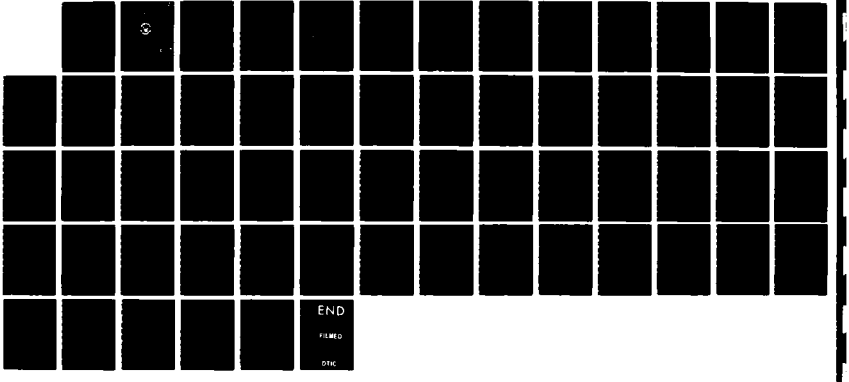
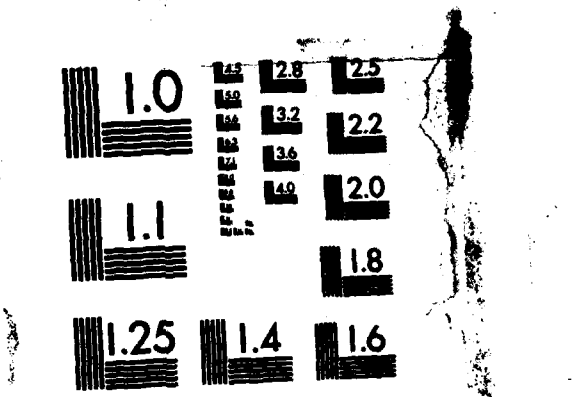
AD-A159 673 A SURVEY OF PROPERTIES OF RELATIONS WHICH HAVE THE CONFLUENCE PROPERTY(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA U OZKAN JUN 85 1/1

UNCLASSIFIED F/G 12/1 NL





NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

A SURVEY OF PROPERTIES OF RELATIONS WHICH HAVE THE CONFLUENCE PROPERTY

by

OZKAN, Ugur
June 1985

Thesis Advisor: Daniel L. Davis

Approved for public release; distribution is unlimited

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A159 673	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A Survey of Properties of Relations Which Have the Confluence Property		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis June 1985
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Ozkan, Ugur		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93943		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93943		12. REPORT DATE June 1985
		13. NUMBER OF PAGES 58
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution is unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Confluence Property, Church-Rosser, Completion-Procedure

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The confluence property arises in a number of areas of Computer Science; from its origins in the lambda-calculus to its use in the theory of abstract data types and term rewriting systems. Its abstract properties and its application to a number of problems, such as algebraic specifications of abstract data types and term rewriting systems, are surveyed here.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

S/N 0102-LF-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Approved for public release, distribution unlimited

A Survey of Properties of Relations Which Have the Confluence Property

by

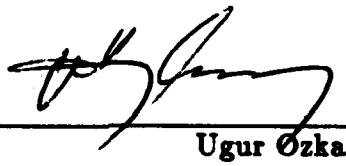
Ugur Ozkan
Lieutenant, Turkish Air Force
B.S., Air War Academy, 1979

Submitted in partial fulfillment of the
requirements for the degree of

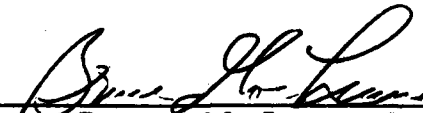
MASTER OF SCIENCE IN COMPUTER SCIENCE

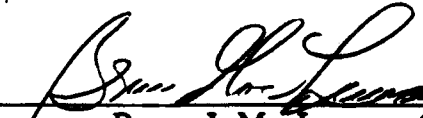
from the

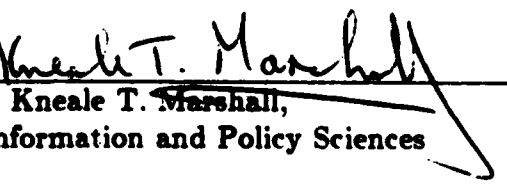
NAVAL POSTGRADUATE SCHOOL
June 1985

Author: 
Ugur Ozkan

Approved by: 
Daniel L. Davis, Thesis Advisor


Bruce J. MacLennan, Second Reader


Bruce J. MacLennan, Chairman,
Department of Computer Science


Kneale T. Marshall,
Dean of Information and Policy Sciences

ABSTRACT

The confluence property arises in a number of areas of computer science; from its origins in the lambda-calculus to its use in the theory of abstract data types and term rewriting systems. Its abstract properties and its application to a number of problems, such as algebraic specifications of abstract data types and term rewriting systems, are surveyed here.

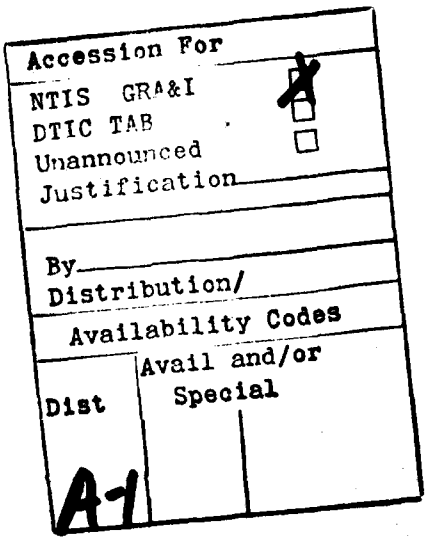


TABLE OF CONTENTS

I. INTRODUCTION

A data type is a class of objects together with a set of operations which may be performed on these objects. An *abstract* data type is a precise description of a class of objects in terms of the semantics of the operations which may be performed on the class (Yurchak [1984]).

Given an abstract data type and two formal terms defined by the operations of the type , we consider whether these two terms are equivalent. In particular, we consider the question of the *decidability* of equality within an abstract data type. This problem in many cases reduces to the question of whether or not the axiom set for the data type is confluent. This thesis is concerned with this second question.

In the first chapter, we survey the historical work that brought the Church-Rosser property into the literature. The second chapter introduces the idea of confluence and its relation to the Church-Rosser property. The rest of this chapter is on theorems related to confluence. In the third chapter, we discuss the algebraic specification of abstract data types, which provides the background to move into the study of term rewriting systems, which is the second part of the third chapter. In the last chapter, we discuss an algorithm for showing that a given axiom set (as rewrite rules) is confluent. This procedure is called the Knuth-Bendix completion algorithm.

II. ORIGIN OF CONFLUENCE

A. IDEAS FROM COMBINATORY LOGIC

1. Introduction

In this chapter, we introduce concepts related to the Church Rosser theorem as they are discussed in Curry and Feys [1958]. Combinatory logic is a branch of mathematical logic whose purpose is the analysis of certain notions of such basic character that they are ordinarily taken for granted. These include the processes of substitution, usually indicated by the use of variables, and also the classification of the entities constructed by these processes into types or categories, which in many systems has to be done intuitively before the theory can be applied. So far it has been observed that these notions, although generally presupposed, are not simple; they constitute a prelogic whose analysis is by no means trivial.

Two questions have initiated this analysis. The first of these is the problem of formulating the foundations of logic as precisely as possible. The second question is the explanation of paradoxes.

In order to get a better idea of the motivation and purpose of combinatory logic, it will be well to elaborate these points a little before we go further.

2. Why a New Functional Notation

There is a lack of a systematic notation for functions in ordinary mathematics. The known notation $f(x)$ does not distinguish between the function itself and the value of this function for an undetermined value of the argument (in fact the same problem occurs in Pascal while passing a function as a parameter to another). This defect is especially striking in theories which employ functional operations, such that functions which admit other functions as

arguments. For special operations such as differentiation and integration there are special notations having unique meanings, but these are not to be generalized.

As an example, assume P is a predicate in a given system. If $f(x)$ is argument of P, which is expressed as $P[f(x)]$ then what is $P[f(x+1)]$? Must $g(x) = f(x+1)$ be formulated first, and then passed to P as $P[g(x)]$, or is $h(x) = P[f(x)]$ formulated first, then $h(x+1)$? It would appear that the results of these two different implementations are the same. For some important operators it seems the same, but is not. For example, let

$$P[f(x)] = \begin{cases} (f(x)-f(0))/x & \text{if } x \neq 0 \\ f'(0) & \textit{otherwise} \end{cases}$$

for $f(x) = x^2$

$$P[g(x)] = P[x^2+2x+1] = x+2,$$

$$h(x) = P[f(x)] = x,$$

$$h(x+1) = x+1 \neq P[g(x)].$$

For the second point, let us look at the so called Russell Paradox. This may be formulated as follows: Let $F(f)$ be the property of properties f defined by the equation

$$F(f) = \textit{not } f(f) \qquad (1)$$

where *not* is the symbol for negation. Then, on substituting F for f, we have

$$F(F) = \textit{not } F(F). \qquad (2)$$

If, we say that $F(F)$ is a proposition, where a proposition is something which is either true or false, then we have a contradiction. But it is an essential step in this argument that $F(F)$ should be a proposition. This is a question of the prelogic; in most systems it has to be decided by an extraneous argument.

Another well-known paradox is:

"I am lying."

We may explain the Russell paradox by claiming the meaning of F or $F(F)$ is "meaningless". Thus, as it is discussed in Curry and Feys [1958], "in

Principia Mathematica (written by Russell and Whitehead) the formation of $f(f)$ is excluded by the theory of types (developed by Russell and Whitehead); in some mathematicians' explanations one can not use (1) as a definition of F because the existence of F cannot be eliminated". Certainly by way of such restrictions we can eliminate paradoxes from a given system. But, as we will discuss in following paragraphs, there is something about the preceding argument which is not explained by such exclusions.

As stated in Curry and Feys [1958], the following requirements are necessary to reach the objectives we have already discussed:

(a) There will be no distinction between different categories of entities, so any construct formed from the primitive entities by means of the allowed operations must be meaningful such that it is acceptable as an entity;

(b) There will be an operation corresponding to the application of a function to an argument;

(c) There will be an equality with the usual properties;

(d) The system must be *combinatorially complete,* such that any function we can define intuitively by means of a variable can be represented formally as an entity of the system.

By means of these four requirements, F defined by (1) is certainly significant, and also the equation (2) is intuitively true. In fact this is what we have to get, since we can not "explain" a paradox by getting rid of it. Instead, as Curry and feys stated "stand and look it in the eye" then we will force them into the open, where we can analyse them. To me, our expectations from this analysis must be to find a way to show that functions like $F(F)$ in (2) are not in the category of propositions. This will be the main objective in the field of combinatory logic as explained in the following paragraphs. Our purpose for the analysis is twofold. As stated in Curry and Feys [1958], the first step is the analysis of the substitution processes, without considering the classification of entities into categories. The second part is the introduction of the machinery for effecting a classification into categories.

In our analysis, a basic role is played by certain operators which represent combinations as functions of the variables they contain. The definition of a combinator is as follows (from Curry and Feys [1958]) " the combinations in question are those formed from the variables alone by means of the operation postulated in the second of the above demands. By the requirement of combinatorial completeness, these operators are represented by certain entities of the system. These entities, and combinations formed from them by the postulated operation, are called *combinators.*

The term 'combinatory logic' [1] is intended to describe a part of mathematical logic which requires reference to combinators, including all that is necessary for an adequate foundation of the more usual logical theories.

The combinators themselves may be defined in terms of an operation of abstraction, or certain of them may be thought of as primitive ideas and the others defined in terms of them. If we consider an operation of abstraction, this leads us to the *calculus of lambda-conversion* of A. Church, and various modifications of it; the second idea leads us to the *(synthetic) theory of combinators.* It is the synthetic theory which gives the ultimate analysis of substitution in terms of a system of extreme simplicity. Before introducing Church's calculus of λ-conversion, we will discuss the notion of a formal system first.

B. FORMAL SYSTEMS

1. Axiomatic Systems

To get a first idea of a formal system we start with elementary geometry as taught in secondary schools (the example is taken from Curry and Feys [1958]).

Elementary geometry begins with certain primitive statements, called *axioms,* which are accepted without proof. From these axioms all other accepted statements are deduced according to logical rules assumed without discussion.

---

[1] The choice of the term *combinatory* instead of *combinatorial* is therefore in agreement with Oxford English Dictionary.

The *theorems* are the axioms and the statements deduced from them. As we will realise, the system is in fact based on axioms we have chosen, so the system is called an axiomatic system.

For a given theory, the statements have to deal with some certain concepts, and some of them may be left undefined, since they are assumed to be intuitively clear. If statements or axioms are left undefined, this is because they are assumed to be intuitively understandable (we do not have to show a long proof for doing a+b+c = c+a+b, by way of commutativity, since it is intuitively evident). The theorems including these undemonstrated axioms or statements inherit their intuitive meaning.

As is well known, such concrete deductive theories have been superseded by 'pure' deductive theories. Here undefined terms are never tied to an interpretation. Undemonstrated statements claim no evidence, as they do not even have presupposed intuitive meanings; they are assumed quite arbitrarily, and the theorems derived from them take part of their arbitrary character. A theory of this character we shall call an *abstract (or pure) axiomatic system.*

2. Transition to Formal System

Even in such a pure axiomatic theory there is always a naive element, since the theory is formalized in terms of logical concepts supposed to be intuitively clear, and the deductions are made because of logical rules whose validity is supposed to be intuitively evident. If we remove this last naive element we arrive at what we call a *formal system.*

A formal system is essentially a set of theorems generated by precise rules and concerning unspecified objects. The determination of the validity of a statement in such a system does not require any experience, nor does it require any a previously known principles, not even those of logic. We should simply be able to understand the symbols employed in a precise way, as we use them in mathematics.

The statements which the formal system formulates we will call its *elementary statements,* those which it asserts its *elementary theorems.* The elementary statements are about unspecified objects which we call the *obs* of the formal system (Curry and Feys [1958]).

3. Example of a Formal System

Let us consider a very simple example of a theory, which we will call the *elementary theory of numerals.*[2] The obs of this elementary theory will be $0, 0', 0'', \ldots$ etc. Elementary statements will be equations between the obs, e.g. $0 = 0, 0' = 0'$. We take as axiom $0 = 0$, and as a rule of derivation "If two obs are equal, their successors are equal". We can then derive elementary theorems such as $0' = 0', 0'' = 0''$.

Let us now state this theory more formally. We have to consider:

a. Obs (objects).

(1) One primitive ob : 0.

(2) One unary operation, indicated by priming.

(3) One formation rule of obs: If x is an ob, then x' is an ob.

b. Elementary statements.

(1) One binary predicate: $=$.

(2) One formation rule of elementary statements:

If x and y are obs, then $x = y$ is an elementary statement.

c. Elementary theorems.

(1) One axiom: $0 = 0$.

(2) One rule of deduction: If $x = y$ then $x' = y'$.

These conventions constitute the definition of the theory as a formal system in the above sense.

The elementary theorems of this system are precisely those in the list:

$0 = 0,$

$0' = 0',$

$0'' = 0'',$

---

[2] Curry and Feys [1958], pp 13-14.

. . .

These are true statements about the system. But once the system has been defined, we can make other statements about it, e.g. the statement

If y is an ob, then $y = y$

is a true statement about the system, although not an elementary theorem. That is an example of what we will call an *epitheorem.*

4. Definition of a Formal System

We define a formal system by a set of conventions which we call its *primitive frame.* This frame has three parts:

(a) a set of objects which we call *obs,*

(b) a set of statements, which are called *elementary statements* concerning these obs,

(c) the set of those elementary statements which are true, constituting the *elementary theorems.*

In the first part, the primitive frame enumerates certain *primitive* obs or *atoms,* and certain primitive *operations,* each of which is a mode of combining a finite sequence of obs to form a new ob. It also defines rules by the criteria that "further obs are to be constructed from the atoms by the operations". Then we come to the point that the obs of the system are precisely those formed from the atoms by the operations according to the rules; furthermore obs constructed by different processes are distinct as obs.

In the second part, the primitive frame enumerates certain *(primitive) predicates* each of which is a way of forming a statement from a finite sequence of obs. It also defines the rules according to which elementary statements are formed from the obs by these predicates. Then we will consider that the elementary statements are precisely those so formed.

Since the first two parts of the primitive frame have features in common, it is rather logical to consider them together, and to extend terminology which can be applied to either. Thus the considerations based on the two parts together constitute the *morphology* of the system; the rules of the morphology constitute

the *formation rules;* and the atoms, operations predicates, taken collectively, constitute the *primitive ideas.* The morphological part of the primitive frame then enumerates the primitive ideas and enunciates the formation rules. To consider simultaneously the properties of the operations and predicates we group them together as *functives.* Thus each functive has a certain finite number of arguments; this number will be called its *degree.* As usual, functives of degree one will be called *unary,* those of degree two *binary,* and so on. Given an n-ary functive, the ob or statement formed from n obs by that functive will be called a *closure.* Occasionally it is acceptable to think of the functives, as predicates of degree 0, certain unanalyzed *primitive statements.* (The terminology is from Curry and Feys [1958])

The third part of the primitive frame states the *axioms* and *deductive rules* of the system. Axioms are elementary statements stated to be true unconditionally. There may be a finite list of these or they may be given by rules determining an infinite number in an effective manner (e.g. by axiom schemes). The deductive rules specify how theorems may be derived from the axioms. The *elementary theorems* are the axioms together with the elementary statements derived from them according to the deductive rules. In contradistinction to the morphology, considerations depending essentially to the third part of the primitive frame will be called *theoretical;* taken collectively, they constitute the *theory proper.*

There is a large intersection between the notion of a formal system and an abstract algebra in ordinary mathematics. Therefore we had better emphasize certain differences. In an algebra, we start with a set of elements and a set of operations. The elements and the operations that establish correspondences among them are explained as existing in advance. The sequences generated by them are called *terms.* Given a term of n elements, an operation of degree n "assigns" to this term one of the elements as a "value". The case $n = 0$ is accepted as a "fixed element" or "constant" (for example 0 in the naturals). These fixed elements are not analogous to the atoms; because it is not the rule, but the exception, that all the elements are obtained from the fixed elements by the operations. Moreover, equality is taken for granted and it often happens that

the same element may be obtained by the operations in many different ways. In this sense the notion of a formal system is totally different. What is given is not a set of elements but the atoms and the operations, and the obs are generated from them. As we stated earlier, obs may be obtainable from the atoms by the operations, but different processes used in construction of obs result in different obs. So an ob can be considered as a process of generation.

5. Variables

The construction of a formal language has to be explained in a communicative language understood by both the speaker and the listener. Let us call this language the *U-language*. In earlier sections, words such as 'statement', 'ob', 'operation', 'theorem' which are used in the presentation of elementary system of numerals, are words which are supposed to have meaning in the U-language before the formal system is introduced. But symbols such as '0', '′','=' are new and they are not in the U-language. Let us call the language in which these symbols are the elementary symbols the *A-language*. (see Curry and Feys [1958] for details)

The word 'variable' has two different meanings. First, a variable is a symbol or expression of the U-language called an intuitive or U-variable. For example, 'x', 'y' used in the example of a formal system are U-variables. These are certainly symbols, not obs, and a formal system is not about them. Secondly, formal systems can have the category of atoms called 'variables' in the primitive frame. These are called *formal variables*. So a formal variable is not a symbol, but an ob.

Three kinds of formal variables are (a)Indeterminates, (b)Substitutive variables, and (c)Bound variables.

- An *indeterminate* is an atom concerning which the primitive frame specifies nothing except that it is an ob.

- *Substitutive variables* are those with respect to which there is a rule of substitution. Such a rule requires that a class of obs be specified for which arbitrary obs or obs of a certain kind may be substituted under certain

circumstances. Substitutive variables are not indeterminates since they play a role with respect to the substitution rule.

In a syntactical system one explains substitution in terms of actual replacement of a symbol by an expression. In a formal system substitution is an operation on obs which has to be defined abstractly. We are not going to explain it in more detail here.

(c) A system contains *bound variables* just when there is formulated a set of substitutive variables and at least one proper operation in which these variables play a special role.So in a formalization of integral calculus

$$\int_0^2 x^2\,dx$$

the variable x is bound. As we see, bound variables are used when we have arguments which are to be interpreted as functions. Bound variables have all the complexities of substitutive variables and some others additionally.(for details see Curry and Feys [1958])

Indeterminates and substitutive variables together are called *free variables.* In other words, every variable which is not bound is free variable. Substitutive variables and indeterminates have much in common. In fact, substitutions of arbitrary obs for the free variables are possible in either case.

6. Monotone Relations

A monotone relation is a relation **R** such that

$$X\ \mathbf{R}\ Y \Longrightarrow A\ \mathbf{R}\ B$$

whenever B is the result of replacing an occurence of a component X of A by Y. A monotone relation which is irreflexive and transitive will be called *a monotone quasi-ordering*; if, in addition, it is symmetric it will be called an *equivalence.* Let R_0 be a given relation, then the monotone quasi-ordering generated by R_0 is the relation R defined by the properties (ρ), (τ), together with (ϵ) $X\ \mathbf{R}_0\ Y \rightarrow X\ \mathbf{R}\ Y$. The monotone equivalence generated by R_0 is that defined by these postulates together with (δ) (in the next section we define this properties).

C. CALCULUS OF λ-CONVERSION

1. λ-notation

Here, we describe the λ-notation originated from the calculus of λ-conversion by A. Church and J.B. Rosser (see Church [1941]). To do that we first must remember that a function is a law of correspondence, i.e., a class of ordered couples, and that to indicate the function we must indicate both elements of each couple. If we abbreviate an expression by M containing x which indicates the value of a function when the argument has the value x, we write $\lambda x(M)$ or $\lambda x.M$ to designate the function itself. Thus $\lambda x(x^2)$ means the function having x^2 for value if x is the value of the argument. Suppose we use D for differentiation and J for integration, then the statements

a) $(x+1)^2 = x^2+2x+1$,

b) x^2 *is a function of* x,

c) $\frac{d}{dx}x^2 = 2x$,

d) $\int_0^3 x^2\,dx = 9$

will become

a) $\lambda x\ (x+1)^2 = \lambda x\ (x^2+2x+1)$

b) $\lambda x.x^2$ *is a function*

c) $D(\lambda x.x^2) = \lambda x.\ 2x$

d) $J(0,3,\lambda x.x^2) = 9$

As for the example in connection with (1), if we let E be an operator such that

$$E(\lambda x.f(x)) = \lambda x.f(x+1)$$

then the first of the two evaluations of $P[f(x+1)]$ is $P[E(\lambda x.f(x))]$, the second is $E[P(\lambda x.f(x))]$. If we use f for $\lambda x.f(x)$ these are $P[Ef]$ and $E[Pf]$ respectively.

2. Functional Abstraction

Idea of functional abstraction

The examples of the last section can be generalized. The idea of a certain generalization of this kind is implied in the evaluation of D, J, P and E as functions; they are functions whose arguments are other functions; except for J, their values are also functions.

As stated earlier we use $\lambda x.M$ to denote a function itself. The formation of $\lambda x.M$ from x and M is called *functional abstraction.* For functions of several arguments we might similarly define *n-ary* functional abstraction as

(1) $\lambda^n x_1, \ldots, x_n .M$

which means the function whose value is M when the arguments are $x_1, \ldots, x_n$.

Certain assumptions are very important. For example, let us take addition, whose value is $x+y$ for the argument x and y. If we regard x as a fixed value, the function $\lambda y\,(x+y)$ (or $\lambda y.x+y$) will stand for the operation of adding the argument to x. If we use the generalized concept of a function, this can be regarded as itself the value of a function of x. This will correspond to our conventions as $\lambda x\,(\lambda y\,(x+y))$ or $\lambda x.\,\lambda y.x+y$. We can adopt the definition:

$$\lambda^2 xy.x+y \equiv \lambda x.\lambda y.x+y$$

If we assume $M=\lambda y.x+y$ then the above equation will become

$$\lambda^1 x.M \equiv \lambda x.M$$

In general

$$\lambda^{n+1} x_1, \ldots, x_n\, y.M \equiv \lambda^n x_1, \ldots, x_n\,(\lambda y.M)$$

Thus we can express functions of any number of arguments by means of simple functional abstraction. From here on, the exponent of λ will be omitted for the sake of simplicity.

Bound variables and functional abstraction

A system contains bound variables when there is a formulated set of substitutive variables (i.e., x in $f(x)=x^2+2x$) and at least one proper operation in which these variables play a special role.

Let us call the proper operation mentioned above a *binding operation*. Other operations will be considered as ordinary operations.

Any binding operation can be defined in terms of a functional operation and an ordinary operation. For example, let f be a primitive binding operation with m binding arguments and n ordinary arguments, shown as

$$f(x_1, \ldots, x_m, M_1, \ldots, M_n)$$

where x_i is a binding argument, M_j is an ordinary argument. Let $M_j' = \lambda x_1, \ldots, x_m . M_j$ and let F be a new ordinary operation of n arguments. Then the above primitive binding operation f will become the ordinary function $F(M_1', \ldots, M_n')$. So by way of bound variables and functional abstraction we are able to use functions as arguments.

3. Conversion Rules

In this section, we will consider how to formulate an equality relation in the system.

As a relation, equality is supposed to satisfy the following properties :

$X \mathbf{R} X$ (Reflexiveness)(ρ)

$X \mathbf{R} Y \Longrightarrow Y \mathbf{R} X$ (Symmetry)(σ)

$X \mathbf{R} Y \ \& \ Y \mathbf{R} Z \Longrightarrow X \mathbf{R} Z$ (Transitivity)(τ)

$X \mathbf{R} Y \Longrightarrow XZ \mathbf{R} YZ$ (Right monotony)(υ)

$X \mathbf{R} Y \Longrightarrow ZX \mathbf{R} ZY$ (Left monotony)(μ)

A relation that is left and right monotone is called a *monotone* relation. In order to have the replacement theorem we must also have the rule

$$A = B \Longrightarrow \lambda x.A = \lambda x.B \quad (\xi)$$

Since these properties of equality will not complete all the properties of it, we need certain other principles. The following section will describe those which are defined in Church [1941] and discussed in Curry and Feys [1958].

β-conversion rules

If we consider the meaning of bound variables, it is clear that they are irrelevant; the correspondence is the same no matter what variable is used to indicate it. Thus we should like to have the axiom scheme

$$\lambda x.X \equiv \lambda y.[y/x]X$$

where $[y/x]$ means substitution of y for x. However, as can be realized, this scheme will create some confusion. Let us look at the following example.

If X were xy, the above equation would be

$$\lambda x.xy = \lambda y.yy$$

where two sides obviously do not have the same meaning. This situation is called *confusion of bound variables.* In another example

$$\int_0^3 6xy\ dx = 27y$$

if we change variable x to y then equation should become

$$\int_0^3 6y^2\ dy = 27y$$

which is false.

To get rid of this confusion of variables, we add some restrictions on the scheme such that

(α) If y is not free in X then

$$\lambda x.X = \lambda y.[y/x]X.$$

In the next step, if $\lambda x.M$ is the function whose unspecified value is M, then its application to any N must be the same as the result of substituting N for x in M, as shown in the formulation

(β) $\quad (\lambda z.M)N = [N/z]M.$

Here, there is a possibility of confusion of variables to be gotten rid of. Assume $M \equiv \lambda y.xy$ and $N \equiv y$. Then substitution of N for x in M without considering the bound variables would lead to $\lambda y.yy$. But if we first transform M to $\lambda z.xz$ by (α) and then substitute, the result will be $\lambda z.yz$. This kind of confusion may occur if there is a free variable in N which is bound in M. This possibility can be solved by adding a restriction to (β). But if we change the definition of substitution in such a way that bound variables are shifted automatically so as to avoid confusion, then (β) may be accepted without restriction.

General Concept

The monotone equivalence generated by (α) and (β) is called β-convertibility, that generated by (α) alone is called α-convertibility. Besides equivalence, the monotone quasi ordering which is called *reducibility* is also used in Church's theorems. We will symbolize it as $\geqslant$. The monotone quasi-ordering which we will use in the Church Rosser theorem is called *reducibility*. Conversion is a transformation of an ob into one with which it is convertible. The ob which can not be reduced in any way is in *normal form*. Certainly there are different kinds of reduction as in the case of conversion. Since α is symmetric α-reduction and α-conversion are the same. A reduction is a transformation of an object into which it is reducible. The converse transformation is called an expansion.

η-conversion Rules

This conversion rule says:

If x is not free in M, then $\lambda x\,(Mx)\ \mathbf{R}\ M$

This rule is intuitively acceptable for convertibility, because both sides of the relation represent the function whose value for the argument X is MX. On the other hand there are purposes for which the rule is not acceptable, because the left side is a function, while the right side may not be. But this is matter of interpretation; because in general, every object is a function too.

The rules (ξ) and (η) together are equivalent to the following rule which is a form of the principle of extensionality:

(ς) If x is not free in either M or N, then

$$Mx = Nx \Longrightarrow M = N$$

The rule (ς) follows from (ξ) and (η) as a result:

$$Mx = Nx \Longrightarrow \lambda x.(Mx) = \lambda x.(Nx) \quad \text{by } (\xi)$$

$$\Longrightarrow M = N \quad \text{by } (\eta)$$

Conversely (η) and (ξ) follow from (ς) together with (β) thus:

$$(\lambda x.Mx)x = Mx \quad \text{by } (\beta)$$

$$\lambda x.Mx = M \quad \text{by } (\varsigma)$$

This proves (η). To prove (ξ) we have

$$M = N \Longrightarrow (\lambda xM)x = (\lambda xN)x \quad \text{by } (\beta)$$

$$\Longrightarrow \lambda xM = \lambda xN \quad \text{by } (\eta)$$

We call the lambda-conversion calculus with the rule (η) the $\beta\eta$-calculus.

Redexes

The terminology introduced here simplifies many of the succeeding formulations.

We call an object which can form the left side of an instance of one of the rules (β), (η), or (δ) (introduced later) a *redex* of the corresponding type, the rightside of the same instance will be called the *contractum* of the redex. A replacement of redex by its contractum will be called a *contraction* of the type of the rule. Thus a redex of type (β), or simply β-redex is an object of the form $(\lambda x.M)N$, its contractum is $[N/x]M$; and a replacement of an instance of $(\lambda x.M)N$ by $[N/x]M$ is a β-contraction.

δ-conversion rules

This is a third kind of reduction and a third type of λ-calculus, giving a rule of conversion of the following kind:

(δ) Let M be an ob which is not β-redex and not of the form $\lambda x.N$, and let M contain no free variables nor any proper components which are redexes of any kind. Let M' be an object such that no constituent of M' is a free variable and M' is not a redex of the same kind as M. Then M is convertible into M'.

An object M to which such a rule may be applied is called a δ-redex. It is clear that a δ-redex is of the form

$$aM_1M_2, \ldots, M_n,$$

where a is a primitive constant and $M_1, \ldots, M_n$ are in normal form and contain no free variables.

A λ-calculus which admits a form of the rule (δ) along with (α) and (β) will be called a $\beta\delta$-calculus; if it admits also the rule (η), it will be called $\beta\eta\delta$-calculus, or simply a *full λ-calculus*. The Church Rosser theorem which is mentioned in the next section is based on an arbitrary full λ-calculus.

D. CHURCH ROSSER THEOREM

One of the main results of calculus of λ-conversion is the so-called Church-Rosser theorem. This theorem, shortly, may be stated as follows:

(χ) If $X = Y$, then there is an ob Z such that $X \geqslant Z$ & $Y \geqslant Z$.

The property (χ) is known as Church-Rosser property. Generally let '=' be infix equivalence relation generated by a relation $\geqslant$. Then the property (χ) is as follows:

If $X = Y$ then there is a Z such that $X \geqslant Z$ & $Y \geqslant Z$

The classical Church-Rosser theorem is, then, the following:

> If $\geqslant$ is the reducibility relation defined earlier for any of the forms of λ-calculus then the property (χ) holds.

The proof of this theorem has been studied (besides Church and Rosser) by many other mathematicians. For references see Curry and Feys [1958].

E. CHURCH-ROSSER AND CONFLUENCE PROPERTIES

The following property which we call (θ) is implied by (χ):

If for some U

$$U \geqslant X \ \& \ U \geqslant Y,$$

then there is a Z such that

$$X \geqslant Z \ \& \ Y \geqslant Z.$$

They are shown in Figure 2.1.

1. Implications

The following theorem shows that (θ) is equivalent to (χ) provided that $\geqslant$ is quasi-ordering.

Theorem: If the relation holds for the properties (ρ) & (τ) & (θ),

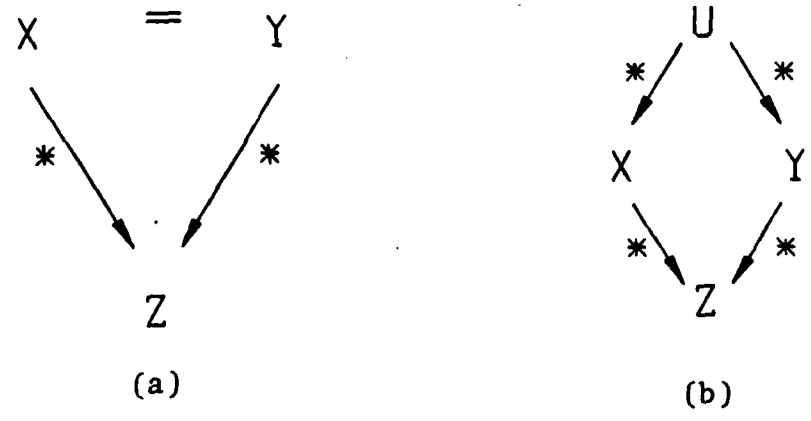


Figure 2.1 (a) Church Rosser Property, (b) Property θ

then it holds for (χ).

Here what we will say is property (θ) is almost confluent.[3]

[3] For proof see Curry and Feys p. 112.

III. PROPERTIES RELATED TO THE CONFLUENCE

In the second chapter we mentioned the relation between the Church-Rosser property and the confluence property. Here we will look at the confluence property in terms of term rewriting systems.

A. GENERAL CONFLUENCE

Let a class of objects be given, and a set P of object pairs such that one is obtained from the other by a *move*, and the two objects are regarded as equivalent if and only if one is obtainable from the other by a sequence of moves. For example, in group theory the objects are words made from an alphabet $a, b, \cdots, a^{-1}, b^{-1}$ (where a^{-1} is the inverse of a) and a move is the insertion or removal of a consecutive pair of letters xx^{-1} or $x^{-1}x$.

In Church's λ-calculus we define λ-conversion as the reflexive and transitive closure of α- and β-conversion rules. λ-conversions are kinds of these moves.

As we defined earlier, the moves of λ-conversion naturally fall into two categories, *reductions* and *expansions*. Also in the example of group theory, cancelling of a pair of letters can be called reduction, the insertion is, then, expansion. This dichotomy between reduction and expansion plays an important role in confluence relations.

If a relation is transitive, confluence is equivalent to the Church-Rosser property, which expresses the fact that equivalence (or interconvertibility) of two terms can be checked by reducing them to a common form.

If A an B are "equivalent", it follows that there exists a third object C obtainable both from A and B by reduction sequences.

Another problem in confluence theorems is the search for "end forms" or "normal forms", i.e., objects which admit no reduction. In any theory in which the confluence property holds no equivalence class can contain more than one normal form (see Lemma 3.1). However, if there exists infinite sequences of reductions which do not terminate then there is a question of whether or not normal form exist. In the following, we will follow the terminology and notation

found in Huet [1980]. We will use arrows as relations, since we are going to deal with rewrite rules which will be explained in the fourth chapter.

1. Notation

Let Σ be an arbitrary set. Let $\rightarrow$, $\rightarrow_a$, $\rightarrow_b$ be symbols for reduction.

ι is identity relation on Σ, that is, $\iota = \{<x,x> \mid x \text{ is in } \Sigma\}$

. is operator for composition of relations. So

$\rightarrow_a . \rightarrow_b = \{<x,y> \mid \text{There is a } z \ x \rightarrow_a z \ \& \ z \rightarrow_b y\}$

$\rightarrow^{-1}$ is inverse relation of $\rightarrow$, that is, $\rightarrow^{-1} = \{<x,y> \mid y \rightarrow x\}$.

With these definitions :

$\rightarrow^0 = \iota$.

$\rightarrow^e = \rightarrow \bigcup \iota$ (Reflexive closure)

$\rightarrow^i = \rightarrow . \rightarrow^{i-1}$, $i > 0$.

$\rightarrow^+ = \bigcup_{i>0} \rightarrow^i$ (Transitive closure)

$\rightarrow^* = \rightarrow^+ \bigcup \iota$. (Reflexive, transitive closure)

$\leftrightarrow = \leftarrow \bigcup \rightarrow$. (Symmetric closure)

If x is element of Σ and there is no y such that $x \rightarrow y$, then x is a $\rightarrow$-*normal form.* Let N be the set of all such elements. For y, an element of Σ, if there exists an x element of N such that $y \rightarrow^* x$, then x is a $\rightarrow$-normal form of y. For a relation $\rightarrow$, we let

$x \rightarrow^* . \leftarrow^* y$ if and only if there exists a $z \mid x \rightarrow^* z$ and $y \rightarrow^* z$.

$x \leftarrow^* . \rightarrow^* y$ if and only if there exists a $z \mid z \rightarrow^* x$ and $z \rightarrow^* y$.

$\Lambda(x) = \{i \mid \text{there is } y \mid x \rightarrow^i y\}$, an element of $N \bigcup \{\infty\}$

$\Delta(x) = \{y \mid x \rightarrow y\}$.

$\Delta^-(x) = \{y \mid x \rightarrow^+ y\}$

$\Delta^*(x) = \Delta^+ \bigcup \{x\}$.

Relation $\rightarrow$ is

(i) *Inductive* iff for every sequence $x_1 \rightarrow x_2 \rightarrow \cdots \rightarrow x_n \cdots$ there is a y such that for all $i \geq 1$ $x_i \rightarrow^* y$

(ii) *acyclic* iff $\rightarrow^+$ is irreflexive (Then $\rightarrow^*$ is a partial ordering)

(iii) *terminating* iff there is no infinite sequence $x_1 \rightarrow x_2 \rightarrow \cdots \rightarrow x_n \rightarrow \cdots$

(then $\rightarrow^*$ is well founded which makes sense in some mathematical discourse. When a careful definition is required, the inductive definitions are used to characterize the set of well founded formulas.)

(iv) *bounded* iff for all x, $\Lambda(x) < \infty$ (then $\rightarrow^*$ has the finiteness property)

Every bounded relation is terminating and every terminating relation is inductive and acyclic. Let P be any predicate on Σ. We say that P is $\rightarrow$-complete iff

For all x in Σ [For all y in $\Delta^+(x)$ $P(y)$] $==>$ $P(x)$.

We say that $\rightarrow$ is locally finite iff for all x in Σ $\Delta(x)$ is finite.

Let $\rightarrow$ be a locally finite relation. For every x in Σ, if $\Lambda(x) = \infty$, then there exists an infinite sequence $x = x_1 \rightarrow x_2 \rightarrow \cdots \rightarrow x_n \rightarrow \cdots$. Therefore a locally finite relation is bounded iff it is terminating.

We say that $\rightarrow$ is *globally finite* iff for all x in Σ $\Delta^*(x)$ is finite. A terminating locally finite relation is globally finite. (reference Huet and Oppen [1980])

2. Confluence Properties

Relation $\rightarrow$ is *locally confluent* iff $\leftarrow.\rightarrow$ is a subset of $\rightarrow^*.\leftarrow^*$, or in another words, for all x,y,z there is u such that $x \rightarrow y$ & $x \rightarrow z$ $==>$ $y \rightarrow^* u$ & $z \rightarrow^* u$.

We say $\rightarrow$ is *globally confluent* iff for all x,y $x \leftarrow^*.\rightarrow^* y$ $==>$ $x \rightarrow^*.\leftarrow^* y$. In Figure 3.1, these properties are shown. In this figure, dashed arrows denote reductions depending on the reductions shown by full arrows.

From now on, we will use confluent to mean globally confluent.

The relation $\rightarrow$ is interpreted as β-reduction in λ-calculus, and the operational semantics in a programming language (see Huet [1980]).

B. RELATION TO CHURCH-ROSSER PROPERTY

Theorem 3.1: A relation is confluent if and only if it has the Church-Rosser property (χ).

Proof:This can be shown by proving that

$\leftarrow\rightarrow^* = \rightarrow^*.\leftarrow^*$.

The only if part is trivial since $\{(\rightarrow^*.\leftarrow^*)\}$ is subset of $\{(\rightarrow \mid \leftarrow)^*\}$. For the if part we have to show that $\{(\rightarrow \mid \leftarrow)^*\}$ is subset of $\{(\rightarrow^*.\leftarrow^*)\}$. Since

Figure 3.1. (a) Local Confluence Property, (b) Global Confluence Property.

$(\rightarrow \mid \leftarrow)^* = \bigcup_{k=1}^{\infty}(\rightarrow \mid \leftarrow)^k$, then for $k = 1$, $(\rightarrow \mid \leftarrow)$ is trivially a subset of $\rightarrow^*.\leftarrow^*$.

We will prove that $(\rightarrow \mid \leftarrow)^*$ is a subset of $\rightarrow^*.\leftarrow^*$, by using induction. To prove that $(\rightarrow \mid \leftarrow)^k$ is a subset assume it is true for all $k \leq n$ hence $\bigcup_{k=1}^{n}(\rightarrow \mid \leftarrow)^k$ is a subset of $\rightarrow^*.\leftarrow^*$. Now we must show it is true for all $k \leq n+1$.

If (x,y) is in $\bigcup_{k=1}^{n+1}(\rightarrow \mid \leftarrow)^k$ then there exists a z such that (x,z) ϵ $\bigcup_{k=1}^{n}(\rightarrow \mid \leftarrow)^k$, (z,y) ϵ $\rightarrow$ or $\leftarrow$. Let us look at these two cases:

Case 1: (z,y) ϵ $\leftarrow$. Since (x,z) ϵ $\bigcup_{k=1}^{n}(\rightarrow \mid \leftarrow)^k$, by assumption (x,z) ϵ $\rightarrow^*.\leftarrow^*$. As a result (x,y) ϵ $\rightarrow^*.\leftarrow^*.\leftarrow$ which is equal to $\rightarrow^*.\leftarrow^*$.

Case 2: (z,y) ϵ $\rightarrow$. Since (x,z) ϵ $\bigcup_{k=1}^{n}(\rightarrow \mid \leftarrow)^k$ we have (x,y) element $\rightarrow^*.\leftarrow^*.\rightarrow$. By confluence property $\leftarrow^*.\rightarrow$ is a subset of $\rightarrow^*.\leftarrow^*$. Therefore (x,y) is in $\rightarrow^*.\rightarrow^*.\leftarrow^*$ which is equal to $\rightarrow^*.\leftarrow^*$.

This completes the proof.

C. LEMMAS ON RELATIONS WHICH ARE CONFLUENT

Lemma 3.1: If a relation is confluent (global), then the normal form of any element, if it exists, is unique.

We can prove this by contradiction. Assume x in Σ has two normal forms y and z. By confluence property $(y,z) \in (\leftarrow^{*}.\rightarrow^{*})$ implies $(y,z) \in (\rightarrow^{*}.\leftarrow^{*})$. Then there is a u such that $y \rightarrow^{*} u$ & $z \rightarrow^{*} u$. But by definition of normal form there is no such u to which both y and z reduce. So y and z are the same object. This completes the proof.

We define a relation $\rightarrow$ to be *semi-confluent* if and only if $\leftarrow.\rightarrow^{*}$ is a subset of $\rightarrow^{*}.\leftarrow^{*}$.

Lemma 2.2 : A relation is confluent if and only if it is semi-confluent.

Proof: Let P(k) be:

$(\leftarrow^{k}.\rightarrow^{*}) => (\leftarrow^{*}.\rightarrow^{*})$

then theorems turns into:

P(1) iff P(k) for all $k \geq 0$.

The only if part is trivial. For the if part, assume P(1) is true. By induction on k:

(1) P(1) is true by assumption

(2) Assume P(n) is true,

(3) Show that P(n+1) is true then the proof is done.

Assume (y,z) is an element of $(\leftarrow^{n+1}.\rightarrow^{*})$. Then there is an x such that $x \rightarrow^{k} y' \rightarrow y$ and $x \rightarrow^{*} z$. So by (2) on (y',z), there is a v such that $y' \rightarrow^{*} v$ & $z \rightarrow^{*} v$. By (1) on (y,v), there is a u such that $y \rightarrow^{*} u$ & $v \rightarrow^{*} u$. Since $z \rightarrow^{*} v \rightarrow^{*} u$, $z \rightarrow^{*} u$. So the pair (y,z) is an element of $(\rightarrow^{*}.\leftarrow^{*})$. This completes the proof.

D. LOCALIZATION OF CONFLUENCE

Lemma 2.3: A terminating relation is confluent if and only if it is locally confluent.

Proof: It is sufficient to prove that local confluence implies global confluence. Assume $\rightarrow$ is locally confluent. Define the set A as follows :

$A = \{ x \mid \leftarrow^{*} x \rightarrow^{*}$ is not a subset of $\rightarrow^{*}.\leftarrow^{*} \}$

If $A = \Phi$, the proof is complete. Assume A is not empty. Let x be a *rightmost* element of A, i.e., if $x \rightarrow y$, then y is not an element of A. Such elements exist,

for if $x_0 \in A$ and x_0 is not rightmost there exists $x_1 \in A$, $x_0 \rightarrow x_1$. If x_1 is not rightmost then there exists $x_2 \in A$ such that $x_0 \rightarrow x_1 \rightarrow x_2$ etc. Thus there exists a sequence of elements in A

$$x_0 \rightarrow x_1 \rightarrow x_2 \rightarrow \cdots \rightarrow x_i \rightarrow \cdots$$

By the terminating condition, this sequence terminates in some element $x_N \in A$, which must be rightmost in A.

Consider $\leftarrow^* x \rightarrow^*$. Let (y,z) be an element of $\leftarrow^* x \rightarrow^*$ that is not in $\rightarrow^*.\leftarrow^*$. Since $\rightarrow$ is locally confluent (y,z) is not in $\leftarrow.\rightarrow$.

Assume $(y,z) \in \leftarrow^* y' \leftarrow x \rightarrow x' \rightarrow^* z$

By local confluence $(y',z') \in \rightarrow^* u \leftarrow^*$ for some u. Also since $x \rightarrow y'$ and $x \rightarrow z', y'$ and z' are not in A. Therefore we have

$$(y,z) \in \leftarrow^* y' \rightarrow^* u \leftarrow^* z' \rightarrow^*$$

where $\leftarrow^* y' \rightarrow^*$ and $\leftarrow^* z' \rightarrow^*$ are subsets of $\rightarrow^*.\leftarrow^*$. (since $(y',z') \in A$)

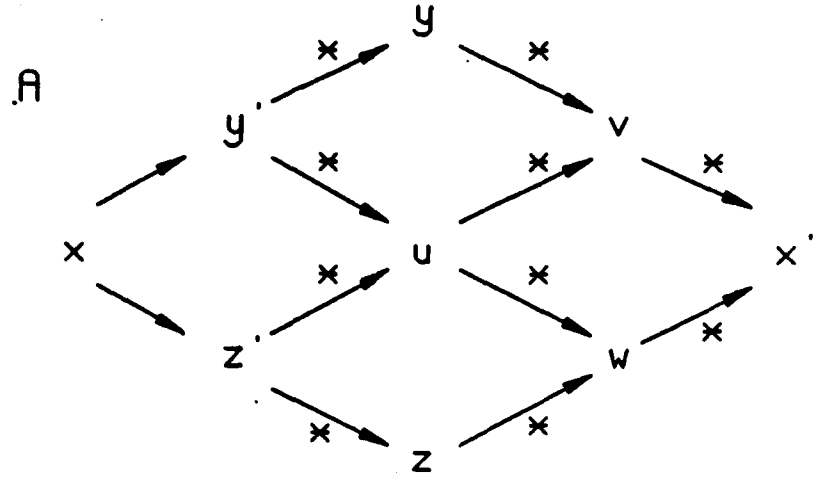


Figure 3.2. Localization of Confluence Property

Assume $(y,u) \in \rightarrow^* v \leftarrow^*$, and $(u,z) \in \rightarrow^* w \leftarrow^*$. Thus $(v,w) \in \leftarrow^* u \rightarrow^*$. But since y',z' are not in A, and $y' \rightarrow^* u$, $z' \rightarrow^* u$ then u is not in A. Thus there exists an x' such that $(v,w) \in \rightarrow^* x' \leftarrow^*$. Then

$$(y,z) = (y,u).(u,z) \in \rightarrow^* v \leftarrow^*.\rightarrow^* w \leftarrow^* \quad \text{which is subset of} \quad \rightarrow^* \rightarrow^* x' \leftarrow^* \leftarrow^* = \rightarrow^* x' \leftarrow^*.$$

But $(y,z) \in \rightarrow^*.\leftarrow^*$, and this is a contradiction. Thus A must be empty, and this completes the proof. Figure 3.2 is a diagramatic representation of the proof.

Corollary: A terminating relation satisfies the Church-Rosser property if and only if it is locally confluent.

Proof: By Lemma 3.3 we showed termination and local confluence is equivalent to global confluence, and by theorem 3.1 we know confluence and the Church-Rosser property are equivalent. Then, since arrows in these theorems are bidirectional, we say that termination and local confluence is equivalent to the Church-Rosser property. This completes the proof.

IV. USES OF THE CONFLUENCE PROPERTY

Determination of confluence in a system is an integral step towards deciding various properties of a system that is formally defined. The systems handled in this chapter are term rewriting systems associated with abstract data types. In the first section we briefly describe algebras to provide a background. We discuss an initial algebra approach for implementation and correctness of abstract data types, as used in a high level language. (see, for example Goguen [1977])

A. AN INITIAL ALGEBRA FOR ABSTRACT DATA TYPES

Abstract data types are a powerful tool in programming in two different aspects.First it is convenient for the user to think in abstract terms, and second, abstraction provides a means for discussing software independently of implementation. Algebras have been found to be a promising method for the specification of abstract data types (Guttag [1978]). We will study an implementation of abstract data types (such as stack, queue) as initial algebras. We assume that reader is familiar with abstraction in terms of computer science.

1. What is an algebra

An *algebra* is composed of two main parts, the first one includes two subparts which are the *carriers* and the collection of *operators* of the algebra. The index set of carriers may be one or more and is called the *sort* set. If there is one element in this set, the algebra on this set is called a one-sorted algebra. As an example of a sort set {real, boolean, integer} is a set of the sorts real, boolean and integer. But, as we realize, since an algebra can have an infinite number of elements, this is not enough to specify an algebra. We will come to that point in the specification of abstract data types.

If S is the set of sorts of an algebra, then the *signature* Σ is defined to be the collection of sets $\Sigma_{w,s}$ where $w \in S^*$, $s \in S$ that describe the sets of operations of the form

$$F: A_{s_1} \times \cdots \times A_{s_n} \rightarrow A_S$$

where $w = s_1 s_2 \cdots s_n$

that are in the algebra.

For example, the operation + on integers may be denoted as:

$+ : intg, intg \rightarrow intg$

and is an element of $\Sigma_{intg\ intg, intg}$. So if Σ is defined only on intg (integer), it is a one-sorted signature. If an operation $F \in \Sigma_{w,s}$, the *arity* of F is $|w|$ where $|w|$ is number of sorts in w. The *sort* of F is s if $F \in \Sigma_{w,s}$.

In a boolean algebra

$T : \rightarrow bool$ and $F : \rightarrow bool$

may be considered as constants (which are 0-ary functions).

In general the components of the signature Σ for integer are as follows:

$\Sigma_{\lambda,s} = \{ 0 \}$,

$\Sigma_{s,s} = \{ - \}$,

$\Sigma_{ss,s} = \{ -, + \}$,

$\Sigma_{sss,s} = \Phi$,

.

.

.

There is an ambiguity here, the operator - is used both as a unary and binary operator, namely, as negation and subtraction respectively. For integers there is no ternary, 4-ary, etc. operations, so their sets are empty.

If two algebras have different carriers but, the same signature Σ, then they are called Σ-algebras. When A and B are Σ-algebras, A is a subalgebra of B means that A is a subset of B (for their carriers) and that each operation named by $F \in \Sigma_{s_1 \cdots s_n, s}$, in A is exactly that in B, restricted to the carriers of A; such that $a_i \in A_{s_i}$, for $i = 1, \ldots, n$,

$$F_A(a_1, \ldots, a_n) = F_B(a_1, \ldots, a_n).$$

Also if A and B are both Σ-algebras, a Σ-homomorphism $h : A \rightarrow B$ is a family of functions $<h_s : A_s \rightarrow B_s>_{s \in S}$ that preserve the operations

(h0) If $F \in \Sigma_{\lambda,s}$, then $h_s(F_A) = F_B$;
(h1) If $F \in \Sigma_{s_1 \cdots s_n,s}$ and $<a_1, \ldots, a_n> \in A_{s_1} \times \cdots \times A_{s_n}$, then
$h_s[F_A(a_1, \ldots, a_n) = F_B[h_{s_1}(a_1), \ldots, h_{s_n}(a_n)]$.

A category C of Σ-algebras consists of a class of Σ-algebras together with all the Σ-homomorphisms between the algebras.

A homomorphism $h : A \rightarrow A'$ is an isomorphism iff there exists $g : A' \rightarrow A$ such that $gh = 1_A$ and $hg = 1_{A'}$ where 1_A is the identity function of A. The homomorphism g is called the inverse of h.

The basic concept of this section is the following:

> An algebra A is *initial* in a category C of Σ-algebras iff for every algebra B in C there exists a *unique* homomorphism $h : A \rightarrow B$.

The following is a corollary of this concept:

> If A and A' are both initial algebras in C, then A and A' are isomorphic. If A'' in C is isomorphic to A, then A'' is also initial (Goguen, Thatcher, and Wagner [1978]).

So the initial algebra in a category C of Σ-algebras characterizes the isomorphism class of an object; and by the meaning of isomorphism, this means it characterizes an object "abstractly", in terms of its structure. An *abstract data type* is the isomorphism class of an initial algebra in a category C of Σ-algebras. Thus we can speak of an initial algebra A in C as being the abstract data type. Certainly the categories C of Σ-algebras we are interested in are those which are finitely describable (since abstract data types are finitely describable). But not all the categories C of Σ-algebras are finitely describable. We are interested in categories C having as objects all Σ-algebras satisfying some finite set ξ of equations (in turn, axioms of a specification which we will describe in the next section) . The set ξ is the second part of an algebraic specification.

Abstract data types can be specified by equations, which are called axioms of the given abstract data type. We will next present the mathematics needed to do this.

There are two main theorems (in fact only one but two versions for different categories). The first one proposes that there is an initial Σ-algebra in

the category C_Σ of all Σ-algebras and the second covers the category $C_{\Sigma,\xi}$ of all Σ-algebras satisfying a set ξ of equations. As you will realize, the first is a special case of the second with $\xi = \Phi$.

Some examples of abstract data types which are initial in a category C_Σ are given below.

Example (1) The set of natural numbers is one of the most common data types. The Σ-algebra for it can be denoted as follows.

$$S = \{nat\},\ \Sigma_{\lambda,nat} = \{0\}\ ,\ \Sigma_{nat,nat} = \{SUCC\},\ \Sigma_{w,s} = \Phi \text{ otherwise.}$$

The basic idea here is that further operations on natural numbers can be expressed in terms of the two basic ones, SUCC and 0.

A property that the algebraic approach shares with all abstract or axiomatic characterizations is independence of representation. So we are not committed to thinking of integers as strings of decimal, or binary, or Roman characters. This is certainly crucial to being able to prove correctness of data representations.

Example (2): Another specification is that of the boolean data type. The Σ algebra for it can be denoted as follows:

$$S = \{bool\},\ \Sigma_{\lambda,bool} = \{T,F\}\ ,\ \Sigma_{bool,bool} = \{not\},\ \Sigma_{bool\ bool,\ bool} = \{And\},\ \text{and}\ \Sigma_{w,s} = \Phi \text{ otherwise.}$$

The carriers of initial Σ-algebras, in categories of algebras satisfying certain identities, will consist of equivalence classes of Σ-terms, and the familiar methods of algebra (substitution of equals for equals, reduction, replacement, etc.) are crucial for our proofs of correctness of data type specifications and for our ideas about automatic implementation of data types from their specifications.

For the type *nat*, an initial algebra T_Σ is isomorphic to the set $\omega = \{0,1,\cdots\}$ of nonnegative integers by the correspondence of n with $SUCC^n(0)$, where $SUCC^n(0)$ is the repetition of SUCC n times.

We want to constrain initial algebras to satisfy certain laws or equations. For example, we want the binary operation TIMES to be associative, i.e., to satisfy

$$\text{TIMES(X, (TIMES(Y, Z)) = TIMES (TIMES(X, Y), Z)}$$

To make clear the ideas of *equation* and *satisfaction* requires a somewhat elaborate preparation. The status of the *variables* (X, Y, Z above) has to be clarified. The basic idea is that for each sort in S, there should be an infinite supply of special symbols disjoint from any signature. To get variables into the terms of an algebra, we can consider them as constants, such as nullary functions. We see here that every variable in an algebra belongs to a specific sort. Since we know each operator must be a single sort, we can replace a variable with a corresponding operator in a given term (but this will not give us any advantage in our work), or in fact any operator belonging the same $\Sigma_{w,s}$ class in Σ can be changed to the other. This is called a *substitution.* In fact as we will see in the following definition, the right hand side of an equation is a substitution of one side by the other. If we consider the right hand side of an equation to be simpler than the left, then the number of operands in the replacing operator must be less than the replaced one. Certainly, if the replacing operator is nullary, this will be the most desirable one.

> Definition: A Σ-equation is a pair $e = \langle L, R \rangle$ where L, R are terms of an algebra A. A must satisfy the equations. The necessary condition for this is that number of variables on the left must be equal those on the right. If A satisfies every e in ξ then such a set of equations is called a Σ-representation (axiom set of A), and the algebra A is called a (Σ,ξ)-algebra, and the category of (Σ,ξ)-algebras is denoted by $C_{\Sigma,\xi}$.

Let $T_{\Sigma,\xi}$ be an initial algebra in the category $C_{\Sigma,\xi}$. We shall say that $T_{\Sigma,\xi}$ is presented by ξ. The construction of such a $T_{\Sigma,\xi}$ needs some machinery.

> Definition: A Σ- *congruence* $\equiv$ on a Σ-algebra is a family $\langle \equiv_s \rangle$ of equivalence relations, $\equiv_s$ on A_s (A_s is the carrier set of sorts A_{s_i}) for $s \in S$, such that if $F \in \Sigma_{s_1 \ldots s_n, s}$, and if $a_j, a_i' \in A_{s_i}$, and if $a_i \equiv_{s_i} a_i'$ for $i = 1, \ldots, n$, then $F_A(a_1, \ldots, a_n) \equiv_s F_A(a_1', \ldots, a_n')$.

If A is a Σ-algebra and $\equiv$ is a Σ-congruence on A, let $(A / \equiv)_s = A_s / \equiv_s$ be the set of $\equiv_s$-equivalence classes of A_s. For $a \in A_s$, let $[a]_s$ denote the $\equiv_s$-class containing a. Note that each element of $a_s / \equiv_s$ is of the form $[a]_s$, but of course the choice of $a \in A_s$ is not uniquely determined. The idea here is to define an algebra by partitioning it into congruence classes.

The definition of the operation $F_{A/\equiv}$ is as follows:

(q0) If $F \in \Sigma_{\lambda,s}$, then $F_{A/\equiv} = [F_A]$;

(q1) If $F \in \Sigma_{s_1 \ldots s_n, s}$ and $[a_i] \in (A / \equiv)_{s_i}$ then $F_{A/\equiv}([a_1], \ldots, [a_n]) = [F_A(a_1, \ldots, a_n)]$.

This is also the definition of the homomorphism classes by the operation F. If A is a Σ-algebra and $\equiv$ is a Σ-congruence on A, then $A / \equiv$ is a Σ-algebra called the *quotient* of A by $\equiv$.

Let A be a Σ-algebra, and let R be a relation on A. Then there is a least Σ-congruence relation on A containing R; it is called the *congruence relation generated by R on A*.

The main theorem is as follows:

Theorem : Let ξ be a Σ-representation and let $\equiv_\xi$ be the Σ-congruence on T_Σ generated by $\xi(T_\Sigma)$. Then $T_\Sigma / \equiv_\xi$, the quotient of T_Σ by $\equiv_\xi$, hereafter denoted $T_{\Sigma,\xi}$, is the initial algebra in the category $C_{\Sigma,\xi}$ of all Σ-algebras satisfying ξ.

B. SPECIFICATION OF ABSTRACT DATA TYPES

In this section we will give specifications in which the set ξ of equations (axiom set) is nonempty. So these specifications rely upon Theorem 1.

1. Specifications

As we stated earlier, initial algebras may consist of an infinite number of objects. We want to find convenient ways to specify them in finite terms so we can use them as abstract data types. By starting with this purpose, a formal definition of a specification is as follows:

Definition: A *specification* is a pair $<\Sigma,\xi>$ where Σ is a composite of sort set S, and an S-sorted signature (note that Σ is extended from the previous meaning) and ξ is a set of Σ-equations.

The basic idea, here, is that $<\Sigma,\xi>$ specifies an abstract data type by defining the algebra $T_{\Sigma,\xi}$.

Sometimes we might want to add further equations on an existing type, so by adding the equation set ξ' on the algebra $T_{\Sigma<\xi}$ we reach $T_{\Sigma,\xi\cup\xi'}$ so that the new type is a quotient of the old.

Let us define a specification syntax similar to one defined in Davis [1984].

SPECIFICATION <Abs-data-type>
  OPERANDS
    <sort set S>
  OPERATIONS
    op1: $S^{'} \rightarrow S$
    ...
    opn: $S^{'} \rightarrow S$
  AXIOMS
    <equation set ξ>

With this syntax, a specification of the data type *integer* is as follows:

SPECIFICATION integer
  OPERANDS
    int
  OPERATIONS
    0 : → int,
    SUCC : int → int,
    PRED : int → int,
  AXIOMS
    PRED(SUCC(X)) = X,
    SUCC(PRED(X)) = X

where SUCC and PRED are inverses of each other, and X is a free variable of sort int.

We can enrich this specification by adding new operations to the specification without disturbing the above specification:

SPECIFICATION
  OPERANDS
    int
  OPERATIONS
    0 : → int,
    SUCC : int → int,
    PRED : int → int,
    ADD : int, int → int
    SUB : int, int → int
    MULT : int, int → int
    NEG : int → int
  AXIOMS
    PRED(SUCC(X)) = X,
    SUCC(PRED(X)) = X,
    ADD (X, 0) = X,

ADD (X, Y) = ADD(Y, X),
ADD (X, ADD(Y, Z)) = ADD(ADD(X, Y), Z),
ADD (X, SUCC(Y)) = SUCC (ADD(X, Y)),
ADD (X, PRED(Y)) = PRED (ADD(X, Y)),
SUB (X, 0) = X,
SUB (X, SUCC(Y)) = PRED (SUB(X, Y)),
SUB (X, PRED(Y)) = SUCC (SUB(X, Y)),
NEG (0) = 0,
MULT (X, 0) = 0,
MULT (X, SUCC(Y)) = ADD(MULT(X, Y), X),
MULT (X, PRED(Y)) = SUB(MULT(X, Y), X),

So the operations SUB, ADD, and MULT are operations *derived* from SUCC, and PRED.

2. Extension

Following the above specification of integer, we may add new operations which involve other sorts. For example, we may want to add *predicates, conditionals* or *relations* to an existing type which does not have the boolean type. In some sense we certainly extend the syntax above for specifications as in the following example (for the sake of simplicity, we will skip the parts already written).

```
SPECIFICATION integer
    EXTEND
      boolean
    WITH
    OPERANDS  int
    OPERATIONS
      ...

      LTE : int, int → bool
    AXIOMS
      ...

      LTE (X, X) = TRUE,
      LTE (X, Y) = LTE(SUCC(X), Y),
      LTE (X, Y) = LTE(SUB(X, Y), 0)
```

Certainly X, Y are free variables of sort int.

3. Decidability of Equivalence

For a given specification (Σ,ξ), when is the equivalence of two terms decidable?

We first give some definitions. A *term* in a specification is (a) a variable symbol, or (b) a function followed by finite number of terms recursively. The *length* of a term is the number of function and variable symbols in it. For instance,

Length(f(x, g(x, y), h(z)) = 7

If we can prove that the axioms of a specification are globally confluent, then it follows that equivalence of any two terms is decidable. To show global confluence, we have to show that the axioms are locally confluent and finitely terminating.

As is shown by the following two theorems from Huet and Lankford [1978], finite termination is in general undecidable.

Theorem 1: The finite termination problem of term rewriting systems is undecidable even if terms are restricted to unary and nullary functions.

Theorem 2: There is no decision procedure for finite termination of term rewriting systems.

Here, the theorems are based on term rewriting systems. But as we will show in the next section, every specification is also a term rewriting system.

Because of these results, we need some sufficient conditions to guarantee finite termination. We define an axiom $x \rightarrow y$ as *nonexpanding* if for every substitution on both x and y, $\text{Length}(\theta(x)) \geq \text{Length}(\theta(y))$ (where θ is the substitution prefix). Proof of the following theorem is a sufficient condition for the finite termination of the rewrite rules.

Theorem 3.1: If a rewriting relation $\rightarrow$ is nonexpanding, then it is finitely terminating.

Proof: Since the relation is nonexpanding, $t \rightarrow^* u$ implies $\text{Length}(u) \leq \text{Length}(t)$. Also the only variables that can occur in u are those in t. Thus there are only

finitely many possibilities for u, and → must be finitely terminating (Guttag, Kapur, Musser [1983]).

As an example let us work on specification of BOOLEAN.

SPECIFICATION Boolean
OPERANDS Bool
OPERATIONS
  TRUE : → Bool
  FALSE : → Bool
  NOT : Bool → Bool
  AND : Bool, Bool → Bool
AXIOMS
  NOT(TRUE()) → FALSE()
  NOT(NOT(x)) → x
  AND(TRUE, x) → x
  AND(FALSE, x) → FALSE

We have to show every sequence of reductions terminates. We induct on reduction of a term with length k. If k=1, then it holds since it must be a nullary operator or a variable which is considered as a constant of the specification. Assume it holds for all terms of length k ⩽ n. Consider a term t of length n+1. Then it is in one of the forms:

NOT(x) or AND(y)

where the length of x and y are ⩽ n. So the reductions will be one of the following forms:

NOT(x) → x'

AND(y) → y'

By the axioms the length of x' and y' are ⩽ n. This completes the inductive proof.

But as we will realize, requiring a rewriting relation to be nonexpanding is somewhat restrictive. Consider if we added another operation OR to above specification, then the rule:

AND(x, OR(y,z)) → OR(AND(x,y), AND(x,z))

will be expanding, which is in fact very useful.

The next step for decision is to show local confluence of the specification. This is the topic of the next chapter which is based on (Knuth and Bendix [1970]).

4. Correctness of Specifications

Some abstract data types, i.e., natural numbers, integers, strings etc., have previously existing mathematical models. Some others are brought by computer science, such as stack, symboltable etc. For these there may be found more or less acceptible mathematical models. Some others are only defined by the user especially for his program, so the responsibility belongs to them.

If there is a mathematical model, it is necessary to find a strict proof if it is correct. In this section we will try to give an explanation of a very heavily used method for proving the correctness of a specification. The idea here is that a specification (Σ,ξ) is correct if $T_{\Sigma,\xi}$ is isomorphic to the mathematical model.

Since we have to show an isomorphism, correctness proofs can be viewed the other way around, that is, we assume (Σ,ξ) is correct and show a model is correct by isomorphism to $T_{\Sigma,\xi}$. To explain the idea, let us give an example on naturals.

Earlier we said that the signature $\Sigma = \{0\}, \{SUCC\}, \phi, \ldots, \phi$ where $\xi = \phi$, specifies the natural numbers. Let $\omega=\{0, 1, 2, \ldots\}$, and let A be a Σ-algebra with $0_A = 0$, and $SUCC_A : n \rightarrow n+1$.

Since A is a Σ-algebra, there is a homomorphism h: $T_\Sigma \rightarrow A$ where T_Σ is an initial algebra in set C of Σ-algebra classes. What we have to show is that h is an isomorphism. So A is an initial algebra. Since h is a Σ-homomorphism, $h(0)= 0_A = 0$ is true. If $h(SUCC^n(0))= n$, then $h(SUCC^{n+1}(0))= SUCC_A(h(SUCC^n(0)))= SUCC_A(n)= n+1$, by definition of homomorphism. So we know h is surjective. It is also injective since n=p implies $SUCC^n(0)= SUCC^p(0)$. Thus h is an isomorphism.

The proof of correctness of naturals involved no equations, that is, the equation set ξ was empty. To prove the specification (Σ,ξ) where ξ is not empty, we need some further development of our methods. The idea is to get a (Σ,ξ)-algebra A whose carriers consist of canonical terms and then to show that A is isomorphic to the mathematical model M.

Definition: We say that a Σ-algebra A is a canonical Σ-term algebra if A_s is a subset of $T_{\Sigma,\xi}$ for each $s \in S$, and if $F(t_1, \ldots, t_n)$ is in A, then t_i is in A and $F_C(t_1, \ldots, t_n) = F(t_1, \ldots, t_n)$.

With this definition, we have to show that for the specification (Σ,ξ) there exists an initial (Σ,ξ)-algebra A which is a canonical term algebra. For doing this, certainly the major step is to show that if A is a canonical Σ-term algebra then A is isomorphic to the initial algebra $T_{\Sigma,\xi}$. For further information the reader may refer to Goguen [1977].

Before ending this section, we have to say that the correctness of specifications involves the *realizability* problem of specifications, which is to decide if an initial algebra for a specification is computable. Equivalently, the problem is to determine if equality of terms in the algebra is decidable.

C. TERM REWRITING SYSTEMS

Term rewriting systems are a very powerful, interesting model of computation. They have been widely used for computation in formula manipulation and theorem proving systems, such as program optimization, program manipulation, and also may be used to represent abstract interpreters for programming languages.

A generalization of these systems consists in considering rewritings on equivalence classes of terms, defined by a set of equations. In this sense, they may be used to define abstract data types. We define a *term rewriting system* R over a set of terms T as a finite set of rewrite rules, each of the form $l(\bar{x}) \rightarrow r(\bar{x})$, where l and r are terms in T containing variables $\bar{x}$. As we realize, the set of rules is composed of a set of directed equations, going from left to right.

To transform a set of equations ξ of an algebra into a term rewriting rule set we may follow the following algorithm as explained in Huet and Oppen[1983].

Let V(N) be the variable set for a term N in Σ, and M=N be an equation in ξ then

1) If V(M) is a subset of V(N), put N → M in T,

2) If V(N) is a subset of V(M), put M → N in T,

3) Otherwise if $\{z_1, \ldots, z_n\}$ are in the intersection of V(M) and V(N), introduce in Σ a new operator H of the appropriate type, and put in T the two rules M → P, N → P with $P = H(z_1, \ldots, z_n)$.

Notice that V(M) may be equal to V(N). If so, we apply first step. The third

step may require Σ' to have sorts not in Σ; It may then be necessary to add extra constants to Σ' for it to be sensible. Also certain Σ-algebras which were models may not be extendible as Σ'-algebras, since the corresponding carriers are empty.

The fundamental difference between equations and term rewriting rules is that equations denote equality (which is symmetric) whereas term rewriting systems treat equations directionally as one-way (left to right) replacements. Before going further, we have to define the notion of a *critical pair* as defined in Dershowitz [1985].

Let $l(\bar{x}) \rightarrow r(\bar{x})$ and $l'(\bar{y}) \rightarrow r'(\bar{y})$ be two rules in T whose variables $\bar{x}$ and $\bar{y}$ have been renamed, if necessary, so they are distinct. We write $l(\bar{x}) = u(v)(\bar{x})$ to indicate that $l(\bar{x})$ contains the (nonvariable) subterm v embedded in the context u. We say that l overlaps (or superposes) l', if $l(\bar{x}) = u(v)(\bar{x})$ and there is (most general) substitution $\bar{\sigma}$ for the variables $\bar{x}$ and $\bar{y}$ such that $v(\bar{\sigma}) = l'(\bar{\sigma})$. In that case, the overlapped term $l(\bar{\sigma})$ can be rewritten to either $r(\bar{\sigma})$ or $u(r')(\bar{\sigma})$. These two possibilties are called a *critical pair*. For example, the two rules F(G(x,y,A)) $\rightarrow$ H(x,y) and G(B,x,y) $\rightarrow$ K(y,x) determine a critical pair <F(K(A,x)),H(B,x)> shown in Figure 4.1.b.

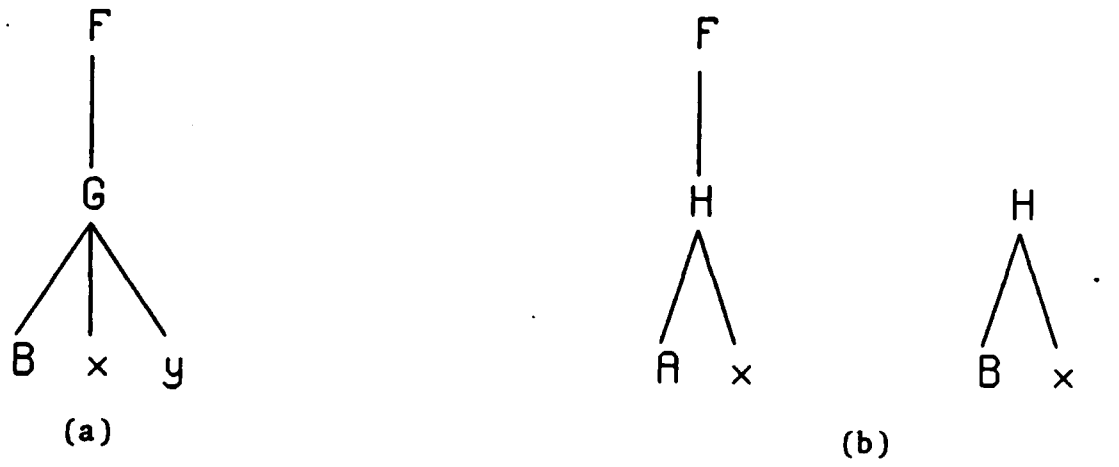


Figure 4.1 Idea of a critical pair.

Five desirable properties of a term rewriting system are as follows (as explained in Dershowitz[1985]):

1) Termination - no infinite derivations are possible,
2) Confluence - each term has at most one normal form,
3) Soundness - terms are only rewritten to equal terms,
4) Completeness - equal terms have the same normal forms,
5) Correctness - all normal forms satisfy given requirements.

A rewrite system T is *canonical* for an equational theory E, if it is terminating, confluent, sound (with respect to E), and complete (with respect to E). Then it can be used to decide whether an equation M=N follows from the axioms in E by checking whether or not unique normal forms of M and N are the same.

Here we work on determination of confluence of a term rewriting system T. The completeness of using rewrite rules to make deductions equationally is specified by the following Church-Rosser property of T.

T is Church-Rosser if and only if, for all M and N, $M =_T N$ if and only if there exists a P such that $M \rightarrow^* P$ and $N \rightarrow^* P$.

We say that P is in *normal form* (relative to T) if and only if there is no P' such that $P \rightarrow P'$, that is no subterm of an instance of P is an instance of a lefthand side of a rule in T. We say P is *T-normal form* of M if $M \rightarrow^* P$ and P is a normal form relative to T. When T is Church-Rosser, the normal form of a term is unique, when it exists. A sufficient condition for the existance of such a unique normal form is the termination of all rewritings.

The confluence property is undecidable for an arbitrary term rewriting system, since a confluence test could be used to decide the equivalence, for instance, of recursive program schemas (Dershowitz [1985]). The decidability of confluence for ground term rewriting systems is open. We say that term σ is *ground* if and only if $V(\sigma(x)) = \phi$. For example, 0+SUCC(0) is a ground term of sort Nat.

We now turn to decidability of confluence for finitely terminating term rewrite systems. The general theorem 3.1 proved in chapter 3 was originally discovered by Newman for rewriting systems (see Newman [1942]).

The next step is to show that local confluence of (finite) term rewriting system is decidable. The following theorem is used as the basis for an algorithm to decide confluence for finitely terminating systems (Knuth and Bendix [1970]):

Theorem 4.1: A terminating rewrite system is confluent if and only if both terms in each of its pairs reduce to the same term.

Combining the Knuth-Bendix theorem and Newman's theorem gives us a decision procedure for the confluence of finitely terminating term systems with a finite number of rules. When such a system T satisfies the critical pair condition it defines a canonical form for the corresponding equational theory $=_T$. We then say that T is a *canonical* term rewriting system.

The next chapter is based on an explanation the algorithm discovered by Knuth and Bendix.

V. AN ALGORITHM FOR TESTING FOR CONFLUENCE

So far, we have discussed the notion of term rewriting systems and their properties. We have said that to show equivalence of any two terms in a term rewrite system is decidable if it is both terminating and locally confluent. To complete this, we have to find a way to test a system for confluence.

The Knuth-Bendix theorem gives a decision procedure for the confluence of terminating rewrite systems. The basic idea is to consider the case where two left-hand sides in a term rewriting system R superpose in a nontrivial way to create an ambiguity of the form $M \rightarrow N_1$, $M \rightarrow N_2$ (then N_1 and N_2 are a critical pair). The system R is nonconfluent if and only if some such pair, N_1 and N_2 reduce to distinct R-normal forms P_1 and P_2 (Huet [1981]).

The Knuth-Bendix completion algorithm attempts to transform a nonconfluent system into a confluent one by adding new rewrite rules, such as $P_1 \rightarrow P_2$. This must be done in such a way that the transformed system is still terminating. Certainly, one round of completion is not sufficient in general, since new ambiguities may have been created. During this completion process, some newly introduced rule may simplify some old rule, either on its left or on its right-hand side. It is essential, both for efficiency and elegance, to keep all rules interreduced as much as possible. But then the question arises as to how the process can be carried out efficiently in an incremental fashion, that is, we do not want to recompute critical pairs between rules that have been previously considered. However, the rules that have been used to resolve these ambiguities may not exist anymore, and so this step must be carefully justified. When a set of equations can be oriented so that the completion process terminates, the resulting term rewriting system defines a decision procedure for the equality problem in the corresponding system.(Huet [1981])

Before presenting the algorithm, we define a *reduction ordering* as a well-founded partial ordering on terms closed by term replacement and substitution. That is, M > N implies that P[M]>P[N] for any term context P[] and σ(M) > σ(N) for any substitution σ. We note that if > is a reduction ordering such that

we have $\lambda > \rho$ for every $\lambda \rightarrow \rho$ in R, then R is obviously terminating. The set of rewrite rules R is *complete* if and only if it is locally confluent (Knuth and Bendix [1970], Theorem 4).

The completion algorithm is as follows (from Huet [1981]):

Initial data: A (finite) set of equations ξ, and a (recursive)
reduction ordering >.
$\xi_0 := \xi$; $R_0 := \Phi$; i := 0; p := 0;
loop
while $\xi_i \neq \Phi$ **do**
Reduce equation: Select equation M=N in ξ_i.
Let M',N' be R_i normal forms of M, N respectively obtained
by applying rules of R_i in any order, until none applies.
If M'=N' **then** $\xi_{i+1} := \xi_i$ - {M=N};
$R_{i+1} := R_i$; i := i+1;
else If (M'>N') **then begin**
λ := M'; ρ := N';
else λ := N'; ρ := M'; **endif**;
Add new rule:Let K be the set of labels k of rules of R_i
whose left-hand side λ_k is reducible by $\lambda \rightarrow \rho$ say to λ_k'.
$\xi_{i+1} := \xi_i$ - {M=N} $\bigcup$ {λ_k' | k: $\lambda_k \rightarrow \rho_k$ is in R_i with k in K};
p:= p+1;
R_{i+1} := {j: $\lambda_j \rightarrow \rho_j'$ |j: $\lambda_j \rightarrow \rho_j$ in R_i with j is not in K} $\bigcup$ {p: $\lambda \rightarrow \rho$}.
The rules coming from R_i are marked or unmarked as they
were in R_i, the new rule $\lambda \rightarrow \rho$ is unmarked;
i:= i+1;
end
else exitloop (failure) **endif**
endwhile;
Compute critical pairs: If all rules in R_i are marked, exitloop
(R_i is confluent and terminating in other terms it is complete.)
Otherwise select an unmarked rule in R_i, say with label k. Let ξ_{i+1}
be the set of all critical pairs computed between rule k and any rule of
rule of R_i of label not greater than k.
Let R_{i+1} be the same as R_i, except that rule k is now marked.
i:= i+1;
endloop.

When given a finite set of equations ξ and a reduction ordering > on terms, the completion algorithm may stop with success, stop with failure or loop forever. When it stops with failure, either the algorithm should be tried again with a different ordering that will order the two terms M', N' which were incomparable; or some new function symbol should be added with a definition in ξ that will

reduce M' or N', or else the method is not applicable. (see Lemma 2 in Huet [1981])

The following examples of the algorithm are taken from Knuth and Bendix [1970], and were programmed for computation in FORTRAN IV on an IBM 7094.

Example 1. *Group theory I.* The first example is the traditional definition of an abstract group. Here we have three operators: A binary operator ., a unary operator -, and a nullary operator e, satisfying the following three axioms.

1. $e \cdot a \rightarrow a$. (Left identity)
2. $a^{-}.a \rightarrow e$. (Inverse for all elements in group)
3. $(a.b).c \rightarrow a.(b.c)$. (Multiplication is associative)

The procedure was first carried out by hand, to see if it would succeed in deriving the identities $a.e = a$, $a^{--} = a$ etc., without making use of any more ingenuity than can normally be expected of a computer's brain. (From now on we will use ab, instead of $a.b$ for simplicity) The success of this hand-computation experiment provided the initial incentive to create a computer program, so that experiments on other axiom systems could be performed.

When the computer program was finally completed, the machine treated the above three axioms as follows: First axioms 1 and 2 were found to be complete, by themselves; but when $\lambda_1 = a^{-}a$ of axiom 2 was superposed on $\mu = ab$ of $\lambda_2 = (ab)c$ of axiom 3, the resulting formula $(a^{-}a)b$ could be reduced in two ways as

$$(a^{-}a)b \rightarrow a^{-}(ab)$$

$$(a^{-}a)b \rightarrow eb \rightarrow b.$$

Therefore a new axiom is added,

4. $a^{-}(ab) \rightarrow b$

Axiom 1 was superposed on the subterm ab of this new axiom, and another axiom resulted:

5. $e^{-}a \rightarrow a$.

The computation continued as follows

6. $a^{--}e \to a$ from 2 and 4,

7. $a^{--}b \to ab$ from 6 and 3,

Now axiom 6 was no longer irreducible and it was replaced by

8. $ae \to a$

Thus, the computer found a proof that e is a right identity; the proof is essentially the following, if reduced to applications of axioms 1, 2, and 3:

$$ae \equiv (ea)e \equiv ((a^{--}a^{-})a)e \equiv$$

$$(a^{--}(a^{-}a))e \equiv (a^{--}e)e \equiv a^{--}(ee) \equiv$$

$$a^{--}e \equiv a^{--}(a^{-}a) \equiv (a^{--}a^{-})a \equiv$$

$$ea \equiv a$$

This ten-step proof is apparently the shortest possible one.

The completion continued further:

9. $e^{-} \to e$ from 2 and 8,
(Now axiom 5 disappeared.)

10. $a^{--} \to a$ from 7 and 8,
(Now axiom 7 disappeared.)

11. $aa^{-} \to e$ from 10 and 2,

12. $a(b(ab)^{-}) \to e$ from 3 and 11,

13. $a(a^{-}b) \to b$ from 11 and 3,

So far, the computation was done almost as a professional mathematician would have performed things. The axioms present at this point were 1, 2, 3, 4, 8, 9, 10, 11, 12, 13; These do not form a complete set, and the ensuing computation reflected the computer's grouping for the right way to complete the set:

14. $(ab)^{-}(a(bc)) \to c$ from 3 and 4,

15. $b(c((bc)^{-}a)) \to a$ from 13 and 3,

16. $b\,(c\,(a\,(b\,(ca))^{-}))\rightarrow e$ from 12 and 3,

17. $a\,(ba)^{-}\rightarrow b^{-}$ from 12 and 4, using 8,

18. $b\,((ab)^{-}c\,)\rightarrow a^{-}c$ from 17 and 3,
(Now axiom 15 disappeared.)

19. $b\,(c\,(a\,(bc))^{-})\rightarrow a$ from 17 and 3,
(Now axiom 16 disappeared.)

20. $(ab)^{-}\rightarrow a^{-}b^{-}$ from 17 and 4.

At this point, axioms 12, 14, 18, and 19 disappeared, and the resulting complete set of axioms included the axioms 1, 2, 3, 4, 8, 9, 10, 11, 13, and 20. A study of those ten reductions shows that they suffice to solve the term problem for free groups with no relations; two terms formed with the operators ., -, and e can be proved equivalent as a consequence of axioms 1, 2, and 3 if and only if they reduce to the same irreducible term, when the above ten reduction are applied in any order. The computation took 30 seconds.

Example 2. *Group theory II.* Suppose we start as in Example 1 but with left identity and left inverse replaced by right identity and right inverse:

1. $ae\rightarrow a$
2. $aa^{-}\rightarrow e$
3. $(ab)c\rightarrow a\,(bc)$

It should be emphasized that the computational procedure is not symmetrical between right and left, due to the nature of the well-ordering, so that this is quite a different problem from Example 1. In this case, axiom 1 combined with axiom 3 generates $a\,(eb)\rightarrow ab$, which has no analog in the system of Example 1. The computer found this system slightly more difficult than the system of Example 1; 24 axioms were generated during the computation, of which 8 did not participate in the final set of reductions. It took 40 seconds.

Example 3. *Inverse property.* Suppose we have only two operators . and - as in the previous examples and suppose that only the single axiom

1. $a^{-}(ab)\rightarrow b$

is given. No associative law, etc., is assumed.

This example can be worked by hand: First we superpose $a^-(ab)$ onto its component (ab), obtaining the term $a^{--}(a^-(ab))$ which can be reduced both to ab and to $a^{--}b$. This gives us a second axiom

2. $a^{--}b \rightarrow ab$

as a consequence of axiom 1. Now $a^-(ab)$ can be superposed onto $a^{--}b$; we obtain the term $a^{--}(a^-b)$ which reduced to b by axiom 1, and to $a(a^-b)$ by axiom 2.

Thus, a third axiom

3. $a(a^-b) \rightarrow b$

is generated. It is interesting (and not well known) that axiom 3 follows from axiom 1 and no other hypothesis. This fact can be used to simplify several proofs which appear in literature, for example in the algebraic structures associated with projective geometry.

A rather tedious further consideration about ten more cases shows that axioms 1, 2, 3 form a complete set. Thus, we can show that $a^{--}b \equiv ab$ is a consequence of axiom 1, but we cannot prove that $a^{--} \equiv a$ without further assumptions.

Some other examples given by Knuth and Bendix explain how a random axiom set can cause the system to degenerate by creating a certain illogical complete set (see Knuth and Bendix [1970] for detail). There are also some weaknesses in the Knuth-Bendix completion procedure. The following example is given to exhibit one of them (example 18 of Knuth and Bendix [1970]).

Example 4. Some unsuccessful experiments. The major restriction of the present system is that it cannot handle systems in which there is a commutative binary operator (for example for an abelian group), where

$a.b \equiv b.a$

Since we have no way of deciding in general how to construe this as a "reduction", the method must be supplemented with additional techniques to cover this case. Presumably an approach could be worked out in which we use two reductions

$\alpha \rightarrow \beta$ and $\beta \rightarrow \alpha$

whenever $\alpha \equiv \beta$ but α is not compatible with β, and to make sure that no infinite looping occurs when reducing terms to a new kind of "irreducible" form. At any rate it is clear that the methods in which this algorithm is involved ought to be extended to such cases, so that rings and other varieties may be studied.

VI. CONCLUSION

In our survey here, an important fact that we have briefly mentioned is the undecidability problem of termination for a set of rewrite rules. Since this property is the main step on the way to proving confluence of a given set of rewrite rules, we have to deal with sufficient conditions for establishing termination. There is some recent work on this problem. One approach finds a cycle if one exists.This is discussed in the paper by J. V. Guttag, D. Kapur, and D. R. Musser [1983]. This procedure is an initial step in this area but not efficient enough. Such problems could be the subject of other thesis research.

Most attempts to apply confluence have been limited because of our inability to solve other, more complex problems, such as termination. Of course, this reflects the ongoing need for broadening our understanding of these problems.

LIST OF REFERENCES

Church A., *The Calculi of Lambda-Conversion*, Princeton U. Press, Princeton New Jersey, 1941.

Curry H.B., and Feys R., *Combinatory Logic*, Vol. 1, North Holland, Amsterdam, 1958.

Navy Postgraduate School Report 52-84-022, *A Formal Method for Specifying Computer Resources in an Implementation Independent Manner*, by Davis D. L., Nov. 1984.

Dershowitz N., *Computing with Rewrite Systems*, to Appear in Information and Control (Revised Jan. 1985).

Goguen j., Thatcher J., and Wagner E.G., *An Initial Algebra Approach to the Specification, Correctness, and Implementation of Abstract Data Types.* Current Trends in Programming Methodology, Vol 4, Ed. Yeh R., Prentice-Hall 1978, pp. 80-149.

Guttag J. V., Kapur D., and Musser D. R., *On Proving Uniform Termination of Rewriting Systems*, SIAM J. Comput. Vol 12 No 1, Feb. 1983, pp. 189-214.

Huet G., *Confluent Reductions: Abstract Properties and Applications to Term RewritingSystems.* J. Assoc. Comput. Mach. 27, 4 (1980), pp. 797-821.

Huet G., *A Complete Proof of Correctness of the Knuth-Bendix Completion Algorithm*, J. Computer and System Sciences, Vol 23, No 1 (1981), pp. 11-21.

Huet G., and Lankford D.S., *On the Uniform Halting Problem for Term Rewriting Systems*, Rapport Laboria 283, IRIA, Mar. 1978.

Huet G., and Oppen D. C., *Equations and Rewrite Rules: A Survey*, in Formal Languages Theory: Perspectives and Open Problems, R. Book, Ed., Academic Press, New York 1980, pp 349-405.

Knuth D. and Bendix P., *Simple Word Problems in Universal Algebras.* "Computational Problems in Abstract Algebra." Ed. Leech J., Pergamon Press, 1970, pp 263-297.

Newman M. H. A., *On Theories with a Combinatorial Definition of Equivalence.* Ann. Math 43, 2 (April 1942), pp. 223-243.

Yurchak J. M., *The Formal Specification of an Abstract Machine: Design and Implementation*, Master's Thesis, Naval Postgraduate School, Monterey, California Dec. 1984.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Superintendent Attn: Library (Code 0142) Naval Postgraduate School Monterey, California 93943-5100	2
3.	Chairman (Code 52) Department of Computer Science Naval Postgraduate School Monterey, California 93943-5100	1
4.	Computer Technology Programs (Code 37) Naval Postgraduate School Monterey, California 93943-5100	1
5.	Daniel Davis (Code 52Vv) Department of Computer Science Naval Postgraduate School Monterey, California 93943-5100	5
6.	Associate Professor Bruce J. MacLennan Code 52ML Department of Computer Science Naval Postgraduate School Monterey, California 93943-5100	1
7.	Ugur Ozkan Hukumet Caddesi, Sunullah Bey Ap. No: 7, D: 4 Kayseri/TURKEY	6
8.	Turk Hava Kuvvetleri Komutanligi Per. Egt. D. Bsk. Bakanliklar/ Ankara/ TURKEY	1
9.	Hava Harp Okulu Komutanligi Kutuphane Yesilyurt/ Istanbul/ TURKEY	2

10. Engin Aytacer 1
Hv. K. K. Per. Bsk.
Sb. Tayin Sb. OBI Ks.
Bakanliklar/ Ankara/ TURKEY

11. Istanbul Teknik Universitesi 1
Kutuphane
Gumussuyu/ Istanbul/ TURKEY

12. Ortadogu Teknik Universitesi 1
Kutuphane
Ankara/ TURKEY

13. Osman Sari 1
Zafer Mahallesi, Kaymakci
Odemis/ Izmir/ TURKEY

END

FILMED

11-85

DTIC