双方向論理と推定代入
(Bidirectional Logic and Inferred Assignment)

進藤　啓介
SHINDO Keisuke
kshindo999@gmail.com

August 23, 2025

Abstract

この論文では論理代数の拡張として、逆論理演算、概念化 (conceptualization)、推定代入 (inferred assignment)、論理因果演算子を新たに導入する。これらを総称して双方向論理と呼んでいる。特に、逆論理演算の結果を推定代入に用いるという単純なアイディアを用いることで、今までにない論理代数の手法が得られる。この手法により、論理式の階層的な制御の全体を１つの巨大な双方向論理ネットワークとして形成することができる。

双方向論理ネットワークの内部を任意の方向に伝搬するために、前提コンテキスト (precondition context) を導入する。ノード上のコンテキストの関連を用いることで、ネットワークの任意のノード間の関連を一意に推定することが可能になる。その結果、推定代入の対象が正しく選択される。さらにはネットワークのノード間のリンク形成、ネットワークの矛盾検出、条件形成を体系的に実行できる。

In this paper, we introduce new extensions to logical algebra: inverse logical operations, conceptualization, inferred assignment, and logical causal operators. These are collectively called bidirectional logic. In particular, the simple idea of using the results of inverse logical operations for inferred assignment provides a novel logical algebraic technique. This technique makes it possible to form the entire hierarchical control of logical expressions as a single giant bidirectional logical network.

We introduce a precondition context to propMate in any direction within a bidirectional logic network. By using the contextual relationships on nodes, we can uniquely infer relationships between any nodes in the network. As a result, the target of inferred substitution is correctly selected. Furthermore, we can systematically form links between network nodes, detect network contradictions, and form conditions.

Keywords: logical algebra, bidirectional logic, inferred assignment, logical causal operator

Contents

1　双方向論理と推定代入

1.1　論理演算の逆演算

論理演算は、例えば以下のような式で表される。これは論理積であり、値の関係を示す真理値表は以下のようになる。

$$A \wedge B = C \tag{1}$$

A	B	C
true	true	true
true	false	false
false	true	false
false	false	false

双方向論理は、論理演算の逆方向の演算を仮定する。たとえば乗算の逆演算が除算であるように、逆演算がある程度定義できる演算がある。この式をあえて C から逆演算することで以下の式になると想定される。論理記号 $\wedge^{-1}$ は、本論文で定義する論理積演算の逆演算である。

$$C \wedge^{-1} A = B \tag{2}$$

C	A	B
true	true	true
true	false	-
false	true	false
false	false	true?false?

論理積演算の逆演算の $\wedge^{-1}$ の結果は、$C = true$、$A = false$ の時は定義できない。更に、$C = true$、$A = false$ の時の B の値は、$true$ なのか $false$ なのかを特定できない。この不確定さが理由で、論理積演算の逆演算は無意味とされて論理演算の対象から除外されてきたが、この逆演算の結果には重要な意味が別にあることを示す。

1.2　推定代入 (inferred assignment)

まずは、以下のような 2 つの論理式を考える。前者は具体的な人物”John” であるが、後者は具体的な人物ではなく、人物に相当する変数 X に対して定義される論理式である。今後は”John” を実体概念と呼び、変数 X を変数概念と呼ぶことにする。

$$\text{John} \wedge \text{hasa} = \text{pen} \tag{3}$$

$$X \wedge \text{hasa} = \text{pen} \tag{4}$$

後者の式を逆演算を用いて変換する。

$$X = \text{pen} \wedge^{-1} \text{hasa} \tag{5}$$

この式に最初の論理式を代入する。

$$X = (\text{John} \wedge \text{hasa}) \wedge^{-1} \text{hasa} \tag{6}$$

この結果、X を”John” と同一視することが可能で、代入が可能であるとみなす。

$$X = \text{John} \tag{7}$$

つまり、論理式を介した変数概念への代入を可能にし、“pen” を持つ人に対する一般論へと展開することができる。例えば以下のように、X はペンを持っていてかつ字を書けると類推した論理式がある。

$$X \wedge \text{hasa} = \text{pen} \tag{8}$$

$$X \wedge \text{can} = \text{write} \tag{9}$$

この式の X に”John” を代入した結果、以下の文が類推される。

$$\text{John} \wedge \text{can} = \text{write} \tag{10}$$

これが推定代入である。推定である理由は、論理式の記述では”John” は”hasa” と”pen” で記述されている部分以外は不確定であるためである。それにも関わらず、部分的な論理式の一致だけで全体の論理式が一致するとみなす。すなわち、“John” が X に包含される可能性を意図的に選択して代入するということである。

$$\text{John} \subset X \tag{11}$$

この推定代入のルールを一般化すると、

- **代入元概念の論理式（の一部）が、代入先変数概念の論理式の一部と一致するならば、代入元の概念が代入先の変数概念全体と包含関係になると推定する。**

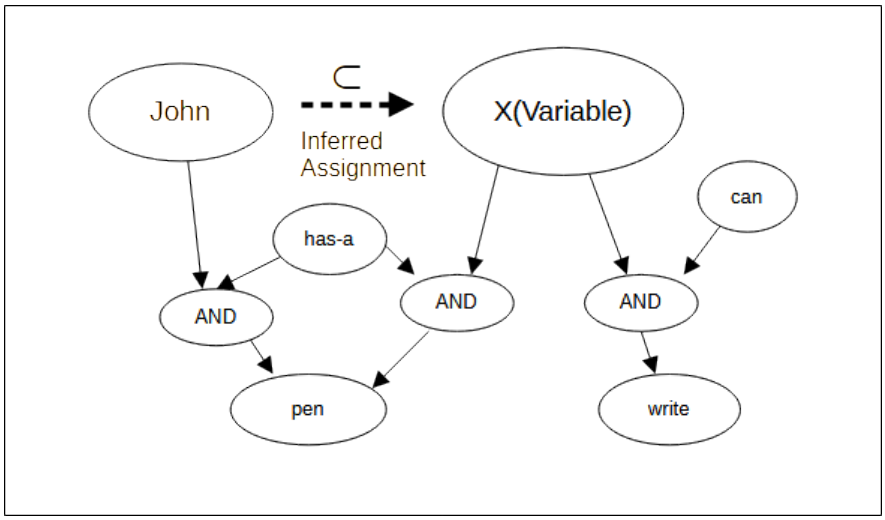

Figure 1: 推定代入 (inferred assignment)

- **この包含関係の推定が常に成立する論理式を選択し、一般法則と見なす。**

このルールを認めることにより、論理式の表現対象が飛躍的に増大する。

1.3　概念と属性（プロパティ）

双方向論理で導入した「概念 (Concept)」とは、複数の属性を持ち、この属性だけで特徴づけられる不確定な集合である。この属性を論理式で記述する方法を示す。まず、概念 (Keith) の持つ属性のキー（name など) と概念とで論理積を行う。論理積の結果と、属性の値 (名前”Keith” など）の間で含意の関係が形成される。含意を用いるのは、値が他の概念でも利用されるためである。

$$Keith \wedge \text{name} \rightarrow \text{”}\ \text{Keith”} \tag{12}$$

$$Keith \wedge \text{familyname} \rightarrow \text{”Emerson”} \tag{13}$$

$$Keith \wedge \text{plays} \rightarrow \text{keyboard} \tag{14}$$

この論文では、「概念」を以下のように定義する。

- **概念=属性の論理式を全て満たすことだけを条件とする抽象的な集合**
- **概念は属性の集合を内包するが、属性以外の集合の大きさは未確定**

この概念を用いた論理式を変数化して、論理式を一般化することを考える。X はここからは変数概念と呼ぶ。変数概念は人物や環境などのあらゆる実体概念を推定代入できる。

$$X \wedge \text{plays} \rightarrow \text{keyboard} \tag{15}$$

$$X \wedge \text{composes} \rightarrow \text{music} \tag{16}$$

$Keith$ の持つ属性の一部を変数概念 X に対して適用すると、$Keith$ が X に代入される。その結果以下が類推される。$Keith$ は属性キー $composes$ との集合関係は未確定のはずだが、概念の集合は明示的に排他的でない限りは必ず重複があると見なす。

$$Keith \wedge \text{composes} \rightarrow \text{music} \tag{17}$$

以上のように、概念と推定代入とを用いれば、部分的な論理式の一致結果を全体の集合に拡大することができる。以下にそれぞれの用語を整理する。

概念：Keith , X

属性キー：name, familyname, plays, composes

属性値：“Keith”,“Emerson”, keyboard ,music

1.4　推定代入の成立条件

推定代入は、代入元の概念が代入先の概念と部分一致するだけで必ず実行される。ただしその推定代入の結果が正しいかどうかは、代入先の概念を用いた論理式で評価される。たとえば以下の例を挙げる。

$$G_1 \wedge can \rightarrow drive \tag{18}$$

$$G_1 \wedge hasa \rightarrow car \tag{19}$$

適用する変数概念を以下のように定義する。

$$F_A \wedge can \rightarrow drive \tag{20}$$

$$F_B \wedge hasa \rightarrow car \tag{21}$$

$$F_R \wedge goesfora \rightarrow drive \tag{22}$$

この変数概念の間の論理積が推定代入の成立条件である。

$$F_A \wedge F_B = F_R \tag{23}$$

まずは、それぞれの変数概念への推定代入は、概念の持つプロパティが部分一致すれば必ず行われる。

$$F_A \supset G_1 \tag{24}$$

$$F_B \supset G_1 \tag{25}$$

すなわち、推定代入の実行そのものに対して条件を付けるのではなく、推定代入された変数概念に対して論理積などの条件を設ける。論理積の結果が $true$ であれば、概念 G_1 がそのまま変数概念 F_R へと到達する。

$$F_R \supset G_1 \tag{26}$$

$$G_1 \wedge goesfora \rightarrow drive \tag{27}$$

だが、以下の概念 G_2 について言えば、変数概念 F_B が $false$ となるので論理積が $false$ になり、変数概念 F_R もまた $false$ となる。

$$G_2 \wedge can \rightarrow drive \tag{28}$$

$$G_2 \wedge hasa \rightarrow \neg car \tag{29}$$

1.5　属性キーインスタンスと順序表現

属性のキーは、名前や性別など単純なオブジェクトで表現できれば簡単である。だが例えば、B さんは A さんから見ると「次」であるが、C さんから見ると「前」でもあるというケースがある。これを表現するための方法を示す。

例として、属性値 a, b, c の互いの前後関係を表現する。複数の属性を特定するため、$prev_{ab}, next_{ab}$ のような属性キーインスタンスがそのつど生成される。概念と属性キーインスタンスとの論理積演算によって、値 a, b と含意で結ばれる。

$$G_{ab} \wedge prev_{ab} \rightarrow a \tag{30}$$

$$G_{ab} \wedge next_{ab} \rightarrow b \tag{31}$$

すべての属性キーインスタンスは属性キーの種類を示す属性キーノードへと包含連結される。

$$prev_{ab}, prev_{bc} \in previous \tag{32}$$

$$next_{ab}, next_{bc} \in next \tag{33}$$

なぜこのような構造になるかというと、ある G_{ab} では b は $next$ の部分集合であるが、別の G_{bc} では b は $prev$ の部分集合になりうる。そのため、$prev_{ab}$ や $prev_{bc}$ を、1つの $prev$ で代替することはできない。そのために、以下のように $prev_{bc}$ や $next_{bc}$ は論理式ごとにその都度生成される。

$$G_{bc} \wedge prev_{bc} \rightarrow b \tag{34}$$

$$G_{bc} \wedge next_{bc} \rightarrow c \tag{35}$$

こうして、a と b との順序関係と、b と c との順序関係とが、異なる属性キーインスタンスを用いて区別される。

1.6　概念の分割階層化

概念はさらなる上位の概念の部分集合となりうる。分割方法は、条件のグループ化が基本である。下の例では Today という属性は 7am,evening の2つの属性に共有されてグループ化していると考える。

$$E \wedge today = G_{Today} \tag{36}$$

$$E \wedge yesterday = G_{Yesterday} \tag{37}$$

$$G_T \wedge 7am = G_{Today7am} \tag{38}$$

$$G_Y \wedge evening = G_{Todayevening} \tag{39}$$

2　論理因果演算子 (Logical causal operator)

2.1　恒等因果

論理演算の結果が別の論理演算とイコールで結び付けられた結果、不確定であった論理演算の関係が確定するとみなす。このイコールを論理演算の因果関係を示す演算子と考えて、式が成立するときだけ $true$ を返す。これは、プログラミング言語でいうところの条件式が成立した結果、論理値に $true$ を代入する作用と似ている。

$$X = if(A = B) \tag{40}$$

このような恒等式の因果関係を示す演算子を定義して、$\mathcal{C}_{id}$：恒等因果 (Identical Causality) と呼ぶこととする。

$$B = \mathcal{C}_{id}(A, X) \tag{41}$$

この演算子 $\mathcal{C}_{id}$ は、$X = true$ の時は、$A = B$ の関係を持つことが確定する。重要なことは、$X = false$ のときは A と B の値がともに不確定な値を取ることである。既存の排他的論理和演算子 $\oplus$ を用いてこの恒等因果 $\mathcal{C}_{id}$ を定義すると以下のような式になる。

$$X \subset X' \tag{42}$$

$$\mathcal{C}_{id}(A, X) = B = A \oplus \neg X' \tag{43}$$

X' を導入することで、$X = false$ のときは B は $true$ と $false$ の両方の値を取ることが出来るようになる。この X' を用いて排他的論理和を適用すれば、A と B の値がそれぞれ全ての組み合わせを取ることができる。

以下に、恒等因果 $\mathcal{C}_{id}$ の演算子の真理値表を示す。$undef$ は $true$ か $false$ の両方の値を取ることができる。

A	X	B
true	true	true
false	true	false
true	false	undef
false	false	undef

恒等因果は逆演算子が定義される。つまり、A と B との等式が成立しているときに、$\mathcal{C}_{id}^{-1}$ の結果が True になり、Bool 値として X に代入される。

$$X = \mathcal{C}_{id}^{-1}(A, B) \tag{44}$$

恒等因果演算子の逆演算で特徴的なことは、$A = true$ のときと $A = false$ の時の双方を確認しなければ、X が判明しないことである。

1. $X = true$

A	B
true	true
false	false

2. $X = false$

A	B
true	true
false	true

3. $X = false$

A	B
true	false
false	true

4. $X = false$

A	B
true	false
false	false

2.2　含意因果

恒等式の代わりに、含意を示す演算子も別に定義することができる。含意の定義により、$(A \to B)$ は $(A \subset B)$ でもある。

$$X = if(A \to B) \tag{45}$$

このような含意の因果関係を示す演算子を定義して、$\mathcal{C}_{im}$：含意因果 (Implicational Causality) と呼ぶこととする。

$$B = \mathcal{C}_{im}(A, X) \tag{46}$$

この演算子 $\mathcal{C}_{im}$ は、$X = true$ の時は、$A \subset B$ の関係を持つことが確定する。$X = false$ のときは A と B の値がともに不確定な値を取る。既存の論理積演算子 $\wedge$ を用いてこの含意因果 $\mathcal{C}_{im}$ を定義すると以下のような式になる。

$$B' = A \wedge X \tag{47}$$

$$\mathcal{C}_{im}(A, X) = B \supset B' \tag{48}$$

以下に、含意因果 $\mathcal{C}_{im}$ の演算子の真理値表を示す。$undef$ は $true$ か $false$ の両方の値を取ることができる。

A	X	B
true	true	true
false	true	undef
true	false	undef
false	false	undef

この含意因果 $\mathcal{C}_{im}$ もまた、逆演算子が定義される。A が B に包含されるとき（含意)、X が $true$ となる。それ以外の場合は $false$ となる。

$$X = \mathcal{C}_{im}^{-1}(A, B) \tag{49}$$

含意因果 $\mathcal{C}_{im}$ もまた、A が $true$ の時と $false$ の時の双方を確認しなければ、X が判明しない。A の時は必ず B であるときに、$\mathcal{C}_{im}^{-1}$ の結果が True になり、Bool 値として X に代入される。表にすると以下のようになる。

1. $X = true$?

A	B
true	true
false	false

1. $X = true$?

A	B
true	true
false	true

3. $X = false$

A	B
true	false
false	true

4. $X = false$

A	B
true	false
false	false

前項で定義した恒等因果$\mathcal{C}_{id}$と異なる点は、この含意因果$\mathcal{C}_{im}$は論理積演算の逆演算であるために、Xの全領域が$true$になっている保証がないという点である。この問題があるために、Xは推定代入によって確定させるほかはない。そのために、対になる別の概念が必要になり、その概念がこの出力Xへと推定代入される。

2.3　写像の論理式化

論理因果演算子の導入により、写像を論理式だけで記述する準備が出来た。写像の定義域と値域とをこれまで導入してきた概念を用いて記述すると、

$$G_i \wedge input1 = a \tag{50}$$

$$G_o \wedge output = b \tag{51}$$

$$G_o = \mathcal{C}_{id}(G_i, m) \tag{52}$$

こうして、写像 m が $true$ の時に、a から b への写像が成立する。写像 m が $false$ の時は a と b の値はそれぞれ不確定となる。このような概念同士の関係の記述は、従来の論理演算だけでは表現できず、恒等因果 $\mathcal{C}_{id}$ によってはじめて可能になる。たとえば写像 m を論理積 (AND) だけで条件を与えようとすると、写像 $m = false$ のときは b が常に $false$ となり、一般性を失う。

$$G_o \neq G_i \wedge m \tag{53}$$

3　変数と関数

3.1　写像論理式

複数の数値を用いた数式を示す写像を、論理式だけで表現したい。だが、現実には互いに素である数値の集合同士で論理式を形成することはありえない。例えば、以下のような論理式は成立しない。

$$"2" \wedge add \wedge "3" = "5" \tag{54}$$

ところが、双方向論理を用いて数値をそれぞれ概念化すれば、論理式での数式の表現が可能になる。

$$G_2 \wedge i_1 \subset "2" \tag{55}$$

$$G_3 \wedge i_2 \subset "3" \tag{56}$$

$$\mathcal{C}_{id}(G_1 \wedge G_2, \mathrm{add}) = G_R \tag{57}$$

$$G_5 \wedge o_1 \subset "5" \tag{58}$$

このように、数値は概念化すれば論理演算の対象となる。それぞれの数値”2”、“3” は概念 G_2、概念 G_3 にそれぞれ変換され、概念の間で論理積演算が適用されている。論理積を適用する理由は、入力パラメータが両方一致することを論理式の適用条件とするためで、片方だけが一致する場合は論理式の適用対象にしないためである。つまり、この概念間の論理積演算が、推定代入の成立条件である。

次に、写像の適用条件を追加するために、論理演算の結果と写像の結果 s_{add} とを論理因果演算子 $\mathcal{C}_{id}$ で結び付ける。論理因果演算子 $\mathcal{C}_{id}$ の第二パラメータが、写像の内容を示す値 add である。逆に言えば、写像の作用は add の $true, false$ で制御できる。わざわざ論理因果演算子を用いる理由は、add が $false$ の時は、入出力の値がどちらも不確定であると見なすためである。

こうして、数式とその適用を、論理式を用いて記述することが可能になる。

3.2　変数化と関数化

これまでは定数を用いて概念を推定代入することを行ってきた。ここからは、属性が用いる値を変数に変える。変数を用いる写像が関数である。関数はその変数から値を 1 つ以上受

け取って、出力変数に値を代入して返す。入出力の値の間の実際の写像は、関数が持つ内部の写像論理式がそれぞれ行うが、関数の外側からは見えないものとする。

概念が持つそれぞれの値を可変にして変数化することで、関数が生成される。関数は１つの概念が入力と出力とに分けられる。入力は可変ドメイン関数概念 F_D、出力は可変レンジ関数概念 F_R であり、これらを論理因果演算子 $\mathcal{C}_{id}$ で結び付ける。$\mathcal{C}_{id}$ の第二パラメータが関数を示す概念 f_{add} と等価になる。

$$F_{D1} \wedge i_1 = v_1 \tag{59}$$

$$F_{D2} \wedge i_2 = v_2 \tag{60}$$

$$F_R \wedge o_1 = o \tag{61}$$

入力値の概念をそれぞれ変数化すると、概念の持つすべての論理式が同時に満足される保証がなくなる。そのため、関数入力概念 F_{D1}, F_{D2} の間で論理積演算を行うことになる。

$$\mathcal{C}_{id}(F_{D1} \wedge F_2, f_{add}) = F_R \tag{62}$$

3.3　関数の適用

ベクトルのように数値をペアにして管理するには、概念 G_V を用いて以下のように表現する。概念の階層表現を用いればどのような数値やベクトルなどでも表現が可能なはずである。

$$G_V \wedge p_1 = \text{"2"} \tag{63}$$

$$G_V \wedge p_2 = \text{"3"} \tag{64}$$

この G_V を、前項の f_{add} の２つの入力概念 G_{D1},G_{D2} に対して代入することを考える。そのためには以下の式でパラメータの代入が必要になる。

$$p_1 \subset N \supset i_1 \tag{65}$$

$$p_2 \subset N \supset i_2 \tag{66}$$

$$o_1 \subset N \tag{67}$$

i_1,i_2 は集合が未確定なので、p_1,p_2 を、自然数 N を経由して代入することが可能になる。

$$F_{D1} \wedge i_1 = \text{"2"} \tag{68}$$

$$F_{D2} \wedge i_2 = \text{"3"} \tag{69}$$

代入されたパラメータを用いて関数が実行され、結果として F_V に対して o_1 の論理式が追加される。o_1 は N からその都度インスタンスとして形成される。

$$F_R \wedge o_1 = \text{"5"} \tag{70}$$

こうして、関数が適用された結果、入力概念 G_V が拡大されて結果の値が追加される。

$$G_V \wedge o_1 = \text{"5"} \tag{71}$$

3.4　概念を関数へと一般化

概念はすべての属性が同時に観測されたことを示すが、関数は属性の間の因果関係を表現する。

まずは、概念 G_1 ではそれぞれの属性が以下のように観測された。p_1,p_2,p_3 はそれぞれ属性名であり、v_1,v_2,v_3 はそれぞれ属性値である。

$$G_1 \wedge p_1 = v_1 \tag{72}$$

$$G_1 \wedge p_2 = v_2 \tag{73}$$

$$G_1 \wedge p_3 = v_3 \tag{74}$$

これが別の概念 G_2 では、以下のように観測された。

$$G_2 \wedge p_1 \neq v_1 \tag{75}$$

$$G_2 \wedge p_2 = v_2 \tag{76}$$

$$G_2 \wedge p_3 \neq v_3 \tag{77}$$

さらに別の概念 G_3 では、以下のように観測された。すなわち、属性の値の間に依存関係があると判明した。

$$G_3 \wedge p_1 = v_1 \tag{78}$$

$$G_3 \wedge p_2 \neq v_2 \tag{79}$$

$$G_3 \wedge p_3 \neq v_3 \tag{80}$$

その結果、関数が生成される。p_1 と p_2 と p_3 との間には論理式が成立すると推測された。

$$F_a \wedge p_1 = v_1 \tag{81}$$

$$F_b \wedge p_2 = v_2 \tag{82}$$

$$F_a \wedge F_b = F_r \tag{83}$$

$$F_r \wedge p_3 = v_3 \tag{84}$$

さらに、生成された論理式は常に成立するわけではなく、$func$ が関数の動作条件となる。動作条件は因果演算子 $\mathcal{C}_{id}$ を用いて表現できる。

$$F_r = \mathcal{C}_{id}(F_a \wedge F_b, func) \tag{85}$$

3.5　関数パラメータのカリー化

関数の入力パラメータに値を与えるという事は、関数を入力値によって部分抽出して別の関数に変換するという動作と解釈される。この部分抽出は、理論計算機科学におけるラムダ式のカリー化と同じものである。

たとえば以下の関数は、属性キー i_1 の入力変数 x から、属性キー o_1 の出力変数 x へと、同じ値を代入する。

$$F \wedge i_1 = x \tag{86}$$

$$F \wedge o_1 = x \tag{87}$$

この関数に値 v の概念を代入することを考える。

$$G \wedge i_1 = v \tag{88}$$

この関数の入力と出力とを比べると、属性キーが i_1 と o_1 とで異なる。そのため変数 x のコンテキストは、入出力がそれぞれ $x \wedge i_1$ と $x \wedge o_1$ に該当し、集合の重複がない。コンテキストに重複がないということは、値 v の情報を出力側の集合に対して代入で伝達することができない。

そのため、関数 F を値 v でカリー化して、F_v をその都度作り出す。

$$F_v \wedge i_1 = v \tag{89}$$

$$F_v \wedge o_1 = v \tag{90}$$

これにより、これまでの写像と同じように、入力概念 G に出力値 v の論理式が追加できる。

$$G \wedge o_1 = v \tag{91}$$

3.6　関数間の接続

関数間の接続と、さらに上位の関数への接続の例を示す。関数は、写像あるいは別の関数の組み合わせで構成が可能であり、動作条件の判定を設けることで関数の動作制御も可能になる。たとえば、以下の関数を論理式だけで記述することを考える。

$$f_{\mathrm{top}}(v_1, v_2, v_3) = \begin{cases} f_3(v_3, f_1(v_1, v_2)) & \text{if} \quad f_2(v_1) \\ f_1(v_1, v_2) & \text{otherwise} \end{cases} \tag{92}$$

ここで関数の入力パラメータの簡略表現を導入する。関数はパラメータとして入力と出力を並べて記述して、個別のパラメータはパラメータ型: 変数名のペアとする。関数の小文字 f は論理因果演算子を用いて定義される関数オブジェクト、関数の大文字 F は関数概念を示す。

$$f_{top}(i_1 : v_1, i_2 : v_2, i_3 : v_3, o : o_1) \tag{93}$$

この関数は、概念を用いた論理式で個別に記述すると以下のようになる。F_{topD1} は関数写像の定義域 (domain)、F_{topR} は関数写像の値域（range）の関数概念である。

$$F_{topD1} \wedge i_1 = v_1 \tag{94}$$

$$F_{topD2} \wedge i_2 = v_2 \tag{95}$$

$$F_{topD3} \wedge i_3 = v_3 \tag{96}$$

$$F_{topR} \wedge o = o_1 \tag{97}$$

$$\mathcal{C}_{id}\{(F_{topD1} \wedge F_{topD2} \wedge F_{topD3}), f\} = F_{topR} \tag{98}$$

同じようにして、関数 f に含まれる部分関数を定義する。部分関数は関数と同じく変数化された概念である。

$$f_1(i_{1,1} : v_1, i_{1,2} : v_2, o_1 : r_1) \tag{99}$$

$$f_2(i_2 : v_1, o_2 : r_2) \tag{100}$$

$$f_3(i_{3,1} : v_3, i_{3,2} : r_1, o_3 : r_3) \tag{101}$$

これらの部分関数を全体の関数 f に結合する。この部分関数は条件による実行制御を行うことも可能である。

$$f \supset f_1 \tag{102}$$

$$f \supset f_2 \tag{103}$$

$$f \supset f_3 \tag{104}$$

f_1,f_2,f_3 の結果を利用して、それぞれに対応する関数概念 F_{1R}, F_{2R}, F_{3R} に逆演算する。

$$F_{1R} \wedge o_1 = r_1 \tag{105}$$

$$F_{2R} \wedge o_2 = r_2 \tag{106}$$

$$F_{3R} \wedge o_3 = r_3 \tag{107}$$

f_1 の出力と f_3 の入力とは以下のようにして結合される。

$$F_{3D1} \wedge i_{3,2} = r_1 \tag{108}$$

最終的な出力 f_{OR} への代入は、f_2 の結果 f_{2R} を用いた条件判断によって選択制御する。

$$(F_{3R} \wedge F_{2R}) \vee (F_{1R} \wedge \neg F_{2R}) = F_{OR} \tag{109}$$

最終的な結果 F_{OR} を、関数全体の出力 F_{topR} の持つ $o1$ と関数パラメータ型 o とに連結する。

$$F_{OR} \wedge o = o_1 \tag{110}$$

このようにして、関数を階層的に定義することができる。

4　前提コンテキスト (precondition context)

4.1　前提コンテキストによる部分注目

推定代入は概念の変数同士で常に成立するわけではなく、成立するための条件が存在する。その条件を明確にするために、前提コンテキスト (文脈) を新たに提案する。この前提コンテキストは、論理代数に用いられる複数の入力値をそれぞれ$true$か$false$のどちらかに仮定して、その組み合わせを前提とする。今後は誤解が無い限り、簡単のためにコンテキストと呼称する。

この概念G_Aの真偽値に対して、コンテキストを導入する。コンテキストC_{GA}はG_Aの内部でかつ臨時に注目すべき部分集合であり、

$$C_n = \{b = true, c = false\} \tag{111}$$

などの前提を意味する。このコンテキストC_nを前提とするG_Aの確率値が、P_{GA}である。

$$P_{GA}(G_A|b = true, c = false, ...) = P_{GA}(G_A|C_n) = true \tag{112}$$

特に、P_{GA}が$true$の時はコンテキストC_nはG_Aに属すると見なすことができる。

$$G_A \subset C_n \tag{113}$$

G_Aからaを用いて論理積演算を行い、$value$に代入する。

$$G_A \wedge a = value \tag{114}$$

そのために入力が規定されていないaに対して、前提$a = true$と$a = false$が仮定としてそれぞれ追加される。

$$P +_a (a|C +_a \{a = true\}) = true \tag{115}$$

$$P -_a (a|C -_a \{a = false\}) = false \tag{116}$$

その結果、前提ごとに、$value$に代入する２つの論理値$P+_{value}$と$P-_{value}$とが生成される。

$$P +_{value} (value|C +_{value} \{a = true, C_n\}) = true \tag{117}$$

$$P -_{value} (value|C -_{value} \{a = false, C_n\}) = false \tag{118}$$

4.2　推定代入

コンテキストを使用すれば、概念間の推定代入の成立条件が厳密に定義できる。まず、変数概念 G_B を用いた別の式を用意する。通常の G_A と違い、変数である G_B はコンテキストが代入されていないという点が異なる。

$$G_B \wedge a = value \tag{119}$$

この式の逆演算を用いて書き換える。

$$value \wedge^{-1} a = G_B \tag{120}$$

$value$ は G_A と共通なので、$value$ から G_B へ向けてコンテキストの伝搬が行われる。

$$P +_a (a|C +_a \{a = true\}) = true \tag{121}$$

$$P -_a (a|C -_a \{a = false\}) = false \tag{122}$$

$$P +_{GB} (G_B|C_{GB}\{a = true, C_{GA}\}) = true \tag{123}$$

$$P -_{GB} (G_B|C_{GB}\{a = false, C_{GA}\}) = \text{unknown} \tag{124}$$

$value$ からの逆演算で、G_B へと代入される際に、２つの論理値 $P+_{GB}$ と、$P-_{GB}$ とが生成される。ここで、P_{GB} は値が unknown であることが重要になる。

$$P - +_{GB}(G_B|C_{GB}\{a = false, x = true, C_{GA}\}) = true \tag{125}$$

$$P - -_{GB}(G_B|C_{GB}\{a = false, x = false, C_{GA}\}) = false \tag{126}$$

このように、P_{GB} が実際は、未知の集合 $x \subseteq a$ によって $true$ と $false$ にさらに分割されている可能性がある。

ここでオッカムの剃刀の考え方を用いる。オッカムの剃刀の一般的な定義は

「説明に不必要な実体を増やしてはならない」

これを言い換えると、

「矛盾の無い説明の中で、最も仮定が少ないものを選択する」

と考える。x の仮定は実は存在してもしなくても G_B の論理式には矛盾がないため、x の仮定そのものが無いと見なす。これを用いると、x は除去されて

$$P -_{GB} (G_B|C_{GB}\{a = false, C_{GA}\}) = true \tag{127}$$

これだけが推定選択される。その結果、相補的なコンテキストは合成されて、$a = true$ と $a = false$ の仮定は相殺される。

$$P_{GB}(G_B|C_{GA})) = P+_{GB}(G_B|C_{GB}\{a = true, C_{GA}\}) = P-_{GB}(G_B|C_{GB}\{a = false, C_{GA}\}) = true \tag{128}$$

こうして、G_B のコンテキストの集合の大きさが暫定的に決定され、以下のように G_A が G_B にすべて包含されると仮定する。

$$P_{GA}(G_A|b = true, c = false, ...) = P_{GA}(G_A|C_{GA}) = true \tag{129}$$

$$G_A \subset G_B \tag{130}$$

この包含関係はオッカムの剃刀に基づく推定であるが、G_B 側の論理式に対して観測結果との矛盾が検出されなければ、包含関係が常に成立すると見なす。つまり言い換えると、矛盾が観測されない限り、G_B の論理式は推定代入が常に可能な一般的な定理であると見なす。

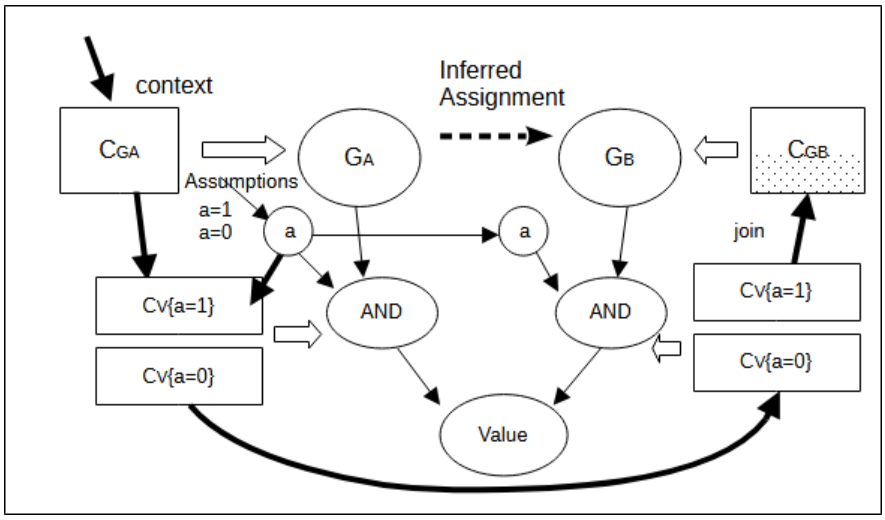

Figure 2: コンテキスト (context)

推定代入は、代入した後に評価される論理式で正否が判定される。この考え方に基づいて、推定代入の一般的なルールを提案する。

- **推定代入はコンテキストが合致すれば常に実行される。**
- **推定代入の結果は代入先の論理式で成立が判定される。**

- **推定代入先の論理式は、代入先の観測によって作成される。あるいは観測によって修正される。**
- **推定代入の成立条件の論理式が全て既知であるという証明はできないが、極めて大量に検証された公理から演繹的に成立条件を導くこともできる。**

4.3　コンテキスト代数

実体概念Gの内外を示す論理値として、コンテキストCを前提とする値Pで表す。この値は、基本的にはコンテキストCを前提とした概念Gの内外の比率を示す伝搬確率値と考えて良い。

$$C_m = \{a = true, b = true..\} \tag{131}$$

$$C_n = \{a = true, b = false..\} \tag{132}$$

$$P_m(G|C_m) = 1 = true \tag{133}$$

$$P_n(G|C_n) = 0 = false \tag{134}$$

表現の便宜のため、論理値を不等号で示すという表記も用いる。論理値は確率そのもので考えるよりも、$true$と$false$の２つの集合が伝搬集合の断面を分断し、確率値でそれらの比率を示すと解釈できる。この解釈では、必要に応じた２つの論理値の分断も可能になる。

$$P_m < 1.0 \Rightarrow P_m < true \tag{135}$$

$$P_n \geq 0 \Rightarrow P_n \geq false \tag{136}$$

通常の論理積での例を挙げる。順方向の論理演算では、コンテキストC_kの非共通前提要素b, c, dは、すべて合成されてC_{n+1}に内包される。共通部分はそのままC_{n+1}へと継承される。

$$G_n = G_k \wedge G_l \tag{137}$$

$$C_k = \{a = true, b = true, c = false, ...\} \qquad C_l = \{a = true, d = true\} \tag{138}$$

$$C_k \wedge C_l = C_n = \{a = true, b = true, c = false, d = true\} \tag{139}$$

4.3.1　順方向コンテキスト合成

順方向の論理演算の際に、２つのコンテキストの内部に相補的な前提要素 ($a = true$ と $a = false$) があれば、コンテキストは空集合となる。

$$C_k = \{a = true, b = true, c = false, ...\} \qquad C_l = \{a = false, d = true\} \tag{140}$$

$$C_k \wedge C_l = \tag{141}$$

このような前提の追加を、コンテキストの合成 (combine) として、$F_{Ccomb}()$ と定義する。

$$C_n = F_{Ccomb}(C_k, C_l) \tag{142}$$

4.3.2　逆方向コンテキスト統合

逆方向の論理演算では、２つのコンテキストの、さらにそれぞれ相補的なコンテキスト２つの計４つのコンテキストを統合する。

$$G_k = G_n \wedge^{-1} G_l \tag{143}$$

合成するコンテキストは以下の 4 通りになる。

$$(C_{n+}, C_{n-}, C_{l+}, C_{l-}) \tag{144}$$

このような前提の統合を、コンテキストの統合 (unification) として、$F_{Cunif}()$ と定義する。

コンテキストの和は、相補的な前提要素を除去する。この例では ($a = true$ と $a = false$) である。

$$C'_n = C_{n+} + C_{n-} \tag{145}$$

$$C'_l = C_{l+} + C_{l-} \tag{146}$$

合成条件は以下の通りで、C'_n と C'_l の相補的な合成結果が等しい場合に限定される。

$$C'_n = C'_l \tag{147}$$

その場合、結果的に以下の等式となる。

$$C_k = F_{Cunif}(C_{n+}, C_{n-}, C_{l+}, C_{l-}) = C'_n = C'_l \tag{148}$$

コンテキストが等しくない場合は、C_k の合成は成立せずに空集合となる。コンテキストの前提要素は、逆演算の推定代入によってのみ減らすことができる。

4.4　コンテキスト論理演算

論理演算の入力２つに対応する、相補的な論理値とコンテキストとを以下のように定義する。

$$P_{k+}(G_{k+}|C_{k+}) \quad C_{k-}(G_{k-}|C_{k-}) \tag{149}$$

$$P_{l+}(G_{l+}|C_{l+}) \quad C_{l-}(G_{l-}|C_{l-}) \tag{150}$$

論理演算の出力に対応する、相補的な論理値のコンテキストを以下のように定義する。

$$P_{n+}(G_{n+}|C_{n+}) \quad P_{n-}(G_{n-}|C_{n-}) \tag{151}$$

4.4.1　論理積演算

論理積演算は、双方の入力論理値が相補的になりうる。

$$G_n = G_k \wedge G_l \tag{152}$$

$$P_n\{G_n|F_{Ccomb}(C_k, C_l)\} = P_k P_l \tag{153}$$

4.4.2　論理和演算

論理和演算は、双方の入力論理値が相補的になりうる。

$$G_n = G_k \vee G_l \tag{154}$$

$$P_n\{G_n|F_{Ccomb}(C_k, C_l)\} = 1 - (1 - P_k)(1 - P_l) \tag{155}$$

4.4.3　排他的論理和演算

排他的論理和演算は、入力論理値が相補的になりうる。

$$G_n = G_k \oplus G_l \tag{156}$$

$$P_n\{G_n|F_{Ccomb}(C_k,C_l)\} = P_k + P_l - 2P_kP_l \tag{157}$$

4.4.4　恒等因果論理演算

恒等因果論理演算 $\mathcal{C}_{id}$ の論理値に対する演算を示す。因果関係を示す第二パラメータ G_l の入力は $true$ で固定されることが多い。第一パラメータ G_k の論理値が相補的になる。

$$G_n = \mathcal{C}_{id}(G_k,G_l) \tag{158}$$

$$P'_l > P_l \tag{159}$$

$$P_n\{G_n|F_{Ccomb}(C_k,C_l)\} = P_k + (1-P'_l) - 2P_k(1-P'_l) \tag{160}$$

4.4.5　含意因果論理演算

含意因果論理演算 $\mathcal{C}_{im}$ の論理値に対する演算を示す。因果関係を示す第二パラメータ G_l の入力は $true$ で固定されることが多い。第一パラメータ G_k の論理値が相補的になる。

$$G_n = \mathcal{C}_{im}(G_k,G_l) \tag{161}$$

$$P'_n = P_kP_l \tag{162}$$

$$P_n\{G_{n-}|F_{Ccomb}(C_{k-},C_{l+})\} > P'_n \tag{163}$$

4.4.6　論理積演算の逆演算

$$G_k = G_n \wedge^{-1} G_l \tag{164}$$

入力されたコンテキストが以下の式の条件を満たすとき、

$$P_{n+}(G_n + |C_{n+}) = true \tag{165}$$

$$P_{n-}(G_n - |C_{n-}) = false \tag{166}$$

$$P_{l+}(G_l + |C_{l+}) = true \tag{167}$$

$$P_{l-}(G_l - |C_{l-}) = false \tag{168}$$

以下のように逆演算が成立する。それ以外では逆演算は成立しない。

$$P_k\{G_l|F_{Cunif}(C_n+, C_n-, C_l+, C_l-)\} = true \tag{169}$$

4.4.7　論理和演算の逆演算

$$G_k = G_n \vee^{-1} G_l \tag{170}$$

入力されたコンテキストが以下の式の条件を満たすとき、

$$P_{n+}(G_n + |C_{n+}) = true \tag{171}$$

$$P_{n-}(G_n - |C_{n-}) = false \tag{172}$$

$$P_{l+}(G_l + |C_{l+}) = true \tag{173}$$

$$P_{l-}(G_l - |C_{l-}) = false \tag{174}$$

以下のように逆演算が成立する。それ以外では逆演算は成立しない。

$$P_k\{G_l|F_{Cunif}(C_n+, C_n-, C_l+, C_l-)\} = false \tag{175}$$

4.4.8　排他的論理和演算の逆演算

$$G_k = G_n \oplus^{-1} G_l \tag{176}$$

入力されたコンテキストが以下の式の条件を満たすとき、

$$P_{n+}(G_n + |C_{n+}) = true \tag{177}$$

$$P_{n-}(G_n - |C_{n-}) = false \tag{178}$$

$$P_{l+}(G_l + |C_{l+}) = true \tag{179}$$

$$P_{l-}(G_l - |C_{l-}) = false \tag{180}$$

以下のように、逆演算が成立する。

$$P_k\{G_l|F_{Cunif}(C_n+, C_n-, C_l+, C_l-)\} = false \tag{181}$$

入力されたコンテキストが以下の式の条件を満たすとき、

$$P_{n+}(G_n + |C_{n+}) = false \tag{182}$$

$$P_{n-}(G_n - |C_{n-}) = true \tag{183}$$

$$P_{l+}(G_l + |C_{l+}) = true \tag{184}$$

$$P_{l-}(G_l - |C_{l-}) = false \tag{185}$$

以下のように、逆演算が成立する。

$$P_k\{G_l|F_{Cunif}(C_n+, C_n-, C_l+, C_l-)\} = true \tag{186}$$

この 2 つ以外のケースでは逆演算は成立しない。

4.4.9　恒等因果演算子の逆演算

$$G_k = \mathcal{C}_{id}^{-1}(G_n, G_l) \tag{187}$$

入力されたコンテキストが以下の式の条件を満たすとき、

$$P_{n+}(G_n + |C_{n+}) = true \tag{188}$$

$$P_{n-}(G_n - |C_{n-}) = false \tag{189}$$

$$P_{l+}(G_l + |C_{l+}) = true \tag{190}$$

$$P_{l-}(G_l - |C_{l-}) = false \tag{191}$$

以下のように、逆演算が成立する。

$$P_k\{G_l|F_{Cunif}(C_n+,C_n-,C_l+,C_l-)\} = true \tag{192}$$

因果演算子は以上の条件満たさない場合でも逆演算が成立するが、以下の値となる。

$$P_k\{G_l|F_{Cunif}(C_n+,C_n-,C_l+,C_l-)\} = false \tag{193}$$

4.4.10　含意因果演算子の逆演算

$$G_k = \mathcal{C}_{im}^{-1}(G_n, G_l) \tag{194}$$

入力されたコンテキストが以下の式の条件を満たすとき、

$$P_{n+}(G_n + |C_{n+}) = true \tag{195}$$

$$P_{n-}(G_n - |C_{n-}) < true \tag{196}$$

$$P_{l+}(G_l + |C_{l+}) = true \tag{197}$$

$$P_{l-}(G_l - |C_{l-}) = false \tag{198}$$

以下のように、逆演算が成立する。

$$P_k\{G_l|F_{Cunif}(C_n+,C_n-,C_l+,C_l-)\} = true \tag{199}$$

因果演算子は以上の条件満たさない場合でも逆演算が成立するが、以下の値となる。

$$P_k\{G_l|F_{Cunif}(C_n+,C_n-,C_l+,C_l-)\} = false \tag{200}$$

4.4.11　含意演算

以下の式で表されるとき、A と B とは含意の関係である。A は B の部分集合であることと等価である。

$$A \to B \quad \equiv \quad A \subset B \tag{201}$$

この集合関係も、双方向論理では A から B への論理演算の一種と考える。つまり、A の論理値の $false$ 値の一部が $true$ 値に変化する演算とみなされる。B のすべてが $true$ 値になることもありうる。

$$P_A < P_B \tag{202}$$

4.4.12　逆含意演算

以下の式で表されるとき、A と B とは逆方向の含意の関係である。B は A の部分集合であることと等価である。

$$A \leftarrow B \quad \equiv \quad A \supset B \tag{203}$$

この集合関係も同様に、双方向論理では A から B への論理演算の一種と考える。つまり、A の論理値の $true$ 値の一部が $false$ 値に変化する演算とみなされる。B のすべてが $false$ 値になることもありうる。

$$P_A > P_B \tag{204}$$

4.5　属性キーの変数化

コンテキスト C が同一であり、論理値に整合性がある概念同士ならば、推定代入が可能になる。例として、以下のように概念 G_1 から、概念 G_2 への推定代入を行う。

$$G_1 \wedge p_1 = a \tag{205}$$

$$G_2 \wedge p_2 = a \tag{206}$$

これまでとの違いは、属性キー p_2 は属性キー p_1 を含む変数になっているという点である。

$$p_1 \subset p_2 \tag{207}$$

概念 G_1 の論理値は以下の通りとする。

$$P_n(G_1|C_n) = true \tag{208}$$

属性キー p_1 は仮定が必要な対象であり、新たな仮定を用いたコンテキストが生成される。

$$P_{p_1+}(p_1|p_1 = true) = true \quad P_{p_1+}(p_1|p_1 = false) = false \tag{209}$$

２つのコンテキストが合成された結果、以下のような２つの相補的な論理値とコンテキストとが生成され、値 a に伝搬される。

$$P_{n+}(a|p_1 = true, C_n) = true \quad P_{n-}(a|p_1 = false, C_n) = false \tag{210}$$

属性キー p_1 は属性キー変数 p_2 に含まれるので、属性キーの論理値が p_2 側に伝搬される。

$$P_{p_2+}(p_2|p_1 = true) = true \quad P_{p_2-}(p_2|p_1 = false) = false \tag{211}$$

値 a から概念 G_2 に向けて逆伝搬を行う。その際に属性キー p_2 の持つコンテキストが合成される。合成された結果の論理値は不確定だが、false ではないとみなされる。

$$P_{n'}(G_2|C_n) > false \tag{212}$$

この結果、別の属性キーを持つ G_1 と G_2 同士で、コンテキストの推定一致が確認される。最終的に、同一のコンテキストを持つ概念 G_2 に対して、概念 G_1 が推定代入される。

$$G_1 \subset G_2 \tag{213}$$

4.6　序列表現の関数適用

序列を論理式で表現する際に、序列に相当するオブジェクトのインスタンスを、概念ごとにその都度生成するという手法を取った。この序列を用いた概念を関数に入力して、値の間の関連を用いることができる。

$$G_{ab} \wedge prev_{ab} = a \tag{214}$$

$$G_{ab} \wedge next_{ab} = b \tag{215}$$

序列オブジェクトを用いる関数は以下のようになる。

$$F_{D1} \wedge prev_v = x \tag{216}$$

$$F_{D2} \wedge next_v = y \tag{217}$$

$$F_R \wedge result = y \tag{218}$$

$$prev_{ab} \subset Previous \supset prev_v \tag{219}$$

$$next_{ab} \subset Next \supset next_v \tag{220}$$

$$F_R = F_D1 \wedge F_D2 \tag{221}$$

a,b それぞれの論理値 P_a, P_b は、G_{ab} におけるコンテキスト C_G を用いて以下のように表現される。それぞれ序列オブジェクト $prev_{ab}$ と $next_{ab}$ を前提として追加している。

$$P_a(a|\{prev_{ab} = true, C_G\}) \subset P_{ab}(G_{ab}|C_G) \tag{222}$$

$$P_b(b|\{next_{ab} = true, C_G\}) \subset P_{ab}(G_{ab}|C_G) \tag{223}$$

最初に、$x = a$ が代入される。次に、x から F_{D1} へと逆演算するときに、$prev_v$ が逆演算条件として必要になる。ここで、コンテキストの一致を用いて、$prev_{ab}$ が $Previous$ を経て $prev_v$ へと代入される。

$$prev_v \supset prev_{ab} \supset C'\{prev_{ab} = true, C_G\} \tag{224}$$

この逆演算により、コンテキストは前提 $prev_{ab} = true$ が相殺されて C_G に戻る。

$$P_{FD1}(F_{D1}|C_G) = true \tag{225}$$

同様に、$y = b$ が代入され、$next_{ab}$ もまた $next_v$ へと代入される。さらに、逆演算で F_{D2} へと伝搬される。

$$next_v \supset next_{ab} \supset C'\{next_{ab} = true, C_G\} \tag{226}$$

$$P_{FD2}(F_{D2}|C_G) = true \tag{227}$$

P_{FD1} と P_{FD2} との論理積の結果、P_{FR} の論理値 $true$ とコンテキスト C_G とが概念 G_{AB} と同一となり、G_{AB} との包含関係が確定する。

$$F_R = F_D1 \wedge F_D2 \tag{228}$$

$$P_{FR}(F_R|C_G) = true \tag{229}$$

$$G_{ab} \subset F_R \tag{230}$$

この結果、y に b が代入されているので、結果が G_{ab} に追加される。

$$F_R \wedge result = b \tag{231}$$

$$G_{ab} \wedge result = b \tag{232}$$

こうして、一般化された序列を持つ関数に対して、序列を持つ概念を代入することができる。コンテキストを用いることで、無関係な序列に関する代入を排除できる。

4.7　論理式の連想結合

コンテキスト C が同一である複数の論理式の概念同士は、１つの概念に統合することが可能である。以下の例では、これら論理式 G_1 と G_2 は同じ時刻 t_1 でそれぞれ別に観測されたものである。典型的にはそれぞれ学習や外部観測によって得られる。

$$G_1 \wedge O_1 = v_1 \tag{233}$$

$$G_1 \wedge time = t_1 \tag{234}$$

$$G_2 \wedge O_2 = v_2 \tag{235}$$

$$G_2 \wedge time = t_1 \tag{236}$$

ここで、t_1 はコンテキスト C_1 を内包する。

$$t_1 \supset C_1\{time = true, C_0\} \tag{237}$$

そのため、逆論理演算でコンテキストを G_1、G_2 へと伝搬した結果、

$$G_1 \supset C_0 \tag{238}$$

$$G_2 \supset C_0 \tag{239}$$

そのため、実際の集合の大きさが不確定な 2 つの概念集合 G_1, G_2 は少なくとも部分的に同一であるとみなすことができるため、統合した概念 G' を作成できる。

$$G' \supset G_1 \tag{240}$$

$$G' \supset G_2 \tag{241}$$

これが、一般化された連想形成とみなすことができる。ただし、コンテキスト C が同一である観測対象の中には、無関係なものも多く含まれるため、最適化する手段が別に必要になる。

ここでは詳細に説明しないが、一般論として発生確率が小さい事象同士の連想は成立可能性が高い。この理屈を前提コンテキストを用いて説明する。確率が小さいということは、暗黙の前提が多いと言い換えることが可能であり、前提が多い概念同士で連想が行われるので、双方の暗黙の前提が相殺される可能性が高いということである。

4.8　論理式の矛盾検出

ある論理式を用いた結果、ほかの論理式との矛盾が発生することがある。その論理式が推測代入を含み、もう１つの論理式が含まない場合は、推測代入に間違いがあると判断できる。推測代入は代入先の論理式が正しいという保証がない場合があり、その場合は観測によって補正を行うことになる。

この論理式の矛盾を発見するために、コンテキストの一致を用いる。

$$G_A \wedge p_1 = v \tag{242}$$

$$G_B \wedge p_1 = \neg v \tag{243}$$

さまざまな論理式を経て、G_A と G_B にそれぞれ同じコンテキスト C_n が到達する。

$$P(G_A|C_n) = true \tag{244}$$

$$P(G_B|C_n) = true \tag{245}$$

その結果、全く別の論理式同士の代入が可能となる。

$$G_A = G_B \tag{246}$$

この G_A と G_B からそれぞれ変数 v に向けて伝搬を行った結果、同じ v に違う結果が到達する。

$$P(v|\{p_1 = true, C_n\}) = true \tag{247}$$

$$P(v|\{p_1 = true, C_n\}) = false \tag{248}$$

この結果、値 v で論理値の矛盾が発生する。この矛盾を補正する手段が必要になる。

4.9　論理式の自律補正

観測された矛盾の解消のために、コンテキストを用いて論理式に成立条件を追加する。そのための手段を解説する。

まず、コンテキスト C_1 では変数概念 VG_1 と概念 G_1 は部分一致しているため、包含関係にあるとみなされる。

$$VG_1 \supset C_1\{a = true, b = true, c = false\} \tag{249}$$

$$G_1 \supset C_1\{a = true, b = true, c = false\} \tag{250}$$

$$VG_1 \wedge p_1 = v \tag{251}$$

$$G_1 \wedge p_1 = v \tag{252}$$

$$VG_1 \supset G_1 \tag{253}$$

次の時点で、論理式 VG_2 と G_2 の間で値 v の矛盾が発見された。すなわち、コンテキスト C_2 のときの論理式には補正が必要である。この補正の方法を探索するために、矛盾が検出されない時のコンテキスト C_1 と、矛盾が検出された時のコンテキスト C_2 とを比較する。

$$VG_2 \supset C_2\{a = true, b = true, c = true\} \tag{254}$$

$$G_2 \supset C_2\{a = true, b = true, c = true\} \tag{255}$$

$$VG_2 \wedge p_1 = v \tag{256}$$

$$G_2 \wedge p_1 = \neg v \tag{257}$$

ここで、コンテキスト C_1 とコンテキスト C_2 の間で値 x が変動している、無関係な別の論理式が発見される。

$$XG_1 \supset C_1\{a = true, b = true, c = false\} \tag{258}$$

$$XG_2 \supset C_2\{a = true, b = true, c = true\} \tag{259}$$

$$XG_1 \wedge p_x = x \tag{260}$$

$$XG_2 \wedge p_x = \neg x \tag{261}$$

すなわち、論理式XGが論理式VGの不一致の要因である可能性がある。この結果を用いて、論理式VGに対して以下のように成立条件XGを論理積として追加する。

$$VG' = VG \wedge XG \tag{262}$$

$$XG \wedge p_x = x \tag{263}$$

もちろんこのような条件付加は、正解候補の１つであって確実な正解ではない。さらに、長い論理式であれば、論理式の中の複数の位置に対して条件追加が可能である。そのため、このような正解候補の論理式をすべて生成して、さらなる観測で確認する必要がある。

さらに、差分が発見された概念YGのコンテキストに、前提要素がより多く付加されている場合がある。そういう場合は、差分の前提要素$(d = true, e = false)$を論理式に変換してYG'として整合性を取る。

$$YG \subset C_1\{a = true, b = true, c = false, d = true, e = false\} \tag{264}$$

$$VG' = VG \wedge YG' \tag{265}$$

$$YG' \wedge d \wedge \neg e = YG \tag{266}$$

逆に、差分側概念のコンテキストの前提要素が少ない場合は、差分側の概念とそのまま論理積を形成できるので、前提要素の論理式を追加する必要はない。

4.10　双方向論理ネットワークへのコンテキスト伝搬

論理因果演算子は、関数の結果比較や適用そのものを論理値化する。これを用いて関数の適用制御や値を全て混在させた論理式にすることにより、さらなる関数制御を階層的に形成することが可能になる。その結果、巨大な双方向論理ネットワークとして統括的に管理することができる。

そして、コンテキストは双方向論理の任意の方向に伝搬が可能であることを用いれば、関数制御などの上位の関数に対しても階層的にコンテキストを伝搬することができる。コンテキスト伝搬が出来ないケースは、概念間の推定代入が不可能な場合や、論理積による空集合化などが考えられる。

この結果、巨大な双方向論理ネットワーク上の離れたノード間の関連を、コンテキスト同士の関連を用いて一意に示すことができる。この任意の関連を用いれば、概念同士の連想形成や、ネットワークで形成された法則の矛盾検出や、法則の成立条件追加などが可能になる。

5　数学への応用

5.1　一般的な数値表現

一般的な１０進数の表現の例を示す。以下の例では”235” である。

$$NS \wedge Digit1 = \text{"2"} \tag{267}$$

$$NS \wedge Digit10 = \text{"3"} \tag{268}$$

$$NS \wedge Digit100 = \text{"5"} \tag{269}$$

$$DigitSet \wedge in10^0 = Digit1 \tag{270}$$

$$DigitSet \wedge mul10^0 = Digit10 \tag{271}$$

$$DigitSet \wedge in10^1 = Digit10 \tag{272}$$

$$DigitSet \wedge mul10^1 = Digit100 \tag{273}$$

$$in10^0, in10^1 \in in10 \tag{274}$$

$$mul10^0, mul10^1 \in mul10 \tag{275}$$

$Digit$ は $mul10$ を用いた関連を用いれば無制限の桁まで生成できる。このような数値表現は関数の入力になり、各桁の計算が適用される。10 進数の数値同士の加算などは、桁ごとに関数が適用されて再帰的に桁上がりを用いて計算していくが、これも部分関数と再帰を用いて記述できるはずである。

一般的な実数はこういう整数表現だけではなくて、方程式を満たす値という定義方法もある。さらには、値を明記せずに方程式を用いて値の範囲だけを示すという定義方法も存在する。これらは方程式の因果演算子の入力値を属性として持つことで表現できる。

5.2　方程式、代数

関数間の写像を定義し、さらにその写像を関数とすることで代数と同じものと考えられる。代数とは具体的には交換関係、結合関係などの関連であり、その適用自体も制御の対象になる。すなわち、代数自体がさらなる上位の関数の制御対象となる。

自然数における基本的な交換関係を例に用いる。

$$x + y = y + x \tag{276}$$

これを双方向論理を用いて記述する。

$$G_{l1} \wedge i_{l1} = x \tag{277}$$

$$G_{l2} \wedge i_{l2} = y \tag{278}$$

$$\mathcal{C}_{id}^{-1}(G_{l1} \wedge G_{l2}, G_l) = add \tag{279}$$

$$G_{r1} \wedge i_{r1} = y \tag{280}$$

$$G_{r2} \wedge i_{r2} = x \tag{281}$$

$$\mathcal{C}_{id}^{-1}(G_{r1} \wedge G_{r2}, G_r) = add \tag{282}$$

この式は自然数における恒等式なので、左辺と右辺は条件抜きでイコールで結び付けられる。

$$G_l = G_r \tag{283}$$

仮に、方程式に成立条件G_qがあるときは、こちらも論理因果演算子$\mathcal{C}_{id}$を用いて表現する。

$$G_q = \mathcal{C}_{id}^{-1}(G_l, G_R) \tag{284}$$

恒等因果の結果G_qは、この交換関係を用いる条件となる。このG_qを用いて他の要素との関連を示す論理式を形成することで、代数適用自体の論理式化が可能になる。あとは、関数の階層化と制御と同じ原理で、代数の適用の選択制御が可能になる。

5.3　再帰関数

関数 f の中で関数 g を再帰的に実行するときの手順を示す。

$$f(in : a, out : b) \tag{285}$$

$$g(in : c, out : d, cond) \tag{286}$$

基本的な方法は、終了条件 $cond$ を満たしていれば関数 g の結果 d を関数 f の結果 b に連結し、満たしていなければ関数 g の出力 d を再び入力 c に連結する。

$$\begin{cases} c = d & \text{if} \quad \text{cond} \\ b = d & \text{otherwise} \end{cases} \tag{287}$$

これを論理式を用いて記述する。まずは関数 f の入出力を定義する。

$$F_{inA} \wedge input = a \tag{288}$$

$$F_{outB} \wedge Output = b \tag{289}$$

関数 g の入力は、最初は f の入力を接続する。それ以降は $cond$ が $false$ である限り関数 g の出力を接続して再帰演算を行う。G_{first} の論理式は省略する。

$$F_{inA} \wedge G_{first} \rightarrow G_{in} \tag{290}$$

$$G_{outD} \wedge \neg G_{cond} \rightarrow G_{in} \tag{291}$$

$$G_{inC} \wedge input = c \tag{292}$$

関数 g の出力の中に、再帰の終了条件が含まれる。終了条件 $cond$ は $false$ ならば、関数の再帰を続行する。

$$G_{out} \wedge g_{outD} = d \tag{293}$$

$$G_{out} \wedge g_{cond} = cond \tag{294}$$

$$G_{cond} \wedge g_{cond} = cond \tag{295}$$

$$G_{cond} \subset G_{outcond} \tag{296}$$

終了条件 $cond$ が $true$ となった結果、関数 g の出力を関数 f からの出力へと接続する。

$$G_{outD} \wedge output = d \tag{297}$$

$$G_{outD} \wedge G_{cond} \rightarrow F_{out} \tag{298}$$

このように、関数の要素に相当する概念を用いることで、再帰も論理演算で記述できる。

5.4　量化子

代数の記述には、全体か、ある一部であることを示すための量化子の概念が必要になる。この量化子に基づいた数式の表現は、因果論理演算子と量化子に基づく変数との関連で示すことができる。

まずは、存在量化∃を用いた式を考える。

$$\exists x | x \in R, f(x) = true \tag{299}$$

f(x) を論理式で記述する。G_{equal} はこの $f(x) = true$ が成立する条件となる概念である。

$$F_i \wedge i_1 = x \tag{300}$$

$$F_o \wedge o_1 = true \tag{301}$$

$$\mathcal{C}_{id}(F_i, G_{equal}) = F_o \tag{302}$$

x を個別に選び出すということは、x のそれぞれの値に対してコンテキスト $C[x]$ が対応する。つまり、コンテキスト $C[x]$ ごとの G_{equal} を、$x \in R$ に対して全て集合的に総和すると、空集合ではないと表現できる。

$$x \in R | \sum_{x} G_{equal}(C\{x\}) \neq \tag{303}$$

次に、全称量化∀を用いた式を考える。

$$\forall x | x \in R, f(x) = true \tag{304}$$

$f(x)$ を同様に論理式で記述する。G_{equal} はこの $f(x) = y$ が成立する条件となる概念で

ある。

$$F_i \wedge i_1 = x \tag{305}$$

$$F_o \wedge o_1 = true \tag{306}$$

$$\mathcal{C}_{id}(F_i, G_{equal}) = F_o \tag{307}$$

x は R のすべての要素を取る。その結果、G_{equal} は R の中のすべてのコンテキスト $C[x]$ に対して $true$ となると表現できる。

$$\forall x \in R | G_{equal}(C\{x\}) \equiv true \tag{308}$$

R 上の x の値をループで網羅的に発行する機構も考えられるが、別の方法を用いる。

6　さらなる応用

6.1　オントロジー

オントロジーはオブジェクトの持つ様々な属性や関係を表現する手法であり、これを論理式だけで記述すれば論理代数の対象にすることができる。属性を表現するには以下の手段を用いる。含意 $A \Rightarrow B$ を用いることにより、他の人物も同じ属性値を持つことができる。

$$Peter \wedge \mathrm{hasName} \Rightarrow \text{”Peter”} \tag{309}$$

$$Peter \wedge \mathrm{Me} \Rightarrow 35 \tag{310}$$

関連性の表現では問題が発生する。$Peter$ は $John$ の包含集合ではない。

$$Peter \wedge \mathrm{hasParent} \Rightarrow John \tag{311}$$

そこで双方を包含する概念を用いれば論理式としての記述が可能になる。

$$G_{PA} \wedge \mathrm{childPA} \Rightarrow Alice \tag{312}$$

$$G_{PA} \wedge \mathrm{parentPA} \Rightarrow Peter \tag{313}$$

ところで、$Peter$ は $Alice$ に対しては $Parent$ であるが、Bob に対しては $Child$ である。

$$G_{BP} \wedge \mathrm{childBP} \Rightarrow Peter \tag{314}$$

$$G_{BP} \wedge \mathrm{parentBP} \Rightarrow Bob \tag{315}$$

このことを表現するために、$child$ と $parent$ は式ごとに生成される属性キーのインスタンスとする。インスタンス $childPA$ などを論理式の $childvariable$ にその都度代入することで、一般的な推論に利用することができる。

$$childPA, childBP \subset Child \supset childvariable \tag{316}$$

$$VG \wedge childvariable \Rightarrow variable \tag{317}$$

上の例では、$Alice$ が $variable$ に代入されたときに、$childPA$ だけが $childvariable$ に代入され、最終的には G_{PA} だけが変数概念 VG に推定代入されることになる。G_{BP} は VG には推定代入されない。VG に論理式 G_{PA} が 1 つだけ選択されて推定代入されることで、選択対

象に対して様々な関数を適用できる。

6.2　幾何学的認識

空間上の物質などのオブジェクト O_1 と O_2 との関連を示すには、例えば以下のように属性を列挙する。

$$G_{12} \wedge source = O_1 \tag{318}$$

$$G_{12} \wedge target = O_2 \tag{319}$$

$$G_{12} \wedge distancemeter \rightarrow 10 \tag{320}$$

$$G_{12} \wedge direction \rightarrow \mathrm{North} \tag{321}$$

$$G_{12} \wedge position \rightarrow PositionObject_{12} \tag{322}$$

$$PositionObject_{12} \wedge X = 100 \tag{323}$$

$$PositionObject_{12} \wedge Y = 40 \tag{324}$$

このように関連概念 G_{12} をさらに属性で階層化することで、オブジェクト間の任意の空間的な関連を記述することも可能になる。これまでと同じく G_{12} を推定代入することで関連を用いて推論を行うことができる。これを用いれば、幾何学や物理的世界をモデル化して論理演算の対象にする、すなわち空間の認識自体を論理式とすることが可能になる。

6.3　コンピュータ

いわゆるチューリング完全と呼ばれる万能のコンピュータは、チューリングマシンと等価であることが要求される。だが、実際のチューリングマシンとの対比よりも以下の定義を用いてコンピュータの根源的な挙動を表現するほうが扱いやすい。

- プログラムの命令発行
- 命令の選択
- メモリリード

- 命令の実行
- メモリライト
- プログラムカウントの変更

  これらだけで全て表現することができる。

6.3.1 序列集合

時系列を示すオブジェクトを定義する。メモリの値は書き込みによって変化するが、書き込まれない限り変動しない。このような時系列を効果的に示すために以下の方法を用いる。

- 現在時刻 t_n
- 現在時刻より未来 $f_{n-1} \supset f_n \supset f_{n+1}$

未来の時系列の間の関係は以下の式で表される。わずかに起点がずれた未来 f_{n+1} と f_n との差分が現在時刻 t_n となる。

$$f_{n+1} = f_n \wedge \neg t_n \tag{325}$$

この結果、現在以降のメモリの状態を示す時間 tf_n は以下のような式で表現される不確定な集合である。

$$t_n \subseteq tf_n \subset f_n \tag{326}$$

この tf_n は、「いつか書き込みで改変される可能性はあるが、改変されていない限りは現在の値を示す」ということを表現できる。この tf_n を利用して、メモリの内容全体を示す概念 G_n の時系列変化を記述することができる。この場合の概念はコンピュータ用語におけるステートとほぼ等価である。

前回のメモリの書き込みは時間 tf_m で行われた。

$$G_{n-1} \wedge Address1 \wedge tf_m = V_1 \tag{327}$$

今回は、メモリの書き込みが時間 tf_n で行われた。

$$G_n \wedge Address1 \wedge tf_n = V_2 \tag{328}$$

こうして、不確定な時間 tf_m は tf_n 以降の時間は含まなくなり、時系列ごとのメモリの値が表現できる。

6.3.2　論理式でのコンピュータの動作表現

時系列の準備ができたところで、コンピュータをもっとも簡略化して論理式で表現する。$LocalRegisters$ の中身は複数存在しうるが単一の式として表現している。命令実行 $func$ の中身は恒等因果を用いているが省略している。$State$ は時間ごとのコンピュータの内部状態を示す概念である。クロックの変数 n はステップごとに増加していく数値である。

- クロック

$$tf_{n+1} = tf_n \wedge \neg t_n \tag{329}$$

$$CurrentTime = tf_n \tag{330}$$

$$NextTime = tf_{n+1} \tag{331}$$

- メモリ

$$Memory \wedge M(adr : Address) \wedge readwrite \wedge CurrentTime \rightarrow value \tag{332}$$

簡略化のためアドレス $Address$ のメモリ内容を示す概念 M は以下の条件式で表現することとする。

$$M(adr : Address) \tag{333}$$

$$M \wedge adr = Address \tag{334}$$

- 命令発行

$$State \wedge ProgramCounter = InstructionAddress \tag{335}$$

$$InstructionMemory \wedge M(adr : InstructionAddress) \wedge readwrite \rightarrow Instruction \tag{336}$$

- 命令デコード

$$decoderfunction(in : Instruction, out : decodedfunc) \tag{337}$$

$$AccessRegisters = State \wedge CurrentTime \wedge Register \tag{338}$$

$$ImmediateValues = Instruction \wedge ImmediateField \tag{339}$$

$$CalculateEffectiveAddressFunc(in : ImmediateValues, in : Registers, out : AccessAddress) = de \tag{340}$$

- メモリリード

$$ReadData = Memory \wedge M(adr : AccessAddress) \wedge readwrite \wedge CurrentTime \tag{341}$$

$$State \wedge CurrentTime \wedge Register = ReadData \tag{342}$$

- 命令の実行

$$InputRegisters = State \wedge CurrentTime \wedge Register \tag{343}$$

$$InstructionExecution(in : InputRegisters, out : OutputRegisters) = decodedfunc \tag{344}$$

$$State \wedge NextTime \wedge Register = OutputRegisters \tag{345}$$

- メモリライト

$$WriteDatas = State \wedge CurrentTime \wedge Register \tag{346}$$

$$Memory \wedge M(adr : AccessAddress) \wedge readwrite \wedge NextTime = WriteDatas \tag{347}$$

- 命令アドレスの制御

$$NextInstructionFunc(in : LocalRegisters, out : NextInstructionAddress) \quad (348)$$

$$State \wedge NextTime \wedge ProgramCounter = NextInstructionAddress \quad (349)$$

6.4　既存の論理演算による知識処理

双方向論理で表現できると思われる知識処理は、既存のコンピュータ、つまり論理演算を用いた人間の手によるプログラムでほとんど実現されている。既存のプログラムでは、人間の手によって暗黙の内に、推定代入と同じ原理で論理回路が結合されて利用されている。以下のような概念 G_m と G_n との間の推定代入を例に挙げる。

$$G_m \wedge StateA = a \quad (350)$$

$$G_m \wedge VariableA = a \quad (351)$$

$$G_n \wedge VariableA = x' \quad (352)$$

$$G_n \wedge StateB = x' \quad (353)$$

G_m の持つ $StateA$ に格納されている値 a は、G_m から G_n への推定代入によって、G_n の持つ $StateB$ へと代入される。この代入の処理は以下のような古典的なメモリ変数の受け渡しと等価となる。こうして概念\$$G_m$ や G_n\$は人間によって暗黙の内に処理されて意識されることはない。

$$MemoryWrite(Address) = StateA \quad (354)$$

$$StateB = MemoryRead(Address) \quad (355)$$

では、双方向論理は何が従来の論理演算と異なるかというと、概念とコンテキストを明確にすることで論理式の組み合わせを定式化して、推定代入の自動適用を可能にするという点である。それに対して、古典的な論理式や推論規則だけでは自動的な推定代入ができない。すなわち、自動的な論理式の生成ができない。

ニューラルネットワークでは、可能な限りの大量の論理接続の組み合わせを生成することで、自然に推定代入に相当する論理回路の結合が含まれている。その中からフィードバックによって得られた正解を取捨選択している。逆に言えば、概念とコンテキストとを用いれば、論理接続の組み合わせを理論上の最小限にできる。探索の組み合わせが指数関数的に増大すると予測される高度な AI において、体系的な方法論を用いて探索を最小化することは必須となるはずである。

7　結論

双方向論理は逆論理演算、概念化、推定代入、論理因果演算子とで構成される。逆論理演算は本来は不確定で無意味な演算であるが、推定代入という手法を認めることよって、逆論理演算の結果と他の概念との関連を矛盾なく示すことができる。このアイディアは十分に単純かつ整合性がある上に、圧倒的な応用力があり、これこそが論理代数の拡張の最有力候補である。

この双方向論理を階層化した双方向論理ネットワークは、論理式とその運用を包括的に管理する。ネットワークの内部ノード同士の関連は、前提コンテキストによって管理されて一意に決定される。この関連を用いて、連想形成、矛盾検出、条件形成などの論理式の改変が十分な根拠をもって実行される。

この新しい双方向論理は、今までの論理式の代数ではできなかった推論規則、代数などの分野に適用することができる。更には、数学そのものや、空間認識、プログラミング言語等を論理代数の対象とすることも可能になる。これに加えて、双方向論理には論理式の自律修正、一般化の方法も含まれるため、一般的な知識の自律生成も可能になると期待できる。最終的には、理論的に整合性のある網羅的で完全な機械学習を実現するための不可欠な道具となる。