Early Stopping Consensus Algorithm

Algorithm 1 Early Stopping Consensus - Short Version. Actions of Process p_i

```
procedure EARLY-SHORT
    integer: $xCurr \leftarrow$ initial-value;
    integer: $xPrev \leftarrow \infty$ // Large value
    set: $fail\_prev_i \leftarrow nil$; // Previous fail-set maintained by $p_i$
    set: $fail\_curr_i \leftarrow nil$; // Current fail-set maintained by $p_i$
    boolean: $shldTerm_i \leftarrow false$ // Used by $p_i$ to keep track if it should terminate
    for ($r \leftarrow 1$ to $f$) do
        $fail\_prev_i \leftarrow fail\_curr_i$; // Store the current set in previous set
        $xPrev = xCurr$; // Store the current value in previous value
        Broadcast($\langle xCurr \rangle$);
        $\langle y_j \rangle \leftarrow$ value (if any) received from $p_j$ in this round;
        for all ($\langle y_j \rangle$) do
            if ($\langle y_j \rangle = nil$) then
                Add $p_j$ to $fail\_curr_i$; // Add $p_j$ to local failed set of $p_i$
            else
                $xCurr \leftarrow min(xCurr, y_j)$; // Take the minimum of the received values
            end if
        end for
        if ($shldTerm_i = true$) then
            break; // We break as we have to terminate
        end if
        if ($fail\_prev_i = fail\_curr_i$) then
            if ($xCurr = xPrev$) then
                break; // Early stopping condition
            end if
            $shldTerm_i = true$; // We must terminate in the next round
        end if
    end for
    output $xCurr$ as the consensus value.
end procedure
```

Runtime Analysis: The above algorithm will run terminate in $min(f' + 2, f + 1)$ rounds where f' is the total number of crashes while f is the upper limit on the number of crashes.

Algorithm 2 Early Stopping Consensus - Long Version. Actions of Process p_i

```
procedure EARLY-LONG
    integer: $x \leftarrow$ initial-value;
    set: $gfs_i \leftarrow nil$; // global fail set maintained by $p_i$
    set: $lfs_i \leftarrow nil$; // local fail set maintained by $p_i$ for the current round
    // $annTerm$ a boolean value which decides if $p_i$ should announce termination or not
    boolean: $annTerm \leftarrow false$;
    // $shldTerm$ a boolean value which decides if $p_i$ should terminate or not
    boolean: $shldTerm \leftarrow false$;
    for ($r \leftarrow 1$ to $f$) do
        Broadcast($\langle x, gfs_i, annTerm \rangle$);
        if ($annTerm = true$) then
            break; // Announce and then terminate
        end if
        $\langle y_j, fs_j, termFlag_j \rangle \leftarrow$ value (if any) received from $p_j$ in this round;
        for all ($\langle y_j, fs_j, termFlag_j \rangle$) do
            if ($\langle y_j, fs_j, termFlag_j \rangle = nil$) then
                Add $p_j$ to $lfs_i$; // Add $p_j$ to local failed set of $p_i$
            else
                $gfs_i \leftarrow gfs_i \cup fs_j$; // Combine the failed set of $p_j$ with $p_i$
                $x \leftarrow min(x, y_j)$; // Take the minimum of the received values
                $shldTerm \leftarrow (shldTerm \vee termFlag_j)$; // Check if $p_j$ requires us to terminate
            end if
        end for
        if ($shldTerm = true$) then
            break; // We break if some other processes asks us to terminate
        end if
        if ($lfs_i \subseteq gfs_i$) then
            $annTerm = true$; // We must announce termination to other processes
        end if
        $gfs_i \leftarrow gfs_i \cup lfs_i$; // Collect all the failed processes identified in this round
        $lfs_i \leftarrow nil$; // Next, reset $lfs_i$ before going into next round
    end for
    output $x$ as the consensus value.
end procedure
```