# Sound Dynamic Deadlock Prediction in Linear Time

* Dynamic Predictive Analyses work by observing concurrent executions and reason about alternative interleavings that can witness concurrency bugs
* Ausgangspunkt des Papers: Kein Deadlock Predictor erfüllt aktuell die Anforderungen Soundness, Hohe Präzision und Effizienz
* Im Paper wird gezeigt, dass Sound and complete Deadlock Prediction unlösbar ist (im Allgemeinen)
* Auch der scheinbar einfachere Task die Anwesenheit potenzieller Deadlocks herauszufinden ist ebenso unlösbar
* Hauptmehrwert des Papers ist Einführung einer neuen Klasse von predictable Deadlocks (sync-preserving deadlocks)
* Sync-preserving deadlocks sind Deadlocks, die durch das Verändern der Ausführungsreihenfolge bei gleichzeitiger Erhaltung der Ausführungsreihenfolge von kritischen und konfliktbehafteten Abschnitten vorhergesagt werden können
* Es werden zwei Algorithmen basierend auf diesem Gedanken vorgestellt: SPDOffline (detektiert alle sync-preserving deadlocks), SPDOnline vorhersagt alle sync-preserving deadlocks die zwei Threads in strictly online fashion involvieren
* SPDOffline ist in linearer Laufzeit
* SPDOnline ist in linearer Laufzeit und am besten für ein runtime monitoring setting gedacht

## 1 Introduction:

* Techniken der Deadlock Detection können in statisch und dynamisch unterteilt werden
* Statische Ansätze analysieren der Quellcode und können das Nichtvorhandensein von Deadlocks beweisen
* Statische Ansätze haben schlechte Performance und sagen oft false positives vorher; Grund dafür sind die zwei Dimensionen des Nicht-determinismus: Inputs und Scheduling
* Dynamische Ansätze analysieren die Ausführung und ermöglichen dadurch bessere Skalierbarkeit und wenige bis keine false positives
* Dynamische Ansätze können auch nicht alle Bugs finden, bieten aber statistische und Abdeckungsgarantien
* Dynamische Ansätze sind der Standard im finden von Concurrency Bugs
* Predictive Analysis ist ein Trend in der letzten Zeit, hier ist das Ziel nicht nur die eine Ausführung zu bewerten, sondern nach einer beobachteten Ausführung auch noch all die unterschiedlichedn Ausführungsmöglichkeiten durchzuspielen, die hätten auftreten können (und natürlich dann den Bug kundtun)
* Dynamische Ansätze werden oft als runtime monitor genutzt und müssen deswegen online operieren und Bugs kundtun, sobald diese auftreten
* Die meisten online Algorithmen erkennen nur deadlock patterns, welche aber zu false positives führen
* Synchronization-preserving reorderings ist Vorarbeit auf der die Lösung im Paper basiert
* Bei SPDOffline wird versucht abstract Deadlock Patterns zu finden und anschließend auf realisierbarkeit zu prüfen (lineare Zeit pro abstraktes Pattern)
* SPDOffline ist skalierbar, da die Anzahl der abstrakten Deadlock Patterns wohl wesentlich kleiner ist als die Anzahl der tatsächlichen Deadlock Patterns (wie in früheren Ansätzen ermittelt)
* Abstract deadlock patterns sollen Cluster von Deadlock patterns mit derselben Signatur sein; Die Signatur zeichnet sich durch die beteiligten Threads und Locks aus. Sind diese bei zwei deadlock patterns gleich gehören sie dem gleichen Cluster an

## 1.1 Synchronization-Preserving Deadlocks

* Deadlock Pattern in Fig 1a: in e2 und e8 kommt es zum Deadlock, da Thread 1 L1 hält und L2 reservieren will und Thread 2 L2 hält aber L1 reservieren will und kein gemeinsamer Lock beschützt diese Operation
* Deadlock Pattern notwendige aber unzureichende Bedingung für einen tatsächlichen Deadlock
* Ein Algorithmus der Sound ist muss herausfinden ob es zu einem Deadlock über einen Zeugen kommt
* Ein Zeuge ist eine Umordnung p von (einem Teil von) dem Trace, welcher aber wieder ein valider Trace ist
* Außerdem müssen e2 und e8 „locally enabled in their respective threads at the end of p” sein???
* Es ist unmöglich zu überprüfen, ob ein Deadlock Pattern tatsächlich realisiert werden kann (ich glaube das realisiert steht hier für „in der Praxis vorkommen“)
* In dem Paper wird sich darauf fokussiert, ob ein Deadlock Pattern einen sync-preserving deadlock formt (sync-preserving deadlocks sind eine Unterklasse von allen vorhersagbaren Deadlocks
* Wann ist ein deadlock pattern ein sync-preserving deadlock? Es muss in einem Sync-Preserving Reordering pSP eines Trace beobachtet werden
* Was ist ein Correct Reordering?
  + Wenn es ein Teil von dem Trace ist
  + Wenn die Ordnung der Operationen innerhalb eines Threads nicht verändert werden
  + Jeder Writer für jeden Reader beibehalten (also nicht weggelassen wird)
* Ein Reordering eines Trace ist dann Sync-Preserving wenn
  + Der Kontrollfluss der gleiche ist
  + Die gemeinsame Ordnung zweier kritischer Sektionen (auf denselben Lock) beibehalten wird
* Nach Abarbeiten des Reorderings wird der Rest der Operationen in den Threads ausgeführt

## Our Contributions

Complexity of Deadlock Prediction:

* Ein einziges Deadlock Pattern zu erkennen ist NP-Schwer
* Selbst bei nur 2 Threads ist das Problem bestenfalls in O(N²) zu lösen
* Erkennung ist schwerer als Vorhersage von Deadlocks
* Vorhersage von Deadlocks ein sehr hartnäckiges Problem

Sync-preserving Deadlock Prediction and Abstract Deadlock Patterns:

* SPDOnline kriegt Trace und findet alle sync-preserving Deadlocks mit 2 Threads in linearer Zeit (in der Praxis gibt es meistens nur zwischen 2 Threads einen Deadlock, weshalb der Algorithmus auf 2 Threads begrenzt wird – wodurch die lineare Laufzeit zu Stande kommt)
* Die Coverage(Abdeckung) leidet dadurch nur gering (Deadlocks mit 3 oder mehr Threads sind eher selten)
* SPDOffline kriegt Trace und findet alle sync-preserving Deadlocks mit beliebig vielen Threads
* SPDOffline hat 2 Phasen:
  + Erst findet er alle abstract deadlock patterns
  + Dann führt er SPDOnline auf jedes abstract deadlock pattern aus um zu entscheiden, ob ein Deadlock vorhanden ist
* Die Laufzeit von SPDOffline bleibt O(N), aber erhöht sich um einen Faktor proportional zur Anzahl der Abstract Deadlock Patterns im Lock Graph

Implementation and Evaluation:

* SPDOffline bringt vor allem starken Performancegewinn gegenüber bisheriger State-of-the-Art-Lösung „SeqCheck“
* SPDOnline erkennt viel mehr als bisher vorherrschende Technik des „DeadlockFuzzer“

## Preliminaries

Deadlock Patterns:

* Sequenz von Events, welche zu einem Deadlock führen
* Deadlock Pattern der größe k ist:
  + Sequenz D aus Events
  + Mit k unterschiedlichen Threads
  + Und k unterschiedlichen Locks
  + Sodass thread(event) gleich dem Thread ist, indem das Event läuft
  + Op(event) ist die Operation des Events, diese muss immer ein acq auf einen Lock sein

Dynamic predictive analysis and correct reorderings:

* Dynamische Analyse zielt darauf ab Bugs dadurch zu finden indem ein Trace eines nebenläufigen Programms observiert wird, oft ohne Zugriff auf den Code
* Dynamische Analyse skalierbar, aber Bugs anhand eines einzigen Trace zu finden bedeutet geringe Abdeckung von Szenarien
* Für bessere Abdeckung Predictive Dynamic Techniques
  + Vorhersage von Bugs in alternativen Ausführungen, welche vom ursprünglichen Trace abgeleitet werden können -> Correct Reorderings

Predictable Deadlocks:

* Ein Deadlock Pattern ist predictable wenn alle Events im Deadlock Pattern nach der Ausführung des Correct Reorderings passieren
* Ein Deadlock-Prediction-Algorithm ist sound, wenn für jeden Input Trace alle Deadlock Reports Predictable Deadlocks sind (keine false positives)
* Ein Deadlock-Prediction-Algorithm ist complete, wenn alle Predictable Deadlocks auch gefunden werden, (keine false negatives)

## The Complexity of Dynamic Deadlock Prediction

## Synchronization-Preserving Deadlocks and Their Prediction

* Sync-preserving Deadlocks können im Gegensatz zu beliebigen Deadlocks effizient erkannt werden
* Die Experimente später zeigen, dass die meisten Deadlocks sync-preserving Deadlocks sind -> Schnelle Detektion kommt mit sehr geringem Precision Loss in der Praxis

### 4.1 Synchronization-Preserving Deadlocks

* Definition Sync-preserving Correct Reordering: Ein korrektes Reordering p eines Trace d ist sync-preserving, wenn für jeden Lock und jede 2 Acq-Events e1 != e2 die Ordnung der beiden Events in p dieselbe ist wie die in dem Trace d
* Das heißt auch, dass andere kritische Sektionen komplett entfernt werden können(in Happens-Before partial order von Lamport mussten alle kritischen Sektionen integriert sein)
* Definition Sync-preserving Deadlocks: d ist ein Trace und D ist ein Deadlock Pattern bestehend aus Events, die den Deadlock hervorrufen; D ist ein sync-preserving deadlock von d, wenn es ein sync-preserving correct reordering p von d gibt, welches hervorruft, dass die Locks in D nach p d-freigegeben sind

### Characterizing Sync-Preserving Deadlocks

* Zwei Aufgabe sind zu erfüllen um ein Correct Reordering welches einen Deadlock witnessed zu finden:
  + Bestimmen welche Events im Correct Reordering vorhanden sein sollen
  + Identifizieren einer Gesamtordnung für die Events
  + Beides Hartnäckige Probleme
* Für Sync-preserving Deadlocks wird gezeigt, dass
  + Die Suche für ein Correct Reordering kann auf das Überprüfen, ob eine wohldefinierte Menge an Events nicht die Events im Deadlock Pattern enthält, reduziert werden
  + Diese Menge kann effizient konstruiert werden
* Definition Sync-preserving Closure:
  + Delta ist Trace
  + S ist Untermenge an Events aus Trace
  + Sync-preserving Closure von S wird durch SPClosure\_delta(S) repräsentiert
  + SPClosure\_delta(S) ist die kleinste Menge S‘, sodass
    - S die Untermenge von S‘ ist
    - Für jede e, e‘ aus den Events von delta gilt dass
      * e kommt nach der Thread Order vor e‘ ODER e = readfrom\_delta(e‘), also e ist Writer von Reader e‘, wenn e‘ aus S‘, dann e aus S‘
      * und für jeden Lock l und zwei individuelle Events e, e‘ aus S‘ mit op(e)= op(e‘) = acq(l), wenn nach Trace Order e < e‘ dann gibt es für e auch noch das dazugehörige Release-Event in S’

Ideen:

* Soundness als Begriff erklären (siehe ChatGPT), andere Quelle für diese Information finden

Fragen:

* Was ist der Unterschied zwischen einer kompletten Deadlock Prediction und die Anwesenheit potenzieller Deadlocks herauszufinden?
* Was ist eine strictly online fashion?
* Was ist das abstract deadlock pattern? (Steht vermutlich im Paper)
* Was soll „locally enabled in their respective threads at the end of p” bedeuten?
  + Bedeutet, dass sie am Ende des umgeordneten Trace frei sind
* Was soll “closed under the thread order and preserving the writer of each read” bedeuten?
  + Antwort: Innerhalb eines Threads darf die Reihenfolge der Operationen nicht verändert werden und jeder Writer soll für jeden dazugehörigen Read beibehalten werden