ExoDPOR: Exogenous Stateless Model Checking

Lars Tveito, Einar Broch Johnsen and Rudolf Schlatte

August 26, 2021

Department of Informatics, University of Oslo
{larstvei,einarj,rudi}@ifi.uio.no

Introduction

- We present ExoDPOR, a framework for exogenous stateless model checking

Introduction

- We present ExoDPOR, a framework for exogenous stateless model checking
  - Given a program and an initial state, it explores all non-equivalent executions

Introduction

- We present ExoDPOR, a framework for exogenous stateless model checking
  - Given a program and an initial state, it explores all non-equivalent executions
- ExoDPOR explores executions by invoking and controlling an external runtime using executable traces

Introduction

- We present ExoDPOR, a framework for exogenous stateless model checking
  - Given a program and an initial state, it explores all non-equivalent executions
- ExoDPOR explores executions by invoking and controlling an external runtime using executable traces
- We formalize requirements for the traces and relations that are communicated between ExoDPOR and the external runtime

Introduction

- We present ExoDPOR, a framework for exogenous stateless model checking
  - Given a program and an initial state, it explores all non-equivalent executions
- ExoDPOR explores executions by invoking and controlling an external runtime using executable traces
- We formalize requirements for the traces and relations that are communicated between ExoDPOR and the external runtime
- ExoDPOR can be instantiated for programming languages or systems supporting deterministic record and replay

Introduction

- We present ExoDPOR, a framework for exogenous stateless model checking
  - Given a program and an initial state, it explores all non-equivalent executions
- ExoDPOR explores executions by invoking and controlling an external runtime using executable traces
- We formalize requirements for the traces and relations that are communicated between ExoDPOR and the external runtime
- ExoDPOR can be instantiated for programming languages or systems supporting deterministic record and replay
- The search orchestrated by ExoDPOR can be parallelized in a straightforward manner and scales over multiple machines

Introduction

- We present ExoDPOR, a framework for exogenous stateless model checking
  - Given a program and an initial state, it explores all non-equivalent executions
- ExoDPOR explores executions by invoking and controlling an external runtime using executable traces
- We formalize requirements for the traces and relations that are communicated between ExoDPOR and the external runtime
- ExoDPOR can be instantiated for programming languages or systems supporting deterministic record and replay
- The search orchestrated by ExoDPOR can be parallelized in a straightforward manner and scales over multiple machines
- We have instantiated ExoDPOR for Real-Time ABS

Context

- Testing sequential programs is nice and reliable

Context

- Testing sequential programs is nice and reliable
    - Low risk of false positives

Context

- Testing sequential programs is nice and reliable
  - Low risk of false positives
- Testing concurrent programs is hard and unreliable

Context

- Testing sequential programs is nice and reliable
  - Low risk of false positives
- Testing concurrent programs is hard and unreliable
  - High risk of false positives

Context

- Testing sequential programs is nice and reliable
  - Low risk of false positives
- Testing concurrent programs is hard and unreliable
  - High risk of false positives
- Testing all executions of a concurrent program would be ideal

Context

- Testing sequential programs is nice and reliable
  - Low risk of false positives
- Testing concurrent programs is hard and unreliable
  - High risk of false positives
- Testing all executions of a concurrent program would be ideal
  - But is rarely feasible

Related Work

- Stateless model checkers are tools for systematically exploring the execution paths of a concurrent program

Related Work

- Stateless model checkers are tools for systematically exploring the execution paths of a concurrent program
- *Partial order reduction* is crucial for avoiding many redundant executions

Related Work

- Stateless model checkers are tools for systematically exploring the execution paths of a concurrent program
- *Partial order reduction* is crucial for avoiding many redundant executions
- *Dynamic partial order reduction* is a state-of-the-art algorithm for stateless model checking

Related Work

- Stateless model checkers are tools for systematically exploring the execution paths of a concurrent program
- *Partial order reduction* is crucial for avoiding many redundant executions
- *Dynamic partial order reduction* is a state-of-the-art algorithm for stateless model checking
  - Requires a runtime with backtracking

Related Work

- Stateless model checkers are tools for systematically exploring the execution paths of a concurrent program
- *Partial order reduction* is crucial for avoiding many redundant executions
- *Dynamic partial order reduction* is a state-of-the-art algorithm for stateless model checking
  - Requires a runtime with backtracking
  - DPOR algorithms are generally sequential and challenging to parallelize

ExoDPOR for Real-Time ABS

- The main feature of ExoDPOR is that it is decoupled from the runtime

ExoDPOR for Real-Time ABS

- The main feature of ExoDPOR is that it is decoupled from the runtime
- ExoDPOR allows us to benefit from the Erlang backend of Real-Time ABS

ExoDPOR for Real-Time ABS

- The main feature of ExoDPOR is that it is decoupled from the runtime
- ExoDPOR allows us to benefit from the Erlang backend of Real-Time ABS
- A change in the Erlang backend does not imply that a change is needed in the ExoDPOR instantiation

ExoDPOR for Real-Time ABS

- The main feature of ExoDPOR is that it is decoupled from the runtime
- ExoDPOR allows us to benefit from the Erlang backend of Real-Time ABS
- A change in the Erlang backend does not imply that a change is needed in the ExoDPOR instantiation
- Much easier to support the full language

LTS and Traces

- We assume a *labeled transition system* (LTS), $(\mathcal{S}, \mathcal{E}, \rightarrow)$, where $\mathcal{S}$ is a set of states, $\mathcal{E}$ a set of events, and $\rightarrow \subseteq \mathcal{S} \times \mathcal{E} \times \mathcal{S}$ the transition relation

LTS and Traces

- We assume a *labeled transition system* (LTS), $(\mathcal{S}, \mathcal{E}, \rightarrow)$, where $\mathcal{S}$ is a set of states, $\mathcal{E}$ a set of events, and $\rightarrow \subseteq \mathcal{S} \times \mathcal{E} \times \mathcal{S}$ the transition relation
- We allow *unlabeled transitions*

LTS and Traces

- We assume a *labeled transition system* (LTS), $(\mathcal{S}, \mathcal{E}, \rightarrow)$, where $\mathcal{S}$ is a set of states, $\mathcal{E}$ a set of events, and $\rightarrow \subseteq \mathcal{S} \times \mathcal{E} \times \mathcal{S}$ the transition relation
- We allow *unlabeled transitions*
- An *execution* is a sequence of transitions

$$\sigma_1 \xrightarrow{e_1} \sigma_2 \xrightarrow{e_2} \ldots \xrightarrow{e_n} \sigma_{n+1}$$

  such that all events are distinct and such that σ_{n+1} is a final state

LTS and Traces

- We assume a *labeled transition system* (LTS), $(\mathcal{S}, \mathcal{E}, \rightarrow)$, where $\mathcal{S}$ is a set of states, $\mathcal{E}$ a set of events, and $\rightarrow \subseteq \mathcal{S} \times \mathcal{E} \times \mathcal{S}$ the transition relation
- We allow *unlabeled transitions*
- An *execution* is a sequence of transitions

$$\sigma_1 \xrightarrow{e_1} \sigma_2 \xrightarrow{e_2} \ldots \xrightarrow{e_n} \sigma_{n+1}$$

  such that all events are distinct and such that σ_{n+1} is a final state
- We conventionally denote a final state by σ_ε

LTS and Traces

- We assume a *labeled transition system* (LTS), $(\mathcal{S}, \mathcal{E}, \rightarrow)$, where $\mathcal{S}$ is a set of states, $\mathcal{E}$ a set of events, and $\rightarrow \subseteq \mathcal{S} \times \mathcal{E} \times \mathcal{S}$ the transition relation
- We allow *unlabeled transitions*
- An *execution* is a sequence of transitions

$$\sigma_1 \xrightarrow{e_1} \sigma_2 \xrightarrow{e_2} \dots \xrightarrow{e_n} \sigma_{n+1}$$

  such that all events are distinct and such that σ_{n+1} is a final state
- We conventionally denote a final state by σ_ε
- An *execution trace* is a sequence of events, denoted $\tau = e_1 \cdot e_2 \cdots e_n$

Record & Replay semantics

$$\frac{\sigma \to \sigma'}{\langle \tau_{\bullet} \mid \tau_{\blacktriangleright} \rangle \triangleright \sigma \xrightarrow{\bullet/\blacktriangleright} \langle \tau_{\bullet} \mid \tau_{\blacktriangleright} \rangle \triangleright \sigma'} \quad (\textsc{Unlabeled Record \& Replay})$$

Record & Replay semantics

$$\frac{\sigma \to \sigma'}{\langle \tau_\bullet \mid \tau_\blacktriangleright \rangle \triangleright \sigma \xrightarrow{\bullet/\blacktriangleright} \langle \tau_\bullet \mid \tau_\blacktriangleright \rangle \triangleright \sigma'} \; (\textsc{Unlabeled Record \& Replay})$$

$$\frac{\sigma \xrightarrow{e} \sigma'}{\langle \tau_\bullet \mid \varepsilon \rangle \triangleright \sigma \xrightarrow{\bullet/\blacktriangleright} \langle \tau_\bullet \cdot e \mid \varepsilon \rangle \triangleright \sigma'} \; (\textsc{Labeled Record})$$

Record & Replay semantics

(Unlabeled Record & Replay)

$$\frac{\sigma \rightarrow \sigma'}{\langle \tau_{\bullet} \mid \tau_{\blacktriangleright} \rangle \triangleright \sigma \xrightarrow{\bullet/\blacktriangleright} \langle \tau_{\bullet} \mid \tau_{\blacktriangleright} \rangle \triangleright \sigma'}$$

(Labeled Record)

$$\frac{\sigma \xrightarrow{e} \sigma'}{\langle \tau_{\bullet} \mid \varepsilon \rangle \triangleright \sigma \xrightarrow{\bullet/\blacktriangleright} \langle \tau_{\bullet} \cdot e \mid \varepsilon \rangle \triangleright \sigma'}$$

(Labeled Record & Replay)

$$\frac{\sigma \xrightarrow{e} \sigma'}{\langle \tau_{\bullet} \mid e \cdot \tau_{\blacktriangleright} \rangle \triangleright \sigma \xrightarrow{\bullet/\blacktriangleright} \langle \tau_{\bullet} \cdot e \mid \tau_{\blacktriangleright} \rangle \triangleright \sigma'}$$

Relations over events

- We have the following relations over $\mathcal{E}$:

Relations over events

- We have the following relations over $\mathcal{E}$:
  $e_i \overset{MHB}{\Longrightarrow} e_j$ if the event e_i *must* happen before e_j in all feasible executions;

Relations over events

- We have the following relations over $\mathcal{E}$:

$e_i \stackrel{MHB}{\Longrightarrow} e_j$ if the event e_i *must* happen before e_j in all feasible executions;

$e_i \star e_j$ if the order of e_i and e_j *may* affect the result of an execution;

Relations over events

- We have the following relations over $\mathcal{E}$:

$e_i \overset{MHB}{\Longrightarrow} e_j$	if the event e_i *must* happen before e_j in all feasible executions;
$e_i \star e_j$	if the order of e_i and e_j *may* affect the result of an execution;
$e_i \overset{HB}{\Longrightarrow}_\tau e_j$	if e_i occurs before e_j in the trace τ and $e_i \overset{MHB}{\Longrightarrow} e_j$ or $e_i \star e_j$.
	We can derive $\overset{HB}{\Longrightarrow}_\tau$ from $\overset{MHB}{\Longrightarrow}$ and $\star$ for a given τ.

Relations over events

- We have the following relations over $\mathcal{E}$:

$e_i \stackrel{MHB}{\Longrightarrow} e_j$	if the event e_i *must* happen before e_j in all feasible executions;
$e_i \star e_j$	if the order of e_i and e_j *may* affect the result of an execution;
$e_i \stackrel{HB}{\Longrightarrow}_\tau e_j$	if e_i occurs before e_j in the trace τ and $e_i \stackrel{MHB}{\Longrightarrow} e_j$ or $e_i \star e_j$.
	We can derive $\stackrel{HB}{\Longrightarrow}_\tau$ from $\stackrel{MHB}{\Longrightarrow}$ and $\star$ for a given τ.

Relations over events

- We have the following relations over $\mathcal{E}$:

$e_i \overset{MHB}{\Longrightarrow} e_j$	if the event e_i *must* happen before e_j in all feasible executions;
$e_i \star e_j$	if the order of e_i and e_j *may* affect the result of an execution;
$e_i \overset{HB}{\Longrightarrow}_\tau e_j$	if e_i occurs before e_j in the trace τ and $e_i \overset{MHB}{\Longrightarrow} e_j$ or $e_i \star e_j$. We can derive $\overset{HB}{\Longrightarrow}_\tau$ from $\overset{MHB}{\Longrightarrow}$ and $\star$ for a given τ.

  Two traces τ_1 and τ_2 are *equivalent*, denoted $\tau_1 \simeq \tau_2$, if $\overset{HB}{\Longrightarrow}_{\tau_1} = \overset{HB}{\Longrightarrow}_{\tau_2}$.

Exogenous exploration

$$\frac{}{\langle exo : \{\tau\} \cup Seeds, \{w\} \cup W, \mathcal{S}s\rangle \rightarrow \langle exo : Seeds, W, \mathcal{S}s\rangle \ \langle w : \langle \varepsilon \mid \tau\rangle \triangleright \sigma_0\rangle} \text{(Explore-single-trace)}$$

Exogenous exploration

$$\text{(Explore-single-trace)}$$
$$\langle exo : \{\tau\} \cup Seeds, \{w\} \cup W, \mathcal{S}s\rangle \rightarrow \langle exo : Seeds, W, \mathcal{S}s\rangle\ \langle w : \langle \varepsilon \mid \tau\rangle \triangleright \sigma_0\rangle$$

$$\text{(Worker-progress)}$$
$$\frac{\langle \tau_\bullet \mid \tau_\blacktriangleright\rangle \triangleright \sigma \xrightarrow{\bullet/\blacktriangleright} \langle \tau'_\bullet \mid \tau'_\blacktriangleright\rangle \triangleright \sigma'}{\langle w : \langle \tau_\bullet \mid \tau_\blacktriangleright\rangle \triangleright \sigma\rangle \rightarrow \langle w : \langle \tau'_\bullet \mid \tau'_\blacktriangleright\rangle \triangleright \sigma'\rangle}$$

Exogenous exploration

(Explore-single-trace)

$$\langle exo : \{\tau\} \cup Seeds, \{w\} \cup W, \mathcal{S}s\rangle \rightarrow \langle exo : Seeds, W, \mathcal{S}s\rangle \ \langle w : \langle \varepsilon \mid \tau\rangle \triangleright \sigma_0\rangle$$

(Worker-progress)

$$\frac{\langle \tau_\bullet \mid \tau_\blacktriangleright\rangle \triangleright \sigma \xrightarrow{\bullet/\blacktriangleright} \langle \tau'_\bullet \mid \tau'_\blacktriangleright\rangle \triangleright \sigma'}{\langle w : \langle \tau_\bullet \mid \tau_\blacktriangleright\rangle \triangleright \sigma\rangle \rightarrow \langle w : \langle \tau'_\bullet \mid \tau'_\blacktriangleright\rangle \triangleright \sigma'\rangle}$$

(Update-with-explored-trace)

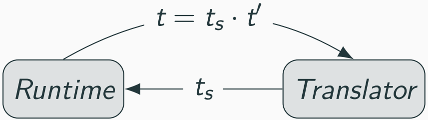
$$\frac{\mathcal{S}s' = addTrace(\mathcal{S}s, \tau) \quad Seeds' = Seeds \cup newSeeds(\mathcal{S}s', \tau)}{\langle exo : Seeds, W, \mathcal{S}s\rangle \ \langle w : \langle \tau \mid \varepsilon\rangle \triangleright \sigma_\varepsilon\rangle \rightarrow \langle exo : Seeds', \{w\} \cup W, \mathcal{S}s'\rangle}$$

Framework

Runtime

Framework

Framework



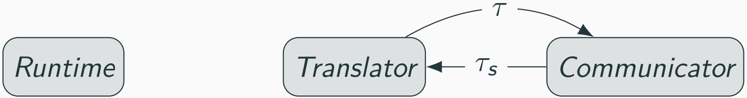
Framework

Runtime

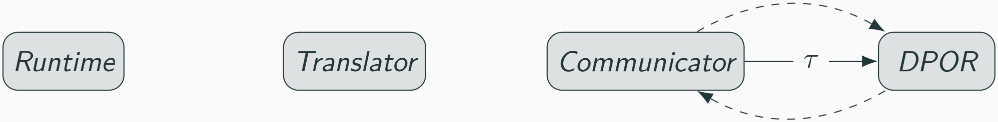
Translator

Communicator

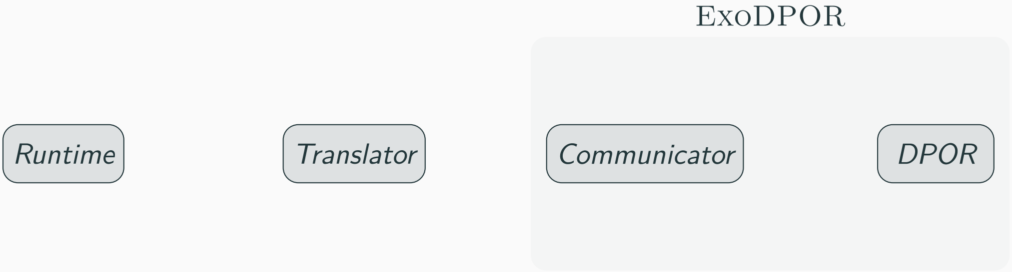
Framework



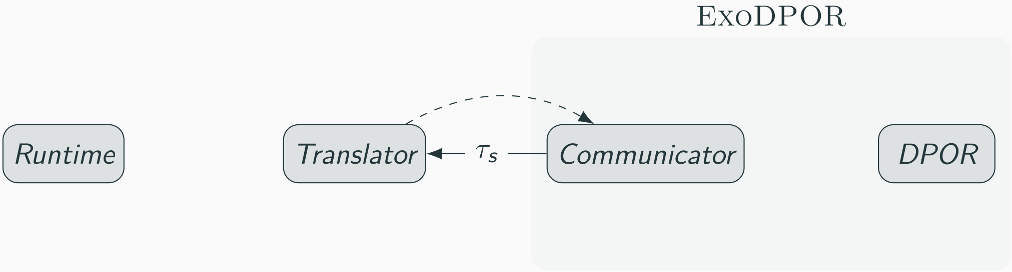
Framework



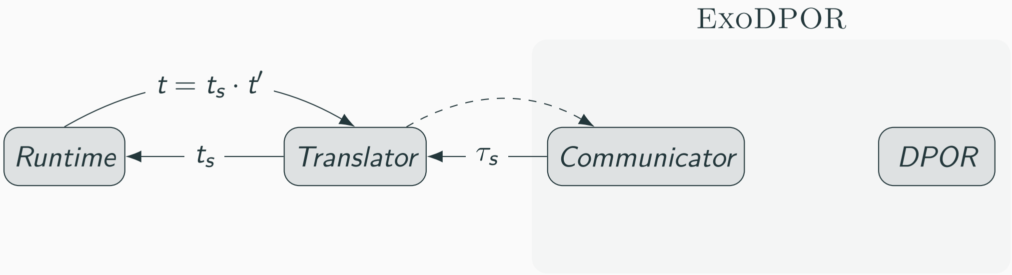
Framework



Framework

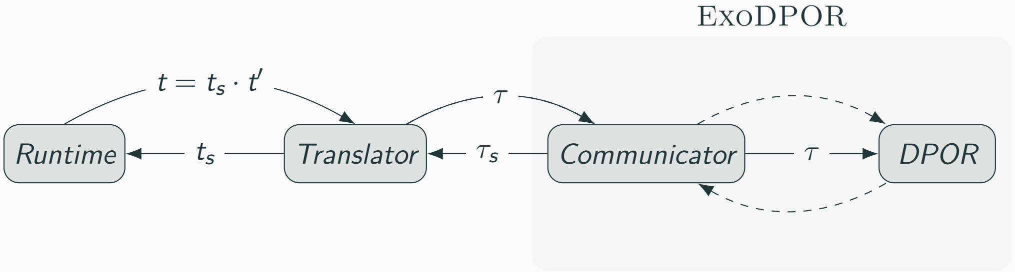


Framework



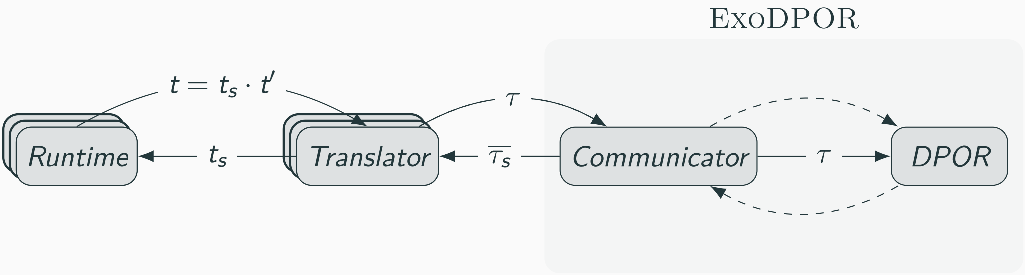
ExoDPOR
$t = t_s \cdot t'$
Runtime
t_s
Translator
τ_s
Communicator
DPOR

Framework



ExoDPOR
$t = t_s \cdot t'$
τ
Runtime
t_s
Translator
τ_s
Communicator
τ
DPOR

Framework



ExoDPOR
$t = t_s \cdot t'$
τ
Runtime
t_s
Translator
$\overline{\tau_s}$
Communicator
τ
DPOR

Current state

- We have a sound implementation of Source-DPOR

Current state

- We have a sound implementation of Source-DPOR
  - Major disadvantage with not being able to dynamically avoid redundant executions

Current state

- We have a sound implementation of Source-DPOR
  - Major disadvantage with not being able to dynamically avoid redundant executions
- We are working on an implementation of Optimal-DPOR (still containing bugs)

Current state

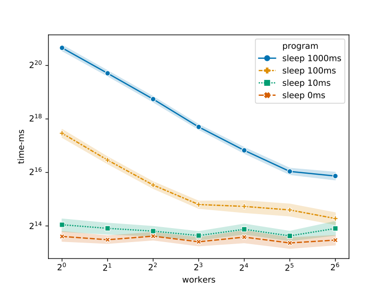
- We have a sound implementation of Source-DPOR
  - Major disadvantage with not being able to dynamically avoid redundant executions
- We are working on an implementation of Optimal-DPOR (still containing bugs)
  - Every seed trace is guaranteed to reach a non-equivalent execution

Current state

- We have a sound implementation of Source-DPOR
  - Major disadvantage with not being able to dynamically avoid redundant executions
- We are working on an implementation of Optimal-DPOR (still containing bugs)
  - Every seed trace is guaranteed to reach a non-equivalent execution
- We have instantiations for two toy languages and for Real Time ABS

Current state

- We have a sound implementation of Source-DPOR
  - Major disadvantage with not being able to dynamically avoid redundant executions
- We are working on an implementation of Optimal-DPOR (still containing bugs)
  - Every seed trace is guaranteed to reach a non-equivalent execution
- We have instantiations for two toy languages and for Real Time ABS
- Parallelization is simple and gives a linear speedup for long-running programs



Conclusion

- We have introduced ExoDPOR, a general framework for stateless model checking with state of the art DPOR-algorithms

Conclusion

- We have introduced ExoDPOR, a general framework for stateless model checking with state of the art DPOR-algorithms
- It has been instantiated for Real-Time ABS

Conclusion

- We have introduced ExoDPOR, a general framework for stateless model checking with state of the art DPOR-algorithms
- It has been instantiated for Real-Time ABS
- We believe an instantiation for ExoDPOR is significantly easier than to implement a model checker from scratch

Conclusion

- We have introduced ExoDPOR, a general framework for stateless model checking with state of the art DPOR-algorithms
- It has been instantiated for Real-Time ABS
- We believe an instantiation for ExoDPOR is significantly easier than to implement a model checker from scratch
- Experiments indicate that parallelization gives a linear speed-up for long-running programs