Deadlock Checking by Behaviour Inference for Lock Handling

Ka I Violet Pun, Martin Steffen

PMA Group, University of Oslo, Norway

The 22nd Nordic Workshop for Programming Theory - NWPT '10
Turku, Finland

10 - 12 November, 2010
Agenda

1. Introduction
2. Syntax and Semantics
3. Type and Effect System
4. Abstract Behaviour
5. Summary
Four necessary conditions for a deadlock

- Mutual Exclusion
- Wait-for
- No-preemption
- Circular wait
Four necessary conditions for a deadlock

- Mutual Exclusion
- Wait-for
- No-preemption
- Circular wait

Deadlock
Each of two or more threads, which form a circular chain, wait for a lock that is held by the next thread in the chain.
Find potential deadlocks in programs by detecting circular wait statically

- Use *program points*, \( \pi \), to characterize locks according to their origin
- Calculate abstract behaviour by effect inference
- Execute the abstract behaviour

Setting:

- No lock creation in a loop/recursive function
- Reentrant lock in a loop (recursive function), the lock counter is set to
  - \( \infty \) (locking)
  - \( \perp \) (unlocking)
Example

```plaintext
let t = fn (z1, z2). z1.lock; z2.lock; stop in
  let x1 = new_{π1} L in
  let x2 = new_{π2} L in
  spawn (t (x1, x2)); t (x2, x1); stop
```

![Diagram of a process with two threads and locks]
Syntax

\[
\begin{align*}
  t & ::= \text{stop} | \nu | \text{let } x: T = e \text{ in } t \\
  e & ::= t | e \ e | \text{if } e \text{ then } e \text{ else } e | \text{spawn } e | \text{new } L | \nu.\text{lock} \\
  & \quad | \nu.\text{unlock} \\
  \nu & ::= x | l | \text{fn } x: T.t | \text{fun } f: T.x: T.t
\end{align*}
\]

Sequential composition \(e_1; e_2\) is represented by let-construct

\[
\text{let } x: T = e_1 \text{ in } e_2, \quad x \notin \text{fv}(e_2)
\]
Syntax

\[ t ::= \text{stop} \mid v \mid \text{let } x: T = e \text{ in } t \]

\[ e ::= t \mid e \ e \mid \text{if } e \text{ then } e \text{ else } e \mid \text{spawn } e \mid \text{new } L \mid v. \text{lock} \]

\[ v ::= x \mid l \mid \text{fn } x: T. t \mid \text{fun } f: T. x: T. t \]

Sequential composition \( e_1; e_2 \) is represented by let-construct

\[
\text{let } x: T = e_1 \text{ in } e_2, \quad x \notin \text{fv}(e_2)
\]

For the operational semantics:

\[
P ::= \emptyset \mid p\langle t \rangle \mid P \parallel P
\]

\[
\sigma \vdash P \rightarrow \sigma' \vdash P' \text{ with } \sigma : L \mapsto \{ \text{free, } p(n) \}
\]
Syntax

\[
\begin{align*}
t & ::= \text{stop} \mid v \mid \text{let } x:T = e \text{ in } t \\
e & ::= t \mid e\,e \mid \text{if } e \text{ then } e \text{ else } e \mid \text{spawn } e \mid \text{new } L \mid v.\text{lock} \\
& \quad \mid v.\text{unlock} \\
v & ::= x \mid l \mid \text{fn } x:T.t \mid \text{fun } f:T.x:T.t
\end{align*}
\]

Sequential composition \(e_1; e_2\) is represented by let-construct

\[
\text{let } x:T = e_1 \text{ in } e_2, \quad x \notin \text{fv}(e_2)
\]

For the operational semantics:

\[
P ::= \emptyset \mid p\langle t \rangle \mid P \parallel P
\]

\[
\sigma \vdash P \rightarrow \sigma' \vdash P' \quad \text{with} \quad \sigma : L \mapsto \{\text{free, } p(n)\}
\]

An example run:

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

K. Pun, M. Steffen  
Deadlock Checking by Behaviour Inference for Lock Handling  
NWPT’10
A deadlocked configuration $\sigma \vdash P$ is of the form:

$P = P' \parallel p_0\langle t_0 \rangle \parallel \ldots \parallel p_k\langle t_k \rangle$, where $k \geq 2$ and where the threads $t_i$ are of the form $t_i = \text{let } x_i : T_i = l_{i+1}. \text{lock in } t_i'$ for $0 \leq i < k$, and $t_k = \text{let } x_k : T_k = l_0. \text{lock in } t_k'$.
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:

$$T ::= \text{Bool} \mid \text{Int} \mid T \rightarrow \varphi \ T \mid L' \mid \text{Thread}$$

$$\varphi ::= \epsilon \mid x \mid \varphi;\varphi \mid \varphi + \varphi \mid \mu x.\varphi \mid \text{spawn } \varphi \mid \nu L'$$

$$L'.\text{lock} \mid L'.\text{unlock}$$

$$r ::= \{\pi\} \mid r \cup r \mid \emptyset$$
The judgment of the type and effect system with the inference algorithm is given by:

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

Types and effects are modified as:

$T ::= \alpha | \text{Bool} | \text{Int} | T \to \varphi \ T | L' | \text{Thread}$

$\varphi ::= \beta | \epsilon | x | \varphi;\varphi | \varphi + \varphi | \mu x.\varphi | \text{spawn } \varphi | \nu L'$

$\mid L'.\text{lock} | L'.\text{unlock}$

$r ::= \rho | \{\pi\} | r \cup r | \emptyset$
The judgment of the type and effect system with the inference algorithm is given by:

\[ \Gamma \vdash e : T :: \varphi, \theta \]

where \( \theta \) is a substitution which maps:
- \( \alpha \mapsto T \), \( \alpha \) is a type variable
- \( \beta \mapsto \varphi \), \( \beta \) is an effect variable
- \( \rho \mapsto r \), \( \rho \) is an annotation variable

Types and effects are modified as:

\[
\begin{align*}
T & ::= \alpha | \text{Bool} | \text{Int} | T \to \varphi T | L^r | \text{Thread} \\
\varphi & ::= \beta | \epsilon | x | \varphi; \varphi | \varphi + \varphi | \mu x.\varphi | \text{spawn } \varphi | \nu L^r \\
& \quad | L^r.\text{lock} | L^r.\text{unlock} \\
r & ::= \rho | \{\pi\} | r \cup r | \emptyset
\end{align*}
\]
For our example:

```plaintext
let t = fn (z₁, z₂). z₁.lock; z₂.lock; stop
in
  let x₁ = new_π₁ L in
  let x₂ = new_π₂ L in
  spawn (t (x₁, x₂)); t (x₂, x₁); stop
```
Type and Effect System

For our example:

```plaintext
let t = fn (z₁, z₂). z₁.lock; z₂.lock; stop
in
  let x₁ = new₁ L in
  let x₂ = new₂ L in
  spawn (t (x₁, x₂)); t (x₂, x₁); stop
```

Effect:

$$\varphi =$$
For our example:

```plaintext
let t = fn (z₁, z₂). z₁.lock; z₂.lock; stop in
  let x₁ = new_π₁ L in
  let x₂ = new_π₂ L in
  spawn (t (x₁, x₂)); t (x₂, x₁); stop
```

Effect:

\[ \varphi = \nu L^{\pi_1}; \nu L^{\pi_2}; \]

\[ \Gamma \vdash \text{new}_\pi L : L^{\pi::} \nu L^{\pi} \quad \text{TE-NewL} \]
Type and Effect System

For our example:

```
let t = fn (z_1, z_2). z_1.lock; z_2.lock; stop
in
  let x_1 = new_{\pi_1} L in
  let x_2 = new_{\pi_2} L in
  spawn (t (x_1, x_2)); t (x_2, x_1); stop
```

Effect:

\[ \varphi = \nu L^{\pi_1}; \nu L^{\pi_2}; \text{spawn} ( \ ) ; \]

\[
\Gamma \vdash e : T :: \varphi \\
\Gamma \vdash \text{spawn } e : \text{Thread} :: \text{spawn } \varphi
\]
Type and Effect System

For our example:

```ml
let t = fn (z₁, z₂). z₁.lock; z₂.lock; stop
in
  let x₁ = new π₁ L in
  let x₂ = new π₂ L in
  spawn (t (x₁, x₂)); t (x₂, x₁); stop
```

Effect:

\[ \varphi = \nu L^{\pi_1}; \nu L^{\pi_2}; \text{spawn}(L^{\pi_1}.\text{lock}; L^{\pi_2}.\text{lock}); \]

\[
\Gamma \vdash \nu : L^r :: \varphi \\
\hline
\Gamma \vdash \nu.\text{lock}: L^r :: \varphi; L^r.\text{lock} \\
\hline
\Gamma \vdash e_1 : T_2 \rightarrow^\varphi T_1 :: \varphi_1 \\
\Gamma \vdash e_2 : T_2 :: \varphi_2 \\
\hline
\Gamma \vdash e_1 e_2 : T_1 :: \varphi_1; \varphi_2; \varphi
\]

K. Pun, M. Steffen  Deadlock Checking by Behaviour Inference for Lock Handling  NWPT’10  10 / 17
For our example:

\[
\text{let } t = \text{fn } (z_1,z_2). \ z_1.\text{lock};z_2.\text{lock};\text{stop} \\
\text{in} \\
\quad \text{let } x_1 = \text{new}_{\pi_1} L \text{ in} \\
\quad \text{let } x_2 = \text{new}_{\pi_2} L \text{ in} \\
\quad \text{spawn } (t \ (x_1,x_2)); t \ (x_2,x_1); \text{ stop}
\]

Effect:

\[
\varphi = \nu L^{\pi_1}; \nu L^{\pi_2}; \text{spawn } (L^{\pi_1}.\text{lock};L^{\pi_2}.\text{lock}); L^{\pi_2}.\text{lock}; L^{\pi_1}.\text{lock}
\]
\[ \Gamma, x : T_1 \vdash e : T_2 :: \varphi \]

**TE-Abs**

\[ \Gamma \vdash \text{fn } x : T_1 . e : T_1 \rightarrow^\varphi T_2 :: \epsilon \]

\[ \Gamma \vdash e_1 : T_2 \rightarrow^\varphi T_1 :: \varphi_1 \quad \Gamma \vdash e_2 : T_2 :: \varphi_2 \]

**TE-App**

\[ \Gamma \vdash e_1 e_2 : T_1 :: \varphi_1; \varphi_2; \varphi \]

\[ \Gamma \vdash e_1 : T_1 :: \varphi_1 \quad \Gamma, x : T_1 \vdash e_2 : T_2 :: \varphi_2 \]

**TE-Let**

\[ \Gamma \vdash \text{let } x : T_1 = e_1 \text{ in } e_2 : T_2 :: \varphi_1; \varphi_2 \]

\[ \Gamma \vdash e : T :: \varphi \]

**TE-Spawn**

\[ \Gamma \vdash \text{spawn } e : \text{Thread} :: \text{spawn } \varphi \]

\[ \Gamma \vdash \text{new}_\pi L : L^\pi :: \nu L^\pi \]

**TE-NewL**

\[ \Gamma \vdash \nu : L' :: \varphi \]

**TE-Lock**

\[ \Gamma \vdash \nu. \text{lock} : L' :: \varphi; L' \cdot \text{lock} \]
To detect a deadlock in a program, we execute the abstract behaviour of the program.

\[ E : p \langle \varphi \rangle \mid p \langle \varphi \rangle \parallel E \]

\[ \sigma : L^\pi \mapsto \{ \text{free, } p(n) \} \]
In our example:

```ocaml
let t = fn (z1,z2). z1.lock;z2.lock;stop
in
  let x1 = new π1 L in
  let x2 = new π2 L in
  spawn (t (x1,x2)); t (x2,x1); stop
```

We have the effect:

\[ \varphi = \nu L^{\pi_1}; \nu L^{\pi_2}; \text{spawn} (L^{\pi_1}.lock;L^{\pi_2}.lock); L^{\pi_2}.lock; L^{\pi_1}.lock \]
\[ \varphi = \nu L^1; \nu L^2; \text{spawn} (L^1.lock; L^2.lock); L^2.lock; L^1.lock \]

\[ \sigma \vdash t_0(\nu L^1; \nu L^2; \text{spawn} (L^1.lock; L^2.lock); L^2.lock; L^1.lock) \]
Deadlock Checking

\[ \varphi = \nu L^{\pi_1}; \nu L^{\pi_2}; \text{spawn} (L^{\pi_1}.\text{lock}; L^{\pi_2}.\text{lock}); L^{\pi_2}.\text{lock}; L^{\pi_1}.\text{lock} \]

\[ \sigma \vdash t_0 \langle \nu L^{\pi_1}; \nu L^{\pi_2}; \text{spawn} (L^{\pi_1}.\text{lock}; L^{\pi_2}.\text{lock}); L^{\pi_2}.\text{lock}; L^{\pi_1}.\text{lock} \rangle \]

\[ \sigma(L^{\pi}) = \text{undef} \quad \sigma' = \sigma[L^{\pi} \mapsto \text{free}] \]

\[ \sigma \vdash p\langle \nu L' \rangle \rightarrow \sigma' \vdash p\langle \epsilon \rangle \quad \text{RE-NEWL} \]
\( \varphi = \nu L^{\pi_1}; \nu L^{\pi_2}; \text{spawn} \left( L^{\pi_1}.\text{lock}; L^{\pi_2}.\text{lock} \right); L^{\pi_2}.\text{lock}; L^{\pi_1}.\text{lock} \)

\[
\begin{align*}
\sigma \vdash t_0 \langle \nu L^{\pi_1}; \nu L^{\pi_2}; \text{spawn} \left( L^{\pi_1}.\text{lock}; L^{\pi_2}.\text{lock} \right); L^{\pi_2}.\text{lock}; L^{\pi_1}.\text{lock} \rangle \\
\rightarrow \quad \sigma \vdash t_0 \langle \epsilon; \nu L^{\pi_2}; \text{spawn} \left( L^{\pi_1}.\text{lock}; L^{\pi_2}.\text{lock} \right); L^{\pi_2}.\text{lock}; L^{\pi_1}.\text{lock} \rangle
\end{align*}
\]

\[
\begin{align*}
\sigma(L^{\pi}) &= \text{undef} & \sigma' &= \sigma[L^{\pi} \mapsto \text{free}] \\
\sigma \vdash p \langle \nu L^{\pi} \rangle & \rightarrow \quad \sigma' \vdash p \langle \epsilon \rangle
\end{align*}
\]
$\varphi = \nu L^\pi_1; \nu L^\pi_2; \text{spawn } (L^\pi_1.\text{lock}; L^\pi_2.\text{lock}); L^\pi_2.\text{lock}; L^\pi_1.\text{lock}$

\[
\sigma \vdash t_0\langle \nu L^\pi_1; \nu L^\pi_2; \text{spawn } (L^\pi_1.\text{lock}; L^\pi_2.\text{lock}); L^\pi_2.\text{lock}; L^\pi_1.\text{lock}\rangle \\
\rightarrow \quad \sigma \vdash t_0\langle \epsilon; \nu L^\pi_2; \text{spawn } (L^\pi_1.\text{lock}; L^\pi_2.\text{lock}); L^\pi_2.\text{lock}; L^\pi_1.\text{lock}\rangle \\
\rightarrow \quad \sigma \vdash t_0\langle \epsilon; \text{spawn } (L^\pi_1.\text{lock}; L^\pi_2.\text{lock}); L^\pi_2.\text{lock}; L^\pi_1.\text{lock}\rangle
\]
\( \varphi = \nu L^{\pi_1}; \nu L^{\pi_2}; \text{spawn} (L^{\pi_1}.\text{lock}; L^{\pi_2}.\text{lock}); L^{\pi_2}.\text{lock}; L^{\pi_1}.\text{lock} \)

\[
\sigma \vdash t_0\langle \nu L^{\pi_1}; \nu L^{\pi_2}; \text{spawn} (L^{\pi_1}.\text{lock}; L^{\pi_2}.\text{lock}); L^{\pi_2}.\text{lock}; L^{\pi_1}.\text{lock} \rangle \\
\rightarrow \sigma \vdash t_0\langle \epsilon; \nu L^{\pi_2}; \text{spawn} (L^{\pi_1}.\text{lock}; L^{\pi_2}.\text{lock}); L^{\pi_2}.\text{lock}; L^{\pi_1}.\text{lock} \rangle \\
\rightarrow \sigma \vdash t_0\langle \epsilon; \text{spawn} (L^{\pi_1}.\text{lock}; L^{\pi_2}.\text{lock}); L^{\pi_2}.\text{lock}; L^{\pi_1}.\text{lock} \rangle
\]
\[ \varphi = \nu L^{\pi_1}; \nu L^{\pi_2}; \text{spawn} \left( L^{\pi_1}.\text{lock}; L^{\pi_2}.\text{lock} \right); L^{\pi_2}.\text{lock}; L^{\pi_1}.\text{lock} \]

\[
\begin{align*}
\sigma \vdash t_0 & \langle \nu L^{\pi_1}; \nu L^{\pi_2}; \text{spawn} \left( L^{\pi_1}.\text{lock}; L^{\pi_2}.\text{lock} \right); L^{\pi_2}.\text{lock}; L^{\pi_1}.\text{lock} \rangle \\
\rightarrow \quad \sigma \vdash t_0 & \langle \epsilon; \nu L^{\pi_2}; \text{spawn} \left( L^{\pi_1}.\text{lock}; L^{\pi_2}.\text{lock} \right); L^{\pi_2}.\text{lock}; L^{\pi_1}.\text{lock} \rangle \\
\rightarrow \quad \sigma \vdash t_0 & \langle \epsilon; \text{spawn} \left( L^{\pi_1}.\text{lock}; L^{\pi_2}.\text{lock} \right); L^{\pi_2}.\text{lock}; L^{\pi_1}.\text{lock} \rangle \\
\rightarrow \quad \sigma \vdash t_0 & \langle L^{\pi_2}.\text{lock}; L^{\pi_1}.\text{lock} \rangle \parallel t \langle L^{\pi_1}.\text{lock}; L^{\pi_2}.\text{lock} \rangle
\end{align*}
\]
\[ \varphi = \nu L^{\pi_1}; \nu L^{\pi_2}; \text{spawn} (L^{\pi_1}.\text{lock}; L^{\pi_2}.\text{lock}); L^{\pi_2}.\text{lock}; L^{\pi_1}.\text{lock} \]

\[ \begin{align*}
\sigma \vdash t_0\langle \nu L^{\pi_1}; \nu L^{\pi_2}; \text{spawn} (L^{\pi_1}.\text{lock}; L^{\pi_2}.\text{lock}); L^{\pi_2}.\text{lock}; L^{\pi_1}.\text{lock} \rangle \\
\rightarrow \sigma \vdash t_0\langle \epsilon; \nu L^{\pi_2}; \text{spawn} (L^{\pi_1}.\text{lock}; L^{\pi_2}.\text{lock}); L^{\pi_2}.\text{lock}; L^{\pi_1}.\text{lock} \rangle \\
\rightarrow \sigma \vdash t_0\langle \epsilon; \text{spawn} (L^{\pi_1}.\text{lock}; L^{\pi_2}.\text{lock}); L^{\pi_2}.\text{lock}; L^{\pi_1}.\text{lock} \rangle \\
\rightarrow \sigma \vdash t_0\langle L^{\pi_2}.\text{lock}; L^{\pi_1}.\text{lock} \rangle \parallel t\langle L^{\pi_1}.\text{lock}; L^{\pi_2}.\text{lock} \rangle
\end{align*} \]

\[ \pi \in r \quad \sigma(L^{\pi}) = \text{free} \lor \sigma(L^{\pi}) = p(n) \quad \sigma' = \sigma[L^{\pi} \mapsto \sigma(L^{\pi}) + 1] \]

\[ \frac{\sigma \vdash p\langle L'.\text{lock} \rangle}{\sigma' \vdash p\langle \epsilon \rangle} \quad \text{RE-LOCK} \]

K. Pun, M. Steffen  
Deadlock Checking by Behaviour Inference for Lock Handling  
NWPT’10 13 / 17
\[ \varphi = \nu L^{\pi_1}; \nu L^{\pi_2}; \text{spawn} (L^{\pi_1}.\text{lock}; L^{\pi_2}.\text{lock}); L^{\pi_2}.\text{lock}; L^{\pi_1}.\text{lock} \]

\[
\sigma \vdash t_0(\nu L^{\pi_1}; \nu L^{\pi_2}; \text{spawn} (L^{\pi_1}.\text{lock}; L^{\pi_2}.\text{lock}); L^{\pi_2}.\text{lock}; L^{\pi_1}.\text{lock}) \nabla t_0(\epsilon; \nu L^{\pi_2}; \text{spawn} (L^{\pi_1}.\text{lock}; L^{\pi_2}.\text{lock}); L^{\pi_2}.\text{lock}; L^{\pi_1}.\text{lock}) \nabla t_0(\epsilon; \text{spawn} (L^{\pi_1}.\text{lock}; L^{\pi_2}.\text{lock}); L^{\pi_2}.\text{lock}; L^{\pi_1}.\text{lock}) \]

\[
\pi \in r \quad \sigma(L^{\pi}) = \text{free} \lor \sigma(L^{\pi}) = p(n) \quad \sigma' = \sigma[L^{\pi} \mapsto \sigma(L^{\pi}) + 1] \]

\[
\sigma \vdash p(L^{\prime}.\text{lock}) \rightarrow \sigma' \vdash p(\epsilon) \quad \text{RE-Lock}
\]
\[ \varphi = \nu L^{\pi_1}; \nu L^{\pi_2}; \text{spawn}(L^{\pi_1}.\text{lock}; L^{\pi_2}.\text{lock}); L^{\pi_2}.\text{lock}; L^{\pi_1}.\text{lock} \]

\[ \sigma \vdash t_0(\nu L^{\pi_1}; \nu L^{\pi_2}; \text{spawn}(L^{\pi_1}.\text{lock}; L^{\pi_2}.\text{lock}); L^{\pi_2}.\text{lock}; L^{\pi_1}.\text{lock}) \]
\[ \rightarrow \quad \sigma \vdash t_0(\epsilon; \nu L^{\pi_2}; \text{spawn}(L^{\pi_1}.\text{lock}; L^{\pi_2}.\text{lock}); L^{\pi_2}.\text{lock}; L^{\pi_1}.\text{lock}) \]
\[ \rightarrow \quad \sigma \vdash t_0(\epsilon; \text{spawn}(L^{\pi_1}.\text{lock}; L^{\pi_2}.\text{lock}); L^{\pi_2}.\text{lock}; L^{\pi_1}.\text{lock}) \]
\[ \rightarrow \quad \sigma \vdash t_0(\text{L}^{\pi_2}.\text{lock}; L^{\pi_1}.\text{lock}) \parallel t(\text{L}^{\pi_1}.\text{lock}; L^{\pi_2}.\text{lock}) \]
\[ \rightarrow \quad \sigma[L^{\pi_2} \mapsto t_0(1)] \vdash t_0(\epsilon; L^{\pi_1}.\text{lock}) \parallel t(\text{L}^{\pi_1}.\text{lock}; L^{\pi_2}.\text{lock}) \]

\[ \pi \in r \quad \sigma(L^{\pi}) = \text{free} \lor \sigma(L^{\pi}) = p(n) \quad \sigma' = \sigma[L^{\pi} \mapsto \sigma(L^{\pi}) + 1] \]

\[ \sigma \vdash p(L^{\pi}.\text{lock}) \rightarrow \sigma' \vdash p(\epsilon) \]

\[ \text{RE-Lock} \]
\[ \varphi = \nu L^{\pi_1}; \nu L^{\pi_2}; \text{spawn} (L^{\pi_1}.\text{lock}; L^{\pi_2}.\text{lock}); L^{\pi_2}.\text{lock}; L^{\pi_1}.\text{lock} \]

\[
\sigma \vdash t_0 \langle \nu L^{\pi_1}; \nu L^{\pi_2}; \text{spawn} (L^{\pi_1}.\text{lock}; L^{\pi_2}.\text{lock}); L^{\pi_2}.\text{lock}; L^{\pi_1}.\text{lock} \rangle \\
\rightarrow \sigma \vdash t_0 \langle \epsilon; \nu L^{\pi_2}; \text{spawn} (L^{\pi_1}.\text{lock}; L^{\pi_2}.\text{lock}); L^{\pi_2}.\text{lock}; L^{\pi_1}.\text{lock} \rangle \\
\rightarrow \sigma \vdash t_0 \langle \epsilon; \text{spawn} (L^{\pi_1}.\text{lock}; L^{\pi_2}.\text{lock}); L^{\pi_2}.\text{lock}; L^{\pi_1}.\text{lock} \rangle \\
\rightarrow \sigma \vdash t_0 \langle L^{\pi_2}.\text{lock}; L^{\pi_1}.\text{lock} \rangle \parallel t \langle L^{\pi_1}.\text{lock}; L^{\pi_2}.\text{lock} \rangle \\
\rightarrow \sigma[L^{\pi_2} \mapsto t_0(1)] \vdash t_0 \langle \epsilon; L^{\pi_1}.\text{lock} \rangle \parallel t \langle L^{\pi_1}.\text{lock}; L^{\pi_2}.\text{lock} \rangle \\
\rightarrow \sigma[L^{\pi_2} \mapsto t_0(1), L^{\pi_1} \mapsto t(1)] \vdash t_0 \langle L^{\pi_1}.\text{lock} \rangle \parallel t \langle L^{\pi_2}.\text{lock} \rangle \]
\[ \sigma \vdash t_0\langle L^\pi_2 . \text{lock}; L^\pi_1 . \text{lock} \rangle \parallel t\langle L^\pi_1 . \text{lock}; L^\pi_2 . \text{lock} \rangle \]

\[ \rightarrow \sigma[L^\pi_2 \mapsto t_0(1), L^\pi_1 \mapsto t(1)] \vdash t_0\langle L^\pi_1 . \text{lock} \rangle \parallel t\langle \epsilon; L^\pi_2 . \text{lock} \rangle \]

OR

\[ \rightarrow \sigma[L^\pi_1 \mapsto t(1), L^\pi_2 \mapsto t_0(1)] \vdash t_0\langle \epsilon; L^\pi_1 . \text{lock} \rangle \parallel t\langle L^\pi_2 . \text{lock} \rangle \]

...
Operational Semantics for Abstract Behaviour

\[
\begin{align*}
\sigma(L^\pi) &= \text{undef} \quad \sigma' = \sigma[L^\pi \mapsto \text{free}] \\
\sigma &\vdash p(\nu L^\prime) \rightarrow \sigma' \vdash p(\epsilon) \\
\pi \in r \quad \sigma(L^\pi) &= \text{free} \lor \sigma(L^\pi) = p(n) \\
\sigma' &= \sigma[L^\pi \mapsto \sigma(L^\pi) + 1] \\
\sigma &\vdash p(L^\prime \cdot \text{lock}) \rightarrow \sigma' \vdash p(\epsilon) \\
\sigma &\vdash p_1(\langle \text{spawn } \varphi \rangle; \varphi') \rightarrow \sigma \vdash p_1(\varphi') \parallel p_2(\varphi)
\end{align*}
\]

RE-NewL

RE-Lock

RE-Spawn
Summary

Conclusion:
- Type and effect system for locks
- Algorithm for inferring behaviour of locks
- Program points abstract concrete locks
- Effect analysis for potential deadlock

Future Work:
- Applying to communication analysis of asynchronous systems
- Relaxing the condition (e.g. lock creation in loop)
- Abstracting processes
Detecting potential deadlocks with state analysis and run-time monitoring.

Ownership types for safe programming: Preventing data races and deadlocks.
In Object Oriented Programming: Systems, Languages, and Applications (OOPSLA) ’02 (Seattle, USA). ACM.
In SIGPLAN Notices.

System deadlocks.

Finite-state call-chain abstractions for deadlock detection in multithreaded object-oriented languages (extended abstract).

Effective, static detection of race conditions and deadlocks: RacerX.

Static deadlock detection for Java libraries.