プログラミング-言語・基礎・実践- 16－12
(1994. 3. 10)

オブジェクトの生成、消滅を表現可能な論理体系 DOL

宮川 晋　　米崎 直樹
東京工業大学 工学部 情報工学科
〒152 東京都 目黒区 大岡山 2-12-1

あらまし

オブジェクトの生成や消滅があるリアクティブシステムの仕様記述、その仕様検証、仕様に適したシステムの自動合成、に適した新しい時相論理 DOL(Dynamic Object Logic) を提案し、これをもちいた仕様記述例を示す。DOL は、三値化された意味論をもつ一階述語時相論理である。時相演算子として Until オペレータならびに Next オペレータをもち、かつオブジェクトの生成や消滅を陽に表現する新しいオペレーターである $\triangleright$ ならびに $\triangleleft$ をもつ。三値論理の意味論は、既存の難解な存在論 (ontology) を用いずにオブジェクトの存在について自然に記述をすることを可能にする。

和文キーワード　　時相論理、オブジェクト、仕様記述、検証、合成、リアクティブシステム

DOL:Logic of object creation and destruction

Shin MIYAKAWA, Naoki YONEZAKI
Department of Computer Science, Tokyo Institute of Technology
2-12-1 Ookayama Meguro-ku Tokyo 152, Japan

Abstract

We give a new version of temporal logic called DOL organized for specification, verification and synthesis of reactive system with dynamic objects. DOL is the 3 valued first order temporal logic with until operator and new modal operators enable us to deal with object creation and destruction explicitly. The 3 valued evaluation system is suitable for characterizing existence of object naturally without complicated ontology. Some examples of reactive system specifications are also presented.

英文 key words　　temporal-logic, object, specification, verification, synthesis, reactive-system

DOL:Logical model formalization of object creation and destruction

Shin MIYAKAWA † Naoki YONEZAKI†‡ *

†Department of Computer Science
Tokyo Institute of Technology
2-12-1 Ookayama Meguro-ku Tokyo 152, Japan

‡School of Information Science
Japan Advanced Institute of Science and Technology, Hokuriku
15 Asahidai Tatsunokuchi Nomi-gun Ishikawa 923-12, Japan

Feb. 1994

Abstract

We give a new version of temporal logic called DOL organized for specification, verification and synthesis of reactive system with dynamic objects. DOL is the 3 valued first order temporal logic with until operator and new modal operators enable us to deal with object creation and destruction explicitly. The 3 valued evaluation system is suitable for characterizing existence of object naturally without complicated ontology. Some examples of reactive system specifications are also presented.

1 Introduction

The concepts of object **creation** and **destruction** are important to specify dynamic systems.

For example, Let us think of a city traffic system. If two cars collide at an intersection and both of them are crashed, then, they must be deleted from the system. To specify such a system, we have to **create** some numbers of cars and let them run in the system, and when the accident occurs, collided cars must be **removed** from the system. These concepts are included in some object oriented programing languages as **new** and **delete** operation respectively and widely used in practice.

Usually most formalization of **object orientation** are made in the algebraic framework (e.g. **object calculus** [11].),and less work has been done in logical way as far as we know.

If something is create in a system, we can naturally recognize that state is changed. In algebraic approach, an expression itself can not describe state change. It is described by the meta-level notion of rewriting.

Temporal logic is one of the most useful conceptual tool in computer science, with which we can describe a statement related to **tense** or **time** explicitly in its language. Especially the (versions of)propositional temporal logic with *until* and *next* operator (PTL)[4] is useful for specifying, verifying and also syntesising reactive systems ([3],[5],[6]), however ,it has not yet enough power to specify them, in the sense that we can not express a creation and a destruction of the system objects in its framework.

In this paper, we try to capture and formalize such notions in temporal logic framework by introducing some new modal operators $\triangleright$ and $\triangleleft$ which express object creation ("**new**" in the object oriented notion.) and destruction ("**delete**"), respectively.

Introducing these notation, of its accord, we have to face the problem of "existence" of object. In the previous traffic system example again, if all cars are broken, **until** the new cars are loaded into the system, the specification of the behavior of cars is **nonsense**.

Using such a new modal operators, although we can express "destruction" of object easily, we have to deal with predicates on the empty object domain. In the case of usual logic, the case of empty domain is excluded from semantic definition, since to evaluate a predicate in such case is nonsense literally. To deal with in the case of

*E-mail:{miyakawa|yonezaki}@cs.titech.ac.jp

non-existence object, 3 valued evaluation system is introduced.

It enable us to express naturally many property of object domain ,(e.g. object existence domain emptiness.)

2 DOL

We introduced a logic called DOL (Dynamic Object Logic) ,inwhich we formalize the following requirements.

1. *To make sense of object creation and destruction, notion of time is inevitable.*

   We adopt the predicate temporal logic with Next and Until operators as the base of our logic.

2. *If once an object is created, it has been existing until it is destruct-ed. Any objects can not exists until it is created and since it is destruct-ed.*

   We employ the notation $\triangleright$ and $\triangleleft$ as object creation and destruction respectively. They have default-logical semantics that embody this principle.

3. *Evaluation of any predicate with non-existing object makes no sense.*

   To evaluate a predicate with objects out of domain of the interpretation, 3 valued ("True", "False", "Undefined") semantics is introduced.

4. *Concept of "something new" shoud be supported. The name of it is not care about.*

   In DOL, new quantifier **new** are introduced. It assign something new anonymous objcet to a variable. Semantic evaluation system automatically created its unique indetifier and we do not use it.

2.1 Temporal Logic with Next and Until operators

Temporal Logic with Next and Until operators [2] is a version of Modal Logic [1].

In this framework, the formulas $\Box P$ and $\Diamond Q$ can be read as "From now, the formula P holds forever", "The formula Q will hold sometime in the future",respectively.

The operator $\bigcirc$ is called "Next". The formula $\bigcirc P$ means "at the next time, P holds.".

The "Until Operator" [ ] is a 2-ary operator. The formula $[B]A$ means "A until B", that is, "A holds until B holds." [1]

With these operators, we can express temporal behaviors of objects and the order of events explicitly.

Note that we have chosen weak until operator in DOL different from [2].

2.2 $\triangleright$ and $\triangleleft$

$\triangleright$ and $\triangleleft$ are distinctive features of DOL. They mean "object creation" and "object destruction",respectively. We present two kinds of these operators: weak one and strong one.

$\dot{\triangleright}(\{a\})$ (weak object creation) means "If a does not exists now, a will have to be created at the next time". $\ddot{\triangleright}(\{a\})$ (strong object creation) means "a does not exists now and a will have to be created at the next time". $\dot{\triangleleft}(\{a\})$ (weak object destruction) means "If a exists now, a will have to be deleted at the next time". $\ddot{\triangleleft}(\{a\})$ (strong object destruction) means "a exists now and a will have to be deleted at the next time".

Incorprating the seconde requirement, we adopt default-semantics for object existence. As a result, the logic become non-monotinic. We argue this non-monotonicity in the formal semantics definition, later.

2.3 3 valued semantics with variable domain

Usual logic uses binary valued semantics and predicate interpretation excluds the case that parameter values are out of domain. DOL have $\triangleleft$ operator and it enable us to delete any objects from the domain of interpretation. Thus DOL employ 3 valued semantics to treat with the case.

To interpret a DOL formula, we use a series of domains $d(t)$ depending on the time point t $d(t)$ expresses the set of existing objects at t. For all thepredicates, their truth-values are defined as **True** or **False** with objects in $d(t)$ but **Undefined** with objects out of $d(t)$.

2.4 Quantifier "new"

$\triangleright$ operator requires the names of objects to be created. It enable us to express things like "at the next time, object a will be created, and its

[1] Until operator is written as AUB or A**until**B in other notation.

nature is ....". (For example this can be written as $\triangleright(\{a\}) \wedge \bigcirc P(a)$ in DOL.)

On the contrary, it is often happens that we also want to express that "at the next time, *something new* will be created and its nature is ...." $\cdots(A)$.

To enable us to express this in DOL, we employ **new** quantifier. **new** quantifier assigns a brand new object to a variable. For example, the DOL formula $\mathbf{new}x(\triangleright(\{x\}) \wedge \bigcirc P(x))$ means (A).

3 DOL syntax

The syntax of the DOL is a version of the first order predicate logic with until operator , new modal operators ($\triangleright$,$\triangleleft$) and new quantifier (**new**). $\triangleright$ and $\triangleleft$ express object generation and destruction, respectively and **new** is for anonymous object quantification.

3.1 Syntax

Def 3.1 (symbols) We employ logical symbols as follows.
Predicate-symbol $= P, Q, R, \ldots,$
Variable-symbol $= x, y, z, \ldots,$
Constant-symbol $= a, b, c, \ldots, c_0, c_1, \ldots,$
Function-symbol $= f, f_0, f_1, \ldots,$
Connective $= \{\wedge, \vee, \neg, \rightarrow, \leftarrow, \leftrightarrow\},$
Quantifier $= \{\forall, \exists, \mathbf{new}\},$
Constant-Proposition $= \{\mathbf{True}, \mathbf{False}\}$
Modal-Operator $= \{[,], \dot{\triangleright}, \ddot{\triangleright}, \dot{\triangleleft}, \ddot{\triangleleft}, \Box, \Diamond, \bigcirc, \}.$

Def 3.2 (Term) **Term** :=
constant-symbol|**variable-symbol**
|**function-symbol**$(\mathbf{Term}_1, \ldots, \mathbf{term}_n)$

Def 3.3 (Term set) **term-set-expression** :=
$\emptyset|\{\mathbf{term}_1, \mathbf{term}_2, \ldots, \mathbf{term}_n\}$

Def 3.4 (Formula) **formula** :=
Predicate$(\mathbf{term}_1, \ldots, \mathbf{term}_n)$(Atomic)
|$\neg$**formula** (Negation)
|**formula**$_1 \wedge$ **formula**$_2$(Conjunction)
|$\forall$**variable-symbol formula** (For all)
| $\bigcirc$ **formula** (Next)
|[**formula**$_1$]**formula**$_2$ (Until)
| **new variable-symbol formula**
| $\dot{\triangleright}$**(term-set-expression)**
(Weak Object Creation)
| $\ddot{\triangleright}$**(term-set-expression)**
(Strong Object Creation)
|$\dot{\triangleleft}$**(term-set-expression)**
(Weak Object Destruction)
|$\ddot{\triangleleft}$**(term-set-expression)**
(Strong Object Destruction)
|**True**

Def 3.5 (The set of free variables of a term t) The set $FV(t)$ of free variables of a term t is defined as usual.
$FV(X_i) = \{X_i\}$ (X_i:variable),
$FV(c_i) = \emptyset$ (c_i:constant),
$FV(f(t_1, \ldots, t_n)) = FV(t_1) \cup \ldots \cup FV(t_n)$.

Def 3.6 (The set of free variables of a set σ of terms)
$FV(\{t_1, \ldots, t_n\}) = FV(t_1) \cup \ldots \cup FV(t_n)$,
$FV(\emptyset) = \emptyset$

Def 3.7 (The set of free variables of a formula ϕ)
Let ϕ, ψ be formulas. The set $FV(\phi)$ of free variables of a formula ϕ is defined as follows.

1. $FV(P(t_1, \ldots, t_n)) = FV(t_1) \cup \ldots \cup FV(t_n)$, $FV(\neg\phi) = FV(\phi)$, $FV(\phi \wedge \psi) = FV(\phi) \cup FV(\psi)$, $FV(\bigcirc\phi) = FV(\phi)$ $FV([\phi]\psi) = FV(\phi) \cup FV(\psi)$, $FV(\forall X\phi) = FV(\phi) - \{X\}$,$FV(\mathbf{True}) = \emptyset$
2. $FV($ **new** $X\phi) = FV(\phi) - \{X\}$
3. $FV(\dot{\triangleright}(\alpha)) = FV(\alpha)$
4. $FV(\ddot{\triangleright}(\alpha)) = FV(\alpha)$
5. $FV(\dot{\triangleleft}(\alpha)) = FV(\alpha)$
6. $FV(\ddot{\triangleleft}(\alpha)) = FV(\alpha)$

Def 3.8 (Closed formula) A term t or a formula ϕ called **closed** if $FV(t) = \emptyset$, $FV(\phi) = \emptyset$, respectively.

Def 3.9 (Abbreviation) We employ some abbreviations as follows,
$\phi \vee \psi \equiv \neg(\neg\phi \wedge \neg\psi)$ (disjunction), $\phi \rightarrow \psi \equiv \neg\phi \vee \psi$ (implication), $\phi \leftarrow \psi \equiv \psi \rightarrow \phi$, $\phi \leftrightarrow \psi \equiv (\phi \rightarrow \psi) \wedge (\phi \leftarrow \psi)$ (equivalence), $\exists X\phi \equiv \neg(\forall X(\neg\phi))$, **False** $\equiv \neg$**True**. $\Box\phi \equiv$ [**False**]ϕ (Always, Forever), $\Diamond\phi \equiv \neg\Box\neg\phi$ (Sometimes, in the future),

4 3 valued first order logic

We will use a 3 valued first order logic (for example, see [8]) as a meta language to define the semantics of DOL formula. Since this logic is not our main subject of this paper, we outlined it briefly. We take usual syntax of first order logic as the syntax of the 3 valued first order logic. The difference between the semantics of 3valued first order logic and usual one, is only the value assignment for predicate symbol and the meaning of connectives.

We take the semantics of them as follows:

Def 4.1 (Semantics)
For a predicate symbol P, semantic function v gives a truth value assignment function for predicates symbols.

$$v(P) : D^n \rightarrow \{\mathbf{t}, \mathbf{f}, \mathbf{u}\}$$

The semantics of connectives and quantifiers are defined as follows.

$\phi \wedge \psi$

$\phi \backslash \psi$	t	f	u
t	t	f	u
f	f	f	f
u	u	f	u

$\neg\phi$	ϕ
f	t
t	f
u	u

and note that $\phi \vee \psi \equiv \neg(\neg\phi \wedge \neg\psi)$, $\phi \rightarrow \psi \equiv \neg\phi \vee \psi$

Let $\langle v, g \rangle$ be a model (V:value assignment and g:variable assignment), We also define semantics of the quantifiers $\exists$ and $\forall$ for non-empty set D as follows:

- $\tilde{V}_g^v(\forall x \in D \psi) =$
$$\begin{cases} \mathbf{t} & \text{forall } d \in D,\ \tilde{V}^v_{g(x/d)}(\psi) = \mathbf{t} \\ \mathbf{f} & \text{exists some } d \in D,\ \tilde{V}^v_{g(x/d)}(\psi) = \mathbf{f} \\ \mathbf{u} & \text{otherwise} \end{cases}$$

- $\tilde{V}_g^v(\exists x \in D \psi) =$
$$\begin{cases} \mathbf{t} & \text{exists some } d \in D,\ \tilde{V}^v_{g(x/d)}(\psi) = \mathbf{t} \\ \mathbf{f} & \text{forall } d \in D,\ \tilde{V}^v_{g(x/d)}(\psi) = \mathbf{f} \\ \mathbf{u} & \text{otherwise} \end{cases}$$

where $g(x/d)$ is a variable assignment which is same as g except d is assigned to x.

5 DOL formal semantics

We define semantics for a closed basic DOL formula ϕ with a **interpretation** I and a **variable assignment** g as follows.

5.1 Interpretation

The Interpretation I is a quadruple $\langle D, TS, v, d \rangle$.

Def 5.1 (Domain D)
A non empty set D is called **Domain.**

Def 5.2 (Time Structure TS)
A time structure TS is a tuple $\langle T, < \rangle$.

- T :a nonempty indexed set of time point $t_i (i = 0, 1, \ldots)$ called **set of worlds.**
- $<$:an total-order relation on T. $t_i < t_j$, if $i < j$.
- t_0: the start point $\in T$.
- s: a successor function on T. $s(t_i) = t_{i+1}$.

Def 5.3 (Domain mapping function d) A function $d : T \rightarrow 2^D$ called a **domain mapping function.** This satisfies a constraint as follows

- $d(t_0) = \emptyset$.

Def 5.4 (v) Semantic Function v is defined as follows.

- For a predicate symbol P,$v(P)$:
$(T \rightarrow (d(t)^n \rightarrow \{\mathbf{t}, \mathbf{f}\})) \cup$
$(T \rightarrow (D^n - d(t)^n) \rightarrow \{\mathbf{u}\})$
- For a n-ary function symbol f, $v(f) : D^n \rightarrow D$.
- For a constant symbol c, $v(c) : D$

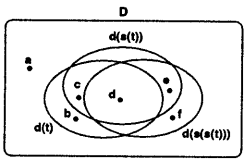

Figure 1: example of d(t)

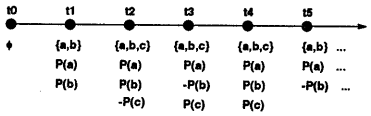

Figure 2: example of the interpretation

5.2 Variable Assignment

We define variable assignment g as follows:

Def 5.5 (variable assignment g)
A function g is a function from the set of variable symbols to D called **variable assignment.**

5.3 Semantics

Def 5.6 (The Semantics of DOL formula)

We define the semantics of a closed basic formula ϕ i.e. $V^I_{g,t_0}(\phi)$ with an interpretation I and a variable assignment g.

NOTE: This semantics are defined with using 3 valued first order logic (as defined above) with equality and number theory. The connectives of the meta language are distinguished by being dotted (ex: $\dot{\vee}$, $\dot{\wedge}$, and so on.) from the object logic's.

First, we define "term assignment" $V^I_{g,t}$ as follows,

Def 5.7 (Term assignment $V^I_{g,t}$)

- For predicate symbol $V^I_{g,t}(P) = v(P)t$
- For constant symbol $V^I_{g,t}(c) = v(c)$
- For function symbol $V^I_{g,t}(f) = v(f)$.
- For variable symbol $V^I_{g,t}(x) = g(x)$.
- $V^I_{g,t}(f(t_1, \ldots, t_n)) =$
  $V^I_{g,t}(f)(V^I_{g,t}(t_1), \ldots, V^I_{g,t}(t_n))$
- $V^I_{g,t}(\{t_1, \ldots, t_n\}) =$
  $\{V^I_{g,t}(t_1), \ldots, V^I_{g,t}(t_n)\}$

We impose one condition on v in I such that to any different closed terms t_1, t_2, $V^I_{g,t}(t_i)$ do not assign the same element in D like *Herbrand interpretation.*

We introduce the triple of $\langle\theta, \gamma, \rho\rangle$, which consists of truth value θ : **t**, **f**, **u**,created object function $\gamma : T \to 2^D$ and destruct-ed object function $\rho : T \to 2^D$, are used for the semantics definition. We define the following selecting function on the tripte. $T(\langle\theta, \gamma, \rho\rangle) = \theta$, $C(\langle\theta, \gamma, \rho\rangle) = \gamma$, $D(\langle\theta, \gamma, \rho\rangle) = \rho$.

We also some special functions $T \to 2^D$.

- $zero(t) = \emptyset$
- $co_{t_i,val}(t) =$
  $\begin{cases} val & \text{if } t = t_i \\ \emptyset & \text{otherwise} \end{cases}$

We define semantics $V^I_{g,t_0}(\phi)$ of formula ϕ as follows by using the function $\hat{V}^I_{g,t}$ defined later.

$V^I_{g,t_0}(\phi) =$
$\begin{cases} \mathbf{t} & \text{if } T(\hat{V}^I_{g,t_0}(\phi)) = \mathbf{t} \text{ and} \\ & \text{for all } i \in N[\\ & d(s(t_i)) = \\ & (d(t_i) \setminus (D(\hat{V}^I_{g,t_0}(\phi))(t_i))) \cup (C(\hat{V}^I_{g,t_0}(\phi))(t_i))] \\ \mathbf{f} & \text{if } T(\hat{V}^I_{g,t_0}(\phi)) = \mathbf{f} \text{ and} \\ & \text{for all } i \in N[\\ & d(s(t_i)) = \\ & (d(t_i) \setminus (D(\hat{V}^I_{g,t_0}(\phi))(t_i))) \cup (C(\hat{V}^I_{g,t_0}(\phi))(t_i))] \\ \mathbf{u} & \text{:otherwise} \end{cases}$

And $\hat{V}^I_{g,t}$ is defined recursively as follows.

- $\hat{V}^I_{g,t}(P(t_1, t_2, \ldots, t_n)) =$
  $\langle V^I_{g,t}(P)(V^I_{g,t}(t_1), V^I_{g,t}(t_2), \ldots, V^I_{g,t}(t_n)),$
  $zero, zero\rangle$
- $\hat{V}^I_{g,t}(\phi \wedge \psi) =$
  $\langle T(\hat{V}^I_{g,t}(\phi)) \dot{\wedge} T(\hat{V}^I_{g,t}(\psi)), \gamma, \rho\rangle$
  ,where $\forall i \in \mathbf{N}(\gamma(t_i) =$
  $C(\hat{V}^I_{g,t}(\phi))(t_i) \cup C(\hat{V}^I_{g,t}(\psi))(t_i))$
  and $\forall i \in \mathbf{N}(\rho(t_i) =$
  $D(\hat{V}^I_{g,t}(\phi))(t_i) \cup D(\hat{V}^I_{g,t}(\psi))(t_i))$
- $\hat{V}^I_{g,t}(\neg\phi) =$
  $\langle \dot{\neg} T(\hat{V}^I_{g,t}(\phi)), C(\hat{V}^I_{g,t}(\phi)), D(\hat{V}^I_{g,t}(\phi))\rangle$
- $\hat{V}^I_{g,t}(\forall x\phi) =$
  - $\langle \dot{\forall}\delta \in d(t) T(\hat{V}^I_{g(x/\delta),t}(\phi)), \gamma, \rho\rangle$
    ,where
    $\forall i \in \mathbf{N}(\gamma(t_i) = \bigcup_{\forall\delta\in d(t)}(C(\hat{V}^I_{g(x/\delta),t})(t_i)))$
    and $\forall i \in \mathbf{N}(\rho(t_i) =$
    $\bigcup_{\forall\delta\in d(t)}(D(\hat{V}^I_{g(x/\delta),t})(t_i)))$
    if $d(t) \neq \emptyset$.
  - $\langle \mathbf{u}, zero, zero\rangle$ if $d(t) = \emptyset$.

  ,where $g(x/\delta)$ is a variable assignment which is same as g except x is assigned by δ.
- $\hat{V}^I_{g,t}(\textbf{ new } x\phi) = V^I_{g',t}(\phi)$
  ,where $g' = g(x/e^t_x)$ and $e^t_x \in D$
  and $\forall t_j \in T, (t_0 \leq t_j < t \to e^t_x \notin d(t_j))$.
  e^t_x has following conditions.
  - $e^{t_i}_x \neq e^{t_j}_y$ iff $x \hat{\neq} y$ or $i \neq j$
  - $e^{t_i}_x \neq e^{t_j}_y$ means $e^{t_i}_x, e^{t_j}_y$ are not identical objects.
  - $x \hat{\neq} y$ means "x","y" are not the same characters.
- $\hat{V}^I_{g,t}(\dot{\triangleright}(\alpha)) =$
  - $\langle \mathbf{t}, zero, zero\rangle$ if $\alpha = \emptyset$
  - $\langle \mathbf{t}, co_{t,V^I_{g,t}(a)}, zero\rangle$ if $\alpha = \{a\}$ and $V^I_{g,t}(a) \notin d(t)$ and $V^I_{g,t}(a) \in d(s(t))$.
  - $\langle (V^I_{g,t}(a) \notin d(t)) \dot{\to} (V^I_{g,t}(a) \in d(s(t))),$ $zero, zero\rangle$ if $\alpha = \{a\}$ and $V^I_{g,t}(a) \in d(t)$ or $V^I_{g,t}(a) \notin d(s(t))$.
  - $\hat{V}^I_{g,t}(\forall x \in \alpha \dot{\triangleright}(x))$,otherwise.
- $\hat{V}^I_{g,t}(\ddot{\triangleright}(\alpha)) =$
  - $\langle \mathbf{t}, zero, zero\rangle$ if $\alpha = \emptyset$
  - $\langle \mathbf{t}, co_{t,V^I_{g,t}(a)}, zero\rangle$ if $\alpha = \{a\}$ and $V^I_{g,t}(a) \notin d(t)$ and $V^I_{g,t}(a) \in d(s(t))$.
  - $\langle (V^I_{g,t}(a) \notin d(t)) \dot{\wedge} (V^I_{g,t}(a) \in d(s(t))),$ $zero, zero\rangle$ if $\alpha = \{a\}$ and $V^I_{g,t}(a) \in d(t)$ or $V^I_{g,t}(a) \notin d(s(t))$.
  - $\hat{V}^I_{g,t}(\forall x \in \alpha \ddot{\triangleright}(x))$,otherwise.
- $\hat{V}^I_{g,t}(\dot{\triangleleft}(\alpha)) =$
  - $\langle \mathbf{t}, zero, zero\rangle$ if $\alpha = \emptyset$
  - $\langle \mathbf{t}, zero, co_{t,V^I_{g,t}(a)}\rangle$ if $\alpha = \{a\}$ and $V^I_{g,t}(a) \in d(t)$ and $V^I_{g,t}(a) \notin d(s(t))$.
  - $\langle (V^I_{g,t}(a) \in d(t)) \dot{\to} (V^I_{g,t}(a) \notin d(s(t))),$ $zero, zero\rangle$ if $\alpha = \{a\}$ and $V^I_{g,t}(a) \notin d(t)$ or $V^I_{g,t}(a) \in d(s(t))$.
  - $\hat{V}^I_{g,t}(\forall x \in \alpha \dot{\triangleleft}(x))$,otherwise.
- $\hat{V}^I_{g,t}(\ddot{\triangleleft}(\alpha)) =$
  - $\langle \mathbf{t}, zero, zero\rangle$ if $\alpha = \emptyset$
  - $\langle \mathbf{t}, zero, co_{t,V^I_{g,t}(a)}\rangle$ if $\alpha = \{a\}$ and $V^I_{g,t}(a) \in d(t)$ and $V^I_{g,t}(a) \notin d(s(t))$.

– $\langle (V^I_{g,t}(a) \in d(t)) \dot{\wedge} (V^I_{g,t}(a) \notin d(s(t))),$
$zero, zero \rangle$ if $\alpha = \{a\}$ and $V^I_{g,t}(a) \notin d(t)$ or $V^I_{g,t}(a) \in d(s(t))$.

– $\hat{V}^I_{g,t}(\forall x \in \alpha \ddot{\triangleleft}(x))$,otherwise.

- $\hat{V}^I_{g,t}(\bigcirc\phi) = \hat{V}^I_{g,s(t)}(\phi)$
- $\hat{V}^I_{g,t}([\phi]\psi) =$
$\langle (\forall t_j (t \leq t_j \dot{\rightarrow} T(\hat{V}^I_{g,t_j}(\psi)))$
$\dot{\vee}(\exists t_k (t \leq t_k \dot{\wedge} T(\hat{V}^I_{g,t_k}(\phi)) \dot{\wedge}$
$\forall t_j (t \leq t_j < t_k \dot{\rightarrow} T(\hat{V}^I_{g,t_j}(\psi))))),$
$\gamma, \delta \rangle$
,where $\forall i \in \mathbf{N} (\gamma(t_i) = \bigcup_{\forall t_j \geq t}$
$(C(\hat{V}^I_{g,t_j}(\psi))(t_i) \cup C(\hat{V}^I_{g,t_j}(\phi))(t_i)))$
and $\forall i \in \mathbf{N} (\delta(t_i) = \bigcup_{\forall t_j \geq t}$
$(D(\hat{V}^I_{g,t_j}(\psi))(t_i) \cup D(\hat{V}^I_{g,t_j}(\phi))(t_i)))$.
- $\hat{V}^I_{g,t}(\mathbf{True}) = \langle \mathbf{t}, zero, zero \rangle$

5.4 Satisfiablity and validity

We define following term. Note that for a closed formula, g has no effect on its semantics .

t_0-satisfiable "a closed formula ϕ is said to be *satisfiable.*" iff some interpretation I exists such as $V^I_{g,t_0}(\phi) = \mathbf{t}$.

t_0-valid "a closed formula ϕ is said to be *valid.*" iff for any interpretation I, $V^I_{g,t_0}(\phi) = \mathbf{t}$.

t_0-unsatisfiable "a closed formula ϕ is said to be *unsatisfiable.*" iff for any interpretation I, $V^I_{g,t_0}(\phi) = \mathbf{f}$.

5.5 Default semantics of object existence

For example, a DOL formula
$\dot{\triangleright}(\{a\}) \wedge \bigcirc\bigcirc\neg\dot{\triangleright}(\{a\})$
is t_0-unsatisfiable by definition. Althogh
$\dot{\triangleright}(\{a\}) \wedge \bigcirc\bigcirc\neg\dot{\triangleright}(\{a\}) \wedge \bigcirc\dot{\triangleleft}(\{a\})$
is t_0-satisfiable. (i.e. for some formulas A, B. $A \wedge B$ is unsatisfiable, but $A \wedge B \wedge C$ is satisfiable.) This means that DOL is a non-monotonic logic.

Thus to define the semantics function V, we used $\hat{V}$ function which returns a triple:
⟨**Truth-Value**,
Function returns the set of objects created at t,
Function returns the set of objects destruct-ed at t⟩

Using this function, we can check whether the domains $d(t)$ in interpretation are just what intended to be expressed or not. If a domain $d(t)$ include objects what have never been created before t, the interpretation is unsuitable for the formula. If a domain $d(t)$ does not include object what is once created but have never been destruct-ed, the interpretation is unsuitable for the formula, In these case $V^I_{g,t_0}(\phi)$ returns **u**.

6 Applications of the logic

We present some specification example in DOL.

6.1 X-window like window system

X-Window system is a practical reactive system application of the object oriented design and implementation. We try to write down such a system to prove expressiblity of our logic.

For the convention, we use DOL + number theory with equality.

1. $\ddot{\triangleright}(\{b1, b2, b3\}) \wedge \bigcirc\Box(Button(b1) \wedge Button(b2) \wedge Button(b3))$

2. $\ddot{\triangleright}(\{c\}) \wedge \bigcirc\Box(Cursor(c) \wedge Position(c, 0, 0))$
(First, two button on the mouse, and one cursor on the screen.)

3. $\bigcirc\Box((Stop(c) \wedge \neg GoRight(c) \wedge \neg GoLeft(c) \wedge \neg GoUp(c) \wedge \neg GoDown(c))$
$\vee(\neg Stop(c) \wedge GoRight(c) \wedge \neg GoLeft(c) \wedge \neg GoUp(c) \wedge \neg GoDown(c)) \vee(\neg Stop(c) \wedge \neg GoRight(c) \wedge GoLeft(c) \wedge \neg GoUp(c) \wedge \neg GoDown(c)) \vee(\neg Stop(c) \wedge \neg GoRight(c) \wedge \neg GoLeft(c) \wedge GoUp(c) \wedge \neg GoDown(c)) \vee(\neg Stop(c) \wedge \neg GoRight(c) \wedge \neg GoLeft(c) \wedge \neg GoUp(c) \wedge GoDown(c)))$
(Second, the cursor is either stopping , going right, going left, going up or going down.)

4. $\bigcirc\Box\forall x, y((GoRight(c) \wedge Position(c, x, y)) \rightarrow \bigcirc Position(c, x+1, y))$

5. $\bigcirc\Box\forall x, y((GoLeft(c) \wedge Position(c, x, y)) \rightarrow \bigcirc Position(c, x-1, y))$

6. $\bigcirc\Box\forall x, y((GoUp(c) \wedge Position(c, x, y)) \rightarrow \bigcirc Position(c, x, y+1))$

7. $\bigcirc\Box\forall x, y((GoDown(c) \wedge Position(c, x, y)) \rightarrow \bigcirc Position(c, x, y-1))$

8. $\bigcirc\Box\forall x, y((Stop(c) \wedge Position(c, x, y)) \rightarrow \bigcirc Position(c, x, y))$
(Third, when the cursor is forced to move, its position is also changed, of course.)

9. $\bigcirc\Box\forall x, y, x', y', c((Position(c, x, y) \wedge Position(c, x', y')) \leftrightarrow ((x = x') \wedge (y = y')))$
(Fourth, condition of predicate "Position")

10. $\bigcirc\Box\forall x, y((Pushed(b1) \wedge Position(c, x, y)) \leftrightarrow (\textbf{new}\ z(\ddot{\triangleright}(\{z\}) \wedge ([\ddot{\triangleleft}\{z\}]Window(z)) \wedge Position(z, x, y))))$
(Fifth, if button $b1$ is pushed, generate a new window immediately.)

11. $\bigcirc\Box\forall z, x, y((Pushed(b2) \wedge Position(c, x, y) \wedge Window(z) \wedge Position(z, x, y)) \leftrightarrow \ddot{\triangleleft}(\{z\}))$
(Last, if button $b2$ is pushed on some window, destroy it immediately.)

12. $\bigcirc\Box\forall z(Pushed(b3) \leftrightarrow (\forall z(Window(z) \wedge ReWrite(z))))$ (And, if button $b3$ is pushed , all windows currently on the screen are rewrited.)

7 Concluding Remarks

We introduced new first order logic DOL (Dynamic Object Logic), and described the features of this logic and showed how the formalization of object creation and destruction be used by using some examples.

New modal operators $\triangleright, \triangleleft$ and quantifier **new** are introduced. They express the notions of "object creation" , "object destruction" and "new object", respectively as are used in "object orientation". We also defined its 3 valued semantics which is useful to capture natures of system objects. Thus, using this logic, we can express the behaviors of system objects and interaction between them naturally.

DOL is a predicate modal logic with variable domain. Although there is an argument about modal logic with variable domain in the paper [7], its semantics is 2 valued as classical logic and excluded the case of empty domain.

We are going to make the semi-t_0-satisfiability checking procedure of this and apply the technique of the "semantic compile" [10]) for MSL [9] to DOL similary. If there is a specification written in DOL, we will be able to check if it is t_0-satisfiable or not, and get suitable programs according to a t_0-satisifiable specification written in DOL. We also intended to axiomize t_0-valid formulas in DOL.

References

[1] Brian F.Chellas, "Modal Logic - an introduction", Cambridge University Press,1980.

[2] Robert Goldblatt, "Logics of Time and Computation", CSLI Lecture Notes Number 7, 1987.

[3] A. Pnueli, "Application of Temporal Logic to the Specification and Verification of Reactive Systems: A Survey of Current Trends", *Lecture Note in Computer Science 224*, pp. 510-584, 1985.

[4] Z. Manna, P. Wolper, "Synthesis of Communicating Processes from Temporal Logic Specifications", *Lecture Note in Computer Science 131*, pp. 253-281, 1981.

[5] M. Abadi, L. Lamport, P. Wolper "Realizable and Un-realizable Specifications of Reactive Systems", *Lecture Note in Computer Science 372*, pp. 1-17, 1989.

[6] A. Pnueli, R. Rosner, "On the Synthesis of Reactive Module", *ACM Symposium on Principle of Programming Language*, pp. 179-190, 1989.

[7] Dov M. Gabbay, "Investigations In Modal And Tense Logics With Applications To Problems In Philosophy And Linguistics" *D.Reidel Publishing Company*,1976.

[8] "Logics For Artificial Intelligence" (Raymond Turner,1984).

[9] N. Yonezaki, "Conceptual Modeling in MSL", *Information Modeling and Knowledge Bases*, 1990.

[10] "Study on MSL compiler" (in Japanese), Shin MIYAKAWA, *Tokyo Institute of Technology, Master's Thesis 1992*

[11] "Towards an Object Calculus", Oscar Nierstrasz, *University of Geneva, "Object-Based Concurrent Computing" LNCS 612 pp.1 - pp.20 Springer-Verlag 1991*