Deadlock Checking by Data Race Detection

Ka I Violet Pun, Martin Steffen, Volker Stolz

PMA Group, University of Oslo, Norway

The 24th Nordic Workshop for Programming Theory - NWPT '12
Bergen, Norway

31st October, 2012

Overview

Goal

Find *potential* deadlocks in programs *statically* by detecting data race

- 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
  - E.g. critical region
  - Protected by locks

Overview

General approach:

- Reduce the problem of deadlock checking to race checking
- Instrument programs with appropriate shared variable accesses, called *race variables*
- Programs with deadlocks
  $\Longrightarrow$ data race in the transformed one

Assume the original programs are race free

Concurrent Calculus

- Functional language
- Higher-order
- Dynamic thread creation
- Dynamic lock creation
- Non-lexically scoped locks

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

Type and effect system

- Captures *static program points* where deadlocks can actually manifest themselves with a *type and effect system*
- Uses *program points* π, to characterize locks according to their origin
- Uses constraints to derive the *smallest* possible types
  - In terms of the originating locations
- Tracks relative change to the lock count
- Analyzes each thread locally

Type and Effect System

Judgements:

$$\Gamma \vdash e : T :: \varphi; C$$

Types and effects are described by:

$$\begin{array}{rcll} T & ::= & B \mid \mathrm{L}^r \mid T \xrightarrow{\varphi} T & \text{types} \\ r & ::= & \varrho \mid \{\pi\} \mid r \cup r & \text{lock/label sets} \\ \color{red}{\varphi} & ::= & \color{red}{\Delta \to \Delta} & \text{effects/pre- and post specification} \\ \Delta & ::= & \bullet \mid \Delta, \varrho{:}n & \text{abstract state} \\ C & ::= & \emptyset \mid \varrho \supseteq r, C & \text{constraints} \end{array}$$

Type and Effect System

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

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

$$\frac{\Gamma \vdash v : \texttt{L}^\varrho :: \Delta_1 \to \Delta_1; C \qquad \Delta_2 = \Delta_1 \oplus \varrho}{\Gamma \vdash v.\ \texttt{lock}: \texttt{L}^\varrho :: \Delta_1 \to \Delta_2; C} \text{ T-LOCK}$$

Second lock point

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

- The type and effect system works thread-locally
- Derives potential *slp* per thread wrt. a given cycle Δ_C

- Abstract cycle Δ_C
  - A sequence of pairs $p_1 : \pi_1; \ldots p_n : \pi_n$
  - Interpreted as process p_1 has π_1 and wants π_2

Second lock point

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

1. $t = \texttt{let}\ x\colon \texttt{L}^{\{\dots,\pi,\dots\}} = v.\ \texttt{lock in}\ t'$.
2. $\Delta_1 \vdash_p t :: \Delta_2$, for some Δ_1 and Δ_2, occurs in a sub-derivation of $\bullet \vdash t_0 :: \Delta$.

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

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

1. $t = \texttt{let}\ x\colon \mathtt{L}^{\{\ldots,\pi,\ldots\}} = v.\ \texttt{lock in}\ t'$.
2. $\Delta_1 \vdash_p t :: \Delta_2$, for some Δ_1 and Δ_2, occurs in a sub-derivation of $\bullet \vdash t_0 :: \Delta$.

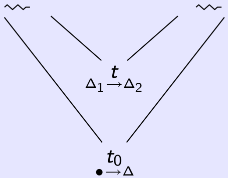
3. there exists π' s.t.

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

Second lock point

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

1. $t = \texttt{let}\ x{:}\ \mathtt{L}^{\{\ldots,\pi,\ldots\}} = v.\ \texttt{lock in}\ t'$.
2. $\Delta_1 \vdash_p t :: \Delta_2$, for some Δ_1 and Δ_2, occurs in a sub-derivation of $\bullet \vdash t_0 :: \Delta$.



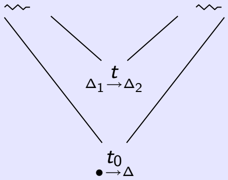
3. there exists π' s.t.

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

Second lock point

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

1. $t = \texttt{let}\ x{:}\ \texttt{L}^{\{\ldots,\pi,\ldots\}} = v.\ \texttt{lock in}\ t'$.
2. $\Delta_1 \vdash_p t :: \Delta_2$, for some Δ_1 and Δ_2, occurs in a sub-derivation of $\bullet \vdash t_0 :: \Delta$.



3. there exists π' s.t.

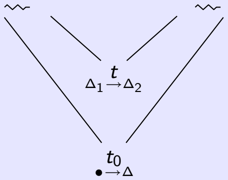
$$\pi' \in \Delta_1, \qquad \Delta_C \vdash p \text{ has } \pi', \quad \text{and} \quad \Delta_C \vdash p \text{ wants } \pi$$

Second lock point

Second lock point

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

1. $t = \texttt{let } x\colon \texttt{L}^{\{\ldots,\pi,\ldots\}} = v.\ \texttt{lock in } t'$.
2. $\color{blue}{\Delta_1}$ $\vdash_p t :: \Delta_2$, for some Δ_1 and Δ_2, occurs in a sub-derivation of $\bullet \vdash t_0 :: \Delta$.



3. there exists π' s.t.

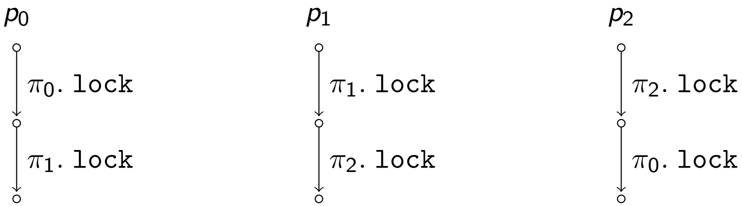
$$\pi' \in \color{blue}{\Delta_1}, \qquad \Delta_C \vdash p \text{ has } \pi', \quad \text{and} \quad \Delta_C \vdash p \text{ wants } \pi$$

Transformation

For three dining philosophers:

- Δ_C is given as

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

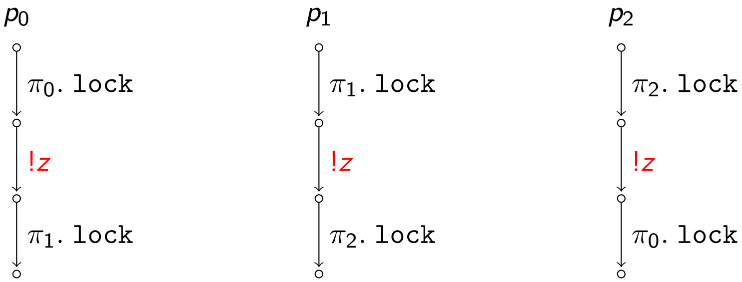


Transformation

For three dining philosophers:

- Δ_C is given as

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



Gate locks

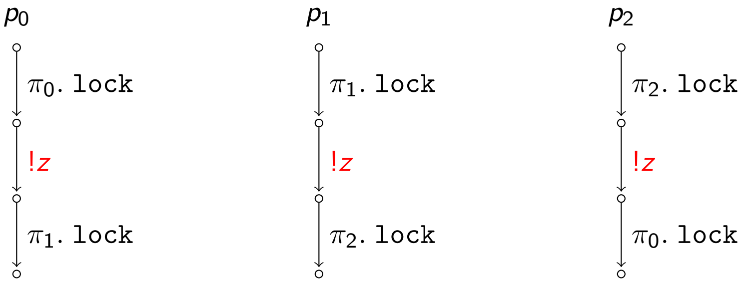
- Reduce deadlock checking to race checking
  - Races are *binary*, whereas deadlocks in general are not
  - To compensate, add locks appropriately
- Gate locks
  - *Short-lived locks*
    - *No* locking-step before a short-lived lock is released
  - Variable access between locking and unlocking steps
  - One variable is guarded by one gate lock
  - Does not lead to more deadlocks

Gate locks

For three dining philosophers:

- Δ_C is given as

$$\begin{aligned} p_0 &: \pi_0 \\ p_1 &: \pi_1 \\ p_2 &: \pi_2 \end{aligned}$$

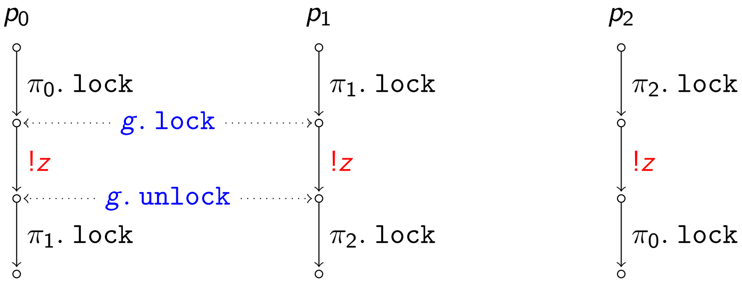


Gate locks

For three dining philosophers:

- Δ_C is given as

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



Gate lock for p_2?

Analyzers

- `Goblint`
  - Does not check deadlocks
- `JFP` (Java Path Finder)
- `Chord`
  - Checks deadlock of length 2
  - Recognizes locks held using `synchronized`

	C	Java			
		synchronized		explicit locks	
	Goblint	JPF	Chord	JPF	Chord
Datarace	*yes*	*yes*	*yes*	*yes*	*yes*
Deadlock 2	*N/A*	*yes*	*yes*	*yes*	*N/A*
Deadlock 3	*N/A*	*yes*	*N/A*	*yes*	*N/A*

Summary

- Formal description of the type and effect system
- Transformation guarantees each *slp* is protected by the same variable
- Prove soundness of the approach
  - Programs with (potential) deadlocks
    $\Longrightarrow$ data race in the transformed one
  - Race free in the transformed program
    $\Longrightarrow$ deadlock free in the original one