Modal μ-calculus on min-plus algebra $\mathbb{N}_\infty$ [*†]

Dai IKARASHI[‡] Yoshinori TANABE[‡,§] Koki NISHIZAWA[¶] Masami HAGIYA[‡,‖]

‡: Graduate School of Information Science and Technology, The University of Tokyo
§: Research Center for Verification and Semantics,
National Institute of Advanced Industrial Science and Technology (AIST)
¶: Tohoku University
‖: NTT Communication Science Laboratories

Abstract

We give an interpretation of modal μ-calculus using min-plus algebra $\mathbb{N}_\infty$, the set of all natural numbers and infinity ∞. Disjunctions are interpreted by min, and conjunctions by plus. By this interpretation, simple formulas can express the shortest path on a Kripke structure, the number of states that satisfy a specified condition, etc. In this paper, we define the semantics of modal μ-calculus on min-plus algebra, and then describe a model checking algorithm for the semantics and its implementation. Although simple iterative computation of a least fixed point in general does not terminate in $\mathbb{N}_\infty$, thanks to abstraction, we can make model checking possible by reducing least fixed point computation to greatest fixed point computation. In addition, we try to apply the semantics to data flow analysis for compiler optimization. Finally, we discuss relationship between our semantics and the theory of Kripke structures on complete Heyting algebra.

1 Introduction

Modal logic can express various kinds of properties on Kripke structures. Especially, modal μ-calculus, which has fixed point operators, can express various crucial properties on Kripke structures, such as reachability and existence of infinite paths, accurately and simply by formulas [5].

To enhance expressiveness of this modal logic, some research has been conducted to define semantics that interprets formulas on algebra that has richer structures than those in the ordinary semantics, i.e., the semantics that interprets formulas as true or false.

For example, in order to formalize multiplexed model checking, Nishizawa et al. investigated the simulation relation, the relation between state formulas and path formulas, etc. on extended Kripke structures that assign elements of complete Heyting algebra as truth values to propositions and transition relations in contrast to the ordinary Boolean algebra $\{0, 1\}$ [4, 7].

In this paper, we propose semantics of modal μ-calculus that interprets disjunctions by min, and conjunctions by plus. By using plus, one can compute or count quantitative measures. As an algebra with plus, we have adopted *min-plus algebra.* Algebraic structures that have two binary operators, min and plus, are generally called min-plus algebra, and have been widely used for analysis of discrete event systems, optimization, etc. [1]. Especially, an algebraic structure that has the following properties is called a dioid:

- min is associative and commutative,
- plus is associative and distributes over min,
- ∞ is zero element with respect to min,
- zero element ∞ is absorptive with respect to plus,
- plus has unit element 0, and
- min is idempotent.

In addition, if

- plus is commutative,

[*]This research has been partially conducted in the research project "Solving the description explosion problem in verification by means of structure transformation" in the research area "New High-performance Information Processing Technology Supporting Information-oriented Society" under Core Research for Evolutional Science and Technology (CREST) of Japan Science and Technology Agency.

[†]This research has been partially supported by Grant-in-Aid for Scientific Research by Ministry of Education, Culture, Science and Technology , Scientific Research(C) (2)18500003, "Abstraction of graphs to multisets using temporal logic".

then it is called a commutative dioid. The algebra $\mathbb{N}_\infty$ that consists of all natural numbers $\mathbb{N}$, including 0, and infinity ∞ is a commutative dioid. In this paper, we define how to interpret modal logic formulas on this algebra $\mathbb{N}_\infty$. A dioid is also known as an idempotent semiring: it satisfies the requirements for a ring except for the existence of inverse elements with respect to min. The algebra that consists of all real numbers (or all integers) and infinity ∞ is an idempotent commutative semifield: inverse elements with respect to plus exist except for ∞.

We briefly introduce the semantics proposed in this paper. As mentioned above, we interpret disjunctions by min, and conjunctions by plus, so the typical element that represents truth in $\mathbb{N}_\infty$ is 0, and ∞ represents falsity. Finite elements other than 0 are also considered to represent truth, i.e., there are various levels of truth.

For example, think of the following modal logic formula.

$$\mu X(a \vee \langle f\rangle(1 \wedge X))$$

This formula contains as a subformula the proposition symbol 1 that is always interpreted as the element 1 of $\mathbb{N}_\infty$.

As we explain in the next section, each element of $\mathbb{N}_\infty$ is allowed as a formula and interpreted as itself. And because $\wedge$ is interpreted by plus, the formula $1 \wedge X$ is interpreted as the result of adding 1 to the interpretation of X.

The formula contains the modal operator $\langle f\rangle$. In the ordinary semantics, a formula of the form $\langle f\rangle\varphi$ holds at a state s if and only if there exists a state s' which is reachable in one step from s by the modality f, and φ holds at s'. In a Kripke structure, each modality f is given a one-step reachability relation for interpreting the modal operator $\langle f\rangle$.

Since the interpretation of $\langle f\rangle\varphi$ is thought to be the disjunction of interpretations of φ at all reachable states, and $\vee$ is interpreted by min in the semantics of this paper, $\langle f\rangle\varphi$ is interpreted as the minimum of interpretations of φ at all states s' reachable from s by f in one step.

In a Kripke structure, for each propositional symbol a and for each state s, it is defined whether a is true or false at s. In the semantics of this paper, if a is true, then a is interpreted as 0, and if a is false, a is interpreted as ∞. Thus, the formula $a \vee \langle f\rangle(1 \wedge X)$ is interpreted as 0 if a is true, and has the same value as that of $\langle f\rangle(1 \wedge X)$ if a is false.

Now, consider the formula $\mu X(a \vee \langle f\rangle(1 \wedge X))$ made of the μ-operator, which gives the least fixed point. In the ordinary semantics, the truth value of this formula is given as the least interpretation of the variable X that satisfies the equation

$$X = a \vee \langle f\rangle(1 \wedge X).$$

Roughly speaking, the truth value of the formula can be computed by expanding X as follows.

$$\begin{aligned} X &= a \vee \langle f\rangle(1 \wedge X) \\ &= a \vee \langle f\rangle(1 \wedge (a \vee \langle f\rangle(1 \wedge X))) \\ &= \cdots \end{aligned}$$

Having the least fixed point as its interpretation means that if the truth value is not determined no matter how many times it is expanded, then the formula is interpreted as false. Thus, the above formula is interpreted at a state s as true if a state where a is true is reachable from s by traversing f iteratively, and otherwise as false.

Since the proposition true is interpreted as the minimum element 0 and the proposition false is interpreted as the maximum element ∞ in the semantics of this paper, the μ-operator is computed as the greatest fixed point. So, the above formula is interpreted as ∞ at s if any state where a is true is not reachable from s by traversing f iteratively. And if some state where a is true is reachable from s, this formula is interpreted as the length of the shortest path to a state where a is true, because 1 is added as f is traversed once.

Here is another example. For a propositional symbol b, the interpretation of its negation $\neg b$ is ∞ if b is true, and 0 if b is false. Thus, the interpretation of the formula $\neg b \vee 1$ is 1 if b is true, and 0 if b is false. The modality o is called global modality, and allows transition to any state from any state. Using this modality, let us interpret the following formula.

$$[o](\neg b \vee 1)$$

For each modality f, $[f]$ is also a modal operator. In the ordinary semantics, a formula of the form $[f]\varphi$ holds at a state s if and only if φ holds at any state s' reachable from s by f in one step. The interpretation of $[f]\varphi$ is thought to be the conjunction of interpretations of φ at all reachable states. In the semantics of this paper, since $\wedge$ is interpreted by plus, $[f]\varphi$ is interpreted as the sum of interpretations of φ at all states s' reachable from s by f in one step.

Accordingly, interpreting the above formula in the semantics of this paper involves counting 1 for each state at which b holds. So after all, no matter where this formula is interpreted, its interpretation is the number of states at which b holds in the whole

Kripke structure.

As in the above examples, by applying the semantics of this paper, one can express various properties on Kripke structures including quantitative measures. We have seen that one can easily express the length of the shortest paths and the number of states at which a specified condition holds.

The goal of this research is to define the above semantics rigorously, formalize the algorithm for model checking according to the semantics, and implement it efficiently. And we also investigate the applicability of the semantics. At present, we have two applications in mind: shape analysis [8] in the style of Tanabe et al. [10, 9], and data flow analysis for compiler optimization [6].

Model checking in general means to compute the interpretation of a given formula on a concrete Kripke structure [2]. In this research, we study the algorithm that interprets a formula according to the semantics on $\mathbb{N}_\infty$. As for the μ-operator, the ordinary iterative algorithm is applicable. First, the interpretation of the variable in the μ-operator is tentatively set to ∞ at every state. As iteration progresses, the interpretation of the variable decreases or does not change. Once it decreases to a finite value from ∞, it decreases only finitely many times. So iteration is guaranteed to terminate in finite times. In other words, $\mathbb{N}_\infty$ is well-founded.In contrast, computation of the ν-operator is not trivial. In this research, we have shown that computation of the ν-operator is also possible. Under some restrictions on formulas, it is as efficient as computation of the μ-operator.

As for the application to data flow analysis, we plan to apply the semantics of the paper to the framework of Lacey et al. [6], where CTL is used to specify conditions on a control flow graph for program transformation. By applying the semantics on $\mathbb{N}_\infty$, one can also obtain quantitative hints for program transformation as interpretations of formulas. In order to enhance such quantitative analysis, we have added the max operator to μ-calculus, and extended the model checking algorithm.

The organization of this paper is as follows. In the next section, we defines the syntax and semantics of formulas on min-plus algebra. In Section 3, which is the main part of this paper, we describe the model checking algorithm under the semantics in this paper. We then briefly describe prototype implementation of the model checking algorithm in Section 4 and its applications in Section 5. Section 6 concludes the paper.

2 Syntax and Semantics of Formulas

In this section, we formalize the syntax and the semantics of formulas.

2.1 Syntax

To define the syntax of formulas, we assume the following sets of symbols.

- PS: a set of proposition symbols
- Mod: a set of modality
- PV: a set of proposition variables

Formulas are then defined as follows:

$$\begin{aligned}\varphi ::= {} & \bot \mid b \mid i \mid X \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \\ & \varphi \to \varphi \mid \langle m\rangle\varphi \mid [m]\varphi \mid \\ & \mu X\varphi \mid \nu X\varphi,\end{aligned}$$

where $b \in \mathrm{PS}$, $i \in \mathbb{N}_\infty$, $m \in \mathrm{Mod}$ and $X \in \mathrm{PV}$. In the formulas $\mu X\varphi$ and $\nu X\varphi$, it is required that X not occur negatively in φ.

We abbreviate $\varphi \to \bot$ as $\neg\varphi$.

2.2 Semantics

Let $\mathcal{K} = (S, R, \lambda)$ be a Kripke structure.

- S is a set. An element of S is called a state.
- $R \in \mathrm{Mod} \to \mathscr{P}(S \times S)$
- $\lambda \in \mathrm{PS} \to \mathscr{P}(S)$

A function $\iota \in \mathrm{PV} \to (S \to \mathbb{N}_\infty)$ is called a valuation. (In this paper, if A and B are sets, $A \to B$ denotes the set of all functions from A to B, and $\mathscr{P}(A)$ denotes the set of all subsets of A.)

For a formula φ, we define $[\![\varphi]\!]^{\mathcal{K},\iota} \in S \to \mathbb{N}_\infty$ as follows. We often omit $\mathcal{K}$ and ι.

- $[\![\bot]\!](s) = \infty$
- For $a \in \mathrm{PS}$,
$$[\![a]\!](s) = \begin{cases} 0 & (s \in \lambda(a)) \\ \infty & (s \notin \lambda(a)) \end{cases}$$
- For $i \in \mathbb{N}_\infty$, $[\![i]\!](s) = i$
- $[\![X]\!](s) = \iota(X, s)$
- $[\![\varphi \vee \psi]\!](s) = \min([\![\varphi]\!](s), [\![\psi]\!](s))$
- $[\![\varphi \wedge \psi]\!](s) = [\![\varphi]\!](s) + [\![\psi]\!](s)$
- $[\![\varphi \to \psi]\!](s) = [\![\varphi]\!](s) \Rightarrow [\![\psi]\!](s)$, where for $i, j \in \mathbb{N}_\infty$, $i \Rightarrow j = \begin{cases} 0 & \text{if } j \leq i \\ j - i & \text{otherwise} \end{cases}$
- $[\![\langle m\rangle\varphi]\!](s) = \min\limits_{(s,s') \in R(m)} [\![\varphi]\!](s')$
- $[\![[m]\varphi]\!](s) = \sum\limits_{(s,s') \in R(m)} [\![\varphi]\!](s')$
- $[\![\mu X\varphi]\!]^\iota(s) =$
$$\sup\{F(s) \mid F : S \to \mathbb{N}_\infty,\ F \leq [\![\varphi]\!]^{\iota[X \mapsto F]}\}$$

- $[\![\nu X\varphi]\!]^\iota(s) =$
  $\min\{F(s) \mid F : S \to \mathbb{N}_\infty,\ F \geq [\![\varphi]\!]^{\iota[X\mapsto F]}\}$,

where, for a function f, $f[x \mapsto y]$ is a function whose value at x is y and whose value at $z \in \mathrm{dom}(f) \setminus \{x\}$ is $f(z)$. Also, for functions $F, G \in S \to \mathbb{N}_\infty$, $F \leq G$ means that $F(s) \leq G(s)$ for any $s \in S$. We often denote $S \to \mathbb{N}_\infty$ by L.

According to the definition, $[\![\mu X\varphi]\!]^\iota$ and $[\![\nu X\varphi]\!]^\iota$ are the maximum and the minimum among $F \in L$ that satisfies $F = [\![\varphi]\!]^{\iota[X\mapsto F]}$, respectively.

3 Model Checking Algorithm

In this section, we give algorithms for model checking under the semantics defined in the previous section.

As mentioned in Section 1, since $\mathbb{N}_\infty$ is well-founded, the interpretation of the μ-operator can be obtained by ordinary iterative computation. In the case of the ν-operator, however, the well-foundedness of $\mathbb{N}_\infty$ does not guarantee that this kind of computation terminates. Thus, a new kind of computation for the fixed point operator is needed.

```
var $LV$ : Exp → $\mathbb{N}_\infty$
var $AV$ : Exp → $\mathbb{N}_\infty$

$comp\_fml(\varphi : \mathrm{Fml}) : L$
  return $comp\_ni(\lceil\varphi\rceil)$

$comp\_ni(\mathcal{E} : \mathrm{Sem}_S) : L$
  initialize $LV$
  initialize $AV$
  $acomp(\mathcal{E})$
  return $comp(\nu to\mu(Abst(\mathcal{E})))$
```

Figure 1: Overall model checking algorithm (without ⇒)

```
var $LV'$ : Exp → $\mathbb{N}_\infty$

$comp\_fml'(\varphi : \mathrm{Fml}) : L$
  initialize $LV'$
  return $comp'(\lceil\varphi\rceil)$
```

Figure 2: Overall model checking algorithm (with ⇒)

3.1 Overview

We first construct a model checking algorithm for formulas without implications. If a formula contains no implications, one can determine whether or not the interpretation of the formula at a state is 0 or not by *abstract computation* described in Section 3.5.1. Then, using the information obtained by abstract computation, one can rewrite ν-operators into μ-operators in the formula. We explain this fact in Section 3.5.1, the algorithm using abstract computation in Section 3.5.2, and the proof of its correctness in Section 3.5.3.

The overall algorithm is defined as the procedure *comp_fml* in Figure 1, which returns $[\![\varphi]\!]$ for a formula φ. We assume that φ contains no free proposition variables.

The procedure *comp_fml* calls the other procedure *comp_ni* in Figure 1 with the result $\lceil\varphi\rceil$ of translating φ into its corresponding semantic expression as defined in Section 3.2. The procedure *comp_ni* first calls the procedure *acomp* which performs the abstract computation mentioned above. The procedure *acomp* is given in Figure 4 and defined in Section 3.5.1.

The function *Abst* rewrites the semantic expression using the information obtained by abstract computation and then the function $\nu to\mu$ rewrites ν-operators into μ-operators. They are defined in Section 3.5.1. The procedure *comp* then computes the interpretation of the rewritten expression by ordinary iterative computation, since it contains no ν-operators. The procedure *comp* is given in Figure 3 and defined in Section 3.3.

These procedures refer to the global variables LV (*Last Valuation*) and AV (*Abstract Valuation*). The variable LV stores a mapping from expressions to elements of $\mathbb{N}_\infty$. We assume that LV is implemented as an array or a table. Assignments like $LV(X(s)) := i$ are assumed to destructively modify the mapping in LV. The variable AV stores abstract valuations obtained by abstract computation.

These variables are initialized in the procedure *comp_ni* with an empty mapping, whose value is undefined for all expressions.

If a formula contains implications, one can compute its interpretation by iteratively making approximate expressions without implications and computing their interpretations by *comp_ni*. The overall algorithm is defined as the procedure *comp_fml'* in Figure 2, which refers to the global variable LV', initializes it, and calls the procedure *comp'* given later in Section 3.6.

Some definitions needed to present the algorithms are given in the next section.

3.2 Definitions

Our model checking algorithm manipulates new kinds of syntactic entities called expressions and semantic expressions. Following are some relevant sets including those of these entities.

- S : the set of all states in a Kripke structure
- Fml : the set of all formulas
- Exp : the set of all expressions
- $\mathrm{Sem}_S = S \to \mathrm{Exp}$

(*expressions and semantic expressions*)
The set Exp of expressions is defined as follows:

$$\begin{aligned}\mathrm{Exp} \ni E ::= \min(\{E_j\}) \mid \textstyle\sum(\{E_j\}) \mid Xs \mid i \mid \\ E_- \Rightarrow E_+ \mid \mu X \mathcal{E} s \mid \nu X \mathcal{E} s,\end{aligned}$$

where $E_-, E_+ \in \mathrm{Exp}$, $\{E_j\}$ is a sequence of Exp, $i \in \mathbb{N}_\infty$, $X \in \mathrm{PV}$, $s \in S$, and $\mathcal{E} \in \mathrm{Sem}_S$. We call an element of Exp an *expression*, and an element of Sem_S a *semantic expression*.

A semantic expression represents a function obtained by interpreting a formula. Namely, we define $\lceil\cdot\rceil \in \mathrm{Fml} \to \mathrm{Sem}_S$ according to the semantics as follows.

(*translation from formulas to semantic expressions*)
Let $\varphi, \psi \in \mathrm{Fml}$, $a \in \mathrm{PS}$, $i \in \mathbb{N}_\infty$, $X \in \mathrm{PV}$, $m \in \mathrm{Mod}$, and $s \in S$.

- $\lceil a \rceil(s) = [\![a]\!](s)$
- $\lceil i \rceil(s) = i$
- $\lceil X \rceil(s) = Xs$
- $\lceil \varphi \vee \psi \rceil(s) = \min(\{\lceil\varphi\rceil(s), \lceil\psi\rceil(s)\})$
- $\lceil \varphi \wedge \psi \rceil(s) = \sum(\{\lceil\varphi\rceil(s), \lceil\psi\rceil(s)\})$
- $\lceil \langle m \rangle \varphi \rceil(s) = \min(\{\lceil\varphi\rceil(s')\}_{(s,s') \in R(m)})$
- $\lceil [m] \varphi \rceil(s) = \sum(\{\lceil\varphi\rceil(s')\}_{(s,s') \in R(m)})$
- $\lceil \varphi \to \psi \rceil(s) = \lceil\varphi\rceil(s) \Rightarrow \lceil\psi\rceil(s)$
- $\lceil \mu X \varphi \rceil(s) = \mu X \lceil\varphi\rceil s$
- $\lceil \nu X \varphi \rceil(s) = \nu X \lceil\varphi\rceil s$

For $X \in \mathrm{PV}$ and $\mathcal{E} \in \mathrm{Sem}_S$, let $\mu X \mathcal{E}$ denote the semantic expression that maps $s \in S$ to $\mu X \mathcal{E} s$. The semantic expression $\nu X \mathcal{E}$ is defined similarly. A function $F \in L = S \to \mathbb{N}_\infty$ can also be seen as a semantic expression since $\mathbb{N}_\infty \subseteq \mathrm{Exp}$.

(*order on* $\mathrm{Exp} \cup \mathrm{Sem}_S$)
For any $E, E' \in \mathrm{Exp} \cup \mathrm{Sem}_S$, we write $E \leq E'$ if and only if E is a subexpression of E'.

By this order, $\mathrm{Exp} \cup \mathrm{Sem}_S$ is well-founded.

(*free and bound variables of expressions*)
The set of all free variables of E (or $\mathcal{E}$) is denoted by $\mathrm{FreeVar}(E)$ (or $\mathrm{FreeVar}(\mathcal{E})$). The set of all bound variables in E (or $\mathcal{E}$) is denoted by $\mathrm{BndVar}(E)$ (or $\mathrm{BndVar}(\mathcal{E})$).

We assume that $\mathrm{FreeVar}(E) \cap \mathrm{BndVar}(E) = \emptyset$ for any $E \in \mathrm{Exp} \cup \mathrm{Sem}_S$. By this assumption, "$X$ is a free variables of E" means that X occurs in E but no $\mu X \mathcal{E}$ or $\nu X \mathcal{E}$ occurs in E, where $X \in \mathrm{PV}, E \in \mathrm{Exp} \cup \mathrm{Sem}_S$ and $\mathcal{E} \in \mathrm{Sem}_S$.

We also assume that for each $X \in \mathrm{PV}$, if E contains a subexpression of the form $\mu X \mathcal{E}$ or $\nu X \mathcal{E}$, it is unique in E. By this assumption, we can define the following operators.

(*op_E and sm_E*)
The operator $op_E \in \mathrm{PV} \to \{\mu, \nu\}$ is defined as

$$op_E X = \begin{cases} \mu & \text{if } \mu X \mathcal{E} \leq E \text{ for some } \mathcal{E} \in \mathrm{Sem}_S \\ \nu & \text{if } \nu X \mathcal{E} \leq E \text{ for some } \mathcal{E} \in \mathrm{Sem}_S \\ \text{not} & \text{defined otherwise,} \end{cases}$$

and $sm_E \in \mathrm{PV} \to \mathrm{Sem}_S$ as

$$sm_E X = \begin{cases} \mathcal{E} & \text{if } \mu X \mathcal{E} \text{ (or } \nu X \mathcal{E}) \leq E \\ & \quad \text{for some } \mathcal{E} \in \mathrm{Sem}_S \\ \text{not} & \text{defined otherwise.} \end{cases}$$

The operators $op_{\mathcal{E}}$ and $sm_{\mathcal{E}}$ are defined similarly.

For instance, if $\mu X \mathcal{E}$ is a subexpression of E for some $\mathcal{E} \in \mathrm{Sem}_S$, then $\mu X \mathcal{E} = (op_E X) X (sm_E X)$ holds. We denote this subexpression $(op_E X) X (sm_E X)$ by $fix_E X$.

(*substitution*)
Let $\mathcal{E} \in \mathrm{Sem}_S$ and $E \in \mathrm{Exp}$. For $X \in \mathrm{FreeVar}(\mathcal{E})$ (or $X \in \mathrm{FreeVar}(E)$) and $\mathcal{F} \in \mathrm{Sem}_S$, $\mathcal{E}[X \mapsto \mathcal{F}] \in Sem_S$ (or $E[X \mapsto \mathcal{F}] \in \mathrm{Exp}$) denotes the result of substituting $\mathcal{F}$ for X in E (or $\mathcal{E}$). The definition of substitution is straight-forward. Note that for the expression Xs, $(Xs)[X \mapsto \mathcal{F}] = \mathcal{F}(s)$ holds.

Finally, we define $0_L \in L$ and $\infty_L \in L$ as the constant functions that map any $s \in S$ to 0 and ∞, respectively. They are the minimum and maximum in L with respect to the point-wise order in $L = S \to \mathbb{N}_\infty$.

3.3 Computation of Expressions

In this section, we define the valuation of an expression or a semantic expression. If a semantic expression is derived from some formula, the valuation of the semantic expression is equal to the interpretation of the formula.

(*valuations of expressions*)
For $\iota \in \mathrm{PV} \to L$, $E \in \mathrm{Exp}$ and $\mathcal{E} \in \mathrm{Sem}_S$, the valuations $[\![E]\!]^\iota \in \mathbb{N}_\infty$ and $[\![\mathcal{E}]\!]^\iota \in L$ are defined. In the following, we assume that $i \in \mathbb{N}_\infty$, $X \in \mathrm{PV}$, $E_+, E_- \in \mathrm{Exp}$, $\{E_j\}_{j \in I}$ is a sequence of Exp (where I is its index set), and $s \in S$.

- $[\![i]\!]^\iota = i$
- $[\![Xs]\!]^\iota = (\iota(X))(s)$
- $[\![\min(\{E_j\}_{j \in I})]\!]^\iota = \min\limits_{j \in I} [\![E_j]\!]^\iota$
- $[\![\sum(\{E_j\}_{j \in I})]\!]^\iota = \sum\limits_{j \in I} [\![E_j]\!]^\iota$
- $[\![E_- \Rightarrow E_+]\!]^\iota = [\![E_-]\!]^\iota \Rightarrow [\![E_+]\!]^\iota$

```
$comp(\mathcal{E} : \mathrm{Sem}_S) : L$
  for each $s \in S$ do $LV(\mathcal{E}(s)) := comp(\mathcal{E}(s))$
  return $\lambda s \in S.\ LV(\mathcal{E}(s))$

$comp(E : \mathrm{Exp}) : \mathbb{N}_\infty$
  case $(E)$ of
    $i \Rightarrow LV(E) := i$
    $Xs \Rightarrow$ skip
    $\min(\{E_j\}_{j\in I}) \Rightarrow LV(E) := \min_{j\in I} comp(E_j)$
    $\sum(\{E_j\}_{j\in I}) \Rightarrow LV(E) := \sum_{j\in I} comp(E_j)$
    $(\mu X\mathcal{E})(s) \Rightarrow LV(E) := iter(X,\ \mathcal{E},\ \infty_L)(s)$
  return $LV(E)$

$iter(X : \mathrm{PV},\ \mathcal{E} : \mathrm{Sem}_S,\ F : L) : L$
  for each $s \in S$ do $LV(Xs) := F(s)$
  while $\exists s \in S.\ LV(Xs) \neq LV(\mathcal{E}(s))$ do
    for each $s \in S$ do $LV(Xs) := LV(\mathcal{E}(s))$
    $comp(\mathcal{E})$
  return $\lambda s \in S.\ LV(Xs)$
```

Figure 3: Computation of valuations (without $\Rightarrow$)

- $[\![\mu X\mathcal{E}s]\!]^\iota =$
  $\sup\{F(s) \mid F \in L, F \le [\![\mathcal{E}]\!]^{\iota[X\mapsto F]}\}$
- $[\![\nu X\mathcal{E}s]\!]^\iota =$
  $\min\{F(s) \mid F \in L, F \ge [\![\mathcal{E}]\!]^{\iota[X\mapsto F]}\}$
- $[\![\mathcal{E}]\!]^\iota(s) = [\![\mathcal{E}(s)]\!]^\iota$

Note that $[\![E]\!]^\iota$ and $[\![\mathcal{E}]\!]^\iota$ no longer depend on the Kripke structure except for its state set.

For any formula φ, $[\![\varphi]\!]^{\mathcal{K},\iota} = [\![\lceil\varphi\rceil]\!]^\iota$ holds. So our goal is now to compute $[\![\lceil\varphi\rceil]\!]^\iota$. The procedure *comp* in Figure 3 computes $[\![\mathcal{E}]\!]^\iota \in L$ and $[\![E]\!]^\iota \in \mathbb{N}_\infty$. As mentioned in Section 3.1, we use the global variable $LV : \mathrm{Exp} \to \mathbb{N}_\infty$ (*Last Valuation*). Each time the valuation of $E \in \mathrm{Exp}$ is computed, the value $LV(E)$ is updated as the valuation $[\![E]\!]^{\iota_{LV}}$ of E, where ι_{LV} denotes a valuation satisfying $\iota_{LV}(Y)(s) = LV(Ys)$ for $Y \in \mathrm{PV}$ and $s \in S$

As for i, Xs, min and $\sum$, *comp* computes their valuations recursively along the above definition. As for the μ-operator, it calls the procedure *iter* explained in the next section. Since ν is rewritten to μ in the case without implications, it does not need to handle the ν-operator.

In Figure 3, $\min_{j\in I} comp(E_j)$ and $\sum_{j\in I} comp(E_j)$ mean to actually compute the operators min and $\sum$ on min-plus algebra.

3.4 Iterative Fixed Point Computation

The interpretation $[\![\mu X\mathcal{E}]\!]^\iota(s)$ of the μ-operator $\mu X\mathcal{E}$ was defined as $\sup\{F(s) \mid F \in L,\ F \le [\![\mathcal{E}]\!]^{\iota[X\mapsto F]}\}$. Because $\mathbb{N}_\infty$ is well-founded as described in Section 1, if S is a finite set, finite iterative computation can compute the interpretation of the μ-operator as follows.

First, define the function $F_0 \in L = S \to \mathbb{N}_\infty$ as ∞_L, i.e., $F_0(s) = \infty$ for any $s \in S$. Next, for $n \ge 0$, define the function $F_{n+1} : S \to \mathbb{N}_\infty$ as

$$F_{n+1} \ = \ [\![\mathcal{E}]\!]^{\iota[X\mapsto F_n]}.$$

Then $F_{n+1} \le F_n$ always holds, and if S is finite, there exists $m \ge 0$ that satisfies $F_{m+1} = F_m$. Here F_m is equivalent to $[\![\mu X\mathcal{E}]\!]^\iota(s)$.

The above iterative computation is performed by $iter(X, \mathcal{E}, \infty_L)$ with the procedure *iter* in Figure 3. During this computation, $LV(\mathcal{E}(s)) \le LV(Xs)$ always holds, and $LV(Xs)$ monotonically decreases and eventually becomes equal to $LV(\mathcal{E}(s)) = [\![\mu X\mathcal{E}]\!]^{\iota_{LV}}$. Note also that by the call of *comp* inside *iter*, at the end of the computation $iter(X, \mathcal{E}, \infty_L)$, $[\![E]\!]^{\iota_{LV}} = LV(E)$ holds for any subexpression $E \le \mu X\mathcal{E}(s)$.

3.5 Algorithm for Semantic Expressions *without* Implications "$\Rightarrow$" and its Correctness

In this section, we introduce a model checking algorithm for semantic expressions without implications. As mentioned in Section 3.1, the core of the algorithm lies in rewriting the ν-operator to the μ-operator using the information on whether each instance of the ν-operator is evaluated to 0 or non-0 by abstract computation.

3.5.1 Abstract Computation

We now introduce the abstract computation, which determines whether the valuation of an expression is 0 or not. Using this information, one can make a new expression whose valuation is equivalent to that of the original one.

We first define the abstraction maps $\alpha_{\mathbb{N}_\infty}$ and α_L:

$$\begin{aligned} \alpha_{\mathbb{N}_\infty}(i) &= \begin{cases} 0 & \text{if } i = 0 \\ \infty & \text{otherwise} \end{cases} \\ (\alpha_L(F))(s) &= \alpha_{\mathbb{N}_\infty}(F(s)), \end{aligned}$$

```
$acomp(\mathcal{E} : \mathrm{Sem}_S) : L$
  for each $s \in S$ do $AV(\mathcal{E}(s)) := acomp(\mathcal{E}(s))$
  return $\lambda s \in S.\ AV(\mathcal{E}(s))$

$acomp(E : \mathrm{Exp}) : \mathbb{N}_\infty$
  case $(E)$ of
    $i \Rightarrow\ AV(E) := \alpha_{\mathbb{N}_\infty}(i)$
    $Xs \Rightarrow$ skip
    $\min(\{E_j\}_{j\in I}) \Rightarrow\ AV(E) := \min_{j\in I} acomp(E_j)$
    $\sum(\{E_j\}_{j\in I}) \Rightarrow\ AV(E) := \sum_{j\in I} acomp(E_j)$
    $\mu X\mathcal{E}s \Rightarrow\ AV(E) := aiter(X,\ \mathcal{E},\ \infty_L)(s)$
    $\nu X\mathcal{E}s \Rightarrow\ AV(E) := aiter(X,\ \mathcal{E},\ 0_L)(s)$
  return $AV(E)$

$aiter(X : \mathrm{PV},\ \mathcal{E} : \mathrm{Sem}_S,\ F : L) : L$
  for each $s \in S$ $AV(Xs) := F(s)$
  while $\exists s \in S.\ AV(Xs) \neq AV(\mathcal{E}(s))$ do
    for each $s \in S$ do $AV(Xs) := AV(\mathcal{E}(s))$
    $acomp(\mathcal{E})$
  return $\lambda s \in S.\ AV(Xs)$
```

Figure 4: Abstract computation

where $i \in \mathbb{N}_\infty$, $F \in L$, and $s \in S$. Note that these abstraction maps preserve the order of their argument.

The procedures *acomp* and *aiter* in Figure 4 compute abstract valuations of expressions including fixed points. The procedure *acomp* is similar to *comp* except that *acomp* abstracts each constant i to 0 or ∞ depending on whether i is 0 or not, and it computes fixed points by *aiter*. Since AV returns only two values 0 and ∞, iteration of *aiter* always terminates as in iterative fixed point computation under the ordinary semantics.

At the end of the computation $acomp(\mathcal{E})$, $AV(E)$ contains the information on whether the valuation of a subexpression $E \leq \mathcal{E}$ is 0 or not. Therefore, using abstract valuations in AV computed by *acomp*, one can rewrite each subexpression E into 0 (as an expression) if $AV(E) = 0$, where AV denotes the final value of the global variable AV in $acomp(\mathcal{E})$. The result of rewriting E is denoted by $Abst(E)$, and that of rewriting $\mathcal{E}$ is denoted by $Abst(\mathcal{E})$ and used in Figure 1.

For $\mathcal{E} \in \mathrm{Sem}_S$, we define the valuation $\iota_\mathcal{E} \in (PV \to L) \to L$ as follows.

$$(\iota_\mathcal{E}(X))(s) = \begin{cases} 0 & \text{if } X \in \mathrm{BndVar}(\mathcal{E}) \text{ and } (sm_\mathcal{E} X)(s) = 0 \\ \infty & \text{otherwise,} \end{cases}$$

where $s \in S$ and $X \in \mathrm{PV}$. The valuation $\iota_\mathcal{E}$ returns 0 for X if X is bound to the expression 0.

Then $\iota_{Abst(\mathcal{E})}(X)(s) = AV(Xs)$ holds for any $X \in \mathrm{FreeVar}(\mathcal{E}) \cup \mathrm{BndVar}(\mathcal{E})$ and $s \in S$, where AV denotes the final value of the global variable AV in $acomp(\mathcal{E})$.

The abstracted semantic expression $Abst(\mathcal{E})$ has the following properties.
(*properties of* $Abst(\mathcal{E})$)
We can show the following property:

$$[\![Abst(\mathcal{E})]\!] = [\![\mathcal{E}]\!].$$

Note that since $\mathcal{E}$ does not contain free variables, the valuations of $\mathcal{E}$ and $Abst(\mathcal{E})$ do not depend on a valuation ι. Furthermore, for any $\iota \in \mathrm{PV} \to L$, $\mathcal{E} \in \mathrm{Sem}_S$, $E \in \mathrm{Exp}$ such that $E \leq Abst(\mathcal{E})$, and $Y \in \mathrm{BndVar}(Abst(\mathcal{E}))$,

$$\begin{aligned} &\alpha_L(\iota(X)) = \iota_{Abst(\mathcal{E})}(X) \\ &\quad \text{for any } X \in \mathrm{FreeVar}(E) \Rightarrow \\ &\qquad [\![E]\!]^\iota = 0 \Leftrightarrow E = 0 \text{ (constant), and} \\ &\alpha_L(\iota(X)) = \iota_{Abst(\mathcal{E})}(X) \\ &\quad \text{for any } X \in \mathrm{FreeVar}(fix_{Abst(\mathcal{E})}Y) \Rightarrow \\ &\qquad \alpha_L([\![fix_{Abst(\mathcal{E})}Y]\!]^\iota) = \iota_{Abst(\mathcal{E})}(Y). \end{aligned}$$

Due to space limintation, we do not give proofs of these properties here. See [14].

3.5.2 Rewriting ν to μ

Using the properties of $Abst(\mathcal{E})$ mentioned above, one can rewrite ν in $Abst(\mathcal{E})$ into μ. We denote this rewriting process by $\nu to\mu$ in Figure 1.

Since $\nu to\mu(Abst(\mathcal{E}))$ contains no ν-operators, the procedure *comp* in Figure 3 can compute the valuation of $\mathcal{E}$.

3.5.3 Proof of Correctness

Abstract computation by *acomp* stores abstract valuations in AV, which decide whether valuations of expressions, including fixed points, are 0 or not. Since $\nu to\mu$ changes ν into μ, we show here that the ν-operator is equivalent to the μ-operator under the condition assured by abstract computation.

We first examine semantic expressions $\mathcal{E} \in \mathrm{Sem}_S$ without μ and ν. In claim 1 below, we show that if one knows the valuation of $\nu X\mathcal{E}$ is not 0 at any state, one can easily compute the valuations of $\nu X\mathcal{E}$ at certain states. In claim 2, using claim 1, we show that the least and greatest fixed points of $[\![\mathcal{E}]\!]$ are equivalent to each other under the same condition as that of claim 1. Finally, in claim 3, we show that μ and ν are equivalent to each other even if the μ- and ν-operators are nested.

Assume that $\mathrm{BndVar}(\mathcal{E}) = \emptyset$ so that $\mathcal{E}$ does not contain μ and ν, and assume $\mathrm{FreeVar}(\mathcal{E}) = \{X\}$ so that X is the only propositional variable in $\mathcal{E}$. Then $[\![\mathcal{E}[X \to \cdot\,]]\!]^\iota$ can be seen as a function from L to L by substitution, which does not depend on ι. We denote this function simply by $[\![\mathcal{E}]\!]$ below.

Remember that $\mathcal{E}$ contains only min, $\sum$, constants and Xs where s is a state. The operator min can be moved outside of plus because on min-plus algebra, plus distributes over min. Consequently, for each $s \in S$ there exists a set I_s such that for each $i \in I_s$ and $s' \in S$ there exists $A_{sis'} \in \mathbb{N}$ and $C_{si} \in \mathbb{N}_\infty$ such that for any $F \in L$

$$([\![\mathcal{E}]\!](F))(s) = \min_{i \in I_s}(\sum_{s' \in S} A_{sis'}F(s') + C_{si}).$$

In the following, we show $[\![\mathcal{E}]\!](\infty_L)$ is partially equivalent to the least fixed point of $[\![\mathcal{E}]\!]$ in the sense mentioned in the claim.

(claim 1) Assume $[\![\mathcal{E}]\!] \in L \to L$ and its least fixed point $F \in L$ satisfies $F(s) \neq 0$ for any $s \in S$. Then for any $s \in S$, if

$$([\![\mathcal{E}]\!](\infty_L))(s) = \min_{s \in S}([\![\mathcal{E}]\!](\infty_L))(s),$$

then

$$F(s) = \min_{s \in S}([\![\mathcal{E}]\!](\infty_L))(s).$$

Namely, if $m = \min_{s \in S}([\![\mathcal{E}]\!](\infty_L))(s)$ and $s_m \in S$ satisfies $([\![\mathcal{E}]\!](\infty_L))(s_m) = m$, then $F(s_m)$ can be computed as $([\![\mathcal{E}]\!](\infty_L))(s_m)$.

(proof) Let $m = \min_{s \in S}([\![\mathcal{E}]\!](\infty_L))(s)$ and $s_m \in S$ be such that $([\![\mathcal{E}]\!](\infty_L))(s_m) = m$. Define $T \subset S$ as

$$T = \{t \in S \mid F(t) = \min_{s \in S} F(s)\}.$$

T is not empty since $\mathbb{N}_\infty$ is well-ordered.

First, since $\infty_L = \max L$, $F = [\![\mathcal{E}]\!](\mathcal{F}) \leq [\![\mathcal{E}]\!](\infty_L)$, which implies $F(s) \leq ([\![\mathcal{E}]\!](\infty_L))(s)$ for all $s \in S$. So $F(s_m) \leq ([\![\mathcal{E}]\!](\infty_L))(s_m)$.

We show $F(s_m) \geq ([\![\mathcal{E}]\!](\infty_L))(s_m)$ by reductio ad absurdum assuming that $F(s_m) < ([\![\mathcal{E}]\!](\infty_L))(s_m)$.

Let $L' = S \setminus T \to \mathbb{N}_\infty$, and define $[\![\mathcal{E}]\!]' \in L' \to L'$ as follows. For any $H \in L'$ and $s \in S \setminus T$,

$$([\![\mathcal{E}]\!]'(H))(s) = ([\![\mathcal{E}]\!](l(H)))(s),$$

where $l \in L' \to L$ is defined as follows. For any $s \in S$,

$$(l(H))(s) = \begin{cases} 0 & \text{if } s \in T \\ H(s) & \text{otherwise.} \end{cases}$$

Since L' is also a complete lattice, $[\![\mathcal{E}]\!]'$ has a fixed point $F' \in L'$.

We show that $l(F')$ is a fixed point of $[\![\mathcal{E}]\!]$, i.e., for all $s \in S$,

$$([\![\mathcal{E}]\!](l(F')))(s) = (l(F'))(s).$$

(Since $(l(F'))(s) = 0$ for all $s \in T$, this is contradictory to the assumptions that $F(s) \neq 0$ and that F is the least fixed point.)

(a) When $s \in S \setminus T$,

$$\begin{aligned} (l(F'))(s) &= F'(s) \\ &= ([\![\mathcal{E}]\!]'(F'))(s) \\ &= ([\![\mathcal{E}]\!](l(F')))(s). \end{aligned}$$

(b) When $s \in T$, since $[\![\mathcal{E}]\!]$ is a function consists of only min and plus as described above, $([\![\mathcal{E}]\!](F))(s) = F(s)$ means

$$\exists i \in I_s \text{ s.t. } \sum_{s' \in S} A_{sis'}F(s') + C_{si} = F(s).$$

Because $F(s) = \min_{s' \in S} F(s')$, this i must satisfy either of the following two propositions.

(1) For all $s' \in S$, $A_{sis'} = 0$ and $C_{si} = F(s)$.

(2) $C_{si} = 0$ and $\exists!\ t_s \in T$ *s.t.* for any $s' \in S$, $A_{sis'} = 1$ if $t_s = s'$, and $A_{sis'} = 0$ otherwise.

If (1) is satisfied, $\sum_{s' \in S} A_{tis'}F(s') + C_{si}$ is constant, thus $([\![\mathcal{E}]\!](\infty_L))(s) \leq C_{si} = F(s)$.

On the other hand, $F(s) \leq F(s_m)$ as $s \in T$, $F(s_m) < ([\![\mathcal{E}]\!](\infty_L))(s_m)$ by the assumption, and $([\![\mathcal{E}]\!](\infty_L))(s_m) \leq ([\![\mathcal{E}]\!](\infty_L))(s)$ as $([\![\mathcal{E}]\!](\infty_L))(s_m) = m$. Therefore $F(s) < ([\![\mathcal{E}]\!](\infty_L))(s)$. It is contradictory to $([\![\mathcal{E}]\!](\infty_L))(s) \leq F(s)$. So, (2) must hold.

Then since $t_s \in T$, $l(F'))(t_s) = 0$ and $A_{sit_s}(l(F'))(t_s) = 0$. This implies $([\![\mathcal{E}]\!](l(F')))(s) = 0$. Since $s \in T$, $(l(F'))(s) = 0$, so $([\![\mathcal{E}]\!](l(F')))(s) = (l(F'))(s)$.

Because of (a) and (b), $l(F')$ is a fixed point of $[\![\mathcal{E}]\!]$, resulting in the contradiction above.

$\square$

When $F \in L$ is the greatest fixed point, a similar claim can be shown without assuming $F(s) \neq 0$.

We denote the least fixed point of $[\![\mathcal{E}]\!]$ by $\nu[\![\mathcal{E}]\!]$, and the greatest fixed point by $\mu[\![\mathcal{E}]\!]$.

For $\mathcal{E} \in Sem_S$ and $i \in \mathbb{N}$, we denote by $[\![\mathcal{E}]\!]^i$ the i-th iteration of the map $[\![\mathcal{E}]\!]$, i.e., $[\![\mathcal{E}]\!]^0$ is the identity map and $[\![\mathcal{E}]\!]^{i+1} = [\![\mathcal{E}]\!]^i \circ [\![\mathcal{E}]\!]$. Do not confuse it with $[\![\mathcal{E}]\!]^\iota \in L$ where ι is a valuation.

We also define $\mathcal{E}^{(X,i)}F \in \mathrm{Sem}_S$ for any $\mathcal{E} \in \mathrm{Sem}_S$, $X \in \mathrm{PV}$, $i \in \mathbb{N}$ and $F \in L$ as follows.

$$\begin{cases} \mathcal{E}^{(X,i+1)}F = \mathcal{E}[X \mapsto \mathcal{E}^{(X,i)}F] \\ \mathcal{E}^{(X,0)}F = \mathcal{E}[X \mapsto F] \end{cases}$$

If $\mathcal{E} \in \mathrm{Sem}_S$ satisfies the same properties as those of claim 1, the following claim also holds.

(claim 2) $[\![\mathcal{E}]\!]^{|S|}(\infty_L) = \nu[\![\mathcal{E}]\!] = \mu[\![\mathcal{E}]\!]$

(proof) We define $S_i \subset S$ for any $i \in \mathbb{N}$ as

$$\begin{aligned} S_i &= \{s \in S \mid ([\![\mathcal{E}]\!]^i(\infty_L))(s) \\ &\qquad = (\nu[\![\mathcal{E}]\!])(s) = (\mu[\![\mathcal{E}]\!])(s)\}, \end{aligned}$$

and show $|S_i| \geq i$ for any $i \leq |S|$ by induction on i.
(i) When $i = 0$, clearly $|S_i| \geq 0$.
(ii) When $0 \lneq i \leq |S| - 1$, let $L' = S \setminus S_i \to \mathbb{N}_\infty$ and define $[\![\mathcal{E}]\!]_i \in L' \to L'$ as

$$[\![\mathcal{E}]\!]_i(G) = ([\![\mathcal{E}]\!](f_i(G)))|_{S \setminus S_i}$$

for any $G \in L'$, where $f_i : L' \to L$ is defined as

$$(f_i(G))(s) = \begin{cases} ([\![\mathcal{E}]\!]^i(\infty_L))(s) & \text{if } s \in S_i \\ G(s) & \text{otherwise} \end{cases}$$

for all $s \in S$. (For $F \in L = S \to \mathbb{N}_\infty$, $F|_{S \setminus S_i}$ denotes the restriction of F to $S \setminus S_i$.)

A semantic expression $\mathrm{Sem}_{S \setminus S_i} \ni \mathcal{E}_i = \mathcal{E}[X \mapsto \mathcal{F}]$ that consists of min and plus represents $[\![\mathcal{E}]\!]_i$, where

$$\mathcal{F} = \begin{cases} \mathcal{E}^{(X,i)}\infty_L(s) & \text{if } s \in S_i \\ Xs & \text{otherwise.} \end{cases}$$

By the definiton of $[\![\mathcal{E}]\!]_i$, $\nu[\![\mathcal{E}]\!]|_{S \setminus S_i}$ is a fixed point of $[\![\mathcal{E}]\!]_i$. Thus $\nu[\![\mathcal{E}]\!]_i \geq \nu[\![\mathcal{E}]\!]|_{S \setminus S_i}$. This implies, for any $s \in S \setminus S_i$, $\nu[\![\mathcal{E}]\!]_i(s) > 0$. Therefore $[\![\mathcal{E}]\!]_i$ satisfies the assumption of claim 1.

Using claim 1, there exists $s \in S \setminus S_i$ such that

$$([\![\mathcal{E}]\!]_i(\infty_{L'}))(s) = (\nu[\![\mathcal{E}]\!]_i)(s).$$

And by definition of $[\![\mathcal{E}]\!]_i$,

$$[\![\mathcal{E}]\!]^{i+1}(\infty_L) = [\![\mathcal{E}]\!]([\![\mathcal{E}]\!]^i(\infty_L)) \leq [\![\mathcal{E}]\!]_i(\infty_{L'})$$

holds. So

$$([\![\mathcal{E}]\!]^{i+1}(\infty_L))(s) \leq ([\![\mathcal{E}]\!]_i(\infty_{L'}))(s) = (\nu\mathcal{E}_i)(s).$$

And since $\nu[\![\mathcal{E}]\!] \leq \mu[\![\mathcal{E}]\!] \leq [\![\mathcal{E}]\!]^{i+1}(\infty_L)$,

$$\begin{aligned} (\nu[\![\mathcal{E}]\!])(s) &\leq (\mu[\![\mathcal{E}]\!])(s) \\ &\leq ([\![\mathcal{E}]\!]^{i+1}(\infty_L))(s) \\ &\leq (\nu[\![\mathcal{E}]\!]_i)(s) \end{aligned}$$

holds. Furthermore, $(\nu[\![\mathcal{E}]\!])|_{S \setminus S_i}$ is a fixed point of $[\![\mathcal{E}]\!]_i$ by the definition of $[\![\mathcal{E}]\!]_i$, so

$$\nu[\![\mathcal{E}]\!]_i \leq (\nu[\![\mathcal{E}]\!])|_{S \setminus S_i}.$$

Thus

$$\begin{aligned} (\nu[\![\mathcal{E}]\!]_i)(s) &\leq (\nu[\![\mathcal{E}]\!])(s) \leq (\mu[\![\mathcal{E}]\!])(s) \\ &\leq ([\![\mathcal{E}]\!]^{i+1}(\infty_L))(s) \\ &\leq (\nu[\![\mathcal{E}]\!]_i)(s). \end{aligned}$$

Therefore

$$(\nu[\![\mathcal{E}]\!])(s) = (\mu[\![\mathcal{E}]\!])(s) = ([\![\mathcal{E}]\!]^{i+1}\infty_L)(s)$$

holds. After all, $|S_{i+1}| \geq |S_i| + 1 \geq i + 1$. By the induction, $|S_i| \geq i$ for all $0 \leq i \leq |S|$. It means $|S_{|S|}| \geq |S|$, i.e., $[\![\mathcal{E}]\!]^{|S|} = \nu[\![\mathcal{E}]\!] = \mu[\![\mathcal{E}]\!]$.

□

If $\mathcal{E} \in \mathrm{Sem}_S$ consists of only min and plus, $[\![\mathcal{E}]\!]^{|S|}$ $(= \nu[\![\mathcal{E}]\!] = \mu[\![\mathcal{E}]\!])$ can also be described as a function consisting of min and plus. Here we define a semantic expression which represents it.

For $\mathcal{E} \in \mathrm{Sem}_S$ that satisfies the same properties as those of the above claims,

$$[\![\mathcal{E}]\!]^i(\infty_L) = [\![\mathcal{E}^{(X,i)}\infty_L]\!]$$

holds.

Even if the condition "for all $s \in S$, $(\nu[\![\mathcal{E}]\!])(s) \neq 0$" does not hold, if

for all $s \in S$,
$(\nu(\text{or } \mu)[\![\mathcal{E}]\!])(s) = 0 \Rightarrow$
$([\![\mathcal{E}]\!]F)(s) = 0$ for any $F \in L$, or
$(\nu(\text{ or } \mu)[\![\mathcal{E}]\!])(s) = 0 \Rightarrow [\![\mathcal{E}]\!](s)$ is constantly zero

is satisfied, claim 2 holds, because constant parts are not concerned with fixed point computation. Also, even if $\mathcal{E}$ contains free variables, claim 2 holds (under an appropriate valuation) because they do not change during fixed point computation of $[\![\mathcal{E}]\!]$ like constants.

One of the properties of $\mathit{Abst}(\mathcal{E})$ we mentioned before has the conclusion:

$$[\![E]\!]^\iota = 0 \Leftrightarrow E = 0 \text{ (constant)}.$$

Clearly, $[\![E]\!]^\iota$ is constantly zero if $E = 0$. Using this assumption, we will show the last claim. We first define $PW : (\mathrm{Exp} \cup \mathrm{Sem}_S) \to (\mathrm{Exp} \cup \mathrm{Sem}_S)$ used in the claim.

- $PW(i) = i$
- $PW(Xs) = Xs$
- $PW(\min(\{E_j\})) = \min(\{PW(E_j)\})$
- $PW(\sum(\{E_j\})) = \sum(\{PW(E_j)\})$
- $PW((\mu X\mathcal{E})s) = (\mathcal{E}^{(X,|S|)}\infty_L)s$
- $PW((\nu X\mathcal{E})s) = (\mathcal{E}^{(X,|S|)}\infty_L)s$
- $PW(\mathcal{E})s = PW(\mathcal{E}s)$,

where $i \in \mathbb{N}_\infty$, $X \in \mathrm{PV}$, $\{E_j\}$ is a sequence of Exp, $\mathcal{E} \in \mathrm{Sem}_S$, and $s \in S$. *PW* rewrites fixed point operators to powers.

(claim 3) Let $\mathcal{E}$ be a semantic expression with no free variables such that for any $\iota : \mathrm{PV} \to L$, $E \in \mathrm{Exp}$ such that $E \leq \mathcal{E}$ and $Y \in \mathrm{BndVar}(\mathcal{E})$,

$$\begin{aligned}
&\alpha_L(\iota(X)) = \iota_{\mathcal{E}}(X) \\
&\quad \text{for any } X \in \mathrm{FreeVar}(E) \Rightarrow \\
&\qquad [\![E]\!]^\iota = 0 \Leftrightarrow E = 0 \text{ (constant), and} \\
&\alpha_L(\iota(X)) = \iota_{\mathcal{E}}(X) \\
&\quad \text{for any } X \in \mathrm{FreeVar}(fix_{\mathcal{E}}Y) \Rightarrow \\
&\qquad \alpha_L([\![fix_{\mathcal{E}}Y]\!]^\iota) = \iota_{\mathcal{E}}(Y).
\end{aligned}$$

Then

$$[\![\mathcal{E}]\!]^\iota = [\![\nu to\mu(\mathcal{E})]\!]^\iota.$$

(proof) First note that

$$\nu to\mu(sm_{\mathcal{E}}(X))(s) = PW(sm_{\mathcal{E}}(X))(s) = 0$$

if $sm_{\mathcal{E}}(X)(s) = \iota_{\mathcal{E}}(X)(s) = 0$, and that

$$\begin{aligned}
[\![\nu X\mathcal{E}]\!]^\iota &\leq [\![\mu X\mathcal{E}]\!]^\iota \\
&\leq [\![\nu to\mu(\nu X\mathcal{E})]\!]^\iota \\
&= [\![\nu to\mu(\mu X\mathcal{E})]\!]^\iota \\
&\leq [\![PW(\nu X\mathcal{E})]\!]^\iota \\
&= [\![PW(\mu X\mathcal{E})]\!]^\iota
\end{aligned}$$

for any $\iota \in \mathrm{PV} \to L$, $\mathcal{E} \in \mathrm{Sem}_S$, and $X \in \mathrm{PV}$.

We show the following proposition by induction on the construction of expressions and semantic expressions. For any $E \in \mathrm{Exp}$, $\mathcal{F} \in \mathrm{Sem}_S$ such that $E, \mathcal{F} \leq \mathcal{E}$ and $\iota \in \mathrm{PV} \to L$,

$$\begin{aligned}
&\alpha_L(\iota(Y)) = \iota_{\mathcal{E}}(Y) \\
&\quad \text{for any } Y \in \mathrm{FreeVar}(E) \Rightarrow \\
&\qquad [\![E]\!]^\iota = [\![PW(E)]\!]^\iota, \text{ and} \\
&\alpha_L(\iota(Y)) = \iota_{\mathcal{E}}(Y) \\
&\quad \text{for any } Y \in \mathrm{FreeVar}(\mathcal{F}) \Rightarrow \\
&\qquad [\![\mathcal{F}]\!]^\iota = [\![PW(\mathcal{F})]\!]^\iota.
\end{aligned}$$

The proof of this proposition is in [14].

Then for $E \in \mathrm{Exp}$ such that $E \leq \mathcal{E}$, if $\iota \in \mathrm{PV} \to L$ satisfies $\iota(Y) = \iota_{\mathcal{E}}(Y)$ for any $Y \in \mathrm{FreeVar}(E)$, then $[\![E]\!]^\iota = [\![\nu to\mu(E)]\!]^\iota$, because $[\![E]\!]^\iota \leq [\![\nu to\mu(E)]\!]^\iota \leq [\![PW(E)]\!]^\iota$ and $[\![E]\!]^\iota = [\![PW(E)]\!]^\iota$. For $\mathcal{F} \in \mathrm{Sem}_S$ such that $\mathcal{F} \leq \mathcal{E}$, a similar proposition holds.

□

3.5.4 Complexity

Let l be the length of the target formula, d_n be the nesting depth of fixed point operators, and d_a be the alternating depth.

Iteration in abstract computation has the same complexity $O((l|S|^2) \cdot |S|^{d_a}))$ as computation of μ- or ν-operators under the ordinary semantics since there are only two values 0 and ∞. Transforming expressions by *Abst* and $\nu to\mu$ has the complexity $O(l|S|^2)$. In the last iteration in *comp_ni*, each fixed point operator iterates at most $|S|$ times by claim 2, so its complexity is $O((l|S|^2) \cdot (|S|^{d_n}))$.

The total complexity is $O((l|S|^2) \cdot (|S|^{d_n}))$ after all, since $d_a \leq d_n$ in general.

3.6 Algorithm for Semantic Expressions *with* Implications and its Correctness

In this section, we introduce a model checking algorithm for semantic expressions with implications. Except for ν-operators, the algorithm is the same as the one in Section 3.5. For ν-operators, we introduce a translation procedure to obtain implication-free expressions.

3.6.1 Algorithm

The whole algorithm is *comp_fml′* in Figure 2. The procedure *comp′* in Figure 5 is the same as *comp* in Figure 3 except for the implication operator and the ν-operator. Implications are computed along the definition of semantics. The ν-operator is handled by the function *trns*.

The basic idea of the algorithm is as follows: when implication operators exist, we cannot compute the least fixed point by replacing ν-operators with μ-operators any more. Therefore we use the ordinary procedure as the basis of our algorithm: starting from the zero function 0_L, we iterate the computation toward the fixed point. Since this procedure does not terminate in general, we need a way to accelerate the computation. For this purpose, we utilize the algorithm in the previous section. In order to do so we need to remove implication operators from the formula. This is done by the function *trns*.

Before explaining the function *trns*, let us suppose for the moment that we could freely handle negative values and the subtraction operation in the min-plus algebra. Let $\mathcal{E} \in \mathrm{Sem}_S$ and $\mathrm{FreeVar}(\mathcal{E}) = \{X\}$. Recall that the function that maps $F \in L$ to $\lambda s \in S.\ [\![\mathcal{E}[X \mapsto F]]\!](s)$ is denoted by $[\![\mathcal{E}]\!]$ and its least fixed point is denoted by $\nu[\![\mathcal{E}]\!]$.

```
$comp'(\mathcal{E} : \mathrm{Sem}_S) : L$
  for each $s \in S$ do $LV'(\mathcal{E}(s)) := comp'(\mathcal{E}(s))$
  return $\lambda s \in S.\ LV'(\mathcal{E}(s))$

$comp'(E : \mathrm{Exp}) : \mathbb{N}_\infty$
  case $(E)$ of
    $i \Rightarrow LV'(E) := i$
    $Xs \Rightarrow$ skip
    $\min(\{E_j\}_{j\in I}) \Rightarrow LV'(E) := \min_{j\in I} comp'(E_j)$
    $\sum(\{E_j\}_{j\in I}) \Rightarrow LV'(E) := \sum_{j\in I} comp'(E_j)$
    $(E_- \Rightarrow E_+) \Rightarrow$ if $comp'(E_+) \leq comp'(E_-)$ then $LV'(E) := 0$
                                                   else $LV'(E) := comp'(E_+) - comp'(E_-)$
    $\mu X\mathcal{E}s \Rightarrow LV'(E) := iter'(X,\ \mathcal{E},\ \infty_L)(s)$
    $\nu X\mathcal{E}s \Rightarrow LV'(E) := comp\nu(X,\ \mathcal{E})(s)$
  return $LV'(E)$

($iter'$ is defined in the same manner as $iter$)

$comp\nu(X : \mathrm{PV},\ \mathcal{E} : \mathrm{Sem}_S) : L$
  for each $s \in S$ do $LV'(Xs) := 0$
  while $\exists s \in S$ s.t. $comp'(\mathcal{E})(s) \neq LV'(Xs)$ do
    for each $s \in S$ do $LV'(Xs)\ += comp\_ni(trns(\nu X\mathcal{E},\ 0_L,\ \mathrm{BndVar}(\nu X\mathcal{E})))(s)$
  return $\lambda s \in S.\ LV'(Xs)$

$trns(\mathcal{E} : \mathrm{Sem}_S,\ F : L,\ V : \mathscr{P}(\mathrm{PV})) : \mathrm{Sem}_S$
  var $\mathcal{E}' : \mathrm{Sem}_S$
  for each $s \in S$ do $\mathcal{E}'(s) := trns(\mathcal{E}(s),\ F(s),\ V)$
  return $\mathcal{E}'$

$trns(E : \mathrm{Exp},\ n : \mathbb{N}_\infty,\ V : \mathscr{P}(\mathrm{PV})) : \mathrm{Exp}$
  var $E' : \mathrm{Exp}$
  case $(E)$ of
    $i \Rightarrow E' := i - n$
    $Xs \Rightarrow$ if $X \in V$ then $E' := \sum(Xs,\ LV'(Xs) - n)$ else $E' := LV'(Xs) - n$
    $\min(\{E_j\}_{j\in I}) \Rightarrow E' := \min(\{trns(E_j,\ n,\ V)\}_{j\in I})$
    $\sum(\{E_j\}_{j\in I}) \Rightarrow E' := \sum(\{trns(E_j,\ LV'(E_j),\ V)\}_{j\in I},\ \sum_{j\in I} LV'(E_j) - n)$
    $(E_- \Rightarrow E_+) \Rightarrow$ if $LV'(E_+) < LV'(E_-)$ then $E' := 0$ else $E' := trns(E_+,\ n + LV'(E_-),\ V)$
    $\mu X\mathcal{E}s \Rightarrow E' := \sum(\mu X trns(\mathcal{E},\ \lambda s \in S.\ LV'(Xs),\ V))s,\ LV'(Xs) - n)$
    $\nu X\mathcal{E}s \Rightarrow E' := \sum(\nu X trns(\mathcal{E},\ \lambda s \in S.\ LV'(Xs),\ V))s,\ LV'(Xs) - n)$
  return $E'$
```

Figure 5: Computation of semantic expressions *with* implications

Suppose that we are computing the least fixed point $\nu[\![\mathcal{E}]\!]$ and its current approximation is G. Instead of simply computing $[\![\mathcal{E}]\!](G)$ as the next approximation, we consider the "parallel shift" l of $[\![\mathcal{E}]\!]$ with the amount of G, which could be defined as $l(F) = [\![\mathcal{E}]\!](F+G)-G$, if we were able to use subtraction. It is easy to see that $\nu[\![\mathcal{E}]\!] - G$ is the least fixed point of l. This argument suggests that we should find an implication-free semantic expression $\mathcal{E}'$ such that $[\![\mathcal{E}']\!]$ effectively (under-)approximates the parallel shift l. The least fixed point $\nu[\![\mathcal{E}']\!]$ can be computed with the algorithm in the previous section, and we can obtain the next approximation by adding $\nu[\![\mathcal{E}']\!]$ to the current approximation.

The function *trns* computes the above-mentioned $\mathcal{E}'$ for given $\mathcal{E}$. The desired property can be written as $[\![\mathcal{E}'[X \mapsto F]]\!](s) = [\![\mathcal{E}[X \mapsto F+\iota_{LV'}]]\!](s) - LV'(s)$ for $s \in S$, since the current approximation value $G(s)$ is stored as $LV'(Xs)$ in the pseudocodes. For translating a subexpression E of $\mathcal{E}(s)$ appropriately, the function *trns* keeps the number n. The intention is that $[\![E']\!] = [\![E]\!] - n$ holds during the computation. If this is satisfied for every subexpression, the desired property for $\mathcal{E}'$ is satisfied. Unfortunately, it is not always the case during the iteration: the equation can break for the expression of the form $E_- \Rightarrow E_-$ when $LV'(E_+) < LV'(E_-)$. However, we can show that the equations for all subexpressions eventually hold and then the iteration terminates.

3.6.2 Correctness

Due to space limitation, we do not present a detailed correctness proof of the algorithm. Instead, a brief sketch is given below.

We need to establish the following three facts:

(1) For subtractions of the form $a-b$ in the function *trns*, $a \geq b$ always holds.

It should hold since the result is an element of $\mathbb{N}_\infty$. Note that the subtraction in the function *comp′* is all right since $comp'(E_+) \geq comp'(E_-)$ is guaranteed by the condition of the if-statement.

(2) For $\nu X\mathcal{E} \in \mathrm{Sem}_S$, we need to show that the valuation for X calculated by the function *compν* does not exceed the correct valuation $[\![\nu X\mathcal{E}]\!]$.

It is clear from the condition of the while statement in the function *compν* that the calculated valuation becomes a fixed point when the computation terminates. Therefore the fact (2) is enough to show the result is the least fixed point.

(3) The iteration terminates.

There is a while loop in the function *compν*. The loop must be terminated.

To see the fact (1), we can show that the following conditions are satisfied during the computation.

- When the function *trns* is called for $\mathcal{E} \in \mathrm{Sem}_S$, $F(s) \leq LV'(\mathcal{E}(s))$ holds for any $s \in S$.
- When the function *trns* is called for $E \in \mathrm{Exp}$, $n \leq LV'(E)$ holds.
- For the subtraction in the form $a - b$, $a \geq b$ holds.

These conditions can be established as invariants: the function *trns* is first called in the function *compν*, for an element of Sem_S. The first condition need to be checked here but it trivially holds. Other calls of the function *trns* reside in itself, and assuming the conditions at the top of the function, one can easily checked that the subsequent calls of the functions satisfy the conditions.

For the fact (2), we need to check the following conditions:

- For the return value $\mathcal{E}'$ of the function *trns* for $\mathcal{E} \in \mathrm{Sem}_S$, $s \in S$ and $\iota : PV \to L$,
$$[\![\mathcal{E}']\!]^\iota(s) + LV'(Xs) \leq [\![\mathcal{E}]\!]^{\iota'}(s).$$
- For the return value E' of the function *trns* for $E \in \mathrm{Exp}$,
$$[\![E']\!]^\iota + LV'(E) \leq [\![E]\!]^{\iota'}.$$

where $\iota' = \iota[Y \mapsto \iota(Y) + \iota_{LV'}(Y) \mid Y \in V]$. These conditions can be checked in the same manner as in the fact (1).

For the fact (3), first note that the values for the negative part converge after finitely many iterations since they monotonically decrease and $\mathbb{N}_\infty$ is well-founded. All expressions that are handled by the function *trns* reside in the positive part. If an expression of the form $E_- \Rightarrow E_+$ is handled by *trns*, $LV'(E_+)$ increases and $LV'(E_-)$ decreases during the iteration. Therefore once $LV'(E_+) \geq LV'(E_-)$ holds, it continuously holds until the end of the computation. Moreover, if there are expressions that satisfy $[\![E_+]\!] \geq [\![E_-]\!]$ and $LV'(E_+) < LV'(E_-)$ at some point, at least one of such expressions satisfies $LV'(E_+) \geq LV'(E_-)$ in the next iteration step: otherwise the approximated parallel shift would be the same as that of the previous iteration step. In such a case, LV' does not change from the previous value, which contradicts the fact that the computation has not reached the fixed point. Therefore after finitely many iterations, the function *trns* takes the correct branch for all expressions of the form $E_- \Rightarrow E_+$. When this is achieved, the exact parallel shift is computed, and it reaches the least fixed point at the next iteration step.

3.6.3 Complexity

Once the values for the negative part converge, the iteration terminates in at most $|S|n$ times, where n

is the number of implications in the formula. However, at the moment, we do not have estimation for the computation required until the values for the negative part are fixed. It is future work to estimate it for the current algorithm or to improve the algorithm so that the complexity can be estimated.

4 Implementation

In this section, we introduce the prototype implementation of the algorithm in the previous section. We have implemented the algorithm without implication $\rightarrow$. In Section 4.1, we briefly describe the program implementing the algorithm. And in Section 4.2, we show the result of an experiment executing the program on simple examples.

4.1 Program

We have implemented the algorithm without implication $\rightarrow$ using C++, where negation is allowed in some cases.

Elements of $\mathbb{N}_\infty$ other than ∞ are implemented as multiple-word integers. The set of states is implemented as an array of integers. Modalities are implemented as arrays of variable-length arrays of pointers to states, and valuations ι as arrays of pointers to $\mathbb{N}_\infty$. Formulas and expressions are implemented as tree structures. Semantic expressions are implemented as arrays of expressions. Abstract computation and concrete computation are realized by the same procedure with a switch, which changes the valuation of a varialbe or constant to 0 or ∞ in the case of abstract computation. And LV, AV are implemented as properties of expressions.

4.2 Preliminary Experimental Results

We have executed the above program on the following three formulas:

- $\mu X(a \vee \langle f\rangle X)$
- $\mu X(a \vee \langle f\rangle(1 \wedge X))$
- the formula (*) in Section 5

to see the efficiency of the program and the algorithm. Transition systems in Kripke structures we used are binary trees, whose depth we changed from 2 to 16. Thus, the number of states is $2^{depth} - 1$.

We show the result on Table 1 and in Figure 6. Both on Table 1 and in Figure 6, execution time was measured by Win32 API, timeGetTime in milliseconds. Table 1 shows execution time in columns "reachability", "s.p.length" and "min-access". In column "reachability", we show execution time to interpret the formula $\mu X(a \vee \langle f\rangle X)$ which represents the reachability to states at which a holds. In column "s.p.length", we show execution time to interpret the formula $\mu X(a \vee \langle f\rangle(1 \wedge X))$. Remember that it represents reachability with shortest path length. In column "min-access", we show execution time to interpret the formula $\nu X(\mathbf{halt} \vee (\mathbf{access\_x} \wedge \langle f\rangle(1 \wedge X)) \vee (\neg\mathbf{access\_x} \wedge \langle f\rangle X))$ which represents the minimum number of accesses as in the next section. In Figure 6, horizontal axis shows the depth of binary tree.

Execution time is about proposional to the number of states in each case. The execution times of "s.p.length" is in double the time of "reachability" although the semantics is extended to infinite space.

depth	reachability	s.p.length	min-access
2	0	0	0
3	0	0	16
4	0	0	32
5	15	31	110
6	16	63	250
7	47	125	531
8	125	203	1,203
9	438	640	2,156
10	1,406	2,094	2,500
11	3,031	2,093	6,438
12	8,469	7,281	19,907
13	14,921	27,093	53,641
14	31,078	65,096	106,844
15	103,923	169,251	310,578
16	256,892	378,426	620,624

Table 1: Execution Time

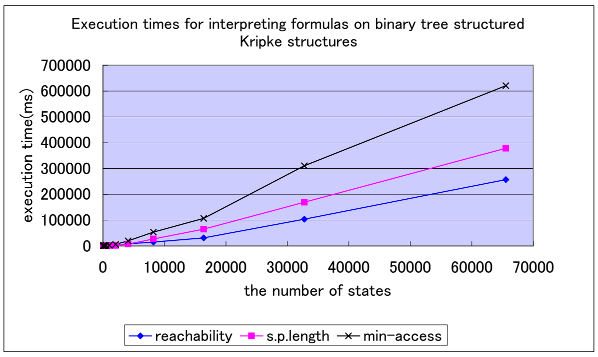

Figure 6: Execution Time

5 Data Flow Analysis

As mentioned in the introduction, Lacey et al. used CTL to specify conditions on a control flow graph for the purpose of program transformation [6]. In this short section, let us explain how our semantics can be applied to their framework.

For example, assume that the proposition **access_x** holds at a node in a control flow graph if the variable **x** is accessed at the node. The minimum number of accesses to the variable **x** on an execution path starting from a node can then be expressed by the following formula under our semantics:

$$\nu X(\mathbf{halt} \vee ((\mathbf{access\_x} \to \langle f\rangle(1 \wedge X)) \wedge (\neg\mathbf{access\_x} \to \langle f\rangle X))).$$

Note that if there exists an execution path on which **x** is never accessed, the formula is interpreted as 0. The ν-operator well matches this situation.

The above formula can be rewritten to the following one which does not contain implication $\to$.

$$\nu X(\mathbf{halt} \vee (\mathbf{access\_x} \wedge \langle f\rangle(1 \wedge X)) \vee (\neg\mathbf{access\_x} \wedge \langle f\rangle X)) \qquad (*)$$

This is further rewritten to the following equivalent formula.

$$\mu X(\nu Y(\mathbf{halt} \vee (\neg\mathbf{access\_x} \wedge \langle f\rangle Y)) \vee (\mathbf{access\_x} \wedge \langle f\rangle(1 \wedge X)) \vee (\neg\mathbf{access\_x} \wedge \langle f\rangle X))$$

Note that the whole formula is led by the μ-operator while the ν-operator is used in the subformula $\nu Y(\mathbf{halt} \vee (\neg\mathbf{access\_x} \wedge \langle f\rangle Y))$, which means that there exists a finite or infinite execution path on which **access_x** never holds. This subformula can be interpreted by the ordinary semantics as true or false. In fact, the algorithm developed in this paper makes this kind of transformation on formulas automatically.

For a more complex example, assume that the proposition **update_x** holds at a node where the variable **x** is updated. The minimum number of accesses to the variable **x** after each update of **x** is then expressed as follows:

$$\varphi \vee \nu Y((\mathbf{update\_x} \wedge (\mathbf{halt} \vee \langle f\rangle\varphi \vee \langle f\rangle Y)) \vee (\neg\mathbf{update\_x} \wedge \langle f\rangle Y)),$$

where φ is the following formula.

$$\nu X((\mathbf{access\_x} \to 1) \wedge (\mathbf{halt} \vee \mathbf{update\_x} \vee \langle f\rangle X))$$

The maximum number of accesses can be expressed by the following formula, if we introduce another modal opeartor $\{f\}$ corresponding to the max operator on $\mathbb{N}_\infty$.

$$\nu X(\mathbf{halt} \vee (\mathbf{access\_x} \wedge \{f\}(1 \wedge X)) \vee (\neg\mathbf{access\_x} \wedge \{f\}X))$$

In this way, for this kind of application, our framework should be extended to include the max operator in order to model check formulas as above.

We have constructed an algorithm for extended semantic expressions which contain the max operator. The algorithm is almost the same as that for min-plus algebra. For an expression of the form $\max(\{E_j\}_{j\in I})$, *trns* chooses a subexpression $E \in \{E_j\}_{j\in I}$ and changes $\max(\{E_j\}_{j\in I})$ into E. Then translated expressions have no max operators. Thus one can iteratively compute the least fixed points as in the case of min-plus algebra. However, the correctness proof of the algorithm is not completed yet.

6 Discussions and Future Work

In this paper, we gave the semantics that interpreted modal μ-calculus formulas on min-plus algebra. It interpreted disjunctions by min, and conjunctions by plus. Using this semantics, we succeeded in expressing numerical measures such as the length of the shortest path and the number of states satisfying some property. However, it is not yet clear how useful the semantics we gave in this paper is since it is not easy to imagine what a formula means intuitively. Especially, finding practical applications of the ν-operator is a future research issue.

The discussions so far also raise the future issue to extend or change the semantics of this paper. Especially, there are many alternative interpretations of modal operators. For example, when weights are defined on transitions in Kripke structures, modal operators should be interpreted by taking weights into account. Another possibility is to define semantics that contains not only min and plus, but also max.

Nishizawa et al. [4, 7] gave a general-purpose framework for modal μ-calculus using elements of complete Heyting algebra as truth values of modal μ-calculus formulas. The algebra $\mathbb{N}_\infty$ used in this paper also forms complete Heyting algebra. But the semantics that is obtained by applying the framework of Nishizawa et al. to $\mathbb{N}_\infty$ is different from the semantics of this paper, because the former interprets disjunctions by sup, while the latter by plus. This difference is also clear since the semantics by Nishizawa et al. is sound with respect to the axioms of intuitionalistic modal logic, while the semantics of this paper is not. Moreover, since the semantics by Nishizawa et al. allows Kripke structures to be extended to multi-valued relations and simulation relations, which are not included by the semantics of this paper. Therefore, the research by Nishizawa et al. and this research are not easily comparable.

However, it is possible to introduce new semantics that include both algebraic structures as follows. Think of an algebraic structure $(L, \bigvee, \bigwedge, \otimes, \twoheadrightarrow, \bigotimes)$ such that $(L, \bigvee, \bigwedge)$ is a complete lattice, $(L, \otimes)$ is a semigroup, $a \otimes b \leq c \iff b \leq a \twoheadrightarrow c$ holds, and $\otimes$ distributes over $\bigvee$. And assume that $\otimes$ can be extended to the operator $\bigotimes$ that allows an arbitrary number of arguments. Based on this algebraic structure, semantics of modal μ-calculus can be given as below.

$$
\begin{array}{lcl}
[\![\varphi \vee \psi]\!]_{K,V} & \overset{\text{def}}{=} & [\![\varphi]\!]_{K,V} \vee [\![\psi]\!]_{K,V} \\
[\![\varphi \wedge \psi]\!]_{K,V} & \overset{\text{def}}{=} & [\![\varphi]\!]_{K,V} \otimes [\![\psi]\!]_{K,V} \\
[\![\varphi \to \psi]\!]_{K,V} & \overset{\text{def}}{=} & [\![\varphi]\!]_{K,V} \twoheadrightarrow [\![\psi]\!]_{K,V} \\
[\![\mu X.\varphi]\!]_{K,V} & \overset{\text{def}}{=} & \bigwedge\{\, W \in [S, L] \mid \\
 & & \quad [\![\varphi]\!]_{K,V[X \mapsto W]} \leq W \,\} \\
[\![\nu X.\varphi]\!]_{K,V} & \overset{\text{def}}{=} & \bigvee\{\, W \in [S, L] \mid \\
 & & \quad W \leq [\![\varphi]\!]_{K,V[X \mapsto W]} \,\} \\
[\![\Diamond\varphi]\!]_{K,V}(s) & \overset{\text{def}}{=} & \bigvee\{\, (s \to t) \otimes [\![\varphi]\!]_{K,V}(t) \mid \\
 & & \quad t \in S \,\} \\
[\![\Box\varphi]\!]_{K,V}(s) & \overset{\text{def}}{=} & \bigotimes\{\, (s \to t) \twoheadrightarrow [\![\varphi]\!]_{K,V}(t) \mid \\
 & & \quad t \in S \,\}
\end{array}
$$

The semantics of this paper is obtained by applying this semantics to the algebra $(\mathbb{N}_\infty, \min, \sup, \text{plus}, \Rightarrow, \Sigma)$, while the semantics by Nishizawa et al. by applying it to $(L, \bigvee, \bigwedge, \wedge, \Rightarrow, \bigwedge)$, which is made from the complete Heyting algebra $(L, \bigvee, \Rightarrow)$.

References

[1] François L. Baccelli, et al. *Synchronization and Linearity, An Algebra for Discrete Event Systems,* Wiley, 1992.

[2] Edmund M. Clarke, Orna Grumberg and Doron Peled. *Model Checking,* MIT Press, 1999.

[3] Clarke, E.M., Grumberg, O. and Long, D. Model Checking and Abstraction, *ACM Transactions on Programming Languages and Systems,* vol.16, No.5, 1994, pp.1512–1542.

[4] Yukiyoshi Kameyama, Yoshiki Kinoshita, and Koki Nishizawa. Weighted Kripke Structures and Refinement of Models, *23rd Japan Society for Software Science and Technology,* September, 2006.

[5] Dexter Kozen. Results on the Propositional μ-Calculus, *Theoretical Computer Science,* Vol.27, No.3, 1983, pp. 333–354.

[6] David Lecay, Neil D. Jones, Eric van Wyk, and Carl Christian Frederiksen. Compiler Optimization Correctness by Temporal Logic, *Higher-Order and Symbolic Computation,* Vol.17, 2004, pp. 173–206.

[7] Koki Nishizawa, Yukiyoshi Kameyama and Yoshiki Kinoshita. Simulations of Multi-Valued Models for Modal μ-Calculus, *Programming Science Technical Report,* AIST-PS-2007-005, 2007.

[8] Mooly Sagiv, Thomas Reps and Reinhard Wilhelm. Parametric Shape Analysis via 3-valued Logic, *ACM Transactions on Programming Languages and Systems,* Vol.24, No.3, 2002, pp. 217–298.

[9] Toshifusa Sekizawa, Yoshinori Tanabe, Yoshifumi Yuasa, and Koichi Takahashi. MLAT: A Tool for Heap Analysis based on Predicate Abstraction by Modal Logic, *The IASTED International Conference on Software Engineering (SE 2008),* (to appear).

[10] Yoshinori Tanabe, Toshifusa Sekizawa, Yoshifumi Yuasa, Koichi Takahashi. The Method of Heap Verification Using Modal Logic (in Japanese), *3rd Dependable Software Workshop (DSW'06),* January 27th, 2006, pp. 39–50.

[11] Fleming Nielson, Hanne Riis Nielson, Chris Hankin. *Principles of Program Analysis,* Springer-Verlag, 1999.

[12] E. A. Emerson and C-L. Lei. Efficient model checking in fragments of the propositional mucalculus. *Logic in Computer Science,* 1986, pp. 257–278.

[13] D. Long, A. Browne, E. Clarke, S. Jha, and W. Marrero. An improved algorithm for the evaluation of fixpoint expressions. vol. 818 of *Lecture Notes in Computer Science*, Springer-Verlag, 1994, pp. 338–350.

[14] Dai Ikarashi. Modal μ-calculus on Min-Plus Algebra $\mathbb{N}_\infty$ and its Applications. Master's thesis, Graduate school of Information Science and Technology, The University of Tokyo, Japan, 2008.