1 Turbo Protocol

Turbo protocol has one type of outcome (the allocation) and one on-chain operation (the transfer). You will already be somewhat familiar with these, as they formed the basis of a lot of the examples in sections **??** and **??**. In this section we will make these more precise, present the related result on distribution and give some example constructions to cover common tasks such as opening and closing sub-channels.

1.1 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.

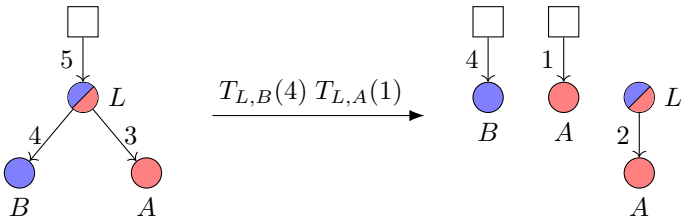

Figure 1: *Allocations pay out in priority order. In the diagram, B is drawn to the left of A to show that B has higher priority in the outcome of L. In this example, L can afford 4 coins for B, but can only afford 1 coin for A.*

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.

1.2 Unbeatable Redistribution

As we mentioned in section **??**, reasoning about redistribution is easy in Turbo: if you can find one strategy to move a certain amount into an address, then no-one else can prevent this from occurring. In this section we will justify this by presenting an algorithm for calculating the funding for each address.

We will restrict ourselves to looking at strategies and counter-strategies involving transfer operations only, ignoring deposits and withdrawals. We claim that deposits and withdrawals cannot be required as part of a strategy and cannot help as part of a counter-strategy. The intuition here is that a deposit into the system cannot reduce the value of any address and cannot increase the value of any address by more than the value of the deposit. Withdrawals can only occur from participant addresses and are the only way funds can leave these addresses, so cannot affect values elsewhere.

We will consider the network of outcomes to be a directed acylcic graph (DAG), where the nodes are channels and the edges represent allocations from one channel to the other. While it is technically possible to create outcomes with cycles, it is also possible for any participant in the channel to prevent this from happening. We therefore consider non-DAG outcome networks to be outside the scope of the protocol.

We commence our value calculation by taking a *topological ordering* of the nodes of the graph. A topological ordering is a way of ordering the nodes such that if $N_1 \mapsto N_2$ is an edge then N_1 comes before N_2 in the list. It is a known result that all DAGs have at least one topological ordering.

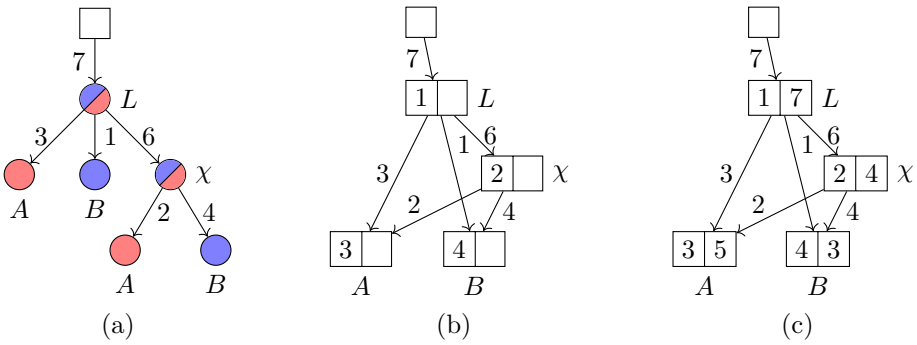

Figure 2: *Diagram (a) shows the outcome network that is the input to the value calculation. In (b), we have reformulated (a) as a DAG with uniquely labelled nodes by merging the two A and B modes. We have also labelled the nodes with a topological ordering. In (c), we have completed the algorithm giving each node its funding/value.*

Once we have ordered the nodes, we work through the nodes in order. For a node, n, we first look at the amount of direct funding allocated to n's address in

the adjudicator. We then look at each of n's parents and calculate the amount that that the parent can afford for n, according to the value of the parent (which will already have been calculated, due to the topological ordering). Finally we set the node's value to be the sum of the direct funding and the amounts afforded by each parent.

It is not hard to see that it is impossible to find a strategy that gives any node a value higher than allocated by this algorithm. It also is not hard to construct a strategy for a node to obtain the value allocated by the algorithm, if necessary by actually implementing the algorithm up until that node. Given that we are only considering counter-strategies with transfers, and we have done every possible transfer on these channels, we know that there are no transfers that can interrupt this algorithm. It is also easy to see that calling transfers out of order does not affect the ultimate result.

In Turbo, it is therefore easy to calculate the value of each node and find unbeatable strategies for extracting the value of that address .

1.3 Ledger Channels

A **ledger** channel is a channel whose sole purpose is to provide funding to other channels. We call the channels that are funded by the ledger channel **sub-channels** of the ledger channel. All ledger channels run the consensus game.

Although this has already been covered, for completeness we will quickly recap how a sub-channel can be considered to be funded by a ledger channel. For example, consider the following setup where a ledger channel, L, allocates the funds it holds to participants A and B and channel χ:

$$[\![L : 10]\!], [L \mapsto (A : 3, B : 1, \chi : 6)]_{A,B} \tag{1}$$

In this example, χ is funded with 6 coins by L for both A and B. To show this,

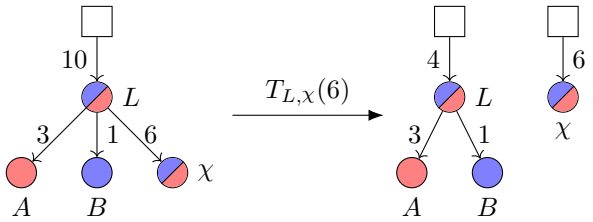

Figure 3: *Offloading a ledger channel.*

we have to have an unbeatable strategy for moving to a situation where χ is directly funded with 6 coins. To do this we first note that the outcome $(A : 3, B : 1, \chi : 6)$ is finalizable for both A and B, so we can start our strategy by putting this outcome on-chain. Once it is on-chain, the transfer operation $T_{L,\chi}(6)$ is

all that is required to make χ directly funded. From the Turbo redistribution result, we know that this redistribution strategy is unbeatable.

Note that offloading χ like this should be seen as an action of last-resort, as after the off-load 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 give some examples to show how this can be accomplished safely.

1.4 Example Constructions

We now give some examples of how to work with ledger channels on Turbo. We have chosen to present examples that demonstrate the key principles instead of presenting general protocols, as we believe that, once seen, these protocols are easy to extend to the general case.

1.4.1 Opening a Sub-channel

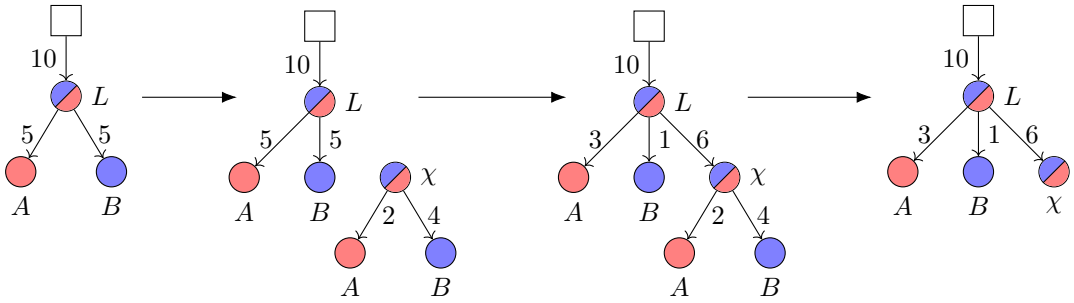

Figure 4

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:

$$[\![L : x]\!],\ [L \mapsto (A : a, B : b)]_{A,B} \tag{2}$$

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

$$[\chi \mapsto (A : a', B : b')]_{A,B} \tag{3}$$

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

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

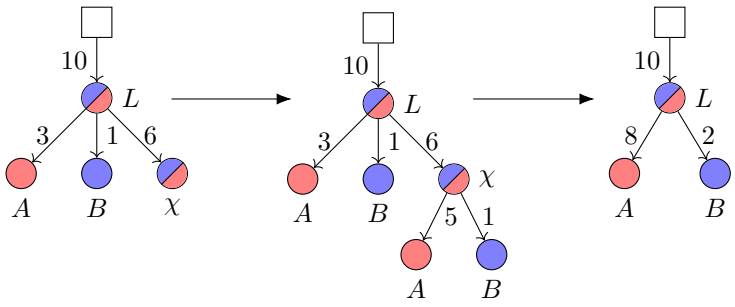

Figure 5

1.4.2 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 $x = a + b + c$:

$$[\![L : x]\!],\ [L \mapsto (A : a, B : b, \chi : c)]_{A,B} \tag{5}$$

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

$$[\chi \mapsto (A : a', B : b')]_{A,B} \tag{6}$$

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

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

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

1.4.3 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 $x = a + b + c$:

$$[\![L : x]\!],\ [L \mapsto (A : a, B : b, \chi : c)]_{A,B} \tag{8}$$

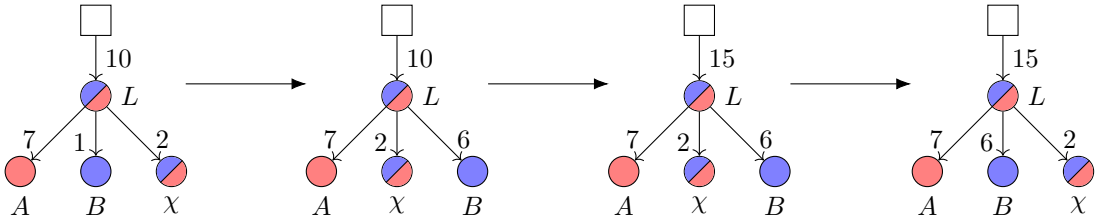

Figure 6

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.

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

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

$$D_L(a')[\![L : x]\!] = [\![L : x + a']\!] \tag{10}$$

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

$$[L \mapsto (A : a + a', B : b, \chi : c)]_{A,B} \tag{11}$$

1.4.4 Partial Withdrawal 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.

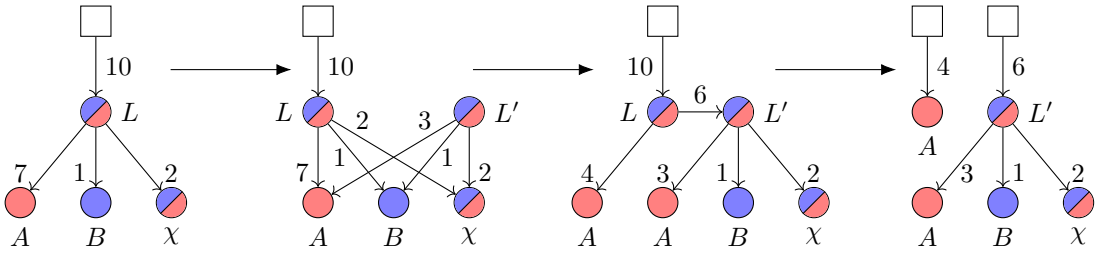

Figure 7: *Cool, huh?*

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

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

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.

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

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

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

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:

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

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

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

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

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

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