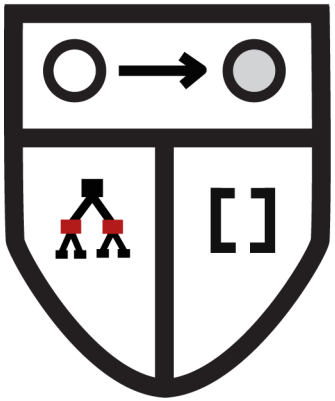


CS 3110

Promises

Nate Foster
Spring 2019

Today's music: *Call Me Maybe* by Carly Rae Jepsen

Review

Previously in 3110: Advanced data structures

- Streams
- Balanced binary trees
- Mutable data structures

Today:

- Promises: a data structure and programming paradigm for concurrency

Concurrency

- Networks have multiple computers
- Computers have multiple processors
- Processors have multiple cores

...all working semi-independently
...all sharing resources

sequential: non-overlapping in duration
concurrent: overlapping in duration

- **parallel:** happening at the same time
- **interleaved:** rapidly switching between

Concurrency

At any given time, my laptop is...

- Streaming music
- Running a web server
- Syncing with web services
- Running OCaml

The OS plays a big role in making it look like those all happen simultaneously

Concurrency

Applications might also want concurrency:

- Web server that handles many clients at once
- Scientific calculations that exploit parallel architecture to get speedup
- GUIs that want to respond to users while doing computation (e.g., rendering) in the background

Programming models for concurrency

Threads: sequential code for computation
Pthreads, OpenMP, java.lang.Thread
OCaml **`Thread`**

Promises: values that are promised to be computed
async/await in JavaScript and .NET, java.util.concurrent.Future, Clojure, Scala
OCaml **`Async`** and **`Lwt`**

(and many others)

PROMISES

Promises

Computation that promises to produce a value sometime in the future

Aka:

- future
- delayed
- deferred

Lwt: OCaml library for promises

Promises

A promise – `'a Lwt.t` – is like a box:

- It starts out empty
- At some point in the future, it could be filled with a value of type `'a`
- Once it's filled, the box's contents can never be changed ("write once")

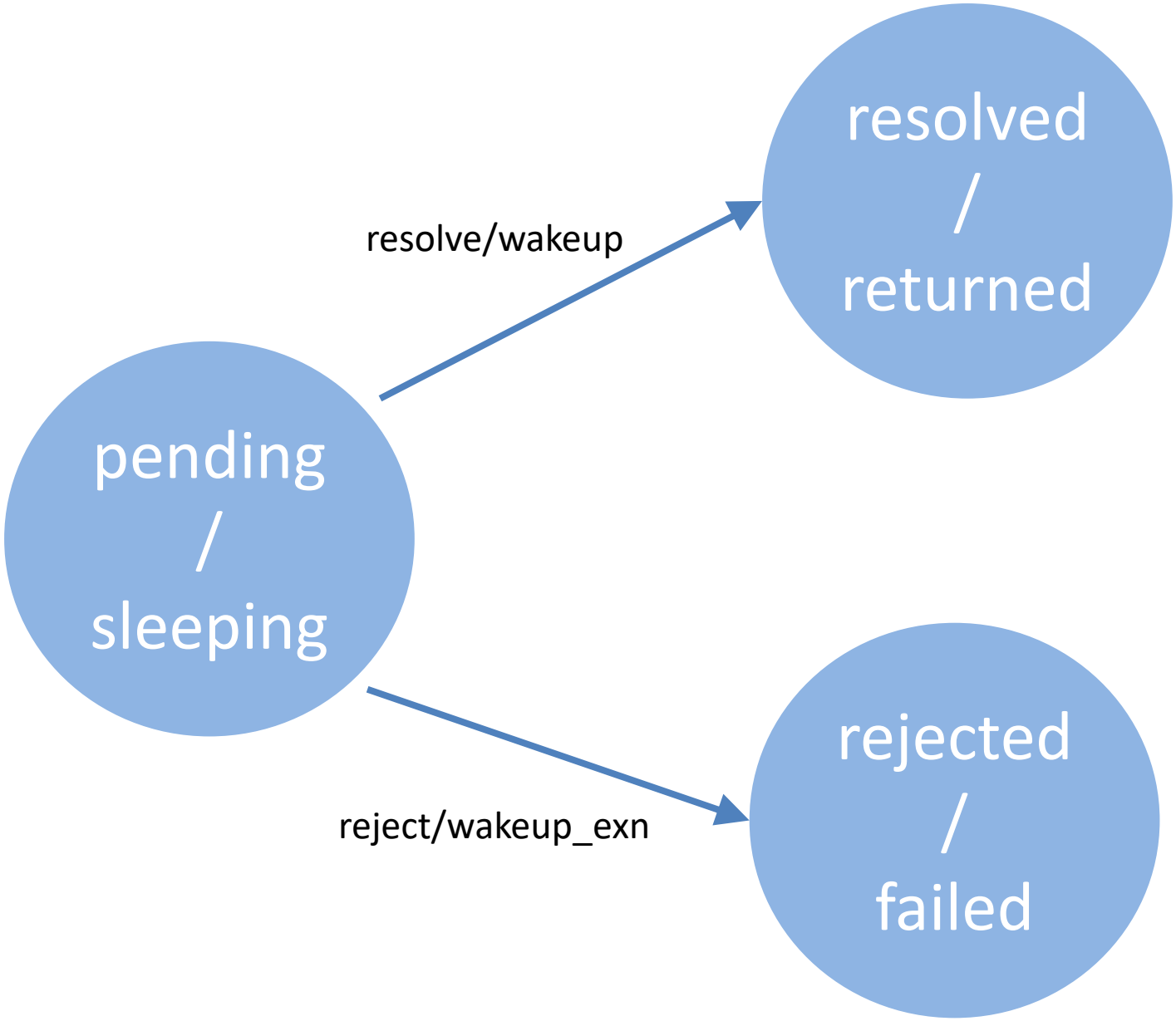
Resolver

A resolver – **`'a Lwt.u`** – is what fills the box

Terminology:

- promise is **pending** aka sleeping: box is empty
- promise is **resolved** aka returned: box is full
- promise is **rejected** aka failed: box contains exn

Demo



Discussion: implement signature for promises

Digression on Cornell history

- ivars = promises+resolvers
- Used for parallel computing in language called Id [Arvind, Nikhil, and Pingali 1986]
  - Keshav Pingali, Cornell CS prof 1986-2006?
- Implemented in *Concurrent ML* by John Reppy (Cornell PhD 1992)

Lwt

Typical use of library is to do asynchronous I/O

- Launch an I/O operation as a promise
- OS helps to resolve promise

Source of parallelism: OS, not OCaml

Demo

call me maybe?

CALLBACKS

Managing Promises

What if program has many promises "in flight"?

- Web server handling many client
- Spreadsheet updating many cells
- Game updating many enemies

Need a way to manage dependencies of computations upon promises…

Demo

```
bind promise callback

bind :
'a Lwt.t
-> ('a -> 'b Lwt.t)
-> 'b Lwt.t
```

```
promise >>= callback
```

```
(>>=) :
'a Lwt.t
-> ('a -> 'b Lwt.t)
-> 'b Lwt.t
```

Demo

Implementing bind

- Store a list of callbacks with each promise
- After promise is resolved, Lwt runs callbacks
- If promise never resolved (or fails), no callback

Callback execution

- Single-threaded: one callback runs at a time
- Cooperative: callback runs to completion
- Nondeterministic: unspecified which runs first

Upcoming events

- [Tomorrow] A5 released

This is resolved.

THIS IS 3110