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 mu-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 nonclassical negation, and abstract attack and defense are regarded as elements of the argumentation concepts rather than communications by algebraic

Yamasaki, S.
Theoretical Basis of Language System with State Constraints.
DOI: 10.5220/0009385100800087
In *Proceedings of the 5th International Conference on Complexity, Future Information Systems and Risk (COMPLEXIS 2020)*, pages 80-87
ISBN: 978-989-758-427-5

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 programing 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. Communication and behavior facilities are involved in abstract monitoring. In this paper, communicative 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 Herbrad 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 thses 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) $(A, \bigvee, \bigwedge, \bot, \top)$ 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* $\bigvee$ and *meet* $\bigwedge$ are defined for any two elements of A.

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

$$c \sqsubseteq (a \Rightarrow b) \text{ iff } a \bigwedge c \sqsubseteq b.$$

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:

$$\bigwedge_{j \in \omega} (l_1^j \bigwedge \ldots \bigwedge l_{n_j}^j \Rightarrow l^j)$$

where l_i^j denotes a or $a \Rightarrow \bot$ 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:

$$\begin{array}{l}(bird \bigwedge \sim abnormal \Rightarrow fly) \bigwedge \\ (\Rightarrow bird) \bigwedge \\ (fly \Rightarrow \sim abnormal) \bigwedge \\ (fly \Rightarrow \sim observed) \bigwedge \\ (\sim fly \Rightarrow observed),\end{array}$$

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

(a) $\bigwedge$ (*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* with the procedure of "*negation as failure*", which means that the procedural proof failure of a proposition may infer its negation (Kowalski and Toni, 1996).

In a language system of this paper, it is a strict negation that the algebraic expression models, rather than default negation, even in considerations on *3-valued model of expressions*, with reference to the implication of the algebra. The 3-valued model of expressions contains more complexity than the 2-valued model, however, theoretical interests are involved for representation simplicity and algebraic treatments, which motivate us to take it for design of language system, as well as for solutions to the theories. We then make summaries of the theories, assuming in Heyting algebra that (a) the implication is based on Heyting algebra, and (b) the evaluation of *not a* (with respect to the value of a for $a \in A$) follows the rule:

a	*not a*
1	0
1/2	0
0	1

As is well known, we note some algebraic properties on the HA with parentheses of operation-priority representation, for the next subsection:

$$\begin{array}{l}((a \Rightarrow b) \bigwedge (b \Rightarrow c)) \sqsubseteq (a \Rightarrow c), \\ a \sqsubseteq not\ (not\ a), \\ not\ (a \bigvee b) = not\ a \bigwedge not\ b.\end{array}$$

2.2 Transformation of Expressions

In an Heyting algebra $(A, \bigvee, \bigwedge, \bot, \top)$, any expression Ex_1 derives some expression Ex_2 of the form (referred to in Section 2.1 and as in standard form):

$$\bigwedge_{j \in \omega} (l_1^j \bigwedge \ldots \bigwedge l_{n_j}^j \Rightarrow y^j),$$

where l_i^j and l^j are an expression a or *not a* (denoting $a \sqsubseteq \bot$), for $a \in A$, such that

$$Ex_2 \sqsubseteq Ex_1.$$

By the method which is as below shown with proof, we may have such a standardization to transform a given expression Ex_1 to Ex_2. If there is some model of Ex_2 in 3-valued domain, then it may be also the model of Ex_1. In this sense, the expression Ex_2 is worthwhile being obtained, as a standard form.

Transformation of Expressions:

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

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

(1) $X \Rightarrow (Y \Rightarrow Z)$ to be replaced by: $X \Rightarrow (Y \bigwedge Z)$.

(2) $X \Rightarrow (Y \bigvee Z)$ to be replaced by: $X \Rightarrow Y$ or $X \Rightarrow Z$.

(3) $X \Rightarrow (Y \bigwedge Z)$ to be replaced by:

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

(4) $X \Rightarrow not\ Y$ to be replaced by:

$$(Y \Rightarrow y) \bigwedge (X \Rightarrow not\ y)$$

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:

$$(X \Rightarrow Y) \bigwedge (\top \Rightarrow Z)$$

for the top element $\top$.

(6) $(X \bigvee Y) \Rightarrow Z$ to be replaced by:

$$not\ (not\ X \bigwedge not\ Y) \Rightarrow Z.$$

(7) $(X \bigwedge Y) \Rightarrow Z$ to be replaced by:

$$((x \bigwedge y) \Rightarrow Z) \bigwedge (X \Rightarrow x) \bigwedge (Y \Rightarrow y)$$

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

(8) $not\ X \Rightarrow Y$ to be replaced by:

$$(x \Rightarrow X) \bigwedge (not\ x \Rightarrow Y)$$

for an arbitrarily chosen $x \in A$.

Proposition 1. *Given an expression Ex_1, an expression Ex_2 of standard form is derived by the Transformation of Expressions such that $Ex_2 \sqsubseteq Ex_1$.*

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 Ex_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. □

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

$$\Psi_F : 2^A \times 2^A \to 2^A \times 2^A,$$
$$\Psi_F(I_1, J_1) = (I_2, J_2),$$

has been noted with *order* of componentwise subset inclusion:

I_2 contains p such that p is obtained by:
$\exists (p_1 \bigwedge \ldots \wedge p_i \bigwedge not\ p_{i+1} \bigwedge \ldots not\ p_j \Rightarrow p)$ in F.
$\quad \forall p_k \in I_1\ (1 \le k \le i), \forall p_{k'} \in J_1\ (i+1 \le k' \le j)$,

J_2 contains q such that q is obtained by:
$\forall (q_1 \bigwedge \ldots \bigwedge q_i, not\ q_{i+1}, \ldots, not\ q_j \Rightarrow q)$ in F.
$\quad \exists q_k\ (1 \le k \le i).\ q_k \in J_1$, or
$\quad \exists q_{k'}\ (i+1 \le k' \le j).\ q_{k'} \notin J_1$, or
$\exists (q_1 \bigwedge \ldots \bigwedge q_i, not\ q_{i+1}, \ldots, not\ q_j \Rightarrow\ not\ q)$ in F.
$\quad \forall q_k\ (1 \le k \le i).\ q_k \notin J_1$, and
$\quad \forall q_{k'}\ (i+1 \le k' \le j).\ q_{k'} \in J_1$.

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. Since the mapping Ψ_F is not monotonic, the method by (pre-)fixpoint of Ψ_F is not always available as a modelling of the given expression 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 $X = y_1; \ldots; y_n$ $(n \ge 0)$ with y_i being a or $not\ a$ for $a \in A$ and with the concatenation operation ";" (which is treated as $\bigwedge$), 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

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

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 $[a?\ fail]$ for $x = not\ a$, and $[X?\ suc]$.
(b) $[x;X?\ suc]$, if $[a?\ fail]$ for $x = not\ a$ where there is $Y \Rightarrow x$ in F with $[Y?\ no\ fail]$, and $[X?\ suc]$.

(4) $[x;X?\ fail]$, if there is no part with $x = a \in A$ being the right side of "$\Rightarrow$" in F, or $[X?\ fail]$.

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

(6) $[x;X?\ fail]$, if $x = not\ a$ such that $[a?\ no\ fail]$, or $[X?\ fail]$.

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.

$$(bird \bigwedge not\ abnormal \Rightarrow fly) \bigwedge$$
$$(\top \Rightarrow bird) \bigwedge$$
$$(fly \Rightarrow not\ abnormal) \bigwedge$$
$$(fly \Rightarrow not\ observed) \bigwedge$$
$$(not\ fly \Rightarrow observed).$$

Following the Procedure, we can see that:

$[bird?\ suc]$ and $[fly?\ no\ fail] \longrightarrow$
$[abnormal?\ fail], [fly?\ suc], [observed?\ fail]$, or
$[bird?\ suc]$, and $[abnormal?\ no\ fail] \longrightarrow$
$[fly\ fail], [abnormal?\ suc], [observed?\ suc]$.

where "$\longrightarrow$" means inferences for the successes and failures of queries.

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

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

The Procedure works to construct a pair (I,J) so that (a) if $[a?\ suc]$ then $a \in I$, and (b) if $[a?\ fail]$ 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*

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

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 Ψ_F for $\Psi_F(I,J) \subseteq_c (I,J)$: We show on the case of $I \cap J = \emptyset$ that if we let $\Psi_F(I,J) = (I_0,J_0)$ then $(I_0,J_0) \subseteq_c (I,J)$.
(1) Let $a \in I_0$. (i) It follows with respect to the mapping of Ψ_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 Y such that $[a?\ suc]$. Note by the Procedure to simulate the mapping Ψ_F that $[not\ c?\ suc]$ comes from $[c?\ fail]$ for $c \in J$, or from $\exists Z.\ (Z \Rightarrow not\ c$ and $[Z?\ no\ fail])$ 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 Ψ_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 Ψ_F may assume $a \in J_0$. It follows by the Procedure that $[a?\ fail]$. 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 $c \notin J)$, then $a \in J_0$. By the Procedure, $[b?\ fail]$ $(b \in J)$ and $[c?\ no\ fail]$ $(c \notin J)$ causes $[a?\ fail]$. Thus $a \in J$. (b) If $\exists Y.\ (Y \Rightarrow not\ a$ in Y, where any $b \notin J$ or any *not* c for $c \in J$ in $Y)$, then $a \in J_0$. By the Procedure, if $b \notin J$ then $[b?\ no\ fail]$, and if $c \in J$ then $[c?\ fail]$. It follows that: $[Y?\ no\ fail]$ such that $[a?\ fail]$. Thus $a \in J$. □

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

(i) $[not\ a?\ suc]$, if $[a?\ fail]$.

(ii) $[not\ a?\ fail]$, if $[a?\ no\ fail]$.

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

(a) a query *not* a? is a success, if $[a?\ fail]$, and
(b) a query *not* a? 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).

States	s_1	s_2
Processing	queriy_1	query_2
Strategy	expression_1	expression_2
Implementation	program_1	program_2

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:

s_1 :
$(bird \bigwedge not\ abnormal \Rightarrow fly) \bigwedge (\top \Rightarrow bird)$
$\bigwedge (fly \Rightarrow not\ abnormal) \rhd s_2$

s_2 :
$(fly \Rightarrow not\ observed) \bigwedge$
$(not\, fly \Rightarrow observed) \rhd s_2$

(i) At the state s_1, there may be 2 models:

$$(\{bird, fly\}, \{abnormal\}),$$
$$(\{bird, abnormal\}, \{fly\}),$$

possibly causing the state transition to s_2.

(ii) At the state s_2, 2 models as follows may be taken:

$$(\{observed\}, \{fly\}), (\{fly\}, \{observed\}),$$

causing the state transition to s_2.

There are 2 cases: (a) If *bird* is not *abnormal* at state s_1, then fly is assigned to 1 so that the *bird* may fly, and the state s_1 transfers to the state s_2. After the state transition from s_1 to s_2, the *bird* may be "not" *observed*, that is, "*not observed*", at the state s_2, where it remains at the state s_2 (that is, the state transition from s_2 to s_2 may be assumed).
(b) If *bird* is *abnormal* at state s_1, then fly is assigned to 0 so that *bird* may not fly, from where the transition to s_2 occurs. At state s_2, *bird* may be *observed* with *not* fly, where the state s_2 transits to itself.

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

$$s_1 : body_1; s_2 : body_2 : \ldots; s_n : body_n\ (n \geq 0).$$

Each $body_i$ (of such a program part "$s_i : body_i$") may contain algebraic expressions with states (to be transited to). That is,

$\bigwedge_i (a_1^i \bigwedge \ldots \bigwedge a_{n_i}^i \bigwedge not\ b_1^i \bigwedge \ldots \bigwedge not\ b_{m_i}^i \Rightarrow y^i)$
(to a state s_j)

for $y^i = c$ or *not* c with $c \in A$.

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

$$prog ::= null_{prog} \mid prog;\ s : body$$
$$body ::= null_{body} \mid body;\ exp \rhd s$$
$$exp ::= \top \mid exp \bigwedge (form)$$
$$form ::= left \Rightarrow a \mid left \Rightarrow not\ a$$
$$left ::= \top \mid left \bigwedge a \mid left \bigwedge not\ a$$

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 $\bigwedge$ means a "meet" in the underlined lattice (algebra), and the symbol $\rhd$ stands for "a transition to".

(iv) $null_{prog}$ and $null_{body}$ denote the empty sequences for the definitions of a program *prog* and a body *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:

if C_1 then A_1
else if C_2 then A_2
...
...
else if C_n then A_n

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:

$$(c_1 \Rightarrow a_1) \bigwedge \ldots \bigwedge (not\ c_1 \bigwedge \ldots \bigwedge not\ c_{n-1} \bigwedge c_n \Rightarrow a_n)$$

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 *condi*, 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, $condi : S \to 2^{\Phi}$, where S is the set of set variables in this language, and Φ stands for the set of logical formulas.

(b) Communication – With a relation *commu*, 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 $commu \subseteq S \times S$.

(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: $behav : S \to 2^{B}$, where S 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 "$exp \rhd s$" 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 exp.

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

HA expression	3-valued model, selected
exp (as a program)	(I,J) (a pair)

Example 4. Given a program of *Example* 3, if we take models:

$(\{bird, fly\}, \{abnormal\})$ at state s_1, and
$(\{fly\}, \{observed\})$ at state s_2,

then a sequence, constructed by a concatenation of

bird (at state s_1) with fly (at state s_2),

may be taken with consistency to both negations of *abnormal* (at state s_1) and *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 *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

$$\mathcal{A} = \langle S, \Sigma, \delta, s_0, S_f \rangle,$$

as a machinery, where:

(i) S is a finite set of states.

(ii) Σ is an alphabet.

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

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

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

The mapping δ is extended to the one $\hat{\delta} : S \times \Sigma^{\star} \to S$ for the set $\Sigma^{\star}$ of finite sequences concatenated from the symbols of Σ such that:

$$\begin{cases} \hat{\delta}(s, nulll_{\Sigma^{\star}}) = s, \text{ with the null sequence } null_{\Sigma^{\star}}, \\ \hat{\delta}(s, u\sigma) = \delta(\hat{\delta}(s,u), \sigma) \text{ for } u \in \Sigma^{\star} \text{ and } \sigma \in \Sigma. \end{cases}$$

We then have the set of sequences

$$L_{\mathcal{A}} = \{w \in \Sigma^{*} \mid \hat{\delta}(s_0, w) \in S_f\}$$

as accepted by the machinery $\mathcal{A}$. In this case, we regard Σ as $\{(I,J) \mid (I,J)$ is a model of some $exp\}$.

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 quantifies ranging over the set of agents. Concerning applications of the second-order predicates to knowledge, the paper (Kooi, 2016) contains the concept of knowing. Distributive knowledge processing is of more complexity even for the state constrained programs.

REFERENCES

Bertolissi, C., Cirstea, H., and Kirchner, C. (2006). Expressing combinatory reduction systems derivations in the rewriting calculus. *Higher-Order.Symbolic.Comput.*, 19(4):345–376.

Cardelli, L. and Gordon, A. (2000). Mobile ambients. *Theoret.Comput.Sci.*, 240(1):177–213.

Dam, M. and Gurov, D. (2002). Mu-calculus with explicit points and approximations. *J.Log.Comput.*, 12(1):119–136.

Dragoni, A., Giorgini, P., and Serafini, L. (1985). Mental states recognition from communication. *J.Log.Program.*, 2(4):295–312.

Droste, M., Kuich, W., and Vogler, H. (2009). *Handbook of Weighted Automata*. Springer.

Giordano, L., Martelli, A., and Schwind, C. (2000). Ramification and causality in a modal action logic. *J.Log.Comput.*, 10(5):625–662.

Governatori, G., Maher, M., Autoniou, G., and Billington, D. (2004). Argumentation semantics for defeasible logic. *J.Log.Comput.*, 14(5):675–702.

Hanks, S. and McDermott, D. (1987). Nonmonotonic logic and temporal projection. *Artifi.Intelli.*, 33(3):379–412.

Hennessy, M. and Milner, R. (1985). Algebraic laws for nondeterminism and concurrency. *J.ACM*, 32(1):137–161.

Kooi, B. (2016). The ambiguity of knowability. *Rev.Symb.Log.*, 9(3):421–428.

Kowalski, R. A. and Toni, F. (1996). Abstract argumentation. *Artifi.Intell.Law*, 4(3-4):275–296.

Kozen, D. (1983). Results on the propositional mu-calculus. *Theoret.Comput.Sci.*, 27(3):333–354.

Kucera, A. and Esparza, J. (2003). A logical viewpoint on process-algebraic quotients. *J.Log.Comput.*, 13(6):863–880.

Merro, M. and Nardelli, F. (2005). Behavioral theory for mobile ambients. *J.ACM.*, 52(6):961–1023.

Milner, R. (1999). *Communicating and Mobile Systems: The Pi-Calculus*. Cambridge University Press.

Mosses, P. (1992). *Action Semantics*. Cambridge University Press.

Naumov, P. and Tao, J. (2019). Everyone knows that some knows: Quantifiers over epistemic agents. *Rev.Symb.Log.*, 12(2):255–270.

Osorio, M., Navarro, J. A., and Arrazola, J. (2004). Applications of intuitionistic logic in answer set programming. *TLP*, 4(3):325–354.

Park, D. M. R. (1970). Fixpoint induction and proof of program semantics. *Machine Intelligence*, 5:59–78.

Paulo, R. and Jose, J. (2017). Instrumenting a context-free language recognizer. In *Proc. of 19th ICEIS - Vol. 2*, pages 203–210.

Reiter, R. (2001). *Knowledge in Action*. MIT Press.

Reps, T., Schwoon, S., and Somesh, J. (2005). Weighted pushdown systems and their application to interprocedural data flow analysis. *Sci.Comput.Program.*, 58(1-2):206–263.

Rutten, J. (2001). *On Streams and Coinduction*. CWI.

Spalazzi, L. and Traverso, P. (2000). A dynamic logic for acting, sensing and planning. *J.Log.Comput.*, 10(6):787–821.

Venema, Y. (2006). Automata and fixed point logic: A coalgebraic perspective. *Inf.Comput.*, 204(4):637–678.