Background

In this section, we introduce the concepts behind our Java tool that solves the problem of offline controlled re-execution for disjunctive predicates. First, we give a brief overview of the notations and model of a distributed computation, and then we introduce the notions of *admissible* *sequences* and *true event graph* that are at the heart of algorithm that solves the offline controlled re-execution problem for disjunctive predicated. Finally, we go over that algorithm and analyze its complexity in terms of the number of processes *n* and the number of events *m* per process.

1. Model of a distributed computation

A distributed computation consists of a number of *n* processes that each execute a certain set of *m* events or instructions. The different processes do not share a common physical clock. Rather, they send messages to each other over a set of channels to synchronize their tasks. In this paper, we assume that messages are never lost and are guaranteed to eventually arrive to their respective destination, that is message delivery is reliable. However, we do not assume that the channels on which processes send messages are FIFO, meaning that messages can arrive out of order.

We introduce the notations used throughout the paper and their corresponding denotations in Table 1.

|  |  |
| --- | --- |
| Lowercase letters *e , f* | Local events on a process |
| Greek letters α, ß | Sequence of events |
| *e.proc* | The process on which event *e* occurs |
| *e.happened* | Returns whether the event *e* has happened |
| *e.pred* | The first event locally preceding *e* |
| *e.succ* | The first event locally succeeding *e* |
| *-i* | Initial event on process P*i* |
| *E.* T | Set of finals events on all processes P*0 …* P*n-1* |
|  | Set of initial events on all processes P*0 …* P*n-1* |
|  | Locally precedes |
|  | Remotely precedes |

*Table 1: Notations used and their corresponding denotations*

The set of events happening on different processor in a computation E for a distributed system forms a reflexive partial order. We use the notation to denote a computation in a distributed system, where E is the set of events happening on all processes and the partial order relation known as Lamport’s happened-before relation. The happened-before relation can be defined formally as follows. We say that if any of the two following conditions hold.

A *cut* is a with respect to a computation is formally defined as follows.

A *frontier cut* Cis the set of events in the cut C, whose successor are not in C. Formally, a frontier cut can be defined as follows where C is the frontier cut.

Having formally defined a cut and a frontier cut, we can define a consistent cut. A cut is said to be consistent if for all events *e* belonging to the cut C, all events that happened-before the event *e* must also belong to the cut. Formally, a consistent cut is defined as follows.

A consistent cut C is legal with respect to an admissible sequence α if for all events αi belonging to the sequence α that are in the cut C, all events that happened before αi are also part of a the cut C. Formally, a legal cut C with respect the a sequence of events α can be defined as follows.

1. Admissible Sequence

We know introduce the notion of admissible sequence that is at the heart of the solving the controlled re-execution problem for disjunctive predicates. The follow theorem was proved by Garg and Mittal in [-].

***THEOREM 1***

*A predicate is controllable in a computation iff there exists an admissible sequence of events with respect to B and the computation*

As seen from the above theorem, finding an admissible sequence of events with respect to the predicate and the computation is the key to solving the controlled re-execution problem for disjunctive predicates.

We now define and elaborate upon the notion of an admissible sequence.

For a given computation and a predicate B, an admissible sequence of events α captures some safe execution of events S, where . An admissible sequence of events must obey to certain properties and conditions for any given execution of a computation. These properties and conditions are detailed below.

* **Agreement:** an admissible sequence α should be consistent with the happened-before relation . Formally,
* **Boundary Condition:** an admissible sequence α must start with an initial state in the computation and end in a final state in the computation. Formally, the boundary condition is defined as follows. where is the first event in the sequence, is the last event in the sequence and *n* is the number of events in the sequence α.
* **Continuity:** an admissible sequence α must be a continuous execution. We can define this property more formally as follows.
* **Safety:**the admissible sequence α must be a safe execution. Meaning, that for any cut that is legal with respect to the sequence α and a predicate B, and where some event in α belongs to the frontier of C, the predicate in question is true. That is

1. True Event Graph

Another notion that is at the heart of solving controlled re-execution problem for disjunctive predicate is the true event graph (TEG). Given the trace of a computation and a disjunctive predicate B, a true event graph is constructed based on the following conditions and considerations.

* An event *e* belongs to the true event graph if the local predicate for the process *e.procc* is true at the event *e*.
* There exists an edge between two events *e* and *f* in the true event graph if
* An event *e* in the true event graph is labeled “initial” if
* An event e in the true event graph is labeled “final” if

We now present the following two theorems that are crucial to solving the controlled re-execution problem for disjunctive predicates. The two theorems presented are proven in [-]. It is clear from the two theorems presented that the key to solving the problem is to construct the true event graph *G* and to find the shortest permissible path in *G*.

***THEOREM 2***

*Let G = (ν, ε) be the true event graph corresponding to a disjunctive predicate B and a computation (E, →). The shortest permissible path in G, if it exists, corresponds to an admissible sequence of events.*

***THEOREM 3***

*Let G = (ν, ε) be the true event graph corresponding to a disjunctive predicate B and a computation (E, →). If B is controllable in (E, →) then there exists a permissible path in G.*