Effect-Polymorphic Behaviour Inference for Deadlock Checking

Ka I Violet Pun, Martin Steffen, Volker Stolz

University of Oslo + Høgskole i Bergen, Norway

Bamberg

December 2017

Goal

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

Overview

- deadlock freedom: global (safety) property
- two stage approach: local $\Leftrightarrow$ global
  1. local level:
     - behavioral effects for lock interactions
     - polymorphic
  2. global level: exploration of the abstract behavior to detect deadlock
- potentially ∞ state space
  - re-entrant lock counter
  - stack-structure for function calls
  - dynamic lock creation

$\Rightarrow$ abstractions needed

Techniques used

- type and effect system with behavioral effects (+ flow information)
- behavior effects as in e.g. [Amtoft et al., 1999]
- type/effect inference using constraint-based formulation as in e.g. loc.cit
- polymorphic analysis, for enhanced precision (let-polymorphism)
- for proving soundness
  - of the effect type system ("subject reduction")
  - of the abstractions

$\Rightarrow$ deadlock (and termination) sensitive simulation

Overview

1. Introduction
2. Syntax and semantics
3. Type and effect system
4. Abstract behavior
5. Summary

Syntax

- small concurrent calculus
- dynamic lock/thread creation
- higher-order functions
- re-entrant, heap-allocated locks

$$
\begin{array}{rcl}
P & ::= & \emptyset \mid p\langle t\rangle \mid P \parallel P \\
t & ::= & v \mid \texttt{let}\ x{:}T = e\ \texttt{in}\ t \\
e & ::= & t \mid v\ v \mid \texttt{if}\ v\ \texttt{then}\ e\ \texttt{else}\ e \mid \texttt{spawn}\ t \mid \texttt{new L} \\
 & \mid & v.\,\texttt{lock} \mid v.\,\texttt{unlock} \\
v & ::= & x \mid l^r \mid () \mid \texttt{true} \mid \texttt{false} \mid \texttt{fn}\ x{:}T.t \mid \texttt{fun}\ f{:}T.x{:}T.t
\end{array}
$$

p
t

e
v

Syntax

- small concurrent calculus
- dynamic lock/thread creation
- higher-order functions
- re-entrant, heap-allocated locks

$$
\begin{array}{rcl}
P & ::= & \emptyset \mid p\langle t\rangle \mid P \parallel P \\
t & ::= & v \mid \texttt{let } x{:}T = e \texttt{ in } t \\
e & ::= & t \mid v\ v \mid \texttt{if } v \texttt{ then } e \texttt{ else } e \mid \texttt{spawn } t \mid \texttt{new L} \\
 & \mid & v.\,\texttt{lock} \mid v.\,\texttt{unlock} \\
v & ::= & x \mid l^r \mid () \mid \texttt{true} \mid \texttt{false} \mid \texttt{fn } x{:}T.t \mid \texttt{fun } f{:}T.x{:}T.t
\end{array}
$$

p
t
e
v

Dining philosophers

```
let l0 = new L; l1 = new L; l2 = new L /* create all locks */
  phil = fn x:L,y:L . ( x.lock; y.lock;      /* eat */
                        y.unlock; x.unlock; /* think */ )
  in spawn(phil(l0,l1)); spawn(phil(l1,l2)); spawn(phil(l2,l0))
```

Operational semantics

$$P ::= \emptyset \mid p\langle t\rangle \mid P \parallel P \qquad (\textit{processes})$$

$$\sigma \vdash P \to \sigma' \vdash P' \text{ with } \quad \sigma : L \mapsto \{\mathrm{free}, p(n)\} \quad (\textit{configurations})$$

An example run (only 2 phil's):

$$\emptyset \vdash p_0\langle t\rangle \to \ldots \to [l_0 \mapsto p_0(1), l_1 \mapsto p_1(1)] \vdash p_0\langle l_1.\,\texttt{lock}\rangle \parallel p_1\langle l_0.\,\texttt{lock}\rangle$$

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 $0 \leq i \leq k-1$).

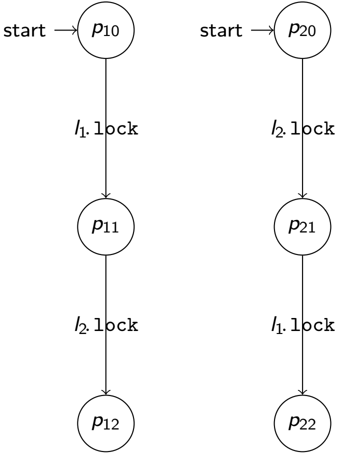


Figure: Deadlock

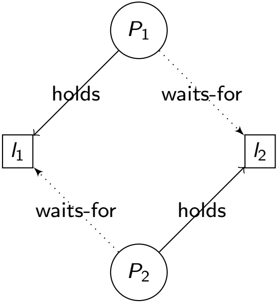


Figure: Wait-for graph

Type and effect system

- *behavioral* effects φ: interactions of a thread with locks

Judgments

$$\Gamma \vdash t : T :: \varphi$$

- simple process "algebra"
- actions: locking/unlocking
- latent effects for function types: $T_1 \xrightarrow{\varphi} T_2$

Types & effects for lock interaction

types

$$T \ ::= \ B \ | \ \mathtt{L} \ | \ T \to T \quad \text{types}$$

Types & effects for lock interaction

types

$$
\begin{array}{rcll}
r & ::= & \{\pi\} \mid r \sqcup r & \text{lock/label sets} \\
\hat{T} & ::= & B \mid \mathtt{L}^r \mid \hat{T} \xrightarrow{\varphi} \hat{T} & \text{types}
\end{array}
$$

Types & effects for lock interaction

types

$$
\begin{array}{lcll}
r & ::= & \{\pi\} \mid r \sqcup r & \text{lock/label sets} \\
\hat{T} & ::= & B \mid \mathtt{L}^r \mid \hat{T} \xrightarrow{\varphi} \hat{T} & \text{types}
\end{array}
$$

effects

$$
\begin{array}{lcll}
\Phi & ::= & 0 \mid p\langle\varphi\rangle \mid \Phi \parallel \Phi & \text{effects (global)} \\
\varphi & ::= & \epsilon \mid \varphi;\varphi \mid \varphi+\varphi \mid \alpha \mid X \mid rec\,X.\varphi & \text{effects (local)} \\
\alpha & ::= & a \mid \tau & \text{transition labels} \\
a & ::= & \mathtt{spawn}\ \varphi \mid r.\mathtt{lock} \mid r.\mathtt{unlock} & \text{labels/basic effects}
\end{array}
$$

Polymorphism, type inference & constraints

- type level variables
  - for sets of π-locations ("regions")
  - for behavior
- to enhance precision: type schemes
- flow and behavior constraints

Thread-local type and effect system

Judgments

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

$$
\begin{array}{rcll}
Y & ::= & \varrho \mid X & \text{type-level variables} \\
r & ::= & \varrho \mid \{\pi\} \mid r \sqcup r & \text{lock/label sets} \\
\hat{T} & ::= & B \mid \mathtt{L}^r \mid \hat{T} \xrightarrow{\varphi} \hat{T} & \text{types}
\end{array}
$$

Thread-local type and effect system

Judgments

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

$$\begin{array}{rcll} Y & ::= & \varrho \mid X & \text{type-level variables} \\ r & ::= & \varrho \mid \{\pi\} \mid r \sqcup r & \text{lock/label sets} \\ \hat{T} & ::= & B \mid \mathtt{L}^r \mid \hat{T} \xrightarrow{\varphi} \hat{T} & \text{types} \\ \hat{S} & ::= & \forall \vec{Y}.\ \hat{T} & \text{type schemes} \end{array}$$

Thread-local type and effect system

Judgments

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

Y	$::=$	$\varrho \mid X$	type-level variables
r	$::=$	$\varrho \mid \{\pi\} \mid r \sqcup r$	lock/label sets
$\hat{T}$	$::=$	$B \mid \mathtt{L}^r \mid \hat{T} \xrightarrow{\varphi} \hat{T}$	types
$\hat{S}$	$::=$	$\forall \vec{Y}{:}C.\ \hat{T}$	type schemes
C	$::=$	$\emptyset \mid \varrho \sqsupseteq r, C \mid X \sqsupseteq \varphi, C$	constraints

Type and Effect System

For our example:

```
let x : L^π1 = new^π1 L in
let y : L^π2 = new^π2 L in
  spawn (y.lock;x.lock;stop); x.lock;y.lock;stop
```

Effect:

$$\varphi = \nu L^{\pi_1}; \nu L^{\pi_2};\ \texttt{spawn}\ (\pi_2.\texttt{lock}; \pi_1.\texttt{lock}\ ); \pi_1.\texttt{lock}; \pi_2.\texttt{lock}$$

Thread-local type and effect system

- instead of checking "subtyping" of "sub-effecting" $\Rightarrow$ generating approriate constraints on-the fly, using fresh variables
- can be seen a generalization of "Algorithm W"
- type schemes for polymorphic analysis, constraints "qualifying" the bound variables

Thread-local type and effect system

$$\frac{\Gamma(x) = \forall \vec{Y}{:}C.\hat{T} \qquad \vec{Y}'\ \mathit{fresh} \qquad \theta = [\vec{Y}'/\vec{Y}]}{\Gamma \vdash x : \theta\hat{T} :: \epsilon; \theta C}\ \text{TA-VAR}$$

$$\frac{\varrho\ \mathit{fresh}}{\Gamma \vdash\ \mathtt{new}^{\pi}\ \ \mathtt{L} :\ \mathtt{L}^{\varrho} :: \epsilon; \varrho \sqsupseteq \{\pi\}}\ \text{TA-NEWL}$$

$$\frac{\hat{T}_1 = \lceil T_1 \rceil_a \qquad \Gamma, x{:}\hat{T}_1 \vdash e : \hat{T}_2 :: \varphi; C \qquad X\ \mathit{fresh}}{\Gamma \vdash \mathtt{fn}\ x{:}T_1.e : \hat{T}_1 \xrightarrow{X} \hat{T}_2 :: \epsilon; C, X \sqsupseteq \varphi}\ \text{TA-ABS}_1$$

$$\frac{\lceil T_1 \to T_2 \rceil_a = \hat{T}_1 \xrightarrow{X} \hat{T}_2 \qquad \Gamma, f{:}\hat{T}_1 \xrightarrow{X} \hat{T}_2, x{:}\hat{T}_1 \vdash e : \hat{T}_2' :: \varphi; C_1 \qquad \hat{T}_2' \leq \hat{T}_2 \vdash_a C_2}{\Gamma \vdash \mathtt{fun}\ f{:}T_1 \to T_2, x{:}T_1.e : \hat{T}_1 \xrightarrow{X} \hat{T}_2 :: \epsilon; C_1, C_2, X \sqsupseteq \varphi}\ \text{TA-A}$$

$$\frac{\Gamma \vdash v_1 : \hat{T}_2 \xrightarrow{\varphi} \hat{T}_1 :: \epsilon; C_1 \qquad \Gamma \vdash v_2 : \hat{T}_2' :: \epsilon; C_2 \qquad \hat{T}_2' \leq \hat{T}_2 \vdash_a C_3 \qquad X\ \mathit{fresh}}{\Gamma \vdash v_1\ v_2 : \hat{T}_1 :: X; C_1, C_2, C_3, X \sqsupseteq \varphi}\ \text{TA-APP}$$

Thread-local type and effect system

$$\frac{\lfloor \hat{T} \rfloor = \lfloor \hat{T}_1 \rfloor = \lfloor \hat{T}_2 \rfloor \qquad \hat{T}_1 \vee \hat{T}_2 = \hat{T}; C \qquad \varphi_1 \sqcup \varphi_2 = X; C' \\ \Gamma \vdash v : \mathtt{Bool} :: \epsilon; C_0 \quad \Gamma \vdash e_1 : \hat{T}_1 :: \varphi_1; C_1 \quad \Gamma \vdash e_2 : \hat{T}_2 :: \varphi_2; C_2}{\Gamma \vdash \mathtt{if}\ v\ \mathtt{then}\ e_1\ \mathtt{else}\ e_2 : \hat{T} :: X; C_0, C_1, C_2, C, C'} \text{ TA-COND}$$

$$\frac{\Gamma \vdash e_1 : \hat{T}_1 :: \varphi_1; C_1 \qquad \lfloor \hat{T}_1 \rfloor = T_1 \\ \hat{S}_1 = close(\Gamma, \varphi_1, C_1, \hat{T}_1) \qquad \Gamma, x{:}\hat{S}_1 \vdash e_2 : \hat{T}_2 :: \varphi_2; C_2}{\Gamma \vdash \mathtt{let}\ x{:}T_1 = e_1\ \mathtt{in}\ e_2 : \hat{T}_2 :: \varphi_1; \varphi_2; C_1, C_2} \text{ TA-LET}$$

$$\frac{\Gamma \vdash t : \hat{T} :: \varphi; C \qquad X\ fresh}{\Gamma \vdash \mathtt{spawn}\ t : \mathtt{Unit} :: X; C, X \sqsupseteq \mathtt{spawn}\ \varphi} \text{ TA-SPAWN}$$

$$\frac{\Gamma \vdash v : \mathrm{L}^{\varrho} :: \epsilon; C \quad X\ fresh}{\Gamma \vdash v.\,\mathtt{lock}: \mathrm{L}^{\varrho} :: X; C, X \sqsupseteq \varrho.\mathtt{lock}} \text{ TA-LOCK} \qquad \frac{\Gamma \vdash v : \mathrm{L}^{\varrho} :: \epsilon; C \quad X\ fresh}{\Gamma \vdash v.\,\mathtt{unlock}: \mathrm{L}^{\varrho} :: X; C, X \sqsupseteq \varrho.\mathtt{unlock}}$$

Soundness of the abstraction (thread local)

- we need the operational "behavior" of the effects for
  - local level: to relate the type system to the semantics (soundness, via subject reduction)
  - global level: deadlock checking (see later)
- defined using the constraints

labelled (weak) transitions

$C \vdash \varphi_1 \xRightarrow[\sqsubseteq]{a} \varphi_2$ given by $C \vdash a; \varphi_2 \sqsubseteq \varphi_1$.

- subject reduction with effects: a form of simulation proof
- however: beware of deadlocks

Subject reduction

$$
\begin{array}{ccc}
C_1; \hat{\sigma}_1 \vdash p\langle \varphi \rangle & & \\
\Big| & & \\
R & & \\
\Big| & & \\
\sigma_2 \vdash p\langle t \rangle & \xRightarrow{\;p\langle a \rangle\;} & \sigma_2' \vdash p\langle t' \rangle
\end{array}
$$

Subject reduction

$$
\begin{array}{ccc}
C_1; \hat{\sigma}_1 \vdash p\langle\varphi\rangle & \overset{p\langle a\rangle}{\Longrightarrow}_{\sqsubseteq} & C_1; \hat{\sigma}_1' \vdash p\langle\varphi'\rangle \\
\Big| & & \Big| \\
R & & R \\
\Big| & & \Big| \\
\sigma_2 \vdash p\langle t\rangle & \overset{p\langle a\rangle}{\Longrightarrow} & \sigma_2' \vdash p\langle t'\rangle
\end{array}
$$

Deadlock sensitive simulation

- for subject reduction: relating one thread with its effect
- globally: compositionality wrt. $\|$
- to relate effects at different levels of abstraction: relate effects

Definition (Deadlock sensitive simulation $\lesssim^{D}_{\sqsubseteq}$)

Assume a heap-mapping θ and a corresponding wait-sensitive abstraction $\leq_\theta$. A binary relation R between configurations is a *deadlock sensitive simulation relation* if the following holds. Assume $C_1; \hat{\sigma}_1 \vdash \Phi_1 \; R \; C_2; \hat{\sigma}_2 \vdash \Phi_2$ with $\hat{\sigma}_1 \leq_\theta \hat{\sigma}_2$. Then:

1. If $C_1; \hat{\sigma}_1 \vdash \Phi_1 \xRightarrow{p\langle a \rangle}_{\sqsubseteq} C_1; \hat{\sigma}'_1 \vdash \Phi'_1$, then $C_2; \hat{\sigma}_2 \vdash \Phi_2 \xRightarrow{p\langle a \rangle}_{\sqsubseteq} C_2; \hat{\sigma}'_2 \vdash \Phi'_2$ for some $C_2; \hat{\sigma}'_2 \vdash \Phi'_2$ with $\hat{\sigma}'_1 \leq_\theta \hat{\sigma}'_2$ and $C_1; \hat{\sigma}'_1 \vdash \Phi'_1 R \; C_2; \hat{\sigma}'_2 \vdash \Phi'_2$.
2. If $waits_{\sqsubseteq}((C_1; \hat{\sigma}_1 \vdash \Phi_1), p, \varrho)$, then $waits_{\sqsubseteq}((C_2; \hat{\sigma}_2 \vdash \Phi_2), p, \theta(\varrho))$.

4 sources of infinity

1. dynamic lock creation
2. Unboundedness of *reentrant* lock counters
3. "control stack" of *non-tail* recursive behaviours
4. process creation

Lock abstraction

- summarizing locks by their point of creation
- non-injective abstraction
- improvement over [Pun et al., 2012]
- abstract heap:
  - abstract lock = location(s) of lock creation
  - abstract lock counter = *sum* of all lock counters of concrete locks it represents

Lock counter abstraction

Problem in state space:

Unbounded lock counters counting up

Solution:

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

Lemma

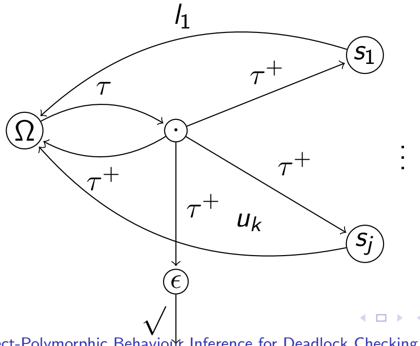
Given a configuration $\sigma \vdash \Phi$*, and let further denote* $\sigma_1 \vdash^{n_1} \Phi$ *and* $\sigma_2 \vdash^{n_2} \Phi$ *the corresponding configurations under the lock-counter abstraction. If* $n_1 \geq n_2$*, then* $\sigma_1 \vdash^{n_1} \Phi \lesssim^D \sigma_2 \vdash^{n_2} \Phi$*.*

Getting rid of the stack: Ω

- replacing context-free (stack) behavior by tail-recursive one
- beyond stack-depth-k: chaotic bevavior Ω
- again: beware of preserving deadlocks
- compositionality

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

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



Some results

Theorem (Finite abstractions)

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

Note: the "size" of the abstraction is *adaptable*

Theorem (Soundness of the abstraction)

Given $\Gamma \vdash P : ok :: \Phi$ *and two heaps* $\hat{\sigma}_1 \leq_\theta \hat{\sigma}_2$ *Further,* $\sigma'_2 \vdash \Phi'$ *is obtained by the mentioned abstractions from* $\sigma_2 \vdash \Phi$*. 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 soundly infers constraints capturing lock behavior
  - Abstract behavior correctly over-approximates the concrete one
  - Deadlocks in a program are correctly detected in the abstract run...
  - type system partially formalized with `Ott` and `Coq` (mono-case)

- Future Work:
  - Applying to communication analysis of asynchronous systems
  - Abstracting processes (probably hard)
  - Implement our algorithm with model checker for real language
  - CEGAR - Counter-Example Guided Abstraction Refinement

[Amtoft et al., 1999] Amtoft, T., Nielson, H. R., and Nielson, F. (1999).
Type and Effect Systems: Behaviours for Concurrency.
Imperial College Press.

[Pun et al., 2012] Pun, K. I., Steffen, M., and Stolz, V. (2012).
Deadlock checking by a behavioral effect system for lock handling.
Journal of Logic and Algebraic Programming, 81(3):331–354.
A preliminary version was published as University of Oslo, Dept. of Computer Science Technical Report 404, March 2011.

[Pun et al., 2013] Pun, K. I., Steffen, M., and Stolz, V. (2013).
Lock-polymorphic behaviour inference for deadlock checking.
Technical report 436, University of Oslo, Faculty of Mathematics and Natural Sciences, Dept. of Informatics.