Types for leak detection in a session-oriented functional programming language

Candidato:
Andrea Tosatto

Relatore:
Dott. Luca Padovani

Corelatore:
Prof. Viviana Bono

Tesi di Laurea Triennale in Informatica
Sing#: Copyless message passing

Exchange heap ($\mu$)

Threads

$\langle E_1 \rangle$

msg

queue

$\langle E_2 \rangle$
Sing# : Copyless message passing

Exchange heap ($\mu$)

Threads

$\langle E_1 \rangle$

$\langle E_2 \rangle$

queue

msg
Sing#: Copyless message passing

```
⟨E₁⟩  Threads  ⟨E₂⟩
```

Exchange heap ($\mu$)

Queue

msg
A dangerous communication model

\[
\text{let } (a, b) = \text{open}() \text{ in } \\
\text{close} (\text{send} \ a \ b)
\]
let (a, b) = open () in close (send a b)

\langle E \rangle
A dangerous communication model

```
let (a, b) = open () in
close (send a b)
```
A dangerous communication model

```ocaml
let (a, b) = open () in
  close (send a b)
```
A dangerous communication model

\[
\text{let } (a, b) = \text{open () in close (send a b)}
\]
Endpoint Types

?Bool.end

!Bool.?Bool.end

?T.end
Types

Type  \( t ::= \)

- \( \text{Bool} \)
- \( \text{Unit} \)
- \( t \rightarrow t \)
- \( t \odot t \)
- \( t \otimes t \)
- \( T \)

Endpoint Type  \( T ::= \)

- \( \text{end} \) (termination)
- \( !t.T \) (input)
- \( ?t.T \) (output)
- \( \alpha \) (type variable)
- \( \text{rec } \alpha.T \) (recursive type)
Looking at types for detecting leaks

```plaintext
let (a, b) = open () in close (send a b)
```

<table>
<thead>
<tr>
<th>$t$ ::=</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{Bool}$</td>
</tr>
<tr>
<td>$\text{Unit}$</td>
</tr>
<tr>
<td>$t \rightarrow t$</td>
</tr>
<tr>
<td>$t \circ t$</td>
</tr>
<tr>
<td>$t \otimes t$</td>
</tr>
<tr>
<td>$T$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$T$ ::=</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{end}$</td>
</tr>
<tr>
<td>$!t.T$</td>
</tr>
<tr>
<td>$?t.T$</td>
</tr>
<tr>
<td>$\alpha$</td>
</tr>
<tr>
<td>$\text{rec } \alpha.T$</td>
</tr>
</tbody>
</table>
Looking at types for detecting leaks

let (a, b) = open () in
    close (send a b)

\[ T = !S.\text{end} \]
Looking at types for detecting leaks

```plaintext
let (a, b) = open () in
  close (send a b)

T = !S.end

S = ?S.end
```

\[
\begin{align*}
t & ::= \text{Bool} \\
& \mid \text{Unit} \\
& \mid t \to t \\
& \mid t \bowtie t \\
& \mid T \\
T & ::= \text{end} \\
& \mid !t.T \\
& \mid ?t.T \\
& \mid \alpha \\
& \mid \text{rec } \alpha.T
\end{align*}
\]
A problematic example

```
let (a, b) = open () in
    close (send a (g b))
```
A problematic example

```
let (a, b) = open () in
  close (send a (g b))
```

\(\langle E \rangle\)
A problematic example

\[
\text{let } (a, b) = \text{open } () \text{ in close (send } a \ (g \ b))
\]
A problematic example

let (a, b) = open () in
  close (send a (g b))
A problematic example

```ocaml
let (a, b) = open () in
close (send a (g b))
```

Types for leak detection in a session-oriented functional programming language
Andrea Tosatto
A problematic example

```ocaml
let (a, b) = open () in
close (send a (g b))
```

Types for leak detection in a session-oriented functional programming language
Andrea Tosatto
A problematic example

```ocaml
let (a, b) = open () in
  close (send a (g b))
```

Unit $\rightarrow$ Unit

$\langle E \rangle$
A problematic example

\[
\begin{align*}
\text{let } (a, b) &= \text{open } () \text{ in} \\
&\text{close } (\text{send } a (g \ b)) \\
\end{align*}
\]

\[
\begin{align*}
g &= \lambda x. \lambda y. (\text{fix}(\lambda f. \lambda z. f \ z)x)
\end{align*}
\]
Type weight (1)

- $\| t \| = \text{“maximum length of chains of pointers starting from object of type } t\text{”}$
- only pointers whose type has finite weight can be sent

\[
\begin{align*}
T &= \text{?Bool.end} & \| T \| &= 1 \\
T &= \text{?T.end} & \| T \| &= \infty \\
t &= s_1 \rightsquigarrow s_2 & \| t \| &= ?
\end{align*}
\]
Type weight (1)

- $\| t \| = \text{“maximum length of chains of pointers starting from object of type } t\text{“}$
- only pointers whose type has finite weight can be sent

\[
T = \text{?Bool.end} \quad \| T \| = 1
\]

\[
T = \text{?T.end} \quad \| T \| = \infty
\]

\[
t = s_1 \rightarrow s_2 \quad \| t \| = ?
\]
Type weight (1)

- $\| t \| = \text{"maximum length of chains of pointers starting from object of type } t\text{"}$
- only pointers whose type has finite weight can be sent

\[
T = \text{?} \text{Bool.end} \quad \| T \| = 1
\]

\[
T = \text{?} T.\text{end} \quad \| T \| = \infty
\]

\[
t = s_1 \rightarrow s_2 \quad \| t \| = ?
\]
Type weight (1)

- \( \| t \| = \) “maximum length of chains of pointers starting from object of type \( t \)”
- only pointers whose type has finite weight can be sent

\[
T = ?\text{Bool.end} \quad \| T \| = 1
\]

\[
T = ?T.\text{end} \quad \| T \| = \infty
\]

\[
t = s_1 \rightarrow s_2 \quad \| t \| = ?
\]
Type weight (1)

- \( \| t \| = \text{“maximum length of chains of pointers starting from object of type } t \text{”} \)
- only pointers whose type has finite weight can be sent

\[
T = \text{?} \text{Bool.end} \quad \| T \| = 1
\]

\[
T = \text{?} T . \text{end} \quad \| T \| = \infty
\]

\[
t = s_1 \xrightarrow{w} s_2 \quad \| t \| = w
\]
Type weight (2)

\[
\begin{align*}
(T\text{-AbsW}) \\
\Gamma, x : t \vdash E : s & \quad \forall u \in \text{dom}(\Gamma) \quad \parallel \Gamma(u) \parallel \leq w \\
\hline
\Gamma \vdash \lambda x. E : t \wedge s
\end{align*}
\]
Type weight (2)

\[(\text{T-AbsW})\]
\[
\Gamma, x : t \vdash E : s \quad \forall u \in \text{dom}(\Gamma) \quad \parallel \Gamma(u) \parallel \leq w
\]
\[
\Gamma \vdash \lambda x. E : t \overset{w}{\rightarrow} s
\]
let (a, b) = open () in
    close (send a (g b))
Type weight (3)

\[
\text{let } (a, b) = \text{open } () \text{ in } \\
\text{close (send a (g b))}
\]

\[
\text{Unit} \xrightarrow{\infty} \text{Unit}
\]
let (a, b) = open () in
  close (send a (g b))

\[
\begin{align*}
\text{Unit} & \xrightarrow{\infty} \text{Unit} \\
(T\text{-SEND}) \\
\Gamma_1 \vdash E_1 : !t.T & \quad \Gamma_2 \vdash E_2 : t \quad \| t \| < \infty \\
\Gamma_1 + \Gamma_2 \vdash \text{send } E_1 \ E_2 : T
\end{align*}
\]
Conclusions

We have deployed a type system to prevent memory leaks...

...BUT...

Does it really work?!

Theorem (Soundness)

If E is well typed, then E is leak free.

Possible extensions

- Subtyping...
Conclusions

We have deployed a type system to prevent memory leaks... … BUT...

Does it really work?!

Theorem (Soundness)

If $E$ is well typed, then $E$ is leak free.

Possible extensions

• Subtyping...
Conclusions

We have deployed a type system to prevent memory leaks...

...BUT...

Does it really work?!

Theorem (Soundness)

\[ \text{If } E \text{ is well typed, then } E \text{ is leak free.} \]

Possible extensions

- Subtyping...
Conclusions

We have deployed a type system to prevent memory leaks...

...BUT...

Does it really work?!

Theorem (Soundness)

If $E$ is well typed, then $E$ is leak free.

Possible extensions

- Subtyping...
Grazie dell’attenzione!