Abstraction of Graph Transformation Systems by Temporal Logic and Its Verification

Mitsuharu Yamamoto[1], Yoshinori Tanabe[2,3], Koichi Takahashi[2], and Masami Hagiya[3,4]

[1] Faculty of Science, Chiba University
[2] Research Center for Verification and Semantics, National Institute of Advanced Industrial Science and Technology (AIST)
[3] Graduate School of Information Science and Technology, University of Tokyo
[4] NTT Communication Science Laboratories

1 Introduction

Abstract model checking has been studied as a promising technique for applying various model checking methods to infinite state systems. Graph transformation systems [11], which can model many distributed and concurrent algorithms, are examples of such infinite systems.

We have been studying abstraction of several kinds of link structures, which are instances of graph transformation systems. First, we introduced abstraction of heap structures using regular expressions mainly for verifying concurrent garbage collection algorithms [14, 15]. In this setting, each cell has a color and a link to another cell. Since cells can be allocated dynamically during execution, it is impossible to enumerate all the execution states of the heap. Thus we need to use abstraction for applying finite verification methods.

The basic strategy is to abstract each cell in a heap structure in terms of regular expressions taken from a fixed finite set. Each regular expression represents a property of a cell concerning its connectivity with other cells. For example, the regular expression $\mathtt{w}^*\mathtt{g}$ holds at a cell that can reach a gray cell via white cells, where `w` is the label of a white cell and `g` is that of a gray cell.

Similarly, one can use branching-time temporal logic formulas in place of regular expressions [16], where temporal operators are used for describing spatial relationship. For example, the temporal formula $\mathbf{E}(\mathtt{w}\ \mathbf{until}\ \mathtt{g})$ in CTL (computation tree logic) also represents a cell that can reach a gray cell via white cells. In [16], we defined abstraction of elementary graph transformation rules.

Some applications require to mention the inverse direction of links as well as the forward direction. Such situations can be naturally expressed by using inverse modalities in temporal logics. So we proposed the use of two-way CTL (2CTL) for abstraction and applied it to analysis of synchronous and asynchronous cellular automata [8]. In contrast to general graph transformation system, they do not allow to change the connectivity of cells, but the label associated to each cell can be changed according to a given rewrite rules. Representation of a rewrite rule also refers to 2CTL formulas.

To enable analysis be done fully automatically, satisfiability checking of 2CTL formulas plays a central role. For example, adjacency of abstract cells can be derived by checking satisfiability of a 2CTL formula, and if it is shown that they cannot be adjacent, the abstract link between them is deleted.

Another kind of well-known infinite system is that of timed or hybrid systems, whose state contains dense time values. Abstraction methods using a finite set of partitions of time values called regions or zones, are widely known. We proposed timed multiset rewriting systems as a framework that subsumes both timed automata [1] and timed Petri nets [4]. Multiset rewriting can be naturally extended to graph transformation by introducing links between elements of a multiset.

Our current challenge is to extend the abstraction method using temporal logic so that we can provide a uniform framework that works on more general graph transformation systems. We are also planning its timed extension by attaching clock values to each node of a graph.

The rest of the paper is organized as follows. Section 2 compares our framework with related work. Section 3 describes why we use temporal logic for abstraction of graph transformation systems. Section 4 shows possible case studies that we are planning. Implementation status of related tools are described in Sect. 5. Section 6 mentions some aspects of formal verification of the algorithms used in the framework, and Sect. 7 summarizes the paper.

2 Related Work

Separation logic [10] is an extension of Hoare logic, and deals with spatial properties of shared mutable data structures using assertions about disjoint parts of the heap. Though it can reason about data structures that have particular shapes such as lists, basic properties of these structures have to be prepared.

"Shape analysis" methods [5, 12] in abstract interpretation also concern representation of spatial properties of pointer structures. Among them, Sagiv et al.'s work [12] on the TVLA system (Three-Valued-Logic Analyzer) is closely related to ours. One of the differences is that they use logical structures for 2-valued first-order logic for representing the concrete space, whereas ours use Kripke structures. What is more crucial is that ours uses decidable temporal logic. Thus we can use satisfiability checking as a tool to promote automation in analysis. We also make some extensions to temporal logic so that it can suit abstraction of graph transformation systems, as detailed in the next section.

3 Why Temporal Logic?

The use of temporal logic for abstraction is the most important feature of our framework. In this section, we explain why we use temporal logic, and what kinds of extensions to temporal logic we have made or we are going to make in order to apply it to abstraction of graph transformation systems.

A concrete graph can be regarded as a Kripke structure $\mathcal{M} = (M, \{R_a\}, \lambda)$ where M is a set of concrete nodes, $\{R_a\}$ is a collection of labeled relations on M such that sR_at if and only if there is a link labeled a from s to t, and λ is a function that maps a label to the set of the associated nodes. As already mentioned, the key idea is to use temporal modalities for the purpose of describing spatial properties of nodes. For example, if a node s points to another node t by a link labeled a and property p holds at the node t (i.e., sR_at and $t \in \lambda(p)$), then $\mathsf{EX}_a p$ holds at the node s (i.e., $\mathcal{M}, s \models \mathsf{EX}_a p$).

An *abstracted node* is characterized by truth values of some temporal formulas. It represents possibly multiple concrete nodes where each formula has the corresponding truth value. In order for abstract nodes to have only a finite number of variations, we first fix a finite set F of temporal formulas in advance. Then each abstract node is determined by $C \subseteq F$ so that $\bigwedge\{\phi \mid \phi \in C\} \wedge \bigwedge\{\neg\phi \mid \phi \in F \setminus C\}$ (denoted by ϕ_C) is satisfied in the corresponding concrete nodes. That is, a concrete node s in a concrete graph $\mathcal{M}$ is represented by an abstract node $C \subseteq F$ if and only if $\mathcal{M}, s \models \phi_C$ holds. Since F is a finite set, the number of abstract nodes is bounded by $2^{|F|}$.

The use of decidable temporal logic enables us to use satisfiability checking as a tool for automated analysis. In general more expressive logics require higher computational complexity. So we may use a subclass of decidable temporal logic such that its satisfiability checking is feasible, or we may do approximate checking that does not violate conservativeness of abstraction.

CTL is a kind of branching-time temporal logic that has been well studied and widely used especially for model checking. However, in order to use it for analysis using abstraction, we sometimes need more expressibility than that of the bare CTL. In the rest of this section, we describe what kinds of extensions we have made as well as our plan of future extensions.

Two-Way Modality and Satisfiability When we describe spatial properties, we often need to mention not only the modality of the forward direction but also that of the inverse direction. In the example of 1-dimensional cellular automata [8], an abstract cell has to have information on the colors of its left and right neighbors. In that case, it is natural to use inverse modality as well as forward one.

For modality a, the inverse modality of a is denoted as $\overline{a}$ and thus $\overline{\overline{a}} = a$. In this two-way setting, we define *abstract links* between abstract nodes as follows: for abstract nodes C, D and modality a, there is an abstract link labeled a from C to D if and only if both $\phi_C \wedge \mathsf{EX}_a \phi_D$ and $\phi_D \wedge \mathsf{EX}_{\overline{a}} \phi_C$ are satisfiable.

Abstraction of the concrete graph structure is represented as an *abstract graph*, which consists of abstract nodes and abstract links between them. In this paper, to simplify the explanation, we focus on the framework in which an abstract graph is fully induced from a set of abstract nodes, so we denote such a graph by the corresponding set $\mathcal{G} \subseteq 2^F$ of abstract nodes. Satisfiability checking plays a central role in automated generation of the abstract graph as above.

An abstract graph $\mathcal{G}$ is said to be *sound* with respect to a concrete graph $\mathcal{M} = (M, \{R_a\}, \lambda)$ if $\forall s \in M.\ \exists C \in \mathcal{G}.\ \mathcal{M}, s \models \phi_C$ holds. Because $C \subseteq F$ satisfying this condition is uniquely determined from s if it exists, we may write such C as $\alpha(s)$ for $s \in M$. If $\mathcal{G}$ is sound with respect to $\mathcal{M}$, then there is an abstract link labeled a from $\alpha(s)$ to $\alpha(t)$ whenever sR_at holds.

As we mentioned, an abstract node is determined by truth values of user-selected two-way temporal formulas that represent characteristic properties of concrete nodes. This method can be regarded as predicate abstraction by two-way temporal formulas. Sagiv et al. [12] proposed parametric framework for shape analysis using predicate abstraction by user-defined predicates on 3-valued logic. Such predicates include not only "pointed to by the variable x" but also "reachable from the variable x". The latter type of predicate is called an "instrumentation predicate", which does not directly appear in the target program but plays an important role in static analysis. How these predicates change according to a program execution step is specified by "predicate update formulas", and automatic generation of those for instrumentation predicates is a difficult problem especially when transitive closure of a predicate is included [9]. Our approach is to use temporal formulas for representing such instrumentation predicates, and to calculate of the weakest precondition of a program execution step. The details are described in Section 5.

Nominal We also use hybrid temporal logic, which differs from the ordinary temporal logic in that it has two kinds of atomic formulas. One is that of ordinary propositional constants, and atomic formulas of the other kind are called nominals. A nominal has the same role as a propositional constant syntactically, but they are different semantically: in a Kripke structure $\mathcal{M} = (M, R, \lambda)$ for hybrid temporal logic, the domain of λ is the union of the set of propositional constants and the set of nominals, and $\lambda(x)$ is required to be a singleton for a nominal x. That is, a nominal x specifies a unique node in a Kripke structure.

Introduction of nominals increases the power of describing properties of a graph. For example consider the property "node n is in a loop". It can be paraphrased as "n is reachable from n by following the arrows (more than once)," and is therefore expressible as "$x \to \mathsf{EXEF}x$" if x is a nominal that specifies n. If x were just a propositional constant, the formula would not express the property.

In [12], abstract heap structures contain two kinds of nodes: summary nodes and non-summary nodes. A summary node represents "more than one concrete node", and a non-summary one represents "exactly one concrete node." The distinction of these kinds of abstract nodes plays an important role in expressivity and accuracy of abstraction. In our framework, the former kind of abstract node corresponds to a set of usual formulas, while the latter kind corresponds to a set of formulas containing a nominal. Since we cannot express the latter kind of abstract node without nominals, they are essential for doing analysis as in [12].

Shape Analysis is Tableau We use a tableau method for checking satisfiability of a given two-way temporal formula. We construct a tableau by repeatedly removing inconsistent tableau nodes starting from all of the possible ones. The tableau constructed in this way effectively encodes all models that make the given formula satisfiable.

Tableau construction and shape analysis using abstract graphs are closely related as follows. For an abstract graph $\mathcal{G}$, consider a formula $\psi_{\mathcal{G}} = \mathsf{A} \bigvee \{\phi_C \mid C \in \mathcal{G}\}$, where A denotes a global modality [2] and $\mathsf{A}\phi$ means that ϕ is true at all points in a model. Then a Kripke structure $\mathcal{M} = (M, \{R_a\}, \lambda)$ is a model of $\psi_{\mathcal{G}}$ if and only if $\forall s \in M.\ \mathcal{M}, s \models \bigvee \{\phi_C \mid C \in \mathcal{G}\}$. Suppose that $\mathcal{M}$ is a model. Since $\forall s \in M.\ \exists C \subseteq F.\ \mathcal{M}, s \models \phi_C$ always holds, we have $\forall s \in M.\ \exists C \in \mathcal{G}.\ \mathcal{M}, s \models \phi_C$. This means $\mathcal{G}$ is sound with respect to $\mathcal{M}$.

Therefore, all the concrete graphs for an abstract graph used in the shape analysis is covered by the tableau corresponding to a particular formula constructed by the abstract graph. The use of decidable temporal logic and its satisfiability checking tightly connects tableau method with shape analysis in this sense.

Spatio-Temporal Logic Although we have used modalities in temporal logic for the purpose of describing spatial properties, they are originally introduced for describing temporal properties, of course. In the example of cellular automata, connectivity of cells are not changed through temporal evolution of cells, because a cell is not generated or destroyed. Therefore one can think of temporal change of spatial properties of a cell such as “for a cell satisfying S, T holds until U via temporal evolution,” where S, T, and U are spatial properties.

In general, properties of graphs to be verified involve not only spatial relations but also temporal ones as above. So we are now trying to establish such “spatio-temporal” logic that can describe interaction of both kinds of modalities.

In our software model checking tool TLAT described in Sect. 5, we are trying to grasp both spatial and temporal properties of a cell in the heap. Refer to Section 5 for the detail.

4 Case Studies

In this section, we give some case studies we are planning with our framework.

Asynchronous transformation of link structures such as that in a concurrent garbage collection algorithm is a typical example of graph transformation. In the concurrent GC environment, there are two kinds of processes, a mutator and a collector, running concurrently. The mutator process accesses the heap to change the link structure according to the given program. The collector process also accesses the heap to collect unused cells without stopping the mutator process.

The key features in our framework are characterization of heap properties using temporal logic formulas and the use of their satisfiability checking for automation. We are planning to apply them to the shape analysis through predicate abstraction and model checking [5]. It enables us to do shape analysis on heaps by a generic abstraction algorithm using the calculation of the weakest preconditions instead of specialized abstract interpretation algorithms. We also mention the current status along this direction in Section 5.

Distributed algorithms over networks can also be regarded as graph transformation systems. Examples include mutual exclusion, routing protocols, propagation of DNS information, and leader election algorithms on mobile ad hoc network. Timed extension is crucial for analyzing network protocols that have the notion of timeouts.

Cellular automata are also special cases of graph transformation. The shape of the graph does not change, but the label associated each node (cell) does. Transition on cellular automata can be synchronous and asynchronous. The synchronous transition changes the labels of all the nodes simultaneously. The asynchronous one changes the label of one node at a time. The dining philosopher problem taken in [8] is an example of asynchronous cellular automata. How abstraction by temporal logic can be used in the deadlock analysis of this problem is shown.

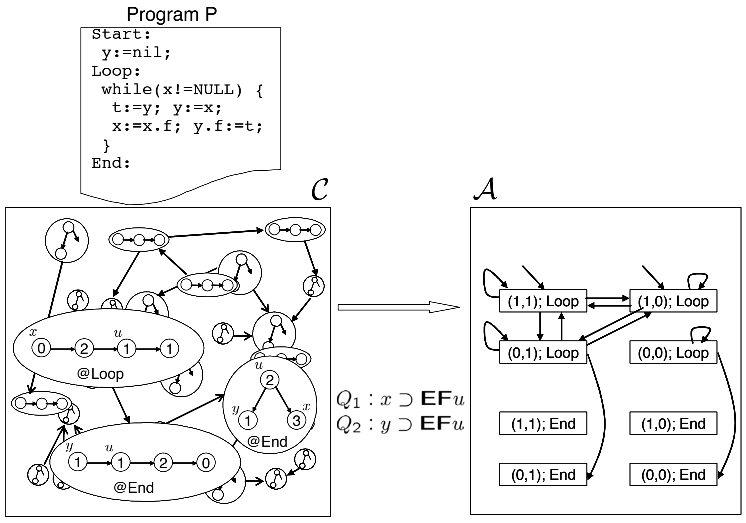
5 Tools

Some tools related to our framework have been implemented or under construction.

Satisfiability Checker As we mentioned, satisfiability checking plays an important role in abstraction under our framework. Although a decidable algorithm is already proposed for two-way modal μ-calculus [20], and for hybrid two-way modal μ-calculus [13], their straightforward implementations cost high computational complexity because of the emptiness problem on infinite tree automata. Moreover, the actual implementation is not reported to the best of our knowledge.

We showed that satisfiability of some subclasses of two-way modal μ-calculus can be effectively checked using a tableau method that is implemented by iterative computation over BDDs [3]. We first implemented a satisfiability checker for two-way CTL [17] with this method. Then it is also applied to alternation-free two-way modal μ-calculus [19] by adding some extra information to the tableau. Overview of this method is explained below.

A simple decision procedure for satisfiability checking in (one-way) CTL, which is a subsystem of the two-way modal μ-calculus, is well-known [6]. By expanding modal formulas (for example, formula $\mathsf{EF}\varphi$ is expanded to $\varphi \vee \mathsf{EXEF}\varphi$), any formula can be regarded as a boolean combination of propositional constants and formulas in the form of $\mathsf{EX}\varphi$ or $\mathsf{AX}\varphi$. We consider a tableau consisting of nodes each of which is a subset of the set of propositional constants and formulas in the form of $\mathsf{EX}\varphi$ and $\mathsf{AX}\varphi$. We regard each formula that belongs to a node is "true" at the node. Starting with the full tableau consisting of all nodes, we repeatedly dispose of "inconsistent" nodes. Among the types of inconsistency, the most important one is the type concerning "eventuality." An example situation is that the formula $\mathsf{EF}\varphi$ is "true" at a node but no node where φ is "true" can



```
Start:
 y:=nil;
Loop:
 while(x!=NULL) {
  t:=y; y:=x;
  x:=x.f; y.f:=t;
 }
End:
```

Fig. 1. Abstraction in TLAT

be reached from there. In the case of (one-way) CTL, this type of inconsistency can be judged relatively easily. However, in the case of two-way logic, the situation is not so simple because of additional inconsistency induced by the inverse modality. As for two-way CTL, an additional check between adjacent tableau nodes suffices. As for alternation-free two-way μ-calculus, we introduced some relation between formulas in tableau nodes. The details are in [19].

We are also developing a hybrid version (i.e., with nominals mentioned in Sect. 3) of a satisfiability checker using the same method.

TLAT We are developing a tool that enables software model checking based on the abstraction method described in the paper, which we call Temporal Logic Abstraction Tool (TLAT). TLAT handles heap structures, called pointer structures, consisting of a finite set of nodes and a finite set of variables. A node stores a value (an element of some fixed finite set) and has pointers with names where each pointer points to a node. A variable also points to a node. And we have introduced a small language called PML (Pointer Manipulation Language), which has a minimum set of assignment statements and control statements to describe operations on pointer structures. Figure 1 shows an example of a program P in PML, which reverses a given "list" headed by a node pointed to by variable x at label Start and the resulting list is headed by a node pointed to by variable y at label End. A PML program such as P induces a transition system $\mathcal{C} = (C, \rightarrow)$, where an element of C is a pair of a pointer structure and a value of the program counter.

We consider hybrid two-way CTL with values (stored in nodes) as propositional constants and variables as nominals. We also consider a pair (v, φ) of a program variable and a formula of the hybrid two-way CTL and call it a *p-formula*. A p-formula (v, φ) is also denoted by $v \supset \varphi$, and can be considered as a predicate on pointer structures: it holds on pointer structure S if the node pointed to by v in S satisfies φ. For example, p-formula $x \supset \mathsf{EF}_f u$ expresses that the node pointed to by variable x reaches the node pointed to by u by following pointers labeled f. We also introduce notation "$@l$" meaning that the current execution point is at label l. In TLAT a specification of a program is expressed by an LTL formula with p-formulas and "$@l$" notations as atomic formulas. For example the specification "Any node in the list headed by the node pointed to by variable x at label Start is contained in the list headed by the node pointed to by variable y at label End" can be expressed by the LTL formula $\square(@\mathtt{Start} \wedge x \supset \mathsf{EF}u \rightarrow \square(@\mathtt{End} \rightarrow y \supset \mathsf{EF}u))$.

It is not possible to verify such a specification by model-checking the transition system $\mathcal{C}$ since it is obviously infinite. TLAT constructs a finite transition system $\mathcal{A} = (A, \rightarrow)$ which is a sound abstraction with respect to LTL formulas in the sense that if an LTL formula holds in $\mathcal{A}$ it is guaranteed that it also holds in $\mathcal{C}$. Let $Q_1, \ldots, Q_n$ be p-formulas appearing in the specification plus p-formulas given by the user as hints for abstraction. An element of A is a pair of a sequence of 0 and 1 of length n and a value pc of the program counter. For $a = (a_1, \ldots, a_n)$ where $(a, pc) \in A$, we denote by $Q(a)$ the conjunction of Q_j (if $a_j = 1$) or $\neg Q_j$ (if $a_j = 0$). Following the spirit of "shape analysis is tableau" in Sect. 3, a heap structure at the program counter pc is abstracted to $(a, pc) \in A$ if and only if it is encoded in the tableau for the formula $\bigwedge\{\mathsf{A}Q_i \mid a_i = 1\} \wedge \bigwedge\{\mathsf{A}\neg Q_i \mid a_i = 0\}$.

If an assignment statement s is in the location of pc, whether $(a, pc) \rightarrow (a', pc+1)$ holds or not can be decided by judging whether $Q(a) \wedge \mathrm{wp}(s, Q(a'))$ is satisfiable or not, where $\mathrm{wp}(s, T)$ is the weakest precondition of T with respect to s and can be calculated as in [18]. TLAT performs model checking with the specification against $\mathcal{A}$. If the result is positive, the verification succeeds. Otherwise the user investigates if the counterexample obtained as the result of the model-check corresponds to a real counterexample in $\mathcal{C}$. If not, a new set of p-formulas is to be added to $Q_1, \ldots, Q_n$ to get a more precise abstract system.

Region Analysis Tool As we already mentioned, we are planning to add clock values to nodes. Traditionally, abstraction of clock values is done by dividing the infinite value space into finite partitions. Because our framework abstracts several concrete nodes into an abstract node, we need to consider an *abstract clock* that represents several concrete clocks whose values are in a certain interval.

We are also planning to develop a checker that does region analysis on abstract clocks. A rough algorithm has been made and we are now trying to implement and evaluate the checker.

6 Formal Verification

Verification algorithms themselves can also be a target of verification. It increases the reliability and confidence of their implementation. In particular, graph algorithms are extensively used in verification, and actually both the satisfiability checking algorithm for two-way temporal logics and the abstract model checking of graph transformation systems are instances of graph algorithms. Our group has some experiences of formal verification on graphs [21, 22] using a theorem proving environment, so we are also planning formal verification of graph algorithms used in our framework. In particular, the tableau method used in the satisfiability checker is already fairly complicated and it deserves formal verification.

Besides ensuring correctness, a merit of formal verification is that it leads to better understanding of the target algorithm. Thus we expect that we can grasp the essence of the algorithm and understand both the possibility of extension and its limitation.

7 Summary and Conclusion

We described some results and ongoing projects on abstraction and verification of graph transformation systems by temporal logic. Crucial aspects in the use of temporal logic are decidability, automation by satisfiability checkers, and several kinds of extensions. A brief sketch of the framework, possible case studies, related tools, and some aspects of formal verification are presented. We are now experimenting the effectiveness of the framework through concrete applications.

Acknowledgments The authors are grateful to anonymous reviewers for their constructive comments.

This research was supported by Core Research for Evolutional Science and Technology (CREST) Program "New High-performance Information Processing Technology Supporting Information-oriented Society" of Japan Science and Technology Agency (JST). This research was also partially supported by the Ministry of Education, Science, Sports and Culture, Grant-in-Aid for Scientific Research on Priority Areas, 16016211, 2004.

References

1. Alur, R., and Dill, D.L.: A Theory of Timed Automata, *Theoretical Computer Science*, Vol. 126, 1994, pp. 183–236.
2. Blackburn, P., de Rijke, M., and Venema, Y.: *Modal Logic*, Cambridge University Press, 2001
3. Bryant, R.E: Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams, *ACM Computing Surveys*, Vol.24, No.3(1992), pp. 293–318.
4. Cerone, A., and Maggiolo-Schettini, A.: Time-Based Expressivity of Time Petri Nets for System Specification. *Theoretical Computer Science*, Vol. 216, 1999, pp. 1–53.
5. Dams, D., and Namjoshi, K. S.: Shape analysis through predicate abstraction and model checking, *Fourth International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI03)*, 2003, pp. 310–324.

6. Emerson, E. A.: Temporal and Modal Logic, *Handbook of Theoretical Computer Science (vol. B): Formal Models and Semantics*, Elsevier, 1990, pp.995–1072.
7. Grädel, E., Thomas, W., and Wilke, T. (eds.): *Automata, Logics, and Infinite Games: A Guide to Current Research*, volume 2500 of LNCS, Springer, 2002.
8. Hagiya, M., Takahashi, K., Yamamoto, M., and Sato, T.: Analysis of Synchronous and Asynchronous Cellular Automata using Abstraction by Temporal Logic, *FLOPS2004*, volume 2998 of LNCS, 2004, pp. 7–21.
9. Reps, T., Sagiv, M., Loginov, A.: Finite Differencing of Logical Formulas for Static Analysis, *European Symposium on Programming*, 2003, pp. 380–398.
10. Reynolds, J. C.: Separation Logic: A Logic for Shared Mutable Data Structures, *Proceedings of the Seventeenth Annual IEEE Symposium on Logic in Computer Science*, 2002, pp. 55–74.
11. Rozenberg, G. (eds.): *Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 1: Foundations*, World Scientific, 1997.
12. Sagiv, M., Reps, T., and Wilhelm, R.: Parametric shape analysis via 3-valued logic, *ACM Transactions on Programming Languages and Systems*, Vol. 24, No. 3, May 2002, pp. 217–298.
13. Sattler, U. and Vardi, M. Y.: The Hybrid μ-Calculus. *Proceedings of the International Joint Conference on Automated Reasoning*, volume 2083 of LNAI, pages 76–91. Springer Verlag, 2001
14. Takahashi, K., and Hagiya, M.: Abstraction of Link Structures by Regular Expressions and Abstract Model Checking of Concurrent Garbage Collection, *First Asian Workshop on Programming Languages and Systems*, 2000, pp. 1–8.
15. Takahashi, K. and Hagiya, M.: Formal Proof of Abstract Model Checking of Concurrent Garbage Collection, *Workshop on Thirty Five years of Automath, Informal Proceedings*, Heriot-Watt University, Edinburgh, April, 2002, pp. 115–126.
16. Takahashi, K., and Hagiya, M.: Abstraction of Graph Transformation using Temporal Formulas. *Supplemental Volume of the 2003 International Conference on Dependable Systems and Networks (DSN-2003)*, 2003, pp. W-65 – W-66.
17. Tanabe, Y., Takahashi, K., Yamamoto, M., Sato T., and Hagiya, M.: An Implementation of a Decision Procedure for Satisfiability of Two-Way CTL Formulas Using BDD (in Japanese), *Computer Software*, Japan Society for Software Science and Technology, Vol. 22, No. 3, 2005, pp. 154–166.
18. Tanabe, Y., Takai, T., Sekizawa, T., Takahashi, K.: Preconditions of Properties Described in CTL for Statements Manipulating Pointers. *Supplemental Volume of the 2005 International Conference on Dependable Systems and Networks*, June 28 – July 1, 2005, pp.228–234
19. Tanabe, Y., Takahashi, K., Yamamoto, M., Tozawa, A., and Hagiya, M.: A Decision Procedure for the Alternation-Free Two-Way Modal μ-Calculus, *TABLEAUX 2005*, volume 3702 of LNAI, 2005, pp. 277–291.
20. Vardi, M.Y.: Reasoning about The Past with Two-Way Automata: *ICALP 98*, volume 1443 of LNCS, 1998, pp. 628–641.
21. Yamamoto, M., Nishizaki, S., Hagiya, M., and Toda, Y.: Formalization of Planar Graphs, *8th International Workshop on Higher-Order Logic Theorem Proving and Its Applications*, volume 971 of LNCS, 1995, pp. 369–384.
22. Yamamoto, M., Takahashi, K., Hagiya, M., Nishizaki, S., and Tamai, T.: Formalization of Graph Search Algorithms and Its Applications. *Theorem Proving in Higher Order Logics: TPHOLs '98*, volume 1479 of LNCS, 1998, pp. 479–496
23. Yamamoto, M., Cottin, J.-M., and Hagiya, M.: Decidability of Safety Properties of Timed Multiset Rewriting, *FTRTFT 2002*, volume 2469 of LNCS, 2002, pp. 165–183.