数理解析研究所講究録
第 709 巻 1989 年 143-165

# Executable and formalized logic programming language based on time interval logic

Naoyuki Nide

Educational Center for Information Processing,
Kyoto University

## Abstract

The tense logic is an extensions of the classical logic which is actively studied. In particular, the time interval logic, or the logic based on intervals of time, has been supported since it fits human intuitions as to time continuity. There are several interval-logical languages but none of them seems to allow both construction of an underlining axiomatic formal system and executability. This paper introduces such a language, on whose formal system model-theoretic soundness and completeness (restricted on the propositional logic for the present) are proven. Besides, it is possible to attain efficient real-time execution by capturing the changes of predicates' truth values as events and performing inferences about them when an external event occurs.

## §1 introduction

Considering situations where we apply a logic programming language to a practical use which essentially includes time notion, such as real-time process controlling, it's natural to work with tense logic, since it can explicitly describe facts about time. Besides, tense logic is useful for AI applications such as knowledge representation.

Roughly speaking, tense logic is separated into two groups—time point logic and time interval logic. Looking at the real situations where we are reasoning facts concerning with time or we are programming using a logic programming language which includes time notion, time interval logic is tend to be used more often. For example, "An elevator continues to move from the time it starts to the time it arrives at another floor", "The door of the elevator is closed while the elevator is moving", etc. These statements seem to be naturally expressed by languages which are based on time interval logic.

Interval logics are formulated mainly in two major ways. One is to treat formulas which have truth values not on time points but on time intervals, and regard "includes" and "before" relations between intervals as modalities[1][2]. The other way is to give truth values to each formula on every time point, and then determine truth values on every time interval[3][4]. The former, which composes formal systems from axioms and rules, gives us brief, concise and strict system, and some good properties like soundness and completeness make the systems easier to treat. When we discuss possibility of using them as real time programming language, however, there is a difficulty that we can't refer to future states in conditional formulas. On the other hand, the latter is formulated from the first so that it can be real-time executed. Looking at logics based on it, operational semantics are given to them, but it seemed that in many cases formal systems aren't given. Moreover, if we choose discrete set of time points in such a logic, there may be some descriptions which mismatch our intuition that time is dense (brief discussion about such examples is in §6).

If there is a language which have merits of both, it can be more widely applicable than conventional ones. For example, time representation have been done quite separately in the two ways above. If we can treat them uniformly, there will be some new suggestions for time representation. Moreover, in such a language, we can possibly use same program or process commonly in both real-time execution and static execution (for example, inference using historical knowledge), because a real-time program itself can also regarded as a database of knowledges about time. (As an example we briefly referred to possibility of program checking in §8).

On such purposes, an time-interval logic system, named AYA, is formulated in this paper. In AYA, predicates' truth values are defined on time intervals, and changes of them are regarded as events. By coupling two events, we can compose a new formula which holds on a time interval from the first event to another event. In such way, we can form the system which can describe events without introducing time point nor time-point predicate. Besides, a formal system based on axioms and rules, which is independent of its model theoretic semantic, can be formulated by regarding coupling of events as modalities. Further, soundness and completeness can be proved on it. This paper also refers to the possibility of using this system as a real time programming language.

The contents of this paper is as follows: in §2, we give a brief explanation of AYA's features using simple example and some comparison with other works. The definition of AYA's syntax is given in §3, and formulation of a model is described in §4. Next, AYA's formal system is constructed in §5, and soundness and completeness (which currently restricted on propositional logic) are given in §6. Then, in §7, we refer to execution of AYA as a real-time programming language, and in §8, an example of execution is given, which based on simple example again. Finally, in §9, we discuss about problems to be solved.

## §2 A sample

As an example of real-time control, we choose a simple elevator controlling. In this example, the elevator has a box which carries people between the 1st(ground) floor and the k-th floor. In every floor there are an upward-call button and a downward-call button. The box has destination buttons which correspond to each floor.

Elevator is real-time automatically controlled and a human indicates his destination by the buttons when necessary. When the elevator arrives at the demanded floor, it stops and the door opens. Once a button is pushed, it has to remain in the "pushed" state until the elevator arrives at the appropriate floor.

To survey AYA, in this section we give only a small part of the program of elevator controlling. Detailed discussion about whole elevator-controlling program is given in §8. Moreover, we give only an intuitive and informal explanation in this section. See the succeeding sections for more details about AYA's syntax and semantics.

First of all, as an example we take up a sentence of the elevator-controlling program written in AYA.

行かねば(n) ∧ より上(n) → 上需要有 …(1)

Atomic formulas in AYA is defined in just the same way as the original first-order logic. For example, "行かねば(n)" is an atomic formula. Symbols such as ∧ and → (imply) are also introduced as usual. (1) represents that "if the elevator must go to the n-th floor and it's higher than the current place, then there is an upward demand."

One of the forms proper to AYA is the limitation of time section by "⇒". For example the following sentence is a formula in AYA.

↓(移動中(x,n)∧行かねば(n)) ⇒ ↑((上需要有∨下需要有)∧¬ドア開) → 停止中(n) …(2)

Generally, if A, B are formulas, ↓A⇒↑B is also a formula. (↓A is an event which represents "A finishes to hold," and ↑B is an event "B begins to hold." They are not formulas themselves.) This formula holds on every time interval from the time when A finishes holding until the time when B begins holding. This sentence represents that "the elevator stops at the n-th floor, where it had to go, from when it finished moving to there, until the door closes and a demand from another floor occurs." In this case, an interval on which 停止中(n) holds is just on the right (i.e. the future) of an interval on which 移動中(x,n)∧行かねば(n) holds. Like this example, natural expressions are enabled in situations we express relations between time intervals directly.

Let us try to express the same thing by a time point logic, for example, [7]. Here we use the following abbreviations:

"移動中(x,n)&行かねば (n)" to A
"(上需要有 | 下需要有)&～ドア開" to B

where the symbols &, |, ～ indicates "and", "not", "or" in [7]. Since [7] has an "until" operator, it seemed to be possible to express the same situation by the following formula.

～A → 停止中(n) until B

But actually it has a problem. Even if the elevator is moving to a floor other than the n-th floor, ～A holds, therefore 停止中(n) also holds. It must be expressed as

●A & ～A → 停止中(n) until B

where ● is an operator in [7] which represents "the time point just before now". It is more natural to use time interval rather than to use "●".

There is another problem which may occur in logical languages with time notion. The previous example of [7] refers to the truth value of A at "a moment before". In our feeling, time points are dense, so the time A finishes holding and the time ～A starts holding is same. But in that example it is written as if ～A started holding at "a moment after" the time A finished holding, which doesn't match our intuition. Problems like this tend to occur in time logics based on discrete time points.

A similar problem also occurs in [4], a time-interval logic on discrete time points. [4] introduces a symbol "↑". "↑X" means that in a certain time interval, at first a binary signal X's value is 0 and it changes to 1 halfway, finally it is 1. "↑X" is defined as follows.

$\vdash \uparrow X \equiv [(X \approx 0);\ \text{skip};\ (X \approx 1)]$

note: the symbol ~~ is accurately doubled wavy lines.

";" is the chop operator. "~~" means its both sides are identically equal on the given interval.
This formula uses the predicate "skip," which holds on every interval of length 1. Therefore " ↑X holds in a certain interval I " means that "in a interval L which is the beginning of a certain interval I, X ~~ 0 holds, and the interval from the 'next moment' of the end of L to the end of I, X ~~ 1 holds." In our intuition, the time X ~~ 0 finishes holding and the time X ~~ 1 starts holding are same, but this formula uses "skip", which makes it slightly different from out feeling.
To express a similar situation (however, not accurately same) in AYA, we can write as follows.

$(\downarrow(\uparrow A \Rightarrow \downarrow(X=0)) \Rightarrow \downarrow A) \rightarrow X=1$

Hereupon A has almost the same meaning as ↑X in [4]. This can express more directly than [4] that the intervals on which X ~~ 0 and on which X ~~ 1 are adjoin.

From here we discuss about some differences and similarities between AYA and other time logics, and merits and demerits of AYA.
For the first, [6] is very similar to AYA. The reason is AYA refers to [6] from the first. [6] defines "interval propositions" which holds on time intervals, then take notice of changes of their truth values, and formulate inference rules about them. Some features are common to [6] and AYA, for example, both regards changes of truth values as events, and enable real-time execution on computers by performing inferences about events.
From the viewpoint of executable language, there also have common merits. For example, in [5] or [7] inferences must be performed for each time point, meanwhile processor of AYAor [6] must execute inferences only when external event occurs. This suggests we can get efficient processor of those.
Another feature of AYA and [6] is that we don't have to keep all the histories of truth values, which also seemed to improve efficiency of their processors. By contrast, In [3], for example truth value of a proposition "□G" at a certain time depends on G's truth value of all the time since a program starts until the current time. Therefore the processor of it have to keep all the history of truth values. It seemed to be considerably large overhead.
One of the differences between [6] and AYA is that [6] is event oriented. In [6] inference rules about occurring of events can be written directly in a program, but in AYA there is an axiomatic formal system and a program is a set of formulas. Programs written in AYA can be executed not only by real-time execution which constructs a model but also by static execution inferring formulas from rules (just like Prolog) . Thus for the example, there is a possibility to useAYA to check a spec of a program written in AYA itself. It can be regarded as an advantage of AYA in comparison with current [6].
There is another difference between them in the viewpoint of model theory. In making a least model, [6] uses the following order: (to say roughly) a model with a smaller number of changes of truth values is smaller. It is more elegant than AYA. The reason of it is that programs in [6] are event driven, but AYA is not completely event driven. In [6], an event directly changes a formula's truth value. But in AYA, a program is in the form of inference rules. For example suppose that a program in AYA has the following two clauses: "↑A⇒↑B→ C" and " ↑A'⇒↑B' → C". Then an event ↑B not necessary changes C's truth value, because "↑A'⇒ ↑B'" may be hold there. Therefore a least model of AYAcan't made in the same way to [6]. It uses an order similar to [5], that a model with less predicates which are true is smaller.
From a viewpoint of formal system, we can formalize AYA as a modal logic in the same way as [1] and [2]. The main difference between AYA and [1], [2] is that AYA tried to incorporate a merit of [3] and [4], i.e. the executability on computers. To do so, AYA cut down a big merit of [1] and [2]. In [1] and [2] there are minimum restrict in making model, but in AYA there are strong restrict, as described in the next chapter. For example, let us think about a predicate "walk(n)" which is true on a time interval while which a man walks just nkilometers. If we use the chop operator (expressed as "∧° " in [1]. In this paper it doesn't used) , we

can use the following situation.

$\text{walk}(n) \wedge^{\circ} \text{walk}(m) \rightarrow \text{walk}(n+m)$

It is an expression of a knowledge that "if a man walks n kilometers, and then walks m kilometers, totally he walks n+m kilometers". But current AYA can't treat this, because two intervals on which a same predicate holds can't overlap, so it can't express the situation in which two intervals on which walk(m) holds overlap.

This is a problem when we try to construct a system which also has a merit of executability on computers. However, applications of time logic isn't restricted on real-time execution such as process controlling. There is wider use of it, for example knowledge expression about time. It will be of use to treat both in same theory in the future work, though it seemed to be difficult.

## § 3 Syntax of AYA

Here we give the definition of formulas of AYA.

<1> atomic formula    p(t1, t2, ⋯, tn)

p is a predicate symbol and each $t_i$ is a term. Terms are constructed by variables, constants and function symbols, just like the ordinary first-order logic.

<2> formula

(1) every atomic formula is a formula.
(2) If A, B are formulas and x is a variable,
$A \rightarrow B$, $\neg A$, $\forall x A$, $\exists x A$ are formulas.

$A \vee B$, $A \wedge B$, $\exists x A$ are defined as aliases of $\neg A \rightarrow B$, $\neg(A \rightarrow \neg B)$, $\neg \forall x \neg A$. We suitably use parenthesis to show combination order.

(3) If u, v are event expressions $u \Rightarrow v$ is a formula.

Hereupon event expressions are defined as follows:

(a) $\phi$, 'start' are event expressions.
(b) If A is a formula, $\uparrow A$, $\downarrow A$, $\overline{\phantom{x}} A$, $\_ A$ are event expressions.
(c) If u, v are event expressions, $u \cap v$, $\sim u$ are event expressions.
(d) If u is an event expression, after(u), before(u) are event expressions.

We abbreviate $\sim(\sim u \cap \sim v)$ to $u \cup v$. In this case also, we suitably use parenthesis.

Now we describe semantics of these formula, simply and informally.

Each formula holds on time intervals. We don't regard any time point as an interval. In § 4, an interpretation function is defined as a function from a formula to intervals on which it holds. Note that in this paper we give the following restriction on the interpretation function.

(1) If a formula holds on two intervals, they don't overlap nor have their borders in common.
(2) If a formula holds on more than one formula, their border points don't accumulate.

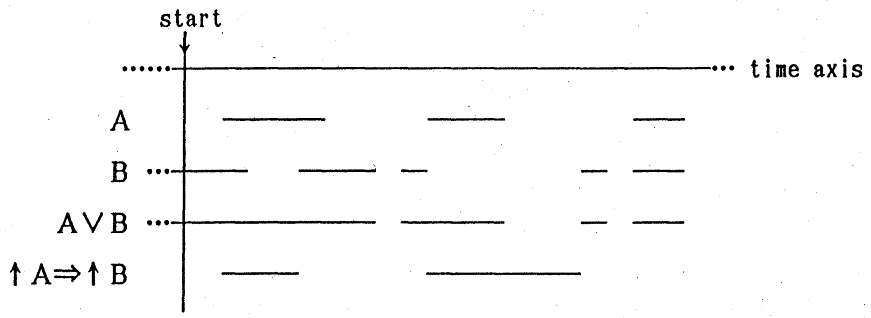
An event expression represents time points at which a certain event occurs. For example $\uparrow A$ expresses time points when A changes from false to true. There are two special event expressions $\phi$, and 'start'. Under every interpretation, $\phi$ corresponds to the empty set of time points, and 'start' corresponds to the origin of time axis. Note that an event expression isn't a formula itself. A formula made of two event expression combined by "$\Rightarrow$" denotes time intervals from an event occurs until another event occurs. (Don't confuse "$\Rightarrow$" with the implication symbol "$\rightarrow$". These are perfectly different things.) For example a formula $\uparrow A \Rightarrow \uparrow B$ holds on every time interval from when A begins holding until when B begins holding. We can also regard it as a formula made of a binary modal operator "$\uparrow\Rightarrow\uparrow$" which takes two arguments A, B.

Practically speaking, $\uparrow A$ and $\uparrow B$ may occur at a same time. So actually we define that $\uparrow A \Rightarrow \uparrow B$ holds from when $\uparrow A$ occurs and $\uparrow B$ doesn't occur until when $\uparrow B$ occurs and $\uparrow A$ doesn't occur. If you rather want to express intervals from when $\uparrow A$ occurs ($\uparrow B$ may occur at the same time) until when $\uparrow B$ occurs but $\uparrow A$ doesn't occur, it's sufficient to write $\uparrow B \cap \sim \uparrow A$.

Let's see another example. $\uparrow A \Rightarrow \phi$ is a formula which begins to hold when $\uparrow A$ occurs, and holds forever. Using these examples, we can represent some situations similar to [5]'s $\square$, until, atnext. For example "$\square A \rightarrow B$" in [5] can be written as "$\uparrow A \Rightarrow \phi \rightarrow B$", "$A \rightarrow B$ until C" can be expressed as "$\uparrow A \Rightarrow \uparrow C \rightarrow B$". Besides, "$A \rightarrow B$ atnext C" corresponds to "$\downarrow(\uparrow A \Rightarrow \uparrow C) \Rightarrow \downarrow B \rightarrow B$".

Since an event expression isn't a formula itself, AYA doesn't have any time point predicates (predicates whose truth values are defined on time points) .

To show an example of interpreting of formulas, we now illustrate a sample situation a few formulas' truth values.



'start' denotes the origin of time axis. It also represents the beginning of a real-time execution of a program. Truth values of each predicate is defined also before 'start', but they doesn't change there.

There's another kind of event expressions in AYA. An event expression $^{-}C$ denotes time points on which c is continuously holding. So "$\uparrow A \cap {}^{-}C \Rightarrow \uparrow B$" is another example of formula. It starts holding when $\uparrow A$ occurs while C is holding, and finishes holding when $\uparrow B$ occurs. Also in this case, we can regard it as a modal operator "$\uparrow \cap {}^{-} \Rightarrow \uparrow$" with three arguments. From this point of view, we can regard AYA as a modal logic with infinite many modal operators.

However, permitting event expressions like $^{-}C$ spoils the guarantee that an event expression always represents a discrete set of time points. One method of preventing this problem is to limit event expressions to ones which represents discrete time points under all interpretation functions, but it makes formalizing difficult. So we take another method in §4. We give meaning to every formula $u \Rightarrow v$, irrespective of whether u and v represent discrete time points. That's why <2>-(3)-(d) is prepared in the definition above. "after(u)" represents a set of time points which satisfy the following condition: "its sufficiently small left (i.e. past) neighbor is included in u", and "before(u)" is defined similarly by the right neighbor. Here we abbreviate "$\sim$after(u)$\cap$ (u$\cup$ before(u))" to "head(u)", then it always denotes discrete set of time. Especially, if u itself is discrete, u equals to head(u). So generally we define interpretation of head(u)$\Rightarrow$head(v) equal to that of $u \Rightarrow v$.

## § 4 composing a model

In this paper, we treat $[0, \infty)$ as a set of time. As described in § 3, to say intuitively, a formula is interpreted as a set of time intervals on which it holds.

First, we call the following thing an "interval sequence" or simply "sequence":

$[(p1,q1), (p2,q2), (p3,q3), \cdots]$
Hereupon $p1<q1<p2<q2<p3<q3<\cdots$

Here $pn, qn \in \{t \mid t=-\infty \text{ or } 0 \leqq t \leqq \infty\}$ . A sequence may be infinitely long. If not, we say the sequence is finite. Though we treat $[0, \infty)$, we also allow $-\infty$ and $\infty$ to be a value of pn or qn. The reason why we do so is that we want to treat all time points uniformly. We'll give a detailed explanation for it later. Practically speaking, only p1 can be $-\infty$ and only qn (when the sequence is finite) can be $\infty$. We call pn and qn (except $-\infty$ and $\infty$) the edge points of this sequence.

An interpretation function f is a function which maps an atomic formula to an interval sequence. In this paper, however, we put the following restriction on the definition of interpretation function.

> If f is an interpretation function, then for any predicate symbol p,
> {edge points of $f(p(t1,\cdots,tn)) \mid$ p's arity is n, t1~tn moves among all terms}
> doesn't have any accumulating points.

For example $f(A)= [(1,2), (3,4)]$ means that A holds on two intervals (1,2) and (3,4).
Note that (p,q) doesn't mean open interval. We simply express an interval by a pair of numbers. Since AYA doesn't use any differences between open interval and closed interval, we don't have to tell them apart when making a model.

If an interpretation function f is given, we can extend it to a function from a formula to a sequence by the following way.

○Suppose $f(A)= [(p1,q1),(p2,q2),\cdots]$ . Then

$f(\neg A)= [(-\infty,p1),(q1,p2),\cdots,(qn,\infty)]$
where n is the length of $f(A)$.

if $p1=-\infty$, $(-\infty,p1)$ is omitted.
if $qn=\infty$, $(qn,\infty)$ is omitted.
if $f(A)$ is infinitely long, so is $f(\neg A)$.

○Suppose $f(\neg A)= [(p'1,q'1),(p'2,q'2),\cdots]$ and
$f(B)= [(r1,s1),(r2,s2),\cdots]$ . Then

$f(A\rightarrow B)= [(t1,u1),(t2,u2),\cdots]$

Hereupon
$t1=\min(p'1, r1)$

un is the minimum element of $\{q'1, s1, q'2, s2, \cdots\}$
which satisfies:

$tn < un$ and $\forall i$ not$(p'1 \leqq un < q'1$ or $r1 \leqq un < s1)$

if such a $u_n$ doesn't exist, un is $\infty$ and $f(A\rightarrow B)$ is finite.

if $n \geqq 2$, tn is the minimum element of $\{p'1, r1, p'2, r2, \cdots\}$
which satisfies:

$u_{n-1} < t_n$

if not exist, $f(A \rightarrow B)$ is finite.

○Let $f(A(t))$ be $[(p1(t),q1(t)),(p2(t),q2(t)),\cdots]$ for every term t.

$f(\forall x A(x)) = [(p1,q1),(p2,q2),\cdots]$

pn is the minimum point which satisfies:

$\forall t\ \exists m\ pm(t) \leqq pn \leqq qm(t)$ and $\exists t\ \exists m\ pm(t)=pn$
and if $(n>1)$ $p_n > q_{n-1}$

if not exist, $f(\forall x A(x))$ is finite.

qn is the minimum point which satisfies:

$\forall t\ \exists m\ pm(t) \leqq qn \leqq qm(t)$ and $\exists t\ \exists m\ qm(t)=qn$ and $qn > pn$

if not exist, qn is $\infty$ and $f(\forall x A(x))$ is finite.

Note: We can define $f(\forall x A(x))$ in such way because we gave non-accumulating condition in § 3.

Next we define $f(u \Rightarrow v)$, where u, v is event expressions. As a preparation, we regard f to be also a function from an event expression to a set of time points (subset of $\{t \mid -\infty < t < \infty\}$).

$f(\phi)$=empty set
$f(\text{start}) = \{0\}$

If $f(A) = [(p1,q1),(p2,q2),\cdots]$,
$f(\uparrow A) = \{p1, p2, \cdots\} - \{-\infty\}$
$f(\downarrow A) = \{q1, q2, \cdots\} - \{\infty\}$
$f(\bar{\ } A) = \{t \mid \exists n\ pn < t < qn\}$
$f(\_A) = f(\bar{\ } \neg A)$

$f(u \cap v) = f(u) \cap f(v)$
$f(\sim u) = \{t \mid t \in R\} - f(u)$
(In the right sides $\cap$, $-$ are set operators. R is the set of real numbers)

$f(\uparrow A)$ and $f(\downarrow A)$ represent time points at which A's truth value changes, and $f(\bar{\ } A)$, $f(\_A)$ are other points. Clearly from the definition, for all A, $f(\uparrow A)$ and $f(\downarrow A)$ don't include any points smaller than 0. In other words, before the origin of time axis, no predicate changes its truth value. For all that, 0 can be $f(\uparrow A)$'s or $f(\downarrow A)$'s element, because we allowed $-\infty$ as a start point of interval. In such way we uniformly treat the time point 0 and other points. Similarly, we allow $\infty$ as an end point of interval to treat the point at infinity and other points uniformly.

Definitions relating to "after", "before" are as follows:

$f(\text{after}(\uparrow A)) = f(\text{before}(\uparrow A))$=empty set
$f(\text{after}(\downarrow A)) = f(\text{before}(\downarrow A))$=empty set
$f(\text{after}(\bar{\ } A)) = f(\bar{\ } A \cup \downarrow A))$
$f(\text{before}(\bar{\ } A)) = f(\bar{\ } A \cup \uparrow A))$
$f(\text{after}(\_A)) = f(\_A \cup \uparrow A))$
$f(\text{before}(\_A)) = f(\_A \cup \downarrow A))$

$f(\text{after}(\sim u)) = f(\sim \text{after}(u))$
$f(\text{before}(\sim u)) = f(\sim \text{before}(u))$

$$f(\mathrm{after}(u \cap v)) = f(\mathrm{after}(u) \cap \mathrm{after}(v))$$
$$f(\mathrm{before}(u \cap v)) = f(\mathrm{before}(u) \cap \mathrm{before}(v))$$
$$f(\mathrm{after}(\mathrm{after}(u))) = f(\mathrm{after}(\mathrm{before}(u))) = f(\mathrm{after}(u))$$
$$f(\mathrm{before}(\mathrm{after}(u))) = f(\mathrm{before}(\mathrm{before}(u))) = f(\mathrm{before}(u))$$

Then head( u ), introduced in § 3, is interpreted as a set of points which satisfies:

The point's left neighbor is included in $\sim u$ and
it or its right neighbor is included in $u$.

Besides, it is discrete (because interval sequence doesn't accumulate) . Therefore, we can define $f(u \Rightarrow v)$ as follows:

If $f(\mathrm{head}(u)) - f(\mathrm{head}(v)) = \{t1, t2, t3, \cdots\}$ and
$f(\mathrm{head}(v)) - f(\mathrm{head}(u)) = \{s1, s2, s3, \cdots\}$ ,

$$f(u \Rightarrow v) = [(p1, q1), (p2, q2), \cdots]$$

Here $p1 = t1$,
$qn$ is the minimum element of $\{s1, s2, s3, \cdots\}$ which is larger than $tn$
if $n > 1$, $pn$ is the minimum element of $f\,\{t1, t2, t3, \cdots\}$
which is larger than $q_{n-1}$

We have extended the domain of $f$ to the set of all formulas. Since each formula is composed finitely, it can be said that if for every predicate symbol $P$, the edge points of $f(P(x))$ for all $x$ don't accumulate, then for every formula $A$ the edge points of $f(A)$ doesn't accumulate.
When $f(A)$ is $[(-\infty, \infty)]$ , we say $A$ is true under $f$. If for all $f$, $A$ is true under $f$, then we call $A$ to be valid. For example $\neg A \vee A$ is valid. $\neg(\uparrow A \cap {}^{-}A \Rightarrow \phi)$ is another valid formula in AYA, because expression $\uparrow A \cap {}^{-}A$'s interpretation is always empty.

## § 5 Construction of AYA's formal system

From here we use the following abbreviations:

If $u$, $v$ are event expressions, then

| | |
|---|---|
| empty( u ) | is short for $\neg((\mathrm{start} \cap \mathrm{after}(u)) \cup \mathrm{head}(u) \Rightarrow \phi)$ |
| $u \subset v$ | is short for empty$(\sim v \cap u)$ |
| $u \equiv v$ | is short for $u \subset v \wedge v \subset u$ |

In this section we give a system restricted on propositional logic. First we give AYA's axioms and inference rules as follows, then we show its soundness and completeness.

In the following, $u$, $v$, $w$ are event expressions and $A$, $B$ are formulas. A1~7 are axioms and R1~3 are rules.
As usual, we write $\Gamma \vdash A$ if we can get a formula $A$ from a set of formula $\Gamma$ and axioms by R1 ~3. Especially when $\Gamma$ is empty we write $\vdash A$ and call $A$ a theorem of AYA.

| | |
|---|---|
| A1.1 | empty$(\phi)$ |
| A1.2 | $u \subset u \cap u$ |
| A1.3 | $u \cap v \subset u$ |
| A1.4 | $u \cap v \subset v \cap u$ |
| A1.5 | $\sim(u \cap v) \cap (u \cap w) \subset \sim v \cap w$ |
| | |
| A2.1 | $(\uparrow A \cup \downarrow A) \subset (\mathrm{start} \cup {}^{-}(\mathrm{start} \Rightarrow \phi))$ |
| A2.2 | $\mathrm{start} \subset \uparrow(u \Rightarrow v) \cup {}_{-}(u \Rightarrow v)$ |

A3 $\mathrm{empty}(\sim\uparrow A \cap \sim\downarrow A \cap \sim{}^{-}A \cap \sim{}_{-}A)$

A4.1 $\mathrm{empty}(\uparrow A \cap {}^{-}A)$
A4.2 $\mathrm{empty}(\uparrow A \cap \downarrow A)$
A4.3 $\mathrm{empty}(\uparrow A \cap {}_{-}A)$
A4.4 $\mathrm{empty}(\downarrow A \cap {}^{-}A)$
A4.5 $\mathrm{empty}(\downarrow A \cap \downarrow A)$
A4.6 $\mathrm{empty}({}^{-}A \cap {}_{-}A)$

A5.1 $\uparrow\neg A \subset \downarrow A$
A5.2 $\downarrow\neg A \subset \uparrow A$
A5.3 ${}^{-}\neg A \subset {}_{-}A$
A5.4 ${}_{-}\neg A \subset {}^{-}A$

A6.1 $\uparrow(A\rightarrow B) \subset ({}^{-}A\cap\uparrow B) \cup (\downarrow A\cap\uparrow B) \cup (\downarrow A\cap{}_{-}B)$
A6.2 $\downarrow(A\rightarrow B) \subset ({}^{-}A\cap\downarrow B) \cup (\uparrow A\cap\downarrow B) \cup (\uparrow A\cap{}_{-}B)$
A6.3 ${}^{-}(A\rightarrow B) \subset {}_{-}A \cup {}^{-}B \cup (\uparrow A\cap\uparrow B) \cup (\downarrow A\cap\downarrow B)$
A6.4 ${}_{-}(A\rightarrow B) \subset {}^{-}A \cap {}_{-}B$

A7.1 $\uparrow(u\Rightarrow v) \subset ({}_{-}(u\Rightarrow v)\cup\uparrow(u\Rightarrow v)) \cap \mathrm{head}(u) \cap \sim\mathrm{head}(v)$
A7.2 $\downarrow(u\Rightarrow v) \subset ({}^{-}(u\Rightarrow v)\cup\downarrow(u\Rightarrow v)) \cap \sim\mathrm{head}(u) \cap \mathrm{head}(v)$
A7.3 ${}^{-}(u\Rightarrow v) \subset ({}^{-}(u\Rightarrow v)\cup\downarrow(u\Rightarrow v)) \cap (\mathrm{head}(u)\cup\sim\mathrm{head}(v))$
A7.4 ${}_{-}(u\Rightarrow v) \subset ({}_{-}(u\Rightarrow v)\cup\uparrow(u\Rightarrow v)) \cap (\sim\mathrm{head}(u)\cup\mathrm{head}(v))$

Note: As described above, head(u) is
short for $\sim\mathrm{after}(u)\cap(u\cup\mathrm{before}(u))$.

A8.1 $\mathrm{empty}(\mathrm{after}(\uparrow A) \cup \mathrm{before}(\uparrow A) \cup \mathrm{after}(\downarrow A) \cup \mathrm{before}(\downarrow A))$
A8.2.1 $\mathrm{after}({}^{-}A) \equiv {}^{-}A \cup \downarrow A$
A8.2.2 $\mathrm{before}({}^{-}A) \equiv {}^{-}A \cup \uparrow A$
A8.2.3 $\mathrm{after}({}_{-}A) \equiv {}_{-}A \cup \uparrow A$
A8.2.4 $\mathrm{before}({}_{-}A) \equiv {}_{-}A \cup \downarrow A$
A8.3.1 $\mathrm{after}(\sim u) \equiv \sim\mathrm{after}(u)$
A8.3.2 $\mathrm{before}(\sim u) \equiv \sim\mathrm{before}(u)$
A8.3.3 $\mathrm{after}(u\cap v) \equiv \mathrm{after}(u) \cap \mathrm{after}(v)$
A8.3.4 $\mathrm{before}(u\cap v) \equiv \mathrm{before}(u) \cap \mathrm{before}(v)$
A8.4.1 $\mathrm{after}(\mathrm{after}(u)) \equiv \mathrm{after}(u)$
A8.4.2 $\mathrm{after}(\mathrm{before}(u)) \equiv \mathrm{after}(u)$
A8.4.3 $\mathrm{before}(\mathrm{after}(u)) \equiv \mathrm{before}(u)$
A8.4.4 $\mathrm{before}(\mathrm{before}(u)) \equiv \mathrm{before}(u)$
A8.5 $\mathrm{empty}(\mathrm{after}(\mathrm{start}) \cup \mathrm{before}(\mathrm{start}) \cup \mathrm{after}(\phi) \cup \mathrm{before}(\phi))$

R1 If $\vdash A$, then $\vdash\mathrm{empty}(\sim{}^{-}A)$.
R2 If $\vdash\mathrm{start} \subset {}^{-}A$ and $\vdash\mathrm{empty}(\downarrow A)$, then $\vdash A$.
R3 If $\vdash\mathrm{empty}(u)$ and $\vdash v \subset u$ then $\vdash\mathrm{empty}(v)$.

empty(u) semantically represents that 'under every interpretation function u is empty.' ( Actually it is proved in the next section.) Therefore these axioms can be regarded as inclusion relations between the set of time points represented by event expressions.

A2~8 are so formed that event expressions can be decomposed to sub expressions (described later) .

A1 and R3 can be composed from Russell's classical by the following transformation.

'or' to '$\cap$'
'imply' to '$\subset$' in the opposite direction
'true' to '$\phi$'
'$\models$ X' (X is a theorem) to 'empty(X)'

Since the system before transformation is complete, in the system composed by only A1 and R3, which can be got by simply transferring it, we can show the following facts:

> If an event expression w satisfies $g(w)=0$ for every function $g$ from event expressions to $\{0, 1\}$ which satisfies:
>
> $$\begin{array}{ll} & g(\phi)=0 \\ \forall u,v & g(u \cap v)=g(u)g(v) \\ \forall u & g(\sim u)=1-g(u) \end{array}$$
>
> Then $\vdash$empty(w) can be inferred only from A1 and R3.

For example, since $\uparrow A \cap \sim\uparrow A$ satisfies this condition, we can prove $\vdash$empty($\uparrow A \cap \sim\uparrow A$) only from A1, R3. Later, in the proof of this system's completeness, we use this fact without proving.

As you can see, this system has too many axioms and rules. One of the future works is to compose a simpler system composed of much fewer axioms and rules which is equivalent to this one.

Now we list some examples of theorems and rules which can be inferred in AYA.

(1) Let's see the following rule.

> If $\vdash(\uparrow A \cup \uparrow B \Rightarrow \uparrow C) \rightarrow P$, then $\vdash(\uparrow A \Rightarrow \uparrow C) \rightarrow P$.

It can be read as "if P holds from when A or B becomes true until when C becomes true, then P holds from when A becomes true until when C becomes true". If this rule can be proved, the following thing, for example, can be inferred directly (DR1 in (2) above is used) .

> If $\vdash$↑ガス漏れ ∪ ↑漏電 ⇒ ↑正常状態 → 警報,
> then $\vdash$↑ガス漏れ ⇒ ↑正常状態 → 警報.

In other words, we can directly infer "an alarm continues to ring from a gas leak occurs until restoration" from the fact that "an alarm continues to ring from a gas leak or an electric leak occurs until restoration".

Actually, this rule can be proved from A1, 6, 7 etc.

Like this, we can directly express and infer facts about time intervals and predicates hold on intervals.

(2) The following theorems and rule can also be proved.

T1 $A \rightarrow (B \rightarrow A)$
T2 $(A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))$
T3 $(\sim B \rightarrow \sim A) \rightarrow (A \rightarrow B)$

DR1 If $\vdash A$, $\vdash A \rightarrow B$ then $\vdash B$

To show an example, we give a proof of T1. Theorems and rules which can be inferred only from A1 and R3 are used without proof.

1 $\vdash {}^{-}A \cap {}^{-}B \cap \downarrow A \subset {}^{-}A \cap \downarrow A$ (A1, R3)
2 $\vdash$ empty(${}^{-}A \cap \downarrow A$) (A4)
3 $\vdash$ empty(${}^{-}A \cap {}^{-}B \cap \downarrow A$) (1, 2, R3)

4, 5 can be proved like 3.
4 $\vdash$ empty(${}^{-}A \cap \uparrow B \cap \downarrow A$)
5 $\vdash$ empty(${}^{-}A \cap \uparrow B \cap {}_{-}A$)

6 If $\vdash$ empty(u), $\vdash$ empty(v), $\vdash$ empty(w) then $\vdash$ empty($u \cup v \cup w$)

(A1, R3)

7 $\vdash \mathrm{empty}({}^{-}A\cap{}^{-}B\cap\downarrow A \cup {}^{-}A\cap\uparrow B\cap\downarrow A \cup {}^{-}A\cap\uparrow B\cap{}_{-}A)$
(3,4,5,6)

8 $\vdash \downarrow(B\to A) \subset {}^{-}B\cap\downarrow A \cup \uparrow B\cap\downarrow A \cup \uparrow B\cap{}_{-}A$ (A6)

9 If $\vdash t\subset u\cup v\cup w$ then $\vdash s\cap t \subset s\cap u \cup s\cap v\cup s\cap w$
(A1, R3)

10 $\vdash {}^{-}A\cap\downarrow(B\to A) \subset {}^{-}A\cap{}^{-}B\cap\downarrow A \cup {}^{-}A\cap\uparrow B\cap\downarrow A \cup {}^{-}A\cap\uparrow B\cap{}_{-}A$ (8, 9)

11 $\vdash \mathrm{empty}({}^{-}A\cap\downarrow(B\to A))$ (7, 10)

12, 13 can be proved like 11.

12 $\vdash \mathrm{empty}(\uparrow A\cap\downarrow(B\to A))$

13 $\vdash \mathrm{empty}(\uparrow A\cap{}_{-}(B\to A))$

14 $\vdash \mathrm{empty}({}^{-}A\cap\downarrow(B\to A) \cup \uparrow A\cap\downarrow(B\to A)\cup \uparrow A\cap{}_{-}(B\to A))$
(11,12,13,6)

15 $\vdash \downarrow(A\to(B\to A)) \subset {}^{-}A\cap\downarrow(B\to A) \cup \uparrow A\cap\downarrow(B\to A) \cup \uparrow A\cap{}_{-}(B\to A)$ (A6)

16 $\vdash \mathrm{empty}(\downarrow(A\to(B\to A)))$ (14, 15, R3)

17, 18 can be proved similarly to 16.

17 $\vdash \mathrm{empty}({}_{-}(A\to(B\to A)))$

18 $\vdash \mathrm{empty}(\uparrow(A\to(B\to A)))$

19 $\vdash \mathrm{empty}(\sim t\cap\sim u\cap\sim v\cap\sim w)$, $\vdash \mathrm{empty}(t)$, $\vdash \mathrm{empty}(u)$, $\vdash \mathrm{empty}(w)$ then $\vdash \mathrm{empty}(\sim v)$ (A1, R3)

20 $\vdash \mathrm{empty}(\sim{}^{-}(A\to(B\to A)))$ (16~19, A3)

21 $\vdash \mathrm{start}\subset {}^{-}(A\to(B\to A))$ (20, A1.3)

22 $\vdash A\to(B\to A)$ (16, 21)

As described above, AYA includes the classical propositional logic.

## § 6 Completeness and soundness of the formal system

Theorem 1 If $\vdash A$, then $A$ is valid.

Proof

Note the following fact. (We omit the proof)

<1> For every event expression $u$,
if $\exists t<0 \ \ t\in f(u)$, then $\forall t<0 \ \ t\in f(u)$.

First, we prove:

<2'> Suppose that $f$ is an interpretation function. $f(u)$ is an empty set iff $f((\mathrm{start} \cap \mathrm{after}(u)) \cup \mathrm{head}(u))$ is empty.

To show "only if" is easy, so we show the "if" part. If $f((\mathrm{start} \cap \mathrm{after}(u)) \cup \mathrm{head}(u))$ is empty, $f(\mathrm{start} \cap \mathrm{after}(u))$ is, too. From <1> , $f(u)$ is a subset of $\{t \mid 0\leqq t\}$ . From this and the fact that $f(\mathrm{head}(u))$ is empty, using non-accumulate property of $f(u)$, we get the conclusion that $f(u)$ is empty.

From <2'> we can get:

<2> For an interpretation function f, f(u) is empty iff
f((start $\cap$ after(u)) $\cup$ head(u) $\Rightarrow \phi$) is 〔 〕 (empty interval sequence),
in other words empty(u) is true under f.

Next we prove that A1$\sim$8 are all valid. All of them are in one of these forms: $\vdash$empty(u) or $\vdash$empty(u)$\wedge$empty(v). So it's sufficient to check "$\forall$f, f(u) and f(v) are both empty" for all of A1$\sim$8.
For A1 it is clear. For A2.1, it can also be said, because f($\uparrow A \cup \downarrow A$) doesn't have any time point before 0, from the definition. Now see A2.2. Since f(head(u)) doesn't have any point before 0, if f(u$\Rightarrow$v)= 〔(p1, q1),…〕 then 0$\leqq$p1. Therefore f(start), i.e. {0}, is included in f($\uparrow$(u$\Rightarrow$v)$\cup$ $_{-}$(u$\Rightarrow$v)).
Other axioms can be checked according to the extended definition of interpretation function.

Thus the soundness of axioms has been proven. From here we prove the soundness of rules. R3 and R1 are clear. For R2, if start$\subset {}^{-}A$ is valid, we can get f(${}^{-}A$)$\ni$0 from <2>. Therefore if f(A)= 〔(p1,q1),…〕, p1 must be $-\infty$. Besides, if empty($\downarrow$A) is valid, f($\downarrow$A) is empty and then q1=$\infty$. Thus A is valid. It finishes the proof of the soundness of rules.
Therefore this system's soundness is warranted.
(End of proof)

Theorem 2 If A is valid, $\vdash$A.

The outline of the proof is as follows: first, we show that for every event expression u, empty(start $\cap$ u) a theorem if it is valid. From this we can show that if ${}^{-}A$ at 0, then we can infer $\vdash$start $\subset {}^{-}A$ from the axioms and rules. Next, for all event expression in the form of $\downarrow$A, we show that if empty($\downarrow$A) is valid we can reduce its proof into a proof of a formula which is in the form of empty(start $\cap$ u) or a formula inferable from only A1 and R3.
If both $\vdash$start $\subset {}^{-}A$ and $\vdash$empty($\downarrow$A) are proven, we can get $\vdash$A from R3.

To begin with, note that if $\vdash u \equiv v$ then $\vdash u \subset v$ and $\vdash v \subset u$. It is because AYA includes classical predicate logic, so if $\vdash A \wedge B$ then $\vdash$A and $\vdash$B. Besides, as a binary relation of u and v, $\vdash u \subset v$ is transitive (from A1 and R3). Thus, as a binary relation of two event expressions, $\vdash u \equiv v$ is an equivalence relation. From here we say that u and v are equivalent if $\vdash u \equiv v$.

Lemma 1

If we replace $\subset$ in A5,6,7 with $\equiv$, they remain being theorems.

Proof

Here we show the proof of A5.1 only. Others can be proved similarly.

1 $\vdash \downarrow A \subset \sim\uparrow A \cup \sim{}^{-}A \cup \sim{}_{-}A$ (A3, A1, R3)
2 $\vdash \sim\uparrow A \subset \sim\downarrow\neg A$, $\vdash \sim{}^{-}A \subset \sim{}_{-}\neg A$, $\vdash \sim{}_{-}A \subset \sim{}^{-}\neg A$
(A5, A1, R3)
3 $\vdash \sim\uparrow A \cup \sim{}^{-}A \cup \sim{}_{-}A \subset \sim\downarrow\neg A \cup \sim{}_{-}\neg A \cup \sim{}^{-}\neg A$
(2, A1, R3)
4 $\vdash \sim\downarrow\neg A \cup \sim{}_{-}\neg A \cup \sim{}^{-}\neg A \subset \uparrow\neg A$ (A3, A1, R3)
5 $\vdash \downarrow A \subset \uparrow\neg A$ (1, 3, 4, A1, R3)
6 $\vdash \downarrow A \equiv \uparrow\neg A$ (1, 5, A1, R3)
(End of proof)

From here we replace A5,6,7 with themselves after the previous replacement. In preparation for lemma 2, we define sub expression, sub formula as follows:

1.1) u is a sub expression of u
1.2) If w is a sub expression of u,
w is also a sub expression of u$\cap$v, v$\cap$u, $\neg$u.

1.3) If w is a sub expression of u,
w is also a sub expression of after(u), before(u).

2.1) A is a sub formula of A
2.2) If C is a sub formula of A,
C is also a sub formula of $A \wedge B$, $B \wedge A$, $\neg A$.
2.3) If one of $\uparrow A$ • $\downarrow A$ • $^{-}A$ • $\_A$ is a sub expression of u, and
C is a subformula of A,
C is also a subformula of $u \Rightarrow v$, $v \Rightarrow u$.

Lemma 2

Let t, t' be event expressions and assume that we can get t' from t by replacing a sub expression u of t with u'.
If $\vdash u \equiv u'$, $\vdash t \equiv t'$.

Proof

We use induction related to the structure of event expression.
It is easy to show that if $\vdash u \equiv u'$ then $\vdash \sim u \equiv \sim u'$, $\vdash u \cap v \equiv u' \cap v$, $\vdash v \cap u \equiv v \cap u'$. Now we prove if $\vdash u \equiv u'$ then $\vdash$before(u)$\equiv$before(u'). It can be done as follows: by induction of the structure of event expression, for every u there is an event expression v which is composed without using "after" or "before" and satisfies before(u)$\equiv$v. Similarly, before(u')$\equiv$v' for a v'. If $\vdash u \equiv u'$, then $u \equiv u'$ is valid (theorem 1), thus $\forall f$ f(u) = f(u'). Then we can check f (before(u))= f(before(u')) from the definition. Thus before (u)$\equiv$before(u') is valid, so $v \subset v'$ and $v' \subset v$ are also valid. v • v' are constructed without using "before" • "after", so $v \subset v'$ and $v' \subset v$ can be inferred only from A1 • R3. $\therefore \vdash$ before(u)$\equiv$before(u'). About "after", we can make a similar proof.
(End of proof)

Lemma 3

Empty(start $\cap$ u) is valid iff it is a theorem.

Proof

The "if" part can be shown by theorem 1, so we show the "only if" part. From A2.2 and A7 we can prove:

$\vdash$(start $\cap$ $\uparrow$(u$\Rightarrow$v)) $\equiv$ start $\cap$ head(u) $\cap$ $\sim$head(v)
$\vdash$(start $\cap$ $\downarrow$(u$\Rightarrow$v)) $\equiv$ $\phi$
$\vdash$(start $\cap$ $^{-}$(u$\Rightarrow$v)) $\equiv$ $\phi$
$\vdash$(start $\cap$ $\_$(u$\Rightarrow$v)) $\equiv$ start $\cap$ $\sim$head(u) $\cup$ head(v)

Using these and A5 • 6 • 8, we apply lemma 2 recursively. Then, by induction of structure of a formula, we can get an event expression e which is equivalent to (start $\cap$ u), and composed from only "start" and event expressions in the form of $\uparrow A$ • $\downarrow A$ • $^{-}A$ • $\_A$ by $\sim$ • $\cap$. If empty(start $\cap$ u) is valid, so is empty(e), thus it can be proved only from a1 and r3. Therefore (start $\cap$ u), equivalent to e, is also provable.
(End of proof)

From this and <2> which is used in proof of theorem 1, the following fact can be shown:

Lemma 3'

Empty(start $\cap$ u) is a theorem iff for every f, f(start $\cap$ u) is empty.

(Proof is omitted)

Now, completeness restricted on formulas in the form of empty(start $\cap$ u) has been shown. Our next aim is to replace $\downarrow$A with an event expression u, which equivalent to $\downarrow$A and satisfy that "empty(u) is provable".

Lemma 4

for every formula A which satisfy $\vdash$start$\subset\bar{}$A,
there exists an event expression e in the form of the following.

e = w1$\cup$w2$\cup\cdots\cup$wn (n can be 0. If so, e = $\phi$)

Hereupon

$w_k$ is $u_{k1}\cap\cdots\cap u_{ki_k}\cap v_{k1}\cap\cdots\cap v_{kj_k}$

ukn is (_B$\cup\uparrow$B), B is A's sub formula ($\neg$ may be added)

and if $n_1\neq n_2$ then $u_{kn_1}\neq u_{kn_2}$

The form of vkm is one of $\uparrow$C, $\downarrow$C, $\bar{}$C, _C
where C is A's sub atomic formula

if vkm is $\uparrow$C or _C then {uk1, uk2,$\cdots$} $\ni$(_C$\cup\uparrow$C)

if vkm is $\downarrow$C or $\bar{}$C then {uk1, uk2,$\cdots$} $\ni$(_$\neg$C$\cup\uparrow\neg$C)

Proof

Using A5,6,7 (as described above, $\subset$'s have been replaced with $\equiv$'s) and A8, apply lemma 1 to $\downarrow$A recursively. However, do not apply it any more to sub expressions in the form of ($\uparrow$B$\cup$_B) or ($\downarrow$B$\cup\bar{}$B) which is got by applying A7. This process terminates finitely and we get an event expression e' which is equivalent to $\downarrow$A and composed only from the following constituents bound by $\sim$ and $\cap$.

S = {Start, $\phi$, $\uparrow$C, $\downarrow$C, $\bar{}$C, _C, $\uparrow$B$\cup$_B, $\downarrow$B$\cup\bar{}$B |
B is A's sub formula, C is A's sub atomic formula}

Besides, we can get e" in the form of the following, which is equivalent to $\downarrow$A, by A1,R3 and lemma 1.

e"= w'1 $\cup$ w'2 $\cup\cdots$

Here w'k = uk1 $\cap$ uk2 $\cap$ uk3 $\cap\cdots$ uij $\in$ S

After these replacements, if there is $\sim$start in {uk1, uk2, $\cdots$}, omit it. Moreover, if there is "start" in it, omit wk itself. The reason why we can do so without losing the equivalency to $\downarrow$A is as follows: from the assumption we get $\vdash$start$\subset\bar{}$A, thus $\vdash$empty(start $\cap\downarrow$A) and $\vdash\downarrow$A$\equiv$start$\cap\downarrow$A, which lead to $\vdash$empty(start$\cap$w'k) and $\vdash$w'k$\equiv$ start$\cap$w'k.
Next, if there is $\sim\phi$ in {uk1, uk2, $\cdots$} omit it, and if there is $\phi$, omit wk. Then replace any formulas in the form of ($\bar{}$B$\cup\downarrow$B) with (_$\neg$B$\cup\uparrow\neg$B). Further, for every atomic C, if there is $\uparrow$C or _C add (_C$\cup\uparrow$C), and if there's $\downarrow$C or $\bar{}$C, add (\_$\neg$C$\cup$ $\uparrow\neg$C).
Last, omit duplicate elements of {uk1, uk2, $\cdots$} and rearrange them into the required order to get e. It's easy to show these operations also keep the equivalency to $\downarrow$A.
(End of proof)

Lemma 5

Suppose that ⊢start⊂¯A.
The necessary and sufficient condition to satisfy "f(↓A) is empty for all f" is as follows:

For every wk which appears in the previous lemma, if we write

$$A_k = B_{k1} \vee B_{k2} \vee \cdots \vee B_{ki_k} \quad \text{here } u_{kn} = {\_}B_{kn} \cup \uparrow B_{kn}$$

and A'=A1∧A2∧…, then the following things hold.

⊢start ⊂ ¯A' and ∀f, f(↓A') is empty

Proof

Since "⊢start ⊂ ¯A' iff f(¯A')∋0", it can be shown using the definition of interval sequence that "f(_A'U↑A') is empty iff both ⊢start ⊂ ¯A' and f(↓A')."
From here we use symbols in common with lemma 4.

$$f(\downarrow A) = f(w1) \cup f(w2) \cdots$$
$$f(w_k) = f({\_}A_k \cup \uparrow A_k) \cap f(v_{k1}) \cap \cdots \cap f(v_{kj_k})$$

Since f(_A'U↑A')=f(_A1U↑A1)U f(_A2U↑A2)U…, we can show that if f(_A'U↑A') is empty, f(↓A) is, too. On the other hand, if f(_A'U ↑A') is not empty for an f, ∃k f(_AkU↑Ak) is not empty. Then there's an f' such that f'(wk) is not empty. The reason is as follows. If vkm is ↑C or _C, (_CU↑C) ∈ {uk1, uk2, …}, and if vkm is ↓C or ¯C, (¯CU↓C) ∈ {uk1, uk2, …}. So, for a p∈f(_A'U↑A') and its sufficiently small right neighborhood N, we can make an interpretation f' from f, which satisfies p∈f'(wk), by changing the interpretation of C (which is atomic) in N. Here, f'(↓A) is not empty, and this finishes the proof.
(End of proof)

Lemma 5'
Suppose ⊢start⊂¯A. ⊢empty(↓A) if ⊢A'. (A' is got in lemma 5)

Proof

Using R1 we can get ⊢empty(_A'U↑A') from ⊢A'. From the definition of A'and A1•R3, ⊢(↓A ⊂ _A'U↑A') can be proven. Thus ⊢empty(↓A).
(End of proof)

Now, suppose that a formula A is valid, Then for all f f(∼¯A ∩ start) is empty, so ⊢start ⊂ ¯A from lemma 3. Therefore, from lemma 5 and 5, whether ⊢A or not is resolved into whether ⊢A', where A'is valid. This process of resolving is applied recursively. If we prove that it terminates finitely, it finishes the proof of theorem 2. In other words, for a formula A (which may not valid), we recursively apply that process and get a sequence A, A', A"…. (From here we abbreviate A'''…' with n apostrophes to A(n).) If A isn't valid, then for a number n, we can know that ⊢start ⊂ A(n) doesn't hold. Otherwise, at some time, e (which is got in lemma 4) will be φ and ⊢A is proven.

Lemma 6
1 A(n) is composed of A's sub formulas by ∧ and ¬ only.

2 If ⊢start ⊂ ¯A(n) and ⊢start ⊂ ¯A'(n) then

⊢(start⇒↓A(n)) → (start⇒↓A(n-1))
⊢↓(start⇒↓A(n)) ⊂ ¯(start⇒↓A(n-1))

Proof

1
For A' it is clear from how to make it. Besides, it can be easily checked that if it holds for A(n-1) it holds also for A(n).

2
It's sufficient to check about A and A'.

We can get $\vdash(\text{start}\Rightarrow\downarrow A')\rightarrow(\text{start}\Rightarrow\_A'\cup\uparrow A')$ easily. Since $\vdash\downarrow A \subset \_A'\cup\uparrow A'$, $\vdash(\text{start}\Rightarrow\_A'\cup\uparrow A')\rightarrow(\text{start}\Rightarrow\downarrow A)$ …(1) can also be proven. From this, $\vdash(\text{start}\Rightarrow\downarrow A')\rightarrow(\text{start}\Rightarrow\downarrow A)$. Further, from (1), $\vdash\downarrow(\text{start}\Rightarrow\downarrow A')\subset{}^{-}(\text{start}\Rightarrow\downarrow A)\cup\downarrow(\text{start}\Rightarrow\downarrow A)$. Using $\vdash\downarrow A\subset\_A'\cup\uparrow A'$ again, $\vdash\text{empty}(\downarrow A\cap\downarrow A')$ can be proven, so $\vdash\downarrow(\text{start}\Rightarrow\downarrow A')\subset(\text{start}\Rightarrow\downarrow A)$.
(End of proof)

It has been already checked that AYA includes the classical propositional logic. Let us write E==F when we can say "$\vdash$E iff $\vdash$F" using only classical logic. Then == is an equivalence relation. Since A(n) is composed of A's sub formula by only $\wedge$ and $\neg$, there exists a formula A'(n) which satisfies:

| | |
|---|---|
| $A'(n)=E1\wedge E2\wedge\cdots$ | If $i\neq j$ then $Ei\neq Ej$ |
| $En=Fn1\vee Fn2\vee\cdots$ | Fnm is A's sub formula ($\neg$ may be added) |
| | If $i\neq j$ then $Fni\neq Fnj$ |
| | (Both En's and Fnm's are in suitable lexicographical order) |
| A'(n) == A(n) | |

Suppose that A has k sub formulas (they must be finite) . The set of formulas which are composed of A's sub formulas by $\wedge$ and $\neg$ is divided into at most k' equivalence classes by ==, where

$$k' = 2^{k}.$$

Now we assume that the resolving process described above continues infinitely without proving "not $\vdash(\text{start}\Rightarrow A(n))$", and lead to contradiction. For some n and m ($n<m$) , A(n) == A(m), because equivalence class of $\{A(n)\mid n\geqq 0\}$ by == is finite. If E == F, $\vdash(\text{start}\Rightarrow\downarrow E)\rightarrow(\text{start}\Rightarrow\downarrow F)$, thus from 2 and 3 of lemma 6, $\downarrow(\text{start}\Rightarrow\downarrow A(m))\subset{}^{-}(\text{start}\Rightarrow\downarrow A(m))$ is a theorem, although it is not valid. It contradicts to the theorem 1.
Therefore the process of resolving terminates finitely. Especially if A is valid, we can infer $\vdash$A in finite application of rules.
(End of proof of theorem 2)

The following theorem also holds, but we omit the proof.

Theorem 3

If $\Gamma$ is a set of formula and A, B are formulas.
If $\Gamma\cup\{A\}\vdash B$, then $\Gamma\vdash A\rightarrow B$.

In this section we've proven completeness and soundness of the system restricted on propositional logic, but in the following sections, for convenience we expand it into a predicate logic by adding following axioms and rules. The proof of soundness can also be done similarly after expansion. Completeness is expected to be proven similarly, but it is one of the future works.

A9.1 $\forall xA(x)\rightarrow A(t)$
A9.2 $A(t)\rightarrow\exists xA(x)$

R4.1 If $\vdash B\rightarrow A(a)$ then $\vdash B\rightarrow \forall x A(x)$
R4.2 If $\vdash A(a)\rightarrow C$ then $\vdash \exists x A(x)\rightarrow C$

Here 'a' is a free variable and B is a formula which doesn't contain 'a'. A(t) is a formula made by substituting all a's in A(a) by a term t. $\exists xA(x)$, $\forall xA(x)$ are made by substituting a's in A(a) by a binding variable x and quantifying.
In § 8 we further assume the Peano's axioms of natural numbers and the following rule about the equal sign.

If $t1=t2$, then $\vdash A(t1)\rightarrow A(t2)$

§ 7 How to execute

In the original first order predicate logic, by regarding a set of Horn clause as a program we can make up a programming language Prolog. In a similar way, we can use AYA as a real-time programming language.

First we take a set D where

$D=\{F\rightarrow A \mid F$ is a formula (may be empty) , $A$ is an atomic formula$\}$

A program P is a finite subset of D.
If under an interpretation f, all formulas of P are true, then we call f a model of P. An execution of P is to make a model of P. It is possible to make a model in turn according to real-time external events, and we show an example of it in § 8. In this way AYA is real-time executable.

For a set P, let S be a set of all predicate symbols of all atomic formulas in P, and let V be $\{\uparrow p, \downarrow p \mid p\in S\}$. We call every element of V an event symbol. If $A=p(x1,x2\cdots)$, we say $\uparrow p$ and $\uparrow A$ correspond. $\downarrow p$ and $\downarrow A$ also correspond.

When we do a real-time execution, A subset of events which correspond to each element of V is the set of external events, and the rest are internal events.

Lemma 1
Suppose that $F1\rightarrow A, \cdots, Fn\rightarrow A$ are all elements of P which has a goal A. Both $\vdash\uparrow A\subset {\_}F\cup\uparrow F$ (where $F=F1\vee F2\vee\cdots$) and $\vdash\downarrow A\subset {\_}F\cup\downarrow F$ can be inferred from P.

It can be proved simply by applying A6 and A1 recursively. We call $\uparrow F$ the deciding event of $\uparrow A$. $\downarrow F$ is $\downarrow A$'s deciding event.

By this lemma, when we determine states of all atomic formula at the origin of time axis (i.e. for every atomic A, decide which one of $\uparrow A$, $\bar{}A$, ${\_}A$, $\downarrow A$ includes the time point 0) so that they don't conflict, and give each moment of occurrence of external events in turn, then we can make a model of P by the following way.

1) For every event symbol of internal event,
   find the deciding event of the predicate corresponding to it.
2) When an external event occurs,
   for each atomic formula A, do:

   if $\neg A$ holds in a left neighbor of current time, reason whether the decide event of $\uparrow A$ occurs, and if so, conclude $\uparrow A$.
   if A holds in a left neighbor of that point, do the same thing about $\downarrow A$.
   But if $\uparrow A$ and $\downarrow A$ are not internal, it is not necessary.

Here in 2), axioms and rules given in §5 are used. Let $\uparrow F$ be $\uparrow A$'s deciding event. At a time p, we conclude $\uparrow A$ if the following condition is satisfied.

> If we assume empty( $\_B \cup \uparrow B$ ) for every formula B which holds in a left neighbor of p, empty( $\sim \uparrow F$ ) can be inferred.

By this, truth values of all atomic formulas can be decided. In the case of real-time execution, according to the passage of time, external events come in order, and truth values are decided real-time. Truth values of non atomic formulas can be decided using those of atomic formulas.

Reasoning in 2) only have to be done when at least one external event occurs. Besides, we only have to keep the information of truth values of the time just before now. We don't have to store up all the histories of truth values since the program starts.

Let us think whether a model made in this way is least in a suitable meaning.
We introduce an order between models as follows. It is essentially equivalent to that of [5].
First, for F which is got from A as in lemma 1, we can make an event expression which is equivalent to $\uparrow F$ as follows:

$$\uparrow F \equiv w1 \cup w2 \cup \cdots \cup wn$$

$w_k$ is $u_{k1} \cap \cdots \cap u_{ki_k} \cap v_{k1} \cap \cdots \cap v_{kj_k}$

ukn is ( $\_B \cup \uparrow B$ )、B is F's sub formula ( $\neg$ may be added)

$n_1 \neq n_2$ then $u_{kn_1} \neq u_{kn_2}$

vkm is on of $\uparrow C, \downarrow C, \bar{~}C, \_C$ where C is F's sub atomic formula

If vkm is $\uparrow C$ or $\_C$ then $\{uk1, uk2, \cdots\} \ni (\_C \cup \uparrow C)$
If vkm is $\downarrow C$ or $\_C$ then $\{uk1, uk2, \cdots\} \ni (\_\neg C \cup \uparrow \neg C)$

In addition, if event symbols corresponding to $\uparrow C \bullet \downarrow C \bullet \uparrow A$ are respectively p, p', q, we define two order between event symbols as follows: $p \ll + q$, $p' \ll - q$.

We can apply similar process on $\downarrow F$. In this case, if event symbols corresponding to $\uparrow C \bullet \downarrow C \bullet \uparrow A$ are respectively p, p', q, we define $p \ll - q$, $p' \ll + q$.

If we write the transitive closure of ( $\ll +$ or $\ll -$ ) as $\ll$, then it is a partial order. $\ll$ can be regarded as a dependency relation between event symbols. Further, if we define $p = q$ iff $p \ll q$ or $q \ll p$, it is an equivalence relation.
Let p1, p2, …, pk be equivalence classes of V's internal events by $=$, and suppose that if $i<j$ then not $pj \ll pi$. Further, let p0 be the set of external events of V. Now we define a partial order of models as follows.

For two models f, f', if the following condition holds, then $f < f'$.

> $\exists i, t \; 0 \leqq i \leqq k \;\; t \in \{t \mid 0 \leqq t\}$ ( t represents time point)
> Interpretations of all atomic formulas by f, f' are same until the time point t. In a sufficiently small right neighbor of t, truth values of predicates corresponding to event symbols inherent in p0, p1, …, $p_{i-1}$ are same under f and f', and a predicate corresponding to an event symbol p which is inherent in pi is false under f, true under f'.

Then the following condition is estimated to be an sufficient condition that the model which is constructed as above is least.

For no event symbols p, q inherent to pi, p≪- q.

A proof is expected to be done like in [5].

## §8 An example of real-time programming

Let's go back to the example of elevator controlling. This time we give more detailed description to show an example of real-time execution.

To begin with, we describe the detail of the elevator system.
The elevator has a box which carries people up and down from the 1st (ground) floor to the k th floor.
At each floor there are two buttons—up-call and down-call, and in the box there are destination buttons corresponding to each floor.
Predicates "上呼び押(n)" "下呼び押(n)" "行き先押(n)" represent that each button is in the state 'pushed'. Predicates "上向き" "下向き" denote the direction of the elevator.
"停止中(n)" expresses that the elevator stops at the n-th floor now and "移動中(m,m+1)" the elevator is now moving upward from m-th floor to m+1-th. If it is moving downward, "移動中(m+1, m)" holds. In practice, just one of these three holds at every time. If |n－m|≠1, 移動中(n,m) never holds. "ドア開" denotes the door of the elevator is now open.
"行かねば(n)" represents the elevator must go to the n-th floor. "より上(n)","より下(n)" indicate whether the n-th floor is higher or lower than the current position of the elevator.
External events are "↑上呼び押(n)","↑下呼び押(n)","↑行き先押(n)", "↓ドア開" and "↓移動中(n,m)". In other words, instructions of destination or calling come from external devices. A process which shuts the door is separated from this program. Furthermore, since this program does not control the speed of the elevator, the information that "it finished passing a floor" comes as an external event, "↓移動中(n,m)".
To make the program simple, it doesn't check the arguments of "↑上呼び押(n)","↑下呼び押(n)","↑行き先押(n)".

Program (Quantifiers of free variables are omitted)

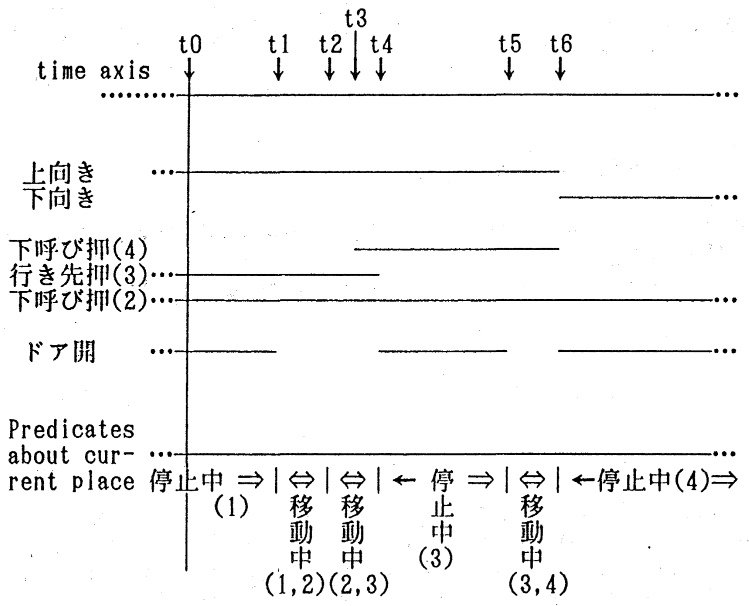
```
↑上呼び押(n) ⇒ ↑停止中(n)∩ ¯上向き                        → 上呼び押(n)
↑下呼び押(n) ⇒ ↑停止中(n)∩ ¯下向き                        → 下呼び押(n)
↑行き先押(n) ⇒ ↑停止中(n)                                  → 行き先押(n)

↓(移動中(x,n)∧行かねば(n)) ⇒ ↑((上需要有∨下需要有)∧¬ドア開)
                                                            → 停止中(n)  ※1
↓停止中(n)∩ ¯上向き ∪ ↓(移動中(n-1,n)∧～行かねば(n)) ⇒ ↓移動中(n,n+1)
                                                            → 移動中(n,n+1)
↓停止中(n)∩ ¯下向き ∪ ↓(移動中(n+1,n)∧～行かねば(n)) ⇒ ↓移動中(n,n-1)
                                                            → 移動中(n,n-1)

行き先押(n)                                                 → 行かねば(n)
下呼び押(n) ∧ ～(行かねば(m)∧ m>n)                         → 行かねば(n)
上呼び押(n) ∧ ～(行かねば(m)∧ m<n)                         → 行かねば(n)

(停止中(m)∨移動中(m,m+1)∨移動中(m+1,m)) ∧ m<n              → より上(n)
(停止中(m)∨移動中(m,m-1)∨移動中(m-1,m)) ∧ m>n              → より下(n)

行かねば(n) ∧ より上(n)                                     → 上需要有
行かねば(n) ∧ より下(n)                                     → 下需要有
↓下向き ⇒ ↑(～上需要有 ∧ 下需要有)                         → 上向き
↓上向き ⇒ ↑(～下需要有 ∧ 上需要有)                         → 下向き
```

↑停止中(n) ⇒ ↓ドア開　　　　　　　　　　　　→ ドア開 ※2

---

For example formula※1 expresses that if the box finishes moving to n-th floor and it must stop there, it stops until the door is closed and the elevator is in the state that it is called from another floor. By ※2 "ドア開" holds from it stops until the door closes. (As described above, the process which closes the door is actually separated to other controlling unit. )
After that, when an external event "the door is closed" comes, "¬ドア開" becomes true. Then if it is called from another floor, the interval on which "停止中(n)" ends and the elevator begins to move. Otherwise, it remains there until called from another floor.

Execution of the program shown above is as follows. In the following figure, each predicate holds on intervals indicated by lines.



As described in §7, execution is performed by making deciding events of all internal events and reasoning whether they occur at each time when external events occur.
The origin point of the time axis is t0, at which the program starts. In the previous figure, at the initial state the box is at the 1st floor and the door is open, in addition the downward button at the 2nd floor and the destination button of the 3rd floor is pushed. (If you want to specify the initial state in the program, you may write, for example, start ⊂ ¯ ドア開. )
At t1, the door closes. Since an external event occur, reasoning about internal events is performed.
For example let us see whether ↓停止中 occurs. the deciding event of it is:

↓(↓(移動中(x,n)∧行かねば(n)) ⇒ ↑((上需要有∨下需要有)∧¬ドア開))

While reasoning whether it occurs, it is checked whether ↓(上需要有∨下需要有) occurs. Since deciding events of ↓上需要有 and ↓下需要有 only occur when the elevator arrives at some floor, ↓(上需要有∨下需要有) does not occur at this time.
From this ↑((上需要有∨下需要有)∧¬ドア開) is inferred. Therefore by the formula ※1, ↓停止中(1) occurs, and by the formula in the next line of ※1, ↑移動中(1,2) also occurs. That is to say, the elevator begins to move upward.
When the elevator approaches the 2nd floor (at t2) , the external event ↓移動中(1,2) causes a reasoning whether it stops there. In the left neighbor of that time, since 行き先押(3) holds, 行かねば(3) also holds, so 行かねば(2) doesn't hold. Thus an event ↓(移動中(1,2)∧~行かねば(2)) occurs, then ↑移動中(2,3) occurs and the elevator passes the 2nd floor without stopping.
And so on. The elevator stops at the 3rd floor, then at t6 arrives at the 4th floor. Since

there are no longer any destinations above, ↓上向き occurs and the elevator changes its direction to downward.

As is clear from this example, in AYA it isn't required that both standing up wand down of a predicate's truth value are external. One of the examples which actively use this point is the description of "移動中(n,n+1)". Standing up of it is an internal event which occurs by other events such as closing of the door or called from other floor, but it stands down by an external signal. From another point of view, it expresses that on an adjoining interval to a particular time interval (in this case an interval on which the elevator stops at the n-th floor of moving to n-th floor), "移動中(n,n+1)" holds. That is to say, we can express a similar state to one which can be expressed by a modal operator R which expresses "holds on the right-next interval", for example "停止(n)→ R 移動中(n, m)".

Since AYA's program is a set of formulas, other than real-time executing we can also think about possibility of static reasoning. For example let us think about a program which is got by a little change of a formula in the previous program.

change ※2 to ↑停止中(n) ⇒ ↓ドア開 ←→ ドア開
where A←→B is short for A→B∧B←A.
(Executed by a method in §7, actually it also holds.)

Then ドア開→停止中(n) can be inferred. In other words it is guaranteed from the program itself that the elevator stops while the door is open.
Like this, it is possible to use AYA as a tool of checking programs written in AYA itself.

## §9 Conclusion

In this paper we formulated a system of interval logic, named AYA. In AYA, truth values of predicates are changed according to events which occur at time points. One of its main feature is that relations between time intervals such as "includes", "next to" can be naturally expressed and treated. Besides, completeness of the system (at least on propositional logic) can be shown. Moreover, it can be used as a real-time programming language and executed efficiently.
One of the future works is realization of its processor. In the example in §8 we expanded the system given in §5 into an predicate logic for the present, but we have to formalize the expansion more closely before making a real system. Besides, when making a processor in practice, some weak points of this language must be improved. For example, in AYA events are restricted to changes of predicates' truth values. Other kinds of event may be needed if we write event-oriented real-time controlling program. For example, "at a certain time wake up a process", "send a signal some minutes after", and so on. [6] includes an element to express the latter, but This paper omit it to make the system simple.
To simplify the system is another important work. Currently AYA's formal system is so complicated. If we can compose a system of rules and axioms such as (1) in §5, it will be more simple and easy to grasp intuitively. Further, proof of completeness for predicate logic is also left.
Besides them, it is also hoped to generalize the definitions of time point or interval, etc. In designing AYA, we consider not only executability but also formalization as a tense logic. One of the merits of [1] and [2] is that they compose formal systems without putting strong restriction on the definition of time, but currently AYA puts a strong restriction on the set of time point and time interval. As an executable language, it is not so serious, but as a language for time representation, there is a room for improvement in the power of expression, as described in the latter part of §2.

## Acknowledgement

The author express his deep gratitude to Prof. Reiji Nakajima, Takashi Sakuragawa, Masami Hagiya, Takashi Hattori, Takashi Susuki for their appropriate suggestions and introductions to beneficial literatures.

## References

[1] I.L.Humberstone: "INTERVAL SEMANTICS FOR TENSE LOGIC",
Journal of Philosophical Logic 8

[2] Peter Roper: "INTERVALS AND TENSES",
Journal of Philosophical Logic 9

[3] 米崎直樹・新淳・蓬萊尚幸: "時制論理プログラミング言語Templog",
日本ソフトウェア科学会第１回大会論文集

[4] Ben Mozkowski: "A Temporal Logic for Multilevel Reasoning about Hardware",
COMPUTER, February 1985

[5] 桜川貴司: "Temporal Prolog",
コンピュータ・ソフトウェア Vol.4 No.3, 1987

[6] 桜川貴司: "タイムポイント・ロジックとインターバル・ロジック" (仮題),
To appear

[7] 桜川貴司・竹中一起・中島玲二・新出尚之・服部隆志:
"RACCO: 実時間プロセス制御システムのモデル記述のための様相論理プログラミング言語",
コンピュータ・ソフトウェア, Vol.5 No.3, 1988