, G, S, \mu \rangle$ を固定せずに、抽象的に定められている。したがって、ある問題を具体的に表現するためには、 $\langle \mathcal{A}, G, S, \mu \rangle$ の各構成要素をやはり具体的に定める必要がある。

本章では、階層関係と部分構造を持つ対象のための特殊化システムを具体的に定めた。これにより、この特殊化システムの上で問題の表現が固定され、表現と意味が対応付けられる。等価変換は、この表現の意味が固定されて始めて具体的に定まる。

このように、特殊化システムを具体的に定めることは、等価変換パラダイムにとって重要なことである。

第12章 LHSの計算の定式化

12.1 はじめに

11章では、LHSの表現を、特殊化システムに基づいて定式化した。本章では、LHSの表現の計算について述べる。LHSでは計算は等価変換に基づいて行われる。

12.2 等価変換に基づく表現の計算

10章では、LHSの原子対象を単一化する方法を、直感的に示した。原子対象の単一化は、5章でも述べたように、LHSの表現を、Unfold変換を用いて等価変換する場合に、必要な手続きである。Unfold変換は、手続き1に示したように、原子対象や原子論理式の形に依らず、一般的に、定められている。したがって、原子対象の単一化を示すことができれば、LHSの表現を、Unfold変換によって等価変換することができる。

12.3節では、原子対象を単一化する手続きを示す。12.6節では、原子対象に構造制約を充足させる手続きを示す。原子対象は、単一化された後に、構造制約を充足させられる。構造制約を充足しない原子対象は、表現の意味から消去される。

12.3 原子対象の単一化

多くの論理型プログラミング言語ではプログラムの実行のためにSLD導出(SLD resolution)を利用する。導出は対象の単一化を利用する。LHSでは、プログラムを実行する方法として、等価変換の一種である展開(unfold)変換[42]を利用する。5.2節でも示したように、展開変換は、その途中で対象の単一化を利用する。この

ように、対象を単一化する手続きは、論理型プログラミング言語一般にとって、重要な基礎をなしている。Lloydは項を対象とした原子論理式の単一化手続きを示した [49]。そこでは、その手続きが単一化手続きであることと、その手続きが有限の時間内に終了することを示した。

論理型プログラミング言語は、項のみを対象とするわけではない。より複雑な知識を効率良く表現するために、項よりも複雑なデータ構造を導入している。項よりも複雑な構造を持つ対象に関する単一化は、当然ながら、項よりも複雑なものになる。LHSでは、階層関係と部分構造を考慮しつつ単一化を行う必要がある。

本章はクラス階層と部分構造を持つ対象を単一化する手続きを示すことを目的とする。これらの対象は項よりも直接的にまたは効率よく、複雑な知識を表現することができる。しかし、クラス階層と部分構造を持つ対象は項よりも複雑な扱いを必要とする。本章の結論として、示された手続きが単一化手続きであり、有限の手続きで停止することを示す。

12.4 単一化手続き

本節では、ある手続きを示す。12.5節では、本節で示した手続きが単一化手続きであることを示す。

定義 19 (単一化子) 2つの原子対象またはクラス対象 $a, b \in \mathcal{A} \cup \mathcal{C}$ に対して、単一化子とは2つの特殊化の組 $(\theta, \sigma) \in S \times S$ でかつ $a\theta = b\sigma$ を満たすものである。□

原子対象とクラス対象を単一化することはできない。これは自明である。

手続き 3 (クラス対象の単一化) 2つのクラス対象 (α, s) と (β, t) に対する、以下に示す手続きを考える。この手続きは、成功した場合には特殊化の組を返す。

1. $u = \Upsilon(s, t)$ を求める。もしも $u = \perp$ だった場合には、手続き全体は失敗し、手続きは停止する。もしも $u \neq \perp$ だった場合には $\tau := [\alpha \triangleright u, \beta \triangleright u]$ とする。
2. $\tau := \tau \cdot [\alpha \xrightarrow{c} \omega, \beta \xrightarrow{c} \omega]$ とする。ここで、 ω は新しいクラス変数である。

3. この手続きは成功し、特殊化の組 (τ, τ) を得る.

□

手続き 3 に示した手続きはクラス対象に対する単一化手続きである. この証明は自明であるので省略する.

ある原子対象が、自分と同じ対象変数を部分構造の中に含む場合、問題がある. 例えば以下のような原子対象を考える.

$$\langle X, x, (\langle X, x', (\dots) \rangle, \dots) \rangle$$

特殊化を原子対象に作用させる場合には、対象変数は原子対象の名札として働く. つまり、特殊化が作用する原子対象は対象変数によって指定される. 同じ対象変数を持つ原子対象は、同じ原子対象であるべきである. 対象変数が同じであるが、異なる部分を持つ原子対象の存在は、計算に混乱をもたらす可能性がある. したがって、上に示した、対象変数として X を持つ外側の原子対象は、部分構造に、同じ対象変数 X を持つ部分対象を持つが、この部分対象も、対象変数として X を持つ原子対象を部分構造に持たなくてはならない. この入れ子構造は無限に連続するので、例えば $[X \stackrel{o}{\rightarrow} Y]$ のような特殊化をこの原子対象に適用すると、入れ子となった部分対象への再帰的な適用が無限に繰り返され、終了しない. そのような種類の原子対象の取り扱いに関する考察は本論文の範囲外とする. すなわち、本論文では、定義 20 に定義される条件を満足する原子対象のみが出現するものとする. この定義を満足する原子対象を正則対象と呼ぶ.

定義 20 (正則対象) 原子対象 $\langle X, (\alpha, s), (a_1, \dots, a_n) \rangle$ ($0 \leq n$) が正則対象であるための必要充分条件は、この原子対象が次の条件を満足することである.

1. 対象変数 X は、全ての部分対象 $a_1, \dots, a_n$ の中に出現しない.
2. 再帰的に、各部分対象 $a_1, \dots, a_n$ は正則対象である.

□

なお、ある原子対象が正則対象である場合、その原子対象は正則であるとも言
う。本論文中に出現する全ての原子対象は正則であると仮定する。あいまいさが
ない場合には、正則対象のことを単に対象と呼ぶ場合がある。

次に示す正則対象 a と b に対し、手続き 4 に示す手続きを適用することを考える。

$$\begin{cases} a = \langle X, (\alpha, s), (a_1, \dots, a_m) \rangle, \\ b = \langle Y, (\beta, t), (b_1, \dots, b_n) \rangle, \text{ where } 0 \leq n \leq m. \end{cases}$$

手続き 4 に示す手続きを対象 a と b に適用する前に、対象中に出現する対象変
数やクラス変数に関して、適切な特殊化を適用することにより、 a と b に出現す
る変数の名前を衝突しないようにしておく。すなわち、特殊化 ρ_a と ρ_b により、
 $V(a\rho_a) \cap V(b\rho_b) = \emptyset$ となるようにする。以降では、再代入 $a := a\rho_a$ と $b := b\rho_b$ が
なされているものと仮定して議論を進める。

部分対象の個数について、この手続きは次の 2 つの場合だけを許す。(1) $n = m$,
または (2) n と m の片方または両方は 0 である。これら以外の場合には、手続き 4
は直に停止し失敗する。

この手続きは再帰的に定義されている。手続きが成功した場合、特殊化の組 (τ, τ)
が得られる。12.5 節において、この組が単一化演算子であることを証明する。

手続き 4 (原子対象の単一化)

1. $m = 0$ の場合、 $0 \leq n \leq m$ であるので、この場合は $n = 0$ である。手順 6 番へ
進む。 $m \neq 0$ かつ $n = 0$ の場合、手順 3 へ進む。どちらの場合にも τ' を $[]$ と
する。
2. これは $n = m \neq 0$ の場合である。

部分対象に対して、再帰的にこの手続きを適用する。 a_1 と b_1 に対して手続き
が成功し、特殊化の組 (τ_1, τ_1) が得られたと仮定する。この τ_1 は a と b のクラ
スと全ての部分対象に対して影響を及ぼす可能性がある。このとき τ_1 は (α, s)
と (β, t) に対して適用可能でなくてはならない。

次の段階では、 $a_2\tau_1$ と $b_2\tau_1$ に、この手続きを適用する。

各 $i = 1, \dots, n = m$ に対して, もし部分対象 $a_i \tau_1 \cdots \tau_{i-1}$ と $b_i \tau_1 \cdots \tau_{i-1}$ に対する手続きの適用が成功したと仮定すると, それぞれの i に対して特殊化の組 (τ_i, τ_i) が得られる. z_i を $a_i \tau_1 \cdots \tau_{i-1} \cdot \tau_i = b_i \tau_1 \cdots \tau_{i-1} \cdot \tau_i$ ($i = 1, \dots, n = m$) とする. τ' を $\tau_1 \cdots \tau_m$ とする.

ある i に関してこの手続きの適用が失敗した場合には, 全体の手続きは停止し, 失敗する.

3. $n = m$ の場合, $\tau^s := []$ とする. そうではなく $n \neq m$ の場合には, $n = 0$ かつ $m \neq 0$ が成り立ち, かつ, 手順 2 は実行されていない. この場合には, $\tau^s := [Y \Rightarrow a_1, \dots, Y \Rightarrow a_m]$ とする.

4. $(\alpha, s)\tau' \cdot \tau^s$ と $(\beta, t)\tau' \cdot \tau^s$ に対して, この手続きを適用する. もしも手続きの適用が失敗した場合には, 全体の手続きは停止し, 失敗する. 成功した場合には, 特殊化の組 (τ^c, τ^c) が得られる.

$(\alpha, s)\tau' \cdot \tau^s = (\alpha', s')$ および, $(\beta, t)\tau' \cdot \tau^s = (\beta', t')$ を仮定すると, $\tau^c = [\alpha' \triangleright u, \beta' \triangleright u, \alpha' \xrightarrow{c} \omega, \beta' \xrightarrow{c} \omega]$ および, $u = Y(s', t')$ と書ける. ここで $\omega \in \mathbf{V}$ は新しい変数である.

補題 4 は, $(\alpha', s')[\alpha' \triangleright u] = (\alpha', Y(s', u)) = (\alpha', u)$ and $(\beta', t')[\beta' \triangleright u] = (\beta', u)$ を保証する.

もしも, 基礎演算子 τ^c がいくつかの部分対象へ適用できないならば, 全体の手続きは停止し失敗する.

5. 対象変数を同じものに付け替える. 対象 a, b は正則であるから, 対象数 X と Y は, それぞれの部分構造の中には出現せず. かつ, $\tau' \cdot \tau^s \cdot \tau^c$ の影響を受けない. $\tau^v := [X \xrightarrow{o} Z, Y \xrightarrow{o} Z]$ とする. ここで, Z は新しい対象変数である. したがって τ^v は部分対象に影響を及ぼさない.

6. 原子対象 a, b に対する全体の手続きは成功し, 特殊化の組 (τ, τ) を得る. ここで, $\tau = \tau' \cdot \tau^s \cdot \tau^c \cdot \tau^v$ である.

□

もともとの、名前を付け替えていない原子対象 a, b に関しては、特殊化の組として $(\tau_a \cdot \rho_a, \tau_b \cdot \rho_b)$ が得られる。

12.5 定理と証明

本節に示す定理や証明においては、手続き4で使った記号などを引き続き使用する。

定理1 (停止性) 2つの正則な対象 a, b に対して、手続き4は有限回の繰返しで停止する。

証明. $n = 0$ かつ $m \neq 0$ の場合、又は、 $n = m = 0$ の場合には、手順2は実行されず、したがって手順4は直ちに手順1において停止する。 $n = m \neq 0$ の場合には、手順2は再帰的に繰返し実行される。手順2以外の手順には再帰的な繰返しはない。したがって、 $n = m \neq 0$ の場合に手順2が有限回の繰返しで停止することを示せばよい。

一般的に、正則な原子対象 $g = \langle G, r, (g_1, \dots, g_k) \rangle$ の高さを $|g|$ と書き、次のように定義する：

$$\begin{cases} |g| = 0, & \text{if } k = 0, \\ |g| = \max_{g_i \in \mathcal{A}} (|g_i|) + 1, & \text{if } 1 \leq k. \end{cases}$$

この定義に依れば、任意の i に対して $1 \leq |g| - |g_i|$ 、かつ、 $0 \leq |g|$ が成り立つ。 g は正則対象なので、 $|g| < \infty$ を満足する。

$n = m$ なので、 $\tau = []$ であり、部分対象の追加はなされないで、この場合には、 n や m の値は最初の値から変化せず、一定である。 $1 \leq |g| - |g_i|$ なので、手順2が実行される度に、対象の高さは1またはそれ以上ずつ減少する。原子対象の高さの最小値は0である。したがって、手順2を有限回繰返すことにより、原子対象の高さは0へ到達する。例えば $\langle X, (\alpha, s), () \rangle$ のような高さが0の原子対象に対し、手続き4は手順1において、直ちに終了する。 $\square$

定理 2 (健全性) 任意の正則対象 a, b に対して, $V(a) \cap V(b) = \emptyset$ を仮定する. 手続き 4 によって特殊化の組 (θ, σ) が得られた場合には, $a\theta = b\sigma$ が成り立つ.

証明.

(1) $n = m = 0$ の場合:

$\tau' = []$ かつ $\tau^s = []$ である.

$$\begin{aligned} a\tau &= \langle X, (\alpha, s), () \rangle \tau^c \cdot \tau^v \\ &= \langle Z, (\omega, u), () \rangle \end{aligned}$$

$$\begin{aligned} b\tau &= \langle Y, (\beta, t), () \rangle \tau^c \cdot \tau^v \\ &= \langle Z, (\omega, u), () \rangle \end{aligned}$$

このように, $a\tau = b\tau$ が成り立つ.

(2) $n = 0$ かつ $m \neq 0$ の場合:

$\tau' = []$ かつ $\tau^s = [Y \Rightarrow a_1, \dots, Y \Rightarrow a_m]$ である.

$$\begin{aligned} a\tau &= \langle X, (\alpha, s), (a_1, \dots, a_m) \rangle \tau^s \cdot \tau^c \cdot \tau^v \\ &= \langle X, (\alpha, s), (a_1, \dots, a_m) \rangle \tau^c \cdot \tau^v \\ &= \langle Z, (\omega, u), (a_1 \tau^c \cdot \tau^v, \dots, a_m \tau^c \cdot \tau^v) \rangle \\ &= \langle Z, (\omega, u), (a_1 \tau^c, \dots, a_m \tau^c) \rangle \end{aligned}$$

原子対象 a は正則対象であるので, 対象変数 X は a の部分構造の中には現れない. $V(a) \cap V(b) = \emptyset$ であるので, 対象変数 Y は a の部分構造の中には現れない. したがって, 基本特殊化 τ^s は a に対して, 影響を与えない. $\tau^v = [X \xrightarrow{o} Z, Y \xrightarrow{o} Z]$ は a の部分対象に対して影響を与えない.

$$\begin{aligned} b\tau &= \langle Y, (\beta, t), () \rangle \tau^s \cdot \tau^c \cdot \tau^v \\ &= \langle Z, (\beta, t), (a_1, \dots, a_m) \rangle \tau^c \cdot \tau^v \\ &= \langle Z, (\omega, u), (a_1 \tau^c, \dots, a_m \tau^c) \rangle \end{aligned}$$

このように, $a\theta = b\sigma$ が成り立つ.

(3) $n = m \neq 0$ の場合:

この場合, n を a, b 両方の部分対象の個数を表わすものとして使用する. $\tau' = \tau_1 \cdots \tau_n$ かつ, $\tau^s = []$ である.

$$\begin{aligned}
 a\tau &= \langle X, (\alpha, s), (a_1, \dots, a_n) \rangle \tau' \cdot \tau^c \cdot \tau^v \\
 &= \langle X, (\alpha, s) \tau_1, (z_1, a_2 \tau_1, \dots, a_n \tau_1) \rangle \\
 &\quad \tau_2 \cdots \tau_n \cdot \tau^c \cdot \tau^v \\
 &= \langle X, (\alpha, s) \tau_1 \cdot \tau_2, (z_1 \tau_2, z_2, \\
 &\quad a_3 \tau_1 \cdot \tau_2, \dots, a_n \tau_1) \rangle \tau_3 \cdots \tau_n \cdot \tau^c \cdot \tau^v \\
 &= \dots \\
 &= \langle X, (\alpha, s) \tau', \\
 &\quad (z_1 \tau_2 \cdots \tau_n, z_2 \tau_3 \cdots \tau_n, \dots, \\
 &\quad z_{n-1} \tau_n, z_n) \rangle \tau^c \cdot \tau^v \\
 &= \langle Z, (\omega, u), \\
 &\quad (z_1 \tau_2 \cdots \tau_n \cdot \tau^c, z_2 \tau_3 \cdots \tau_n \cdot \tau^c, \dots, \\
 &\quad z_{n-1} \tau_n \cdot \tau^c, z_n \tau^c) \rangle
 \end{aligned}$$

$$\begin{aligned}
 b\tau &= \langle Z, (\omega, u), \\
 &\quad (z_1 \tau_2 \cdots \tau_n \cdot \tau^c, z_2 \tau_3 \cdots \tau_n \cdot \tau^c, \dots, \\
 &\quad z_{n-1} \tau_n \cdot \tau^c, z_n \tau^c) \rangle
 \end{aligned}$$

このように, $a\tau = b\tau$ が成り立つ.

(1) ~ (3) はあり得る全ての場合を含む. また (1) ~ (3) の全ての場合において $a\tau = b\tau$ が成り立つ. したがって, あり得る全ての場合において $a\tau = b\tau$ が成り立つ. $\square$

12.6 構造制約の充足

定義 12 に, 原子対象が構造制約を充足している状態を定義した. 本節では, この定義にしたがい, 与えられた原子対象に対して, 構造制約を充足させるための

手続きを示す.

手続き 5 (構造制約の充足) 原子対象 $g = \langle X, c, (g_1, \dots, g_n) \rangle$ ($0 \leq n$) について構造制約を充足させる手続きは, 以下のようになる.

1. $c = (\alpha, a_1, \dots, a_x)$ とする.

(a) 構造制約 ($C == d_1 \oplus \dots \oplus d_m$) ($0 \leq m$) が存在し, $C = (b_1, \dots, b_y)$ のときに,

$a_x = b_x$ かつ $C \sqsubseteq (a_1, \dots, a_x)$ を満足する場合, 次へ進む.

(b) そうでない場合, 構造制約充足手続きは失敗して終了する.

2. $n = m$ の場合, 次へ進む. そうでない場合, 構造制約充足手続きは失敗して終了する.

3. 構造制約の中に配置されている, 構造変数があれば, これを, 原子対象 g との対応により, 具体化する.

4. 各部分構造 g_i ($i = 1, \dots, m$) について, 次の操作を繰り返す.

(a) もしも $d_i \in \mathbf{C}^*$ であるならば, $g_i = (\alpha_i, c_i) \in \mathbf{C}^*$ である場合には,

$$g_i := g_i[\alpha_i \triangleright d_i]$$

とする. この特殊化が適用不可能な場合, または, $g_i \in \mathcal{A}$ である場合には, 構造制約充足の手続きは失敗して終了する.

(b) もしも $d_i = [e] \in \mathcal{A}$, $e \in \mathbf{C}^*$ であるならば, $g_i = \langle X_i, (\alpha_i, c_i), sl_i \rangle \in \mathcal{A}$ である場合には,

$$g_i := g_i[\alpha_i \triangleright e]$$

とする. この特殊化が適用不可能な場合, または, $g_i \in \mathbf{C}$ である場合には, 構造制約充足の手続きは失敗して終了する.

5. 各, 部分対象 g_i ($i = 1, \dots, n$) について, 構造制約を充足させる. ある i について, 構造制約充足の手続きが失敗して終了した場合には, 全体の手続きも失敗して終了する. そうでない場合には, 原子対象 g に対する構造制約充足の手続きは, 成功して終了する.



なお、構造変数を具体化する場合には、構造変数は常に、原子対象の中の定数、または、定数の列(クラス)と対応する。定数または定数の列(クラス)は変数ではないので、構造変数の具体化は、常に成功する。つまり、構造制約を利用する時点で、構造変数は必ず具体的に定まっている。

12.7 本章のまとめ

本章では、原子対象どうしの単一化手続きを形式的に定めた。単一化手続きは Unfold 変換を適用する場合に必要となる。Unfold 変換自身は具体的な単一化を定めていないので、対象を具体的に固定する度に単一化手続きも具体的に定める必要がある。

また、本章では、構造制約を充足させるための手続きも示した。Unfold 変換を利用して計算を進めてゆく途中で、構造制約を満足しなくなった原子対象は、表現の意味の中に残ることができない。したがって、問題の答には影響を及ぼさない。

第13章 高階表現

13.1 はじめに

本章では、LHS による知識表現の 1 つとして、いわゆる高階な表現を示す。高階と呼ばれる表現は、複雑な知識を表現する場合には、常に必要となる。しかし、階の概念が必要かどうかは、議論のある部分である。高階な表現には、例えば文献 [22, 25] などで述べられているような、単一化の場合における問題がある。

階の概念は、述語を特別な記号であるとしたことに、由来する。したがって、LHS のように、述語を特別扱いせず、定数と同一視する場合には、階という概念は、そもそも現れない。

本章では、論理パラダイムで高階と呼ばれている表現のうち、知識表現に有用であると思われる 2 種類の表現に注目する。これらは厳密には 2 階の表現と呼ばれる。これらの表現を LHS でどのように表現するか示す。

13.2 2 種類の高階表現

述語 P を持つ原子論理式が頭部表われる節のことを P 節と呼ぶことにする。2 つの原子論理式の論理積を示す and 節は、Prolog 風の記法を用いると図 13.1 のように表現される。

$$\text{and}(P, Q) :- P, Q.$$

図 13.1: Prolog 風の and 節

Prolog の理論的基礎は 1 階述語論理である [49]. この理論では, 原子論理式の表われるべき場所に変数が出現することを許さない. 上の and 節の右辺にある変数 P と Q は原子論理式が表われるべき場所に出現している. 1 階述語論理の理論では, これは許されない. したがって, これらの変数 P と Q は, 理論的には, Prolog を超越している. Prolog は, このような表現を, 実用的な観点から許し, 採用している.

1 階述語論理によって許されていない場所に表われる変数は, 高階な変数と呼ばれる. 節, 原子論理式, 述語は, 高階な述語を使用する場合には, 高階であると言われる.

高階な表現を扱うために, たくさんの論理型プログラミング言語が, その理論的基礎とともに開発されてきている. そのような理論は, 高階論理と呼ばれる. 例えば LIFE [1] は全ての対象を ψ -項として表現する. LIFE の変数は, 原子論理式を含む任意の ψ -項を参照することができる. それゆえに, 上で示した and 節を LIFE の理論的基礎の中で捉えることは可能である.

高階な表現の別な例として, map 節を考える. これは Prolog 風の表記を用いると図 13.2 のように表わされる.

```
map(P, [], []) :- .  
map(P, [E1|R1], [E2|R2]) :-  
    P(E1, E2), map(P, R1, R2).
```

図 13.2: Prolog 風の map 節

and 節の場合には, 高階な変数は原子論理式を参照していた. これに対して, 2 つ目の節の右辺にある原子論理式 $P(E1, E2)$ に現われる変数 P は, 述語の名前を参照する. このような変数の出現も, やはり, 1 階述語論理では許されていない. したがって, 上に示した map 節は高階な節である. LIFE は述語の名前を参照するための高階な変数を持たないため, map 節は LIFE の理論的基礎を超越している.

M. Hanus は上に示したmap節のような高階な表現を、1階の表現のみを利用して表わすことを試みた [24]. 彼はこのために、新しい述語apply2と、新しい定数λnot, λincを導入した. これらの定数は、述語notや述語incに対応する. そして、図13.3のように、map節を表現した.

```
map(P, [], []) :- .  
map(P, [E1|R1], [E2|R2]) :- .  
    apply2(P, E1, E2), map(P, R1, R2).  
apply2(λnot, X, Y) :- not(X, Y).  
apply2(λinc, X, Y) :- inc(X, Y).  
not(true, false).  
not(false, true).  
inc(0, 1).  
...
```

図13.3: Hanus のmap節

ここに示したmap節は、全て1階の表現のみを利用して表わされている. 1階述語論理が、出現を許していない場所に、変数は表われていない. これで、もともと高階な表現であったmap節を、1階の表現のみで表わすという問題は解決されたように見えるが、実はそうではない. M. Hanus も述べているが、新しい問題が現われてしまう. プログラムを記述した人間は、定数λnotと述語notの間には対応関係があると思っている. しかし、それを保証する枠組がない. それゆえ、意味的には誤りである節:

```
apply2(λinc, X, Y) :- not(X, Y).
```

は構文的には正しいものとして、M. Hanus の方法の中では捉えられてしまう. もとものの、高階な表現で現わされたmap節には、そのような意味的に誤った節を

許すという意図はなかった。このことは、高階なmap節を1階の表現のみを利用して表わす方法として、M. Hanusの方法は充分ではないことを示している。本節の結論をまとめると、

1. 高階な表現の中には、and節のように、原子論理式を変数で参照するものと、map節のように、述語を変数で参照するものの2種類が存在する。

正確な表現を用いれば、本論文で扱おうとしている2つの高階な表現はどちらも「2階」の表現と呼ばれる。このように、高階な表現には階の数を付けて呼ぶ。したがってさらに高階な表現も存在する。本論文では、実用上有用な表現として2階の表現を例として採用している。

2. 高階な表現を1階の表現のみで表わすことには無理がある。

となる。

13.3 原子論理式を対象変数で参照する

LHSの変数は、原子対象を参照することができる。原子対象は原子論理式を表わすことができる。したがって、LHSの変数は、原子論理式を参照することができる。

13.2節では、原子論理式を変数で参照する節の例としてand節を例に挙げた。このand節はLHSの表現を使うと図13.4のように表現できる。

$$[(\text{and})\ P:[(\text{pred})]\ Q:[(\text{pred})]] \leftarrow P, Q.$$

図13.4: LHSのand節

13.4 述語を対象変数で参照する

13.2 節に示した2つの高階な表現のうち、述語を変数で参照するmap節をLHSで表現することを考える。

最初に図13.5に示すような、LHSにおけるmap節の表現を考えてみる。

```

[(map) [(pred)] L:[(nil)] L] ←.
[(map) P:[(pred)]
      [(cons) E1:[()] R1:[(list)]]
      [(cons) E2:[()] R2:[(list)]]] ←
P:[(pred) E1 E2], [(map) P R1 R2].

```

図13.5: 原子対象を利用したmap節の表現

この表現では、クラスの矛盾が起こってしまう。変数Pによって参照される原子対象**P:[(pred)]**は部分対象を持たない。しかし、Pによって参照される、もう1つの、原子対象**P:[(pred) E1 E2]**は部分対象E1とE2を持つからである。もちろんP:[(pred)]とP:[(pred) E1 E2]を単一化して、両方をP':[(pred) E1 E2]とすることはできる。しかしながら、最初のP:[(pred)]は、述語mapの引数として、述語名のみを表現していた。したがって最初のP:[(pred)]は、部分対象を持つてはいけない。逆に、2番目の原子対象P:[(pred) E1 E2]は、P:[(pred)]が参照する述語に、引数E1とE2を与えて、原子論理式を構成しなければならない。したがって、引数E1とE2ををなくしてしまうことはできない。このように、変数Pで参照される2つの原子対象は、お互いに矛盾した存在となってしまう。3番目の変数**P**では、述語名を参照したいので、最初のPと役割は同じである。

13.5 述語をクラスで参照する

これは省略記法に起因する問題ではない。これは、原子対象のみを利用した場合の、高階な表現に関する LHS の記述力の限界を表わしている。そこで、解決策として、原子論理式には原子対象を対応させ、述語名にはクラス対象を対応させることを考える。この方法を用いると図 13.5 の表現は図 13.6 のように書き換えられる。

```

[(map) (pred) L:[(nil)] L]←.
[(map) α:(pred)
  [(cons) E1:[()] R1:[(list)]]
  [(cons) E2:[()] R2:[(list)]]]←
[α E1 E2], [(map) α R1 R2].

```

図 13.6: クラス対象を利用したmap 節の表現

クラス対象を用いた表現により、述語名のみを変数で参照する場合と、原子論理式を変数で参照する場合のどちらも、矛盾することなく表現できていることがわかる。

13.6 本章のまとめ

本章では、論理パラダイムで言うところの、いわゆる高階な表現を、LHS で扱う方法を述べた。本章では、特に有用と思われる2階の高階表現を2種類選び、これを LHS で表現した。

原子論理式は、原子対象の変数で参照することができ、また、述語に当たる部分はクラス対象の変数で参照することができることがわかった。

第14章 型推論

14.1 はじめに

本章では、LHS の表現に対する型推論の手続き [38] を示す。本章において「型」という語は、対象に何らかの情報を付加するものという意味を持つ。いくつかの型付きの論理型言語においては、 x で表わされる対象で τ という型を持つものを、 $x:\tau$ と表現する。型は対象が属すべき集合を示すことがある。例えば「人間」、「動物」、「車」などは型の例である。

「型推論」とは、対象が持つ型を詳細化するための方法である。型推論を行う場合には、宣言的記述の中に明示された情報や、その外から与えられる情報を利用する。例えば、型の情報を持たない、あるいは空の型を持つ対象 x に型推論を施すことにより、 x はある型 τ で型付けされることが判明する可能性がある。この場合 x は $x:\tau$ へと変化する。 τ よりも τ' の方がより詳細な型だと仮定する。 $x:\tau$ に対して、更なる型推論を施した結果、 x の型が τ' であると判明する可能性がある。この場合には、 $x:\tau$ は $x:\tau'$ へと変化する。このように、型推論を行う際には、型の順序関係が定められている必要がある。

LHS では、型はクラスで実現される。本章は、クラスを型と見た型推論の手続きを示す。本章が示す型推論の手続きは SPS (Structure Preserving Specialization) と呼ばれる。SPS は与えられた表現を変換し、型の情報を詳細化する。型を詳細化するために、単一化を利用する。

14.2 宣言的記述の変換

本節は *SPS* (structure preserving specialization) と呼ばれる, 特殊化システム上の宣言的記述 (問題の表現) に対する変換手続きを示す. この名前が示すように, *SPS* は与えられた表現の構造を変更することなく, 特殊化する. この場合, 表現とは確定節の集合であり, 表現の構造とは, 確定節の個数と, 体部 (ボディ) にある原子論理式の個数のことを示す.

ある表現 d を *SLD* 導出により計算する場合, 体部にある, ある原子論理式 b は, ある頭部 (ヘッド) h と単一化される. そして, h がもともと持っていた情報と, h と b が単一化されることにより b から h へ伝播した情報は h を頭部とする節の体部へと伝播する. ここにおいて, h がもともと保持している情報を, *SLD* 導出を適用する前に体部へと伝播させると, もとの d は d' へと変化するが, d' から導出される答は, d から導出される答から変化しない. したがって $\mathcal{M}(d) = \mathcal{M}(d')$ が成り立つ. *SPS* はこの考えに基づく.

同様に, 体部から頭部へ情報を伝播させることも考えられ得る. しかし, 後に述べるように, この場合, 表現の意味が保存されない. 頭部から体部へ情報を伝播させる *SPS* を *SPSHtoB* (*SPS* Head to Body) と呼び, 逆に体部から頭部へと情報を伝播させるものを *SPSBtoH* (*SPS* Body to Head) と呼ぶ. 本節では, 表現の意味を保存する変換である, *SPSHtoB* を扱う.

14.2.1 定義と命題

定義 21 (**C**上の最殊共通祖先) 2つの定数 $a, b \in \mathbf{C}$ が与えられているものとする. 定数 $x \in \mathbf{C}$ で以下の条件を満足するものを a と b の最殊共通祖先 (the Most Specific Common Ancestor, *MSCA*) と呼び, $\lambda(a, b) = x$ と書く.

$$\begin{cases} x \xrightarrow{*} a, x \xrightarrow{*} b, \\ x' \xrightarrow{*} a, x' \xrightarrow{*} b \Rightarrow x' \xrightarrow{*} x \text{ for } \forall x' \in \mathbf{C}. \end{cases}$$

2つの定数の最殊共通祖先は, 常に存在するわけではない. もし定数 a と b の最殊共通祖先が存在しない場合には $\lambda(a, b) = \perp$ と書く. □

$\forall a \in \mathbf{C}$ に対して, $\lambda(a, \top) = \lambda(\top, a) = \top$ と定義する. $\forall a, b \in \mathbf{C}$ に対して, $\lambda(a, b) = \lambda(b, a)$ が成り立つ. 証明は最殊共通祖先の定義から自明なので省略する.

定数 $a, b \in \mathbf{C}$ の最殊共通祖先は, もしそれが存在するならば, 次に述べる手続き 6 によって得ることができる.

手続き 6

1. もしも a が根にあるなら, $s_1 := a$ とする.
2. a が根にない場合, 条件 CI に依り, 列 $s_1, \dots, s_m \in \mathbf{C}$ ($1 \leq m$) で $s_1 \rightarrow \dots \rightarrow s_m \rightarrow a$ かつ $s_i \neq a$ ($i = 1, \dots, m$) を満すものを一意に得ることができる.
3. 同様に, $t_1 := b$ とするか, または, 列 $t_1 \rightarrow \dots \rightarrow t_n \rightarrow b$ ($1 \leq n$) を一意に得る.
4. 整数 k ($1 \leq k$) を, $s_k = t_k$ を満足する最大の値とする. もしそのような k が存在しない場合には $k := 0$ とする.

□

次の 2 つの命題が, この手続きによって得られる列を特徴づける. これらの命題の中では, 手続き 6 で使用した記号や記法そのまま使用する.

命題 5 もしも $k = 0$ ならば, $\lambda(a, b) = \top$ である.

証明. 任意の定数 $x \in \mathbf{C}$ が, $x \xrightarrow{*} a$ と $x \xrightarrow{*} b$ の両方を同時に満足しないということなので, a と b の最殊共通祖先は存在しない. □

命題 6 もしも $1 \leq k$ ならば, $\lambda(a, b) = t_k$ (または s_k) である.

証明.

[共通祖先であること] 自明である.

[最殊であること] 列 $t_1, \dots, t_k \in \mathbf{C}$ で, $t_1 \rightarrow \dots \rightarrow t_k$ ($1 \leq k$) を満足するものが存在する. もしも, 定数 $t \in \mathbf{C}$ で $t \xrightarrow{*} a$ かつ $t \xrightarrow{*} b$ を満足するものが存在するならば, この t は $t_1, \dots, t_k$ の中のいずれかである. すなわち, ある i ($1 \leq i \leq k$) に対して $t = t_i \rightarrow \dots \rightarrow t_k$ だとなることができ, したがって, $t \xrightarrow{*} t_k$ が成り立つ. □

命題 7 $\lambda(a, \lambda(b, c)) = \lambda(\lambda(a, b), c)$.

証明. 2つの列 $t_1 \rightarrow \cdots \rightarrow t_x \rightarrow b_{x+1} \rightarrow \cdots \rightarrow b$ と $t_1 \rightarrow \cdots \rightarrow t_x \rightarrow c_{x+1} \rightarrow \cdots \rightarrow c$ が存在する. ここで, $t_1 \in \mathbf{C}$ は根に位置する. $b_{x+1} \neq c_{x+1}$ が成り立つ. したがって, この場合, $\lambda(b, c) = t_x$ が成り立つ.

(1) もしも, $x=0$ である場合には, $\lambda(b, c) = \top$ となる. したがって, $\lambda(a, \lambda(b, c)) = \lambda(a, \top) = \top$ となる. この場合には, もしも $\lambda(a, b) \neq \top$ が成り立つとしても $\lambda(\lambda(a, b), c) = \top$ となる.

(2) もしも, $1 \leq x$ であり, $\lambda(a, t_x) = t_y$ である場合には, $0 \leq y \leq x$ となる.

(2.1) もしも, $y=0$ である場合には, $\lambda(a, b) = \top$ となる.

(2.2) もしも, $1 \leq y \leq x$ である場合には, $\lambda(a, b) = t_y$ であり, $\lambda(t_y, c) = t_y$ である.

このように, (1), (2.1), (2.2) の全ての場合において, $\lambda(a, \lambda(b, c)) = \lambda(\lambda(a, b), c)$ が成り立つ. $\square$

定義 22 ($\mathbf{C}^*$ 上の最殊共通祖先) 2つのクラス $s, t \in \mathbf{C}^*$ に対して $\mathbf{C}$ 上の場合と同様に, あるクラス $u \in \mathbf{C}^*$ で次の条件を満足するものを, s と t の最殊共通祖先と言い, $\Lambda(s, t) = u$ と書く.

$$\begin{cases} u \succeq s, u \succeq t, \\ u' \succeq s, t \Rightarrow u' \succeq u \text{ for } \forall u' \in \mathbf{C}^*. \end{cases}$$

$\square$

2つのクラス $s = (s_1, \dots, s_m)$ と $t = (t_1, \dots, t_n)$ ($0 \leq n \leq m$) に対して, 次に示す手続き, 手続き 7 を考える.

手続き 7

1. l を $\lambda(s_l, t_l) = \top$ となる最小の値とする. もしも, 全ての $i = 1, \dots, n$ に対して $\lambda(s_i, t_i) \neq \top$ となる場合には, $l := 0$ とする. そうでない場合には $1 \leq l \leq n$ である.

2. $1 \leq l$ の場合, 個々の $i = 1, \dots, l-1$ に対して $u_i := \lambda(s_i, t_i)$ とする. そして,
 $u := (u_1, \dots, u_{l-1})$ とする.
3. $l = 0$ の場合, $u := ()$ とする.

□

この手続きはいつも成功する. 命題 8 は, この手続きが与える u が a と b の最殊共通祖先であることを保証する. 命題 8 の中では, 手続き 7 の中で使用した記号や記法をそのまま使用する.

命題 8 $u = \Lambda(s, t)$.

証明.

[最殊祖先] ある $v = (v_1, \dots, v_k) \in \mathbf{C}^*$ ($k \leq l-1$) に対して $v \succeq s, t$ が成り立つものと仮定する. 任意の $i = 1, \dots, k$ に対して $v_i \xrightarrow{*} c_i$ かつ $v_i \xrightarrow{*} d_i$ が成り立つ. 定義 21 に依れば, $u_i = \lambda(s_i, t_i)$ は s_i と t_i の共通な祖先のうちで, 最も特殊なものである. すなわち, $v_i \xrightarrow{*} u_i$ が成り立つ. したがって, $u \succeq v$ が成り立つ. もしも, $l-1 < k$ の場合には, $v_l \xrightarrow{*} s_l, t_l$ を満足する v_l は存在しない. したがって, $v \succeq c$ かつ $v \succeq d$ を満足するクラス $v \in \mathbf{C}^*$ は存在しない.

[共通祖先] 定義 21 に依る.

□

命題 9 2つのクラスの最殊共通祖先は, もしも存在するならば一意である.

証明. 命題 2 の証明と同じである.

□

命題 9 と命題 8 に依り, 2つのクラスの最殊共通祖先を常に一意に得ることができる.

定義 23 ($\mathcal{A}$ 上の最殊共通祖先) 以下に示すつの原子対象 $A, B \in \mathcal{A}$ が与えられているとする:

$$\begin{cases} A = \langle X, c, (a_1, \dots, a_m) \rangle \\ B = \langle Y, d, (b_1, \dots, b_n) \rangle \quad (0 \leq n \leq m) \end{cases}$$

このとき A と B の最殊共通祖先を $A \sqcap B$ と書き、以下のように再帰的に定義する:

$$A \sqcap B = \langle Z, \Lambda(c, d), (a_1 \sqcap b_1, \dots, a_n \sqcap b_n) \rangle.$$

ここで、変数 $Z \in \mathbf{V}$ は新しい変数である。 □

命題 10 $A \sqcap (B \sqcap C) = (A \sqcap B) \sqcap C$.

証明. 命題 7 と 命題 8, および定義 23 に依る。 □

原子対象の集合 $A = \{A_1, \dots, A_n\}$ ($A_i \in \mathcal{A}, i = 1, \dots, n$) に対して、 $\sqcap A$ は $A_1 \sqcap \dots \sqcap A_n$ を表わす。命題 10 に依り、 $\sqcap A$ は結合法則を満足し、一意に得ることができる。

14.2.2 頭部から体部へ: SPSHtoB

ある表現 $P = \{C_1, \dots, C_M\}$ ($0 \leq M$), $C_i = (H_i \leftarrow B_{i,1}, \dots, B_{i,n_i})$ ($0 \leq n_i$) が与えられたとする。 $H = \{H_1, \dots, H_M\}$ とおく。 B を表現 P 中の体部に存在する全ての原子対象の集合とする。 B の要素を、順番に番号付けし、 $B = \{B_1, \dots, B_N\}$ ($N = \sum_{i=1}^M n_i$) とする。

個々の原子対象 B_i に対して、SPSHtoB は

- (1) B_i と単一化可能な頭部の最殊共通祖先を計算する。
- (2) 求めた最殊共通祖先と、各 B_i を単一化し、最殊共通祖先から B_i へ情報を伝播させる。この結果、 B_i はより詳細な情報を持つようになる可能性がある。

この手続きは、与えられた表現 P に対して、形式的に書くと、次のようになる:

1. 個々の $i = 1, \dots, M$ に対して、体部の原子対象 B_i と単一化可能な頭部を集める。このような頭部を全て集めてできる集合を $\check{H}_i$ とする。
2. $h = \sqcap \check{H}_i$ を計算し求める。
3. h と B_i を単一化する。

14.2.3 体部から頭部へ: SPSBtoH

SPSBtoH 手続きは、明らかに、最初に与えられた表現の意味を等価に保たない。なぜなら、表現を計算する場合には、何らかの問い合わせが与えられ、問い合わせは1つの体部のみからなる原子対象であり、これを考慮していないからである。

したがって、SPSBtoH 手続きを、等価変換として利用することはできない。

14.3 型推論を適用する例

本節では、append 節の表現を例として、これに SPS 手続きを適用する。最初に、

$$\begin{aligned} \mathbf{C} &= \{\text{append}, \text{list}, \text{cons}, \text{nil}\}, \\ \rightarrow &= \{\text{list} \rightarrow \text{cons}, \text{list} \rightarrow \text{nil}\}, \\ \mathbf{V} &= \{A, \dots, Z\} \end{aligned}$$

が与えられているものとする。

append 節の表現は、式 14.1 となる。

$$\begin{aligned} &[(\text{append}) [(\text{nil})] [X \ ()] X] \leftarrow. \\ &[(\text{append}) [(\text{cons}) [E \ ()] [X \ ()]] \\ &\quad [Y \ ()] \\ &\quad [(\text{cons}) E [Z \ ()]] \leftarrow \\ &[(\text{append}) X Y Z] . \end{aligned} \tag{14.1}$$

式 14.1 で与えられた表現に対して、頭部にある原子対象の集合 H は

$$\begin{aligned} H &= \{[(\text{append}) [(\text{nil})] X:[\ ()] X], \\ &\quad [(\text{append}) [(\text{cons}) E:[\ ()] X:[\ ()]] \\ &\quad Y:[\ ()] \\ &\quad [(\text{cons}) E Z:[\ ()]]]\} \\ &= \{H_1, H_2\} \end{aligned}$$

である。体部にある原子論理式の集合 B は

$$B = \{[(\text{append}) X:[\ ()] Y:[\ ()] Z:[\ ()]]\}$$

である.

[SPSHtoB の適用]

1. B の要素を 1 つ選ぶ. 今回は B の要素は $B_{2,1}$ のみであるので, これを選ぶ.
2. H_1 と H_2 は $B_{2,1}$ と単一化可能である. $h = \sqcap\{H_1, H_2\}$ を計算する. $h = [(\text{append}) [(\text{list})] [()] [()]]$ が求まる.
3. h と $B_{2,1}$ を単一化する. その結果 $B_{2,1}$ は

$$B'_{2,1} = [(\text{append}) \text{ X:} [(\text{list})] \text{ Y:} [()] \text{ Z:} [()]]$$

へと変化する.

このとき, 変数 X に付いているクラス (型) が, もとの $()$ から (list) へと変化 (詳細化) している. $B_{2,1}$ は H_2 を頭部とする節の体部である. $B'_{2,1}$ の変化に伴い, 対応する頭部の H_2 も

$$\begin{aligned} H'_2 = & [(\text{append}) [(\text{cons}) \text{ E:} [()] \text{ X:} [(\text{list})]] \\ & \text{Y:} [()] \\ & [(\text{cons}) \text{ E Z:} [()]]] \end{aligned}$$

へと変化する.

最終的に得られた表現を, 式 14.2 に示す.

$$\begin{aligned} & [(\text{append}) [(\text{nil})] [X \text{ } ()] X] \leftarrow. \\ & [(\text{append}) [(\text{cons}) [E \text{ } ()] [X \text{ } (\text{list})]] \\ & \quad [Y \text{ } ()] \tag{14.2} \\ & [(\text{cons}) E [Z \text{ } ()]] \leftarrow \\ & [(\text{append}) [X \text{ } (\text{list})] Y Z]. \end{aligned}$$

SPSHtoB 手続きを複数回適用すると, 更にクラスの情報が詳細化される可能性がある. しかし, 今回の例では, これ以上の詳細化はない.

14.4 本章のまとめ

本章では、LHS の宣言的記述に対する型推論の手続きを示した。この手続きは、確定節の頭部にある情報と体部にあるクラスの情報を最殊共通子孫の向きへ特殊化する。これにより、遍在していたクラスの情報が平均化され、その過程で、型が特殊化される可能性がある。

この意味で、SPS 変換は宣言的記述を、ある程度までしか特殊化できない。本章で示した型推論の手続きは、問題を計算する前の段階の、前処理のような場面で効果を発揮する。

第15章 等価変換の例

15.1 はじめに

この例は、文献 [44](小林, 1990) から着想を得ている。また、文献 [13] と関連が深い。

11 章と 12 章で定式化した LHS を、本章では、一次方程式の処理に応用する。

一次方程式を解くためには、「両辺に同じ数を加えても、式の意味は変化しない」などの、通常、初等的な数学で用いている、数式を処理するための手続きを利用し、解を導く。

本章では、一次方程式を LHS の原子対象として表現する。また、一次方程式を解くための規則を、原子対象として表現する。原子対象は特殊化システムにより基礎付けられている。したがって、一次方程式のための表現と計算の枠組は、等価変換パラダイムで基礎付けられる。

15.2 定数の階層関係

一次方程式を表現するためには、等号、変数、数、演算子を概念として扱う必要がある。また、これらを組み合わせた、方程式という概念を表現する必要がある。割り算のためには、0でない数、という概念も必要である。

これらの考察より導かれる、一次方程式を表現するための、定数の階層関係と、構造制約は図 15.1 のようになる。

図 15.1 中で、nonzero は 0 でない数を意味している。これを列挙することはできないので「…」で表わしている。0 以外の全ての実数を nonzero の子として許す。aopr は加算 (+) と減算 (-) の演算子をまとめて表現している。topr も同様


```

any → = | opr | expr | equat
opr → aopr | topr
aopr → + | -
topr → × | ÷
expr → num | var | c-exp
num → nonzero | 0
nonzero → ...
var → a | ... | z

c-exp == [(expr)] ⊕ [(opr)] ⊕ [(expr)]
equat == [(expr)] ⊕ [(=)] ⊕ [(expr)]

```

図 15.1: 一次方程式のための定数の階層関係と構造制約

である。

ところで，一般的には，演算子には優先順位が付加されている．原子対象で数式を表現する場合には，自然に，内側の原子対象ほど，強く結び付いているということを表現できている (例えば., 式 (7.2)). したがって，原子対象としては，特に演算子の優先順位を考慮する必要はない．

15.3 原子対象と基底対象

15.2 節で示した，定数の階層関係と構造制約から，クラス，クラス対象，原子対象，基底対象が定まる．例えば “ $3 \times x + 1 = 2 \times x - 3$ ” という一次方程式は，基底対象として，式 (15.1) のように表現できる．

$$\begin{aligned}
&[(\text{equat}) \ [(\text{c-expr}) \ [(\text{c-expr}) \ [(3)] \ [(\times)] \ V:[(x)]] \\
&\quad \quad \quad [(+)] \ [(1)]] \\
&[(=)] \tag{15.1} \\
&[(\text{c-expr}) \ [(\text{c-expr}) \ [(2)] \ [(\times)] \ V] \\
&\quad \quad \quad [(-)] \ [(3)]]]
\end{aligned}$$

これは、全ての定数が、階層関係の底にあるので、基底対象である。また、構造制約を満足している。

式 15.1 では、全てを底にある定数で表現した。今度は、もう少し抽象度を高くして、一般的な式を表現することを考える。例えば任意の実数 a, b, c, d と変数 x に対する “ $a \times x + c = b \times x + d$ ” という式を表現すると、式 15.2 のようになる。

$$\begin{aligned}
&[(\text{equat}) \ [(\text{c-expr}) \ [(\text{c-expr}) \ A:[(\text{num})] \ [(\times)] \ V:[(x)]] \\
&\quad \quad \quad [(+)] \ C:[(\text{num})]] \\
&[(=)] \\
&[(\text{c-expr}) \ [(\text{c-expr}) \ B:[(\text{num})] \ [(\times)] \ V] \\
&\quad \quad \quad [(+)] \ D:[(\text{num})]]] \tag{15.2}
\end{aligned}$$

これは、原子対象である。任意の実数 a は、原子対象 $A:[(\text{num})]$ として表現されている。

15.4 式変形の規則

式 15.1 に示したような一次方程式を解いて、“ $x = -4$ ” を導くためには、原子対象に対する図 15.2 に示すような、変換規則が必要である。

次節では、これらの規則を確定節として表現する。

1. 両辺に、同じ演算を施す.
2. $E: [(expr)] - E \rightarrow 0$
3. $E: [(expr)] \pm 0 \rightarrow E$
4. $E: [(expr)] \div E \rightarrow 1$
5. $E: [(expr)] \times, \div 1 \rightarrow E$
6. 分配法則 $a \times, \div (b \pm c) = (a \times, \div b) \pm (a \times, \div c)$

図 15.2: 式変形の規則

15.5 一次方程式のための表現

等価変換手続きを原子対象で記述し、等価変換規則も問題の表現の中に組み入れることにより、一次方程式を、等価変換パラダイムの上で処理することが可能になる。しかし、等価変換手続きを自動的に適用し、一次方程式を解く手続きを自動的に導く研究は本論文の範囲外である。研究をこの方向へ進めたのが、例えば文献 [44] などである。結論としては、等価変換を大域的に評価する仕組みを導入し、常に回答へ向って計算が進むように保証する必要があることがわかっている。

本節では 15.4 節で直感的に示した式変形の規則を、確定節として表現する。図 15.2 に、確定節として表現された式変形の規則を示す。図 15.2 では、全ての規則が

$[(rule) E: [(equat)] D: [(equat)]] \leftarrow .$

の形をしている。これは、等式 E を等式 D へ変換しても良いことを意味している。それぞれの規則の前に r_n という名前を付けた。

15.6 等価変換の例

図 15.3 に示した，一次方程式を変換する規則に加え，

```
[(path) E:[(equat)] D:[(equat)]
  R:[(list rule)]] ← ...
```

という確定節を用意しておく．これは，一次方程式Eに，変換規則の列Rを適用するとDに変換されることを表わす．この適用の仕組みは本論文の範囲にはない(比較せよ., [13]).

以上のように，変換規則とその適用のための確定節を用意し，これに対して

```
← [(path)
    [(equat) [(c-expr)
              [(c-expr)
                [(3)] [(×)] V:[(x)]]
                [(+) [(1)]]
                [(=)]
                [(c-expr)
                  [(c-expr)
                    [(2)] [(×)] V]
                    [(-)] [(3)]]]]
    [(equat) V E:[(num)]]
    [(list rule)]]
```

という問い合わせを与える．これは，式(15.1)に示した，一次方程式を含む例である．

変換規則(rule)とその適用の方法(path)，問い合わせを全て合せたものが，問題の表現である．これらを等価変換することにより，以下のように答が求まる．ここでは答として変数Eを示す．

```
[(equat) V E:[(-4)]] ←.
```

これは $E = -4$ を意味する．なお，実際に計算を進めるためには，加算などの演算

を実行する仕組みが必要である。本論文では、演算を実行する組み込みの述語 *exec* の存在を仮定する。

具体的でない一次方程式を問い合わせに与えると、その方程式を解くための手続きを得ることができる。例えば、問い合わせとして以下のものを考える。

```
← [(path)
    [(equat) [(c-expr)
               [(c-expr)
                 A: [(num)] [(×)] B: [(expr)]]
                 [(+)] C: [(num)]]
               [(=)]
               [(c-expr)
                 [(c-expr)
                   D: [(num)] [(×)] B]
                 [(+)] E: [(num)]]]]
    [(equat) B F: [(num)]]
    [(list rule)]]
```

これは $ab+c=db+e$ という一次方程式を表現している。

この表現に対する計算の結果は、次のようになる。

```
[(equat) B F] ←
    [(exec) A [(-)] D G: [(nonzero)]],
    [(exec) E [(-)] C H: [(nonzero)]],
    [(exec) H [÷] G F: [(num)]].
```

これは $b=f, f=(e-c) \div (a-d)$ を表わしている。つまり、等価変換の結果、与えられた抽象的な一次方程式を解くための解法が、導かれたことを示す。

15.7 本章のまとめ

本章では、等価変換の例として、一次方程式を処理する例を示した。この例の最後に示したように、一次方程式を処理するために、今回採用した枠組では、抽象的な入力を与えると、それを解くための解法を求める。これは、プログラムを生成する場合の研究に関連が深い。


```

r1:[(rule 1)
    [(equat) E:[(expr)] [(=)] D:[(expr)]]
    [(equat) [(c-expr) E
                R:[(opr)]
                A:[(expr)]]
        [(=)]
        [(c-expr) D R A]]] ←.

r2:[(rule 2)
    [(c-expr) E:[(expr)] [(-)] E]
    [(0)]] ←.

r3:[(rule 3)
    [(c-expr) E:[(expr)] [(aopr)] [(0)]]
    E] ←.

r4:[(rule 4)
    [(c-expr) E:[(nonzero)] [(÷)] E]
    [(1)]] ←.

r5:[(rule 5)
    [(c-expr) E:[(expr)] [(topr)] [(1)]]
    E] ←.

r6:[(rule 6)
    [(c-expr)
        A:[(expr)]
        R:[(topr)]
        [(c-expr)
            B:[(expr)] Q:[(aopr)] C:[(expr)]]]
    [(c-expr)
        [(c-expr) A Q B]
        R
        [(c-expr) A Q C]]] ←.

```

図 15.3: 確定節により表現された等価変換の手続き

第16章 むすび

本論文は、階層関係と部分構造を持つ対象の表現と計算を定式化した。

本研究で言う定式化とは、対象を等価変換パラダイムと特殊化システムの上で把握することである。等価変換パラダイムは、特定の対象や表現系を固定せずに、計算の正当性を広く保証する枠組である。具体的な対象を等価変換パラダイムの上で把握するためには、具体的な表現系を固定する必要がある。本研究では、この表現系に、特殊化システムを採用した。

本研究の重要な成果は以下の2つである。

1. 階層関係と部分構造を持つ対象のための表現系として、具体的な特殊化システムを考案したこと。
2. 階層関係と部分構造を持つ対象を等価変換するために、単一化手続きを考案したこと。

特殊化システムの上で対象を表現するために、原子対象を考案した。さらにクラス対象を導入し、原子対象を拡張した。これらは、定数の順序集合と変数の集合を用いて定義される。原子対象は、論理パラダイムにおける原子論理式概念を包含する。クラス対象を用いることにより、いわゆる高階な表現が可能になった。

本研究では、等価変換手続きとして Unfold 変換を採用した。このため、特殊化システムに基づいて、原子対象とクラス対象の単一化手続きを示した。また、この手続きが単一化であること、および、有限の時間で終了することを示した。

結論として、階層関係と部分構造を持つ対象の表現と計算を等価変換パラダイムの上で定式化することに成功した。

謝辞

本研究は、著者が苫小牧工業高等専門学校に勤務しながら、北海道大学の社会人博士課程に入学して、進められました。社会人博士課程における担当教官を快諾して下さった宮本 衛市教授(北海道大学 工学部 情報システム工学専攻)に感謝いたします。

本研究について、常に暖かく見守り、また、ときに厳しい叱咤激励を下された、赤間 清教授(北海道大学 情報メディア教育研究総合センター)に、感謝いたします。赤間先生からは、本研究に関して、たくさんの有益な示唆や助言をいただきました。また、本研究に関し、助言や示唆を与えていただいた、北海道大学情報システム工学専攻複雑系工学講座の教授の方々に感謝いたします。

本論文を執筆するにあたり、様々な助言や有益な示唆を与えてくれ、また、執筆を励ましてくれた、小栗栖 修君(金沢大学 理学部 計算科学科)、後藤 謙太郎君(北海道大学 理学部 数学科)に感謝いたします。

最後に、社会人入学を許可して下さった苫小牧工業高等専門学校の校長先生、また情報工学科の同僚に感謝いたします。

参考文献

- [1] Hassan Aït-Kaci. An introduction to LIFE – programming with logic, inheritance, functions, and equations. Technical report, Digital Equipment Corporation Paris Research Laboratory, 1993. <http://www.isg.sfu.ca/ftp/pub/hak/-publish/ilps93-life.ps.Z>.
- [2] Hassan Aït-Kaci and et.al. *The Wild LIFE Handbook*. Digital Equipment Corporation, Paris Research Laboratory, prepublication edition edition, 1999. <http://www.isg.sfu.ca/life/>.
- [3] Hassan Aït-Kaci and Roger Nasr. LOGIN: A logic programming language with built-in inheritance. *The Journal of Logic Programming*, Vol. 3, pp. 185 – 215, 1986.
- [4] 赤間清. PAL: 継承階層を扱う拡張 Prolog. 情報処理学会論文集, Vol. 28, No. 4, pp. 322 – 329, 1987.
- [5] 赤間清. 拡張されたロジック・プログラムの理論 I. 人工知能学会研究会資料 SIG-FAI-8803-3 (12/8), 人工知能学会, 1988.
- [6] 赤間清. 媒介表現系上の論理プログラムの宣言的意味論. コンピュータソフトウェア, Vol. 10, No. 1, pp. 3 – 19, 1990.
- [7] 赤間清, 川口雄一, 宮本衛市. マルチセット領域の特殊化システムと等式制約の等価変換. 電子情報通信学会技術研究報告 SS97-55, 電子情報通信学会, 1月 1998.

- [8] 赤間清, 川口雄一, 宮本衛市. マルチセット領域上の等式制約の等価変換. 人工知能学会誌, Vol. 13, No. 3, pp. 395 – 403, 5 月 1998.
- [9] 赤間清, 川口雄一, 宮本衛市. 項領域における包含制約の等価変換. 人工知能学会誌, Vol. 13, No. 2, pp. 274 – 282, 3 月 1998.
- [10] 赤間清, 川口雄一, 宮本衛市. 論理的問題の等価変換による解法 (2) – SLD 導出の限界. 人工知能学会誌, Vol. 13, No. 6, pp. 936 – 943, 11 月 1998.
- [11] 赤間清, 野村祐士, 宮本衛市. プログラム変換による自然言語の意味解釈. コンピュータソフトウェア, Vol. 12, No. 5, 1995.
- [12] 赤間清, 繁田良則, 宮本衛市. 論理的問題の等価変換による解法 (1) – その理論的基礎. 人工知能学会誌, Vol. 13, No. 6, pp. 928 – 935, 1998.
- [13] 秋元亨介. 8 パズルの解法. Technical report, 北海道大学 工学部 複雑系講座 表現系分野, 12 月 7 日 1999. ゼミ資料.
- [14] R. バード. プログラム理論入門, 情報処理シリーズ, 第 3 巻. 培風館, 1981.
- [15] R. バード, P. ワドラー. 関数プログラミング. 近代科学社, 1991.
- [16] Bob Carpenter. *The Logic of Typed Feature Structures*, Vol. 32 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1992.
- [17] Weidong Chen, Michael Kifer, and David S. Warren. HiLog: A first-order semantics for higher-order logic programming constructs. In E.L. Lusk and R.A. Overbeek, editors, *Proceedings of the North American Conference on Logic Programming*, pp. 1090 – 1114, Cleveland, Ohio, 1989.
- [18] E.W. ダイクストラ. プログラミング原論, サイエンスライブラリ 情報電算機, 第 42 巻. サイエンス社, 1983.
- [19] E.W. ダイクストラ, C.A.R. ホーア, O.-J. ダール. 構造化プログラミング, サイエンスライブラリ 情報電算機, 第 32 巻. サイエンス社, 1975.

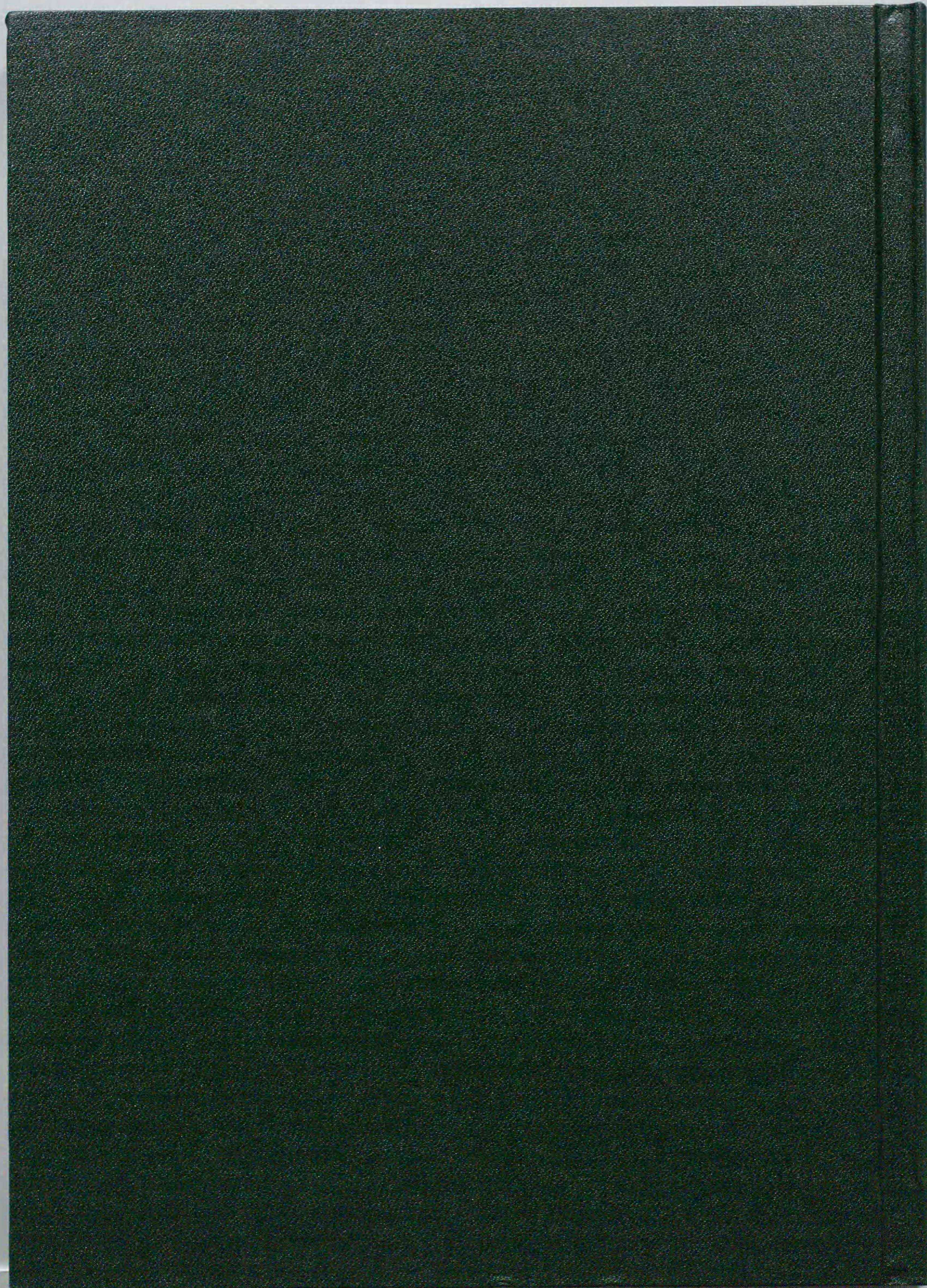
- [20] Martin C. Emele. TFS the typed feature structure representation formalism. In *Proceedings of the International Workshop on Sharable Natural Language Resources*, august 10-11 1994. <http://www2.ims.uni-stuttgart.de/~emele/TFS.html>.
- [21] Martin Emele and Rémi Zajac. Typed unification grammars. In *Proceedings of COLING '90*, pp. 293 – 298, 1990.
- [22] 萩谷昌己. 高階単一化と証明の一般化. 人工知能学会誌, Vol. 6, No. 3, pp. 78 – 86, 5 月 1991.
- [23] 萩谷昌己. ソフトウェア科学のための論理学, 岩波講座ソフトウェア科学, 第 11 卷. 岩波書店, 1994.
- [24] Michael Hanus. Logic programming with type specifications. In Frank Pfenning, editor, *Types in Logic Programming*, Logic Programming Series, pp. 91 – 140. The MIT Press, 1992.
- [25] 原尾政輝, 岩沼宏治. 高階ユニフィケーションアルゴリズムの複雑さについて. コンピュータソフトウェア, Vol. 8, No. 1, pp. 41 – 53, 1989.
- [26] 井田哲雄. 計算モデルの基礎理論, 岩波講座 ソフトウェア科学, 第 12 卷. 岩波書店, 1991.
- [27] 井田哲雄. プログラミング言語の新潮流, 計算機科学/ソフトウェア技術講座, 第 2 卷. 共立出版株式会社, 1988.
- [28] 川口雄一, 赤間清, 宮本衛市. 型付き宣言型言語の UL/α への埋め込み. 日本ソフトウェア科学会第 9 会大会論文集, Vol. A4, No. 4, pp. 269 – 272, 1992.
- [29] 川口雄一, 赤間清, 宮本衛市. GLP の理論に基づいた型付き論理型言語. 日本ソフトウェア科学会第 10 会大会論文集, Vol. B7-2, pp. 205 – 208, 1993.

- [30] 川口雄一, 赤間清, 宮本衛市. UL/ α と型. 苫小牧工業高等専門学校紀要 28, 苫小牧工業高等専門学校, 3月 1993.
- [31] 川口雄一, 赤間清, 宮本衛市. 宣言的プログラムの理論に基づく型付き論理型言語. 電子情報通信学会技術研究報告 SS94-16, 電子情報通信学会, 1994.
- [32] 川口雄一, 赤間清, 宮本衛市. 宣言型計算モデルに基づく、クラスと部分構造を持つ対象の表現と計算. 電子情報通信学会技術研究報告 378, 電子情報通信学会, 1995.
- [33] 川口雄一, 赤間清, 宮本衛市. 単一化に基づく単純で強力な型計算体系. 苫小牧高専情報工学科教育研究報告 jerr950609.01, 苫小牧工業高等専門学校情報工学科, 6月 1995.
- [34] 川口雄一, 赤間清, 宮本衛市. クラスと部分構造を持つ対象を扱う、論理に基づく単純な枠組の提案. 苫小牧工業高等専門学校紀要 31, 苫小牧工業高等専門学校, 3月 1996.
- [35] 川口雄一, 赤間清, 宮本衛市. 階層や部分構造を扱う論理型プログラムに対する、宣言的計算モデルの理論に基づく意味論の構築. 日本ソフトウェア科学会第 13 会大会論文集, Vol. 27-11-D1, pp. 369 – 372, 9月 1996.
- [36] 川口雄一, 赤間清, 宮本衛市. クラスと部分構造を持つ対象の表現と計算 — 論理に基づいた単純な枠組の提案 —. 人工知能学会誌, Vol. 12, No. 1, pp. 48 – 57, 1月 1997.
- [37] 川口雄一, 赤間清, 宮本衛市. 階層や部分構造を扱う宣言型言語に対する意味論の構築. Technical Report jerr970604.01, 苫小牧工業高等専門学校 情報工学科, yuuichi@jo.tomakomai-ct.ac.jp, 6月 1997.
- [38] Yuuichi Kawaguchi, Kiyoshi Akama, and Eiichi Miyamoto. Applying program transformation to type inference on a logic language. *IEICE Trans. on Inf. & Syst.*, Vol. E81-D, No. 11, pp. 1141 – 1147, November 1998.

- [39] 川口雄一, 赤間清, 宮本衛市. マルチセット領域の宣言的プログラムの基礎的な等価変換. 電子情報通信学会技術研究報告 630, 電子情報通信学会, 3 月 1998.
- [40] 川口雄一, 赤間清, 宮本衛市. マルチセット領域の特殊化システムと等式制約の等価変換. 電子情報通信学会技術研究報告 522, 電子情報通信学会, 1 月 1998.
- [41] Yuuichi Kawaguchi, Kiyoshi Akama, and Eiichi Miyamoto. Two basic objects for higher-order expression. In *Eighteenth IASTED International Conference: Applied Informatics (Innsbruck, Austria)*. IASTED, February 14-17 2000. accepted.
- [42] 淵一博, 古川康一, 溝口文雄. プログラム変換, 知識情報処理シリーズ, 第 7 巻. 共立出版株式会社, 1987.
- [43] Michael Kifer and James Wu. A first-order theory of types and polymorphism in logic programming. Technical Report #90/23, Department of Computer Science, SUNY at Stony Brook, July 1990. <http://www.cs.sunysb.edu:80/~kifer/papers.html>.
- [44] 小林貴幸. 問題解決における戦略的知識の学習. 北海道大学大学院 工学研究科 情報工学専攻 修士課程 学位論文, 1990.
- [45] 小暮潔. 素性構造 (1). 人工知能学会誌, Vol. 8, No. 2, 1993.
- [46] 小暮潔. 素性構造 (2). 人工知能学会誌, Vol. 8, No. 3, 1993.
- [47] R.A. コワルスキ. 論理による問題の解法 - Prolog 入門, 情報処理シリーズ, 第 8 巻. 培風館, 1987.
- [48] T. K. Lakshman and Uday S. Reddy. Typed Prolog: A semantic reconstruction of the Mycroft-O'Keefe type system. <ftp://a.cs.uiuc.edu/pub/faculty/reddy/typed-prolog/Types.tar.Z>, 1991.
- [49] John Wylie Lloyd. *Foundations of Logic Programming*. Springer-Verlag, second edition, 1987.

- [50] Kim Marriott and Peter J. Stuckey. *Programming with Constraints: An Introduction*. The MIT Press, 1998.
- [51] Robin Milner, Mads Tofte, Robert Harper, and David MacQueen. *The Definition of Standard ML*. The MIT Press, revised edition, 1997.
- [52] Jhon C. Mitchell. *Foundations for Programming Languages*. Foundations of Computing. The MIT Press, 1996.
- [53] Gopalan Nadathur and Dale Miller. An overview of λ Prolog. In R. Kowalski and K. Bowen, editors, *Proceedings of the Logic Programming: Fifth International Conference on Logic Programming*, pp. 810 – 827. The MIT Press, 1988.
- [54] Gopalan Nadathur and Frank Pfenning. The type system of a higher-order logic programming language. In Frank Pfenning, editor, *Types in Logic Programming*, Logic Programming Series, chapter 9, pp. 245 – 283. The MIT Press, 1992.
- [55] Gopalan Nadathur and Dale Miller. Higher-order logic programming. Technical report, Computer Science Department, Duke University, 1995. `ftp://ftp.cis.upenn.edu/pub/papers/miller/HandbookLogicAILP.dvi.Z`.
- [56] 野木兼六. 構造帰納法に基づくアルゴリズムの発見. コンピュータソフトウェア, Vol. 7, No. 4, pp. 39 – 59, 10 月 1990.
- [57] 野木兼六. ソーティングアルゴリズムの新しい分類. コンピュータソフトウェア, Vol. 9, No. 4, pp. 53 – 72, 7 月 1992.
- [58] 大堀淳. プログラミング言語の基礎理論, 情報数学講座, 第 9 卷. 共立出版株式会社, 1997.
- [59] 繁田良則, 赤間清, 宮本衛市. 関数型言語から論理型言語への変換について. 日本ソフトウェア科学会第 8 会大会論文集, Vol. D3, No. 3, pp. 281 – 284, 1991.

- [60] 高橋正子. 計算論, コンピュータサイエンス 大学講座, 第 24 卷. 近代科学社, 1996.
- [61] 塚本昌彦, 西尾章治郎, 長谷川利治. Dot 記法と is-a 関係にもとづく項表現を用いた演繹・オブジェクト指向データベースモデル. コンピュータソフトウェア, Vol. 9, No. 1, pp. 10 – 26, 1992.
- [62] 渡辺慎哉, 赤間清, 宮本衛市. オブジェクト指向型言語から論理型言語への変換について. 日本ソフトウェア科学会第 8 会大会論文集, Vol. D3, No. 4, pp. 285 – 288, 1991.
- [63] 渡辺慎哉, 赤間清, 宮本衛市. GLP から見た並行計算系. 日本ソフトウェア科学会第 9 会大会論文集, Vol. A5, No. 1, pp. 349 – 352, 1992.
- [64] Eyal Yardeni, Thom Frühwirth, and Ehud Shapiro. Polymorphically typed logic programs. In Frank Pfenning, editor, *Types in Logic Programming*, Logic Programming Series, chapter 2, pp. 63 – 90. The MIT Press, 1992.

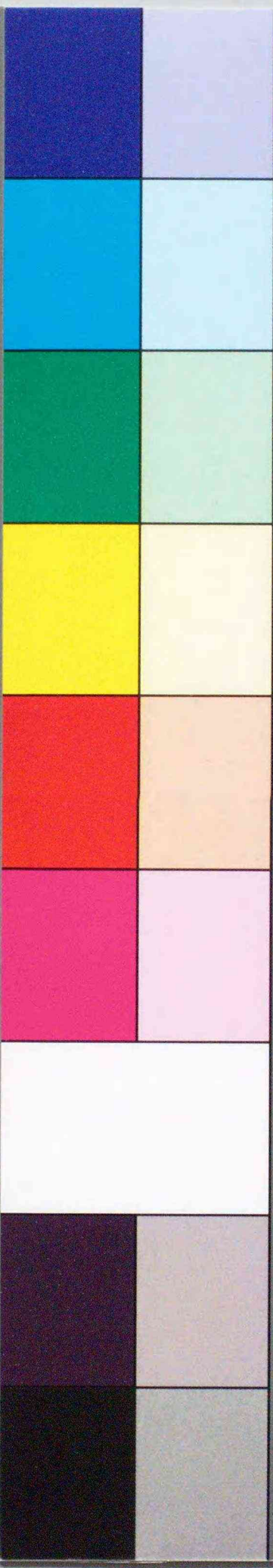


Inches 1 2 3 4 5 6 7 8
cm 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

Kodak Color Control Patches

© Kodak, 2007 TM: Kodak

Blue Cyan Green Yellow Red Magenta White 3/Color Black



Kodak Gray Scale



© Kodak, 2007 TM: Kodak

A 1 2 3 4 5 6 M 8 9 10 11 12 13 14 15 B 17 18 19

