A Study on Amalgamated Programming Systems(融合型プログラミングシステムに関する研究)

氏　　名	グレン・マンスフィールド Glenn Mansfield
授与学位	工学博士
学位授与年月日	昭和63年3月25日
学位授与の根拠法規	学位規則第5条第1項
研究科，専攻の名称	東北大学大学院工学研究科 （博士課程）情報工学専攻
学位論文題目	A Study on Amalgamated Programming Systems （融合型プログラミングシステムに関する研究）
指導教官	東北大学教授　野口　正一
論文審査委員	東北大学教授　野口　正一　　東北大学教授　伊藤　貴康 東北大学教授　佐藤　雅彦　　東北大学助教授　白鳥　則郎

論　文　内　容　要　旨

1. INTRODUCTION

In the realm of descriptive languages, the two major contending programming styles are logic programming and functional programming. *Logic programming* excels in its power of reasoning and inference but lacks the capacity to handle functions and the equality relation in a natural way. On the other hand *Functional programming* has a built-in equality and can handle functions in a natural fashion but lacks the capability of inference. The complementary features of the two styles have prompted researchers to search for a formalism that unifies the two programming methodologies.

AMLOG is an equational logic programming language which amalgamates the two programming paradigms. The model theoretic semantics is based on first order equational definite logic. The proof theoretic semantics is presented in terms of deductive systems. Resolution and evaluation, which form the operational core of logic programming and functional programming,

respectively, are embodied in the operational semantics of the language which uses *goal-superposition* and *reflection.* The fundamental concepts of *Confluence, Church-Rosser, reduction* etc. which are used to define the properties of TRS's, are redefined for an AMLOG program in terms of deductive systems. However, the scope of the completeness results of AMLOG is restricted to the so called confluent class of programs. In this context, we have attempted to extend the operational semantics of AMLOG beyond the confluent class of programs.

2 The amalgamated language AMLOG

In the folowing we have adopted the usual definitions and notations for terms, substitutions , contexts,.. etc.

2.1 Equational Definite Clauses

Definition An *equational definite clause* (or a *conditional equation*) is of the form $L = R \leftarrow L_1 = R_1, \ldots, L_n = R_n$ where, $L, R, L_i, R_i, 0 \leq i \leq k$, are first order terms. []

Denotationally it means- if $L_i = R_i$ is satisfied for all i, $0 \leq i \leq n$, then $L = R$.

2.2 The operational semantics

In an AMLOG program the head of an equational clause is oriented from left to right.

Definitions A *program* is a finite set **P** of *oriented equational definite clauses* of the form $L \rightarrow R \leftarrow L_1 = R_1, \ldots, L_n = R_n$.

A *goal* is a finite sequence of equations of the form $M_1 = N_1, \ldots, M_k = N_k$.

Let $G = (M_1 = N_1, \ldots, M_k = N_k)$ be a goal. The *reflection rule* allows the removal of an equation $M_i = N_i$, $1 \leq i \leq k$, from G where $\theta M_i \equiv \theta N_i$ by a most general unifier θ of the terms M_i and N_i. The resultant goal $G' = \theta(M_1 = N_1, \ldots, M_{i-1} = N_{i-1}, M_{i+1} = N_{i+1}, \ldots, M_k = N_k)$ is called a *reflectant* of G on $M_i = N_i$.

The *superposition rule* yields a *superposant* G' of G by an oriented clause, $L \rightarrow R \leftarrow L_1 = R_1, \ldots, L_n = R_n$, at a non-variable occurrence of G, say at M_i/u_i, iff $\theta(M_i/u_i) \equiv \theta L$ by a mgu θ, and,

G' is the goal- $\theta(M_1 = N_1, .., M_i[u_i \leftarrow R] = N_i, L_1 = R_1, .., L_n = R_n, M_{i+1} = N_{i+1}, \ldots, M_k = N_k)$.

For goals G and G', $G \Rightarrow G'$ indicates that G' is either a reflectant or a superposant of G. We may write $G \Rightarrow^{\theta} G'$ to specify the used unifier θ.

A *computation* from a goal G is a, possibly infinite, sequence of goal derivations - $G_0 (= G) \Rightarrow^{\theta_1} G_1 \Rightarrow^{\theta_2} G_2 \Rightarrow .. G_i \Rightarrow^{\theta_{i+1}} G_{i+1} \Rightarrow \ldots$

An *empty goal*, denoted by ε, is an empty sequence of equations. A computation *successfully terminates* if G_n is an *empty goal* for some $n \geq 0$. []

Theorem 2.1 [*Soundness of the Computation Mechanism*] *Let* **P** *be an equational logic program and* $G = (M = N)$ *a goal. If there is a successfully terminating computation* $G \Rightarrow^{\theta_1} G_1 \Rightarrow .. \Rightarrow^{\theta_n} \varepsilon$, *then the universal closure of the result* $\theta_n .. \theta_1 G$ *is the logical consequence of* **P**. []

The converse of Theorem 2.1 is not true in general. Not all logical consequences of a program **P** have a successfully terminating computation.

To handle the completeness problem a program is viewed as a reduction relation. The relation **red** in the following deductive system *RD* corresponds to this concept.

Definition A deductive system *RD* for *Reduction* in an equational logic program **P** consists of the following inference rules:

(*RD reflection*) $\dfrac{}{M \textbf{ red } M}$ (*RD replacement*) $\dfrac{M \textbf{ red } N}{L[M] \textbf{ red } L[N]}$ (*RD transition*) $\dfrac{M \textbf{ red } L \quad L \textbf{ red } N}{M \textbf{ red } N}$

(*RD convergence*) $\dfrac{M \textbf{ red } L \quad L \textbf{ red } N}{M \downarrow N}$ (*RD modus ponens*) $\dfrac{\theta M_1 \downarrow \theta N_1 .. \theta M_n \downarrow \theta N_n}{\theta M \textbf{ red } \theta N}$

where, $M \leftarrow N \leftarrow M_1 = N_1, \ldots, M_n = N_n \in \mathbf{P}$. []

Definition A program **P** is *confluent* iff for any terms M, N_1 and N_2,

$\mathbf{P} \vdash_{RD} (M \textbf{red} N_1)$, $(M \textbf{red} N_2)$ implies $\mathbf{P} \vdash_{RD} (N_1 \textbf{ red } L)$, $(N_2 \textbf{ red } L)$ for some L. **P** is a *Church-Rosser* iff for any two terms M and N, $\mathbf{P} \vdash_{EQ} (M = N)$ implies $\mathbf{P} \vdash_{RD} (M \textbf{ red } L), (N \textbf{ red } L)$ for some L. []

Proposition 2.1. *A program* **P** *is confluent iff it is a Church-Rosser .* []

Definition A program **P** defines a *reduction relation* R_P: $(M_0 , N) \in R_P$, iff there is a successful computation- $(M_0 = N) \Rightarrow^{\theta_1} (M_1 = N, \Delta_1) \Rightarrow \ldots \Rightarrow^{\theta_{n-1}} (N = N) \Rightarrow^{\theta_n} \varepsilon$ such that $D(\theta_i) \cap \{Var(M_0) \cup Var(N)\} = \phi$ for each i, $0 \leq i \leq n-1$, and N remains unchanged during the computation. A term M is said to be R_P*-normal,* in **P**, if there is no term N except M itself such that $(M , N) \in R_P$. A substitution θ is R_P*-normalized* if $\theta(x)$ is R_P*-normal,* for every variable x. []

Theorem 2.3 $(M , N) \in R_P$ *iff* M **red** N *is provable from* **P** *in RD.* []

Theorem 2.4. [R_P*-normalized Completeness for Confluent Programs*] *Let* P *be a confluent equational logic program,* $G = (M = N)$ *a goal and,* θ *an* R_P*-normal substitution. If* $\theta(M = N)$ *is a logical consequence of the program* P*, then there is a successfully terminating computation from* $M = N$ *with an answer substitution* σ *such that* $\theta = \zeta\sigma$ *for some* ζ. []

3. Extension of the operational semantics

3.1 A More powerful deductive system *CRD*

With a view to extending the scope of the completeness result beyond confluent programs, we identify a subset $\mathbf{P_c}$ of the program **P** as the problem part. This part gives rise to the non-confluence of **P** and, as such, requires more powerful computation rules. To indicate this part we denote a program by $\mathbf{P[P_c]}$.

Definition A deductive system *CRD* for *conditional reduction* in an equational logic program $\mathbf{P[P_c]}$ consists of the following inference rules:

(*CRD reflection*) $\dfrac{}{M \textbf{ cred } M}$

(*CRD replacement*) $\dfrac{M \textbf{ cred } N}{L[M] \textbf{ cred } L[N]}$

(*CRD transition*) $\dfrac{M \textbf{ cred } L \quad L \textbf{ cred } N}{M \textbf{ cred } N}$

(*CRD convergence*) $\dfrac{M \textbf{ cred } L, N \textbf{ cred } L}{M \downarrow_c N}$

(*CRD modus ponens*) $\dfrac{\theta L_1 \downarrow_c \theta R_1 \ldots \theta L_n \downarrow_c \theta R_n}{\theta L \textbf{ cred } \theta R} \quad L \rightarrow R \leftarrow L_1 = R_1, \ldots, L_n = R_n \in \mathbf{P}$

(*CRD c-modus ponens*)

$$\frac{\theta L \text{ cred } M,\ \theta L_1 \downarrow_c \theta R_1 \ \ldots\ \theta L_n \downarrow_c \theta R_n}{M \text{ cred } \theta R} \qquad L \to R \leftarrow L_1 = R_1, \ldots, L_n = R_n \in \mathbf{P_c}\ []$$

Definition A program $\mathbf{P[P_c]}$ is *c-confluent* iff, for any terms M, N_1 and N_2, $\mathbf{P[P_c]} \vdash_{CRD} (M \text{ cred } N_1), (M \text{ cred } N_2)$ implies $\mathbf{P[P_c]} \vdash_{CRD} (N_1 \text{ cred } L), (N_2 \text{credL})$ for some L. It is a *C-Church-Rosser* iff, $\mathbf{P[P_c]} \vdash_{EQ} (M = N)$ implies $\mathbf{P[P_c]} \vdash_{CRD} (M \text{ cred } L), (N \text{cred } L)$ for some L. []

Proposition 3.1. *All programs* $\mathbf{P[P]}$ *are c-confluent.* []

It is clear that confluence implies c-confluence while the converse is not true in general.

Lemma 3.1. *A program* $\mathbf{P[P_c]}$ *is a C-Church-Rosser iff it is c-confluent.* []

3.2 The extended AMLOG

In AMLOG the '=' symbol appearing in equations of the form $M = N$ in the condition part and in goals is considered to be an abbreviation of $M \rightsquigarrow K, N \rightsquigarrow K$ where K is a fresh variable. Thus, the general form of a goal in AMLOG is

$$(M_1 \rightsquigarrow N_1,\ M_2 \rightsquigarrow N_2, \ldots\ M_k \rightsquigarrow N_k).$$

Given a subgoal $M \rightsquigarrow N$ the computation carried out by the AMLOG interpreter involves looking for a substitution θ such that $\theta M \ {}^*\!\!\rightarrow\ \theta N$. The resultant modification in the definition of goal superposition leaves the soundness and completeness results of section 2 unaffected.

3.3 Conditional Computation

Several authors have shown that *completion* may be viewed as a computation mechanism. Superposition is indeed a computation rule based on completion. In fact, *goal superposition*, used in SLOG and in AMLOG, is a form of linear restriction on *completion*. The more powerful computation rule introduced in the following embodies a greater part of the completion mechanism. We will be considering a program $\mathbf{P[P_c]}$, where, $\mathbf{P_c} \subseteq \mathbf{P}$ and c-superposition will be using rules from $\mathbf{P_c}$ only.

Definition Let $G = (M_1 \rightsquigarrow N_1, \ldots, M_k \rightsquigarrow N_k)$ be a goal. The *c-superposition*

rule yields a *c-superposant* G' of G, at some non-variable occurrence of some M_i, say at M_i/u, by an instance θ of a clause $L \to R \leftarrow \Lambda \in \mathbf{P_c}$ where G' is the goal

$\theta(M_1 \rightsquigarrow N_1, \ldots, L \rightsquigarrow M_i/u, \Lambda, M_i[u_i \leftarrow R] \rightsquigarrow N_i, M_{i+1} \rightsquigarrow N_{i+1}, \ldots, M_k \rightsquigarrow N_k)$.

A *c-computation* involves reflections, superpositions and c-superpositions. []

Theorem 3.1. [*Soundness of the C-Computation Mechanism*] *Let* $\mathbf{P[P_c]}$ *be an equational logic program and* $G = (M = N)$ *a goal. If there is a successfully terminating c-computation* $G \Rightarrow^{\theta_1} G_1 \Rightarrow^{\theta_2} \ldots \Rightarrow^{\theta_n} \varepsilon$, *then the universal closure of the result* $\theta_n \ldots \theta_1 G$ *is a logical consequence of* $\mathbf{P}$. []

The reduction relation defined by a program may be extended to a *c-reduction* relation by including c-superposition in the computation mechansim. Further, it may be shown that the c-reduction relation exactly corresponds to the **cred** relation in the deductive system *CRD*. By similar extensions we may obtain the definitions of *CR*$_P$*-normal terms* and *CR*$_P$*-normal substitutions*.

It is clear that when $\mathbf{P_c} = \phi$, all the definitions are reduced to the corresponding forms of section 2.

Corollary 3.1. *Let* $\mathbf{P[P_c]}$ *be a c-confluent program. If the equation* $M = N$ *is a logical consequence of* $\mathbf{P}$, *then there is a successfully terminating c-computation from the goal* $M = N$ *with the empty answer substitution.*

Corollary 3.2. *Let* $\mathbf{P[P]}$ *be an equational logic program. If the equation* $M = N$ *is a logical consequence of* $\mathbf{P}$ *then there is a successfully terminating c-computation from the goal* $M = N$ *with an empty answer substitution.* []

3.4 The completeness of c-computations

Theorem 3.2 [*CR*$_P$*-normalized Completeness of C-computations*] *Let* $\mathbf{P[P_c]}$ *be a C-confluent equational logic program,* $G = (M = N)$ *a goal, and* θ *a CR*$_P$*-normal substitution. If* $\theta(M = N)$ *is the logical consequence of the program* $\mathbf{P[P_c]}$, *then there is a successfully terminating c-computation from* $M = N$ *with an answer substitution* σ *such that* $\theta = \zeta\sigma$ *for some* ζ. []

4 Practical issues

The completenss result for the extended AMLOG interpreter, stated in

section 3.4, guarantees that there is a successfully terminating computation from a goal, a CRp-normal instance of which is a logical conseequence of the program. However, the c-superposition rule is very difficult for practical implementation purposes due to the following reasons:

(1) Given a clause for c-superposition, there is no method for mechanically computing the appropriate instance of the clause. The following example illustrates the case.

example 4.3.

Program : $\{H(F(*x, *x)) \rightarrow A. ; F(*x, G(*x)) \rightarrow B.; C \rightarrow G(C).\}$

This program is c-confluent. And indeed there exists a successfull c-computation for the goal

$$H(B) \rightsquigarrow A \Rightarrow A \rightsquigarrow A, F(*x, *x) \rightsquigarrow B \Rightarrow F(*x, *x) \rightsquigarrow B \Rightarrow F(C, C) \rightsquigarrow B \Rightarrow$$
$$F(C, g(C)) \rightsquigarrow B \Rightarrow \quad B \rightsquigarrow B \Rightarrow \quad Success$$

However the substitution C/x used in the third step cannot be found mechanically. []

(2) There is no criteria whatsoever for mechanically selecting a clause for c-superposition.

4.1 Cp-superposition

Definition Let $G = (M_1 \rightsquigarrow N_1, \ldots M_k \rightsquigarrow N_k)$ be a goal. The *c_p-superposition rule* yields a *c_p-superposant* G' of G, at some non-variable occurrence of some M_i, say at M_i/u. by a program clause $L \rightarrow R \leftarrow \Lambda \in \mathbf{P_c}$ where G' is the goal

$$(M_1 \rightsquigarrow N_1, .., \Lambda, M_i[u_i \leftarrow R] \rightsquigarrow N_i, M_{i+1} \rightsquigarrow N_{i+1}, \ldots, M_k \rightsquigarrow N_k). \quad []$$

Theorem 4.1 *If there is a successful c-computation from a goal G such that the required instances of the clauses used in the c-superpositions are CRp-normal then , there is a successful C_p-computation from G.* []

4.2 A restricted deductive system C_sRD

To address the problem of selecting the clause we consider the following restricted form of the *c-modus ponens* rule in the deductive system *CRD.*

(C_sRD *c-modus ponens*)

$$\frac{\theta l_1 \ \mathbf{c_s red}\ m_1,..,\theta l_k \ \mathbf{c_s red}\ m_k\ ;\ \theta L_1 \downarrow_{c_s} \theta R_1 \ldots \theta L_n \downarrow_{c_s} \theta R_n}{f(\, m_1, .. , m_k) \ \mathbf{c_s red}\ \theta R \qquad f(\, l_1, .. , l_k) \ \rightarrow R \leftarrow L_1 = R_1, \ldots , L_n = R_n \in \mathbf{P_c}}\ []$$

4.3 A restricted form of c-superposition

It is quite clear that an attempt is being made to restrict the choice of candidates for c-superposition by making it mandatory that the root function symbol of the head of the program clause and the subterm must be identical. To prevent repeated computations on the same subterm by the same clause, the identical root function symbols are stripped in the c_s-superposition.

Definition. Given a goal **G:** $M \rightsquigarrow N$ and a program clause $\mathbf{C} : f(l_1, .. \, l_k) \rightarrow R \leftarrow \Lambda \in \mathbf{P_c}$ such that for some $u \in Ocr(M)$, M/u has the root function symbol f *i.e.* $M/u \equiv f(m_1, .. , m_k)$,the c_s-*superposant* **G'**, obtained by c_s-*superposing* an instance θ of **C** at M/u, is **G':** $l_1 \rightsquigarrow m_1, .. , l_k \rightsquigarrow m_k, \Lambda , M[u \leftarrow R] \rightsquigarrow N.$ []

4.4 Completeness of c_s-computations

Theorem 4.2 [C_sR_P-*normalized Completeness of* C_s-*Computations*]

Let $\mathbf{P[P_c]}$ *be a* c_s-*confluent equational logic program,* $G = (M = N)$ *a goal, and* θ *a* C_sR_P-*normal substitution. If* $\theta(M = N)$ *is the logical consequence of the program* $\mathbf{P[P_c]}$, *then there is a successfully terminating* c_s-*computation from* $M = N$ *with an answer substitution* σ *such that* $\theta = \zeta\sigma$ *for some* ζ.

4.5 Sufficient conditions for c_s-confluence

We examine the case of unconditional equational programs. We conjecture that the results may be extrapolated to the conditional case. Let **P** be an unconditional equational logic program containing clauses of the form $\{ l_i \rightarrow r_i \}$. We will consider programs which do not overlap at the root i.e. $\sigma\, l_i \neq \sigma\, l_j$ for any i, j, σ.

Definition Given an equational logic program **P**, suppose that the j^{th} clause overlaps the i^{th} clause at the u^{th} occurrence i.e. $\sigma_i\, l_i/u \equiv \sigma_j\, l_j$, where $l_i \rightarrow r_i$, $l_j \rightarrow r_j \in \mathbf{P}$ and $u \neq ()$. The *critical pair* generated by the overlap is given by

$CP: < \sigma_i l_i[u \leftarrow \sigma_j r_j], \sigma_i r_i >$.

The *consequent rule* generated by the critical pair $CP: <p,q>$ is $R_{CP}: p \rightarrow q$. The *expanded program* $\mathbf{P_1}$ is given by $\mathbf{P} \cup \mathbf{R_1}$. Where, $\mathbf{R_1}$ is the set of consequent rules generated by the overlap of the clauses in **P**. []

Proposition 4.1 Let **P** be a (left and right) linear overlapping program and $\mathbf{P_1} = \mathbf{P} \cup \mathbf{R_1}$ the corresponding *expanded* program. If the clauses in **P** do not overlap with the clauses in $\mathbf{R_1}$ and the clauses in $\mathbf{R_1}$ do not overlap among themselves then $\mathbf{P_1}$ is confluent.

4.6 An Implementation technique for c_s-superposition

Given a program **P**, let $\mathbf{P_c}$ be the set of all the overlapping clauses in the program. Every clause $F(t) \rightarrow R \leftarrow \Lambda \in \mathbf{P_c}$,which is overlapped at t, is transformed to $F(*x) \rightarrow R \leftarrow *x = t, \Lambda$.

Proposition 4.2 C_s-superposition by a clause belonging to $\mathbf{P_c}$ is equivalent to superposition by the corresponding transformed clause. []

5 Conclusion

An amalgamated equational logic programming language AMLOG is presented. In the first stage, the operational semantics given in terms of two computation rules- reflection and superposition is shown to be complete for confluent programs and normal answer substitutions. In the second stage, the operational semantics has been extended by enriching the computation rules with the c-superposition rule. A restricted form, c_s-superposition, of the newly introduced computation rule is found more tractable and readily implementable. The scope of the completeness result covers c_s-confluent programs.

An interpreter of the language has been implemented in C on the SUN work-stations (TOSHIBA Co. AS3000) at Tohoku University.

審 査 結 果 の 要 旨

情報化社会の急速な発展に伴い，従来までの数値計算や事務処理とは異なる知的な処理を行うシステムの開発が重要な問題となってきた。このためには，知識を活用し推論等高度な処理を行う情報処理システムの研究が不可欠であるが，特にこの処理システムのための新しいプログラミング言語の研究，開発が重要となる。本論文の著者は，これまでの関数型及び論理型プログラミングの考え方に着目し，両者の特長を融合したプログラミング言語を開発し，その性質を数理論理学の立場から明らかにした。本論文はこれらの成果をまとめたもので，全編6章より成る。

第1章は，序論である。

第2章では，本論文の基礎となる諸概念，諸定義を述べている。特に，等号確定節に対する健全かつ完全な演繹体系を明らかにし，以下で展開される融合型プログラミング言語の論理的基礎を与えている。

第3章では，従来の関数型及び論理型プログラミングパラダイムを統合した，知識処理に適した融合型プログラミング言語を提案し言語の性質について詳細に考察している。まず初めに，この言語が関数型言語と論理型言語を自然に融合したものであることを示した後，リダクションに基づく計算機構を与え，その健全性と完全性の問題を議論している。この結果，プログラムが合流性，即ち等価的にChurch－Rosser性を満たせば，完全性が保証されることを示している。この結果は，この分野での従来の成果を包含するものであり，有用な知見である。

第4章では，第3章のリダクションシステムを拡張し，プログラムの合流性が成り立たない場合にも計算機構の完全性を保証する強力な計算機構について考察している。ここで提案する計算機構を用いれば，解がある場合にはこれを求める手続きが必ず存在することを示している。また，処理系の立場から，探索空間を縮小させ効率よく処理を行うための方法についても言及している。

第5章では，第4章までの結果を用いて，効率よい処理系を作成するための実現方法について述べている。

第6章は結論である。

以上要するに本論文は，今後の情報工学の重要な研究課題である知識情報処理システムの基礎となる，融合型プログラミング言語について研究を行ったもので，情報工学の発展に寄与するところが少なくない。

よって，本論文は工学博士の学位論文として合格と認める。