EPTL - A Temporal Logic for Weakly Consistent Systems (Short Paper)

Mathias Weber, Annette Bieniusa, Arnd Poetzsch-Heffter

EPTL - A Temporal Logic for Weakly Consistent Systems (Short Paper)

Mathias Weber, Annette Bieniusa, and Arnd Poetzsch-Heffter

University of Kaiserslautern, Kaiserslautern, Germany
{m_weber,bieniusa,poetzsch}@cs.uni-kl.de

Abstract. The high availability and scalability of weakly-consistent system attracts system designers. Yet, writing correct application code for this type of systems is difficult; even how to specify the intended behavior of such systems is still an open question. There has not been established any standard method to specify the intended dynamic behavior of a weakly consistent system.
In this paper, we present a event-based parallel temporal logic (EPTL), that is tailored to specify properties of weakly consistent systems. In contrast to LTL and CTL, EPTL takes into account that operations of weakly consistent systems are in many cases not serializable and have to be treated respectively to capture their behavior. We embed our temporal logic in Isabelle/HOL and can thereby leverage strong semi-automatic proving capabilities.

1 Introduction

To improve availability and fault tolerance, information systems are often replicated to several nodes and globally distributed. In such system scenarios, designers face a trade-off between availability, fault tolerance, and consistency. To achieve high availability, designers might weaken the consistency constraints between the nodes. In weakly consistent systems, we might refrain from making the objects consistent after each operation. For example, the replicated state might consist of several objects and communication is done via asynchronous message passing.

In such systems with weak consistency semantics, concurrent modifications of a replicated object can lead to a divergent system state as the order in which updates are applied can differ among the nodes. To avoid the divergence, these update conflicts need to be resolved e.g. using CRDTs [9]. The main idea of CRDTs is to leveraging mathematical properties of the data structure and its operations to automatically solve conflicts due to concurrent modifications of the replicated object state.

The standard notion of time as being linear is known to not work well in weakly consistent systems[6]. Instead of assuming linear time, we follow Burckhardt et al. [2] in representing time as a partial order on the events in the system.

The topic of specifying weakly consistent systems is an open research question. LTL[8] is a classical specification language for dynamic properties of systems. It is widely used to specify properties of reactive systems. LTL is known for formulas which are easy to understand as well as its formal foundation. As we will show in Section 2, it can be difficult to capture the concurrent nature and asynchronous communication typical for weakly consistent systems in LTL (and CTL).

We want to decouple the specification of the behavior of the system from the behavior of the data types and want to enable to choose among different implementations based on the required properties described in the system specification. Our focus is on the understandability of the specification as well as a solid formal foundation.

Abstract executions Our goal is to have a specification language for properties of weakly consistent systems. The specification should be independent of the conflict resolution strategy used in the concrete implementation because this strategy partially depends on the required properties.

Weakly consistent systems are composed of multiple processes. Instead of sharing the state directly and protecting concurrent accesses using locks, each process obtains a replica of the shared object and solely interacts with this object. The values of the replicas are synchronized by asynchronously distributing the operations to all replicas. A typical data structure used in such systems is a *multi-value register (MVR)*. This datatype ensures that all written values of concurrent write operations are visible to subsequent read operations. The `put` operations allows to assign a new value to the register, the `get` operation allows to access the current state. Since the result of the `get` operation can consist of multiple concurrently written values, the operation returns a set of values. This means that if concurrently we have an operation writing the value 1 and one operation writing the value 2, the value of the register after synchronization of the operations is the set $\{1, 2\}$. Note that this property of multi-value registers usually leads to non-serializable system traces.

When formally specifying the semantics of the multi-value register, we want to abstract away from details concerning communication and process structure. Following Burckhardt et al. [2], we model the execution of a weakly consistent system as an abstract execution. An abstract execution A consists of a set of events E and a visibility relation $vis \subseteq E \times E$. The set E denotes the events representing the execution of operations on different nodes of the distributed weakly consistent system. The events have a unique identity and carry the metadata about the object and operation executed on it as well as information relevant for the specific use case (e.g. the subject executing the operation). The vis relation models the dependency between events. For two events e_1 and e_2, if $(e_1, e_2) \in vis$, then e_1 can influence the effect of e_2. The local order of events for each process is usually included in the visibility relation. To capture causality, vis must be irreflexive, transitive and antisymmetric. This corresponds to causal visibility. In addition, the visibility relation needs to be well-founded, so we can talk about the next events in the execution. The relation can also be depicted in an *event*

graph where the nodes of the graph are the events and the edges represent the visibility relation. Transitive edges are left out for readability.

We annotate the nodes of event graphs with *operation expressions* as follows: $op(p_1, \ldots, p_n)$ describes that an event e represents an execution of operation op with parameters p_1 to p_n. If the returned value is relevant, we denote it as $op(p_1, \ldots, p_n) \Rightarrow retval$ where $op(p_1, \ldots, p_n)$ is defined as above and $retval$ represents the returned value.

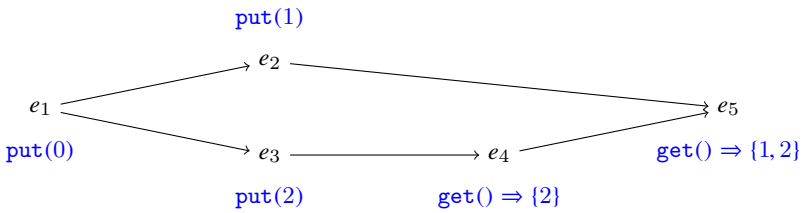

Fig. 1. Event graph of a multi-value register.

In form of an event graph, the example for the multi-value register can be depicted as in Figure 1. Event e_1 corresponds to an initial `put` operation, which assigns the single value 0. The `put`(1) operation of e_2 happens concurrently with another operation, `put`(2) of e_3, that also modifies the state of the register. Both events are visible to event e_5 associated to the `get` operation which yields the set $\{1, 2\}$ as result. As the example shows, this abstract execution is only concerned with the partial order of events with respect to the visibility relation; the event graph abstracts away from the details of a specific implementation (e.g. which process executes an operation or how operations are distributed to the other process).

The paper makes the following contributions: 1) We show why current temporal logics are not suitable to specify the intended behavior of weakly consistent systems. 2) Our temporal logic called event-based parallel temporal logic (EPTL) based on an abstract execution of the system allows to express properties on the global partial order of the events of the system and takes into account the non-serializability of operations. 3) We have proven laws that allow to rewrite EPTL formulas while retaining the semantics. EPTL is modeled in Isabelle/HOL and all laws are formally verified.

2 Event-based parallel temporal logic (EPTL)

In this section we present a new variant of temporal logic, namely event-based parallel temporal logic (EPTL).

Why LTL and CTL are not suitable Traditionally, linear time logic[8] (LTL) is interpreted on Kripke structures representing the reachable states of the system. This has two implications: (1) Time is usually seen as linear thereby totally ordering the events of the system and (2) LTL formulas specify properties of the states of the system, not the events. The leads to problems when trying to use LTL to specify properties of weakly consistent systems.

If we regard the serializations of abstract executions like the one in Figure 1 as independent event graphs, none of the executions would yield a result for a `get` operation that consists of more than one value since no operation would happen concurrently. We would need to encode the temporal information of the original abstract execution into the possible systems states. But this does not scale well and would typically require knowledge of the implementation of the replicated data type. Since the goal is to have a specification logic for the intended behavior of the system, this approach is not an option.

Computation tree logic[3] (CTL) allows for branching time, which solves parts of the issues discussed before. But in weakly consistent systems, the order of events forms a directed acyclic graph (DAG). Allowing to express that multiple different events can be successor of a single event and taking copies of future events is not an option. We would still loose the information that events can have happened concurrently in the past. The extended version [11] includes more details.

Syntax and Semantics Instead of being based on possible states of the system, EPTL is directly based on events following many previous works [1, 4, 5, 10]. For an abstract execution $A = (E, \mathsf{vis})$, we define the partial order $e_1 \leq_A e_2 \equiv e_1 = e_2 \vee (e_1, e_2) \in \mathsf{vis}$. When A is clear from the context, we simply write $e_1 \leq e_2$. The satisfaction relation $(A, e) \models \varphi$ is defined recursively over the structure of the formula as follows:

$$
\begin{array}{ll}
(A,e) \models Q & \text{iff } Q[I](e) \text{ for variable interpretation } I \\
(A,e) \models \neg\varphi & \text{iff } (A,e) \not\models \varphi \\
(A,e) \models (\varphi_1 \vee \varphi_2) & \text{iff } (A,e) \models \varphi_1 \text{ or } (A,e) \models \varphi_2 \\
(A,e) \models EX\varphi & \text{iff } \exists e_1.e < e_1 \text{ and } e_1 \text{ is a minimum wrt } < \text{ and } (A,e_1) \models \varphi \\
(A,e) \models AX\varphi & \text{iff } \forall e_1.e < e_1 \text{ if } e_1 \text{ is a minimum wrt } <, \text{ then } (A,e_1) \models \varphi \\
(A,e) \models (\varphi\ U\ \psi) & \text{iff } \exists e_1.e \leq e_1 \text{ such that } (A,e_1) \models \psi) \text{ and} \\
 & \quad \forall e_3.e \leq e_3 \text{ such that } (A,e_3) \not\models \varphi \text{ exists } e_2 \text{ such that} \\
 & \quad\quad e \leq e_2 \text{ and } e_2 \leq e_3 \text{ and } (A,e_2) \models \psi
\end{array}
$$

An interpretation I assigns values to all free variables occurring in an EPTL formula. $Q[I]$ stands for the proposition Q in which all free variables are replaced by their interpretation according to I. An EPTL formula φ is said to be valid if $(A, e) \models \varphi$ for all interpretations I. An abstract execution A satisfies an EPTL property φ, $A \models \varphi$, if all starting events of the abstract execution satisfy φ. The starting events of an abstract execution A are all events that are minimal with respect to the partial order $\leq_A$ i.e. they have no predecessor events.

The logical operators $\wedge$ and $\Rightarrow$ and the remaining temporal logic operators can be defined as usual. The main difference to LTL is that we have two different step operators EX and AX and a different semantics for the until operator U which is tailored to weakly consistent systems. Because the events in the system are ordered using a partial order, the next step is no longer unambiguous. Because of branches of concurrent events, a step might address multiple subsequent events. We want to have the possibility to address either at least one (EX) or all (AX) events that happen immediately after the current event. Also, the semantics of the until operator U has to be adapted to the partial order.

The definition of the until operation is stronger than in previous work [1, 4, 10] to be able to express strong properties about weakly consistent systems like the correctness of access control. Since this is a safety-critical question, we need a specification that is easy to understand and at the same time has a strong semantics on the execution of such a weakly consistent application. In general, access control is about specifying which operations are permitted to be executed by some subject or user on some object in the system. In a simple access control system we consider three types of operations: $\texttt{grant}(op, s, o)$ gives subject s the right to perform operation op on object o. $\texttt{revoke}(op, s, o)$ takes away the right of subject s to perform operation op on object o. $\texttt{exec}(op, s, o)$ represents the execution of operation op performed by subject s on object o. Corresponding propositions (e.g. $\texttt{grant}_P(op, s, o)$) are true for an event e if e represents the execution of the corresponding operation with the given parameters (e.g. $\texttt{grant}(op, s, o)$).

Based on the given operations, we can define the properties we require from our simple access control system. We want to start with a default policy that initially no user has the right to execute any operation on the system until an administrative user grants this right to him/her. To simplify the example, we do not consider the details of rights to perform grant and revoke operations and assume that there is some administrative user in the system that has the right to perform these operations. Using the weak version of until defined as $(A, e) \models \varphi \; W \; \psi$ iff $(A, e) \models G\varphi \vee (\varphi \; U \; \psi)$, we can define the initial policy by the following property:

$$A \models \neg\texttt{exec}_P(op, s, o) \; W \; \texttt{grant}_P(op, s, o)$$

The dependency between grant and revoke should work like this: Whenever the right of a subject is revoked, this operation should not be executed until a subsequent grant allows the operation again. This can be specified in EPTL in the following way:

$$A \models G(\texttt{revoke}_P(op, s, o) \Rightarrow AX(\neg\texttt{exec}_P(op, s, o) \; W \; \texttt{grant}_P(op, s, o)))$$

This property both models the semantics of the revoke and grant operations. A grant operation allows an operation that was previously revoked and a subsequent revoke operation disables the operation for the specified user again.

We see that the specifications are both readable and understandable as well as short. The strong semantics of the until operator ensures that revoking the right of a user disallows the operation on all future concurrent paths in the event graph.

Laws Most of the laws of LTL can be shown to also hold in EPTL. Some implications like the distributivity of the conjunction and disjunction are only one-directional. The most important exception to the LTL laws is the induction formula for the until operator, which does not hold in EPTL.

$$\varphi \; U \; \psi \not\equiv \psi \vee (\varphi \wedge X(\varphi \; U \; \psi))$$

This makes reasoning in EPTL inconvenient. But since EPTL is mainly intended as a specification logic for the intended behavior of the system which translates to properties on abstract executions, this restriction is not a big issue.

Proofs and Extended Version We have modeled EPTL in the theorem prover Isabelle/HOL. All laws of EPTL are formalized and verified in the interactive theorem prover and are used by the tool to simplify formulas. Even though we did not yet find an efficient automatic checking procedure for EPTL, the proofs can be done in semi-automatic fashion in HOL. Together with the strong automation of Isabelle/HOL this should make for a comfortable environment in which to show that the presented model is suitable to implement access control.

An extended version of this paper [11] includes the proven laws and extended examples.

3 Related Work

Alur et al. [1] presented a global partial order logic called ISTL. Same as we, they do not restrict the view on the system to the state sequence observed by a local process. The logic is based on a partially ordered set of local states which can also be seen as a branching structure. This branching structure represents all possible sequences of global states that may be derived from the partial order. This state based approach makes it unsuitable for reasoning about weakly consistent systems. As described in Section 2, encoding the events and the conflict resolution strategy into a state requires knowledge about the implementation of the conflict resolution strategy. Since the concrete implementation has to be abstracted from in the specification of the behavior of a weakly consistent system, ISTL is not suitable as a specification language for weakly consistent systems.

The other line of research about partial order semantics uses Mazurkiewicz traces [7]. The base for these traces is a finite set of actions, which can be seen as state transformations of resources of the system under investigation. Two actions are independent if they act on disjoint set of resources. Only independent actions are allowed to be performed concurrently. This restriction is the reason why Mazurkiewicz traces cannot be used to reason about weakly consistent systems in the given form. In these considered systems, the resources are replicated

objects where each process performs operations on its copy. When looking at these operation from a global view, they all change the same shared object. In this sense, the operations are not independent, even though they are possibly performed concurrently. It is not obvious how to apply Mazurkiewicz traces to weakly consistent systems.

4 Conclusion

We presented the new temporal logic EPTL that is tailored to specify properties of weakly consistent systems. As the example of access control shows, it allows for a concise and readable, yet machine-checkable specification. The complete logic is modeled in Isabelle/HOL and all laws are verified using the theorem prover. All theory files are available under `https://softech-git.informatik.uni-kl.de/mweber/EPTL/tree/master`.

References

1. Alur, R., McMillan, K., Peled, D.: Deciding Global Partial-Order Properties. Formal Methods in System Design 26(1), 7–25 (2005)
2. Burckhardt, S., Gotsman, A., Yang, H., Zawirski, M.: Replicated data types: Specification, verification, optimality. In: Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. pp. 271–284. POPL '14, ACM, New York, NY, USA (2014)
3. Clarke, E.M., Emerson, E.A.: Design and synthesis of synchronization skeletons using branching time temporal logic. In: Logics of Programs. pp. 52–71. Springer, Berlin, Heidelberg (1981)
4. Diekert, V., Gastin, P.: Pure future local temporal logics are expressively complete for mazurkiewicz traces. Inf. Comput. 204(11), 1597–1619 (Nov 2006)
5. Havelund, K., Rosu, G.: Testing linear temporal logic formulae on finite execution traces. Tech. rep., RIACS (2001)
6. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Commun. ACM 21(7), 558–565 (Jul 1978)
7. Mazurkiewicz, A.: Concurrent Program Schemes and their Interpretations. DAIMI Report Series 6(78) (1977)
8. Pnueli, A.: The temporal logic of programs. In: , 18th Annual Symposium on Foundations of Computer Science, 1977. pp. 46–57 (1977)
9. Shapiro, M., Preguiça, N.M., Baquero, C., Zawirski, M.: Conflict-free replicated data types. In: Défago, X., Petit, F., Villain, V. (eds.) Stabilization, Safety, and Security of Distributed Systems - 13th International Symposium, SSS 2011, Grenoble, France, October 10-12, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6976, pp. 386–400. Springer (2011)
10. Thiagarajan, P.S., Walukiewicz, I.: An Expressively Complete Linear Time Temporal Logic for Mazurkiewicz Traces. Information and Computation 179(2), 230–249 (2002)
11. Weber, M., Bieniusa, A., Poetzsch-Heffter, A.: EPTL - A Temporal Logic for Weakly Consistent Systems abs/1704.05320 (2017), `https://arxiv.org/abs/1704.05320`