Deadlock Checking by Data Race Detection

Ka I Violet Pun, Martin Steffen, Volker Stolz

PMA Group, University of Oslo, Norway

Bad Honnef

6 Mai 2013

Overview

Goal

Find *potential* deadlocks in programs
. . . by detecting data races
. . . through *static analysis*

- Data race:
  - Simultaneous access to shared data with at least one write access
  - Shared data: mutable, unprotected
- Deadlock:
  - Multiple processes wait for shared resources in a cycle
  - Here: Locks

Overview

General approach:

- Reduce the problem of deadlock checking to race checking
- Instrument programs with accesses to additional global variables (race variables) at appropriate *program points*
- Programs with deadlocks
  $\Longrightarrow$ data race in the transformed one
- Also: no data race $\Longrightarrow$ deadlock freedom

(Assume the original programs are race free)

Calculus

- Concurrent language
- Higher-order
- Dynamic lock creation
- Reentrant locks
- Non-lexically scoped locks

$$
\begin{array}{rcl}
t & ::= & \mathtt{stop} \mid v \mid \mathtt{let}\ x{:}T = e\ \mathtt{in}\ t \\
e & ::= & t \mid v\ v \mid \mathtt{if}\ e\ \mathtt{then}\ e\ \mathtt{else}\ e \mid \\
  &   & \mathtt{spawn}\ t \mid \mathtt{new}\ L \mid v.\ \mathtt{lock} \mid v.\ \mathtt{unlock} \\
v & ::= & x \mid l \mid \mathtt{fn}\ x{:}T.t \mid \mathtt{fun}\ f{:}T.x{:}T.t
\end{array}
$$

Calculus

- Concurrent language
- Higher-order
- Dynamic lock creation
- Reentrant locks
- Non-lexically scoped locks

$$
\begin{array}{rcl}
t & ::= & \mathtt{stop} \mid v \mid \mathtt{let}\ x{:}T = e\ \mathtt{in}\ t \\
e & ::= & t \mid v\ v \mid \mathtt{if}\ e\ \mathtt{then}\ e\ \mathtt{else}\ e \mid \\
 & & \mathtt{spawn}\ t \mid \mathtt{new}\ L \mid v.\,\mathtt{lock} \mid v.\,\mathtt{unlock} \\
v & ::= & x \mid l \mid \mathtt{fn}\ x{:}T.t \mid \mathtt{fun}\ f{:}T.x{:}T.t
\end{array}
$$

Type and effect system

- Static analysis framed as a type and effect system
- Type:
  Captures *results* of the computations of a program
- Effect:
  Captures *behaviour* during the computations of a program

Type and effect system

- Uses *program points* π, to characterize locks according to their origin
- Tracks relative change to the lock count
- Captures *static program points* where deadlocks may manifest themselves
- Uses constraints to derive the *smallest* possible types and effects
- Context-sensitive analysis
  - Polymorphic types
  - Increase precision

Second lock point

- *Second lock point* *(slp)*
  - A *static* over-approximation of program points where deadlocks may manifest themselves
  - p holds π_1 and tries to take π_2

- The type and effect system works thread-locally
- Derives *slp* per thread

Abstract cycle Δ_C

- A cycle of locks
- A sequence of pairs $p_1 : \pi_1; \ldots p_n : \pi_n$
  - p_i: process identifiers
  - π_i: program points where locks are created
- Interpreted as: process p_1 has π_1 and wants π_2

Second lock point

Second lock point relative to Δ_C

Given Δ_C and $\bullet \vdash_p t_0 : \Delta$, t is a *static second lock point* if:

Second lock point

Second lock point relative to Δ_C

Given Δ_C and $\bullet \vdash_p t_0 : \Delta$, t is a *static second lock point* if:

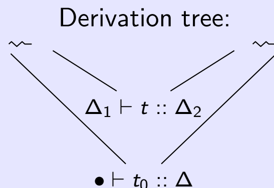
1. $t = \texttt{let}\ x{:}\mathrm{L}^{\{\ldots,\pi,\ldots\}} = v.\texttt{lock in}\ t'$.

Second lock point relative to Δ_C

Given Δ_C and $\bullet \vdash_p t_0 : \Delta$, t is a *static second lock point* if:

1. $t = \texttt{let}\ x{:}\mathrm{L}^{\{\ldots,\pi,\ldots\}} = v.\texttt{lock in}\ t'$.
2. $\Delta_1 \vdash_p t :: \Delta_2$ within $\bullet \vdash t_0 :: \Delta$.

Derivation tree:



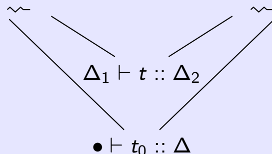
Second lock point

Second lock point relative to Δ_C

Given Δ_C and $\bullet \vdash_p t_0 : \Delta$, t is a *static second lock point* if:

1. $t = \texttt{let}\ x{:}\mathrm{L}^{\{\ldots,\pi,\ldots\}} = v.\texttt{lock in}\ t'$.
2. $\Delta_1 \vdash_p t :: \Delta_2$ within $\bullet \vdash t_0 :: \Delta$.
3. there exists π' s.t.

$$\pi' \in \Delta_1, \qquad \Delta_C \vdash p\ \text{has}\ \pi', \quad \text{and} \quad \Delta_C \vdash p\ \text{wants}\ \pi$$



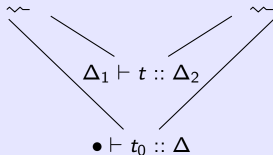
Second lock point

Second lock point relative to Δ_C

Given Δ_C and $\bullet \vdash_p t_0 : \Delta$, t is a *static second lock point* if:

1. $t = \texttt{let } x{:}L^{\{\ldots,\pi,\ldots\}} = v.\texttt{lock in } t'$.
2. $\Delta_1 \vdash_p t :: \Delta_2$ within $\bullet \vdash t_0 :: \Delta$.
3. there exists π' s.t.

$$\pi' \in \Delta_1, \qquad \Delta_C \vdash p \text{ has } \pi', \quad \text{and} \quad \Delta_C \vdash p \text{ wants } \pi$$



Second lock point

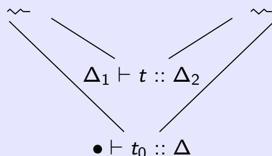
Second lock point relative to Δ_C

Given Δ_C and $\bullet \vdash_p t_0 : \Delta$, t is a *static second lock point* if:

1. $t = \texttt{let}\ x{:}\mathrm{L}^{\{\ldots,\pi,\ldots\}} = v.\texttt{lock in}\ t'$.
2. $\Delta_1 \vdash_p t :: \Delta_2$ within $\bullet \vdash t_0 :: \Delta$.
3. there exists π' s.t.

$$\pi' \in \Delta_1, \qquad \Delta_C \vdash p \text{ has } \pi', \quad \text{and} \quad \Delta_C \vdash p \text{ wants } \pi$$

Derivation tree:



Second lock point

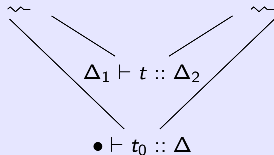
Second lock point relative to Δ_C

Given Δ_C and $\bullet \vdash_p t_0 : \Delta$, t is a *static second lock point* if:

1. $t = \texttt{let}\ x{:}\texttt{L}^{\{\ldots,\pi,\ldots\}} = v.\texttt{lock in}\ t'$.
2. $\Delta_1 \vdash_p t :: \Delta_2$ within $\bullet \vdash t_0 :: \Delta$.
3. there exists π' s.t.

$$\pi' \in \Delta_1, \quad \Delta_C \vdash p \text{ has } \pi', \quad \text{and} \quad \Delta_C \vdash p \text{ wants } \pi$$

Derivation tree:



Type and Effect System

Judgements:

$$\Gamma \vdash e : \hat{T}$$

Types and effects are described by:

Type and Effect System

Judgements:

$$\Gamma \vdash e : \hat{T}$$

Types and effects are described by:

$$\hat{T} \ ::= \ B \ | \ \mathtt{L}^r \ | \ \hat{T} \xrightarrow{\varphi} \hat{T} \ | \ \forall \vec{Y}{:}C.\hat{T} \quad \text{types}$$

Type and Effect System

Judgements:

$$\Gamma \vdash e : \hat{T}$$

Types and effects are described by:

$\hat{T}$	$::=$	$B \mid \mathtt{L}^{r} \mid \hat{T} \xrightarrow{\varphi} \hat{T} \mid \forall \vec{Y}{:}C.\hat{T}$	types
r	$::=$	$\{\pi\} \mid r \sqcup r \mid \varrho$	lock/label sets/data-flow information

Type and Effect System

Judgements:

$$\Gamma \vdash e : \hat{T} :: \varphi$$

Types and effects are described by:

$$\begin{array}{rcll} \hat{T} & ::= & B \mid \mathtt{L}^r \mid \hat{T} \xrightarrow{\varphi} \hat{T} \mid \forall \vec{Y}{:}C.\hat{T} & \text{types} \\ r & ::= & \{\pi\} \mid r \sqcup r \mid \varrho & \text{lock/label sets/data-flow information} \end{array}$$

Type and Effect System

Judgements:

$$\Gamma \vdash e : \hat{T} :: \varphi$$

Types and effects are described by:

$\hat{T}$	$::=$	$B \mid \mathrm{L}^r \mid \hat{T} \xrightarrow{\varphi} \hat{T} \mid \forall \vec{Y}{:}C.\hat{T}$	types
r	$::=$	$\{\pi\} \mid r \sqcup r \mid \varrho$	lock/label sets/data-flow information
φ	$::=$	$\Delta \rightarrow \Delta$	effects/pre- and post specification

Type and Effect System

Judgements:

$$\Gamma \vdash e : \hat{T} :: \varphi$$

Types and effects are described by:

$$\begin{array}{rcll}
\hat{T} & ::= & B \mid \mathrm{L}^r \mid \hat{T} \xrightarrow{\varphi} \hat{T} \mid \forall \vec{Y}{:}C.\hat{T} & \text{types} \\
r & ::= & \{\pi\} \mid r \sqcup r \mid \varrho & \text{lock/label sets/data-flow information} \\
\varphi & ::= & \Delta \to \Delta & \text{effects/pre- and post specification} \\
\Delta & ::= & \bullet \mid \Delta, r{:}n \mid X & \text{lock env./abstract state}
\end{array}$$

Type and Effect System

Judgements:

$$\Gamma \vdash e : \hat{T} :: \varphi; C$$

Types and effects are described by:

$\hat{T}$	$::=$	$B \mid \mathrm{L}^r \mid \hat{T} \xrightarrow{\varphi} \hat{T} \mid \forall \vec{Y}{:}C.\hat{T}$	types
r	$::=$	$\{\pi\} \mid r \sqcup r \mid \varrho$	lock/label sets/data-flow information
φ	$::=$	$\Delta \rightarrow \Delta$	effects/pre- and post specification
Δ	$::=$	$\bullet \mid \Delta, r{:}n \mid X$	lock env./abstract state
C	$::=$	$\emptyset \mid \varrho \sqsupseteq r, C \mid X \geq \Delta, C$	constraints

see: "Type and Effect Systems", Amtoft/Nielson/Nielson, 1999

Type and Effect System

$$\frac{\Gamma \vdash t : \hat{T} :: \bullet \rightarrow \Delta_2; C}{\Gamma \vdash \texttt{spawn}\ t : \texttt{Thread} :: \Delta_1 \rightarrow \Delta_1; C}\ \textsc{T-Spawn}$$

Type and Effect System

$$\frac{\Gamma \vdash t : \hat{T} :: \bullet \to \Delta_2; C}{\Gamma \vdash \texttt{spawn}\ t : \texttt{Thread} :: \Delta_1 \to \Delta_1; C} \text{ T-SPAWN}$$

$$\frac{}{\Gamma \vdash \texttt{new}_\pi\ \texttt{L} : \texttt{L}^{\{\pi\}} :: \Delta \to \Delta} \text{ T-NEWL}$$

Type and Effect System

$$\frac{\Gamma \vdash t : \hat{T} :: \bullet \to \Delta_2; C}{\Gamma \vdash \mathtt{spawn}\ t : \mathtt{Thread} :: \Delta_1 \to \Delta_1; C} \text{ T-SPAWN}$$

$$\frac{\varrho \textit{ fresh}}{\Gamma \vdash \mathtt{new}_\pi\ \mathtt{L} : \mathtt{L}^\varrho :: \Delta \to \Delta; \varrho \sqsupseteq \{\pi\}} \text{ T-NEWL}$$

$$\frac{\Gamma \vdash v : \mathtt{L}^\varrho :: \Delta \to \Delta}{\Gamma \vdash v.\ \mathtt{lock} : \mathtt{L}^\varrho :: \Delta \to \Delta \oplus (\varrho{:}1)} \text{ T-LOCK}$$

Type and Effect System

$$\frac{\Gamma \vdash t : \hat{T} :: \bullet \to \Delta_2 ; C}{\Gamma \vdash \texttt{spawn}\ t : \texttt{Thread} :: \Delta_1 \to \Delta_1 ; C}\ \textsc{T-Spawn}$$

$$\frac{\varrho\ \textit{fresh}}{\Gamma \vdash \texttt{new}_\pi\ \texttt{L} :\ \texttt{L}^\varrho :: \Delta \to \Delta ; \varrho \sqsupseteq \{\pi\}}\ \textsc{T-NewL}$$

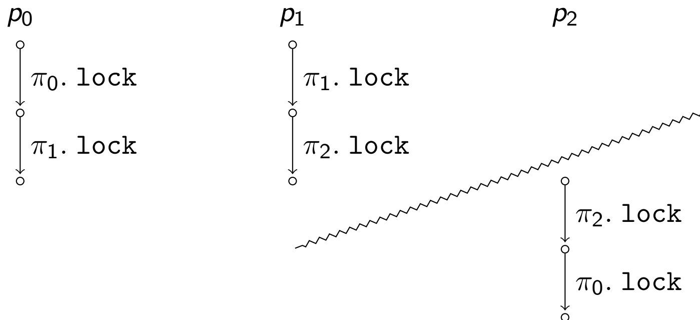
$$\frac{\Gamma \vdash v : \texttt{L}^\varrho :: \Delta \to \Delta ; C_1 \qquad X\ \textit{fresh} \qquad C_2 = X \geq \Delta \oplus (\varrho{:}1)}{\Gamma \vdash v.\ \texttt{lock} : \texttt{L}^\varrho :: \Delta \to X ; C_1, C_2}\ \textsc{T-Lock}$$

Transformation

An example of cycle length three

- Δ_C is given as

$$p_0 : \pi_0$$
$$p_1 : \pi_1$$
$$p_2 : \pi_2$$

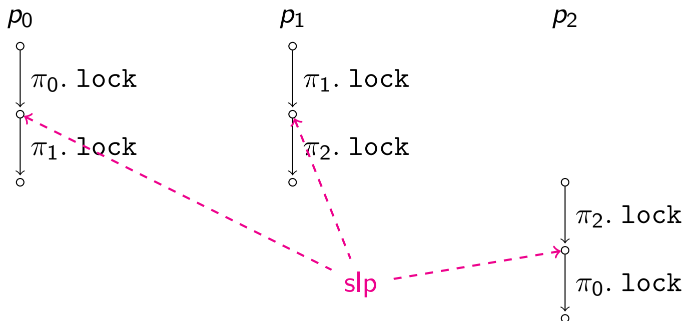


Transformation

An example of cycle length three

- Δ_C is given as

$$p_0 : \pi_0$$
$$p_1 : \pi_1$$
$$p_2 : \pi_2$$



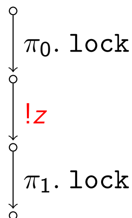
Transformation

An example of cycle length three

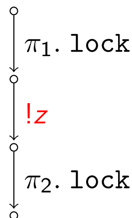
- Δ_C is given as

$$p_0 : \pi_0$$
$$p_1 : \pi_1$$
$$p_2 : \pi_2$$

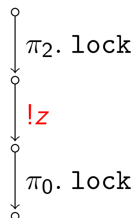
p_0



p_1

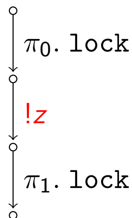


p_2

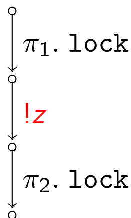


Transformation

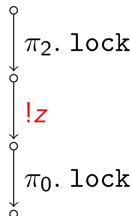
p_0



p_1



p_2



- Races are *binary*
- Deadlocks are *n-ary*
- To compensate, add locks appropriately

Gate locks

- Gate locks
  - *Short-lived*
    - No locking-step before a short-lived lock is released
    - Does not lead to more deadlocks
  - Variable access between locking and unlocking steps
  - One variable is guarded by one gate lock

Gate locks

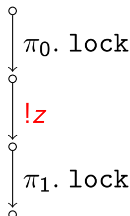
An example of cycle length three

- Δ_C is given as

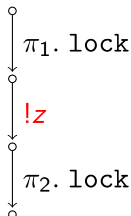
  $p_0 : \pi_0$
  $p_1 : \pi_1$
  $p_2 : \pi_2$

- Add gate locks in n - 1 processes
- Analyze n combinations

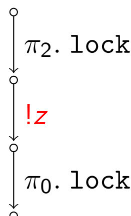
p_0



p_1



p_2



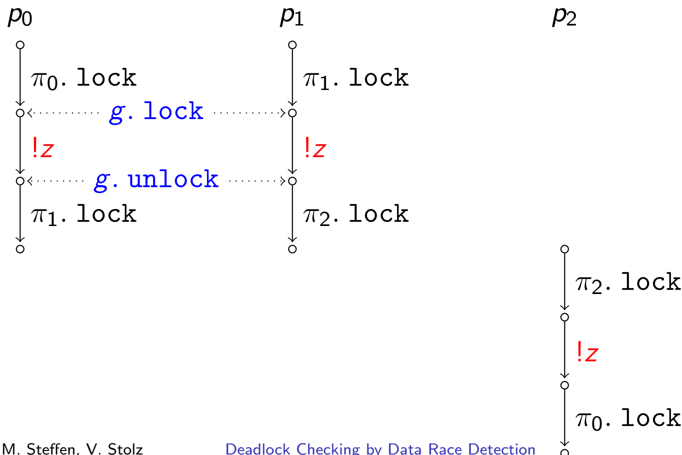
Gate locks

An example of cycle length three

- Δ_C is given as

$$p_0 : \pi_0$$
$$p_1 : \pi_1$$
$$p_2 : \pi_2$$

- Add gate locks in n - 1 processes
- Analyze n combinations



Analyzers

- `Goblint`
  - Race checker for `C` programs (. . . and more)
  - Does not check deadlocks
- `JPF` (Java Path Finder)
  - Model checker for Java bytecode
- `Chord` (Checker of races and deadlocks)
  - Checks deadlock of length 2
  - Recognizes locks held using `synchronized`

	C	Java			
		synchronized		explicit locks	
	Goblint	JPF	Chord	JPF	Chord
Datarace	✓	✓	✓	✓	✓
Deadlock 2	✗	✓	✓	✓	✗
Deadlock 3+	✗	✓	✗	✓	✗

Summary

- Develop a formal type and effect system
- Transformation guarantees each *slp* is protected by the same variable
- Prove soundness of the approach
  - Programs with deadlocks
    $\Longrightarrow$ data race in the transformed one
  - Race *free* in the transformed program
    $\Longrightarrow$ deadlock *free* in the original one
- Implementation of *second lock point* analysis in `Goblint`'s framework

Summary

- Develop a formal type and effect system
- Transformation guarantees each *slp* is protected by the same variable
- Prove soundness of the approach
  - Programs with deadlocks
    $\Longrightarrow$ data race in the transformed one
  - Race *free* in the transformed program
    $\Longrightarrow$ deadlock *free* in the original one
- Implementation of *second lock point* analysis in `Goblint`'s framework

Summary

- Develop a formal type and effect system
- Transformation guarantees each *slp* is protected by the same variable
- Prove soundness of the approach
  - Programs with deadlocks
    $\Longrightarrow$ data race in the transformed one
  - Race *free* in the transformed program
    $\Longrightarrow$ deadlock *free* in the original one
- Implementation of *second lock point* analysis in `Goblint`'s framework