Distributed Algorithm on AHCv2: Shavit-Francez Distributed Termination Detection Algorithm

Release V1.0.0

İmre Kosdik

May 11, 2024

CONTENTS

CHAPTER

ONE

SHAVIT-FRANCEZ TERMINATION DETECTION ALGORITHM

1.1 Abstract

A computation in a distributed algorithm terminates when the algorithm reaches a state where no potential applicable steps exist. In distributed systems, determining whether a particular computation has terminated is a crucial need because the execution of other computations may depend on the completion of the execution. Because the processes in a distributed system do not know the global state of the system and do not share any global clock, inferring if a distributed computation has ended is a challenging problem. The *Shavit-Francez Algorithm* is a fundamental termination detection algorithm that addresses these issues, ensuring that the system reaches a consistent state where all the processes complete their computations before proceeding with other tasks. Implementing the *Shavit-Francez Algorithm*, along with analyzing the message and time complexity by running the algorithm on different network topologies and various numbers of nodes, provides valuable insights on the algorithm's efficiency and applicability over real-world distributed systems.

1.2 Introduction

When a problem requires more than one process to be solved, distributed systems are a good candidate. These processes work together to solve subproblems. Therefore, it is crucial to identify when a process has completed its execution because its output is used as input to another process to continue its execution. Termination detection in distributed computing is a challenging task in distributed systems because processes are unaware of the global state of the system due to communication delays, and there is no shared global system.

Termination detection algorithms have a significant role in ensuring a consistent state where all processes complete their computations and are ready to proceed with the upcoming tasks. Achieving a consistent state also preserves the correctness of the system. Termination detection can also take part in efficient resource management by releasing not-needed resources. Additionally, efficient resource management may be beneficial in preventing deadlocks since their primary cause is indefinitely waiting to acquire resources.

Termination is a property of the global state of distributed computing. However, due to the decentralized and asynchronous nature of distributed systems, acquiring the global state of the system is a significant challenge. Since termination detection algorithms rely on additional control messages, the message overhead can thoroughly impact the system's performance. Also, as the distributed system expands, the complexity and overhead of maintaining the algorithm can result in scalability issues. Additionally, designing algorithms such that the underlying computation does not interfere with the ongoing executions is another challenge.

Shavit-Francez Algorithm is an effective way to detect termination without interfering with the overall execution of the distributed system. The algorithm doesn't rely on synchronous communication, simplifying the design and implementation of the system. In contrast to the increasing number of nodes, the message-sharing overhead from the algorithm remains low, which means it has less impact on the system's performance. In this paper, we aim to comprehensively explain the implementation details of *Shavit-Francez Algorithm* and present the results of experiments we conducted on the algorithm related to its time and message complexity.

We contribute to the field of distributed systems by:

- Implementating *Shavit-Francez Algorithm* on the AHCv2 platform. We explain the implementation details in Section 1.3.
- Conducting experiments on the algorithm over different network topologies and node counts. We discuss the experiment setup and results in Section 1.4

1.3 Shavit-Francez Termination Detection Algorithm

1.3.1 Background and Related Work

Global termination in a distributed system occurs when all processes reach the local termination state, no messages are in transit, and processes do not send or receive any message. Local termination is the state where the process completed its execution, meaning it is passive (idle) and is ready to continue its computation upon receiving any message. A process is active when it is performing some computation. In a distributed system, only the active processes can send messages. Therefore, a computation in a distributed system terminates when all its processes are idle.

The primary consideration behind the termination detection algorithms is adding a control algorithm to the system running to detect whether the basic algorithm has reached a termination state. The basic algorithm is the one currently running in the distributed system. Initiators of the basic algorithm are active processes and can trigger the execution of the control algorithm. The control algorithm consists of the termination detection and the announcement phases. The messages that the control algorithm sends or receives are control messages. Ideally, the termination detection algorithm should not need additional communication channels to send or receive its control messages, and should not interfere with the basic algorithm running on the system.

Shavit-Francez Algorithm [ShavitFrancez1986] is the generalization of Dijkstra-Scholten Termination Detection Algorithm [DijkstraSholten1980] for distributed systems. In Dijkstra-Scholten Algorithm, the initiator of the basic algorithm maintains a tree of active processes. If a process makes another process active by sending a message, that process joins the tree as a child of the process. A process can only leave the tree if it transitions to passive state and it has no children in the tree. Once the tree becomes empty, the initiator announces the termination. *Shavit-Francez Termination Detection Algorithm* [ShavitFrancez1986] maintains a forest instead of a single tree due to the nature of the distributed system. Each initiator maintains its tree and constitutes it to the forest. The condition for a process to join a tree is that it is not already a member of any of the trees in the forest. Other than that, the algorithm continues as in the Dijkstra-Scholten Algorithm. Instead, it starts a wave in which only those processes not part of a tree participate. Because each initiator is only aware of the emptiness of its tree, an empty tree does not guarantee that the whole forest is empty. The wave algorithm ensures that all the trees in the forest collapse before announcing termination. Once the wave decides, the initiator can then announce the termination. A wave algorithm is not complete unless all the processes take part in its execution. Following this property, the algorithm ensures that if none of the waves started by the processes are complete because a process refuses to take part, the initiator maintaining the last tree to be empty will start a wave that eventually decides and announces the termination. [Fokking2013]

The wave algorithm we choose for the implementation in *Shavit-Francez Termination Detection Algorithm* is *Echo Algorithm* [Fokking2013]. The *Echo Algorithm* initiator begins by sending messages to all of its neighbors. If a non-initiator receives a message for the first time, it sets its parent as the sender process and sends a message to all its neighbors except its parent. After receiving messages from all its neighbors, the non-initiator notifies its parent. Finally, the initiator receives messages from all its neighbors and decides.

1.3.2 Shavit-Francez Termination Detection Algorithm: Shavit-Francez Termination Detection Algorithm

The *Shavit-Francez Termination Detection Algorithm* is proposed by Nir Shavit and Nissim Francez to detect termination in the distributed system algorithms. General flow of execution of the algorithm is as follows:

1. If a process is the initiator of the basic algorithm, it sets its active property to true, indicating that it is doing some computation at the time. (Line 5)
2. If a process sends a basic message while executing the basic algorithm, then it increases its number of children by 1, because the process that is sent message becomes its children. (Line 8)
3. If a process receives a basic message while it is passive, then it becomes active (Line 12) and sets its parent to the process sending the basic message (Line 13). If it was already active, then it informs the sending process with an ACKNOWLEDGE message. (Line 15)
4. If a process receives an ACKNOWLEDGE message, it decreases its number of children by 1 (Line 19), calls the LeaveTree procedure. (Line 20)
5. If at some point a process becomes passive then it sets its active property to false (Line 23), calls the LeaveTree procedure. (Line 24)
6. Inside the LeaveTree procedure, a passive process with no children sends its parent an ACKNOWLEDGE message (Line 29), and then sets its parent to None (Line 30).
7. If a passive process with no children has no parent at all, it starts a wave inside the LeaveTree procedure. (Line 32).
8. A a passive process with no children receiving a wave message acts as what the wave algorithm dictates (Line 37) and if the wave algorithm decides, it announces termination. (Line 38)

Listing 1: Shavit-Francez Termination Detection Algorithm

```
1  bool active<p> // set when p becomes active, and reset when p becomes passive
2  nat cc<p> // keeps track of the number of children of p in its tree
3  proc parent<p> // the parent of p in a tree in the forest
4
5  if p is an initiator then
6     active<p> <- true
7  end if
8
9  if p sends a basic message then
10    cc<p> <- cc<p> + 1
11 end if
12
13 if p receives a basic message from a neighbor q then
14    if active<p> = false then
15       active<p> <- true
16       parent<p> <- q
17    else
18       send <ack> to q
19    end if
20 end if
21
22 if p receives <ack>
23    cc<p> <- cc<p> - 1
24    perform procedure LeaveTree<p>
```

(continues on next page)

(continued from previous page)

```
end if

if p becomes passive
   active<p> <- false
   perform procedure LeaveTree<p>;
end if

Procedure LeaveTree<p>
   if active<p> = false and cc<p> = 0 then
       if parent<p> !=  then
            send <ack> to parent<p>
            parent<p> <-
       else
         start a wave, tagged with p
   end if
end if

if p receives a wave message then
   if active<p> = false and cc<p> = 0 then
       act according to the wave algorithm
       in the case of a decive event, call Announce
   end if
end if
```

1.3.3 Echo Algorithm:

The *Echo Algorithm* [Fokking2013] takes part in making sure that all the trees in the forest collapsed and thus, concluding that the basic algorithm terminated. Since this paper focuses on the implementation details of the *Shavit-Francez Termination Detection Algorithm*, we do not explicitly describe the pseudocode we provided for the wave algorithm. We only add the pseudocode here because we make use of this algorithm while implementing the *Shavit-Francez Termination Detection Algorithm*.

Listing 2: Echo Algorithm

```
nat received<p>;
proc parent<p>;

if p is the initiator then
   send <wave> to each r in Neighbors<p>
end if

if p receives a <wave> from neighbor q then
   received<p> <- received<p> + 1
   if parent<p> !=  and p is a non-initiator then
       parent<p> <- q
       if |Neighbors<p>| > 1 then
          send <wave> to each r in Neighbors<p>\{q}
       else
          send <wave> to q
       end if
   else if received<p> = |Neighbors<p>| then
```

(continues on next page)

(continued from previous page)

```
18          if parent<p> !=  then
19             send <wave> to parent<p>
20          else
21             decide
22          end if
23       end if
24    end if
```

1.3.4 Example With Terminating Distributed System Algorithm

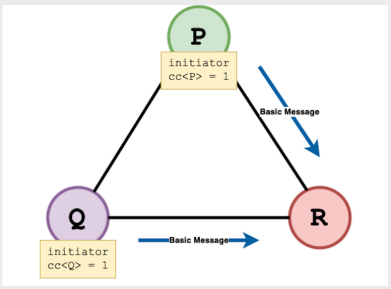

Fig. 1: Step 1

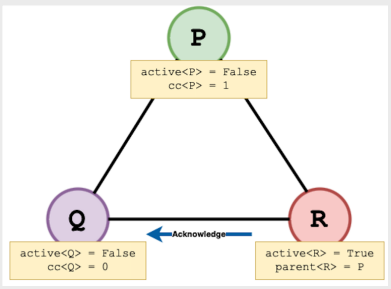

Fig. 2: Step 2

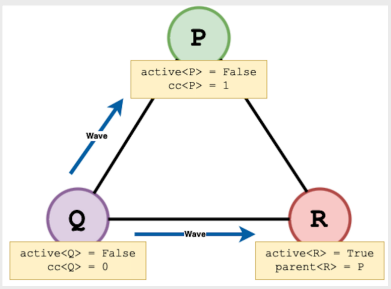

Fig. 3: Step 3

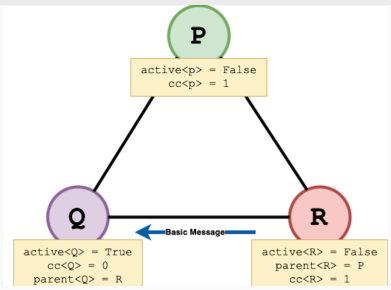

Fig. 4: Step 4

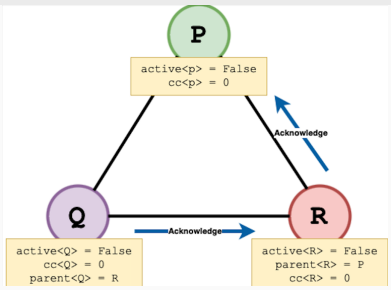

Fig. 5: Step 5

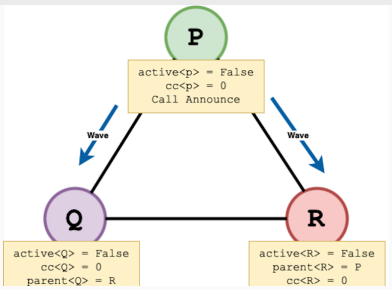

Fig. 6: Step 6

Assume that there are three processes p, q, r in an undirected network. One way to execute the *Shavit-Francez Algorithm* is as follows:

1. At the start, the initiators p and q both send a basic message to r, and set cc<p> and cc<q> to 1. Next, p and q become passive.(See Figure 1)
2. Upon receipt of the basic message from p, r becomes active and makes p its parent. Next, r receives the basic message from q, and sends back an acknowledgment, which causes q to decrease cc<q> to 0.(See Figure 2)
3. Since q became passive as the root of a tree, and cc<q> = 0, it starts a wave. This wave does not complete, because p and r refuse to participate.(See Figure 3)
4. r sends a basic message to q, and sets cc<r> to 1. Next, r becomes passive.(See Figure 4)
5. Upon receipt of the basic message from r, q becomes active, and makes r its parent. Next, q becomes passive, and sends an acknowledgment to its parent r, which causes r to decrease cc<r> to 0. Since r is passive and cc<r> = 0, it sends an acknowledgment to its parent p, which causes p to decrease cc<p> to 0.(See Figure 5)
6. Since p became passive as the root of a tree, and cc<p> = 0, it starts a wave. This wave completes, so that p calls Announce.(See Figure 6)

1.3.5 Example With Non-Terminating Distributed System Algorithm

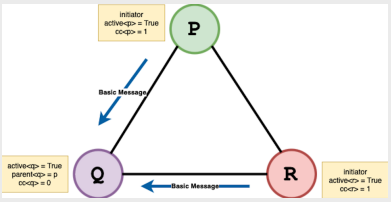

Fig. 7: Step 1

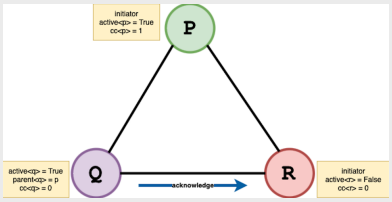

Fig. 8: Step 2

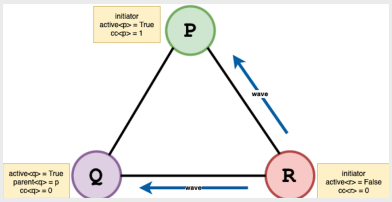

Fig. 9: Step 3

Assume that there are three processes p, q, r in an undirected network. One way to execute the *Shavit-Francez Algorithm* is as follows:

1. At the start, the initiators p and r both send a basic message to q, and set cc<p> and cc<r> to 1. (See Figure 7)
2. Upon receipt of the basic message from p, q becomes active and makes p its parent. Next, q receives the basic message from r, and sends back an acknowledgment, which causes r to decrease cc<r> to 0. (See Figure 8)
3. Next, r becomes passive.
4. Since r became passive as the root of a tree, and cc<r> = 0, it starts a wave. This wave does not complete, because p and q refuse to participate.(See Figure 9)
5. Since neither p nor q becomes passive at some point, the algorithm cannot complete the wave and cannot announce termination.

1.3.6 Correctness

1. **Safety**: The *Announce* is called when a decision occurs in the wave algorithm. This implies that each process p has sent a wave message or has decided, and the algorithm implies that cc<p> was 0 when p did so. No action makes cc<p> more than 0 again, so (for each p) cc<p> is 0 when *Announce* is called. [Tel2001]
2. **Liveness**: Assume that the basic computation has terminated. Within a finite number of steps the termination-detection algorithm reaches a terminal configuration, and as in the correctness statement below it can be shown that in this configuration the forest is empty. Consequently, all events of the wave are enabled in every process, and that the configuration is terminal now implies that all events of the wave have been executed, including at least one decision, which caused a call to *Announce*. [Tel2001]
3. **Correctness**: Define S to be the sum of all cc<p> for each process p. Initially S is zero, S is incremented when a basic message is sent, S is decremented when a control message is received, and S is never negative. This implies that the number of control messages never exceeds the number of basic messages in any computation. [Tel2001]

1.3.7 Complexity

1. *Shavit-Francez Algorithm*: The worst case message complexity is O(M + W) where M is the number of the messages sent by the underlying computation and W is a message exchange complexity of the wave algorithm, which is 2E for *Echo Algorithm* where E is. [Tel2001]
2. *Echo Algorithm*: The message complexity is O(2E), where E is the number of edges. [Fokking2013]

1.4 Implementation, Results and Discussion

1.4.1 Implementation and Methodology

We utilized the Python (version 3.12) scripting language and the Ad-Hoc Computing (adhoccomputing) library while implementing the Shavit-Francez Termination Detection Algorithm. We also employed the networkx library to generate various network topologies and the matplotlib library to visualize them. Each component in the topology can be the initiator for the termination detection algorithm. It is up to us which component to choose the initiator/initiators. After that, we must send an event to the initiators to execute the termination detection algorithm. Either initiator components can send the event to themselves, or other non-initator nodes can send it to the initiators. Since termination detection is the algorithm that runs on top of the basic algorithm running in the system, we needed to simulate a basic algorithm by creating additional messages that we could send to the component externally. We use "*BECOMEPASSIVE*" message to simulate processes finishing their execution and "*SENDBASICMESSAGE*" to simulate messages that the basic algorithm exchanges on its execution. An important consideration is that one can only send these messages if the process is active. Another consideration is that, the components need to be aware of who is executing the control algorithm. Therefore, the process starting the algorithm send a message to its neighbors indicating that it is the initiator for this execution.

For a distributed system in that its processes never become passive, we should expect that the algorithm does not announce the termination and, therefore, no output in the command prompt. As an example, we can consider a system with deadlocks. Since none of the processes can continue because they need resources from others, the algorithm cannot announce the termination. To create this scenario, we could think that the "SENDBASICMESSAGE" event acts as a "REQUEST" and create a cyclic graph. In other cases, sending a "BECOMEPASSIVE" event to a process acts as if the process finishing its execution, and we should see that the algorithm announces the termination in the commant prompt.

We implemented both the Echo Algorithm and the Shavit-Francez Termination Detection Algorithm by employing the pseudocode descriptions given in [Fokking2013]. We used the same message types given in the descriptions to achieve the message passing between the components. The make the component who is the initiator of the basic algorithm send itself "DETECTTERMINATION" message to trigger the algorithm. After that, depending on the basic-messages exchanged between the processes and the status of the processes, the algorithm announces the termination.

1.4.2 Results

In order to evaluate the message complexity of the Shavit-Francez Termination Detection Algorithm, we designed three different experiments. For the first experiment, we generate ring topologies with various number of node counts. The node with the first index becomes the initiator for the algorithm. Each node in the topology sends basic messages to all of its neighbors and then becomes passive. According to this configuration, we execute the algorithm for 2, 3, 5, 10, 20, 30, 50, 100 and 500 number of nodes and examine how many wave messages and basic messages are exchanged along with time it takes to complete the execution.

For a ring topology, if there are n number of nodes, then there exists n number of edges in the topology. On the table, we observe that the number of exchanged wave messages is exactly two terminates the number of edges. That means, it is consistent with the message complexity of the Echo Algorithm, which is O(2E). Since each node sends basic message to each of its neighbors, then there should be 2 * n basic messages exchanged in the topology. So, the results we get gives us the complexity of the underlying computation, which is O(M), where M is 2 * n. We also observe that, the time it takes to complete the execution of the algorithm is directly proportional to the number of nodes in the topology.

Table 1: Message Complexity Analysis of Termination Detection Algorithm on a Ring Topology

Node Count	Time Elapsed Until Termination	Number of Exchanged Messages
2	1.207543134689331	2 (wave) + 2 (basic)
5	1.5333149433135986	10 (wave) + 10 (basic)
10	2.045008897781372	20 (wave) + 20 (basic)
20	3.0964579582214355	40 (wave) + 40 (basic)
30	4.135018825531006	60 (wave) + 60 (basic)
50	6.216088056564331	100 (wave) + 100 (basic)
100	11.443176984786987	200 (wave) + 200 (basic)
500	53.18007707595825	1000 (wave) + 1000 (basic)

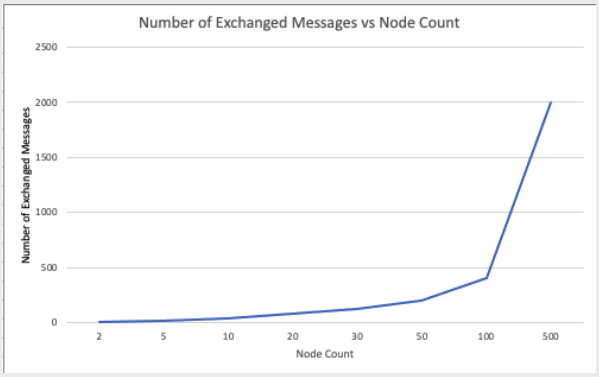

Fig. 10: The Relationship Between Node Count and Number Of Exchanged Messages

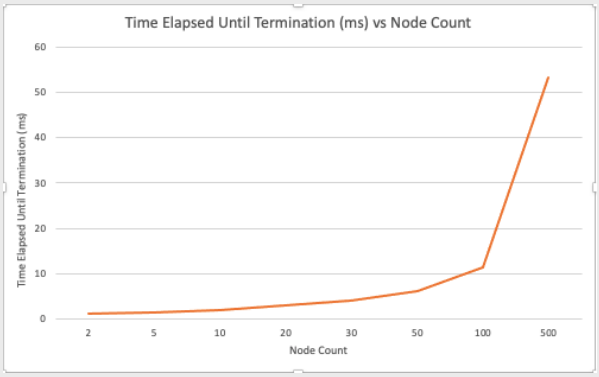

Fig. 11: The Relationship Between the Node Count and the Elapsed Time

For the second experiment, we generate complete topologies with different node counts. As in the first configuration, there is one initiator and each node sends basic message to all of its neighbors. After executing the algorithm for topologies with 2,5, 10, 20, 30, 40 and 50 node counts, we conclude that, the same relationship between the number of edges and the exchanged messages exist in this experiment as well. It seems that, there are not much difference between elapsed time of this experiment with the first experiment. Note that, we could not execute the algorithm for node counts larger than 40 because adhoccomputing library could not create new threads.

Table 2: Message Complexity Analysis of Termination Detection Algorithm on a Complete Topology

Node Count	Time Elapsed Until Termination
2	1.211822748184204
5	1.5393249988555908
10	2.0759570598602295
20	3.2077481746673584
30	4.605236291885376
40	10.92475700378418

For the last experiment, since more than one node can trigger the termination detection algorithm, we analyze the message complexity of a complete topology of 10 nodes with 1 to 5 number of initiators. Other than that, the experiment setup is exactly like that in the first and second experiments. There are 90 wave messages and 90 basic messages exchanged between the nodes. Each of the initiator is able to announce termination so we present the best time elapsed

among different initiator results. To conclude, even though the number of initiators change, number of exchanged messages stay the same. Since each initiator forms its own forest and the number of messages do not change, the elapsed time does not get affected by the initiator count.

Table 3: Message Complexity Analysis of Termination Detection Algorithm on a Complete Topology with Different Number of Initiators

Initiator Count	Time Elapsed Until Termination
1	2.073668956756592
2	2.172826051712036
3	2.1762309074401855
4	2.169590950012207
5	2.158134937286377

1.4.3 Discussion

We run three distinct experiments to analyze the message and time complexity of the algorithm. To distinguish between the experiments, we did not change the underlying computation, that is, we make each process send each of its neighbors basic messages. By not changing the underlying computation, we could observe the changes in the elapsed time over different topologies. For a small number of nodes, less than 50, we observe that the topology does not affect the time elapsed to finish executing the algorithm. Unfortunately, due to the nature of the ad-hoc computing library, we could not compare two different topologies over a large number of nodes. Also, while running experiments, we had to add delays between the events because we observed that, in the absence of the delays, we could not see the exchange of all the messages we sent. To conduct experiments, we created an event to simulate some basic algorithms running in our topologies. Since the expectation is that the basic algorithm is to run separately from the control algorithm, having to add a specific function to pass basic messages to components may conflict with it. The experiment results demonstrated that the complexity of the underlying computation is proportional to the number of edges in the topology in all experiments. For the wave algorithm, our experiment results coincide with the worst-case complexity of the Echo Algorithm. To sum up, we see that the experiment results are in line with the worst-case complexity of the termination detection algorithm.

1.5 Conclusion

The Shavit-Francez algorithm is a reliable and versatile solution for detecting the termination of distributed computations across various network topologies. In our research, we have emphasized the importance of this algorithm by exploring its implementation details and analyzing its message complexity. Our study of the algorithm's message complexity has revealed its favorable features. Using the ad-hoc computing library, we observed a message complexity of O(M + 2E), where M represents the complexity of the fundamental computation. Our experimental results closely match this complexity measure, highlighting the algorithm's efficiency and low overhead in actual distributed systems. If the algorithm is included in the ad-hoc computing library, anyone can use it in their research. We discussed how we implemented the underlying computation logic in Section 1.4.3. In future work, we could approach the issue differently and change the implementation to accept any message kind to reflect the variety of algorithms and make the message-passing mechanism independent from the termination-detection algorithm.

CHAPTER

TWO

ASSESSMENT RUBRIC

Your work and documentation will be assessed based on the following list of criteria.

2.1 Organization and Style

[15 points] The documentation states title, author names, affiliations and date. The format follows this style?

1. Structure and Organization: Does the organization of the paper enhance understanding of the material? Is the flow logical with appropriate transitions between sections?
2. Technical Exposition: Is the technical material presented clearly and logically? Is the material presented at the appropriate level of detail?
3. Clarity: Is the writing clear, unambiguous and direct? Is there excessive use of jargon, acronyms or undefined terms?
4. Style: Does the writing adhere to conventional rules of grammar and style? Are the references sufficient and appropriate?
5. Length: Is the length of the paper appropriate to the technical content?
6. Illustrations: Do the figures and tables enhance understanding of the text? Are they well explained? Are they of appropriate number, format and size?
7. Documentation style: Did you follow the expected documentation style (rst or latex)?

2.2 Abstract

[10 points] Does the abstract summarize the documentation?

1. Motivation/problem statement: Why do we care about the problem? What practical, scientific or theoretical gap is your research filling?
2. Methods/procedure/approach: What did you actually do to get your results?
3. Results/findings/product: As a result of completing the above procedure, what did you learn/invent/create? What are the main learning points?
4. Conclusion/implications: What are the larger implications of your findings, especially for the problem/gap identified?

2.3 Introduction and the Problem

[15 points] The problem section must be specific. The title of the section must indicate your problem. Do not use generic titles.

1. Is the problem clearly stated?
2. Is the problem practically important?
3. What is the purpose of the study?
4. What is the hypothesis?
5. Are the key terms defined?

2.4 Background and Related Work

[15 points] Does the documentation present the background and related work in separate sections.

1. Are the cited sources pertinent to the study?
2. Is the review too broad or too narrow?
3. Are the references/citation recent or appropriate?
4. Is there any evidence of bias?

2.5 Implementation and Methodology

[15 points] Does the documentation present the design of the study.

1. What research methodology was used?
2. Was it a replica study or an original study?
3. What measurement tools were used?
4. How were the procedures structured and the implementation done?
5. Were extensive exprimentations conducted providing not only means but also confidence intervals?
6. What are the assessed parameters and were they adequate?
7. How was sampling and measurement performed?

2.6 Analysis and Discussion

[15 points] Does the documentation present the analysis?

1. Did you collected enough and adequate data for analysis?
2. How was data analyzed?
3. Was data qualitative or quantitative?
4. Did you provide main learning points based on analysis and results?
5. Did findings support the hypothesis and purpose?

6. Did you provide discussion as to the main learning points?
7. Were weaknesses and problems discussed?

2.7 Conclusion and Future Work

[15 points] Does the documentation state the conclusion and future work clearly?

1. Are the conclusions of the study related to the original purpose?
2. Were the implications discussed?
3. Whom will the results and conclusions effect?
4. What recommendations were made at the conclusion?
5. Did you provide future work and suggestions?

CHAPTER

THREE

CODE DOCUMENTATION

3.1 ShavitFrancez.ShavitFrancez

Classes

class ShavitFrancez.ShavitFrancez.**ShavitFrancezComponentModel**(*componentname*, *componentinstancenumber*, *topology*, *context=None*, *configurationparameters=None*, *num_worker_threads=1*, *child_conn=None*, *node_queues=None*, *channel_queues=None*)

on_init(*eventobj*)

on_receiving_send_basic_message(*eventobj*)

on_receiving_detect_termination(*eventobj*)

This method makes the proces receiving the DETECTTERMINATION event active and sets its parent to itself. Then it makes the process send basic messages to all of its neighbors.

on_receiving_basic_message(*eventobj*)

This method makes the process receiving the basic message active if not already and sets its parent to the process sending the basic message. If the process was already active, then it sends an acknowledge message to the process sending the basic message.

on_receiving_acknowledge_message(*eventobj*)

This method decreases the number of children of the process receiving the acknowledge message by one and calls the leave tree procedure for it.

on_receiving_become_passive(*eventobj*)

The process receiving the BECOMEPASSIVE event transitions to passive state if not already passive, and calls the leave tree procedure.

on_message_from_bottom(*eventobj*)

This method calls the related methods according to the message type of the MFRT event.

send_basic_message()

This method increases the number of children of the process and makes the process send basic messages to its neighbors

send_wave_message()

This method sends wave messages to the process' all its neighbors

on_receiving_start_wave(*eventobj*)

This method implements the Echo algorithm on processes that are currently not active and do not have any children.

generate_message(*messagetype*, *messageto*)

leave_tree()

This method checks if the process is not currenty active and does not have any children. According to that, if also the process has a parent, it sends acknowledge message to its parent and leaves the parent's tree. If the process does not have any parent, then it starts a wave.

decide()

announce()

class ShavitFrancez.ShavitFrancez.**ShavitFrancezEventTypes**(*value*, *names=<not given>*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

DETECTTERMINATION = 'DETECTTERMINATION'

BECOMEPASSIVE = 'BECOMEPASSIVE'

SENDBASICMESSAGE = 'SENDBASICMESSAGE'

class ShavitFrancez.ShavitFrancez.**ShavitFrancezMessageTypes**(*value*, *names=<not given>*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

ACKNOWLEDGE = 'ACKNOWLEDGE'

BASICMESSAGE = 'BASICMESSAGE'

WAVE = 'WAVE'

NOTIFYPROCESSES = 'NOTIFYPROCESSES'

Attention: For RST details, please refer to reStructuredText Documentation.

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[ShavitFrancez1986] Shavit, N. and Francez, N. A new approach to the detection of locally indicative stability. In proc. Int. Colloq. Automata, Languages, and Programming (1986), L. Kott (ed.), vol. 226 of Lecture Notes in Computer Science, Springer-Verlag, pp. 344-358.

[Fokking2013] Wan Fokkink, Distributed Algorithms An Intuitive Approach, The MIT Press Cambridge, Massachusetts London, England, 2013

[DijkstraSholten1980] Dijkstra, E. W. and Scholten, C. S. Termination detection for diffusing computations. Inf. Proc. Lett. 11, 1 (1980), 1-4.

[Tel2001] Tel, G, Introduction To Distributed Algorithms, The Cambridge University Press, Cambridge, United Kingdom, 2001

PYTHON MODULE INDEX

S

INDEX

W