Polymorphic Behavioural Lock Effects for Deadlock Checking

Ka I Violet Pun, Martin Steffen, Volker Stolz

PMA Group, University of Oslo, Norway

The 23rd Nordic Workshop for Programming Theory - NWPT '11
Västerås, Sweden

26th $\sim$ 28th October, 2011

Overview

- Find *potential* deadlocks in programs *statically* by detecting cyclic wait
  - Each of two or more processes, which form a circular chain, wait for a shared resource that is held by the next process in the chain.
  - Shared resources here: *locks*

Overview

- Capture abstract behaviour as effects with a type and effect system
- Use *program points* π, to characterize locks according to their origin
- Execute the abstract behaviour to detect deadlock
- Limit potential infinite state space by:
  - Put an upper bound for reentrant lock counter
  - Transform effects into coarser, tail-recursive effect
  - Don't allow recursive thread/lock creation
- Prove deadlock presrvation by defining a *Deadlock and Termination Sensitive Simulation*

Syntax

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

Sequential composition $e_1; e_2$ is represented by `let`-construct

$$
\texttt{let}\ x{:}T = e_1\ \texttt{in}\ e_2\ , \qquad x \notin fv(e_2)
$$

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

Sequential composition $e_1; e_2$ is represented by `let`-construct

$$\texttt{let}\ x{:}T = e_1\ \texttt{in}\ e_2\ , \qquad x \notin fv(e_2)$$

Dining Philosophers

```
let l1 = new_π1 L, l2 = new_π2 L, l3 = new_π3 L,
    l4 = new_π4 L, l5 = new_π5 L in
let grab = fn:L×L⟶L. (l, r). l.lock; r.lock in
let release = fn:L×L⟶L. (l, r). l.unlock; r.unlock in
let phil = fun PHIL:L×L⟶L.(l, r). think; grab(l, r);
    eat; release(l, r); PHIL (l, r) in
    spawn(phil(l1,l2));...;spawn(phil(l5,l1))
```

Operational semantics

$$P ::= \emptyset \mid p\langle t\rangle \mid P \parallel P \qquad (\textit{Processes})$$
$$\sigma \vdash P \rightarrow \sigma' \vdash P' \text{ with } \quad \sigma : L \mapsto \{\mathrm{free}, p(n)\} \quad (\textit{Configuration})$$

An example run:

$$\emptyset \vdash p_0\langle t\rangle \rightarrow \ldots \rightarrow [l_1 \mapsto p_1(1), l_2 \mapsto p_0(1)] \vdash p_1\langle l_2.\,\mathtt{lock}\rangle \parallel p_0\langle l_1.\,\mathtt{lock}\rangle$$

Circular Wait

Definition (Waiting for a lock)

Given a configuration $\sigma \vdash P$,

$$waits(\sigma \vdash P, p, l)$$

if it is not the case that $\sigma \vdash P \xrightarrow{p\langle l.\texttt{lock}\rangle}$, and furthermore there exists a σ' s.t. $\sigma' \vdash P \xrightarrow{p\langle l.\texttt{lock}\rangle} \sigma'' \vdash P'$.

Definition (Deadlock)

A configuration $\sigma \vdash P$ *is deadlocked* if $\sigma(l_i) = p_i(n_i)$ and furthermore $waits(\sigma \vdash P, p_i, l_{i+_k 1})$
(where $k \geq 2$ and for all $0 \leq i \leq k-1$).

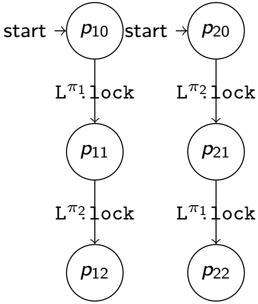


Figure: Deadlock

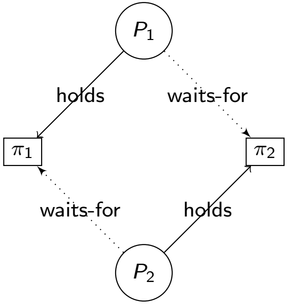


Figure: Wait-for graph

Type and Effect System

The judgment of our type and effect system is given by:

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

Types and effects are described by:

U	$::=$	$\texttt{Bool} \mid \texttt{Int} \mid \texttt{L}^r \mid \texttt{Thread}$	basic types
T	$::=$	$U \mid \vec{U} \xrightarrow{\varphi} U \mid \forall \varrho . T$	types
r	$::=$	$\pi \mid \varrho$	location annotations

Type and Effect System

The judgment of our type and effect system is given by:

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

Types and effects are described by:

U	$::=$	$\texttt{Bool} \mid \texttt{Int} \mid \texttt{L}^r \mid \texttt{Thread}$	basic types
T	$::=$	$U \mid \vec{U} \xrightarrow{\varphi} U \mid \forall \varrho . T$	types
r	$::=$	$\pi \mid \varrho$	location annotations
Φ	$::=$	$\mathbf{0} \mid p\langle \varphi \rangle \mid \Phi \parallel \Phi$	effects (global)
a	$::=$	$\texttt{spawn}\, \varphi \mid \nu \texttt{L}^r \mid \texttt{L}^r.\texttt{lock} \mid \texttt{L}^r.\texttt{unlock}$	labels/basic effects
α	$::=$	$a \mid \tau$	transition labels

The judgment of our type and effect system is given by:

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

Types and effects are described by:

U	$::=$	$\texttt{Bool} \mid \texttt{Int} \mid \texttt{L}^r \mid \texttt{Thread}$	basic types
T	$::=$	$U \mid \vec{U} \xrightarrow{\varphi} U \mid \forall \varrho. T$	types
r	$::=$	$\pi \mid \varrho$	location annotations
Φ	$::=$	$\mathbf{0} \mid p\langle\varphi\rangle \mid \Phi \parallel \Phi$	effects (global)
φ	$::=$	$\epsilon \mid X \mid \varphi;\varphi \mid \varphi + \varphi \mid rec\, X.\varphi \mid \alpha$	effects (local)
a	$::=$	$\texttt{spawn}\, \varphi \mid \nu\texttt{L}^r \mid \texttt{L}^r.\texttt{lock} \mid \texttt{L}^r.\texttt{unlock}$	labels/basic effects
α	$::=$	$a \mid \tau$	transition labels

To detect a deadlock in a program, we execute the abstract behaviour of the program. In our example:

```
let l₁ = new_π₁ L, l₂ = new_π₂ L, l₃ = new_π₃ L,
    l₄ = new_π₄ L, l₅ = new_π₅ L in
let grab = fn:L×L→L. (l, r). l.lock; r.lock in
let release = fn:L×L→L. (l, r). l.unlock; r.unlock in
let phil = fun PHIL:L×L→L.(l, r). think; grab(l, r);
    eat; release(l, r); PHIL (l, r) in
    spawn(phil(l₁,l₂));...;spawn(phil(l₅,l₁))
```

We have the effect:

$$\nu \mathtt{L}^{\pi_1};\ldots;\nu \mathtt{L}^{\pi_5}; \mathtt{spawn}\ (\varphi_p(\pi_1,\pi_2));\ldots;\mathtt{spawn}\ (\varphi_p(\pi_5,\pi_1))$$

$$\begin{aligned}\varphi_p(\varrho_1,\varrho_2) = \mathit{rec}\, X.\quad & \mathtt{think};\ \mathtt{L}^{\varrho_1}.\mathtt{lock};\ \mathtt{L}^{\varrho_2}.\mathtt{lock};\ \mathtt{eat};\\ & \mathtt{L}^{\varrho_1}.\mathtt{unlock};\ \mathtt{L}^{\varrho_2}.\mathtt{unlock};\ X\end{aligned}$$

Deadlock Checking

$$[] \vdash p\langle \nu\mathtt{L}^{\pi}1; \ldots; \nu\mathtt{L}^{\pi}5; \mathtt{spawn}\ (\varphi_p(\pi_1, \pi_2)); \ldots; \mathtt{spawn}\ (\varphi_p(\pi_5, \pi_1))\rangle \xrightarrow{p\langle \nu\mathtt{L}^{\pi}1;\ldots;\nu\mathtt{L}^{\pi}5\rangle}$$

$$[\pi_1 \mapsto \mathit{free}] \ldots [\pi_5 \mapsto \mathit{free}] \vdash p\langle \mathtt{spawn}\ (\varphi_p(\pi_1, \pi_2)); \ldots; \mathtt{spawn}\ (\varphi_p(\pi_5, \pi_1))\rangle \xrightarrow{p\langle \mathtt{spawn}(\varphi_p(\pi_1,\pi_2));\ldots\rangle}$$

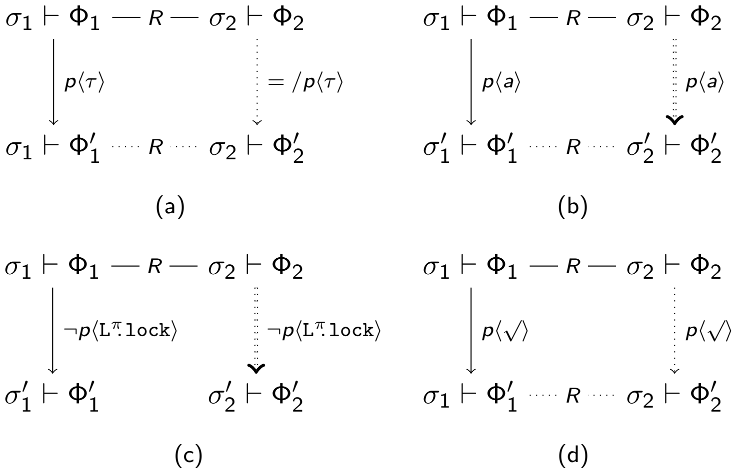
$$\vdots$$

$$\begin{aligned}[\pi_1 \mapsto \mathit{free}] \ldots [\pi_5 \mapsto \mathit{free}] \vdash p_1\langle &\mathtt{L}^{\pi}1.\mathtt{lock}; \mathtt{L}^{\pi}2.\mathtt{lock}; \mathtt{L}^{\pi}1.\mathtt{unlock}; \mathtt{L}^{\pi}2.\mathtt{unlock};\\ &\mathit{rec}\ X.\mathtt{L}^{\pi}1.\mathtt{lock}; \ldots\rangle \parallel \ldots \parallel\\ p_5\langle &\mathtt{L}^{\pi}5.\mathtt{lock}; \mathtt{L}^{\pi}1.\mathtt{lock}; \mathtt{L}^{\pi}5.\mathtt{unlock}; \mathtt{L}^{\pi}1.\mathtt{unlock};\\ &\mathit{rec}\ X.\mathtt{L}^{\pi}5.\mathtt{lock}; \ldots\rangle \xrightarrow{p_1\langle \mathtt{L}^{\pi}1\ \mathtt{lock}\rangle}\end{aligned}$$

$$\vdots$$

$$\begin{aligned}[\pi_1 \mapsto p_1(1)][\pi_2 \mapsto p_2(2)] \ldots [\pi_5 \mapsto p_5(1)] \vdash p_1\langle &\mathtt{L}^{\pi}2.\mathtt{lock}; \mathtt{L}^{\pi}1.\mathtt{unlock}; \mathtt{L}^{\pi}2.\mathtt{unlock};\\ &\mathit{rec}\ X.\mathtt{L}^{\pi}1.\mathtt{lock}; \ldots\rangle \parallel \ldots \parallel\\ p_5\langle &\mathtt{L}^{\pi}1.\mathtt{lock}; \mathtt{L}^{\pi}5.\mathtt{unlock}; \mathtt{L}^{\pi}1.\mathtt{unlock};\\ &\mathit{rec}\ X.\mathtt{L}^{\pi}5.\mathtt{lock}; \ldots\rangle \ \not\xrightarrow{p_1\langle \mathtt{L}^{\pi}2\ \mathtt{lock}\rangle}\end{aligned}$$

Deadlock and termination sensitive simulation $\lesssim^D / \lesssim^{DT}$



Infinite State Space

Two sources of infinity

- Unboundedness of *reentrant* lock counters
- Unboundedness of the “control stack” of *non-tail* recursive behaviour descriptions

Problem in state space:

Unbounded lock counters counting uuuuuuupppppp (with recursion). . .

Solution:

Fix upper bound; unlocking from upper bound becomes non-deterministic.

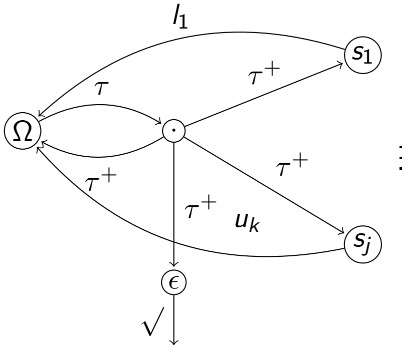
Lemma

Given a configuration $\sigma \vdash \Phi$*, and let further denote* $\sigma_1 \vdash^{n_1} \Phi$ *and* $\sigma_2 \vdash^{n_2} \Phi$ *the corresponding configurations under the lock-counter abstraction. If* $n_1 \geq n_2$*, then* $\sigma_1 \vdash^{n_1} \Phi \lesssim^D \sigma_2 \vdash^{n_2} \Phi$*.*

Random Behaviour Ω

Lemma (Ω is maximal wrt. $\lesssim^{DT}$)

Assume φ over a set of locations r, then $\sigma \vdash p\langle\varphi\rangle \lesssim^{DT} \sigma \vdash p\langle\Omega\rangle$.



Theorem (Finite abstractions)

The lock counter abstraction and behavior abstraction (when abstracting all locks and recursions) results in a finite state space.

Theorem (Soundness of the abstraction)

Given $\Gamma \vdash P : ok :: \Phi$ *and two heaps* $\sigma_1 \equiv \sigma_2$. *Further,* $\sigma_2' \vdash \Phi'$ *is obtained by lock-counter resp. behavior abstraction of* $\sigma_2 \vdash \Phi$. *Then if* $\sigma_2' \vdash \Phi'$ *is deadlock free then so is* $\sigma_1 \vdash P$.

Summary

- Conclusion:
  - We have proven that our type systems is correct in the aspect of capturing behavior of a program
  - Abstract behavior correctly over-approximates the concrete one
  - Deadlocks in a program are correctly detected in the abstract run. . .
  - Inference algorithm is partially formalized with `Ott` and `Coq`
- Future Work:
  - Applying to communication analysis of asynchronous systems
  - Relaxing the condition (e.g. lock creation in loop)
  - Abstracting processes
  - Implement our algorithm with model checker for real language
  - CEGAR - Counter-Example Guided Abstraction Refinement