# 遷移関係の詳細化による正則モデル検査の再構成と拡張 *

Reconstructing and Extending
Regular Model Checking by Refinement of Transition Relation

櫻田英樹 † 萩谷昌己 †‡
Hideki Sakurada Masami Hagiya

† 日本電信電話株式会社 NTT コミュニケーション科学基礎研究所
NTT Communication Science Laboratories, NTT Corporation

‡ 東京大学大学院情報理工学研究科
Graduate School of Information Science and Technology, University of Tokyo

Regular model checking is a framework for verifying parameterized and infinite-state systems. Techniques for regular model checking such as quotienting require a well-specified transition relation where the invariant of the system is represented implicitly. To relax the restriction, we propose a technique, which we call *refinement of transition relation*, and reconstruct and extend regular model checking by the technique. We apply the technique to a token-passing system and a program that dynamically mutates a heap structure.

## 1 Introduction

Regular model checking[6] is a framework for verifying parameterized and infinite-state systems. In regular model checking, sets of configurations and transition relations are represented by finite automata. Each step of a transition is computed by ordinary operations on finite automata.

For example, we consider a simple token-passing system. It consists of a finite but unbounded number of linearly connected processes. Each process either has a token or not. In a transition of the system, the process having a token passes it to the right. A sequence of transitions in the system is represented by a matrix. An example of a matrix is shown in Fig.1. Horizontal arrows represent the neighboring relation, and vertical arrows represent local transitions. Processes having a token or not

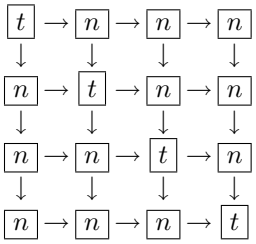



Figure 1: Token-Passing System

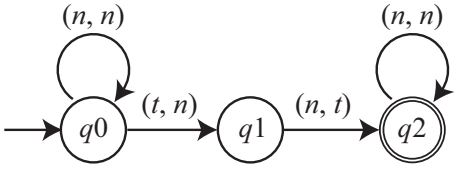



Figure 2: Finite Automaton for the Transition Relation of the Token-Passing System

are indicated by the letter '$t$' or '$n$,' respectively. A configuration of the system is represented by a word over the set $\Sigma = \{t, n\}$. For example, the initial state in Fig.1 is represented by *tnnnn*. The transition relation is represented by the automaton over $\Sigma \times \Sigma$ shown in Fig.2. A transition from $s_1 \cdots s_n$ to $s'_1 \cdots s'_n$ is possible if and only if $(s_1, s'_1) \cdots (s_n, s'_n)$ is accepted by the automaton. We refer to such transition relations as *regular transition relations.* The composition of a pair of regular transition relations is computed by the ordinary operations on automata. The result is also a regular transition relation. We can therefore calculate the $n$-fold composition $T^n$ of a transition relation $T$ for a given constant $n$.

To verify safety or liveness[7] of a system in regular model checking, we need to calculate the transitive closure $T^+$ of the transition relation $T$, which is not in general regular. Even if it is regular, it may not be computable by iterations. Some techniques have been proposed to calculate or approximate transitive closures under certain conditions. *Quotienting* is techniques to exactly calculate tran-

*This work is partially supported by the Ministry of Education, Culture, Sports, Science and Technology Grant-in-Aid for Scientific Research (B)(2) 18500003.

sitive closures and has been applied for verifying many systems. In the example above, we obtain the transitive closure shown in Fig.3.

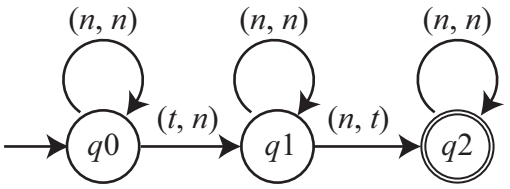



Figure 3: Finite Automaton for the Closure of the Transition Relation of the Token-Passing System

However, to obtain the expected result by quotienting, we need to provide a well-specified transition relation that implicitly contains the information about the invariant of the system. In the example, the automaton of the transition relation is well-specified because it contains the information about the invariant: only one token appears in each reachable configuration. If we provide the transition relation where multiple tokens are passed (Fig.4) instead, we can not obtain the transitive closure, which is not regular, by quotienting, although the set of reachable configurations remains the same as of the original transition relation.

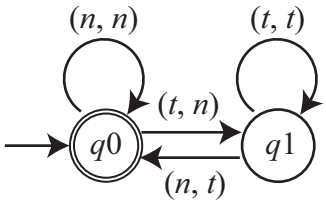



Figure 4: Finite Automaton for the Transition Relation where Multiple Tokens are Passed

In this paper, we relax such restriction by introducing a technique which we call *refinement of transition relation*. Moreover, we reconstruct regular model checking by the technique so that we can (semi-) automatically derive the invariant of a transition relation and compute its transitive closure. We also extend regular model checking to analyze dynamic heap structures. Specifically we make the following contributions:

- We present a technique called *refinement of transition relation*. We compute the invariant of the token-passing system and refine states of the processes. We also give a refined transition relation and represent it as a set of tiles.
- From the refined transition relation, we calculate an approximation of the transitive closure of the system by constructing a neighborhood graph of the system. We also show that the approximation coincides with the exact transitive closure.
- We extend the construction to analyze a program that operates on a heap. As an example, we prove the termination of a program that reverses linear lists.

## 2 Refinement of Transition Relation

We start with the token-passing system with an arbitrary number of tokens that has the rule: a process has a token after a transition if and only if it has a left neighbor that has a token before the transition. We assume that only the leftmost process has a token in the initial configuration.

In our refinement, we first give a set of atomic predicates on processes and then we calculate the temporal evolution of the properties represented in first-order logic. For our example, we give the atomic predicates as follows.

$t_0(x)$: Process $x$ has a token.

$n_0(x)$: There is a process with a token that is reachable from process $x$ by traversing one or more links from left to right.

$m_0(x)$: There is a process with a token that is reachable from process $x$ by traversing one or more links from right to left.

We also use a binary relation $\rightarrow$ and its transitive closure $\twoheadrightarrow$. We have $x \rightarrow y$ if and only if $x$ is the left neighbor of process $y$. We have the following logical equivalences among the atomic predicates.

$$\begin{aligned} n_0(x) &= \exists y.(x \twoheadrightarrow y \wedge t_0(y)) \\ m_0(x) &= \exists y.(y \twoheadrightarrow x \wedge t_0(y)) \end{aligned}$$

We refer to a Boolean combination of the atomic predicates as a (refined) state. The initial configuration is represented by states as follows.

$$\boxed{t_0 \wedge \neg n_0 \wedge \neg m_0} \rightarrow \boxed{\neg t_0 \wedge \neg n_0 \wedge m_0} \rightarrow \boxed{\neg t_0 \wedge \neg n_0 \wedge m_0} \rightarrow \cdots$$

The transition rule of the system is represented as follows.

$$\mathbf{wp}(t_0(x)) = \exists y.(y \rightarrow x \wedge t_0(y))$$

The left-hand side $\mathbf{wp}(t_0(x))$ is the weakest precondition of $t_0(x)$, i.e. $t_0(x)$ holds after a transition if and only if $\mathbf{wp}(t_0(x))$ holds before the transition.

The weakest preconditions of $n_0$ and $m_0$ are calculated as follows.

$$\begin{aligned}
\mathbf{wp}(n_0(x)) &= \exists y.(x \twoheadrightarrow y \wedge \mathbf{wp}(t_0(x))) \\
&= \exists y, z.(x \twoheadrightarrow y \wedge z \to y \wedge t_0(z)) \\
&= t_0(x) \vee n_0(x) \\
\mathbf{wp}(m_0(x)) &= \exists y.(y \twoheadrightarrow x \wedge \mathbf{wp}(t_0(x))) \\
&= \exists y, z.(y \twoheadrightarrow x \wedge z \to y \wedge t_0(z)) \\
&= \exists y.(y \to x \wedge m_0(y))
\end{aligned}$$

From these equations, we calculate the second configuration as follows.

$$\begin{array}{ccccccc}
\boxed{t_0 \wedge \neg n_0 \wedge \neg m_0} & \to & \boxed{\neg t_0 \wedge \neg n_0 \wedge m_0} & \to & \boxed{\neg t_0 \wedge \neg n_0 \wedge m_0} & \to \cdots \\
\downarrow & & \downarrow & & \downarrow & \\
\boxed{\neg t_0 \wedge n_0 \wedge \neg m_0} & \to & \boxed{t_0 \wedge \neg n_0 \wedge \neg m_0} & \to & \boxed{\neg t_0 \wedge \neg n_0 \wedge m_0} & \to \cdots
\end{array}$$

Let $t = t_0 \wedge \neg n_0 \wedge \neg m_0$, $m = \neg t_0 \wedge \neg n_0 \wedge m_0$, and $n = \neg t_0 \wedge n_0 \wedge \neg m_0$. By calculating the third configuration in the same way, we notice that the processes in the subsequent configurations are either in $t$, $m$, or $n$ state. Therefore $t \vee m \vee n$ is the invariant. This implies that at most one process holds a token in any configurations reachable from the initial configuration.

We can also calculate the set of possible transitions, which we call *tiles*, on two neighboring processes. A tile is a pair $(s_1, s_1')(s_2, s_2')$ of pairs of states such that $s_1(x) \wedge s_1'(x) \wedge \mathbf{wp}(s_2'(x)) \wedge \mathbf{wp}(s_2'(x))$ is satisfiable. The set of tiles of the token-passing system is shown in Fig.5. We say

$$\text{(A)}\ \begin{array}{ccc} \boxed{n} & \to^{\to} & \boxed{n} \\ \downarrow & & \downarrow \\ \boxed{n} & \to & \boxed{n} \end{array} \qquad \text{(B)}\ \begin{array}{ccc} \boxed{n} & \to & \boxed{t} \\ \downarrow & & \downarrow \\ \boxed{n} & \to & \boxed{n} \end{array} \qquad \text{(C)}\ \begin{array}{ccc} \boxed{t} & \to & \boxed{m} \\ \downarrow & & \downarrow \\ \boxed{n} & \to & \boxed{t} \end{array}$$

$$\text{(D)}\ \begin{array}{ccc} & \to & \boxed{m} \\ & & \downarrow \\ \boxed{t} & \to & \boxed{m} \end{array} \qquad \text{(E)}\ \begin{array}{ccc} & \to & \boxed{m} \\ & & \downarrow \\ \boxed{m} & \to & \boxed{m} \end{array}$$

Figure 5: Tiles of the Token-Passing System

a state $s$ *requires a right or left neighbor*, which is indicated by an arrow as right or left superscript of the state in tiles, if and only if $s(x) \supset \exists y.x \to y$ or $s(x) \supset \exists y.y \to x$ is true for any process $x$, respectively. In our example, $n$ and $m$ require a right and left neighbor, respectively.

From the set of tiles, we generate matrices where each $2 \times 2$ segment is tiles and the leftmost and the rightmost states do not require right or left neighbors, respectively. By construction, for any transition sequence of a system, there exists a generated matrix that is *satisfied* by the sequence. We say that a generated matrix is satisfied by a transition sequence if and only if the height and the width of the matrix is identical to the length and the number of processes of the sequence, respectively, and for any $(i, j)$ element $l$ of the matrix, $l(x)$ holds for the $i$-th process $x$ in the $j$-th configuration in the sequence.

The converse does not hold in general. It however holds for our example: for any generated matrix the first row of which is satisfied by the initial configuration, it is satisfied by the transition sequence that is constructed by letting each process in state $t$ has a token. The proof is by induction on the height of the matrix. We show $n_0(x)$ holds for process $x$ in a configuration if $x$ corresponds to an occurrence of state $n$ in the matrix. Other cases are proved similarly. Because $n$ does not appear in the rightmost column of the matrix, either tiles (A), (B), or (C) occurs at the position where the occurrence of $n$ in question corresponds to the bottom-left letter of the tile. In case of (B) and (C), $n_0(x)$ trivially holds. In case of (A), the letter above the occurrence of $n$ in question is also $n$. By the induction hypothesis, $n_0(x)$ holds in the previous configuration. This implies that there is a process $y$ that has a token and is reachable from $x$. In the current configuration, the right neighbor of $y$ has a token. Therefore $n_0(x)$ holds.

## 3 Closure Construction by Vertical Tiling

In this section, we show that we can construct the transitive closure of a transition relation from vertical compositions of tiles.

We first define the neighboring relation $\to$ between *histories*. A history is a sequence of refined states of a system. For histories $h = s_0 s_1 \cdots s_l$ and $h' = s_0' s_1' \cdots s_l'$, we have $h \to h'$ if and only if $(s_i, s_i')(s_{i+1}, s_{i+1}')$ is a tile for all $i = 0, 1, \ldots, k-1$. For example, we have $tnn \to mtn$, which is depicted as follows.

$$\begin{array}{ccc} \boxed{t} & \to & \boxed{m} \\ \downarrow & & \downarrow \\ \boxed{n} & \to & \boxed{t} \\ \downarrow & & \downarrow \\ \boxed{n} & \to & \boxed{n} \end{array}$$

A pair of neighboring histories $s_0 s_1 \cdots s_l \to s_0' s_1' \cdots s_l'$ can be seen as a sequence $(s_0, s_0')(s_1, s_1') \cdots (s_l, s_l')$. We therefore identify the neighboring relation as a language over the set of pairs of states. The language is regular because it is generated from a finite set of tiles. More precisely, it is a local language (Sect.6.1 of

[10]) over the set of pairs of refined states that is generated from a set of tiles. The neighboring relation is extended over sets of histories. We have $L \to L'$ if and only if there exists $h \in L$ and $h' \in L'$ such that $h \to h'$.

The neighboring relation of the token-passing system is given as a union of the relations shown in the first column of Tab.1. The second and the third columns are the sets of left and right neighbors in each relation, respectively.

Table 1: Neighboring Relation Over the Histories of the Token-Passing System

| Relation | Left | Right |
|---|---|---|
| $(m,m)^+$ | $m^+$ | $m^+$ |
| $(m,m)^+(t,m)$ | $m^+t$ | $m^+m$ |
| $(m,m)^+(t,m)(n,t)$ | $m^+tn$ | $m^+mt$ |
| $(m,m)^+(t,m)(n,t)(n,n)^+$ | $m^+tnn^+$ | $m^+mtn^+$ |
| $(t,m)$ | $t$ | $m$ |
| $(t,m)(n,t)$ | $tn$ | $mt$ |
| $(t,m)(n,t)(n,n)^+$ | $tnn^+$ | $mtn^+$ |
| $(n,t)$ | $n$ | $t$ |
| $(n,t)(n,n)^+$ | $nn^+$ | $tn^+$ |
| $(n,n)^+$ | $n^+$ | $n^+$ |

By merging overlapping sets of left or right neighbors in Tab.1, we obtain the languages $m^+, m^+t, m^+tn^+, t, tn^+$, and $n^+$. From the languages we construct the neighborhood graph of the token-passing system shown in Fig.6. The edges in the graph represent the neighboring relation between languages.

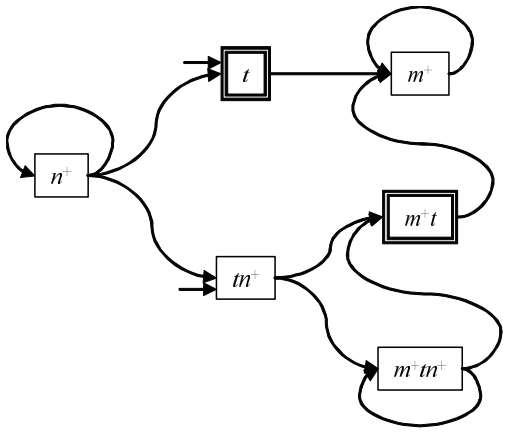



Figure 6: Neighborhood Graph of the Token-Passing System

A sequence of transitions of a system can be represented as a sequence of neighboring histories $h_1 \to h_2 \to \cdots \to h_k$, where $k$ is the number of processes in the sequence. By construction of the neighborhood graph, for any such sequence, there exists a finite path $L_1 \to L_2 \to \cdots \to L_k$ in the neighborhood graph that satisfies the following conditions:

1. $h_i \in L_i$ for any $i = 1, \ldots, k$, and
2. Any state that appears in a history belonging to $L_1$ or $L_k$ does not require a left or right neighbor, respectively.

The inverse also holds for the token-passing system. It is shown by the similar discussion to [9, 1, 6, 8]. In general, for any finite path in the neighborhood graph of a system, there exists a sequence of neighboring histories of the system that corresponds to the finite path if there exists a relation $\sim$ over histories that satisfies the following conditions for any horizontal edge $L_1 \to L_2$:

1. For any history $h_1 \in L_1$, there exists $h_1' \in L_1$ and $h_2 \in L_2$ such that $h_1 \sim h_1' \to h_2$, and
2. $\sim$ is a backward simulation, i.e. for any histories $h_1 \in L_1$ and $h_2, h_2' \in L_2$, if $h_1 \to h_2 \sim h_2'$, then there exists $h_1' \in L_1$ such that $h_1' \to h_2'$.

For our example, the least congruence $\sim$ that contains $n \sim nn$ satisfies these conditions. The relation ignores the number of the repetition of the left-copying state $n$.

## 4 Application to Heap Analysis

In this section, we apply the technique developed in the previous section to a heap analysis problem. We prove the termination of a program that operates on a heap. We derive contradiction by assuming there is infinite sequence of transitions.

A *heap* is a set of *cells* that may refer to one another. Here we restrict a cell to refer to at most one cell. Cells may also be referred to from some *program variables*. The references in a heap may change through an execution of a program. We consider reverse program (Fig.7) as an example of a program that operates on a heap. It reverses a list, a sequence of cells each of which refers to the next one, referred to from variable $t$ and the result is referred to from variable $u$. With an execution of the block in the while loop, the first cell in the list referred to from $t$ is prepended to the list referred to from $u$ (Fig.9). For simplicity, we consider a modified version (Fig.8) of reverse where assignments are executed simultaneously.

Let us choose atomic propositions:

$t_0$: referred to from variable $t$,

```
u:=nil;
while (t != nil) do
begin
  v := u;
  u := t;
  t := t.next;
  u.next := u
end
```

Figure 7: List Reverse

```
u:=nil;
while (t != nil) do
begin
  (t, u, t.next) := (t.next, t, u)
end
```

Figure 8: List Reverse by Simultaneous Assignment

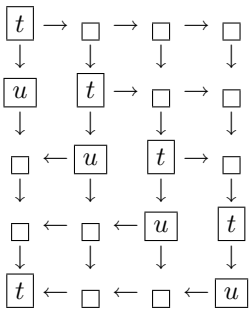


Figure 9: Execution of reverse

$u_0$: referred to from variable $u$,

$m_0$: reachable from variable $t$,

$n_0$: reachable from variable $u$,

**1**: referred to from exactly one cell and

**0**: referred to from no cell.

From the atomic propositions, let $t$, $u$, $m$, and $n$ be the propositions defined as follows.

$$\begin{aligned} t &= t_0 \wedge \neg u_0 \wedge \mathbf{0} \\ u &= u_0 \wedge \neg t_0 \wedge \mathbf{0} \\ m &= m_0 \wedge \neg t_0 \wedge \neg u_0 \wedge \mathbf{1} \\ n &= n_0 \wedge \neg t_0 \wedge \neg u_0 \wedge \mathbf{1} \end{aligned}$$

As for the propositions and their weakest preconditions, we have:

$$\begin{aligned} (\exists y.t(y) \wedge y \rightarrow x) \wedge m(x) &\supset \mathbf{wp}(t(x)) \\ u(x) &\supset \mathbf{wp}(\exists y.u(x) \wedge y \rightarrow x) \\ (\exists y.m(y) \wedge y \rightarrow x \wedge m(x) &\supset \mathbf{wp}(m(x)) \\ t(x) &= \mathbf{wp}(u(x)) \\ n(x) \vee u(x) &\supset \mathbf{wp}(n(x)). \end{aligned}$$

The heap structure may change in each step of the program execution. We therefore define a *dynamic tile* by extending the definition of a tile. A *dynamic tile* is a pair $(s_1, l_1, s_1')(s_2, l_2, s_2')$ of triples such that $s_1(x) \wedge l_1(x, y) \wedge s_1'(y) \wedge \mathbf{wp}(s_2(x)) \wedge \mathbf{wp}(l_2(x, y)) \wedge \mathbf{wp}(s_2'(y))$ is satisfiable, where $l_i$ is either $\rightarrow$, $\leftarrow$, or *nolink* and both of $l_1$ and $l_2$ must not be *nolink*. Binary predicate *nolink* is defined as $nolink(x, y) = \neg(x \rightarrow y) \wedge \neg(y \rightarrow x)$. The set of dynamic tiles of reverse are shown in Fig.10. The definition of neighboring relation $\rightarrow$

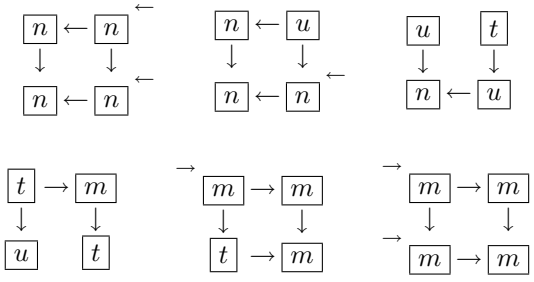


Figure 10: Dynamic Tiles of reverse

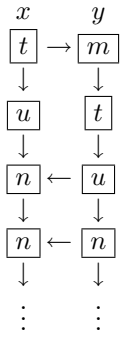
over histories is also extended. For histories $h = (s_0, s_1, \ldots, s_n)$ and $h' = (s_0', s_1', \ldots, s_n')$, we have $h \rightarrow h'$ if and only if there is a sequence of triples $(s_0, l_0, s_0') \cdots (s_n, l_n, s_n')$ that satisfies:

1. $l_0$ is $\rightarrow$, and
2. $(s_i, l_i, s'_i)(s_{i+1}, l_{i+1}, s'_{i+1})$ is a dynamic tile for each $i = 0, \ldots, n-1$.

We prove the termination of reverse by constructing the neighborhood graph of infinite histories of the system instead of finite histories. The neighboring relation over infinite histories is constructed from the following infinite sequences of triples.

$$
\begin{aligned}
&(m \rightarrow m)^\omega \\
&(m \rightarrow m)^+(t \rightarrow m)(u, nolink, t)(n \leftarrow u)(n \leftarrow n)^\omega \\
&\qquad (t \rightarrow m)(u, nolink, t)(n \leftarrow u)(n \leftarrow n)^\omega \\
&\qquad\qquad (u, nolink, t)(n \leftarrow u)(n \leftarrow n)^\omega \\
&\qquad\qquad\qquad (n \leftarrow u)(n \leftarrow n)^\omega \\
&\qquad\qquad\qquad\qquad (n \leftarrow n)^\omega
\end{aligned}
$$

Let us observe a cell $x$ whose history is $tun^\omega$. When the cell is in state $n$, it must be referred to by another cell $y$. The possible histories of the two cells are uniquely determined as follows by the set of tiles.




Therefore $x$ must refer to $y$ in the initial configuration. Similarly, a cell $x$ whose history belongs to $m^+tun^\omega$ must refer to another cell in the initial configuration. Therefore a path in the neighborhood graph of infinite histories (Fig.11) that starts at $tun^\omega$ never terminates. This implies either there is a loop in the initial configuration or there are an infinite number of cells. This contradicts our assumptions.

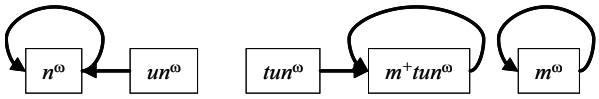



Figure 11: Neighborhood Graph of reverse

## 5 Conclusion and Future Work

We have so far proposed a technique called *refinement of transition relation*. We have shown that it can be used in regular model checking to refine the states of processes and to compute the invariant from a simple specification of the system. We have also applied the technique to analysis of a program that dynamically mutates a heap structure.

Our approximation of transitive closures seems to be related to abstract regular model checking[5]. One of the differences is that we refine transition relations so that approximation becomes more precise. Another difference is that we first calculate the set of histories to build transitive closures while in abstract regular model checking configurations are abstracted after each step of transitions. The detailed comparison is left as future work.

The heap structure considered in this paper has no sharing and consists of cells each of which refer to at most one cell. Regular model checking has been applied to heap structures with sharing[3] and with cells each of which may refer to several cells[4]. Our technique might be useful in combination with these results.

The authors plan to extend our technique to a nonregular class of languages. In such an extension, symbolic representations of nonregular sets of histories and transitive closures are needed. One of the candidates of such representations is CQDD[2]. A CQDD consists of finite automata with linear arithmetical constraints on number of occurrences of symbols. CQDDs have been used to represent nonregular sets of configurations in the analysis of FIFO-channel systems. The extension would be useful, for example, in verifying a system whose safety depends on synchronization of components of the system such as counters.

## Acknowledgments

The authors thank Yoshinori Tanabe, Izumi Takeuti, Mitsuharu Yamamoto, Yoshinobu Kawabe, Ken Mano, Yasuyuki Tsukada, Kiyoshi Shirayanagi, Toshiharu Sugawara for fruitful discussions and their supports.

### References

[1] Parosh Aziz Abdulla, Bengt Jonsson, Marcus Nilsson, and Julien d'Orso. Regular model checking made simple and efficient. In Lubos Brim, Petr Jancar, Mojmír Kretínský, and Antonín Kucera, editors, *CONCUR*, volume 2421 of *Lecture Notes in Computer Science*, pages 116–130. Springer, 2002.

[2] Ahmed Bouajjani and Peter Habermehl. Symbolic reachability analysis of fifo-channel systems with nonregular sets of configurations. *Theor. Comput. Sci.*, 221(1-2):211–250, 1999.

[3] Ahmed Bouajjani, Peter Habermehl, Pierre Moro, and Tomás Vojnar. Verifying programs with dynamic 1-selector-linked structures in regular model

checking. In Nicolas Halbwachs and Lenore D. Zuck, editors, *TACAS*, volume 3440 of *Lecture Notes in Computer Science*, pages 13–29. Springer, 2005.

[4] Ahmed Bouajjani, Peter Habermehl, Adam Rogalewicz, and Tomás Vojnar. Abstract tree regular model checking of complex dynamic data structures. In Kwangkeun Yi, editor, *Static Analysis*, volume 4134 of *Lecture Notes in Computer Science*. Springer, 2006.

[5] Ahmed Bouajjani, Peter Habermehl, and Tomás Vojnar. Abstract regular model checking. In Rajeev Alur and Doron Peled, editors, *CAV*, volume 3114 of *Lecture Notes in Computer Science*, pages 372–386. Springer, 2004.

[6] Ahmed Bouajjani, Bengt Jonsson, Marcus Nilsson, and Tayssir Touili. Regular model checking. In E. Allen Emerson and A. Prasad Sistla, editors, *CAV*, volume 1855 of *Lecture Notes in Computer Science*, pages 403–418. Springer, 2000.

[7] Ahmed Bouajjani, Axel Legay, and Pierre Wolper. Handling liveness properties in (*mega-*)regular model checking. *Electr. Notes Theor. Comput. Sci.*, 138(3):101–115, 2005.

[8] Dennis Dams, Yassine Lakhnech, and Martin Steffen. Iterating transducers. *J. Log. Algebr. Program.*, 52-53:109–127, 2002.

[9] Bengt Jonsson and Marcus Nilsson. Transitive closures of regular relations for verifying infinite-state systems. In Susanne Graf and Michael I. Schwartzbach, editors, *TACAS*, volume 1785 of *Lecture Notes in Computer Science*, pages 220–234. Springer, 2000.

[10] Arto Salomaa. *Jewels of Formal Language Theory*. Financial Times Prentice Hall, 1981.