7-2009

PAT: Towards flexible verification under fairness

Jun SUN
Singapore Management University, junsun@smu.edu.sg

Yang LIU

Jin Song DONG

Jun PANG

PAT: Towards Flexible Verification under Fairness

Jun Sun[1], Yang Liu[1], Jin Song Dong[1], and Jun Pang[2]

[1] School of Computing, National University of Singapore
[2] Computer Science and Communications, University of Luxembourg

Abstract. Recent development on distributed systems has shown that a variety of fairness constraints (some of which are only recently defined) play vital roles in designing self-stabilizing population protocols. Current practice of system analysis is, however, deficient under fairness. In this work, we present PAT, a toolkit for flexible and efficient system analysis under fairness. A unified algorithm is proposed to model check systems with a variety of fairness effectively in two different settings. Empirical evaluation shows that PAT complements existing model checkers in terms of fairness. We report that previously unknown bugs have been revealed using PAT against systems functioning under strong global fairness.

1 Introduction

In the area of system/software verification, liveness means something good must eventually happen. A counterexample to a liveness property is typically a loop (or a deadlock state) during which the good thing never occurs. Fairness, which is concerned with a fair resolution of non-determinism, is often necessary and important to prove liveness properties. Fairness is an abstraction of the fair scheduler in a multi-threaded programming environment or the relative speed of the processors in distributed systems. Without fairness, verification of liveness properties often produces unrealistic loops during which one process or event is unfairly favored. It is important to systematically rule out those unfair counterexamples and utilize the computational resource to identify the real bugs.

The population protocol model has recently emerged as an elegant computation paradigm for describing mobile ad hoc networks [1]. A number of population protocols have been proposed and studied [6]. Fairness plays an important role in these protocols. For instance, it was shown that the self-stabilizing population protocols for the complete network graphs only works under *weak fairness*, whereas the algorithm for network rings only works under *strong global fairness* [2]. It has further been proved that with only *strong local fairness* or weaker, uniform self-stabilizing leader election in rings is impossible [2]. In order to verify (implementations of) those algorithms, model checking techniques must take the respective fairness into account. However, current practice of model checking is deficient with respect to fairness.

One way to apply existing model checkers for verification under fairness is to reformulate the property so that fairness become premises of the property. A liveness property ϕ is thus verified by showing the truth value of the following formula: $fairness\ assumptions \Rightarrow \phi$. This practice is deficient for two reasons. Firstly, a typical system may have multiple fairness constraints, whereas model checking is PSPACE-complete in the size of the formula. Secondly, partial order reduction which is one of

A. Bouajjani and O. Maler (Eds.): CAV 2009, LNCS 5643, pp. 709–714, 2009.

the successful reduction techniques for model checking becomes ineffective. Partial order reduction ignores/postpones invisible actions, whereas all actions/propositions in *fairness constraints* are visible and therefore cannot be ignored or postponed. An alternative method is to design specialized verification algorithms which take fairness into account while performing model checking. In this work, we present a toolkit named PAT (http://www.comp.nus.edu.sg/~pat) which checks linear temporal logic (LTL) properties against systems functioning under a variety of fairness (e.g., weak fairness, strong local/global fairness, process-level weak/strong fairness, etc.). A unified on-the-fly model checking algorithm is developed. PAT supports two different ways of applying fairness, one for ordinary users and the other for advanced users. Using PAT, we identified *previously unknown bugs* in the implementation of population protocols [2,6]. For experiments, we compare PAT and SPIN over a wide range of systems.

This work is related to research on categorizing and verifying fairness [2,8,9]. We investigate different forms of fairness and propose a verification algorithm (and a tool) which handles many of the fairness notions. In automata theory, fairness/liveness is often captured using accepting states. Our model checking algorithm is related to previous works on emptiness checking for Büchi automata and Streett automata [4]. In a way, our algorithm integrates the two algorithms presented in [3,7] and improves them in a number of aspects.

2 Background

Models in PAT are interpreted as labeled transition systems (LTS) implicitly. Let a be an action, which could be either an abstract event (e.g., a synchronization barrier if shared by multiple processes) or a data operation (e.g., a named sequential program). Let Σ be the set of all actions. An LTS is a 3-tuple $(S, init, \rightarrow)$ where S is a set of states, $init \in S$ is an initial state and $\rightarrow \subseteq S \times \Sigma \times S$ is a labeled transition relation.

For simplicity, we write $s \xrightarrow{a} s'$ to denote that (s, a, s') is a transition in $\rightarrow$. $enabled(s)$ is the set of enabled actions at s, i.e., a is in $enabled(s)$ if and only if there exist s' such that $s \xrightarrow{a} s'$. An execution is an infinite sequence of alternating states and actions $E = \langle s_0, a_0, s_1, a_1, \cdots \rangle$ where $s_0 = init$ and for all i such that $s_i \xrightarrow{a_i} s_{i+1}$. Without fairness constraints, a system may behave freely as long as it starts with an initial state and conforms to the transition relation. A fairness constraint restricts the set of system behaviors to only those fair ones. In the following, we focus on event-level fairness. Process-level fairness can be viewed as a special case of event-level fairness.

Definition 1 (Weak Fairness). *Let $E = \langle s_0, a_0, s_1, a_1, \cdots \rangle$ be an execution. E satisfies weak fairness, or is weak fair, iff for every action a, if a eventually becomes enabled forever in E, $a_i = a$ for infinitely many i, i.e., $\Diamond\Box\ a$ is enabled $\Rightarrow \Box\Diamond\ a$ is engaged.*

Weak fairness [8] states that if an action becomes enabled forever after some steps, then it must be engaged infinitely often. An equivalent formulation is that every computation should contain infinitely many positions at which a is disabled or has just been taken, known as justice condition [9]. Weak fairness has been well studied and verification under weak fairness has been supported to some extent [5].

Definition 2 (Strong Local Fairness). *Let $E = \langle s_0, a_0, s_1, a_1, \cdots \rangle$ be an execution. E satisfies strong local fairness iff for every action a, if a is infinitely often enabled, then $a = a_i$ for infinitely many i, i.e., $\Box\Diamond\ a\ is\ enabled \Rightarrow \Box\Diamond\ a\ is\ engaged$.*

Strong local fairness [8,9,2] states that if an action is infinitely often enabled, it must be infinitely often engaged. This type of fairness is useful in the analysis of systems that use semaphores, synchronous communication, and other special coordination primitives. Strong local fairness is stronger than weak fairness (since $\Diamond\Box\ a\ is\ enabled$ implies $\Box\Diamond\ a\ is\ enabled$). Verification under strong local fairness or compassion conditions has been discussed previously [3,4,7]. Nonetheless, there are few established tool support for formal verification under strong local fairness.

Definition 3 (Strong Global Fairness). *Let $E = \langle s_0, a_0, s_1, a_1, \cdots \rangle$ be an execution. E satisfies strong global fairness iff for every s, a, s' such that $s \xrightarrow{a} s'$, if $s = s_i$ for infinite many i, then $s_i = s$ and $a_i = a$ and $s_{i+1} = s'$ for infinitely many i.*

Strong global fairness [2] states that if a *step* (from s to s' by engaging in action a) can be taken infinitely often, then it must actually be taken infinitely often. Strong global fairness concerns about both actions and states. It can be shown by a simple argument that strong global fairness is stronger than strong local fairness. Strong global fairness requires that an infinitely enabled action must be taken infinitely often in *all* contexts, whereas strong local fairness only requires the enabled action to be taken in *one* context. A number of population protocols reply on strong global fairness, e.g., self-stabilizing leader election in ring networks [2] and token circulation in rings [1]. As far as the authors know, there are no previous work on verification under strong global fairness.

A number of other fairness notions have been discussed by various researchers. We remark that our approach can be extended to handle other kinds of fairness.

3 Verification under Fairness

Verification under fairness is to examine only fair executions of a given system and to decide whether certain property is true. Given a property ϕ, model checking is to search for an infinite fair execution which fails ϕ. In the following, we present PAT's unified algorithm to verify whether a system is feasible under different fairness constraints. A system is feasible if and only if there exists at least one infinite execution which satisfies the fairness constraints. Applied to the product of the system and (the negation of) the property, the algorithm can be easily extended to do model checking under fairness. The soundness of the algorithm and a discussion on its complexity is discusssed in [10].

Without loss of generality, we assume that a system contains only finite states. A system is feasible under fairness if and only if there exists a loop which satisfies the fairness. Feasibility checking is hence reduced to loop searching. In this work, we develop a unified algorithm, shown below, extending existing SCC-based algorithms to cope with fairness. Notice that nested DFS is not ideal as whether an execution is fair or not depends on the path instead of one state [5]. The algorithm is based on Tarjan's algorithm for identifying SCCs. It searches for fair strongly connected subgraph on-the-fly. The basic idea is to identify one SCC at a time and then check whether it is fair or not. If it is, the search is over. Otherwise, the SCC is partitioned into multiple smaller strongly connected subgraphs, which are then checked recursively one by one.

procedure $feasible(S, T)$
1. **while** there are states which have not been visited
2. **let** $scc := findSCC(S, T)$;
3. $pruned :=$ ***prune(scc,T)***;
4. **if** $pruned = scc$ **then**;
5. generate a feasible path; **return** true;
6. **endif**
7. **if** $feasible(pruned, T)$ **then return** $true$; **endif**
8. **endwhile**
9. **return** $false$;

Let S and T be a set of states and transitions. The main loop is from line 1 to 8. Line 2 invokes Tarjan's algorithm (implemented as $findSCC$) to identify one SCC within S and T. In order to perform on-the-fly verification, $findSCC$ is designed in such a way that if no S and T are given, it explores states and transitions on-the-fly until one SCC is identified. Function $prune$ (at line 5) is used to prune *bad states* from the SCC. Bad states are the reasons why the SCC is not fair. The intuition is that there may be a fair strongly connected subgraph in the remaining states. If the SCC satisfies the fairness assumption, no state is pruned and a fair loop (which traverses all states/transitions in the SCC) is generated and we conclude true at line 5. If some states have been pruned, a recursive call is made to check whether a fair strongly connected subgraph exists within the remaining states. The recursive call terminates in two ways, either a fair subgraph is found (at line 5) or all states are pruned (at line 9). If the recursive call returns false, there is no fair subgraph and we continue with another SCC until there is no state left.

By simply modifying function $prune$, the algorithm can be used to handle a variety of fairness. For instance, the following defines the function for weak fairness.

$$prune_{wf}(scc, T) = \begin{cases} S \text{ if } always(scc) \subseteq engaged(scc); \\ \varnothing \text{ otherwise.} \end{cases}$$

where $always(scc)$ is the set of events which are enabled at every state in scc and $engaged$ is the set of events labeling a transition between two states in scc. If there exists an event e which is always enabled but never engaged, by definition scc does not satisfy weak fairness. *If scc does not satisfy weak fairness, none of its subgraphs does.* As a result, either all states are pruned or none of them are. The following defines the function for strong local fairness.

$$prune_{slf}(scc, T) = \{s : scc \mid once(scc) \subseteq engaged(scc)\}$$

where $once(scc)$ contains events which are enabled at one or more states in scc. A state is pruned if and only if there is an event enabled at this state but never engaged in scc. By pruning the state, the event may become never enabled and therefore not required to be engaged. The following defines the function for strong global fairness.

$$prune_{sgf}(S, T) = \begin{cases} \varnothing \quad \text{if there exists } s : scc \text{ such that } s \xrightarrow{a} s' \text{ and } s' \notin scc; \\ scc \text{ otherwise.} \end{cases}$$

All states are pruned if there is a transition from a state in the SCC to a state not in the SCC. *If an SCC is not strong global fair, none of its subgraphs is (since the subgraph must contain a step to a pruned state and thus can not be strong global fair).*

Action Annotated Fairness. In PAT, we offer an alternative approach, which allows users to associate fairness to only part of the systems or associate different parts with different fairness constraints. The motivation is twofold. Firstly, previous approaches treat every action or state equally, i.e., fairness is applied to every action/state. In verification practice, it may be that only certain actions are meant to be fair. Our remedy is to allow users to associate fairness constraints with individual actions. The other motivation of action annotated fairness is that *for systems with action annotated fairness, it remains possible to apply partial order reduction to actions which are irrelevant to the fairness annotations*.

A number of different fairness may be used to annotate actions. Unconditional action fairness is written as $f(a)$. An execution of the system is fair if and only if a occurs infinitely often. It may be used to annotate actions which are known to occur periodically. For instance, a discrete clock may be modeled as: $Clock() = f(tick)\{x = x + 1\} \rightarrow Clock()$ where x is a discrete clock variable. By annotating $tick$ with unconditional fairness, we require that the clock must progress infinitely and the system disallows unrealistic *timelock*, i.e., execution of infinite actions which takes finite time. Weak (strong) action fairness is written as $wf(a)$ ($sf(a)$). An execution of the system is fair if and only if a occurs infinitely often given it is always (once) enabled. Unconditional action fairness does not depend on whether the action is enabled or not, and therefore, is stronger than weak/strong action fairness. The algorithm can then be applied to check systems with action annotated fairness with small modifications.

Model	Property	Size	Weak Fair			Strong Local Fair		Strong Global Fair	
			Result	PAT	SPIN	Result	PAT	Result	PAT
LE_C	$\Diamond\Box oneleader$	5	Yes	4.7	35.7	Yes	4.7	Yes	4.1
LE_C	$\Diamond\Box oneleader$	6	Yes	26.7	229	Yes	26.7	Yes	23.5
LE_C	$\Diamond\Box oneleader$	7	Yes	152.2	1190	Yes	152.4	Yes	137.9
LE_C	$\Diamond\Box oneleader$	8	Yes	726.6	5720	Yes	739.0	Yes	673.1
LE_OR	$\Diamond\Box oneleader$	3	No	0.2	0.3	No	0.2	Yes	11.8
LE_OR	$\Diamond\Box oneleader$	5	No	1.3	8.7	No	1.8	–	–
LE_R	$\Diamond\Box oneleader$	4	No	0.3	< 0.1	No	0.7	Yes	19.5
LE_R	$\Diamond\Box oneleader$	5	No	0.8	< 0.1	No	2.7	Yes	299.0
LE_R	$\Diamond\Box oneleader$	6	No	1.8	0.2	No	4.6	–	–
TC_R	$\Diamond\Box onetoken$	5	No	< 0.1	< 0.1	No	< 0.1	Yes	0.6
TC_R	$\Diamond\Box onetoken$	7	No	0.2	0.1	No	0.2	Yes	13.7
TC_R	$\Diamond\Box onetoken$	9	No	0.4	0.2	No	0.4	Yes	640.2
$peterson$	bounded bypass	3	Yes	0.1	1.25	Yes	0.1	Yes	0.1
$peterson$	bounded bypass	4	Yes	1.7	> 671	Yes	1.8	Yes	2.4
$peterson$	bounded bypass	5	Yes	58.9	–	Yes	63.7	Yes	75.4

Experiments. In the following, we show PAT's capability and efficiency over a range of systems where fairness is necessary. The following data are obtained by executing SPIN 4.3 and PAT 2.2 with Core 2 CPU 6600 at 2.40GHz and 2GB RAM.

The models include recently proposed self-stabilizing leader election protocols [6], e.g., for complete networks (LE_C), odd sized rings (LE_OR), and network rings (LE_R), token circulation for network rings (TC_R), and Peterson's algorithm for mutual exclusion. All models, with configurable parameters, are embedded in the PAT

package. Because of the difference between process-level weak fairness and weak fairness, the models are manually twisted in order to compare PAT with SPIN fairly (refer to [10] for details). SPIN has no support for strong local or global fairness. The only way to perform verification under strong local/global fairness in SPIN is to encode the fairness constraints as part of the property. However, even for a network of 3 nodes, SPIN needs significant amount of time to construct the (very large) Büchi automata, which makes it infeasible for such purpose.

In summary, PAT complements existing model checkers with the improvement in terms of the performance and ability to handle different forms of fairness. We remark that fairness does play an important role in these models. All of the algorithms fail to satisfy the property without fairness. Model LE_C and $peterson$ require at least weak fairness, whereas the rest of the algorithms require strong global fairness. It is thus important to be able to verify systems under strong local/global fairness. Notice that TC_R satisfies the property for a network of size 3 under weak fairness. There is, however, a counterexample for a network with more nodes. The reason is that a particular sequence of message exchange which satisfies weak fairness but fails the property needs the participation of at least 4 network nodes. This suggests that our approach has its practical values. *We highlight that previously unknown bugs in implementation of LE_OR [6] have been revealed using PAT.* We translated the algorithms *without* knowing how it works and generated a counterexample within seconds. SPIN is infeasible for this task because the algorithm requires strong global fairness [6].

References

1. Angluin, D., Aspnes, J., Fischer, M.J., Jiang, H.: Self-stabilizing Population Protocols. In: Anderson, J.H., Prencipe, G., Wattenhofer, R. (eds.) OPODIS 2005. LNCS, vol. 3974, pp. 103–117. Springer, Heidelberg (2006)
2. Fischer, M.J., Jiang, H.: Self-stabilizing Leader Election in Networks of Finite-state Anonymous Agents. In: Shvartsman, M.M.A.A. (ed.) OPODIS 2006. LNCS, vol. 4305, pp. 395–409. Springer, Heidelberg (2006)
3. Geldenhuys, J., Valmari, A.: More efficient on-the-fly LTL verification with Tarjan's algorithm. Theoritical Computer Science 345(1), 60–82 (2005)
4. Henzinger, M.R., Telle, J.A.: Faster Algorithms for the Nonemptiness of Streett Automata and for Communication Protocol Pruning. In: Karlsson, R., Lingas, A. (eds.) SWAT 1996. LNCS, vol. 1097, pp. 16–27. Springer, Heidelberg (1996)
5. Holzmann, G.J.: The SPIN Model Checker: Primer and Reference Manual. Addison Wesley, Reading (2003)
6. Jiang, H.: Distributed Systems of Simple Interacting Agents. Ph.D thesis, Yale Univ. (2007)
7. Kesten, Y., Pnueli, A., Raviv, L., Shahar, E.: Model Checking with Strong Fairness. Formal Methods and System Design 28(1), 57–84 (2006)
8. Lamport, L.: Proving the Correctness of Multiprocess Programs. IEEE Transactions on Software Engineering 3(2), 125–143 (1977)
9. Lehmann, D.J., Pnueli, A., Stavi, J.: Impartiality, Justice and Fairness: The Ethics of Concurrent Termination. In: Even, S., Kariv, O. (eds.) ICALP 1981. LNCS, vol. 115, pp. 264–277. Springer, Heidelberg (1981)
10. Sun, J., Liu, Y., Dong, J.S., Pang, J.: Towards a Toolkit for Flexible and Efficient Verification under Fairness. Technical Report TRB2/09, National Univ. of Singapore (December 2008)