Theoretical Basis of Language System with State Constraints

Susumu Yamasaki

HCI Group, Department of Computer Science, Okayama University, Tsushima-Naka, Okayama, Japan

Keywords: Programming Language, Algebraic Expression, State Constraint System.

Abstract: This paper presents theoretical basis of a language system whose program is described as algebraic expressions and implemented as abstract state machine. The behaviors of the described expressions may be captured (with their models) as causing sequences for state transitions, where composition and alternation for state transitions are mechanized in algebraic structure. Monitoring facilities to the language system may be described with state concepts, as well. With respect to intuitionistic logic and logical program containing negatives, Heyting algebra expressions are taken rather than already established nonmonotonic reasoning programs with negations, where 3-valued domain may be of use for the undefined to be allowable such that positives and negatives may be consistently evaluated, instead of rigid 2-valued settlements. We may have a standard form of Heyting algebra expressions in accordance to logical and AI programming, where the expressions are constrained with states. The states may be regarded as environmental conditions or objects as in object-oriented programming. As regards 3-valued models of given expressions, monotonic mapping cannot be in general associated with, but some ways are presented to approximate fixed points of a mapping for the given expression. Then the formal description of programs may be given with reference to state transitions, which is thought of as proposing a language system structure.

1 INTRODUCTION

As resources, programming languages are very significant in complex information systems. Whether they are viewed from abstract or concretized points, they must also involve communication functions not only with other programming environments but also with human to the network. In this paper, we have a consistent theory for programming method of a language system based on abstract state machine, which program implementation is made with and its algebraic basis is specialized to, possibly for application to complex information systems. The abstract notion of states is assumed, from resource environments of programmable and communicative capabilities. The backgrounds of this paper are common to those which the paper on modal μ-calculus extension (S. Yamasaki, 2020) refers to, such that this paper may aim at the programming language system primarily based on abstract state machine.

As backgrounds of operational ways to capture programs with, we have seen on abstract state machine structures and actions that: (i) by composed actions as programs in dynamic logic, acting and sensing failures, and actions to generate and execute plans are discussed as advanced works (Spalazzi and Traverso, 2000). (ii) the action is formulated as a key role in strategic reasoning of abstract state machine, applicable to AI systems. (iii) actions are also captured in logical systems, as in the papers (Giordano et al., 2000; Hanks and McDermott, 1987). (iv) the procedural action is expressed by denotational approach in the book (Mosses, 1992), while the procedural method is essentially operational, that is, for programs to be implementable. (v) the actions may be reviewed, with functional programs (Bertolissi et al., 2006). (vi) algebraic systems of abstract state machine are discussed, in the literature (Droste et al., 2009; Reps et al., 2005). (vii) regarding structure of streams possibly caused by abstract state machine, there is the note (Rutten, 2001).

As programming systems applied to communications, we pay attention to the frameworks:

(a) AI developments are presented on the basis of logics with knowledge (Reiter, 2001). Based on beliefs and intentions, modal operations are used to represent mental states (Dragoni et al., 1985).

(b) Regarding communication technology, argumentation is sometimes popular, in terms of non-classical negation, and abstract attack and defense are regarded as elements of the argumentation concepts rather than communications by algebraic...
processes. From model theoretic views, the argumentation may be expressed by means of 3-valued logic to the semantics for defeasible reasonings to implement argumentation (Governatori et al., 2004).

(c) Mobile ambients (Cardelli and Gordon, 2000; Merro and Nardelli, 2005) have been studied from the views to make communication environments clearly described, as well as process algebras (Hennessy and Milner, 1985; Kucera and Esparza, 2003; Park, 1970).

(d) The modal mu-calculus, related to abstract state machine, contains a fixed point notation by which actions and communications are satisfied with given conditions. The modal logic with fixed point operator is refined in the paper (Venema, 2006). The papers (Dam and Gurov, 2002; Kozen, 1983) are classical enough to formulate the proof systems with fixed points and their approximations.

With reference to such works, this paper deals with theoretical basis of:

1. a language system based on algebraic expressions whose models may cause state transitions, for a programming system with state constraint designs to implement and to describe complex AI facilities and reasonings compactly, and

2. Heyting algebra expressions, to be able to involve a treatment of negation (for expressiveness as a program), different from the one of default negation often used in AI researches.

For the construction of complex information systems by means of programming in this language system, some monitoring functions are to be facilitated, where the descriptions of functions may be made with abstract states constraining programming. Communicational and behavior facilities are involved in abstract monitoring. In this paper, communicational and behavioral means are regarded as abstract functions. As well as functional aspects from descriptions on abstract state machine, algebraic expressions as designs (programs) contain logical aspects.

Compared with default or defeasible logic in AI programming,

(i) defeasibility is beforehand assumed in the given rules, to be more complex, but

(ii) the plain program consisting of rules by Heyting algebra expressions is simpler without treating ambiguity of rules containing default negation.

The paper is organized as follows. Section 2 is concerned with Heyting algebra expressions, and with the standardization of expressions, closely related to logical programming with default negation and to state constraint programming. Section 3 discusses the 3-valued models of algebraic expressions. Section 4 presents a language system with monitor facilities, whose program can be interpreted in terms of models for Heyting algebra expressions and by structures for the implementation aspects as well as for monitoring. Concluding remarks and references to related topics on knowledge are mentioned in Section 5.

2 UNIVERSAL EXPRESSIONS FOR LOGICAL DESIGNS

Regarding operations, processing, or program implementation, we need logical descriptions of designs for functions with reference to complex systems and their dynamic workings. Logical programming has been studied for universal tools of design, where first-order predicate calculus is primary. If an infinite set of proposition letters is allowable, the propositional logic may simulate the first-order calculus with Herbrand base. Heyting algebra may be a basis for propositional logic to be as universal as the first-order calculus, where Heyting algebra is more flexible than Boolean algebra. In these senses, we here examine Heyting algebra for expressions concerning descriptions of design ideas. Because of the treatment for negatives of Heyting algebra expressions, some theoretical basis is motivated to be made clear, for us to formulate a language system containing algebraic expressions as programs of a language.

2.1 Algebraic Expressions

Heyting algebra (HA) \((A, \vee, \wedge, \bot, \top)\) equipped with the partial order \(\sqsubseteq\) and an implication \(\Rightarrow\) is assumed as follows:

(i) \(\bot\) and \(\top\) are the least and the greatest elements of the algebra (set) \(A\), respectively, with respect to the partial order \(\sqsubseteq\).

(ii) the join \(\vee\) and meet \(\wedge\) are defined for any two elements of \(A\).

(iii) as regards the implication \(\Rightarrow\),

\[ c \sqsubseteq (a \Rightarrow b) \iff a \wedge c \sqsubseteq b. \]

The element \(a \Rightarrow \bot\) is denoted as “not a” for \(a \in A\), where not \(\bot = \top\) and not \(\top = \bot\).

The expression \(F\) (over the underlined set \(A\) of the algebra), which is here paid attention to, is of the form:

\[ \wedge_{j \in \alpha} (l_{1j} \wedge \ldots \wedge l_{nj} \Rightarrow l_j) \]
where $l_j^i$ denotes $a$ or $a \Rightarrow \perp$ for $a \in A$. The expression $F$ is regarded as modelling some programing.

Related Programming:

A “logic program” with respect to its Herbrand base may be regarded as containing the predicate $pr$ (with or without “$\sim$” as a procedure) preceded by the conjunction of:

$$pr_1, \ldots, pr_m \text{ (as a procedural body)}$$

for $pr_1, \ldots, pr_m$ (predicates or their negations).

**Example 1.** Assume the following propositional formula $F$:

$$(\text{bird} \land \sim \text{abnormal} \Rightarrow \text{fly}) \land \left(\Rightarrow \text{bird}\right) \land \left(\text{fly} \Rightarrow \sim \text{abnormal}\right) \land \left(\sim \text{fly} \Rightarrow \text{observed}\right),$$

where, for the propositions bird, abnormal, fly, observed,

(a) $\land$ (meet) is used as and,

(b) $\sim$ (default negation) is assigned in place of not,

(c) $\Rightarrow$ is used for logical implication; and

(d) the parentheses are used to have priorities of (logical) connectives.

Then it might be expected that (a) with the assertion of bird and by default negation of abnormal, fly may be implied, and (b) the bird is not observed as default, however, it may not be, because of the part (of some ambiguity with other parts), fly $\Rightarrow \sim$ abnormal, such that neither fly nor default negation of abnormal can be implied so that it may be unknown for bird not to be observed.

This example presents a common view on logic programs with answer set programming (Osorio et al., 2004) or on defeasible logic (Governatori et al., 2004), where the negation may be taken as default negation (denoting $a \subseteq \perp$, for $a \in A$, such that

$$E_{x_2} \subseteq E_{x_1}.$$ 

By the method which is as below shown with proof, we may have such a standardization to transform a given expression $E_{x_1}$ to $E_{x_2}$. If there is some model of $E_{x_2}$ in 3-valued domain, then it may be also the model of $E_{x_1}$; In this sense, the expression $E_{x_2}$ is worthwhile being obtained, as a standard form.

**Transformation of Expressions:**

In what follows, the cases (of what the given expression exp1 is) are presented such that transformations may be available, where $X$, $Y$, and $Z$ stand for some (sub)expressions. There is a relation: $exp_2 \subseteq exp_1$, between a replacing expression $exp_2$ and a replaced one $exp_1$ in each transformation of the items (1) – (8).

(1) For the right side of the primary operation $\Rightarrow$, the items (1) – (4) are applied:

(a) $X \Rightarrow (Y \Rightarrow Z)$ to be replaced by: $X \Rightarrow (Y \land Z)$.

(b) $X \Rightarrow \neg Y$ to be replaced by: $\neg Y \land (X \Rightarrow Z)$.

(c) $X \Rightarrow (Y \land Z)$ to be replaced by:

$$X \Rightarrow (Y \land (X \Rightarrow Z)).$$

(4) $X \Rightarrow \neg Y$ to be replaced by:

$$\neg Y \land (X \Rightarrow \neg y)$$
for an arbitrarily and newly chosen element \( y \in A \) (or variable \( y \) over \( A \)).

(II) For the left side of the primary operation \( \Rightarrow \), the items (5) – (8) are applied.

(5) \( (X \Rightarrow Y) \Rightarrow Z \) to be replaced by:

\[
(X \Rightarrow Y) \wedge (\top \Rightarrow Z)
\]

for the top element \( \top \).

(6) \( (X \not\land Y) \Rightarrow Z \) to be replaced by:

\[
\text{not} (not X \not\land Y) \Rightarrow Z.
\]

(7) \( (X \not\land Y) \Rightarrow Z \) to be replaced by:

\[
((x \not\land y) \Rightarrow Z) \wedge (X \Rightarrow x) \wedge (Y \Rightarrow y)
\]

for arbitrarily and newly chosen \( x, y \in A \).

(8) \( \text{not} X \Rightarrow Y \) to be replaced by:

\[
(x \Rightarrow X) \wedge (\text{not} x \Rightarrow Y)
\]

for an arbitrarily chosen \( x \in A \).

**Proposition 1.** Given an expression \( E_{x_1} \), an expression \( E_{x_2} \) of standard form is derived by the Transformation of Expressions such that \( E_{x_2} \subseteq E_{x_1} \).

**Proof.** By the recursive applications of (I) (containing the items (1) – (4)), reducing the right side of the implication \( \Rightarrow \) to the simple form (which is represented as \( a \) or not \( a \) for \( a \in A \)), we may have an expression \( \exp \) of the form

\[
\bigwedge_j (X_j \Rightarrow x_j),
\]

where each \( X_j \) is a (sub)expression and each \( x_j \) is \( a \) or not \( a \) (for \( a \in A \)). By the recursive applications of (II) (containing the items (5) – (8)) with the (I) case from the item (5) or (8)) to each \( X_j \Rightarrow x_j \) of \( \exp \), reducing the left side of the implication \( \Rightarrow \), we may have an expression \( E_{x_2} \) of standard form. This comes from the reason for structure of expression form, why the application of (I) case from the item (5) or (8) is available in (II) case, for the (sub)form not to be the simple but to be of less length. \( \square \)

The model of the expression \( F \) of standard form is discussed in the next section.

### 3  3-VALUED MODEL OF EXPRESSIONS

We have known applicable fixed point as model of the expression \( F \), over the 3-valued domain \( \{0, 1/2, 1\} \).

With the set \( A \) for an expression \( F \) of standard form (where \( p \Rightarrow \bot \) is denoted by not \( p \)), a mapping

\[
\Psi_F : 2^A \times 2^A \rightarrow 2^A \times 2^A,
\]

has been noted with order of componentwise subset inclusion:

\[
I_2 \text{ contains } p \text{ such that } p \text{ is obtained by:}
\]

\[
\exists(p_1 \land \ldots \land p_n \not\land \text{not} p_{i+1} \land \ldots \land \text{not} p_f \Rightarrow p) \text{ in } F.
\]

\[
\forall p_k \in I_1 (1 \leq k \leq i), \forall p_{k'} \in J_1 (i + 1 \leq k' \leq j),
\]

\( J_2 \) contains \( q \) such that \( q \) is obtained by:

\[
\forall(q_1 \land \ldots \land q_i, \not\land q_{i+1}, \ldots, \not\land q_f \Rightarrow q) \text{ in } F.
\]

\[
\exists q_k (1 \leq k \leq i). q_k \in J_1, \text{ or}
\]

\[
\exists q_{k'} (i + 1 \leq k' \leq j). q_{k'} \not\in J_1, \text{ or}
\]

\[
\exists(q_1 \land \ldots \land q_i, \not\land q_{i+1}, \ldots, \not\land q_f \Rightarrow \not\land q) \text{ in } F.
\]

\[
\forall q_k (1 \leq k \leq i). q_k \not\in J_1, \text{ and}
\]

\[
\forall q_{k'} (i + 1 \leq k' \leq j). q_{k'} \not\in J_1.
\]

If \( \Psi_F(I, J) \subseteq_c (I, J) \) (with the componentwise set inclusion \( \subseteq_c \)) and \( I \cap J = \emptyset \), then \( (I, J) \) can be a model of \( F \).

Because of the negation interpretation, some approximations cannot be taken, even with monotonic mappings (the least fixed points of which are worthwhile referring to), extended from those already established by A. van Gelder et al. (1991) and by M. Fitting (1985).

In what follows, we suppose the set \( A \) (for HA expressions) and the expression \( F \) of standard form.

We now have a procedure with respect to construction of some model \( (I, J) \), where \( \Psi_F(I, J) \subseteq_c (I, J) \), and an adjusting (as another procedure) to be sound to the constructed model. Making use of “negation as failure” rule with its variants, we take the notations as follows:

(1) With a given expression \( F \) of standard form, query of the sequence “\( X? \)” is assumed, where \( X = y_1; \ldots; y_n \) with \( y_i \) being \( a \) or not \( a \) for \( a \in A \) and with the concatenation operation “;” (which is treated as \( \land \)), where in case of “\( n = 0 \)”, \( X \) is null (the empty query).

A sequence query may be denoted as \( y; X? \) (with \( y \) being \( b \) or not \( b \) for \( b \in A \), and with \( X \) a sequence query), or \( Y; X? \) with \( Y \) and \( X \) queries.

(2) The notations

\[
[X? \text{ succ}], [X? \text{ fail}], \text{ and } [X? \text{ no fail}]
\]

stand for the returns of the Procedure to say that (i) the query \( X? \) is a success, (ii) the query \( X? \) is a failure, and (iii) the query \( X? \) is not a failure, respectively.
Procedure (for Model of an Expression):

The routines are inductively defined with queries to be a success, a failure, and not a failure.

1. null? suc.

2. [x;X? suc], if \( x = a \) (for \( a \in A \)) and there is \( Y \Rightarrow x \in F \) such that \([Y;X? suc]\).

3. (a) \([x;X? suc] \) if \([a? f ail] \) for \( x = not a \), and \([X? suc] \).
   (b) \([x;X? suc] \) if \([a? f ail] \) for \( x = not a \) where there is \( Y \Rightarrow x \in F \) with \([Y;null f ail] \), and \([X? suc] \).

4. \([x;X? f ail] \), if there is no part with \( x = a \in A \) being the right side of “⇒” in \( F \), or \([X? f ail] \).

5. (a) \([x;X? f ail] \) if \( x = a \) (a \( a \in A \)) such that \([Y? f ail] \) for any \( Y \) (where \( Y \Rightarrow x \in F \)), or \([X? f ail] \).
   (b) \([x;X? f ail] \) if \( x = a \) such that \([Y? no f ail] \) for some \( Y \) where \( Y \Rightarrow not a \in F \), or \([X? f ail] \).

Example 2. Assume the algebraic expression in correspondence with the propositional formula \( F \) in Example 1. Let the expression be referred to by \( F \), with the same name of the formula.

\[
(bird \land not abnormal \Rightarrow fly) \land \\
(I \Rightarrow bird) \land \\
(fly \Rightarrow not abnormal) \land \\
(fly \Rightarrow not observed) \land \\
(not fly \Rightarrow observed).
\]

Following the Procedure, we can see that:

\[
[bird? suc] \text{ and } [fly? no f ail] \rightarrow \\
[abnormal? f ail], [fly? suc], [observed? f ail], \text{or} \\
[bird? suc], \text{ and } [abnormal? no f ail] \rightarrow \\
[fly f ail], [abnormal? suc], [observed? suc].
\]

where “⇒” means inferences for the successes and failures of queries.

Now let the pair \((I,J)\) be defined by:

\( I = \{ a \in A \mid [a? suc] \} \), \( J = \{ b \in A \mid [b? f ail] \} \).

The Procedure works to construct a pair \((I,J)\) so that (a) if \([a? suc] \) then \( a \in I \), and (b) if \([a? f ail] \) then \( a \in J \). For such a constructed pair \((I,J)\) of the expression \( F \) over the set \( A \), we would like to see that \((I,J)\) is just a model in the following sense.

Proposition 2. Assume the pair \((I,J)\) constructed by the Procedure for a HA expression \( F \) of standard form such that

\[
I = \{ a \in A \mid [a? suc] \} , \\
J = \{ b \in A \mid [b? f ail] \} .
\]

If \( I \cap J = \emptyset \), then the pair \((I,J)\) is a model of \( F \).

Proof. The reason comes from the case examinations by induction on constructions of the Procedure, for the effects of the mapping \( \Psi_F \) for \( \Psi_F(I,J) \subseteq (I,J) \): We show on the case of \( I \cap J = \emptyset \) that if we let \( \Psi_F(I,J) = (I_0,J_0) \) then \( (I_0,J_0) \subseteq (I,J) \).

1. Let \( a \in I_0 \). (i) It follows with respect to the mapping of \( \Psi_F \) that there may be some \( Y \Rightarrow a \) in \( F \) such that (i) \( Y = null \), or (ii) with \( b \in I \) for any \( b \in Y \), and with \( c \in J \) for any not \( c \in Y \).

   In case of (i), \([a? suc] \) and \( a \in I \). In case of (ii), by the Procedure construction, (a) \([b? suc] \) for \( b \in I \) and (b) \([not c? suc] \) for \( c \in J \), respectively, in \( F \) such that \([a? suc] \). Note by the Procedure to simulate the mapping \( \Psi_F \) that \([not c? suc] \) comes from \([c? f ail] \) for \( c \in J \), or from \( J \subseteq Z \Rightarrow not c \) and \([Z? no f ail] \) for \( c \in J \) Thus \( a \in I \). Finally, if \( a \in I_0 \) then \( a \in I \).

2. Let \( a \in J_0 \). Both the cases (i) and (ii) (as below) comes from that \( (I_0,J_0) \) can be obtained with the application of \( \Psi_F \) to \((I,J)\), and derives that \( a \in J \):

   (i) If there is no part \( Y \Rightarrow a \) in \( F \) so that the mapping \( \Psi_F \) may assume \( a \in J_0 \). It follows by the Procedure that \([a? f ail] \) That is, \( a \in J \).

   (ii) (a) If \( \forall Y \), \( Y \Rightarrow a \) entails that there is some \( b \in Y \) such that \( b \in J \), or some not \( c \in Y \) such that \( c \not\in J \), then \( a \in J_0 \). By the Procedure, \([b? f ail] \) \( b \in J \) and \([c? no f ail] \) \( c \not\in J \) causes \([a? f ail] \). Thus \( a \in J \). (b) If \( \forall Y \), \( Y \Rightarrow not a \) in \( Y \), where any \( b \not\in J \) or any not \( c \) for \( c \in J \) in \( Y \), then \( a \in J_0 \). By the Procedure, if \( b \not\in J \) then \([b? no f ail] \), and if \( c \in J \) then \([c? f ail] \). It follows that \([Y? no f ail] \) such that \([a? f ail] \). Thus \( a \in J \). 

The procedure contains rules regarded as variants of negation as failure:

(i) \([not a? suc] \), if \([a? f ail] \).

(ii) \([not a? f ail] \), if \([a? no f ail] \).

To make the procedure free from the rule \([a? no f ail] \) or \([Y? no f ail] \) (for a query sequence \( Y \)), we may make the procedure adjusted into a simpler version with assumption that \([a? no f ail] \) is reasoned by \([a? suc] \). Therefore classical “negation as failure” may be adopted:

(a) a query \([not a? suc] \) is a success, if \([a? f ail] \), and
(b) a query \([not a? f ail] \) is a failure, if \([a? suc] \).
4 LANGUAGE SYSTEM

In this section, we have a formal aspect of a language for its program to be realized as abstract state machine and as a semiring to explain an algebraic structure for implementation of the program. The state constraint system may contain query processing in database, algebraic expression of strategies, and program implementation assigned to states (likely reflecting environments, situations and designing concepts as classes).

<table>
<thead>
<tr>
<th>States</th>
<th>s₁</th>
<th>s₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processing</td>
<td>query₁</td>
<td>query₂</td>
</tr>
<tr>
<td>Strategy</td>
<td>expression₁</td>
<td>expression₂</td>
</tr>
<tr>
<td>Implementation</td>
<td>program₁</td>
<td>program₂</td>
</tr>
</tbody>
</table>

Having an outlook on the state constraint system, queries, algebraic expressions, and programs are regarded as synonymous for database, strategic plans and designs, which are constrained at states, that is, are compiled to the states. The formal description of such a state constrained framework as a language system is now given theoretically.

As regards the state constraint system, the expressions for design may be considered as resources at states, where the design idea based on some allowable assumptions for inferences related to implementations like query processing, strategic reasoning and program execution. In this sense, not only technical views but also social views may be contained in the concepts of state constraints. Therefore the state constraint language system should be studied for compact manners free from complex design methods.

4.1 State Constraint Language

The expression as in Example 1 may be separated into two expressions with states (for constraints), where the default negation is replaced by the negation not in Heyting algebra.

Example 3. We now have the next forms from the expression in Example 2, by separating F into 2 parts constrained by 2 states, respectively:

\[
\begin{align*}
\text{s₁} : & \quad (\text{bird} \land \text{not abnormal} \Rightarrow \text{fly}) \land (\top \Rightarrow \text{bird}) \\
& \land (\text{fly} \Rightarrow \text{not abnormal}) \triangleright s₂ \\
\text{s₂} : & \quad (\text{fly} \Rightarrow \text{not observed}) \land \\
& (\text{not fly} \Rightarrow \text{observed}) \triangleright s₂
\end{align*}
\]

(i) At the state s₁, there may be 2 models:

\[
\begin{align*}
\{\{\text{bird}, \text{fly}\}, \{\text{abnormal}\}\}, \\
\{\{\text{bird}, \text{abnormal}\}, \{\text{fly}\}\},
\end{align*}
\]

possibly causing the state transition to s₂.

(ii) At the state s₂, 2 models as follows may be taken:

\[
\{\{\text{observed}, \{\text{fly}\}\}, \{\{\text{fly}\}, \{\text{observed}\}\}\},
\]

causing the state transition to s₂.

There are 2 cases: (a) If bird is not abnormal at state s₁, then fly is assigned to 1 so that the bird may fly, and the state s₁ transfers to the state s₂. After the state transition from s₁ to s₂, the bird may be “not observed, that is, “not observed”, at the state s₂, where it remains at the state s₂ (that is, the state transition from s₂ to s₂ may be assumed).

(b) If bird is abnormal at state s₁, then fly is assigned to 0 so that bird may not fly, from where the transition to s₂ occurs. At state s₂, bird may be observed with not fly, where the state s₂ transits to itself.

As in Example 3, we may have a sequence of a program:

\[
s₁ : \text{body₁} ; s₂ : \text{body₂} \ldots ; s_n : \text{body}_n \quad (n \geq 0).
\]

Each \(\text{body}_i\) (of such a program part “\(s_i : \text{body}_i\)” may contain algebraic expressions with states (to be transited to). That is,

\[
\bigwedge_i (a_{i_1} \land \ldots \land a_{i_m} \land \text{not } b_{i_1} \land \ldots \land \text{not } b_{i_m} \Rightarrow y)
\]

(to a state \(s_{i_j}\))

for \(y' = c\) or not \(c\) with \(c \in A\).

Such analysis motivates a language, whose program is, in Backus-Naur Form, inductively represented.

\[
\begin{align*}
\text{prog} ::= \text{null}_{\text{prog}} \mid \text{prog} \; s \; \text{body} \\
\text{body} ::= \text{null}_{\text{body}} \mid \text{body} ; \text{exp} \triangleright s \\
\text{exp} ::= \top \mid \exp \land (\text{form}) \\
\text{form} ::= \text{left} \Rightarrow a \mid \text{left} \Rightarrow \text{not } a \\
\text{left} ::= \top \mid \text{left} \land a \mid \text{left} \land \text{not } a
\end{align*}
\]

where the parentheses are used to denote the scope such that:

(i) \(s\) is a state variable.

(ii) \(a\) is a variable over the domain (set) of the given algebra, possibly denoting the greatest element \(\top\) and the least element \(\bot\).

(iii) the symbol \(\land\) means a “meet” in the underlined lattice (algebra), and the symbol \(\triangleright\) stands for “a transition to”.

(iv) \(\text{null}_{\text{prog}}\) and \(\text{null}_{\text{body}}\) denote the empty sequences for the definitions of a program \(\text{prog}\) and a body \(\text{body}\) (constrained by a state), respectively, where the semicolon “;” denotes an operation of concatenation.
**Conditional Statements:**

Whether it is in the manner of functional or procedural programming, the program contains the conditional statement:  
\begin{align*}
\text{if } C_1 \text{ then } A_1 \\
\text{else if } C_2 \text{ then } A_2 \\
\vdots \\
\text{else if } C_n \text{ then } A_n
\end{align*}

where \( C_1, \ldots, C_n \) are conditions, and \( A_1, \ldots, A_n \) are some functions (commands). It can be represented by the HA expression:
\[
(c_1 \Rightarrow a_1) \land \cdots \land \left( \neg c_1 \land \cdots \land \neg c_{n-1} \land c_n \Rightarrow a_n \right)
\]

with the null sequence \( \emptyset \), where \( c_i \) and \( a_i \) are algebraic elements in accordance with conditions \( C_i \) and function implementations \( A_i \) (successful, undefined or failed), respectively, for \( 1 \leq i \leq n \).

**Monitoring to Programming:**

For the construction of complex systems, monitoring is to be facilitated to the language constrained by states, as in the paper on modal mu-calculus extension (S. Yamasaki, 2020).

(a) Conditioning Including Awareness – With a function \( con_	ext{di} \), conditioning is regarded as a mapping of each state to a set of logical formulas, where logical formulas represent conditions. It involves awareness to (programming and network) environments, \( con_	ext{di} : S \rightarrow 2^\Phi \), where \( S \) is the set of set variables in this language, and \( \Phi \) stands for the set of logical formulas.

(b) Communication – With a relation \( com_	ext{mu} \), communications between states (that is, state constrained programs) may be described. It may follow the process algebra (Milner, 1999) as established principle. By the set \( S \) of states, the relation is just as \( com_	ext{mu} \subseteq S \times S \).

(c) Behavioral Aspects – The behavioral aspects of programming with interaction to the human side, a function of each state to a set of behaviours is required: \( behav : S \rightarrow 2^B \), where \( B \) is the set of states, and \( B \) is a set of behaviours.

### 4.2 Structure for State Transition

Concerning algebraic aspects of behaviours for the program of this form, regular language properties may be abstracted, rather than context-free language properties as in the paper (Paulo and Jose, 2017). A *star semiring* structure is given by the method (S. Yamasaki, in *COMPLEXIS* 2017).

Given an expression \( F \) (which is "exp" of \( \text{exp} \triangleright \text{s} \) over the set \( A \)) at some state to "body" in a program, we may have a pair \((I,J) \in 2^A \times 2^A\), which can be a 3-valued model of \( \text{exp} \).

With Heyting Algebra (HA) expressions to a constraint state, models of expressions can be considered with state transitions.

<table>
<thead>
<tr>
<th>HA expression</th>
<th>3-valued model, selected</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{exp} ) (as a program)</td>
<td>((I,J)) (a pair)</td>
</tr>
</tbody>
</table>

**Example 4.** Given a program of Example 3, if we take models:
\[
((\{\text{bird, fly}\}, \{\text{abnormal}\}) \text{ at state } s_1, \quad ((\{\text{fly}\}, \{\text{observed}\}) \text{ at state } s_2,
\]

then a sequence, constructed by a concatenation of \( \text{bird} \) (at state \( s_1 \)) with \( \text{fly} \) (at state \( s_2 \)), may be taken with consistency to both negations of \( \text{abnormal} \) (at state \( s_1 \)) and \( \text{observed} \) (at state \( s_2 \)).

With the domain \( A \), we can have denumerable expressions \( F_1, F_2, \ldots \), the models of which may be:
\[
(I_1, J_1), (I_2, J_2), \ldots \in 2^A \times 2^A.
\]

On such pairs causing state transitions based on the program description (of \( \text{body} \) at some state), we consider the operations of (a) the composition of models, and (b) the alternation to models.

As regards sequences causing state transitions, we know formality of automata, having a quintuple \( \mathcal{A} = (S, \Sigma, \delta, s_0, S_f) \), as a machinery, where:

(i) \( S \) is a finite set of states.

(ii) \( \Sigma \) is an alphabet.

(iii) \( \delta : S \times \Sigma \rightarrow S \) is a mapping.

(iv) \( s_0 \in S \) is the initial state.

(v) \( S_f \subseteq S \) is a final subset.

The mapping \( \delta \) is extended to the one \( \hat{\delta} : S \times \Sigma^* \rightarrow S \) for the set \( \Sigma^* \) of finite sequences concatenated from the symbols of \( \Sigma \) such that:
\[
\hat{\delta}(s, \text{null}_{\Sigma^*}) = s, \quad \text{with the null sequence } \text{null}_{\Sigma^*},
\]
\[
\hat{\delta}(s, u\sigma) = \delta(\hat{\delta}(s, u), \sigma) \text{ for } u \in \Sigma^* \text{ and } \sigma \in \Sigma.
\]

We then have the set of sequences
\[
L_{\hat{\delta}} = \{ w \in \Sigma^* \mid \hat{\delta}(s_0, w) \in S_f \}
\]
as accepted by the machinery \( \mathcal{A} \). In this case, we regard \( \Sigma \) as \( \{(I,J) \mid (I,J) \text{ is a model of some } \text{exp} \} \).
5 CONCLUSION

The primary result of this paper is a formal system formulation to a programming language, where the program (constrained by states) is algebraic expressions, applicable to analysis, and expected to design of complex AI. The language may contain state constraint programming and monitoring facilities. The implementations (accompanying state transitions) are globally captured with star semiring structure. The system realizes abstract state machine with:

(a) Compact and formal description of state constraint programs,
(b) Model theories of algebraic expressions as programs, and
(c) State transitions to represent processing sequences (associated with models).

From the views of AI and software technologies to complex systems with human, this idea based on abstract state machine may allow human computer interaction (HCI) to determine state transitions. The interaction can be involved in the program containing some expressions at states to the language system of this paper with some algebraic elements.

As a logical framework, this paper gives a theoretical basis with compact description, but its practical aspects or applications are to be examined for further studies. A complex AI to be interactive with cognitive facilities should be examined. As a software technology, semantics for implementation of programs of this language system is to be made clearer even in 3-valued logic, with respect to object-oriented programming (where the object class may be regarded as a state). Compared with sophisticated works on logical frameworks in logic and computation possibly for AI, there are concepts and ideas on knowledge. “Distributed knowledge” is discussed (Naumov and Tao, 2019), with quantified variables of quantities ranging over the set of agents. Concerning applications of the second-order predicates to knowledge, the paper (Kooi, 2016) contains the concept of knowing. Distributed knowledge processing is of more complexity even for the state constrained programs.

REFERENCES


