Representation and Analysis of Mobile Computing using TeleLog– A Mobile Logic Language

SATO, Hiroyuki

Information Technology Center,
The University of Tokyo,
Tokyo, 113-8658, Japan.

ABSTRACT

In the era of the Internet, mobile agents provide novel and promising approach to analysis of computation in the communication network. In this paper, we define TeleLog, and analyze mobile computing using modal logic. $\boldsymbol{go}$ and $\boldsymbol{here}$ primitives are augmented with ordinary logic languages. They are given operational semantics in terms of mobile computing. Moreover, modality of TeleLog is analyzed, and applied to the expressive power analysis of TeleLog. We represent a security mechanism using hook and assert functions that can be represented for the first time by using modality.

Keywords: mobile agent, logic programming, modality, security

1 Introduction

In the era of the Internet, mobile agents provide novel and promising approach to analysis of computation in the communication network. A mobile agent is a computer program that can move around different hosts across the network while carrying code and data. Mobility is observed commonly in internet computing such as Java applets, Web crawling, e-commerce, and some vicious computer virus. Therefore, it is significant to analyze key aspects of mobile computing such as security on a firm semantic basis.

TeleLog[16] is a logic programming approach to mobile agents. Like other logic programming languages, TeleLog is an executable fragment of some kind of logic. There is given a natural logic-based semantic framework for logic languages. In this meaning, behaviors of logic languages are relatively easy to analyze. Our goal is to analyze mobile computing using TeleLog.

This paper presents an approach to the analysis of mobile computing using modal logic. Modal logic is one of major methods in extending logic. Specifically, we give operational semantics to mobile computing in Kripke-like frame. A place corresponds to a world in Kripke's sense. Mobility means that a program can explicitly move around Kripke worlds. Actually, modality can represent a variety of concepts including time, provability, and belief. This paper tries adding mobility to this menu.

We also show that modality is essential to represent some useful concepts of mobile computation. In particular, security checking mechanism is given a representation in TeleLog. Because security is one of key aspects in mobile computing, we can conclude that this fact implies the usefulness of TeleLog.

The rest of this paper is organized as: Section 2 defines TeleLog, and gives its basic semantics without modality. Section 3 is devoted to the analysis of modality of TeleLog. This analysis is applied to the expressive power analysis of TeleLog in Section 4. Section 5 surveys related work. Section 6 gives a summary of this paper.

2 TeleLog

2.1 Syntax

TeleLog[16] is a logic programming language for mobile calculi. Computation is processed on places. A program can interact with its embracing "places" using $\boldsymbol{go}$ and $\boldsymbol{here}$ primitives.

2.2 Syntax

Definition 1 The syntax of TeleLog is defined in Figure 1. A program is a located Horn clause.

In addition to ordinary Prolog syntax, we consider addresses. $\boldsymbol{go}$ and $\boldsymbol{here}$ are defined as primitives related to addresses.

2.3 Semantics of TeleLog without Modality

First, we give operational semantics to TeleLog without modality. It is given based on resolution on formulas and unification on terms in the same way as ordinary Prolog except $\boldsymbol{go}$ and $\boldsymbol{here}$. Here, we concentrate on axioms of $\boldsymbol{go}$ and $\boldsymbol{here}$.

P	$::=$	$p(e_1, \cdots, e_n)$	Predicate
M	$::=$	$\Box P \mid \Box M$	Modality
U	$::=$	$P \mid M$	Unit Clause
L	$::=$		Literals
		U	unit clause
		$\mid \boldsymbol{go}\ \ addr$	go
		$\mid \boldsymbol{here}\ \ addr$	here
$\mathcal{C}$	$::=$		Horn clauses
		U	unit clause
		$\mid P : -L_1, \cdots, L_n$	program clause
		$\mid ? - L_1, \cdots, L_n$	goal clause
$\mathcal{L}$	$::=$		located Horn clauses
		$\boldsymbol{addr}[\![\mathcal{C}]\!]$	

Figure 1: Syntax of TeleLog

Definition 2 The axiom of $\boldsymbol{go}$ is given as:

$$\frac{\boldsymbol{l}[(? - \boldsymbol{go}\ \ \boldsymbol{m}, G_2, \cdots, G_n)\theta]}{\boldsymbol{m}[(? - G_2, \cdots, G_n)\theta]}$$

Here θ is a substitution that represents correspondence between variables and terms. Note that in the above definition, a substitution can move from $\boldsymbol{l}$ to $\boldsymbol{m}$. This corresponds to collecting terms from multiple addresses.

Definition 3 The axiom of $\boldsymbol{here}$ is given as:

$$\frac{}{\boldsymbol{l}[\boldsymbol{here}\ \ \boldsymbol{l}]}$$

Because the semantic domain is extended to include addresses, we need extension to resolution and unification.

Because TeleLog is extended so that a program may move among addresses, in a unifier, we may have terms defined on different addresses. Therefore, we extend the unification and the most general unifier (mgu) in order to include remote terms. We also denote this extended unifier by mgu. The mechanism of collecting terms from multiple addresses is a major function of mobile computing.

Definition 4

$$\frac{\boldsymbol{l}[? - (G_1, \ldots, G_n)\phi] \quad \boldsymbol{l}[H : -B_1, \cdots, B_m]}{\boldsymbol{l}[? - (B_1, \cdots, B_m, G_2, \cdots, G_n)\phi\theta]}$$

where $(\theta = mgu(G_1\phi, H)_{\boldsymbol{l}})$

In $\theta = mgu(G_1\phi, H)_{\boldsymbol{l}}$, θ is the most general unifier of $G_1\phi$ and H at location $\boldsymbol{l}$.

In this definition, substitutions θ and ϕ represent correspondence between variables and terms. mgu is calculated on a specific address $\boldsymbol{l}$, but substitutions can collect terms from multiple addresses as discussed in the definition of $\boldsymbol{go}$.

Actually, the mechanism of $\boldsymbol{go}$ and $\boldsymbol{here}$ is so general that we can encode a remote subroutine call in which we visit a remote place, do something, and return to the original place.

Example 1 Consider the program fragment below:

$$\boldsymbol{here}\ \ Addr, \boldsymbol{go}\ \ \boldsymbol{m}, G, \boldsymbol{go}\ \ Addr, \cdots \qquad (*)$$

This sequence is an implementation of the remote resolution call provided that the call always succeeds. On any address, this program first moves to m, does some work (G), and finally returns to the original address as shown in Figure 2.

3 Modality in TeleLog

In this section, we discuss modality in TeleLog.

The resolution scheme for modal logic is already given in [8], and the one for hybrid logic is given in [1]. Even before these schemes, a number of logic programming languages with modality are proposed [11]. This paper adds another resolution scheme for TeleLog, an executable logic with restricted modality. Our scheme can express some important concepts of mobility such as security.

3.1 Operational Semantics

Considering the semantics of TeleLog, the modality must be taken into consideration.

The engine of fully-fledged TeleLog is based on that of TeleLog without modality discussed in Section2. To discuss modality, we need an extra environment.

Definition 5 The *modality environment* of TeleLog consists of a pair Θ, Ξ of set of clauses. The form

$$\Theta, \Xi \triangleright \boldsymbol{l}[C]$$

represents a resolution of TeleLog which satisfies relations in Figure 3.

Intuitively, in the form $\Theta, \Xi \triangleright l[C]$, Θ represents formulas that wait to be proved at every address where $\boldsymbol{go}$ goes,

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\boldsymbol{l}[?-\boldsymbol{here}\ \ Addr,\boldsymbol{go}\ \ \boldsymbol{m},\boldsymbol{go}\ \ Addr,\cdots]\theta \quad \boldsymbol{l}[\boldsymbol{here}\ \ \boldsymbol{l}]}{\boldsymbol{l}[?-\boldsymbol{go}\ \ \boldsymbol{m},G,\boldsymbol{go}\ \ Addr,\cdots]\theta[Addr/\boldsymbol{l}]}}{\boldsymbol{m}[?-G,\boldsymbol{go}\ \ Addr,\cdots]\theta[Addr/\boldsymbol{l}]} \qquad \boldsymbol{m}[G:-\cdots]}{\vdots}}{\boldsymbol{m}[?-\boldsymbol{go}\ \ Addr,\cdots]\widetilde{\theta}[Addr/\boldsymbol{l}]}}{\boldsymbol{l}[?-\cdots]\widetilde{\theta}[Addr/\boldsymbol{l}]}$$

Figure 2: Resolution associated with remote resolution call: the effect of G is reflected in the change from θ to $\widetilde{\theta}$.

1.

$$\frac{A \in \Xi}{\Theta, \Xi \triangleright \boldsymbol{l}[A]}$$

2.

$$\frac{\Theta, \Xi \triangleright \boldsymbol{l}[?-G_1,\cdots,G_n] \quad \Theta, \Xi \triangleright \boldsymbol{l}[H:-C_1,\cdots,C_m]}{\Theta, \Xi \triangleright \boldsymbol{l}[?-(C_1,\cdots,C_m,G_2,\cdots,G_n)\theta]}\ \theta = mgu(G_1, H)$$

3. (modality)

$$\frac{\Theta, \Xi \triangleright \boldsymbol{l}[?-\Box A, G_1,\cdots,G_n]}{\Theta \cup \{A\}, \Xi \triangleright \boldsymbol{l}[?-G_1,\cdots,G_n]}$$

4. (go)

$$\frac{\Theta, \Xi \triangleright \boldsymbol{l}[?-\boldsymbol{go}\ \ \boldsymbol{m}, G_1,\cdots,G_n] \quad \Theta, \Xi \triangleright \boldsymbol{l}[\Box A_i]\theta_i|_{i=0}^{m} \qquad \begin{array}{c} \surd \\ \emptyset, \Xi \triangleright \boldsymbol{m}[?-\wedge\Theta] \\ \vdots \\ \emptyset, \Xi \triangleright \boldsymbol{m}[]\theta \end{array}}{\Theta, \Xi \cup \{A_0,\cdots A_m\} \triangleright \boldsymbol{m}[?-G_1,\cdots,G_n]\theta_0\cdots\theta_m\theta}\ \surd, \boldsymbol{l} \rightsquigarrow \boldsymbol{m}$$

Figure 3: Resolution Rules of TeleLog

while Ξ represents extra axioms which are forced to be valid from outside address. Particularly for modality, if a clause succeeds in a path, the process is a proof including that on every place on the path that a clause visit, every $\Box$-ed clause is true, hence $\Box$-ed clause is true in the path. If we consider the path as a Kripke frame, we can give semantics to this program relative to this frame.

According to Kripke semantics of modal logic, if $\Box A$ appears in a resolution, A must be checked whenever a program moves its address. If $\Box A$ appears as a unit clause in an address, it must be valid wherever a program moves from the address to another address. We do not allow a modal formula to appear as LHS of a program clause. Therefore, the rule 2. of Definition 5 does not apply to modal formulas.

Actually, if a program clause such as $\boldsymbol{l}[\Box A : -\Box B]$ is allowed, we must check $\Box B$ to resolve $\Box A$ at $\boldsymbol{l}$. Then at every address from $\boldsymbol{l}$, we must check B to prove A at every address from $\boldsymbol{l}$, which proliferates resolution processes, and add undesirable complexity. Instead, we put a restriction on modality, and obtain its clear semantics.

4 Expressive Power of Modality of TeleLog

The modality in TeleLog can express some significant concepts of mobile computing such as hook and assert. Review the rules of modality and $\boldsymbol{go}$ in Definition 5.

$$\frac{\Theta, \Xi \triangleright \boldsymbol{l}[? - \Box A, G_1, \cdots, G_n]}{\Theta \cup \{A\}, \Xi \triangleright \boldsymbol{l}[? - G_1, \cdots, G_n]}$$

If we encounter a modal formula $\Box A$ during a resolution, A must be resolved in every address thereafter visited. Therefore, A must be saved in Θ-part, and when a $\boldsymbol{go}$ is executed, it is resolved. This process is represented in the resolution of Θ-part in the rule of $\boldsymbol{go}$ (4. in Figure 3) which requires both the success of resolution of $\emptyset, \Xi \triangleright \boldsymbol{m}[? - \wedge\Theta]$, and adds extra axioms given as $A_0, \cdots A_m$ from the outer world.

Therefore, we can say that if the resolution succeeds, the check (resolution of Θ-part) is done, and we can move $\boldsymbol{l}$ to $\boldsymbol{m}$ with additional axioms $A_0, \cdots A_m$. This implements a hook mechanism, which can be used to describe the security check mechanism in mobile calculus.

One more point to be noted in the rule of $\boldsymbol{go}$ is that if $\Box A$ is asserted as a unit clause, then at every address thereafter visited, A is treated as a unit clause in Ξ-part. This can be used as side-effect in a resolution.

4.1 Hook and Security Check

Processing $\Box$-ed clauses is a kind of prologues of function calls. Actually, if we register some actions as a set of $\Box$-ed clauses, we can use them as hooking mechanism at entering an address.

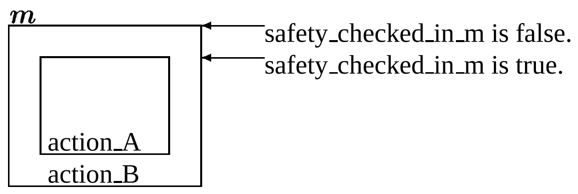

Figure 4: Security Layer given by Example 2

Hooking at entering an address is important in the sense that we can insert hidden actions there. As an important application of this kind of hooking, we show security checking mechanism below. Note that security checking by verifying certificates must be performed:

1. at the very moment an agent enters an address, the address spawns the verifier process, or

2. if the address is required to verify the certificate that an agent has, the verifier process is spawned.

TeleLog cannot represent the first option because effects are brought only by an agent. As for the second option, if we have `assert` mechanism as in Prolog, we can represent the security check mechanism below.

Example 2 In an address $\boldsymbol{m}$, let the program below be given:

```
verify_certificate(X) :-
        cert_check(X),
        assert(safety_checked_in_m).
verify_certificate(X).

cert_req_actionA(X) :-
        safety_checked_in_m,
        actionA(X).
cert_not_req_actionB(X) :- actionB(X).
```

Assume that `cert_check(X)` is given to check the certificate passed as `X`.

Let a goal clause be written as below.

```
?- []verify_certificate(my_cert),
   goal_1, goal_2, go m, ...
```

When the program makes a $\boldsymbol{go}$ to $\boldsymbol{m}$, according to the resolution rule of $\boldsymbol{go}$, the clause `verify_certificate(my_cert)` is checked first. If the check succeeds, `safety_checked_in_m` is asserted. After this assertion, safety-requiring actions can be performed. If the check (`cert_check`) fails, safety-requiring actions cannot be performed, but other actions are still possible (Figure 4).

If
a program is written without `verify_certificate`, the security is not checked. Only not-safety-requiring actions are possible. In this way, we can describe a security layer.

4.2 Assert

Assertions have side effects in the sense that it changes the semantics of a program. $\Box A$ acts as an assertion of A in the addresses a program visits. Although this modality is restricted to unit modality, this extension can partially express functions of `assert` primitive.

Particularly, we show that a restricted form of `assert` can be represented in TeleLog. This enables us to describe security mechanism in pure Telelog.

Example 3 Let an address $\boldsymbol{LocM}$ be defined as follows:

```
LocM[verify_certificate(X) :-
      cert_check_strict(X).]
LocM[[]safety_checked_in_m.]
```

Let an address $\boldsymbol{M}$ be defined as follows:

```
M[verify_certificate(X) :-
      safety_checked_in_m.]
M[verify_certificate(X) :-
      cert_check(X), LocM::true.]

M[cert_req_actionA(X) :-
      safety_checked_in_m, actionA(X).]
M[cert_not_req_actionB(X) :- actionB(X).]
```

If a program enters address $\boldsymbol{M}$, its certificate is checked, and goes to $\boldsymbol{LocM}$. In $\boldsymbol{LocM}$, $\Box$`safety_checked_in_m` is asserted. This means that at the point of the return from $\boldsymbol{LocM}$ (the success of `LocM::true`), `safety_checked_in_m` is valid. At the return, because `verify_certificate` succeeds by the first clause, there is no infinite travel between $\boldsymbol{M}$ and $\boldsymbol{LocM}$.

In this meaning, `assert(A)`, a Prolog extension can be represented as `LocM::A`.

This result indicates that security check can be expressed without *assert* primitive.

Note that TeleLog itself is open to the security policy in the real world. For example, the key predicate `safety_checked_in_m` must be confidential to other addresses. Moreover, the connectivity of $\boldsymbol{LocM}$ to $\boldsymbol{M}$, and its dis-connectivity to addresses other than $\boldsymbol{M}$ must be separately expressed(Figure 5). However, despite these open problems, we can say that TeleLog is a candidate for expressing security mechanism.

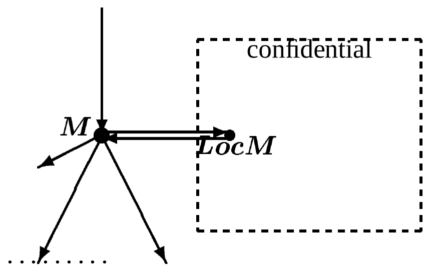

Figure 5: Connectivity of M and LocM

5 Related Work

Extensions of logic programming languages and unification to include code mobility have been tried mainly in AI. In particular, BinProlog/Jinni[18, 17] extends Prolog to facilitate communication between agents via global tuple space. Unification in tuple space is critical in its remote resolution.

Addition of modality to logic programming languages has a long history [11]. In its earlier stages, temporal logic was a major concern [3, 4, 15]. A resolution system for modal logic is given in [8]. For hybrid logic, [1] gives a resolution system. Our resolution gives another system. The difference to previous researches lies in that we give a special treatment to modality that enables the representation of hooking, and security mechanism as its application. Besides, logic programming, Modal-ML [20] implements [12]. In Modal-ML, modality is used as representing a stage of compilation.

There have been proposed a number of programming languages such as Telescript[19], Odyssey, Voyger, and Aglets[9] to represent mobility. Among those, Telescript is the first industrial strength mobile agents programming language designed by White. TeleLog owes much to Telescript in the idea of representing mobility.

Another line that has direct connection with this paper is mobile calculus. It has emerged as a variant of process algebra [2], but soon has become one major line [5, 10, 14]. Among those, Dπ[13] has introduced resources together with locations and $\boldsymbol{go}$ as ours. Our system is type-free while Dπ is concerned with types for locations, channels, and transmissions. [6] uses locations as layering services in applications, host, and net.

The expanding Internet world requires appropriate calculus for mobile calculus that expresses Java applets or Web crawling [7]. Mobile calculus is also closely related to mobile shopping, an important branch of e-commerce, which Telescript has as background.

6 Concluding Remarks

In Section 2, we have defined TeleLog, and gives its basic semantics without modality. go and $here$ primitives are introduced to ordinary logic languages. They are given operational semantics in terms of mobile computing. In Section 3, we have focused on modality of TeleLog. Its operational semantics is given. This analysis has been applied to the expressive power analysis of TeleLog in Section 4. We have represented a security mechanism using hook and assert functions that can be represented first by using modality.

References

[1] Areces, C., de Nivelle, H., de Rijk, M., "Resolution in Modal, Description and Hybrid Logic," J. Logic and Computation, 11(5), 2001, 717–736.

[2] Baeten, J, Weijland, W., Process Algebra, 1990.

[3] Baudinet, M., Temporal logic programming is complete and expressive, Proc. 1989 Principle of Programming Languages 1989, 267–280.

[4] Baudinet, M., On the expressiveness of temporal logic programming, Information and Conmputation, 117(2), 1995, 157–180.

[5] Cardelli, L, Gordon, A., "Types for Mobile Ambients," Proc. 1999 Principles of Programming Languages, 1999, 79–92.

[6] Chothia, T., Stark, I., "A Distributed π-Calculus with Local Areas of Communication," Proc. High-Level Concurrent Languages '00, ENTCS 41.2, 2000.

[7] Fielder, J., Hammer, J., "Using the Web Efficiently: Mobile Crawlers," Proc. 7th AoM/IAoM Int'l Conf. Computer Science, 1999, 324–329.

[8] Fitting, M., "Destructive Modal Resolution," J. Logic and Computation, 1(1), 1990, 83–97.

[9] Lange, B.D., Oshima, M., Programming and Deploying Java Mobile Agents with Aglets, 1998.

[10] Milner, R., Communicating and Mobile Systems: the Pi-Calculus, 1999.

[11] Orgun, M.A., Ma, W., An Overview of Temporal and Modal Logic Programming, Proc. 1st Int'l Conf. Temporal Logic, 1994.

[12] Davies, R, Pfenning, F., "A modal analysis of staged computation," Proc. 1996 Principles of Programming Lsanguages, 1996, 258–270.

[13] Hennesy, M., Riely, J., "Resource Access Control in Systems of Mobile Agents," Proc. High-Level Concurrent Languages '98, ENTCS 16.3, 1998.

[14] Riely, J., Hennessy, M., "A Typed Language for Distributed Mobile Processes," Proc. 1998 Principles of Programming Languages,1998, 378-390.

[15] Sistla, P, A., Zuck, L.D., Reasoning in a restricted temporal logic, Information and Computation 102(2), 1993, 167–195.

[16] Taguchi, K., Sato, H., "TeleLog, A Mobile Logic Programming Language," Proc. 2001 Joint Symp. Parallel Processing, 2001, 213–220.

[17] Tarau, P., "Jinni: Intelligent Mobile Agent Programming at the Intersection of Java and Prolog" *http://www.binnetcorp.com/Jinni*

[18] Tarau, P., Dahl, V., "Mobile Threads through First Order Continuations," Proc. APPAL-GULP-PRODE'98, 1998.

[19] White, J.E., "Telescript Technology: Mobile Agents," Software Agents, 1997.

[20] Wickline, P., Lee, P., Pfenning, F., "Run-time Code Generation and Modal-ML," Proc. 1998 Programming Languages Design and Implementation, 1998, 224–235.