4 Turbo Protocol

- turbo protocol has one type of outcome the allocation and one on-chain operation - the transfer - already familiar with these, as they formed the basis of a lot of the examples in sections .. and .. - make this precise and talk about redestributing

4.1 Allocations and Transfer

An **allocation** is a list of pairs of addresses and totals, $(a_1{:}v_1, \ldots, a_m{:}v_m)$, where each total, v_i, represents that quantity of coins due to each address, a_i. We assume that each address only appears once in the allocation and require that implementations enforce this by ignoring any additional entries for a given address after the first.

The allocation is in priority order, so that if the channel does not hold enough funds to pay all the coins that are due, then the addresses at the beginning of the allocation will receive funds first. We say that 'A **can afford** x for B', if B would receive at least x coins, were the coins currently held by A to be paid out in priority order.

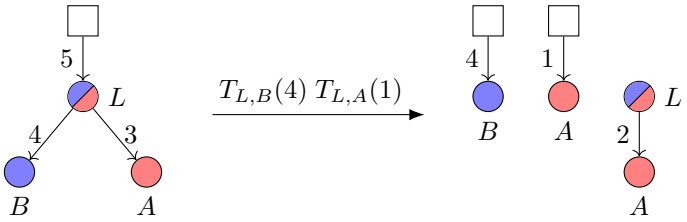

Figure 1: *Allocations pay out in priority order. In the diagram, B is drawn to the left of A to show that B has higher priority in the outcome of L. In this example, L can afford 4 coins for B, but can only afford 1 coin for A.*

Turbo introduces the **transfer** operation, $T_{A,B}(x)$, to trigger the on-chain transfer of funds according to an allocation. If A can afford x for B, then $T_{A,B}(x)$:

1. Reduces the funds held in channel A by x.
2. Increases the funds held by B by x.
3. Reduces the amount owed to B in the outcome of A by x.

If A cannot afford x for B, then $T_{A,B}(x)$ fails, leaving the on-chain state unchanged.

4.2 Unbeatable Redistribution

- section in theory - if you have a strategy, involving just transfers, it is unbeatable - roughly speaking adding money can't hurt

- algorithm for calculating the value for each node in an outcome graph

- we will assume the graph of outcomes is a dag - any participant can ensure this is the case

- number the edge of the dag - topological ordering - work through each node in order paying out in order - amount in node at the end of its turn is its value/funding

- depositing extra can't hurt - withdrawing can only be done by participants, and in this case it's part of the value - the order of transfers out of a channel doesn't change anything

- impossible to do better without depositing - so any strategy you have must give this - and the whole outcome is unbeatable

4.3 Ledger Channels

A **ledger** channel is a channel whose sole purpose is to provide funding to other channels call these sub-channels Ledger channels run the consensus game

Already shown this, but we'll give a quick recap of ledger channels funding another channel offchain

To see how this works, consider the following setup where a ledger channel, L, allocates the funds it holds to participants A and B and channel χ:

$$[\![L : 10]\!], [L \mapsto (A : 2, B : 3, \chi : 5)]_{A,B} \tag{1}$$

We have chosen the simplest example here, where L is funded by the coins it holds on-chain, but it is completely possible to have ledger channels themselves funded by other ledger channels or (later) by virtual channels.

- here L funds χ - easy to see how to offload χ

We claim that the ledger channel L funds the channel χ in the above setup. This is because either participant has the power to convert the situation above into a situation where χ is funded on-chain:

$$T_{L,\chi}(5)[\![L : 10 \mapsto (A : 2, B : 3, \chi : 5)]\!] = [\![L : 10 \mapsto (A : 2, B : 3), \chi : 5]\!] \tag{2}$$

After this operation, we say that the channel χ has been **offloaded**. Note that we do not need to wait for χ to complete before offloading it. The offload converts χ from an off-chain sub-channel to an on-chain direct channel, without interrupting its operation in any way.

The offload should be seen as an action of last-resort. It is important that offloading is allowed so that either player can realize the value in channel χ if required, but it has the downside of forcing all sub-channels supported by L to be closed on-chain. It is in the interest of both participants to open and close sub-channels collaboratively. We next show how this can be accomplished safely.

4.4 Example Constructions

- only examples - demonstrate the principles - easy to extend to the general case

4.4.1 Opening a sub-channel

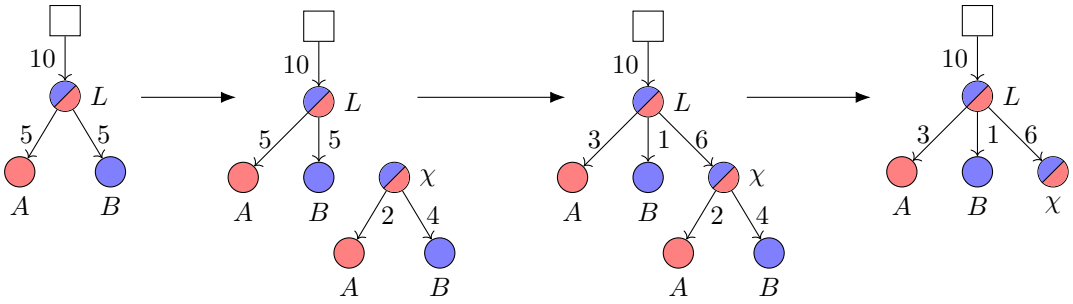

Figure 2

The utility of a ledger channel derives from the ability to open and close sub-channels without on-chain operations. Here we show how to open a sub-channel.

1. Start in a state where A and B have a funded ledger channel, L, open:

$$[\![L : x]\!],\ [L \mapsto (A : a, B : b)]_{A,B} \tag{3}$$

2. A and B prepare their sub-channel χ and progress it to the funding point. With $a' \leq a$ and $b' \leq b$:

$$[\chi \mapsto (A : a', B : b')]_{A,B} \tag{4}$$

3. Update the ledger channel to fund the sub-channel:

$$[L \mapsto (A : a - a', B : b - b', \chi : a' + b')]_{A,B} \tag{5}$$

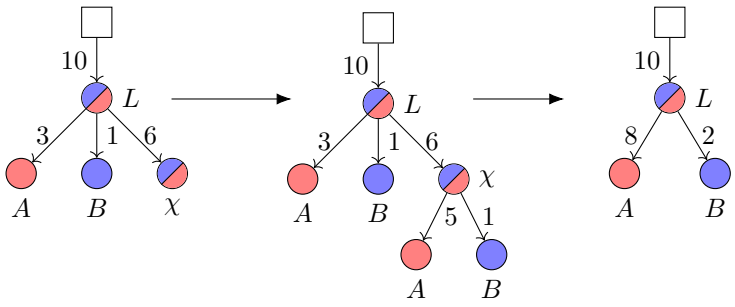

Figure 3

4.4.2 Closing a sub-channel

When the interaction in a sub-channel, χ, has finished we need a safe way to update the ledger channels to incorporate the outcome. This allows the sub-channel to be defunded and closed off-chain.

1. We start in the state where χ is funded via the ledger channel, L. With $x = a + b + c$:

$$[\![L : x]\!],\ [L \mapsto (A : a, B : b, \chi : c)]_{A,B} \tag{6}$$

2. The next step is for A and B to concluded channel χ, leaving the channel in the conclude state. Assuming $a' + b' = c$:

$$[\chi \mapsto (A : a', B : b')]_{A,B} \tag{7}$$

3. The participants then update the ledger channel to include the result of channel χ.

$$[L \mapsto (A : a + a', B : b + b')]_{A,B} \tag{8}$$

4. Now the sub-channel χ has been defunded, it can be safely discarded.

4.4.3 Topping up a ledger channel

Here we show how a participant can increase their funds held in a ledger channel by depositing into it. They can do this without disturbing any sub-channels supported by the ledger channel.

1. In this process A wants to deposit an additional a' coins into the the ledger channel L. We start in the state where L contains balances for A and B, as well as funding a sub-channel, χ. With $x = a + b + c$:

$$[\![L : x]\!],\ [L \mapsto (A : a, B : b, \chi : c)]_{A,B} \tag{9}$$

2. To prepare for the deposit the participants update the state to move A's entry to the end, simultaneously increasing A's total. This is a safe operation due to the precedence rules: as the channel is currently underfunded A would still only receive a if the outcome went to chain.

$$[L \mapsto (B : b, \chi : c, A : a + a')]_{A,B} \tag{10}$$

3. It is now safe for A to deposit into the channel on-chain:

$$D_L(a')[\![L : x]\!] = [\![L : x + a']\!] \tag{11}$$

4. Finally, if required, the participants can reorder the state again:

$$[L \mapsto (A : a + a', B : b, \chi : c)]_{A,B} \tag{12}$$

4.4.4 Partial checkout from a ledger channel

A partial checkout is the opposite of a top up: one participant has excess funds in the ledger channel that they wish to withdraw on-chain. The participants want to do this without disturbing any sub-channels supported by the ledger channels.

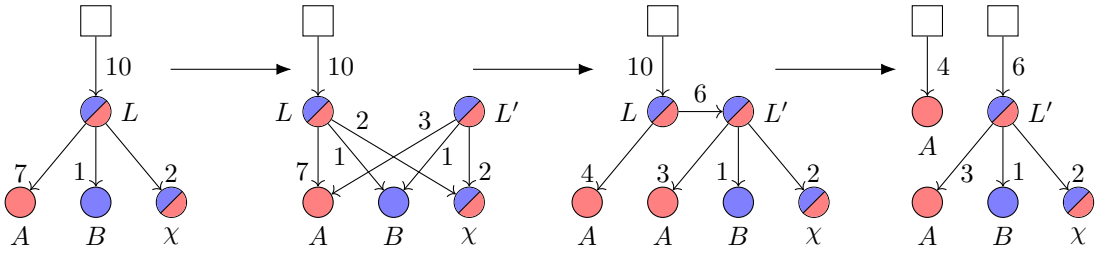

Figure 4: *Cool, huh?*

1. We start with a ledger channel, L, that A wants to withdraw a' coins from:

$$[\![L : x]\!],\ [L \mapsto (A : a + a', B : b, \chi : c)]_{A,B} \tag{13}$$

2. The participants start by preparing a new ledger channel, L', whose state reflects the situation they want to be in after A has withdrawn their coins. This is safe to do as this channel is currently unfunded.

$$[L' \mapsto (A : a, B : b, \chi : c)]_{A,B} \tag{14}$$

3. They then update L to fund L' alongside the coins that A wants to withdraw. They conclude the channel in this state:

$$[L \mapsto (L' : a + b + c, A : a')]_{A,B} \tag{15}$$

4. They then finalize the outcome of L on-chain. This can be done without waiting the timeout, assuming they both signed the conclusion proof in the previous step:

$$[\![L : x \mapsto (L' : a + b + c, A : a')]\!] \tag{16}$$

5. A can then call the transfer operation to get their coins under their control.

$$T_{L,A}(a')[\![L : x \mapsto (L : a + b + c, A : a')]\!] = \\ [\![L : x - a' \mapsto (L' : a + b + c),\ A : a]\!] \tag{17}$$

6. At any point in the future the remaining coins can be transferred to L':

$$T_{L,L'}(a + b + c)[\![L : x \mapsto (L : a + b + c),\ A : a']\!] = \\ [\![L' : a + b + c,\ A : a']\!] \tag{18}$$

Note that A was able to withdraw their funds instantly, without having to wait for the channel timeout.