

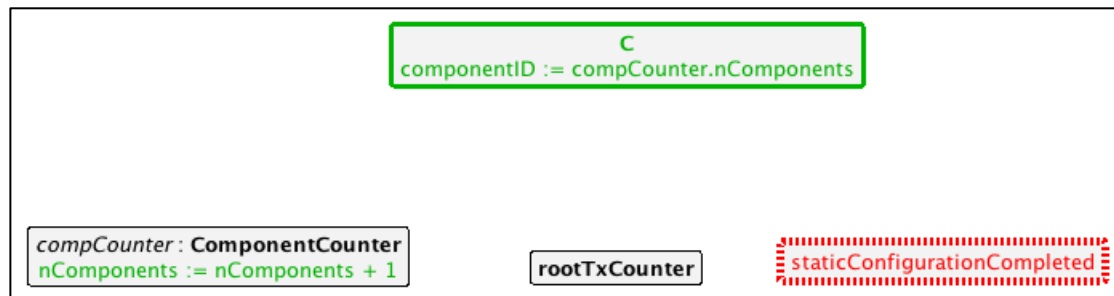
## Graph Grammar for Version Consistency

This document collects the rules showing the evolution of dynamic dependencies on the basis of the distributed management algorithm.

### 1. Definition of the Static Configuration

This set of rules support the definition of a static configuration with an arbitrary number of components and interconnections. Once the definition is completed rules StaticConfigurationCompleted can be applied to proceed with the execution of the algorithm

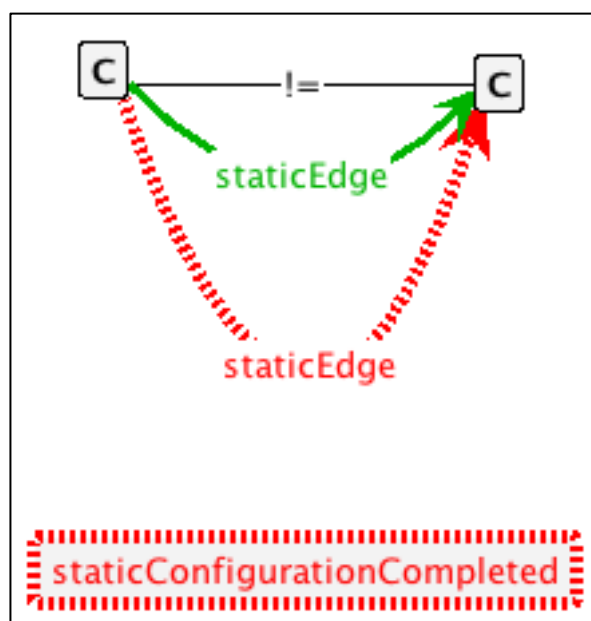
#### Rule 1: AddComponent



#### Description:

If the static configuration is not completed, it is possible to add new components. Each component has an unique id starting from 0. The node compCounter keeps track of the number of created components to maintain the id of the component unique (the first created component has id 0, the second id 1, ...)

#### Rule 2: AddComponent

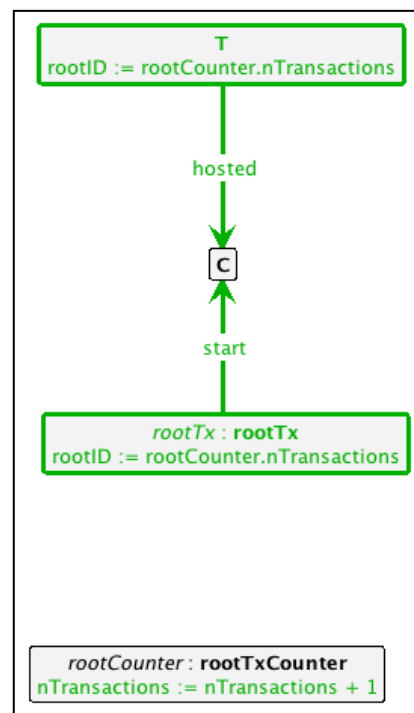


**Description:**

If the static configuration is not completed, it is possible to add a static edge between two components (two distinct instances of type C).

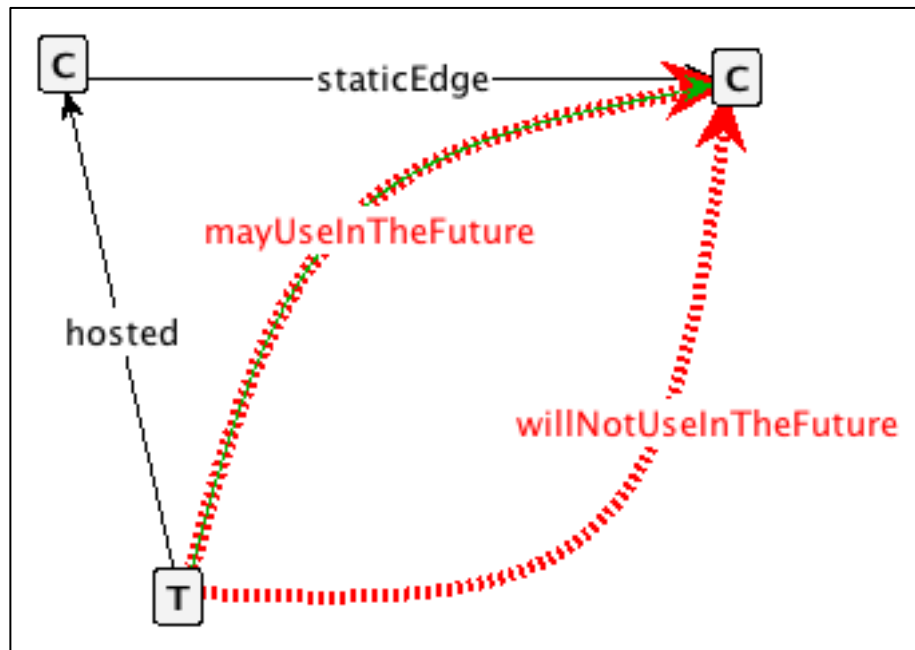
**Rule 3: StaticConfigurationCompleted**

When the definition of the static configuration is completed we can fire this simple rule. It represents a flag that avoids that new components are added or connected while the algorithm is running.

**2. Modeling transaction****Rule 3: StartRootTransaction****Description:**

A root transaction  $rootTx$  can start in any component. It is identified by  $rootID$  that is unique thanks to the global counter  $rootCounter$ . The consequence of the initiation of the root transaction is that the component hosts a Transaction with  $rootID$  equals to the id of the node  $rootTx$ . In this way we can also specify that a  $rootTransaction$  is also a transaction.

#### Rule 4: addToFutureSet



#### Description:

A transaction represents a local part of the logic of the application.

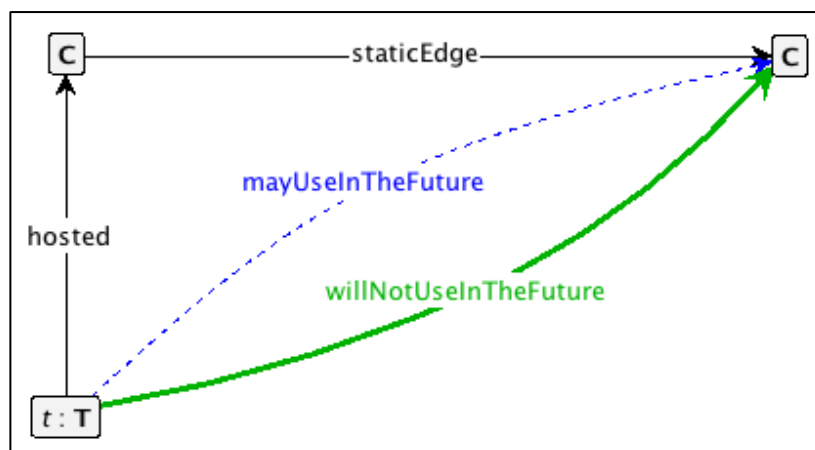
The information we need for the algorithm (and that we can infer by the static analysis of the code) regards the possibility of invoking a certain component in the future. This is modeled with an arc called **mayUseInTheFuture** starting from a generic transaction to a statically dependent component.

The opposite condition, that is that a transaction will never invoke in the future a certain component, is modeled with the arch **willNotUseInTheFuture**

By adding the arcs **mayUseInTheFuture** and **willNotUseInTheFuture**, we model the insertion and removal of components in the Future Set.

This rule states the following: if **mayUseInTheFuture** is not already present and if **willNotUseInTheFuture** is not present, an arc **mayUseInTheFuture** can be added to the graph connecting the transaction to the component which may be used in the future.

#### Rule 5: removeFromFutureSet

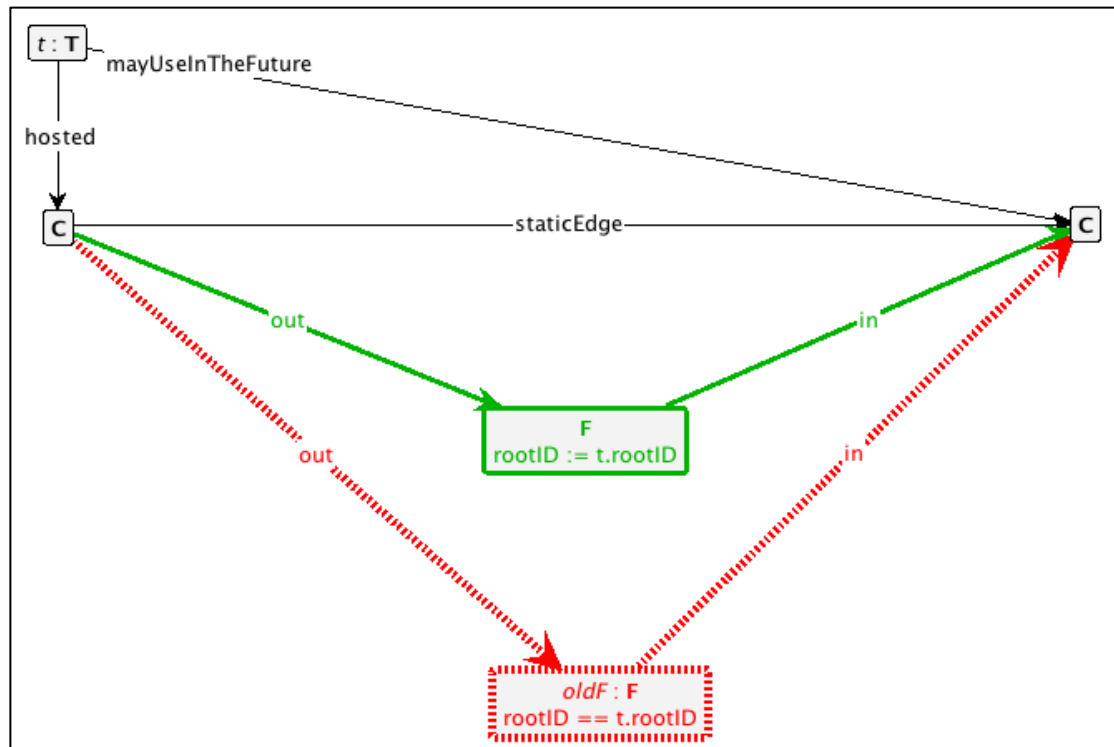


#### Description:

This rule is the dual of Rule4 and models the removal of a component from the future set. It states the following: If a component will not be used anymore by a given transaction, the arc **mayUseInTheFuture** is removed and the arc **willNotUseInTheFuture** is added.

### 3. SetUp step

#### Rule 6: addDirectF



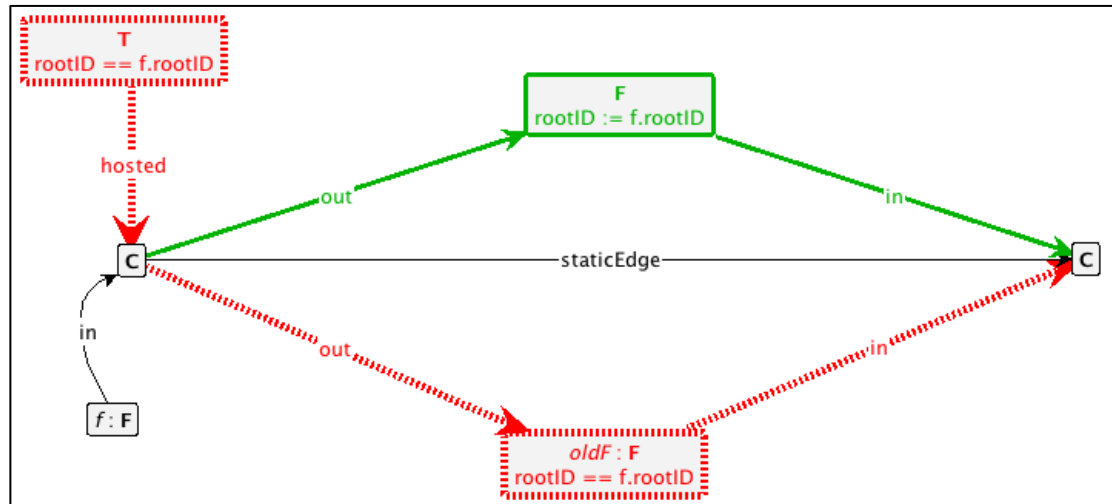
#### Description:

This rule is applied for component which just initiated hosting the root transaction. The root transaction just started and therefore it is possible to know which components can be used in the future.

A future edge can be added from one component hosting a transaction to another component if:

- 1) it does not already exist a future edge linking the same components and labeled with the same rootID.
- 2) The component sending the message for adding the edge is hosting a transaction and that transaction may use in the future the component receiving the message.

### Rule 7: addRecursiveF

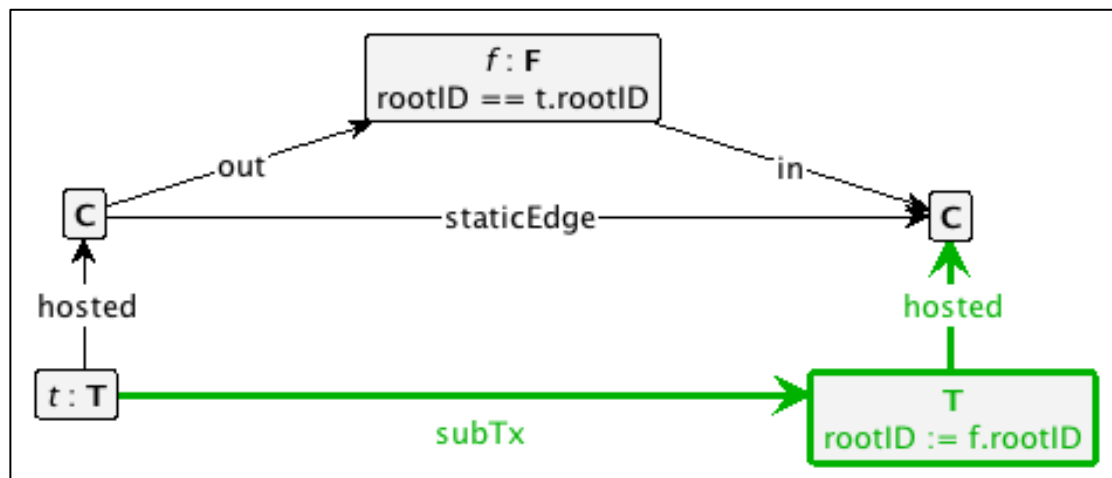


**Description:**

This rule is applied for “recursively” propagating the future edges. Whenever a component received a future edge, it immediately propagate the edges to all the other statically dependent components. The combination of Rule 6 and Rule 7 defines the setup step.

#### 4. Progress step

### Rule 8: startSubTx

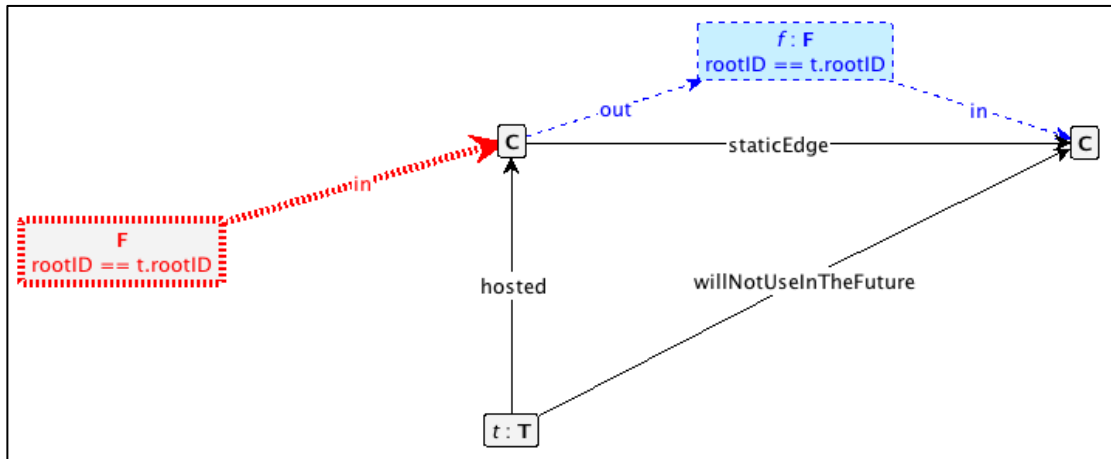


### Description

A new subtransaction of a given transaction  $t$  can be initiated if:

- 1) there exists a static edge between the two components
- 2) there exists a future edge between the two components

### Rule 9: RemoveF

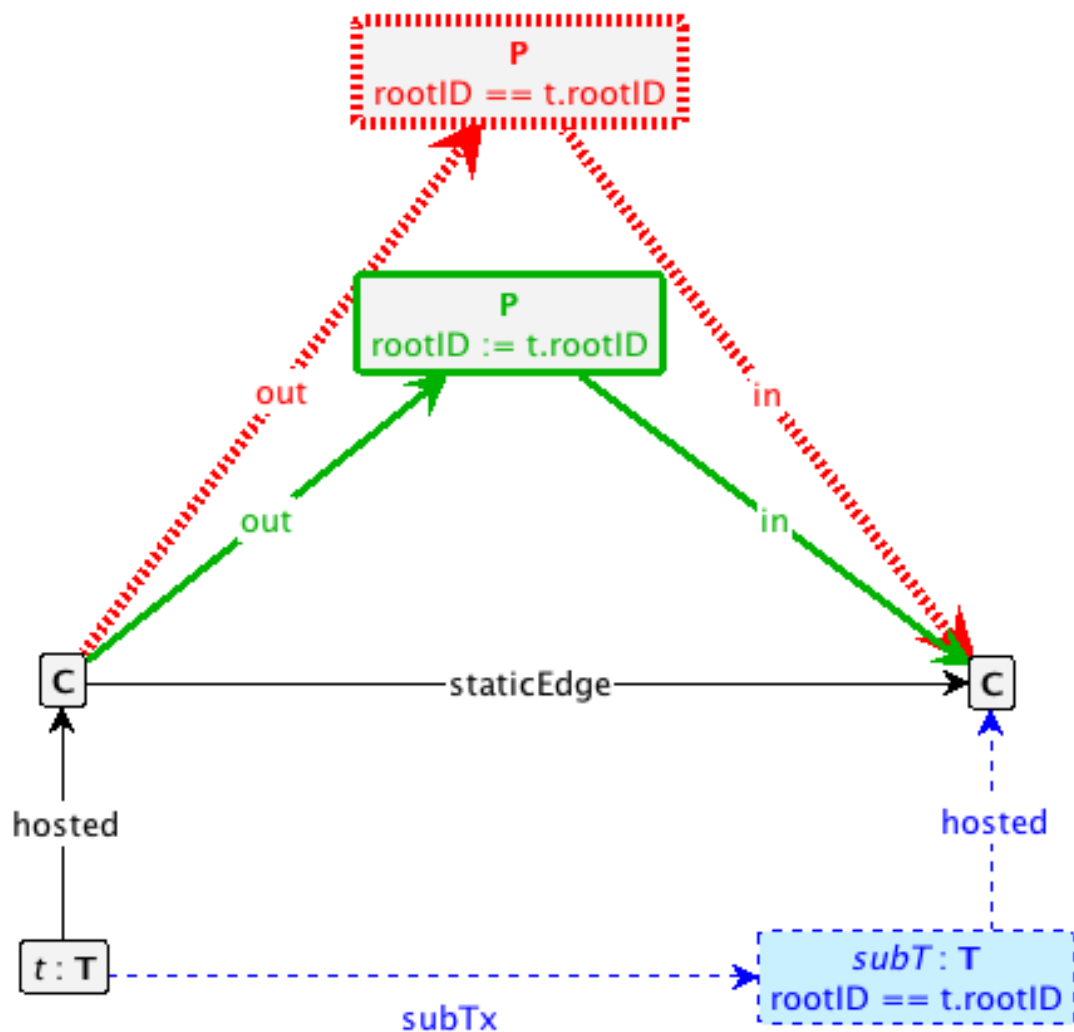


### Description:

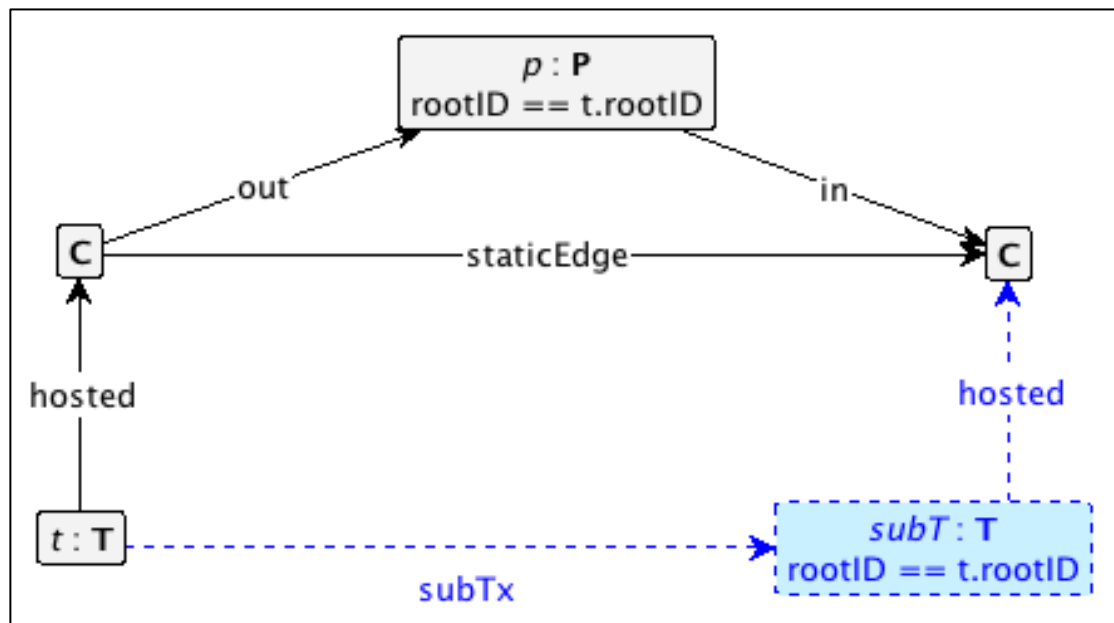
A future edge between two components can be removed if

- 1) the source component is hosting a transaction that  $willNotUseInTheFuture$  the target component (see Rule 5)
- 2) The source component does not have incoming future edges labelled with the same  $rootID$

### Rule 10: endSubTx\_addPastEdge



### Rule 11: endSubTx

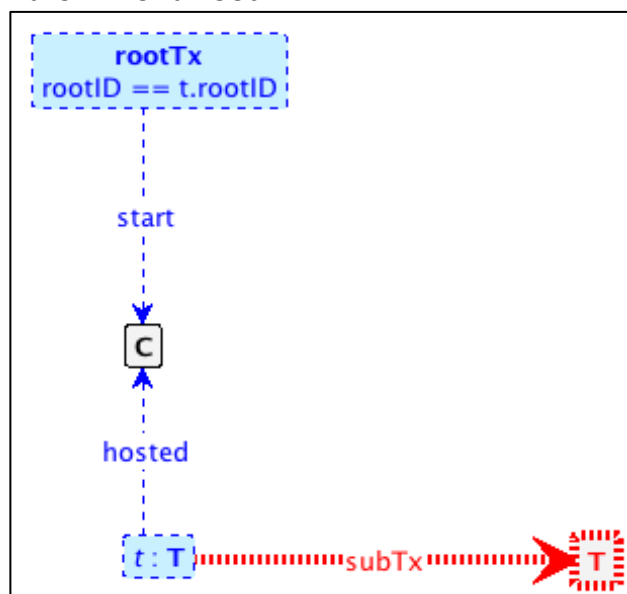


### Description

If a subtransaction terminates, a past edge is created from the initiating component to the target component.

If the past edge is already present only the arcs and nodes related to the subtransaction is removed and no other past edges are created.

### Rule 12: endRootTx

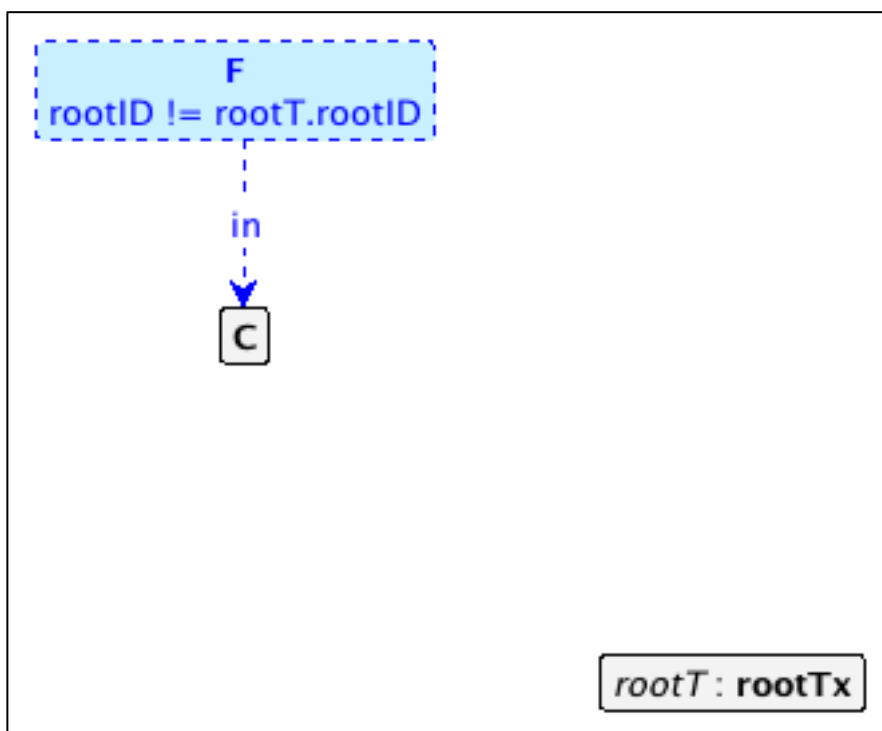
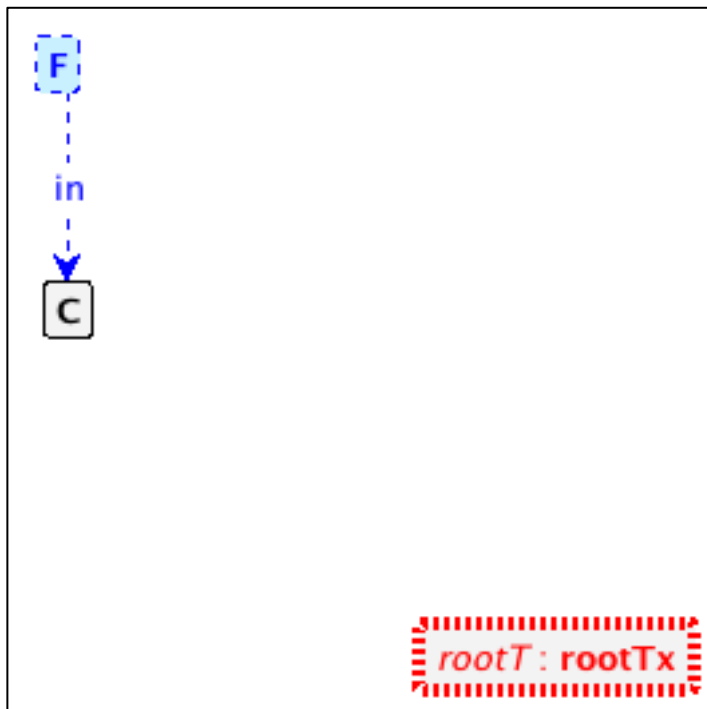


### Description:

A root transaction can end (and thus removed from the graph) only if there are no other initiated subtransactions in execution.

## 5. Cleanup step

Rules 13 and 14: removeDanglingF

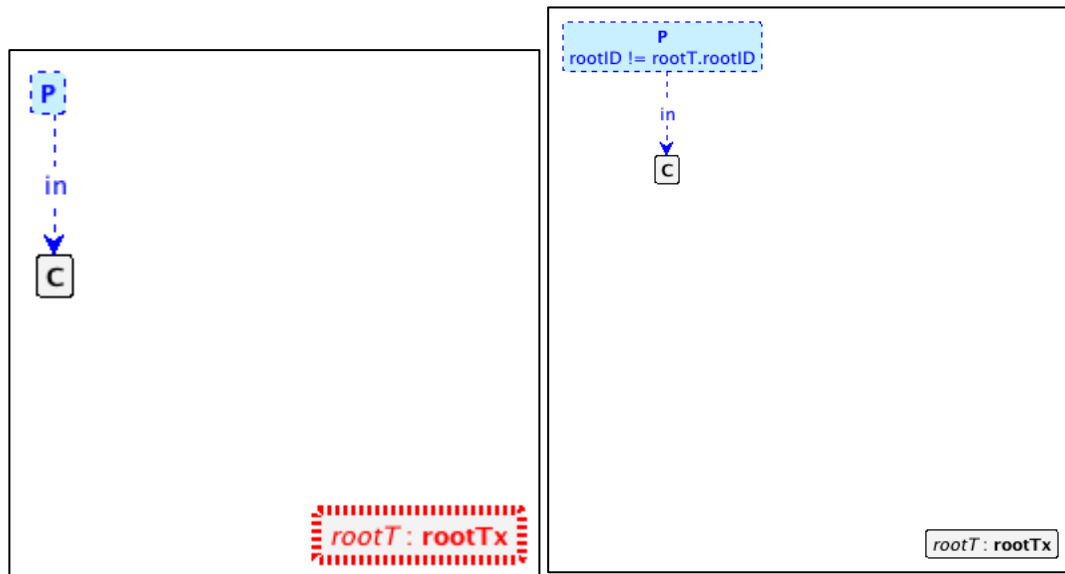


### Description:

Remove dangling future edges if no rootTransactions are in execution(rule 13) or if the rootTransactions in execution have a different rootID w.r.t. the future edges to be removed (rule 14).



## Rules 15 and 16: RemoveDanglingP

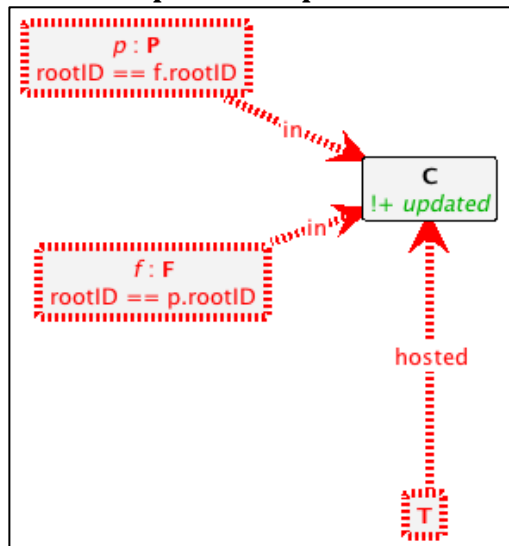


### Description:

As for future edges (rules 13 and 14), remove dangling past edges if no rootTransactions are in execution (rule 15) or if the rootTransactions in execution have a different rootID w.r.t. the past edges to be removed (rule 16).

## 6. Freeness condition

### Rule 17: updateComponent



### Description:

A component is free, if it is not hosting any transaction (this substitutes the original condition on the self edges), **and** there are no incoming past and future edges labelled with the same rootID.

## Proving that the grammar satisfies the properties of dynamic dependencies.

- 1.(host-validity) Given a transaction  $T$ , the hosting component  $C$  must have a dedicated pair of edges  $C - \frac{future}{root(T)} - > C$  and  $C - \frac{past}{root(T)} - > C$  during the whole duration of  $T$  (self-edges).

**We remove the notion of self-edges and change the freeness property accordingly. Therefore we do not need anymore this property**

- 2.(locality) Each future or past edge  $C - \frac{future(past)}{T} - > C'$  must be paired with a static edge  $C - \frac{static}{T} - > C'$ ;

**Proof.** The property holds by construction. The creation of future edges are performed as a result of the application of Rule 6 and Rule 7. In these rules a future edge can only be created if it exists a paired static edge. The same is true for past edges w.r.t. to Rule 10.

3. (future-validity) A future edge  $C - \frac{future}{T} - > C'$  begins to exist no later than the first time a transaction  $T' \in \text{ext}(T)$  is initiated, and continues to exist at least until no transactions hosted by  $C$  will initiate further  $T'' \in \text{ext}(T)$  on  $C'$  through  $C - \frac{static}{T} - > C'$ ;

**Proof.** Let us divide the proof in two parts. In the first part we prove that a future edge  $C - \frac{future(past)}{T} - > C'$  begins to exist no later than the first time a transaction  $T' \in \text{ext}(T)$  is initiated. This is true because the creation of future edges occur only in the setup step (rule 6 and rule 7). In this step no subtransactions have been already initiated on the target component. Moreover, the initiation of the subtransaction occurs in rule 8 which requires already the presence of a future edge (labeled with rootID of the subtransaction to be initiated) between the source component and the target component. In the second part we prove that a future edge  $C - \frac{future(past)}{T} - > C'$  continues to exist at least until no transactions hosted by  $C$  will initiate further  $T'' \in \text{ext}(T)$  on  $C'$  through  $C - \frac{static}{T} - > C'$ . The removal of future edges occurs in three rules: Rule 13, Rule 14, and Rule 9. Rules 13 and 14 are applied during the cleanup step. This means that the root transaction  $T$  has already terminated. According to Rule 12, no other subtransactions  $T'' \in \text{ext}(T)$  can be in execution on  $C'$  and, given the fact that  $T$  terminated, according to Rule 11, no further subtransactions can be initiated. A future edge can be removed also during the progress step in Rule 9. However, Rule 9 to be applied (and therefore to remove the future edge), requires that no future edges are incoming on  $C$ . If no future edges are incoming to  $C$ , according to Rule 8, no further transactions can be initiated on  $C$ . After

the application of Rule 9, again because of Rule 8, no further subtransactions  $T'' \in \text{ext}(T)$  can be initiated on  $C'$ .

4. (past-validity) A past edge  $C - \frac{\text{past}}{T} - > C'$  begins to exist no later than the end of any transaction  $T' \in \text{ext}(T)$  initiated by a transaction hosted by  $C$  on  $C'$  through  $C - \frac{\text{static}}{} - > C'$ , and continues to exist at least until the end of  $T$ .

**Proof.** The proof is divided in two parts. The first part we prove that a past edge  $C - \frac{\text{past}}{T} - > C'$  begins to exist no later than the end of any transaction  $T' \in \text{ext}(T)$  initiated by a transaction hosted by  $C$  on  $C'$  through  $C - \frac{\text{static}}{} - > C'$ . This is true since the creation of past edges only occurs in Rule 10. When Rule 10 is applied, the termination of the first subtransaction on  $C'$  implies the contemporary creation of a past edge. The termination of future subtransactions is modeled in Rule 11 which, to be applied, requires the presence of the past edge. In the second part, we prove that a past edge  $C - \frac{\text{past}}{T} - > C'$  continues to exist at least until the end of root transaction  $T$ . This is true since the removal of past edges occurs only during the cleanup step by applying Rule 15, and Rule 16. The application of these rules requires that the root transaction  $T$  is already terminated.