Channeling Work

Steve Powell Rob Harrop

November 17, 2010

Abstract

A short specification arising from Rob's investigation into serving messages on a collection of channels with a limited number of threads, preserving ordering constraints.

Contents

1 Introduction

The primitives in this description are *Channel*s and items of *Work*. It is assumed that work is a message transfer or acknowledgement of some kind. It is not important. What is important is that a series of items of *Work* needs to be done for a *Channel* and we cannot allow two items of work to be processed for the same *Channel* at the same time. So we introduce the primitive sets:

$$[\mathit{Channel}, \mathit{Work}]$$

and with these we can describe the general state.

2 The general state of things

The basic state of the system is a collection of *Channel*s with a sequence of items of work associated with each:

Pool

$$\mathit{pool} : \mathit{Channel} \mathrel{+\!\!\!\!\rightarrow} \operatorname{seq} \mathit{Work}$$

The *pool* of known channels ($\operatorname{dom} \mathit{pool}$) is partitioned into those which are *dormant*, those *ready* for work to be done, and those which are currently being processed (*inprogress*).

We define a convenience schema for each partition. This is to name them for use in explicitly stating preservation later.

The *dormant* ones have no work (but we cannot impose this constraint without the *pool* – we do it later):

Dormant

$$\mathit{dormant} : \mathbb{P}\, \mathit{Channel}$$

the *ready* ones are ordered:

Ready

$$\mathit{ready} : \operatorname{iseq} \mathit{Channel}$$

and the rest are 'in progress':

InProgress

$$\mathit{inprogress} : \mathbb{P}\, \mathit{Channel}$$

2.1 The state of the union

We can now assemble the entire system state as follows:

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{State}} \\ \hline
\textit{Pool} \\
\textit{Dormant} \\
\textit{Ready} \\
\textit{InProgress} \\ \hline
\langle \textit{dormant}, \operatorname{ran} \textit{ready}, \textit{inprogress} \rangle \ \mathsf{partition} \ \operatorname{dom} \textit{pool} \\
\forall\, c : \textit{dormant} \bullet \textit{pool}\ c = \langle\rangle \\ \hline
\end{array}$$

where we make explicit that these channel collections partition those known (in dom *pool*), and can also impose the constraint that the *dormant* channels have no work.

3 Work delivery

In general, as it comes in, work is added to the sequence of items associated with a channel in the *pool*. However, the precise change of state depends upon which partition the channel is in at the time.

We therefore define three 'deliver work' state transitions; one for each partition. In each case different partitions change as a result. They share the same signature, and underlying pool change, however:

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{DeliverWorkCommon}} \\ \hline
\Delta \textit{State} \\
w? : \textit{Work} \\
c? : \textit{Channel} \\ \hline
c? \in \operatorname{dom} \textit{pool} \\
\textit{pool}' = \textit{pool} \oplus \{ c? \mapsto \textit{pool}\ c? \frown \langle w? \rangle \} \\ \hline
\end{array}$$

each of them changes the *State*, and takes an item of work and a channel as input. In every case, the work is added to the *pool*.

When the channel is dormant it moves into the ready queue (and the 'in progress' partition remains unchanged):

DeliverDormant

$DeliverWorkCommon$
$\Xi InProgress$

$c? \in dormant$
$dormant' = dormant \setminus \{c?\}$
$ready' = ready \frown \langle c? \rangle$

(Note that in this case the resulting sequence of work is simply $\langle w? \rangle$.)

When the channel is already 'in progress' the partitions stay unchanged:

DeliverInProgress

$DeliverWorkCommon$
$\Xi InProgress$
$\Xi Dormant$
$\Xi Ready$

$c? \in inprogress$

(and we need say nothing more).

When the channel is ready (in the *ready* queue) the partitions stay unchanged as well:

DeliverReady

$DeliverWorkCommon$
$\Xi InProgress$
$\Xi Dormant$
$\Xi Ready$

$c? \in \operatorname{ran} ready$

(and we need say nothing more).

Since the preconditions of the *Deliver* schemas are disjoint, we may combine them without introducing any further non-determinism:

$$DeliverWork \mathrel{\widehat{=}} DeliverDormant \lor DeliverInProgress \lor DeliverReady$$

4 Start work

When there is time (and available computing resources) we can start some work. We consider only doing one piece of work at a time, though it is easy to consider 'batching' work items together.

The operation that starts it picks a piece of work to do, and gives the work and channel as outputs. The dormant channels are unaffected, and the channel moves from ready to 'in progress':

$$
\begin{array}{|l}
\textit{StartWork} \\
\hline
\Delta \mathit{State} \\
\Xi \mathit{Dormant} \\
c! : \mathit{Channel} \\
w! : \mathit{Work} \\
\hline
\mathit{ready} \neq \langle\rangle \\
\langle c! \rangle \frown \mathit{ready}' = \mathit{ready} \\
\mathit{inprogress}' = \mathit{inprogress} \cup \{c!\} \\
\langle w! \rangle \frown \mathit{pool}'\ c! = \mathit{pool}\ c! \\
\{c!\} \ntriangleleft \mathit{pool}' = \{c!\} \ntriangleleft \mathit{pool} \\
\hline
\end{array}
$$

where the channel simply gets 'taken' from the front of the ready queue; the work gets 'taken' from the front of the work queue for the channel; the channel gets put in the 'in progress' partition and no other channels are affected.

Notice that here, although apparently there might be no work left, the channel is not put into the *dormant* partition until the current work is completed. It is quite in order for non-*dormant* channels to have no work *pro tem*, though we expect such a channel to be placed in *dormant* eventually.

5 Our work is done

After the work is completed the channel can be taken out of 'in progress'. Of course, there might already be more work to do (or not) and these cases are distinguished.

We define a 'partial operation' to identify the essence of work completion:

$$
\begin{array}{|l}
\textit{EndWorkCommon} \\
\hline
\Delta \mathit{State} \\
\Xi \mathit{Pool} \\
c? : \mathit{Channel} \\
\hline
c? \in \mathit{inprogress} \\
\mathit{inprogress}' = \mathit{inprogress} \setminus \{c?\} \\
\mathit{pool}' = \mathit{pool} \\
\hline
\end{array}
$$

This only happens for channels in *inprogress* and this channel is always removed from there. None of the work queues change as a result of this transition.

If there is no more work to do, the channel becomes dormant (and the ready queue remains unchanged):

$$
\begin{array}{|l}
\hline
\mathit{EndWorkNoMoreToDo} \\
\mathit{EndWorkCommon} \\
\Xi \mathit{Ready} \\
\hline
\mathit{pool}\ c? = \langle\rangle \\
\mathit{dormant}' = \mathit{dormant} \cup \{c?\} \\
\hline
\end{array}
$$

(We could have deduced the dormant relation – and, as it happens, the pool constraint – from the core work and the fact that the ready queue does not change.)

If there *is* more work to do, the channel becomes *ready* (though not so ready as some *al*ready):

$$
\begin{array}{|l}
\hline
\mathit{EndWorkMoreToDo} \\
\mathit{EndWorkCommon} \\
\Xi \mathit{Dormant} \\
\hline
\mathit{pool}\ c? \neq \langle\rangle \\
\mathit{ready}' = \mathit{ready} \frown \langle c? \rangle \\
\hline
\end{array}
$$

The channel is placed on the 'end' of the ready queue.

Should we be so inclined, since the preconditions of the two *EndWork* schemas are disjoint, we can unambiguously combine them:

$$\mathit{EndWork} \mathrel{\widehat=} \mathit{EndWorkMoreToDo} \lor \mathit{EndWorkNoMoreToDo}$$

6 To begin with...

After constructing this description, we note that there is no channel creation nor deletion described, and that the initial state is not given. This is an ideal opportunity to record our intentions in these areas. We can even document the rules for pieces of work left over when a channel 'crashes', in some way, or if the item of work 'fails'.

We limit this brief note to talk about the initial state.

In the absence of channel creation or deletion, we will provide the set of channels initially:

$$\begin{array}{|l}
\textit{InitialState} \\
\hline
\textit{State}' \\
\textit{cs}? : \mathbb{P}\ \textit{Channel} \\
\hline
\textit{dormant}' = \textit{cs}? \\
\textit{pool}' = \{c : \textit{cs}? \bullet c \mapsto \langle\rangle\}
\end{array}$$

The set of channels is precisely the dormant set initially, and the pool records no work at all. The partition constraint on *State* ensures that the rest of the initial state is determined:

$$\textit{InitialState} \vdash \textit{ready}' = \langle\rangle \land \textit{inprogress}' = \varnothing$$

7 Summary

Essentially, this simple arrangement makes sure that pieces of work for the same channel never overtake each other, even if there are free servers ready to do more work at all times.

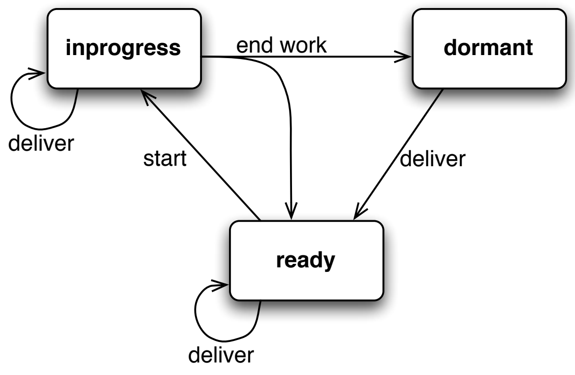

Figure 1: Channel state movements

The operations defined here can be diagrammed (see figure 1) as transitions between three 'states' of a channel, corresponding to the partitions of the dom *pool* collection.

This picture was constructed *after* the specification was written, and seems bleedin' obvious. 'Twas ever thus.

This document fully type-checks (with Fuzz).

7.1 ...footnote

Figure 2 is a picture of the "whiteboard" resulting from the original discussion and from which this document was created.

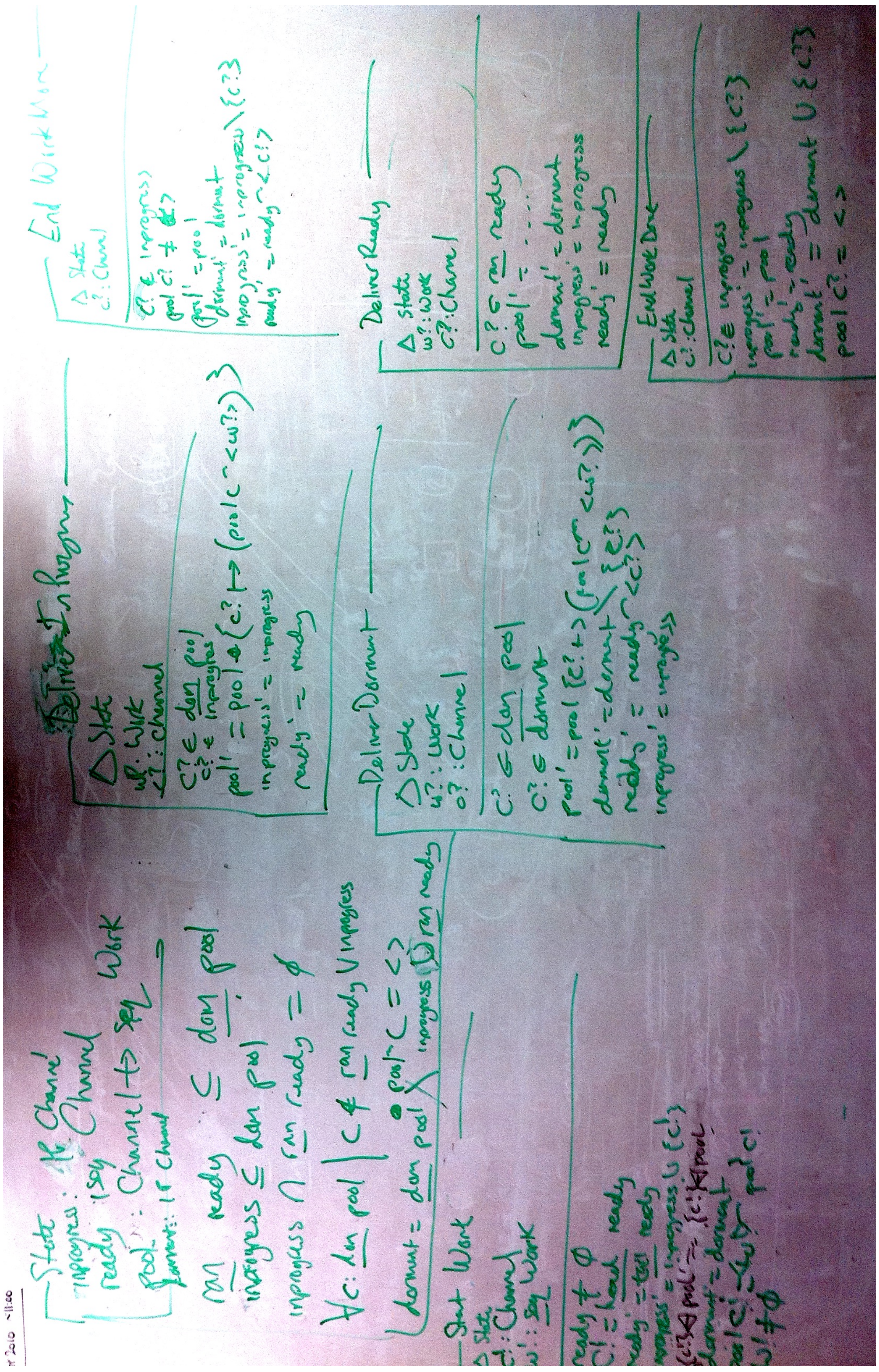

Figure 2: board marks