- CandyLamport global snapshot algorithm

  When a process $CandyLamport(A)_i$ that has not previously been involved in the snapshot algorithm receives a $snap_i$ input, it records the current state of A_i. Then it immediately sends a $marker$ message on each of its outgoing channels; this $marker$ indicates the boundary between the messages that are send out before the local state was recorded and the messages sent out afterward.

  Then $ChandyLamport(A)_i$ begins recording the messages arriving on each incoming channel in order to obtain a state for that channel; it records messages on the channel just until it encounters a $marker$. At this point, $ChandyLamport(A)_i$ has recorded all the messages sent on that channel before the neighbor at the other end recorded its local state.

  There is one remaining situation to consider: suppose that process $ChandyLamport(A)_i$ receives a $marker$ message before it has recorded the state of A_i. In this case, immediately upon receiving the first $marker$ message, $ChandyLamport(A)_i$ records the current state of A_i, sends out $marker$ messages, and begins recording incoming messages. The channel upon which it has just received the $marker$ is recorded as empty. The formal code appears below.

$CandhyLamport(A)_i$:

Signature:
As in A_i, plus:

Input:
  $snap_i$
  $receive("marker")_{j,i},\ j \in nbrs$
Output:
  $report(s, C)_i,\ s \in states(A_i)$,
  $send("marker")_{i,j},\ j \in nbrs,\ m\ a\ message\ of\ A$
Internal:
  internal-send$(m)_{i,j},\ j \in nbrs,\ m$ a message of A

States:
As for A_i, plus:
$status \in \{start, snapping, reported\},\ initially\ start$
snap-state, a state of A_i, initially $null$
for every $j \in nbrs$:
    channel-snapped(j)a Boolean, initially $false$
    send-buffer(j), a FIFO queue of A messages and $markers$,
    initially empty
    snap-channel(j), a FIFO queue of A messages,
    initially empty

```
Transitions:

snap_i
  Effect:
    if status = start then
       snap-state = state of A_i
       status = snapping
       ∀ j ∈ nbrs
           add "marker" to send-buffer(j)

receive("marker")_{j,i}
  Effect:
    if status = start then
       snap-sate = state of A_i
       status = snapping
       ∀ j ∈ nbrs
           add "marker" to send-buffer(j)
       channel-snapped(j) = true

send(m)_{i,j}
  Precondition:
    m is first on send-buffer(j)
  Effect
    remove first element of send-buffer(j)

report(s,C)_i
  Precondition:
    status = snapping
    ∀ j ∈ nbrs : channel-snapped(j) = true
    s = snap-state
    ∀ j ∈ nbrs : C(j) = snap_channel(j)
  Effect:
    status = reported

Input of A_i ≠ receive
  Effect:
    As for A_i

Locally controlled action of A_i ≠ send
  Precondition:
    As for A_i
  Effect:
    As for A_i
```

```
internal-send(m)_{i,j} in A_i
  Precondition:
    As for send(m)_{i,j} in A_i
  Effect:
    add m to send-buffer(j)
```

Theorem 1: The $ChandyLamport(A)$ algorithm determines a consistent global snapshot for A.

Proof: Fix any fair execution of $ChandyLamport(A)$ in which some process receives a $snap$ input. We first argue that every process eventually performs a $report$ output. As soon as any $snap$ input occurs at some process $ChandhyLamport(A)_i$ that process records the state of A_i and send out $marker$ on all its channels. As soon as any other process $ChandyLamport(A)_j$ receives a $marker$ on any channel, it records the state of A_j and send out $markers$ on all its channels, if it has not previously done so. Because of the connectivity of the graph, $markers$ thus eventually propagate to all processes, and all processes record their local states. Also every process $ChandyLamport(A)_i$ eventually performs a $report$, as claimed.

Now we argue that the returned global state is consistent. That is, we let α denote the contained fair execution of A, and we produce the required alternative execution $\alpha^{'}$ and its requited prefix. Namely let α_1 be the portion of α before the first $snap$ and $alpha_2$ the portion of α after the last $report$. Execution $alpha^{'}$ begins with α_1 and ends with α_2; the only reordering involves the events of α between the first $snap$ and the last $report$.

Each event of α between the first $snap$ and the last $report$ occurs at some process $ChandyLamport(A)_i$. These events can be divided into two sets: S_1 those that precede the event ($snap_i$ or $receive(marker)_{j,i}$) of $ChandyLamport(A)_i$ at which the state of A_i is recorded, and S_2 those that follow this event. The reordering places all S_1 events before all S_2 events while preserving the order of events of each A_i and the order of each send (derived from an internal-send) which respect to the corresponding receive. The fact that such a reordering is possible depends on the fact that there is no internal-send(m)$_{i,j}$ event that follows the recording of the state at A_i and whose corresponding $receive(m)_{i,j}$ event precedes the recording of the state of A_j. (If an internal-send(m)$_{i,j}$ follows the recording of the state of A_i, then m is placed in send-buffer$(j)_i$ after the $marker$. But this implies that the $marker$ arrives at $ChandyLamport(A)_j$ before m does, which means that the state of A_j is already recorded by the time m arrives). Reordering the events of α in this way and filling in states of each A_i as in α yields the sequence $\alpha^{'}$.

Now consider the prefix α_3 of $\alpha^{'}$ ending just after all the events in S_1. We claim that $\alpha^{'}$ and its prefix α_3 satisfy all the needed properties; the key fact is that the results returned by all the processes constitute exactly the global state of A after α_3. But the messages in transit for i to j after α_3 are exactly the messages whose $internal - send(m)_{i,j}$ events occur after the recording of the state of A_j. These are exactly the messages that arrive at $ChandyLamport(A)_j$ from $ChandhyLamport(A)_i$ ahead of the $marker$ and after

$ChandyLamport(A)_J$ records the state of A_J, which are exactly the messages recorded by $ChandyLamport(A)_j$ for this channel.□.

- Example: Two-dollar bank

  Let A be a simple special case of the banking system in which the underlying graph G has only two nodes, 1 and 2, and in which total amount of money in the system is \$2. Suppose each process begins with \$1. We use notation $CL(A)_i$ as shorthand for the process $ChandyLamport(A)_i$.

  Consider fair execution of $CL(A)_i$ depicted in Figure 1. In this diagrams, the # symbols denote $markers$.

  – (a) $snap_1$ occurs, causing $CL(A)_1$ to record the state of A_1 as \$1. Then $CL(A)_1$ sends a $marker$ to $CL(A)_2$ and starts recording incoming messages.

  – (b) A_1 sends \$1 to A_2; the dollar enters the channel from $CL(A)_1$ to $CL(A)_2$, behind the $marker$.

  – (c) A_2 send \$1 to A_1.

  – (d) A_1 receives the dollar and $CL(A)_1$ records it in snap-channel$(2)_1$.

  – (e) $CL(A)_2$ receives the $marker$ from $CL(A)_1$, records the state of A_2 as \$0, sends a $marker$ to $CL(A)_1$, records the state of the incoming channel as empty, and reports its results.

  – (f) $CL(A)_1$ receives the marker from $CL(A)_2$, records the state of the incoming channel as the sequence consisting of one message (the \$1 it received before the $marker$), and reports its results.

  – (g) A_2 receives the dollar.

  The global state returned by the algorithm is shown in (h). It consists of \$1 at A_1, \$1 in the channel from A_2 to A_1, and no money at A_2 or in the channel from A_1 to A_2. This yields the correct total \$2.

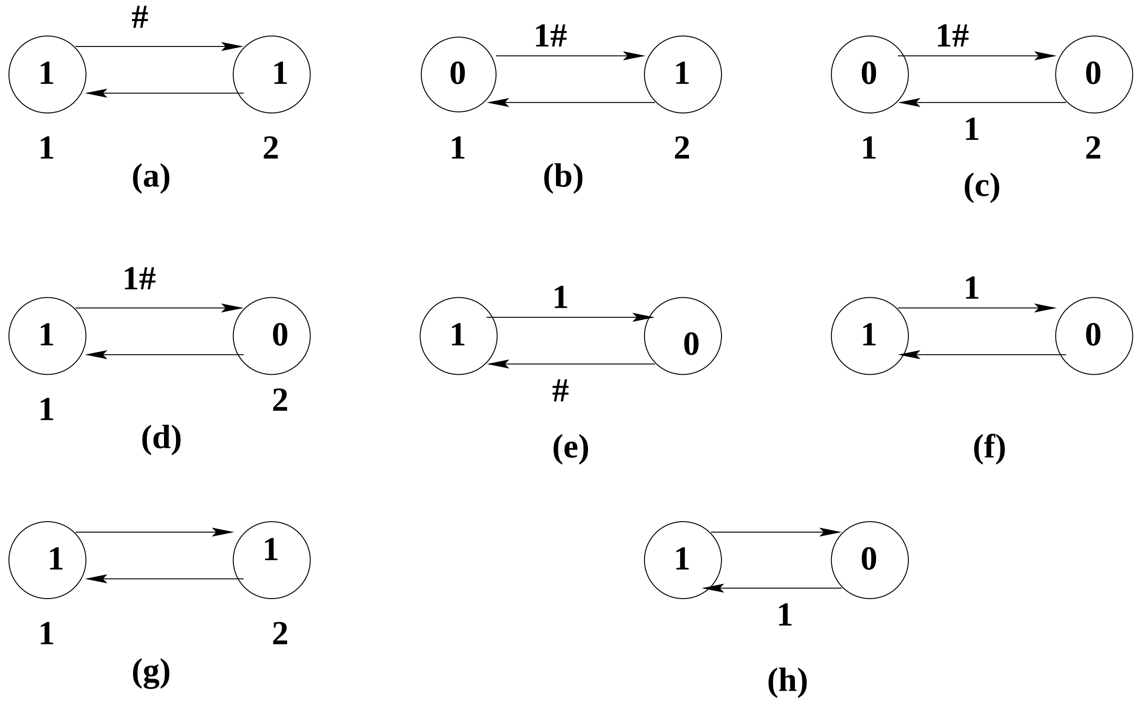

Figure 1: Execution of ChandyLamport(A), for the two-dollar bank

- Further reading:

  - Nancy A. Lynch: Distributed Algorithms, Chapter 19, pages 617–639, Morgan Kaufmann, 1996.
  - Gabriel Bracha and Sam Toueg. Distributed deadlock detection. *Distributed Computing*, 2(3):127–138, December 1987.