

文学の実装に関する試論 物語世界の登場人物、およびその固有名の、オブジェクト指向プログラミング言語による設定について

赤間啓之、三宅真紀

東京工業大学大学院社会理工学研究科人間行動システム専攻
Email: { akama, mmiyake } @dp.hum.titech.ac.jp

概要

物語世界をコンピュータ上で自動生成するシステムを構想するにあたり、フィクションとしての物語が存立する場を問う哲学的議論、とくに、いわゆる「可能世界」をめぐってその根源的な条件を追究する議論は、きわめて重要である。Saul A・Kripkeは、『名指しと必然性 様相の形而上学と心身問題』において、可能世界論という形で、まさしく登場人物の固有名に関する哲学的な問題を取り上げた。われわれは、それに対し、コンピュータサイエンスの分野において、ことにオブジェクト指向プログラミング言語の中に、その問題をとらえなおそうと思う。とりわけ、Kripkeが回避した「分身」、あるいは偶発的同一性のテーマは、一般言語学の祖、Ferdinand de Saussureの言う、形相的同一性と実質的同一性の本質的な区別と関わるので、われわれは、SaussureとKripkeの議論を、本質的に同一なものとして、Java言語によるソースコードの形で記述することになる。

Keywords: オブジェクト指向プログラミング、可能世界論、偶発的同一性、形相的同一性、固有名、実質的同一性

An Essay on the Loading of Literature --The Setting of Characters and
Their Proper Names in the Story World [Narrative Universe]
by The Object-oriented Programming Language

Hiroyuki Akama, Maki Miyake
(Department of Human System Science, Tokyo Institute of Technology)
Email: {akama, mmiyake}@dp.hum.titech.ac.jp

Abstract

When planning a system which forms the story world automatically on the computer, the philosophical argument to locate the point where the story (i.e. fiction) occurs becomes very relevant. It is the argument which investigate, among others, the fundamental conditioning of "the possible world". Saul A・Kripke took up in "Naming and Necessity" a philosophical problem about the proper name of the character as an aspect of the possible world theory. We will take up the same problem in the field of computer science, more specifically in the object-oriented programming language. We will discuss that the "counterpart (alter ego)" concept or the contingent identity theme which Kripke avoided concerns the essential distinction between formal identity and substantial identity--distinction proposed by Ferdinand de Saussure, one of the founders of general linguistics. We will try to show the homology of the arguments of Saussure and Kripke through the use of source codes in the Java language.

1. 固有名と固定指示

本稿では、小方孝(1999)ら LCC 関係者が進めているような、コンピュータ上で物語世界を自動生成するシステムの構想にあたり、認知科学の視点と合わせ、従来の文学・哲学の知見に照らして、真剣な考慮に値するであろう、ごく基礎的な問題をひとつ取り上げる。それは、フィクション(仮構)としての物語が存立する場(トポス)を問う哲学的議論とくに、いわゆる「可能世界(possible world)」、すなわち「現実が別様だったなら何が起こり得たか」という反実仮想の世界をめぐる、その根源的な条件を追究する議論である。

よく知られているように、Saul A・Kripke は、『名指しと必然性 様相の形而上学と心身問題』において、可能世界論という形で、まさしく登場人物の固有名に関する哲学的な問題を取り上げた。彼はそこで、後に Hinrich Sch tze らの潜在的意味分析(LSA)がそうしたように、可能世界を現実(事実)世界への参照が可能な座標空間として、その中に登場人物と物語世界を連帯させて位置付けている(註1)。そこに、Kripke の決定的な弱点があるが、それはいろいろな発展性を秘めた「弱点」であり、論者は、その発展的可能性を、今日的な視野から、オブジェクト指向プログラミング言語の中に求めたいと思う。

周知のように、Kripke の言う「固有名(proper name)」は「固定指示子(rigid designator)」と同義であり、常に同一のものを指示する同一の操作子である。Kripke は、固有名があらゆる反事実的状況(可能世界)においても、固定的に同一の人物を指示する、と規定した。つまり、ある特定の人物が、彼とともに想像された可能世界ごとに、異なる反事実的属性を持つのはかまわないとしても、彼(ないし彼女の)の固有名は、その人物の存在意義を示すと一般に考えられている確定記述とは、いかなる場合もけっして置き換えられない。

たとえば、アリストテレス(という固有名)は、Russell-Frege らが考えるように、イコール「古代最後の偉大な哲学者」(という確定記述)なのではない。アリストテレスが「古代最後の偉大な哲学者」ではなく一介の商人にすぎなかった可能世界においても、「アリストテレス」という固有名は固定的にそれが指示する同一人物(われわれの知っているアリストテレス当人)を指示し続ける。

つまり、Kripke と Russell では、「アリストテレス」という固有名の scope(作用域)が異なるのである(註2)。そして Kripke の議論は、我々が独自にこれをプログラミング言語における変数の scope(使用範囲)と読み替えるのを可能にする。Kripke の挙げる例で言うと、たとえば「アリストテレスは犬が好きだった」という命題を、Kripke は「アリストテレスが「古代最後の偉大な哲学者」でなかった可能世界においても、アリストテレスは「アリストテレス」と呼ばれ、その人は犬が好きだった」というように解釈する。ところが Russell-Frege の解釈では、「古代最後の偉大な哲学者は一人であって、彼は犬が好きだった」ということになるだろう。ここでは明らかに「アリストテレス」という固有名の scope(使用範囲・作用域)が異なるのである。

ただ Kripke に対しては、同名別人 = 偶発的同一性という、固定指示論にとって躓きの石になりかねない反論の余地が残されており、Kripke 自身がそれについて、「かつては気にしていたが今は気にしていない」とかなり主観的な評を述べている点に注意したい。同名別人 = 偶発的同一性は、名前の指示を固定的なものにしない(同名性が固有名! = 固定指示子;、つまり、固有名と固定指示子は異なるとする)という、現在の Kripke にとっては明らかな謬見であるが、彼はあらためて「諸対象が「偶然的に同一」なことはありえない」と考え、同名別人 = 偶発的同一性論を捨て去ることで、固有名 = 固定指示子というテーゼに行き着いたのである。

しかし、Kripke は頑として認めないことだが、固定指示子は、論理的操作子として整序させ、重複を避けることが可能である一方、固有名は当然のことながら自然言語の中でしばしば重複する。同名別人(固有名の重複)の無視は、各個別の「固有名」の scope(使

用範囲・作用域)をいちじるしく狭く限定しないかぎりは主張することができない。そこからわれわれは、Kripkeがプログラミングにおける変数のscopeという観点から、「固有名」を適用範囲が限定された「ローカル変数」として定位していると結論づける。

そもそも名前nameとはひとつの文字列Stringである、と同時に名指されたname対象(オブジェクト)への参照であるという二重性をもつ。だから、同名別人とは、文字列Stringの内容の同一性ではなく、その文字列Stringを引数にもつ名前nameオブジェクトへの参照の同一性だ、と規定することが可能である。そしてわれわれがKripke読解を通じ、物語世界を創出するための固有名プログラミングを提案する理由は、このような名前nameの機能の二重性を活かすためである。

たとえば、先の例「アリストテレスは犬が好きだった」にしても、Kripkeはこの「単」文命題を、ローカル変数1個(“アリストテレス”)と式1個(“犬が好きだった”)から成り立つ小さなメソッドとしてとらえるのに対し、Russell-Fregelは、これを複合的な命題に分節化し、Kripkeよりscopeの大きい変数を使用して、様々な値が代入可能な指示作用の非固定性を実現していると形式化することが可能である。たとえば、それぞれのケースをオブジェクト指向の代表であるJava言語で記述してみると、以下に示すように、Kripkeの議論は、インターフェースRigidDesignatorをNameForKripkeクラスが実装する形で、Russell-Fregeの議論はNameForRussell1クラス、NameForRussell2クラスのそれぞれ二様独立した形で、JDKをインストールしたコンピュータ上で、最もシンプルな文学作品として実行される。(註3)

```
/*Kripkeにとっての固有名*/
```

```
public interface RigidDesignator{
    public static final String thatOnlyOnePerson="Aristotle";
    /* String 型のファイナルな変数(定数)がもつ固定的な値*/
    public abstract void possibleWorld1(String str);
    public abstract void possibleWorld2(String str);
    /*メソッドはシングニチャだけ宣言、実装はしない*/
}

public class NameForKripke implements RigidDesignator{
    public static void main(String args[]){
        RigidDesignator name = new NameForKripke();//インスタンス name
        name.possibleWorld1("fond of dogs");//可能世界 1
        name.possibleWorld2("the last great philosopher of antiquity");
        /*可能世界 2*/
    }
    public void possibleWorld1(String str){
        String Aristotle;//Aristotle はローカル変数に限定
        Aristotle = str;//scopeはこのメソッドの内部
        System.out.println(thatOnlyOnePerson + " was " + Aristotle + ".");
    }
    public void possibleWorld2(String str){
        String Aristotle;//ローカル変数--->同名性はOK、ぶつからない
        Aristotle = str;//scopeはこのメソッドの内部に限定
        System.out.println(thatOnlyOnePerson + " was not " + Aristotle + ".");
    }
}

}
```

```
/*Russell-Fregelにとっての固有名*/
```

```

public class NameForRussell1{
    String Aristotle = "the last great philosopher of antiquity.";
    /*インスタンス変数,Kripke の場合に比べ Aristotle は scope が広い*/
    public static void main(String args[]){
        NameForRussell1 name1= new NameForRussell1();//インスタンス name1
        NameForRussell1 name2 = new NameForRussell1();//可能世界用インスタンス name2
        String person1 = new String("Aristotle was ");
        String person2 = new String("Aristotle could not be ");//可能世界
        String person3 = new String("Someone else was ");//可能世界
        System.out.println(person1 + name1.Aristotle);
        System.out.println(person2 + name2.Aristotle);
        System.out.println(person3 + name2.Aristotle);
    }
}

```

あるいは別の記述を同時に提案する。

```

public class NameForRussell2{
    public static void main(String args[]){
        NameForRussell2 name = new NameForRussell2();//インスタンス name
        String name1 = new String("the last great philosopher of antiquity");
        String name2 = new String("the last great philosopher of antiquity");
        String Aristotle = name1;
        /*Kripke に比べ Russell らの Aristotle は scope が広い*/
        System.out.println("-----<<normal>>-----");
        if(Aristotle==name1){//代入したので参照も同一、真
            System.out.println("Aristotle" + " was fond of dogs.");
            System.out.println(Aristotle + " was fond of dogs.");
            System.out.println(" ");
        }
        System.out.println("-----<<nonrigid designator>>-----");
        if(Aristotle.equals(name2)){//参照は同一でないが内容は同一、真
            System.out.println("Aristotle" + " was fond of dogs.");
            System.out.println(Aristotle + " was fond of dogs.");
            System.out.println("Aristotle was " + Aristotle + ".");
            System.out.println("Aristotle could not be " + Aristotle + ".");
        }
        /*Russell の考え方、非固定的、別人が役割を負い得た*/
        System.out.println(" ");
    }
    System.out.println("-----<<contingent identities>>-----");
    if(Aristotle==name2){//インスタンスの内容が同一でも参照が異なる命題、実行されない
        System.out.println("Aristotle will be " + Aristotle + " forever.");//偽な
    }
    else{
        System.out.println("Aristotle is not " + Aristotle + ".");
        System.out.println("And " + Aristotle + " is not Aristotle.");//Aristotle
        /*Aristotle はインスタンス name が参照するメソッドの引数*/
        System.out.println(" ");
    }
    System.out.println("----<<other person 's fondness for dogs>>----");
    name.Otherpersons(Aristotle);
}
public void Otherpersons(String str){

```

```
String otherperson;
otherperson = str;
System.out.println(otherperson + " was fond of dogs.");
}
```

}(註4)

以上のいくつかのプログラムにおいて、KripkeとRussell-Fregeでは、nameオブジェクトの処理に明確な相違が出ることに注意したい。Kripkeの場合、nameオブジェクトは単一であり、複数化しない。複数化するのはメソッドで表された可能世界possibleWorld?()と、その実装部中のローカル変数Aristotleの方である。このAristotleという固有名は、文字列と同時に変数として使用されており、Kripkeにあっては、

- 1)String型のローカル変数で、scope **は狭く**、同一メソッド内である、
- 2)String型のファイナルな変数(定数)がもつ**固定的な値**である、

という二重性をもつ。この二重性を区別することを、Kripkeは「固定性を作用域に還元する見解は端的に間違っている(p.13)」(“the view that reduces rigidity to scope is simply in error.”, p.12)という単純な言葉で主張したのではないだろうか。言い換えれば、ソースコードのレベルで、RigidDesignatorのインターフェースとしての存在とNameForKripkeのクラスとしての存在を、独立したファイルに区別することである。固有名の固定指示性、および固有名とその存立する場をめぐる言語的な形式性・抽象性については、RigidDesignatorインターフェースがそれを集中的に名目管理し、反対にマイナーで「ローカル」な可能世界(の実体)については、NameForKripkeクラス内で細分化され、囲い込まれ、具体的輪郭を与えられている。

一方、Russell-Fregeの場合は、いくつかの形式化が可能であるにしても、Kripkeの場合と明らかに異なるのは、nameオブジェクトが複数化し、Aristotleも様々な変数の形を取って、ローカル変数の狭いscopeに限定されない、ということである。ただ、注意したいのは、Russell-Frege型のプログラムは、Kripkeが回避したはずの同名別人=偶発的同一性の問題が顕在化してしまう。

```
String name1 = new String("the last great philosopher of antiquity");
String name2 = new String("the last great philosopher of antiquity");
```

name1とname2では、オブジェクトの内容は同一でも、オブジェクトへの参照(アドレス)が異なる。オブジェクトの内容("the last great philosopher of antiquity")が同一なのでname1.equals(name2)は真だが、new演算子を2回使ってオブジェクト生成しているので、オブジェクトは同一でなく、(参照は一致せず)name1==name2は偽である。

だがこの問題は、Kripkeの可能世界論では見事に回避されている「分身」のテーマを介在させはしないか。複数の可能世界にわたって遍在する「同一人物」は分身どうし、ということになりはしないか。偶発的同一性には、言葉のレベルでの偶発的同一性、つまり同姓同名(同音異義)の他に、現実のレベルでの偶発的同一性、つまり分身(ドッベルゲンガー、カウンターパートなど他にいろいろな言い方が可能である)の二種類が考えられる。我々がここで、同一性の観念を分類するため、一般言語学の祖、Ferdinand de Saussureの言う、形相的同一性 言葉上での同一性と、実質的同一性 現実個体の同一性の本質的な区別を取り上げるのもそのためである。

2. Saussureと同一性

さて、前節で述べたとおり、Saul A・Kripkeの可能世界論において、Kripkeの言う「固有名(proper name)」は固定指示子(rigid designator)であり、固有名は反事実的

状況（可能世界）においても、固定的に同一の人物を指示していた。しかしこの議論が、同名別人（同音異義）という偶発的同一性を非本質的と見ることによって成立していたことは否定できない。しかも Kripke が、同名別人（同音異義）という、あくまで「言語における」という、限定の付いた偶発的同一性にだけ留意した点は、検討の余地がある。同じ偶発的同一性でも、言葉のレベルと現実のレベルで明確に区別する、言うなれば David Lewis(1983,86)的な可能世界も考慮する必要があるからである。

われわれがあらためて、現代言語学の祖、Saussure における同一性の観念を振り返るのもそのためである。Saussure は形相的同一性、実質的同一性という二つの同一性を提示したが、これらは、双方とも Saussure の言う「言語記号の恣意性」を考慮した上で、Kripke の言う「偶発的同一性」の二つのサブカテゴリーとして解釈される。

そもそも Saussure にとり、形相とは言語そのもののあり方であって、純粹価値体系、体系内関係のことを示す。すなわち言葉においては、各構成要素は独立の実体ではなく、他の要素との相対的關係によって初めて定義される。これを消極的示差性に基づく価値と言う。そして、形相的同一性とは、言葉の上での同一性であり、それを媒介として、現実には、適合する複数の具体例(インスタンス)が考えられるものである。『一般言語学講義』の挙げる有名な例、「ジュネーヴ=パリ、午後8時45分発、急行二本」は、それらを構成する実体、たとえば、車両本体、運転手、乗客のメンバー、実際の厳密な発車時刻がその日によって異なると、形相的に同一である。

それに対し、実質は、物理的な顕在現象、触知可能な事象として、音・映像の感覚的素材に固有の、即自的具体性をもつ。だから、実質的同一性とは、現実には指示される個体の、構成要素上の、とりかえのきかない同一性である。だが Saussure 学において、実質的同一性の例として挙げられているのは、しばしばかなり極端なものであって、一卵性双生児の存在（外面での区別不可能性）とか、自分が盗まれた服をある古着屋でふと発見したという事例（本来、保証のかぎりでない同一性）とか、いわば可能世界論的に捉えられた「分身」（極限まで類似しているが、どこかしら違うという(註5)、フロイトのいう無気味なもの(das Unheimliche)、ラカンの言う不安(l'angoisse)が漂う)と存在と件が類似している場合が多い。(註6)

ここに、Saussure の言う二つの同一性が孕む、根本的な問題が露呈する。両者は、想像された不可能な状況を、否定的媒介として絡ませないかぎり、哲学的には措定できないのである。いわば、形相は、対象を不在化させ、実質は、同一性そのものを不在化させてしまうのだ。形相は媒介であり、身体・実体ではない。形相は、究極的には、関係性そのものではあっても、具体的な個体としてのインスタンスではない。だがまさしくそのことによって、 $インスタンス1=インスタンス2$ という、同一性表現の間にあるもの(=)が実体化されてしまうのである。

ならば、むしろ、形相的同一性を、具体例(インスタンス)の完全な不在という条件下で思考する方が適当なのではないだろうか。反対に、実質は媒介ではなく、個において孤立している。だが、実質的同一性の例に挙げられているものはことごとく、「ほかならぬこの」という限定を受けるため、対象が極端に稀少化してしまっているのだ。中にはそれが、いったん消えて再び現れる、というような、ダイナミックな時間性をもち、それが同一性の根拠に対する試練となる場合もある。実質は、究極的には、空間中の遍在のみならず時間上の経過を通じて同一でありつづけるもの(比較の前提)を認めない。

たしかに形相的同一性は、オブジェクトの情報の同一性であり、それによってシフィアの同一性が議論可能になる。また実質的同一性は、オブジェクトそのものの同一性であり、それによってシフィアの同一性が議論可能になる。しかし、Saussure の根源的な二重性は、Saussure を超えたところで批判的に解釈される必要があると言える。われわれが、それを、オブジェクト指向のもつ二重性、すなわち 1) 形相(的同一性)・設計としてのインターフェース(媒介)と、2) 実質(的同一性)・実装のためのクラスという二重性と重ね合わせて論じ

るのも、そのためである。

一般にオブジェクト指向言語の特徴として、設計と実現の分離があるが、それは、抽象的に表された型枠と具体的な事物の生成を切り離すことである。これをふまえ、Saussureにとってのオブジェクトを、形相/実質の区別を、同一性のもつ二面性として、すなわちシフィアツ/シフィエの区別として、処理可能にする操作子、すなわち言語記号と規定してみよう。その上でJava言語における同一性の処理法、すなわち、オブジェクトそのものの同一性（参照の一致）をあらわす関係演算子 $a=b$ と、オブジェクトそのものではなく内容の同一性をあらわすメソッド $a.equals(b)$ を利用し、Saussureにおける偶発的同一性を示すプログラムを、「(違った日の)ジュネーヴ=パリ、午後8時45分発、急行二本(は形相的に同一)」を例に記述してみる。それが(インターフェースをクラスが実装する)Kripkeの固有名プログラミングと、本質的に同型であることを証明してみよう。

```
public interface GeneveParisExpress {
    public static final double timeOfDeparture = 8.45; //根本的に恣意的、無契約
    public abstract void identityOfTrain(double date); //シグニチャだけ(実装するクラスが自前で定義し、契約履行)
    public abstract String realityOfTrain(double date); //シグニチャだけ(実装するクラスが自前で定義し、契約履行)
}

public class ThisTrain implements GeneveParisExpress{
    public static void main(String args[]){
        GeneveParisExpress t1, t2; //インターフェース型の変数宣言
        String st1, st2;
        t1 = new ThisTrain(); // インスタンス生成
        t2 = new ThisTrain();
        t1.identityOfTrain(4.11); //インターフェースのメソッド実現契約
        t2.identityOfTrain(4.12);
        st1 = t1.realityOfTrain(4.11); //インターフェースメソッド実現
        st2 = t2.realityOfTrain(4.11);
        if(st1==st2) // 同一オブジェクトを参照しているならば-->偽
            System.out.println(" relational and material identity ");
        if(st1.equals(st2)) //オブジェクトの内容が同一ならば-->真
            System.out.println(" relational but not material identity "); //同一日4月
            11日の列車ですら、物質的には同一と見なさない
    }

    public void identityOfTrain(double date){
        System.out.println(" The same train at " + timeOfDeparture + " h on " + date);
    } //インターフェースからのファイナル変数により形相的に同一

    public String realityOfTrain(double date){
        String st; // ローカル変数宣言
        st = String.valueOf(date); //dateの値を文字列として読む
        return st; // 文字列を返す
    }
} (註7)
```

このプログラムの例は、インスタンスの生成なくして形式的同一性を考えるべきだというSaussure批判が、裏を返せば実質的同一性そのものの不在化につながるということを示している。すなわち、きわめて逆説的ではあるが、同一日、4月11日の同一列車そのものですら、実質的にはけっして同一と見なされない、存在が自己同一性を持つことさえできないのである。

詳しく見てゆくと、インターフェースであるGeneveParisExpressはThisTrainクラスで実装されているが、インターフェース内ではインスタンスは生成できず（このことは先述のごとく、形式的同一性の極限条件と解釈できる）、形式的同一性、実質的同一性にそれぞれ対応するメソッドidentityOfTrain(double date)、realityOfTrain(double date)も、宣言だけで定義されていない。オブジェクト指向における「契約によるプログラミング」では、これらのメソッドがインターフェースを実装したクラスで必ず定義されることを要求するが、このことは奇しくもSaussure自身が「契約」と命名している観念と根源的レベルで発想が一致している。

つまり、インターフェースに定義されたファイナル変数やシグニチャを宣言されたメソッドは、選択のレベルで Saussure の言う「(世界の)言語的切り分け (d coupage linguistique)」、ないし「根本的恣意性」を表現している。だが急行の出発時刻を格納する変数 timeOfDeparture が、インターフェース型だと宣言されている以上、それによって、「インターフェースを実装するクラスで、インターフェースのメソッドが実現される」という契約が生じ、クラスのプログラマーはそれを必ず履行しなくてはならない。このいわゆる「契約に基く設計 design by contract」こそが、まさしく Saussure の言う「契約(contrat)」のことである。つまり、特に Gadet(1987)らの解釈を拡大させて言うと、インターフェースのもつ恣意性が制限ないし相対化されることを、Saussure は「契約」と呼んでいるのである。

それはさておき、Saussure における GeneveParisExpress インターフェースと（それを実装する）ThisTrain クラスの関係が、Kripke における RigidDesignator インターフェースと（それを実装する）NameForKripke クラスの関係と、「契約」の意味で同型なのを確認しておこう。すなわち、固定指示子はインスタンス化してとらえるべきではなく、同名別人そして/もしくは偶発的かつ形相的同一性は、その名前を持つ具体例(インスタンス)を実質的に生成せずに思考できるからこそ問題にならないのである。そして実際、「name オブジェクトへの<参照>の同一性」という観点から、オブジェクト指向的に言えば、Kripke もまた暗黙のうちに「設計」(「宣言」)のレベルを「実装」(「具体化」)のレベルと分離して考える。その意味でならば「固有名」は、インスタンス(オブジェクト)を生成しない(ここでは Java 言語などで言う意味での)インターフェース内のファイナル変数(定数)として、やはり「固定指示子」の役割を果たしていると言える。

それに対し、Kripke の批判する Russell-Frege の固有名論は、固有名をさまざまなメソッド(関数)の String 型のパラメータとして扱う場合、「内容は同一でも、オブジェクトへの参照(アドレス)が異なる」といういわゆる分身のテーマ、すなわち現実における偶発的同一性までその射程に入れることができる。すなわち、Saussure にとってのもうひとつの同一性、すなわち実質的同一性が関わってくるのである。ここでは比較操作そのものの不可能性(いや、存在が自己同一性を持つこと自体の不可能性)や、いったん消えて再び現れるというような、時間性 同一性の根拠に対する試練、パラドクスなどが問題になるだろう。

いずれにせよ可能世界論は、Saussureの記号学と同様、物語空間論として、コンピュータサイエンスの側からオブジェクト指向論を通じ、発展的な解釈が可能である。Kripkeにおいては、偶発的同一性（つまり名前の指示を固定的なものにしない、同名別人という反例）を捨て去ることで、固有名イコール固定指示子というテーゼに行き着いた。このテーマを、「nameオブジェクトへの<参照>の同一性」という観点から取り上げると、オブジェクト指向の流れの中で、物語世界を創出するための（固有名）プログラミングが見えてくる。登場人物 固有名の論理を形式化することは、計算文学論、とくに文学の実装に関する議論において重要なテーマのひとつである。

けっきょく、オブジェクト指向的な物語論は、コンピュータに小説を書かせるための予備的な、あるいは<最初の一步>的な手続きにすぎないのかもしれない。われわれはその目的にあわせ、ここでは、文学に内在する問題--とくに解釈・鑑賞・批評等の伝統的な問

題--も、具体的な作品からの特異な事例も、特に取り上げなかった。コンピュータのもつ能力、可能性を發揮させずしてそれらを単独で提示するのは、けっして生産的なことではないように--むしろ不毛なことのよう--思えたからである。物語の世界はどこにあるのか？--それはいつも「制度としての物語」の内部にあるわけではない(コンピュータの中にさえ物語は存在するのだ)。そして、そのことをあえて額面通り受け取って考えることから、文学論のすべてが始まると言って過言でないのである。

註

1)ただし、Kripke自身の可能世界定位とは、若干の落差むしろ、多少脈絡の欠落はあるが、Kripkeの発言に沿って、<確率・統計的>に把握することも可能ではないか？その意味で、最近のLatent Semantic Analysis (LSA:潜在的意味分析)はかなりKripkeの脈絡に適合する。

彼にとって、可能世界は良く言われるような並行宇宙ではなく、ある意味で確率、統計的な存在である。可能世界は、言語のうちに約条されているものであって、「学校の確率論のミニ世界を大きく膨らませた以上のものではない」(Kripke, p.20)「事実、近代確率論の基礎をなす「標本空間」という一般的概念は、まさにそのような可能世界という空間についての概念である。」(Kripke, p.21)と、彼は述べている。

では、LSA から見た可能世界とは何だろうか。LSA (あるいはSVD:特異値分解)によって、データマトリックス内の不在項や量的稀少項は 可能な存在・存在量が想定権利をもつかぎりにおいて 特異空間内で数値を増大させる(cf.p189-192)。LSA (SVD)によって、各項目(たとえば登場人物)は、それが帰属する集合・世界と連帯して(本質的区別はないものとして)LSA空間内に位置付けられる。その意味で、Kripkeの可能世界論は、数理解析学の側からも、LSAを通じて、発展的な解釈が可能である。

Kripkeの可能世界論は物語空間論として、オブジェクト指向の側からもLSAの側からも捉え返すことが可能だろう。だがこのような発展的解釈の視座設定をめぐって、まだいろいろと検討すべき課題が多いことは確かである。

2)ラッセルの場合のように、作用域(scope)が広くなると、固有名は非-固定指示化するわけであり、クリプキはその傾向を間違いでであると断じている。「アリストテレスは哲学者でなかったかもしれない(It might have been the case that Aristotle was not a philosopher.)は真理を表現しているが、ラッセルの理論に反して「古代最大の哲学者は哲学者でなかったかもしれない(It might have been the case that the greatest philosopher of antiquity was not a philosopher.)」はそうではない、と私は考えた...ところで、最後に引用された文は、そこで使われている記述を、私の意図に反して広い作用域をもつものと解釈すれば、真理を表すことになる。(p.14))

3)同様に、Kripkeがあげる一角獣の物語(Kripke, p.26-27)も以下のような形式でプログラミング可能である。「よく引き合いに出されるのは一角獣の例である。すなわち、一角獣が存在しないことは誰もが認めているが、もちろん一角獣は存在したかもしれない、と言われる。ある状況のもとでは、一角獣は存在したであろう、と言うわけである。これは私がそうではないと考える事柄の一例である。私の考えでは、正しくは、一角獣が存在しえないことは必然であるという言い方をするのではなく、いかなる状況のもとでなら一角獣が存在したことになるのかわれわれにはわからない、とだけ言うべきなのである。その上、われわれが一角獣神話によって知っている一角獣に関する事柄をすべて満たす動物が過去に存在したことを決定的に証明する化石を、たとえば考古学者や地質学者が明日発見したとしても、それは一角獣がいたことを示す証拠にはならないだろう、と私は思う。」

```
public interface DesignatorUnicorn{//インターフェース内ではインスタンスは生成しない
    public static final String unicorn = "UNICORN";//固定指示子
    public abstract void KripkeMethod1(String str);
    public abstract void KripkeMethod2(String str);
    public abstract void KripkeMethod3(String str);
}
public class NameUnicorn implements DesignatorUnicorn{
    public static void main(String args[]){
        DesignatorUnicorn name = new NameUnicorn();//インスタンス name
        String unicorn_discovered = new String("UNICORN");//固定指示子と同じ内容のインスタンス生成
        System.out.println("-----Method1:designator:non-existence of instance-----");
        name.KripkeMethod1(unicorn_discovered);
    }
}
```

```

System.out.println("-----Method2:proper name-----");
    name.KripkeMethod2(unicorn_discovered);//固有名に固定指示子が介入しない
    name.KripkeMethod2(unicorn);//固有名に固定指示子を代入
System.out.println("-----Method3:Does proper name equal designator?-----");
    name.KripkeMethod3("");//固有名が存在しなくても固定指示子が代用
    name.KripkeMethod3(unicorn);//固有名に固定指示子を代入
    name.KripkeMethod3(unicorn_discovered);//固有名に固定指示子が介入しない
}

public void KripkeMethod1(String str){
    if(str==unicorn)
        System.out.println("There were unicorns!");
    if(str.equals(unicorn))
        System.out.println("I think that even if archeologists or geologists were¥n"
+
"to discover tomorrow some fossils conclusively showing the existence of animals¥n" +
"in the past satisfying everything we know about unicorns from the myth of unicorn, ¥n" +
"that would not show that there were unicorns.");
}
/*「われわれが一角獣神話によって知っている一角獣に関する事柄をすべて満たす動物が過去に存在した
ことを決定的に証明する化石を、たとえ考古学者や地質学者が明日発見したとしても、それは一角獣がい
たことを示す証拠にはならないだろう。」*/

public void KripkeMethod2(String str){
    String unicorn;//unicornはローカル変数(固有名)でメソッド内しかscopeに持たない
    unicorn = str;//ローカル変数はメソッド内部では強力
    System.out.println("For " + unicorn + " we will use the term 'name' so that it does not
include definite descriptions of that sort.");
}

public void KripkeMethod3(String str){
    if(str.equals("")){
        System.out.println("For " + unicorn + " we will use the term 'name' so that it does not
include definite descriptions of that sort.");
    }
/*ここでは、ローカル変数=固有名(String unicorn;)の宣言はない。空文字列の時は、「確定記述を含まな
い場合である」ということにして、<固定指示子>であるファイナル変数が、存在できない<固有名>のカヴ
ァーに来るようにする。つまり<固定指示子>は<固有名>のふりをする事ができる。*/
    else{
        String unicorn;
        unicorn = str;
        System.out.println("For " + unicorn + " we will use the term 'name' so that it does not
include definite descriptions of that sort.");
/*空文字列でない時は、確定記述はローカル変数としての<固有名>に代入され、<固定指示子>は関与しな
い。しかし、出力される結果は、空文字列であってもなくても表面的には同じになる。つまり<固有名>は<
固定指示子>のふりをする事ができる。*/
    }
}
}
}

```

4) NameForKripke.class の実行結果は以下の通り。

```

Aristotle was fond of dogs.
Aristotle was not the last great philosopher of antiquity.

```

NameForRussell1.class の実行結果は以下の通り。

Aristotle was the last great philosopher of antiquity.
Aristotle could not be the last great philosopher of antiquity.
Someone else was the last great philosopher of antiquity.

NameForRussell2.class の実行結果は以下の通り。

-----<<normal>>-----

Aristotle was fond of dogs.
the last great philosopher of antiquity was fond of dogs.

-----<<nonrigid designator>>-----

Aristotle was fond of dogs.
the last great philosopher of antiquity was fond of dogs.
Aristotle was the last great philosopher of antiquity.
Aristotle could not be the last great philosopher of antiquity.

-----<<contingent identities>>-----

Aristotle is not the last great philosopher of antiquity.
And the last great philosopher of antiquity is not Aristotle.

--<<other person's fondness for dogs>>--

the last great philosopher of antiquity was fond of dogs.

5)たとえば、丸山(1981)より。「A氏が、昨日東京駅午前八時発特急ひかり号を利用したという話を聞いたB氏が、「私も先月、同じ列車を利用した」と答え、C氏が「私も昨日、あなたと同じ列車に乗っていた」と言ったとする。B氏とC氏は<<同じ>>列車と言ったが、その内容がすこぶる異なるものであることは文脈から明らかであろう。...一度こわされて復旧した街路は、以前の道が砂利道であり、それが今度はコンクリート舗装されたとしても同じ道と見るのがB氏の視点に立つ同一性だとすれば、顔立ちや背たけがよく似た双子の兄弟も決して同じ人間ではないとか、火事で失われてしまった同じ家具は二度と入手できないと言うときの同一性は、C氏の視点に立つ同一性である。B氏の視点は<<形相>>の視点、C氏の視点は<<実質>>の視点なのだ。(p.132-133)」

6)フロイトは、『無気味なもの』(1919)という論文の中で、ホフマンの小説、『砂男』を例に二重自我、ドッベルゲンガーの問題を取り扱っている。

7) ThisTrain.class の実行結果は以下の通り。

The same train at 8.45h on 4.11
The same train at 8.45h on 4.12
relational but not material identity

文献

(邦訳があるものの引用は、原則として翻訳に拠った)

Fran oise Gadet(1987), Saussure, Une science de la langue, PUF.(フランソワーズ・ガデ (1995)、『ソシュール言語学入門』(立川健二訳)、新曜社)

Saul A.Kripke (1980), Naming and necessity, Basil Blackwell and Harvard University Press.

(ソール A・クリプキ、『名指しと必然性 様相の形而上学と心身問題』(矢木沢 敬、野家啓一訳)、産業図書、1985)

David Lewis (1983,86), Philosophical papers, 2 volumes, Oxford University Press.

Bertrand Meyer (1988), Object-Oriented Software construction, Interactive Software Engineering.

(バートランド・メイヤー(1990)、『オブジェクト指向入門』(二木厚吉監訳/酒匂寛・酒匂順子訳)、アスキー出版局)

Ferdinand de Saussure (1916), Cours de linguistique g n rale Payot.

(F・D・ソシュール(1983)、『一般言語学講義』(小林英夫訳)、岩波書店)

Ferdinand de Saussure (1967-74), Cours de linguistique g n rale Edition critique e par Rudolf Engler.

Hinrich Sch tze (1997), Ambiguity Resolution in Language Learning, Computational and Cognitive Models, CSLI Publications.

小方孝、『物語生成システムの観点からの物語言説論の体系化へ向けた試み』、情報処理学会研究報告、99-CH-44、p.31-38.

立川健二ノ山田広昭 (1990)、『現代言語論ーソシュール フロイト ウィトゲンシュタイン』、新曜社.

ジークムント・フロイト(1978)、フロイト著作集 3、人文書院.

丸山圭三郎 (1981)、『ソシュールの思想』、岩波書店.