データベースシステム 91－7
（1992. 11. 5）

ハイパーテキストと問い合わせの形式的なモデル

渡辺 隆一　　箕原 辰夫　　所 真理雄
慶應義塾大学大学院 計算機科学専攻

ハイパーテキストについての形式的なモデル化は、現在まであまり研究されてこなかった領域の研究である。そのため、個々のハイパーテキストのシステムがもつ機能について、形式的に比較し、議論することができなかった。我々は、ハイパーテキストに適用される操作の意味論までに立ち入って、形式的なハイパーテキストのモデルを提案する。ハイパーテキストにおける一般的な情報検索は、航行機構を用いることにより実現されている。我々は航行機構に加えて、新たな検索機構として関係論理と等価な集合検索能力を持つ問い合わせの機構を我々のハイパーテキストのモデル上に提案し、情報検索能力を増すことを試みる。本論文で提案するモデルや機構は、すべて形式的に表現されているので、この結果は様々な具体的なハイパーテキストのシステムの設計に応用することが可能である。

A Formal Model on Hypertext and Query

Ryuichi Watanabe　　Tatsuo Minohara　　Mario Tokoro
Department of Computer Science
Keio University
3-14-1,Hiyoshi,Kohoku-ku,Yokohama,223,Japan
{*ryuichi,minohara,mario*}*@mt.cs.keio.ac.jp*

In the research field of hypertext, one cannot establish a comparison among the facilities of the proposed hypertext systems for the lack of their precise formal models. We propose a formal model of hypertext that has a power of expression and integrates the common proposed structures in many hypertext systems. The operations applied on the structures in the model are also formally defined. In an ordinary hypertext system, a navigation mechanism is provided as the main facility for searching the contents. In order to expand the ability to search, we propose a query mechanism to be used in addition to the navigation mechanism. The expressive power of the query mechanism are equivalent to one of the relational calculus. Since the model and the mechanisms proposed in this paper are defined in a formal way, they can be applied to various practical hypertext systems.

1 はじめに

様々なハイパーテキストシステムが今日までに提案され実装されているが、その情報検索は主に航行機構を用いることにより実現されている。航行機構によりユーザは、設計者が事前に定義されたリンクをたどり、いくつものノードを情報を探索して移動することができる。しかし、定められたリンクをたどるだけでは、文字列の検索などのように検索が困難な場合がある。この問題は既に、[9] において指摘されている。さらに、ほとんどのハイパーテキストシステムにおいては、簡単な文字列検索の機能しか用意されていないので、関係データベースにあるような強力な問い合わせ機構をハイパーテキストでも提供すべきである。航行機構は、航行で訪れるそれぞれの地点において、検索者の恣意にゆだねて次の地点を検索していくような、目的が曖昧で漠然とした検索においてもっとも効果的である。一方、問い合わせ機構は目的が明確な全文検索などに向いている。このことより、両方の検索機構がハイパーテキストに必要であることは容易に分かる。

本論文では、ハイパーテキストに問い合わせ機構を組み入れるための基礎となる理論的な枠組を提案する。問い合わせ機構を用いることにより、システムの設計者によってあらかじめ定義された内容やリンクの環境とは別に、ユーザが自分の視点に基づいた環境を構築することができる。例えば、NoteCards[17] では Guided Tour により既存のハイパーテキスト内を新しいシナリオに基づいて航行することができる。ユーザの観点に基づいた環境を構築するためには問い合わせの機構が欠かせない。問い合わせによりハイパーテキストから任意の情報を抽出することができるので、ハイパーテキストの別の側面、ユーザの観点、新たな航行のシナリオなど、様々な視点に基づいたハイパーテキストをユーザが構築することが可能になる。

問い合わせの機構を構築するために、本論文ではまずハイパーテキストに形式的なモデルを与える。多くのハイパーテキストシステムにおいては形式的な記述は与えられていないが、システムについて議論する上で形式的な記述が与えられていることは重要である。それは、形式的なモデルが、そのハイパーテキストがどのような機能を有しているかを正確に表す仕様として用いることができるからだ。強力な機構を持つと主張していても、それがどのような機能であるかを正確に表せないようなハイパーテキストシステム。それは、そのシステムを使用するに有効であるかも知れないが、ハイパーテキスト全体の研究にとっては、再利用性の低いものとは言えないだろうか？我々はハイパーテキストのモデルを、その構造とそれに適用される操作の 2 つの観点から形式的に定義する。リンクの種類やノードの内容などの、いわゆるハイパーテキストの意味はアプリケーションに依存したものと考えることができる。よって、アプリケーションに独立であるような一般的な構造のみについて我々は議論する。

本論文の構成であるが、まず第 2 章において我々のハイパーテキストシステムの構造および操作の形式的な記述を示す。第 3 章において我々のハイパーテキストに対する問い合わせ言語の構文規則とその問い合わせで得られる集合に関しての意味を与える。そして第 4 章では関連する研究と我々の研究とを比較し議論する。最後に第 5 章で結論と我々の研究の現状と将来への展望について述べる。

2 ハイパーテキストの形式化

近年、多くの活動的な研究者によって様々なハイパーテキストシステムが提案され実装されている。しかし、それらの研究の主な目的はハイパーテキストの可能性を探索するということであった。即ち、「ハイパーテキストの得意な分野は？」、「ハイパーテキストを使う際の問題は？」、「その問題を解決するには？」といった質問に対する答を探すために労力が注がれていた。ハイパーテキストという研究分野がこのようにまだ未知の問題領域を探ることに比重が置かれており、その成果を客観的に外に示すことができる研究を熟成する前の段階にあるということも手伝って、ハイパーテキストシステムの形式化に関する研究はあまり行なわれていない [5, 8, 16, 2]。

本章では、ハイパーテキストの形式的なモデルの記述を呈示し、その有効性や表現性に関しての一般的な議論が行なえる基盤を作る。ここでの形式化は使用者が与える意味論から分離した、ハイパーテキストに関しての純粋な構造および操作のみに基づいて構築していく。

2.1 ハイパーテキストの構造

通常のハイパーテキストはノード、ブロック、リンクといった実体により構成される [5]。ここで導入する他の有効な構造としてはアンカーと束縛がある。アンカーを

定義することにより、ハイパーテキストにおけるリンク付けの柔軟性や表現力が上がる [12]。そして、束縛によりユーザは実体に名前付けをすることが可能になる。これらの構造を持つ実体の総和であるハイパーテキストを最初に定義する。

定義 1 (実体とハイパーテキスト) 実体は、内包的なグループ、外延的なグループ、ノード、ブロック、アンカー、リンク、束縛のいずれかであり、ハイパーテキスト $\mathcal{H}$ はの実体の異種型多重集合である。 □

幾つかの異なる構造を持った実体により構成されているので、ここではハイパーテキストを異種型 (heterogeneous) 集合と定義した。

定義 2 (グループ) グループ G はハイパーテキスト $\mathcal{H}$ の異種型多重部分集合である。 □

グループはハイパーテキスト中の任意の実体の集合である。ハイパーテキストが異種型集合であるので、グループも異種型集合である。グループを定義することにより、問い合わせにより集められる実体の集合や、問い合わせを用いずに集められた実体の集合などの任意の集合を表現することが可能となる。前者を内包的なグループ、後者を外延的なグループと呼ぶ。前の定義では、ハイパーテキストはグループを要素として持つような集合であり、また同じ実体が重複することがあるので、我々はハイパーテキストを多重集合として定義した。

定義 3 (ノード) ハイパーテキストにおけるノードはブロックの並びである。ノードは組$< id, b_1 \ldots b_n, a_1 \ldots a_n >$により表される。$id$ は識別子、$b_1 \ldots b_n$ と $a_1 \ldots a_n$ はブロックの並びとそれに対応する修辞属性の並びである。ブロックのグループはメタ変数 $N_1, \ldots, N_n$ により表し、ノードはメタ変数 $n_1, \ldots, n_n$ により表す。 □

ハイパーテキストのノードはブロックの並びより構成され画面上のウィンドウに 1 対 1 に対応する。ブロックの並びに対しては、追加、挿入、削除、結合などの操作が可能である。

修辞属性というのは、ハイパーテキストの内の実体をシステム上で表現するときに用いられる属性のことである。例えば、ある実体がテキストのブロックであった場合は、そのフォント、サイズ、色などが修辞属性として記憶されていれば、ウィンドウ上に表現するときに、それらが用いられる。この論文では、修辞属性はメディアに依存するものであるので、修辞属性の具体的な内容について言及することは避ける。しかし、我々のモデルでは、修辞属性が固有なものとして、ノードやブロックなどの実体に付随できる能力があることに注目されたい。

定義 4 (ブロック) ハイパーテキストにおけるブロックは、値もしくはブロックの並びを表す。前者は原始ブロック、後者は複合ブロックと呼ばれる。原始ブロックは組$< id, v, a >$により表される。id は識別子、vは値、a は修辞属性である。複合ブロックは組$< id, b_1 \ldots b_n, a_1 \ldots a_n >$により表される。$id$ は識別子、$b_1 \ldots b_n$と $a_1 \ldots a_n$はブロックの並びとそれに対応する修辞属性の並びである。ブロックのグループはメタ変数 $B_1, \ldots, B_n$ により表し、ブロックはメタ変数 $b_1, \ldots, b_n$により表す。 □

原始ブロックはテキスト、イメージ、音声、動画といった実際の値を表し、ハイパーテキストにおける情報の最小単位となる。複合ブロックはそれらの原始ブロックや他の複合ブロックを並びとして持つブロックである。これらのブロックやノードにはアンカーが結びつけられることがある。

定義 5 (アンカー) アンカーはハイパーテキスト中の点を表す。アンカーは組$< id, n >$または$< id, ns[bs] >$により表される。id は識別子、n はノード、$ns[bs]$ はブロックである。ただし、ns は *Node Specification*、bs は *Block Specification* である[1]。アンカーはメタ変数 $a_1, \ldots, a_n$ により表す。 □

アンカーはノードやブロックに結びつけられることにより、ハイパーテキスト中のある 1 つの点を表し、リンクのリンク元やリンク先を表すために用いられる。また、本の栞のようにただ単に興味のある情報に印をつけるために用いられることもある。

定義 6 (リンク) リンクは点と点を結ぶ有向弧である *(点はアンカーにより表される)*。リンクは組$< id, k, s, d >$により表される。id は識別子、kはリンクの種類、s はリンク元の点、d はリンク先の点である。リンクはメタ変数 $l_1, \ldots, l_n$により表す。 □

リンクは 2 点を有向弧で結ぶことにより、ユーザが関連のある情報を効率良く検索 (航行) することを可能にする。リンク元とリンク先はアンカーでなければならない。アンカーはノードもしくはブロックに結びつけられるので、リンクには以下の 4 つの種類があることにな

[1] 詳細は定義 15を参照のこと。

る：1) ノード～ノード、2) ノード～ブロック、3) ブロック～ノード、4) ブロック～ブロック。リンクが直接ノードやブロックに結び付けられているではなく、間接的にアンカーを介して結び付けられていることにより、ノードやブロックの変更に対してより柔軟な対応をすることが出来る。

これらすべての実体または実体の集合は名前付けが可能である。名前とそれに対応する実体や実体の集合との対応付けを束縛と呼ぶ。名前はハイパーテキスト中で一意であると仮定する。よって、束縛は他の実体と異なり識別子を持たない。

定義 7 (束縛) 束縛は名前から実体または実体の集合への対応付けを表す。束縛は組$< name, e >$により表される。$name$は一意な名前、eは実体または実体の集合である。束縛はメタ変数$\beta_1, \ldots, \beta_n$により表す。 □

2.2 簡単な英語辞典の例

ここではハイパーテキスト上の簡単な英語辞典$\mathcal{H}$の例をあげる。1つのノードは1つの単語項目を表し、少なくとも3つのブロックを含んでいる。最初のブロックはタイトル、2つ目のブロックは品詞、3つ目以降のブロックは意味の説明文を表す複合ブロックである。したがってブロックは単語の意味の数だけ存在する。意味を表す複合ブロックは単語を含んだ原始ブロックの並びを持っていて、一連のブロックが1つの意味の説明文を表す(図1参照)。

$$
\begin{aligned}
node &= \text{各単語項目} \\
node[1] &= \text{タイトル} \\
node[2] &= \text{品詞} \\
node[3[1 \sim w_1]] &= \textit{1}\text{個目の意味の説明文} \\
node[4[1 \sim w_2]] &= \textit{2}\text{個目の意味の説明文} \\
&\vdots \\
node[m[1 \sim w_n]] &= n\text{個目の意味の説明文} \\
&\quad \text{ただし } m{=}n{+}2
\end{aligned}
$$

この英語辞典に含まれるノードとブロックは5種類のリンクによって互いに結びつけられる。リンクの種類を以下にあげる。

リンクの種類	リンク元	リンク先
synonym (同義語)	ノード	ノード
antonym (反意語)	ノード	ノード
refer (参照)	ノード	ノード
explained (説明)	ブロック	ブロック
derivative (派生語)	ノード	ノード

さて、この辞典$\mathcal{H}$に *'easy'* という単語が入っている場合を考える。この単語に対応する項目は一般的な英語辞典においては次のようである (ただしこの例は少し簡略化してある)。

easy *adj* **1** not difficult **2** free from pain
3 not much in demand

すると対応するノードとブロックの定義は次のようになる。

$$
\begin{aligned}
&< n_1, b_1 b_2 b_3 b_4 b_5, a_{b1} a_{b2} a_{b3} a_{b4} a_{b5} > \\
&\quad < b_1, \textit{'easy'}, a_1 > \\
&\quad < b_2, \textit{'adjective'}, a_2 > \\
&\quad < b_3, b_{31} b_{32}, a_{b31} a_{b32} > \\
&\qquad < b_{31}, \textit{'not'}, a_{31} > \\
&\qquad < b_{32}, \textit{'difficult'}, a_{32} > \\
&\quad < b_4, b_{41} b_{42} b_{43}, a_{b41} a_{b42} a_{b43} > \\
&\qquad < b_{41}, \textit{'free'}, a_{41} > \\
&\qquad < b_{42}, \textit{'from'}, a_{42} > \\
&\qquad < b_{43}, \textit{'pain'}, a_{43} > \\
&\quad \ldots
\end{aligned}
$$

さらに、*'easy'* の同義語 *'effortless'* がこの辞典に入っていて、そのノード識別子がn_2であるとすると、a_1とa_2という識別子を持つ次の2つのアンカーおよび1つのリンクl_1が定義される。

$$
\begin{aligned}
&< a_1, n_1 > \\
&< a_2, n_2 > \\
&< l_1, \textit{'synonym'}, a_1, a_2 >
\end{aligned}
$$

2.3 ハイパーテキストの操作

様々なアプローチや概念に基づいたハイパーテキストシステムが提案され実装されているが、そのほとんどにおいて基本的な操作は同じである [5]。基本的な操作とは、実体の生成、破壊、更新、ノード間の航行である。本論文ではさらにグループへの追加、削除や、実体の名前付け、連想などの操作を定義する。我々はこれらの操作を形式化しその意味を定義する。

2.3.1 基本的な操作

どのハイパーテキストシステムにも共通な操作として、生成、破壊、更新がある。これらの操作はハイパーテキストを文書作成ツールとして使う場合には必ず必要な

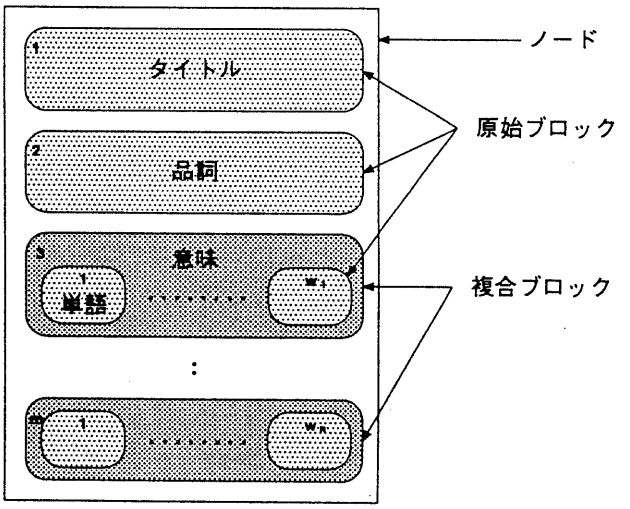

図 1: 簡単な英語辞典 $\mathcal{H}$のノードの構造

ものである。ハイパーテキストで文書を作成する場合、ユーザはまず必要なノードを作り、その中にデータを書き込み、それらのノードをアンカーやリンクにより結びつける。また、不必要な実体ができた場合、ユーザはそれを破壊するか、内容を書換えて再利用することができる。我々はこれらの操作に形式的な定義および意味を与える。

定義 8 (生成、破壊、更新) 実体の生成とは新たな実体を生成しハイパーテキストに加える操作である。この操作は関数 **create()** により表され、引数により生成する実体が指定される。この操作の意味は次のように形式的に定義される。

$$\frac{\mathcal{H} \quad \mathbf{create}(e)}{\mathcal{H} \cup \{e\}}$$

実体の破壊とはハイパーテキストから実体を削除する操作である。この操作は関数 **destroy()** により表され、引数により破壊する実体が指定される。この操作の意味は次のように形式的に定義される[2]。

$$\frac{\mathcal{H} \quad e \in \mathcal{H} \quad \mathbf{destroy}(e)}{\mathcal{H} - \{e\}}$$

実体の更新とはその属性を追加、削除、変更する操作である。この操作は関数 **update()** により表される。これは2つの引数をとり、1つ目に更新する実体を指定すると、2つ目に更新された実体が返る。この操作の意味は次のように形式的に定義される。

$$\frac{\mathcal{H} \quad e \in \mathcal{H} \quad \mathbf{update}(e, e')}{(\mathcal{H} - \{e\}) \cup \{e'\}}$$

□

2.3.2 グループに関する操作

共通な性質を持った実体の集合をグループとして扱うことが可能である。例えば、'*a*'で始まる単語を含んだノードの集合はグループとして扱える。また、動物の名前を表すノードに結びつけられたアンカーの集合もグループとなる。グループを扱うためには、グループに任意の実体を追加したり削除したりする操作が必要である。ここではそれらの操作の形式的な定義および意味を与える。

定義 9 (追加、削除) 追加とは指定したグループに実体を加える操作である。この操作は関数 **add()** により表され、引数によりグループと追加する実体を指定する。この操作の意味は次のように形式的に定義される。ただし、G はグループである。

$$\frac{G \quad \mathbf{add}(G, e)}{G \cup \{e\}}$$

[2]厳密には、破壊される実体が別の実体を含んでいた場合、再帰的に探索が行なわれ、それらが他の実体によって共有されていないかが調べられる。そして、共有されている実体は削除されず、共有されていない実体が削除される。しかし、紙面の制約があるためその詳細については割愛する。

削除とは指定したグループから実体を取り除く操作である。この操作は関数 **delete()** により表され、引数によりグループと削除する実体を指定する。この操作の意味は次のように形式的に定義される。

$$\frac{G \quad e \in G \quad \mathbf{delete}(G, e)}{G - \{e\}}$$

□

2.3.3 名前付けの操作

名前付けはとても重要な操作である。なぜなら任意の実体に名前付けが行なえることにより、ユーザは実体をその識別子のみならず、名前で指定することが可能になるからである。実体の集合もまた名前付けが可能である。例えば、動物の名前を表すノードの集合には *'animal'* という名前を付けることができる。我々は名前付けを以下のように形式化する。

定義 10 (名前付け) 名前付けとは実体または実体の集合に名前を割り当てる操作である。この操作は演算子 $\stackrel{\text{def}}{=}$ により表され、左辺に名前、右辺に実体または実体の集合を指定する。この操作の意味は次にように形式的に定義される。

$$\frac{\mathcal{H} \quad e \in \mathcal{H} \quad name \stackrel{\text{def}}{=} e}{\mathcal{H} \cup \{< name, e >\}}$$

$$\frac{\mathcal{H} \quad \{e_1, \ldots, e_n\} \in \mathcal{H} \quad name \stackrel{\text{def}}{=} \{e_1, \ldots, e_n\}}{\mathcal{H} \cup \{< name, \{e_1, \ldots, e_n\} >\}}$$

□

グループも実体の集合であるので、普通の実体の集合と同様に名前付けが可能である。例えば、*'a'* で始まる単語を含んだノードのグループは *'A'* と名前付けすることが可能である。

2.3.4 選択の操作

ハイパーテキスト中の任意の実体を選択するには2通りの方法がある。1つは直接、識別子や名前により指定する方法であり、もう1つは問い合わせを用いる方法である。我々は前者の方法を特に連想と呼び、以下のように形式化する。

定義 11 (連想) 連想は指定された実体または実体の集合を返す操作である。実体の指定はその識別子または名前で行なう。この操作は関数 **associate()** により表され、引数により連想する実体の識別子または名前を指定する。この操作の意味は次のように形式的に定義される。

$$\begin{aligned} \{e\} \in \mathcal{H} \quad &\vdash \quad \mathbf{associate}(id) \Rightarrow e \\ \{< name, e >\} \in \mathcal{H} \quad &\vdash \quad \mathbf{associate}(name) \Rightarrow e \\ \{< name, E >\} \in \mathcal{H} \quad &\vdash \quad \mathbf{associate}(name) \Rightarrow E \end{aligned}$$

ただし、$e =< id, x_1, \ldots, x_n >$、$E = \{e_1, \ldots, e_n\}$ である。 □

後者の問い合わせを用いる方法に関しては後の章で詳しく述べる。

2.3.5 航行

航行によるハイパーテキストのアクセスは最も一般的な方法である。ユーザは任意のリンクを選びノードやブロックを渡り歩くことが可能である。ただし、すべてのブロックはいずれかのノードに含まれているので、この操作は単にノード間の航行と捉えられる。航行の意味は、ユーザが現在着目しているノードへのポインタ、すなわち航行ポインタを定義することにより形式化できる。

定義 12 (航行ポインタ) 航行ポインタはハイパーテキスト $\mathcal{H}$中の1つのノードを指し示す。特別な変数 NPにより航行ポインタを表すが、その値は航行ポインタが指し示すノードの識別子である。 □

航行ポインタはノードの中身を表示するウィンドウと捉えることができる。航行とはユーザがハイパーテキスト中でノードからノードへと移動する操作であるので、このポインタの値を現在のノードから次のノードへと変化させる操作として形式化することができる。

定義 13 (航行) 航行は NPの値を更新する操作である。NPがノード n を指し示し、n 中に適当なアンカー a が結びつけられていて、そのアンカーに対して適当なリンク lが定義されているような $\mathcal{H}$において、ユーザはノード n からアンカー a'を含んでいるような別のノードへリンク lを通って航行することができる。この操作は関数 **navigate()** により表される。引数としてはリンク l の種類とアンカー a の識別子を与える。そして、航行の意味は次の2つの式により形式的に定義される。

$$\frac{\begin{array}{ll} NP = n & \exists a \in anchor(n) \\ \exists l \in link(\exists kind, a, \exists a') & \\ \multicolumn{2}{c}{\mathbf{navigate}(a, kind)} \end{array}}{NP = node(a')}$$

$$\frac{NP = n \qquad n \in implies(\exists b) \quad \exists a \in anchor(b) \quad \exists l \in link(\exists kind, a, \exists a') \quad \mathbf{navigate}(a, kind)}{NP = node(a')}$$

これらの式においては以下の関数[3]が仮定されている。

- $node(x) : Anchor \rightarrow Node$
  アンカー x が結びつけられているノードを返す。
- $implies(x) : Block \rightarrow$ **Set of** $Nodes$
  ブロック x が含まれているノードの集合を返す。
- $anchor(x) : Node\ or\ Block \rightarrow$ **Set of** $Anchors$
  ノードまたはブロック x に結びつけられているアンカーの集合を返す。
- $link(k, s, d) : Kind \times Anchor \times Anchor \rightarrow$ **Set of** $Links$
  リンク元がアンカー s でリンク先がアンカー d でその種類が k であるようなリンクの集合を返す。

□

上記の式により、あるノードから別のノードへと、ノード内のブロックから他のノードへという2種類の航行が定義された。ノードからブロックへと、ブロックからブロックへの2種類の航行に関する定義は示さないが、航行の基本的な単位はノードであるので、リンク先のブロックが属するノードが航行の後にアクセスされるものと仮定する。

3 問い合わせ

問い合わせの形式的な表記に関しては2通りの方法がある。1つはいくつかの代数演算子を定義して代数により表現する方法である。もう1つの方法は論理表現を用いて論理式により表現する方法である。多くの場合、代数で表現された問い合わせの意味は論理表現で与えられる [7, 15] ので、代数表現を用いる場合にはそれを論理表現に変換する必要が出てくる。厳密には、定義した代数演算子における完全性および健全性を、変換して得た論理表現においても示す必要がある。しかし紙面の都合上、本論文では論理表現を直接用いることにする。実装段階の際は、問い合わせの効率的な解釈のためにも代数演算子を定義しなければならない。本章で定義する論理表現は Reiter[14] によって定義されたものと似ている。本論文では問い合わせの表現能力は関係論理程度に押える。なぜなら再帰を含む問い合わせなどは、不動点理論 [13, 18] などの複雑な理論が必要であり、紙面が限られているためここで述べることができないからである。以降では簡単な問い合わせ言語の定義をいくつかの例と共に示す。まず、最初に構文規則の定義を示し、次にモデル理論に基づいた意味の定義を示す。

3.1 問い合わせの構文規則

我々の簡単な問い合わせ言語においては、すべての問い合わせは整合論理式 (well formed formula) を用いて表される。問い合わせ言語の構文規則は、リテラルや項などの基本的な部分の定義を最初に行ない、それらに基づき全体を定義する。また、本章の後半で言語の意味を定義するために自然意味論 [10] を用いる。自然意味論においては、BNF 表記の中に現れる非終端記号は構文規則におけるクラスであり、その言語で記述されたスクリプトはインスタンスである。これ以降の定義においては非終端記号を構文規則におけるクラスとして扱うことがある。

3.1.1 基本的な構成要素

定義 14 (リテラル) この言語のリテラルは英数字と記号とからなり、変数、定数、述語の3つの種類がある。

- 変数は英文字からなる。
- 定数は文字列、数値、もしくは構造を持った定数である。
- 述語はハイパーテキスト $\mathcal{H}$ においてグローバルに定義された名前である。

以下のメタ変数は各クラスのインスタンスを表すのに用いる。

$$\begin{aligned} Variable &::= x, y, z, \ldots \\ Constant &::= a, b, c, \ldots \\ Predicate &::= P, Q, R, \ldots \end{aligned}$$

□

文字列の比較には UNIX オペレーティングシステムにおける正規表現が用いられるものとする。例えば、文字定数 '.*tion' は 'tion' で終る単語を表す (e.g. 'definition' や 'contribution' など)。また、連接演算子 ‖ を文字定数に適用することができる。例えば、'data' ‖ 'base' は 'database' に評価される。

[3] 本来ならば、これらの関数は次の章で導入される演算子のように形式的にその意味を与えるべきであるが、ここでは紙面の制約により割愛する。

組み込み述語　はシステムによってあらかじめ定義された特別な述語で、定数を比較するのに用いられる。例えば、以下のような演算子が用意されていて、数値を比較したり文字列を辞書順に比較したりするのに用いられる。

$$= \quad \neq \quad \leq \quad \geq \quad > \quad <$$

定義 15 (項の構文規則) ハイパーテキスト $\mathcal{H}$ 中のノードやブロックは項により表現する。定数や並びや変数も項である。

Term	::=	*Node Specification*
	|	*Block Specification*
	|	*Node Specification*[*Block Specification*]
	|	*Sequent Constructor*
	|	*Variable*
	|	*Constant*
Node Specification	::=	*Identifier Expression*
Identifier Expression	::=	#(*Cardinal Expression*)
	|	*Variable*
Block Specification	::=	*Block Index*[*Block Specification*]
	|	*Block Index*
Block Index	::=	*Cardinal Expression*
	|	˜ *Cardinal Expression*
	|	*Cardinal Expression* ∼ *Cardinal Expression*
Cardinal Expression	::=	*Integer Constant*
	|	*Variable*
	|	*Cardinal Expression* + *Cardinal Expression*
	|	*Cardinal Expression* − *Cardinal Expression*
	|	*Cardinal Expression* × *Cardinal Expression*
	|	*Cardinal Expression* ÷ *Cardinal Expression*
	|	− *Cardinal Expression*
	|	(*Cardinal Expression*)
Sequent Constructor	::=	[*Term*]
	|	[*Term*, ... , *Term*]
	|	!(*Term*)
	|	*Term* ‖ *Term*

□

構文規則クラス *Node Specification* は問い合わせにおいてアクセスするノードを指定するのに用いられる。特定のノードを指定する場合、識別子により直接指定することも可能である。例えば、#(5624) と #(703 × 8) は同一のノードを表す。

構文規則クラス *Block Specification* はノードから任意のブロックを取り出すのに用いられる。例えば、*a*[24[5]] はノード *a* の 24 番目のブロックの中の 5 番目のブロックを表す。構文規則クラス *Block Specification* に対して、構文規則クラス *Sequent Constructor* を用いると実体の並びを項として表すことができる。並びを記述する演算子としては、*Sequent Constructor*, *Unfold Operators*, 連接演算子の 3 種類がある。これらの演算子の形式的な定義は後で述べるが、ここではいくつかの直観的な例を示す。まず、*a* が [*'Intuition'*, [*'noun'*], [*'immediate'*, *'understanding'*]] という並びを表すと仮定する。そこで *a* に対していくつかの演算を行なうと以下のようになる。

Block Specification	$a[3[2]]$	=	*'understanding'*
Sequent Constructor	$[[a[1]], a[2]]$	=	$[[$*'Intuition'*$], [$*'noun'*$]]$
Unfold Operator	$!(a)$	=	[*'Intuition', 'noun', 'immediate', 'understanding'*]
連接演算子	$a[2]\|\|a[3]$	=	[*'noun', 'immediate', 'understanding'*]

3.1.2 論理式

定義 16 (原子式の構文規則) この言語におけるすべての述語は原子式 (Atomic Formula) の構文規則に従う。述語に渡される引数の数は述語の序数に対応する。

$$\textit{Atomic Formula} \quad ::= \quad \textit{Predicate}(\textit{Term}, \ldots, \textit{Term})$$

□

省略形: 原子式が 1 つ以上の引数を持つとき、それは引数の並びとして考える。原子式は次式と構文的に等価である。

$$p(t_1, \ldots, t_n) \equiv p([t_1, \ldots, t_n])$$

ただし、$p \in \textit{Predicate}$、$t_1, \ldots, t_n \in \textit{Term}$である。

例えば、組み込み述語の 1 つである 2 項関係=は 2 つの引数をとるので、2 つの実体 a_1, a_2の等価関係は$= (a_1, a_2)$として表す。前章で用いた例に基づき、いくつかの原子式の例を示す。ただし、$\textit{title} = 1$で$\textit{functional label} = 2$で$n$はノードを表す。

$$\begin{aligned} &\leq (n[\textit{title}], \textit{'elephant'}) \\ &\neq (n[\textit{functional label}], \textit{'noun'}) \\ &= (n[\textit{title}], \textit{'*tion'}) \\ &= (\textit{'machine'}, n[3[1]]) \end{aligned}$$

原子式においては、組み込み述語として用意されている 2 項関係以外に、ハイパーテキスト $\mathcal{H}$中のグループの名前を述語として用いることができる。例えば、前章で用いた簡単な英語辞典の部分集合を Pとすると、$P(x)$ は x が Pに含まれているかどうかを表す。

定義 17 (整合論理式の構文規則) 以下に示す構文規則において限量子 ($\forall$ または $\exists$) に束縛された変数の有効範囲はそれに続く整合論理式に限られるものとする。

$$\begin{aligned} \textit{Well Formed Formula} \quad ::= \quad & \textit{Atomic Formula} \\ \mid \quad & (\textit{Well Formed Formula} \wedge \textit{Well Formed Formula}) \\ \mid \quad & (\textit{Well Formed Formula} \vee \textit{Well Formed Formula}) \\ \mid \quad & (\textit{Well Formed Formula} \rightarrow \textit{Well Formed Formula}) \\ \mid \quad & \neg\ \textit{Well Formed Formula} \\ \mid \quad & (\exists\ \textit{Variable})(\textit{Well Formed Formula}) \\ \mid \quad & (\forall\ \textit{Variable})(\textit{Well Formed Formula}) \\ \mid \quad & (\exists\ \textit{Variable}/\textit{Boundary})(\textit{Well Formed Formula}) \\ \mid \quad & (\forall\ \textit{Variable}/\textit{Boundary})(\textit{Well Formed Formula}) \\ \textit{Boundary} \quad ::= \quad & \textit{Predicate} \end{aligned}$$

□

例えば、以下の式はすべて整合論理式である。

- $(\forall x/P)(\leq (x, 1000))$
  グループ Pに含まれるすべての実体が 1000 より小さいかどうかを表す。

- $(\exists x)((Q(x) \rightarrow P(x))$
  グループ Q に含まれていれば必ずグループ Pに含まれているという実体が存在するかどうかを表す。

省略形: 限量子において *Boundary* が指定された場合、整合論理式は以下のように置き換えられる。

$$\begin{aligned} (\forall x/\tau)(W) &\equiv (\forall x)(\tau(x) \rightarrow W) \\ (\exists x/\tau)(W) &\equiv (\exists x)(\tau(x) \wedge W) \end{aligned}$$

ただし、$x \in \textit{Variable}$、$\tau \in \textit{Boundary}$、$W \in \textit{Well Formed Formula}$である。

3.1.3 問い合わせ

定義 18 (問い合わせの構文規則) 問い合わせは構文規則 *Qualifier* を用いることにより記述される。

Qualifier	::=	<*Constructor* | *Well Formed Formula*>
Constructor	::=	*Candidate*
	|	[*Candidate*]
	|	[*Candidate*, ... , *Canadidate*]
	|	!(*Candidate*)
	|	*Candidate* || *Candidate*
Candidate	::=	*Node Specification*
	|	*Node Specification/Boundary*
	|	*Block Specification*
	|	*Block Specification/Boundary*

□

省略形: *Candidate* において *Boundary* が指定された場合、以下の規則に基づいて置き換えられる。

$$< x/\tau \mid W > \equiv < x \mid \tau(x) \wedge (W) >$$

ただし、$x \in$ *Node Specification or Block Specification*、$\tau \in$ *Boundary*、$W \in$ *Well Formed Formula* である。

構文規則クラス *Qualifier* を用いることにより様々な問い合わせを柔軟に記述することが可能である。例えば、我々の問い合わせ言語では以下のような高度な問い合わせを自然に記述することができる。

- $Q \stackrel{\text{def}}{=}$
  $< x \mid (\exists i)((\exists j)(\geq (i,3) \wedge = (x[i[j]], \text{'animal'}))) >$
  意味の説明文の中に 'animal' という単語を含んだノードを検索しグループ Q とする。
- $< x[\textit{functional label}] \mid = (x[\textit{title}], \text{'dis.*'}) >$
  'dis' で始まる単語の品詞を検索する。
- $< x/Q \mid (\forall y)(\geq (x[\textit{title}], y[\textit{title}])) >$
  グループ Q 中で辞書順で最後のノードを検索する。
- $< x \mid (\exists y)((\exists i)((\exists j)(\geq (i,3) \wedge = (y[i[j]], \textit{'sing'})$
  $\wedge = (x, y[i[j+1]])))) >$
  すべてのノードの説明文において 'sing' という単語の次に来る原始ブロックを検索する。

3.2 ハイパーテキストにおける問い合わせの意味

我々は問い合わせの意味を定義するために、変数の変域すなわち領域 (domain) を定義する。そしてこの領域に基づき、モデル理論的な意味を構築する。ただし、紙面に限りがあるため本論文では健全性と完全性の証明は行なわない。しかし、それらが必要になった場合は本論文の付録に付ける。

定義 19 (有効領域) 有効領域 (effective domain) D は以下のように定義される。

- D は $\mathcal{H}$ 中の実体のグループである。または、
- D は定数の可算無限または有限な領域である。または、
- D は並びのグループであり各並びの各要素はいずれかの有効領域に属する。

$c \in D$ は定数 c が有効領域 D に属することを表す。 □

我々は問い合わせの意味に関して健全な領域を定義するために、特別に有効領域の概念を取り入れた。グループに関して入れ子は考えないが、有効領域に関しては入れ子も考えることにする。どの有効領域においてもその要素は既存の実体や定数や並びから構築することができる。ハイパーテキスト $\mathcal{H}$ のグループのみがグループを要素として持つことができる。ここではまず、有効領域に関するいくつかの演算や関係を示し、後にそれらを用いて意味を定義する。

定義 20 (有効領域の操作および関係) 有効領域に関する和、積、差は以下のように定義される。これらの演算の結果もまた有効領域である。包含関係の定義もその下に示す。

$$\forall x \in D\ \forall y \in E \ \vdash \ x \in D + E\ y \in D + E$$
$$\forall x \in D\ \forall y \in E \ \vdash \ x \,\|\, y \in D \times E$$
$$\forall x \in D\ \forall y \in E \ \vdash \ \textbf{if } x = y \textbf{ then } x \notin D - E$$

$$\textbf{else } x \in D - E$$
$$D \subseteq E \ \vdash \ \forall x \in D, x \in E$$

ただし、$A \vdash B$は「A のもとで B が演繹できる」ことを意味する。 □

省略形: 次の省略形は述語に渡される引数の領域を定義するのに用いられる。

$$\prod_{i=1}^{n} D_i \equiv (\ldots((D_1 \times D_2) \times D_3) \times \ldots) \times D_n$$

次の定義において問い合わせの解釈を導入する。解釈は問い合わせの記述を並び (またはノード) の外延的なグループに変換する。

定義 21 (マッピングと解釈) 我々の問い合わせ言語 $Q = (Literal, Qualifier)$ の**解釈**は組 $I = (D, K, E)$ によって表される。ただし、

- Dは Q に含まれる変数の変域を表す有効領域である。
- Kは定数リテラルから D上の定数へのマッピングである (i.e. 各 $c \in Constant$に関して $K(c) \in D$)。
- Eは Q に含まれる述語から Dの要素のグループへのマッピングである (i.e. 序数が n の各 $P \in Predicate$に関して $E(P) \subseteq \prod_{i=1}^{n} D_i$)。$E(P)$ は解釈 Iにおいて Pの**外延**と呼ばれる。

関数ρは問い合わせに現れる変数を適当な有効領域の定数へと変換する。ρは変数の有効範囲にともなって変わる。有効範囲内のρは**変数環境**と呼ばれる。次の式は変数環境のもとで Q の構文規則が意味へと解釈されることを表している。

$$\vdash_\rho Syntax \Rightarrow_I Meaning$$

変数環境とは別に新たな環境を加える場合は次のように記述される。

$$Environment \vdash_\rho Syntax \Rightarrow_I Meaning$$

□

ここで、各構文規則に対する解釈を具体的に示す。リテラルに関する解釈は既に前の定義で示した通りである。

定義 22 (項の意味) 以下の解釈の定義においては、各メタ変数はそれぞれその後に続く構文規則クラスに属するものとする。

$$i, i_1, i_2 \in Block\ Index$$
$$b \in Node\ or\ Block$$
$$a, a_1, a_2, \ldots, a_k, a_{k+1}, \ldots, a_{n-1}, a_n \in Term$$

各メタ変数は適当な有効領域における定数を表す。

$$\begin{aligned}
b = [a_1, \ldots, a_n] \vdash_\rho b[i] &\Rightarrow_I a_i \\
b = [a_1, \ldots, a_n] \vdash_\rho b[\hat{}\, i] &\Rightarrow_I [a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n] \\
b = [a_1, \ldots, a_n], i_1 \leq i_2 \vdash_\rho b[i_1 \sim i_2] &\Rightarrow_I [a_{i_1}, \ldots, a_{i_2}] \\
\vdash_\rho [a] &\Rightarrow_I [a] \\
\vdash_\rho [a_1, \ldots, a_n] &\Rightarrow_I [a_1, \ldots, a_n] \\
\vdash_\rho a_1 \parallel a_2 &\Rightarrow_I [a_1, a_2] \\
\vdash_\rho a_1 \parallel [a_2, \ldots, a_n] &\Rightarrow_I [a_1, a_2, \ldots, a_n] \\
\vdash_\rho [a_1, \ldots, a_{n-1}] \parallel a_n &\Rightarrow_I [a_1, \ldots, a_{n-1}, a_n] \\
\vdash_\rho [a_1, \ldots, a_k] \parallel [a_{k+1}, \ldots, a_n] &\Rightarrow_I [a_1, \ldots, a_n] \\
\vdash_\rho !(a) &\Rightarrow_I a \\
\vdash_\rho !([a]) &\Rightarrow_I a \\
\vdash_\rho !([a_1, \ldots, a_n]) &\Rightarrow_I [c_1 \parallel \ldots \parallel c_n] \\
& \quad \text{ただし } c_i = !(a_i) \text{ s.t. } 1 \leq i \leq n
\end{aligned}$$

□

有効領域の概念を用いることにより、次の定義において我々の言語のモデル理論的解釈を示すことが可能となる。

定義 23 (原子式の意味) 以下の定義において、D_iは定数 c_iに対応する有効領域を表し、Pは構文規則クラス *Predicate* のインスタンスを表す。2 つの特別な値 **true** と **false** は Boolean 有効領域の定数である。

$$[c_1,\ldots,c_n] \in P, P \subseteq \prod_{i=1}^{n} D_i \quad \vdash_\rho P(c_1,\ldots,c_n) \Rightarrow_I \textbf{true}$$

$$[c_1,\ldots,c_n] \notin P, P \subseteq \prod_{i=1}^{n} D_i \quad \vdash_\rho P(c_1,\ldots,c_n) \Rightarrow_I \textbf{false}$$

□

原子式の中に組み込み述語=などが現れた場合においても、積の部分領域 (subdomain) として部分的な有効領域が作られる。例えば、$=(x_1,x_2)$ は x_1が常に x_2と同じであるような有効領域であると考えられる。

定義 24 (整合論理式の意味) 以下の定義において W, W_1, W_2は構文規則クラス *Well Formed Formula* のインスタンスを表す。メタ変数ν, ν_1, ν_2は Boolean 定数 (**true** または **false**) である。意味 (*Meaning*) の中における演算子 **and** と **or** と **not** は通常の論理学上の意味に従う。Dは有効領域を表す。

$$\textbf{and}\ \frac{D \vdash_\rho W_1(e_1) \Rightarrow_I \nu_1 \quad D \vdash_\rho W_2(e_2) \Rightarrow_I \nu_2}{D \vdash_\rho W_1(e_1) \wedge W_2(e_2) \Rightarrow_I \nu_1 \ \textbf{and}\ \nu_2}$$

$$\textbf{or}\ \frac{D \vdash_\rho W_1(e_1) \Rightarrow_I \nu_1 \quad D \vdash_\rho W_2(e_2) \Rightarrow_I \nu_2}{D \vdash_\rho W_1(e_1) \vee W_2(e_2) \Rightarrow_I \nu_1 \ \textbf{or}\ \nu_2}$$

$$\textbf{imply}\ \frac{D \vdash_\rho W_1(e_1) \Rightarrow_I \nu_1 \quad D \vdash_\rho W_2(e_2) \Rightarrow_I \nu_2}{D \vdash_\rho W_1(e_1) \rightarrow W_2(e_2) \Rightarrow_I \textbf{not}\ \nu_1\ \textbf{or}\ \nu_2}$$

$$\textbf{not}\ \frac{D \vdash_\rho W(e) \Rightarrow_I \nu}{D \vdash_\rho \neg W(e) \Rightarrow_I \textbf{not}\ \nu}$$

限量子 ($\forall$ または $\exists$) を伴った整合論理式は、限量された変数が局所的な有効範囲を持つ新たな変数環境を作る。

$$\textbf{forall}\ \frac{\forall d \in D \vdash_\rho W(d) \Rightarrow_I \textbf{true}}{D \vdash_\rho (\forall x)W(x) \Rightarrow_I \textbf{true}} \qquad \frac{\exists d \in D \vdash_\rho W(d) \Rightarrow_I \textbf{false}}{D \vdash_\rho (\forall x)W(x) \Rightarrow_I \textbf{false}}$$

$$\textbf{exists}\ \frac{\exists d \in D \vdash_\rho W(d) \Rightarrow_I \textbf{true}}{D \vdash_\rho (\exists x)W(x) \Rightarrow_I \textbf{true}} \qquad \frac{\forall d \in D \vdash_\rho W(d) \Rightarrow_I \textbf{false}}{D \vdash_\rho (\exists x)W(x) \Rightarrow_I \textbf{false}}$$

□

定義 25 (問い合わせの意味) 以下の定義において、Sと Dは有効領域を表す。メタ変数 x は有効領域における定数を表す。yは構文規則クラス *Constructor* に属する。$W(x)$ は構文規則クラス *Well Formed Formula* の引数 x を持ったインスタンスである。

$$\frac{S \subseteq D \vdash_\rho \forall x \in S,\ W(x) \Rightarrow_I \textbf{true} \quad S \subseteq D \vdash_\rho \forall x \in (D-S),\ W(x) \Rightarrow_I \textbf{false}}{D \vdash_\rho < y \mid W(y) > \Rightarrow_I S}$$

□

上記の定義を用いてハイパーテキスト $\mathcal{H}$における問い合わせの意味を次のように定義する。

定義 26 (ハイパーテキストにおける問い合わせ) 構文規則クラス *Qualifier* のインスタンス qが環境 $\mathcal{H}$のもとでSに解釈されるとすると、$\mathcal{H}$において発行された問い合わせ qは S に評価される。

$$\frac{\mathcal{H} \quad \mathcal{H} \vdash_\rho q \Rightarrow_I S}{\mathcal{H} \quad \mathcal{H} \vdash_\rho \textbf{evaluate}(q) \Rightarrow_I S}$$

□

我々の問い合わせ言語の完全性および健全性を示すためには次の推論規則 (公理) が必要である。紙面の制約上、本論文で意味についてはこれ以上述べないことにする。

推論規則 27 (問い合わせ解釈の一様性)

$$\frac{D \vdash_\rho q \Rightarrow_I S}{D'(\subseteq D) \vdash_\rho q \Rightarrow_I S'(\subseteq S)}$$

ただし、$q \in$ *Qualifier*である。 □

4 関連研究

4.1 ハイパーテキストのデータモデル

一部の研究者によってハイパーテキストの形式的な記述に関する提案がなされている。Tompa[16] はhypergraph[4] に基づいたデータモデルを提案している。Campbell によって提案されたHAM[5] では、強力なグラフベースのデータモデルがサポートされており、フィルタと呼ばれる問い合わせ機構が用意されている。しかし、このフィルタ機構はコンテキスト、ノード、リンクの属性にしか適用することができない。Garg[8] はハイパーテキストを一階述語の式の集合としてモデル化したが、彼のモデルはノード間のリンクしかサポートしておらず、さらに航行の意味を含んでいない。Afrati[2] はGarg のモデルを一般化して、構造を持ったノードを取り入れたり、リンクにスクリプトを記述できるよう拡張した。彼のモデルはハイパーテキストの構造に対する強力な問い合わせ機構を提供しているが、情報そのものに対する問い合わせはあまり考慮されていない。既に述べた通り、ハイパーテキストにおいて情報を検索する場合、航行機構と情報に対する強力な問い合わせ機構の両方が不可欠である。我々の提案する問い合わせ機構により、ハイパーテキストの情報検索能力を増すことが可能である。

4.2 複雑なデータ構造に対する問い合わせ

本論文では関係論理と等価な集合検索能力を持つ問い合わせ言語を定義した。演繹データベースの研究分野においては、再帰を含む問い合わせや、否定を含んだ層状検索が可能である [13, 18]。例えば、再帰を含んだ問い合わせを用いることにより推移的閉包が簡単に表現できる。我々はこれらの高度な問い合わせをサポートするためには、提案した問い合わせ言語の表現能力の拡張が必要だと考えている。もちろん、再帰的問い合わせがハイパーテキストの集合検索において、どの程度実質的に有効であるかという点に対しては疑問の余地がある。しかしながら、オブジェクト指向データベースと同様に、ブロックの構造が多重化して、非循環グラフを形成するようなノードを持つハイパーテキストを扱う上では、そのグラフ上のどの位置にあるかということも含めて検索できる要求は基本的なものと考えることができるだろう。我々の言語では論理表現を用いているので、拡張は自然でとても簡単なものになるだろう。さらに、拡張した言語には正しい意味を与える必要があるが、拡張された後の我々の検索言語に対しても自然に正確な意味を与えることができるであろう。いくつかの高度な問い合わせ言語が、演繹オブジェクト指向データベースの分野において提案されている [1, 3, 6, 7, 11, 15]。これらの問い合わせ言語においては、複雑な構造を持った実体の集合に対する問い合わせが可能である。個々の実体はオブジェクト [1, 3, 7, 15] または複合項 (complex term)[6, 11] の概念を用いてモデル化されている。しかし、我々は主にインデックスの付けられる並びを用いて実体をモデル化した。インデックスを用いることによりパラメータ化が可能であるので、ノードから入れ子のブロックへのアクセスの方が、オブジェクトからメンバへのアクセスより強力であると考えたからである。HiLog では高階の問い合わせを記述することができる [6]。ハイパーテキストの集合検索において、高階の問い合わせを取り入れるかどうかについては、その妥当性も含めて、考察が必要である。

5 結論および将来への展望

本論文では、構造とそれに適用される操作の2つの観点からハイパーテキストの形式化を行ない、ハイパーテキストに形式的なモデルを与えた。そして、そのモデルを用い、従来のハイパーテキストの航行機構の欠点を補う情報検索機構として強力な問い合わせ機構を提案した。本論文で提案されたハイパーテキストの形式的な意味論は、具体的なハイパーテキストシステムを設計する上で、その機能の厳密な仕様を表現するための手段と成り得るものである。

我々の今後の予定は、本論文で提案した問い合わせ機構を応用することにより、視点機構を構築することである。視点機構は、問い合わせを用いてハイパーテキストから任意の情報を抽出することにより、様々な視点に基づいたハイパーテキストをユーザが構築することを可能にする。さらに、この視点機構は従来の関係データベースにあるような固定された視点ではなく、ユーザのノード間の航行に伴い動的に変化する性質を持ったものである必要がある。また、[13] に示されている再帰的な問い合わせ機構を導入することも予定している。

参考文献

[1] Abiteboul, S., and Grumbach, S. COL: A Logic-Based Language for Complex Objects. In *Workshop on Database Programming Languages* (1987), pp. 253–276.

[2] AFRATI, F., AND KOUTRAS, C. D. A Hypertext Model Supporting Query Mechanisms. In *Hypertext: Concepts, Systems and Applications*, N. Streitz, A. Rizk, and J. André, Eds., Cambridge University Press, Cambridge, UK, 1990, pp. 52–66. Proceedings of the European Conference on Hypertext.

[3] BANCILHON, F., CLUET, S., AND DELOBEL, C. A Query Language for the O_2 Object-Oriented Database System. In *Proceedings of the 2nd International Workshop on Database Programming Languages* (1989), R. Hull, R. Morrison, and D. Stemple, Eds., Morgan Kaufmann Publisher, pp. 122–138.

[4] BERGE, C. *Graphs and Hypergraphs.* American Elsevier, New York, 1973.

[5] CAMPBELL, B., AND GOODMAN, J. M. HAM: A General Purpose Hypertext Abstract Machine. *Communications of the ACM 31*, 7 (July 1988), 856–861.

[6] CHEN, W., KIFER, M., AND WARREN, D. S. HiLog as a Platform for Database Languages (or why predicate calculus is not enough). In *Proceedings of the 2nd International Workshop on Database Programming Languages* (1989), R. Hull, R. Morrison, and D. Stemple, Eds., Morgan Kaufmann Publishers, pp. 315–329.

[7] CLUET, S., DELOBEL, C., LÉCLUSE, C., AND RICHARD, P. Reloop, an Algebra Based Query Language for an Object-Oriented Database System. In *Proceedings of the 1st Conference on Deductive and Object-Oriented Databases* (1989), pp. 294–313.

[8] GARG, P. K. Abstraction mechanisms in hypertext. *Communications of the ACM 31*, 7 (July 1988), 862–870.

[9] HALASZ, F. G. Reflections on NoteCards: Seven Issues for the Next Generation of Hypermedia Systems. *Communications of the ACM 31*, 7 (July 1988), 836–852.

[10] KAHN, G. Natural Semantics. In *Proceedings of 4th Annual Symposium on Theoretical Aspects of Computer Science* (1987), G. Goos and J. Hartmanis, Eds., Springer-Verlag, pp. 22–39. LNCS Volume 247.

[11] KIFER, M., AND LAUSEN, G. F-logic: A higher-order language for reasoning about objects, inheritance and schema. In *Proceedings of the ACM Symposium on Principles of Database Systems* (1989), ACM Press, pp. 231–239.

[12] MEYROWITZ, N. Hypermedia and the Desktop of Tomorrow. In *FRIEND21 Workshop* (October 1990).

[13] MINKER, J., Ed. *Foundations of Deductive Databases and Logic Programming.* Morgan Kaufmann Publishers, 1988.

[14] REITER, R. "Towards a Logical Reconstruction of Relational Database Theory". In *On Conceptual Modelling*, M. L. Brodie, J. Mylopoulos, and J. W. Schmidt, Eds., Springer-Verlag, 1984, ch. 8, pp. 191–238.

[15] SHAW, G., AND ZDONIK, S. An Object-Oriented Query Algebra. In *Proceedings of the 2nd International Workshop on Database Programming Languages* (1989), R. Hull, R. Morrison, and D. Stemple, Eds., Morgan Kaufmann Publisher, pp. 103–112.

[16] TOMPA, F. W. A Data Model for Flexible Hypertext Database Systems. *ACM Transactions on Information Systems 7*, 1 (January 1989), 85–100.

[17] TRIGG, R. H. Guided Tours and Tabletops: Tools for Communicationg in a Hypertext Environment. *ACM Transactions on Office Information Systems 6*, 4 (October 1988), 398–414.

[18] ULLMAN, J. D. *"Principles of Database and Knowledge Base Systems"*. Vol. 1,2, Computer Science Press, 1988.