Deadlock Freedom for Session Types Using Separation Logic

Jules Jacobs

Radboud University Nijmegen
julesjacobs@gmail.com

Overview

- Channels for concurrency
- Operational semantics
- Session types
- Typing channel references
- Typing channel references with separation logic
- The connectivity tree
- Deadlock freedom

Channels for concurrency

- Lambda calculus + channels
- Fork: `let c' = fork(fun c => ...)`
- Send: `let c' = send(c, 3)`
- Receive: `let (c',x) = recv(c)`
- Close: `let () = close(c)`

Channels for concurrency

Example:

```
let c' = fork(fun c =>
      send(c, 3)
      let k = recv(c)
      if k < 10 then send(c,"hello") else send(c,"hi")
      close(c)
    )

let n = recv(c')
send(c', 2*n)
let msg = recv(c')
close(c')
print(msg)
```

Potential bugs

Safety bugs:

- Wrong type of message, e.g. string instead of int
- Example:
- Using a channel after close
- Example:

Concurrency bugs:

- Not sending a message (receive blocks $\implies$ deadlock)
- Both sides do a receive
- Example:

Leaks:

- Not closing a channel (leak)
- Example:

Session types

Existing system, citation: Honda et al. 1900

Session types

- Linear lambda calculus
- Channels have protocol types: sequence of send (!) receive (?) and End
- Example: `!Int, ?Int, !String, End`
- Dual: `?Int, !Int, ?String, End`
- If `c : !T, R` and `x : T` then `send(c,x) : R`
- If `c : ?T, R` then `recv(c,x) : R x T`

State passing

Example:

```
let c' = fork(fun c =>
      let c = send(c, 3)
      let (c,k) = recv(c)
      if k < 10 then
        let c = send(c,"hello")
        close(c)
      else
        let c = send(c,"hi")
        close(c)
    )
let (c',n) = recv(c')
let c' = send(c', 2*n)
let (c',msg) = recv(c')
close(c')
print(msg)
```

Session types prevent bugs

Safety bugs:

- Wrong type of message, e.g. string instead of int
- Example:
- Using a channel after close
- Example:

Concurrency bugs:

- Not sending a message (receive blocks $\implies$ deadlock)
- Both sides do a receive
- Example:

Leaks:

- Not closing a channel (leak)
- Example:

Problem statement

- Deadlock freedom for session types has not been mechanised
- Lambda calculus
- Cut elimination

Key ideas:

- Research accounting via separation logic
- Connectivity graphs for deadlocks

Operational semantics

- Small-step operational semantics
- State: threads $\times$ buffers
- Buffers indexed by addresses (e.g. natural numbers); two buffers per channel
- Representation of a channel is address $\times$ bool, e.g. `send(#(324, true))`
- `send(#(324,true), "hello")` $\mapsto$ `#(324,true)`
- Send adds message to the other party's buffer
- Receive takes a message out of its own buffer
- Fork allocates a new pair of buffers, returns `#(n,true)` to one side and passes `#(n,false)` to the other
- Close deallocates its own buffer

```
endpoint = nat x bool
heap = endpoint -> list val
local_step : expr x heap -> expr x heap -> Prop
step : list expr x heap -> list expr x heap -> Prop
```

Formal properties

Formal properties:

- Safety
- Deadlock freedom
- Leak freedom

Theorem

Typed expr -¿ previous 3 properties

Progress & preservation

Formal properties

High level approach:

- Progress & preservation (explain this) $\implies$ safety
- Progress & preservation for linear types
- Connectivity tree

Typing channel references

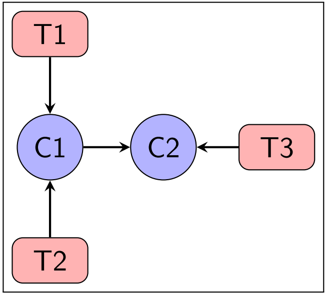
- When the operational semantics takes steps, channel references (e.g. `#(324, true)`) enter into the program.
- How do we type them?
- We need a typing rule for channel references.
- We need an environment Σ for channel references, similar to Γ for variables
- If $((342, \mathit{true}), T) \in \Sigma$ then `#(324,true) : T`

Typing channel references with separation logic

How to explain this...?

The connectivity graph

- A graph containing threads (boxes) and channels (circles)
- An arrow $T \rightarrow C$ means that thread T has a reference to channel C
  - This means that some data structure that T holds contains a channel reference to C
- An arrow $C1 \rightarrow C2$ means that channel $C1$ has a reference to channel $C2$
  - This means that somewhere in the buffer of $C1$ there is a reference to $C2$



Deadlock freedom

- Invariant: the connectivity graph is a tree.
- Even when channel references are put in buffers.
- Ensured by linearity.

Deadlock freedom

Theorem

There is a thread that's not blocked.

Proof.

We find a thread that's not blocked using this algorithm:

- Start at any thread in the connectivity graph.
- If the thread is not blocked, we're done.
- If the thread is blocked, step to the channel that it's blocked on.
- If we're at a channel

□