A Model of Time Dependent Behavior in Concurrent Software Systems

Debra S. Lane

Technical Report No. 87-28

University of Calfornia at Irvine
Department of Information and Computer Science

November 2, 1987

ABSTRACT

A great difficulty in building distributed systems lies in being able to predict what the systems behavior will be. A distributed or communicating system is defined here to be one in in which the hardware consists of a set of processors each with their own memory, connected by some communication medium (there is no shared memory), and the software is assumed to be of the CSP (Hoare's Communicating Sequential Processes) type.

In the past few years some theories have been proposed to model features of communicating systems. Milner's Calculus of communicating Systems (CCS), Winskel's Synchronization Trees (ST), Hennessy's Acceptance Trees (AT), and Hoare and Brookes's theory of communicating processes are examples of formal models of such systems. All of these models concentrate on modelling observable properties of a system.

Event Dependency Trees (EDT) is a new representation of communicating systems that models the time dependent nature of such systems. None of the representations mentioned above explicitly represent time but time is precisely the factor that introduces so much variability and complexity into such software and systems. EDT provides a representation based on trees and a set of operations over the EDT trees that can be used to produce a representation of the system behavior. The model supplies potentially important information for the design and construction of distributed, parallel software systems.

A Model of Time Dependent Behavior in Concurrent Software Systems

Introduction

A great difficulty in building distributed systems lies in being able to predict what the system behavior will be. A distributed or communicating system is defined here to be one in which the hardware consists of a set of processors each with their own memory, connected by some communication medium (there is no shared memory), and the software is assumed to be of the CSP (Hoare's Communicating Sequential Processes) type. The problem is that while it is easy to understand how each process behaves in and of itself, it is nearly impossible to predict all the ways in which the processes will interact and influence each other's execution. It is necessary to understand their interaction in order to determine how the system behaves (so that one might convince oneself or others that the system performs as intended).

In the past few years some theories have been proposed to model features of communicating systems. Milner's Calculus of Communicating Systems (CCS) [MILN80], Winskel's Synchronization Trees (ST) [WINS84], Hennessy's Acceptance Trees (AT) [HENN85B], and Hoare and Brooke's theory of communicating processes [BROO84] are examples of formal models of such systems. All of these models concentrate on modelling observable properties of a system.

This paper presents a new representation of communicating systems called Event Dependency Trees (EDT) that models the time dependent nature of such systems. None of the representations mentioned above explicitly represent time but time is precisely the factor that introduces so much variability and complexity into such software and systems. Many models in computer science assume that events occur instantaneously, but here it is assumed that every event occurs with a certain time delay represented explicitly by an event name and a variable for the time delay. Communication events are important because that is how processes interact. Events preceding the communication events, even if they are only executions of sequential pieces of code, are also very important, however, because they determine the exact manner in which the communication events will occur.

Besides modelling time explicitly, EDT differs from CCS, ST, and AT in its representation of system behavior. Both CCS and ST represent system behavior as interleavings of events. The combine tree operation in those models produces the set of interleavings. AT represents the system as a state–transition graph. The tree combine operation in AT takes two state–transition graphs and produces a larger one. In EDT, the system behavior is represented as a partial ordering of events. The combine tree operation in EDT produces the partial ordering of events in a way that indicates how particular sets of events contend with each other to produce the various execution paths.

EDT show the right amount of information about system behavior, not too much as in an interleaving representation, and not too little as in a state–transition model. It is possible to identify each execution path by its *unique* event ordering. In interleaving many event orderings produce the same execution path because many times it is irrelevant that some event occurred before or after another since they don't influence each other's execution. EDT shows exactly those events that influence each other's execution and also those that are not related.

EDT also provides answers to the questions "Why is one execution path chosen over another?" or "How is a particular execution path chosen?" The answer is that some set of events occurs before a different, contending set of events. CCS, ST, and AT all show the possible execution paths but indicate only that they arise because of nondeterminism. What is the source of such nondeterminism? There are two ways in which nondeterminism arises in such systems: (1) through the use of guarded commands, and (2) through the use of the communication constructs. EDT models the nondeterminism that arises through the use of communication constructs in CSP–type languages.

This paper tries to provide an intuitive feel for the structure of Event Dependency Trees, their operations, and how they model time dependent behavior (i.e., their explicit representation of time and depiction of system behavior).

Event Dependency Trees

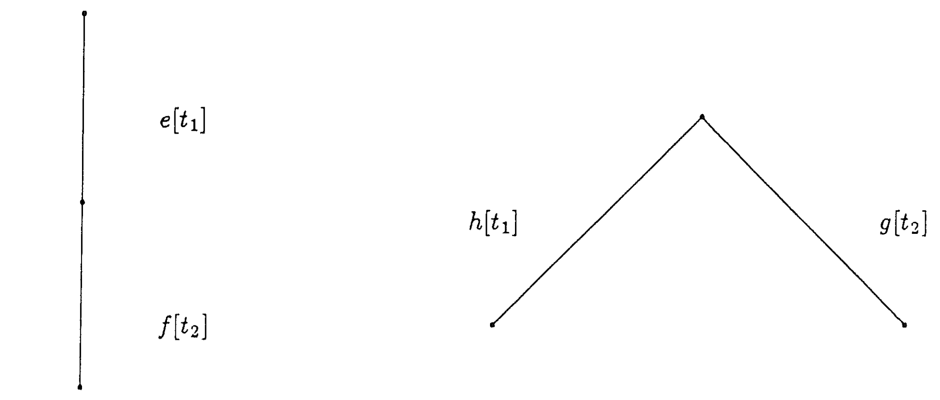
The primary motivation for developing Event Dependency Trees (EDT) is to provide a technique for managing the complexity that arises when a piece of software is composed of many communicating processes. Since EDT tries to capture the manner in which interprocess communication determines the course of execution, for the present time the internal structure of processes is ignored. It is assumed that a process is only a sequential execution of events; control structures are not modelled.

Notation

Trees

Components of a computation are represented as trees in which each arc is labelled with an event and an associated time delay. It is assumed that there is no time overhead associated with events other than what is shown. For example, the label $e[t_1]$ means that t_1 is the amount of time it takes for event e to occur. All trees are composed from the following (see Figure 1).

1) **Sequence** – $e[t_1]$ occurs and then $f[t_2]$ occurs, i.e., e must complete before f can begin. t_1 may equal t_2.



(a) sequence (b) choice

Figure 1

Primitive Event Dependency Trees

2) **Choice** – if $t_1 < t_2$ then event h will occur, if $t_2 < t_1$ then g is executed, it is never the case that $t_1 = t_2$, one event or the other will always occur first.

Events

There are three types of events: communication, execution, and null. The communication events are assumed to be synchronized message exchanges, where there is a receiver (or passive participant) and a sender (or active participant). Therefore, communication events are further subdivided into three types: (1) a receiving communication event, (2) a sending communication event, and (3) a synchronized communication event. The null event is graphically represented as a tree with only a root node, and this is called the null tree. The following notation is used:

1) $\overrightarrow{e[t]}$ denotes a sending communication event that takes time t.
2) $\overleftarrow{e[t]}$ denotes a receiving communication event that takes time t.
3) $\overset{\leftrightarrows}{e[t]}$ denotes a synchronized communication event that takes time t.
4) $e[t]$ denotes an execution event that takes time t.
5) τ_0 denotes the null tree, which is also the null event.

These are the only events that can occur in EDTs. Using this model, all portions of the computation that take time are accounted for.

Labelling trees is subject to some restrictions. First, three functions on labels are defined. ε represents the empty string. Each event has a name, e, a time, t, and a type (exec, send, recv, sync, or null). The name of the null tree is ε or the empty string and the time of the null tree is 0. the functions *name*, *type*, and *time*, when applied to an event, return the respective information about that event.

Each arc of a tree contains an event label; the label consists of a name, a type, and a time. Event names will be taken from some alphabet of Roman characters. Event times will be denoted by the variable t and a subscript, e.g., t_1, t_2, t_{33}. $\alpha, \beta, \gamma, \delta$ will be used as variables that range over a set of event labels or event names. Let A be some alphabet. Some additional sets are defined. $\overrightarrow{A}$ is a set of labels $\ni \forall \alpha \in A$, there is a corresponding $\beta \in \overrightarrow{A}$ such that α and β have the same names but $type(\alpha) = exec$ and $type(\beta) = send$. $\overleftarrow{A}$, and $\overset{\leftrightarrows}{A}$ are defined analogously. $\forall \alpha \in A \exists \beta \in \overleftarrow{A} \ni name(\alpha) = name(\beta),\ type(\alpha) = exec,\ type(\beta) = recv.$ $\forall \alpha \in A \exists \beta \in \overset{\leftrightarrows}{A} \ni name(\alpha) = name(\beta),\ type(\alpha) = exec,\ type(\beta) = sync.$ Thus, $a \in A \Rightarrow \overrightarrow{a} \in \overrightarrow{A}, \overleftarrow{a} \in \overleftarrow{A}$, and $\overset{\leftrightarrows}{a} \in \overset{\leftrightarrows}{A}$. Let $\mathcal{A} = A \cup \overrightarrow{A} \cup \overleftarrow{A} \cup \overset{\leftrightarrows}{A}$.

Usually, a set of trees (sometimes called a forest) will be used to represent some processes. Let L be the set of labels for the forest. Although the time portion of the label has been temporarily ignored, it is assumed that each $\alpha \in L$ has an associated time, t_i where $i \in \mathcal{NAT}$. The restrictions on labelling the forest are as follows:

1) $L \subset \mathcal{A}$.

2) $\forall \alpha, \beta \in L$, if $type(\alpha) = exec$ then $name(\alpha) \neq name(\beta)$. In other words, the name of execution events is unique.

3) $\forall \alpha, \beta \in L$, if $type(\alpha) = recv$ then $name(\alpha) = name(\beta)$ only if $type(\beta) = send$. This says that there is only one receive event with any given name, but there could be many send events with the same name.

4) $\forall \alpha, \beta \in L$, if $type(\alpha) = sync$ then $name(\alpha) = name(\beta)$ only if $type(\beta) = send$ or $type(\beta) = sync$. For any synchronized event, there can be send events with the same name or other synchronized events with the same name.

5) $\forall \alpha, \beta \in L$, if $type(\alpha) = send$ then $name(\alpha) = name(\beta)$ only if for all other events $type(\beta) = send$ or $type(\beta) = recv$, or for all other events $type(\beta) = send$ or $type(\beta) = sync$. Send events can have the same name as other send events and a receive event, or, other send events and synchronized events.

6) $\forall \alpha, \beta \in L$, if $time(\alpha) = t_i$ $time(\beta) = t_j$ then $i \neq j, i, j \in \mathcal{NAT}$. Each label must have a unique time variable.

There are some further labelling restrictions on any single tree in the forest. If α, β are labels within a single tree, then $name(\alpha) = name(\beta)$ only if $type(\alpha) = type(\beta) = sync$. The only time labels in a single tree can have the same name is if the events with the same name are synchronized events. Synchronized events arise only as the result of a binary operation on trees called combine that is defined later in this chapter. Thus, if there are no combined trees in a given set of trees, then the restrictions for labelling any single tree in the set imply that all labels for that tree have distinct names.

Functions on Trees

Communication events are important because they denote interaction between processes. The notion of matching communication events, which occurs between trees not within a tree represents this interaction.

Definition 2.2. *Let $\mathcal{A}$ be a set of events. $\forall \alpha, \beta \in \mathcal{A}$, α and β are matching communication events, denoted $\alpha \overset{mce}{=} \beta$ if and only if*

i) $name(\alpha) = name(\beta)$,

ii) $type(\alpha) = send$ *and* $type(\beta) \in \{recv, sync\}$ *OR* $type(\alpha) \in \{recv, sync\}$ *and* $type(\beta) = send$.

Thus, matching communication events are two events with the same event name in which either (i) one is a receiving communication event and one is a sending communication event, e.g., $c[\overleftarrow{t_2}]$ and $c[\overrightarrow{t_1}]$, or (ii) one is a synchronized communication event and one is a sending communication event, e.g., $c[\overset{\leftrightarrows}{t_1}]$ and $c[\overrightarrow{t_2}]$.

Now, given two arbitrary trees, it is necessary to determine whether or not they have matching communication events and if they do, to identify them. First, it is necessary to be able to talk about the events contained in some tree. Some more notation is required.

Definition 2.3. *$\mathcal{L}_\tau$ is the set that contains all the event labels in tree τ.*

So, for example, $\mathcal{L}_\tau$ of τ in Figure 1 equals the set $\{a[t_1], b[t_2], c[t_3], d[t_4]\}$.

Next, a function $\mathcal{COMM}$ is defined that takes an EDT and maps it to a list of the communication events it contains.

Definition 2.4. *Let τ be some EDT. $\mathcal{COMM}(\tau) = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ where $\forall i \in \{1, \ldots, n\}, \alpha_i \in \mathcal{L}_\tau$, $type(\alpha_i) \in \{send, recv, sync\}$ and there does not exist any $\beta \in (\mathcal{L}_\tau \setminus \{\alpha_1, \ldots, \alpha_n\}) \ni type(\beta) \in \{send, recv, sync\}$.*

It is now possible to determine if two trees have any matching communication events. Some notation is provided to represent that fact. For the following definitions, let $\mathcal{EDT}$ be a set of EDT .

Definition 2.5. *Let $\tau, \mu \in \mathcal{EDT}$, and $\mathcal{COMM}(\tau) = (\alpha_1, \ldots, \alpha_n)$, $\mathcal{COMM}(\mu) = (\beta_1, \ldots, \beta_m)$. If $\exists i \in \{1, \ldots, n\}$ and $\exists j \in \{1, \ldots, m\} \ni \alpha_i \overset{mce}{=} \beta_j$, then $\mathcal{COMM}(\tau) \ominus \mathcal{COMM}(\mu)$.*

$\mathcal{MATCH}$ is a function that maps two trees to a list of all their matching communication events.

Definition 2.6. *Let $\tau, \mu \in \mathcal{EDT} \ni \mathcal{COMM}(\tau) \ominus \mathcal{COMM}(\mu)$. $\mathcal{MATCH}(\tau, \mu) = ((\alpha_{i_1}, \ldots, \alpha_{i_k}), (\beta_{j_1}, \ldots, \beta_{j_k}))$ where $k \in \{1, \ldots, min\{n, m\}\}$ and $\alpha_{i_\ell} \overset{mce}{=} \beta_{j_\ell}$.*

If $\mathcal{MATCH}$ contains more than one pair of matching communication events, then if the portion of the multiple pairs in one tree occurs in a chain, then the respective portion in the other tree must also occur in a chain. There cannot be branch nodes occurring between one portion of the pair in one tree and not in the other. The reason is that the resulting tree will contain a deadlock. For further details see [LANE87].

There are two more pieces of information that are needed: the length of the path from the root node to some designated event in the tree, and a representation indicating which branches to take to arrive at the designated event, beginning at the root of the tree.

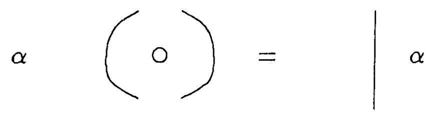
Definition 2.7. *Let $\tau \in \mathcal{EDT}$, $\alpha \in \mathcal{L}_\tau$. $\mathcal{PATH}(\tau, \alpha) = n$, where $n \in \mathcal{NAT}$ is the length of the path from the root node to α.*

Definition 2.8. *Let $\tau \in \mathcal{EDT}$, $\alpha \in \mathcal{L}_\tau$, r the root node, and ε the empty string. $\forall \alpha, \forall s \in \mathcal{NAT}^*$, and $\forall i \in \mathcal{NAT}$,*

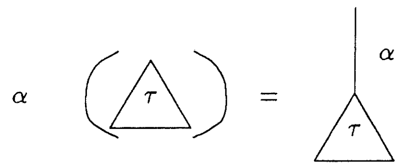
i) $\mathcal{DEST}(\alpha, \varepsilon) = \alpha$,

ii) $\mathcal{DEST}(\alpha, si) =$ *the ith child of* $\mathcal{DEST}(\alpha, s)$.

$\mathcal{DEST}$ is not defined in some cases (e.g., the third child of a node with only two children).



(a) $\alpha(\tau_0) = \alpha$



(b) $\alpha(\tau) = \alpha \sum_{i=1}^{n} \beta_i \tau_i$

Figure 2

The Prefix Operation

Definition 2.9. *Let $\tau \in \mathcal{EDT}$, $\alpha \in \mathcal{L}_\tau$, and r the root node. $\mathcal{ROUTE}(\tau, \alpha) = s \ni s \in \mathcal{NAT}^*$ and $\mathcal{DEST}(\alpha, s) = r$.*

Operations

In the beginning of the chapter the meaning of nodes and arcs in Event Dependency Trees was described. This section defines operations on trees that illustrate how EDTs are constructed. The operations preserve the meaning of branches in the trees.

Prefix

The prefix operation is a very simple operation. It allows events to be added to trees. See Figure 2.

Definition 2.8. *Let α be any event in $\mathcal{L}$, $\tau = \sum_{i=1}^{n} \beta_i \tau_i$ be some tree. Then*

i) $\alpha(\tau_0) = \alpha$,

ii) $\alpha(\tau) = \alpha \sum_{i=1}^{n} \beta_i \tau_i$.

Combine

A very important operation is one that combines two trees producing another tree. This can be thought of as taking two concurrent processes and showing how they interact and affect each other. If the two processes do not exchange information (i.e., they don't send messages to each other), then they will not affect each other and the corresponding trees that represent them will be denoted as a tuple (of trees) called a pseudo tree. Each pseudo tree is actually a forest of trees. Two trees will be combined into a single (new) tree when they have matching communication events. The tree that contains the sending communication event will be referred to as the *active tree* and the tree that contains the other event in the matching communication events pair, the *passive tree.*

The combine operation takes two trees that contain at least one pair of matching communication events and produces a single tree as follows:

i) The matching communication events form a single synchronized communication event.

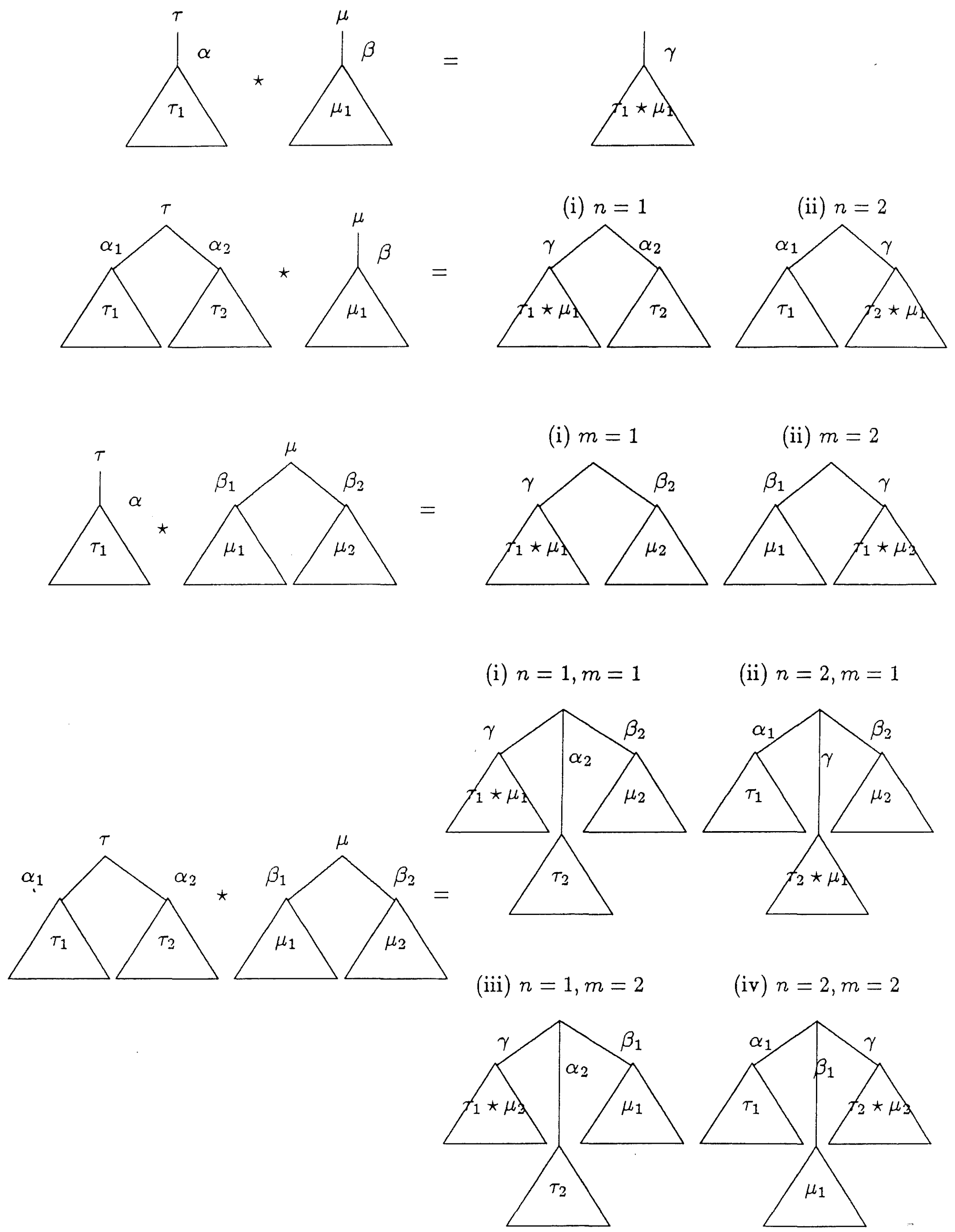
ii) If the passive tree contains a receiving event, then a single path from the root of the new tree is formed from the paths in each of the two original trees that contain the matching communication events. Except for the arc labelled by the new synchronized communication event, the arcs on the new path are labelled by tuples of events, (α, β), where α represents the event on the active tree, and β represents the event on the passive tree.

iii) If the passive tree contains a synchronized event, then two paths from the root of the new tree are formed. One path is the same path in the passive tree that contains the synchronized communication event. The other path is formed by creating tuples of events. The first element in each tuple is taken from the active tree and the second element is taken from the passive tree.

iv) The remaining parts of the two trees are reproduced in the new tree.

There are four cases that arise when combining two trees that contain matching communication events. Each tree is broken into a subtree prefixed by an arc. The new tree is defined in terms of event labels and subtrees from the original trees. Selective subtrees are again combined. Referring now to Figure 3, assume δ_1 and δ_2 are the pair of matching communication events such that $\delta_1 \in \mathcal{L}_\tau$, τ the active tree, and $\delta_2 \in \mathcal{L}_\mu$, μ the passive tree. Furthermore, assume that $\mathcal{PATH}(\tau, \delta_1) = \mathcal{PATH}(\mu, \delta_2)$. The final piece of information necessary is the location of each event in the matching communication events pair in each tree. Let $\mathcal{ROUTE}(\tau, \delta_1) = ns, n \in \mathcal{NAT}, s \in \mathcal{NAT}^*$ and $\mathcal{ROUTE}(\mu, \delta_2) = mr, m \in \mathcal{NAT}, r \in \mathcal{NAT}^*$. Now examine Figure 3.

There are four cases to consider, based on the structure of the trees and the location of the matching communication events. The first case shows two trees, τ and μ that each have a sequence of events leaving the root node. As shown the result is a tree with a sequence of events leaving the root node, labelled by γ, followed by a subtree that is the combination of τ_1 and μ_1. Case 2 in Figure 3 combines one tree that consists of two branches with one that has a single path from the root. The result is one of two cases: (a) If the matching communication event lies down the leftmost branch then the leftmost branch is the combination of τ's leftmost branch with μ, the rightmost branch is merely copied into the new tree; (b) If the matching communication event lies down the rightmost branch, then the rightmost branch of τ is combined with μ and the leftmost branch is copied into the new tree. The third case is similar to the second except that τ is the tree with the single path from the root and μ is the tree with two branches. Finally, the fourth case occurs when both trees have branches from the root. The



if $\alpha = a[t_1], \beta = b[t_2]$ then $\gamma = (a[t_1], b[t_2])$

$t_{1,2} = MAX(t_1, t_2)$

if $\alpha = a[\overrightarrow{t_1}], \beta = a[\overleftarrow{t_2}]$ then

then $\gamma = a[\overleftrightarrow{t_{1,2}}]$

Figure 3

Combining Trees With Matching Events: Case 1 result is one of four cases all of which consist of trees with three branches, one branch from each of the original trees that remains unchanged, and one branch that is the combination of a path from each of the original trees.

As trees are combined, two kinds of events appear that are not present in an initial set of trees, synchronized communication events and tuples of events. Synchronized communication events have already been defined, tuples of events appear now for the first time. Some additional notation is needed for manipulating tuples of events.

Definition 2.8. *Let $\alpha = (a[t_1], b[t_2])$ be a tuple of two events where $\alpha.1 = a[t_1]$ and $\alpha.2 = b[t_2]$.*

In addition, event labels must be merged to form one new event label. Event labels are only merged when two trees containing matching communication events are combined into one tree. Labels are merged according to the following rules.

Definition 2.9. *Let α and β be two event labels and let δ be the event in the matching communication events pair possessed by the passive tree. Furthermore, let $time(\alpha) = t_1$ and $time(\beta) = t_2$.*

$$[\alpha, \beta] = \begin{cases} (\alpha, \beta), & if \quad \alpha \overset{mce}{\neq} \beta \quad and \quad type(\delta) = recv, \\ (\alpha, \beta.1), & if \quad \alpha \overset{mce}{\neq} \beta \quad and \quad type(\delta) = sync, \\ \gamma, & if \quad \alpha \overset{mce}{=} \beta \end{cases}$$

where

$$name(\gamma) = name(\alpha)$$

$$type(\gamma) = sync$$

$$time(\gamma) = t_{1,2} = max(t_1, t_2)$$

There are three ways that event labels get merged. If the two labels being merged represent a pair of matching communication events, then they form one synchronized event. If the two labels being merged do not represent a pair of matching communication events, then they will form a tuple of events. The portions of the tuple come from different places depending on whether the passive tree contains a *recv* or a *sync* event type on its portion of the matching communication events pair. Figure 3 shows how trees with matching communication events are combined when the passive tree contains a *recv* matching communication event type. Figure 4 shows what happens when the matching communication event contained by the passive tree is a *sync* event type.

Again there are four cases. The difference is that in all cases the passive tree is copied into the new tree, positioned at the root. The remaining part of the new tree is formed almost exactly as in the previous case where the passive tree contained a *recv* event type. The only difference is when tuples of events are formed to label the new arcs, the second portion of the tuple comes from the second portion of the tuple in the passive tree. In the previous case, the whole arc label was used rather than just a portion.

A definition is now provided for the combine operation. The definition formally states the rules for combining trees that have matching communication events, which was given pictorially in Figures 3 and 4.

Definition 2.10 Let τ, $\mu \in \mathcal{EDT} \ni \mathcal{COMM}(\tau) \ominus \mathcal{COMM}(\mu)$. *Then* $\tau \star_{mce} \mu = \sum_{i=1}^{n} \alpha_i \tau_i \star_{mce} \sum_{j=1}^{m} \beta_j \mu_j$ *equals*

i) $[\alpha_1 \beta_1](\tau_1 \star_{mce} \mu_1)$, *if* $n = m = 1$,

ii) $\sum_{k=1}^{n} \gamma_k \nu_k$, *if* $n > 1$, $m = 1$, $\mathcal{ROUTE}(\tau, \delta_1) = ls$, $l \in \mathcal{NAT}, s \in \mathcal{NAT}^*$,

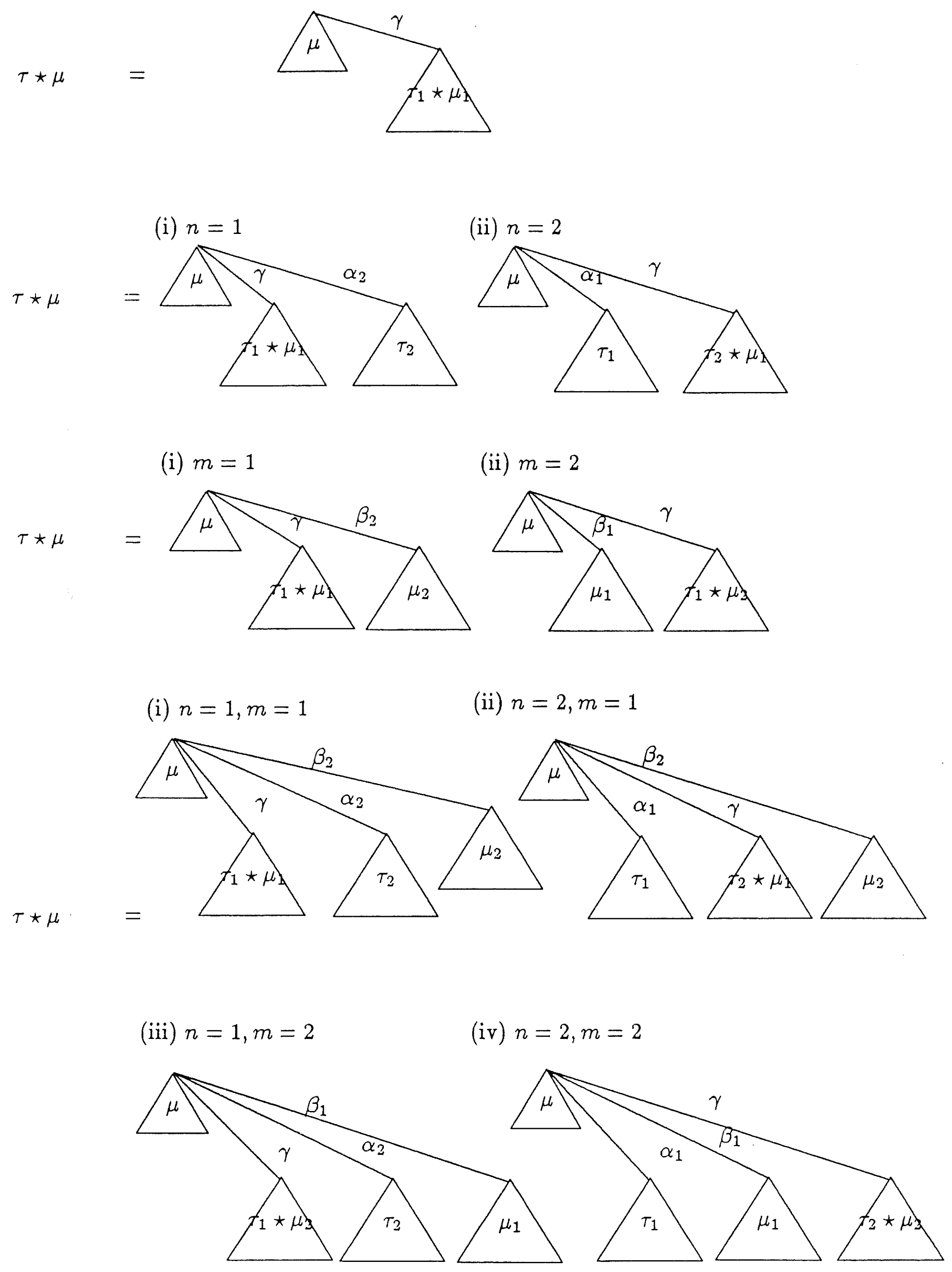
$$\forall k \neq l, \gamma_k \nu_k = \alpha_k \tau_k,$$

$$\forall k = l, \gamma_k = [\alpha_k \beta_k], \nu_k = \tau_k \star_{mce} \mu_1,$$

iii) $\sum_{k=1}^{m} \gamma_k \nu_k$, *if* $n = 1$, $m > 1$, $\mathcal{ROUTE}(\mu, \delta_2) = ls$, $l \in \mathcal{NAT}, s \in \mathcal{NAT}^*$,

$$\forall k \neq l, \gamma_k \nu_k = \beta_k \mu_k,$$

$$\forall k = l, \gamma_k = [\alpha_1 \beta_k], \nu_k = \tau_1 \star_{mce} \mu_k,$$



if $\alpha = a[t_1], \beta = (b_1[t_2], b_2[t_3])$
then $\gamma = (a[t_1], b_2[t_3])$

if $\alpha = a[\overrightarrow{t_1}], \beta = a[\overleftarrow{t_2}]$ then

then $\gamma = a[\overleftrightarrow{t_{1,2}}]$

$t_{1,2} = MAX(t_1, t_2)$

Figure 4

Combining Trees With Matching Communication Events: Case 2

iv) $\sum_{k=1}^{n+m-1} \gamma_k \nu_k$, *if* $n > 1$, $m > 1$, $\mathcal{ROUTE}(\tau, \delta_1) = ls$, $\mathcal{ROUTE}(\mu, \delta_2) = qx, l, q \in \mathcal{NAT}, s, x \in \mathcal{NAT}^*$,

$$\forall k = 1, \ldots, (l-1), \gamma_k \nu_k = \alpha_k \tau_k,$$

$$\forall k = l, \ldots, (l+q-1), \gamma_k \nu_k = \beta_j \mu_j, j = 1, \ldots, (q-1),$$

$$\forall k = l+q, \gamma_k \nu_k = [\alpha_l \beta_q] \tau_l \star_{mce} \mu_q,$$

$$\forall k = (l+q+1), \ldots, n, \gamma_k \nu_k = \alpha_k \tau_k,$$

$$\forall k = (l+q+l+n), \ldots, (n+m-1), \gamma_k \nu_k = \beta_j \mu_j, j = (q+1), \ldots, m.$$

Frequently, two trees will not contain matching communication events. The next definition indicates the result of combining such trees and all other cases that arise. Any tree that is combined with the null tree simply gives back the original tree. The combine operation is idempotent, that is, the result of combining any tree with itself is that tree ($\tau \star \tau = \tau$). Finally, combining two different trees that do not contain matching communication events produces a forest of two trees.

Definition 2.11. *Let $\tau, \mu \in \mathcal{EDT}$ and let τ_0 be the null tree.*

$$\tau \star \mu = \begin{cases} \tau, & \mathit{if} \quad \mu = \tau_0, \\ \mu, & \mathit{if} \quad \tau = \tau_0, \\ \tau, & \mathit{if} \quad \tau = \mu, \\ \tau \star_{mce} \mu, & \mathit{if} \quad \tau \neq \mu \neq \tau_0, \quad \mathit{and} \quad \mathcal{COMM}(\tau) \circeq \mathcal{COMM}(\mu) \\ \langle \tau, \mu \rangle, & \mathit{if} \quad \tau \neq \mu \neq \tau_0, \quad \mathit{and} \quad \mathcal{COMM}(\tau_i) \not\circeq \mathcal{COMM}(\tau_j) \end{cases}$$

Trees of the form τ will be called basic trees, and trees of the form $\langle \tau, \mu \rangle$ will be called pseudo trees. Some more notational conventions are followed. If there is a set of many trees that need to be combined then each tree is denoted by $\tau_i, i \in \mathcal{NAT}$, rather than by a separate greek variable (μ, ν). As trees are combined the new trees are denoted as $\tau_{1,2}$ if $\tau_1 \star_{mce} \mu$.

Two definitions of combine operations already exist but more are needed. The $\star$ operation introduced a new type of tree, the pseudo tree. None of the existing definitions indicate how to combine trees if one or more of the trees are pseudo trees. The following two definitions show how to combine pseudo trees with basic trees, the $\star\star$ (doublestar) operation. A triplestar, $\star\star\star$, operation combines pseudo trees with pseudo trees. And finally, the circlestar operation, $\circledast$, is defined to operate between any two types of trees, whether they are basic or pseudo trees.

Definition 2.12. *Let τ_0 be the null tree, $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$, a set of basic trees, and $\mathcal{P} = \{\rho_1, \ldots, \rho_m\}$, a set of pseudo trees. $\forall \rho_i = \langle \tau_{i_1}, \tau_{i_2} \rangle \in \mathcal{P}$ and $\tau_j \in \mathcal{T}$,*

$$\rho_i \star\star \tau_j = \begin{cases} \langle \tau_{i_1} \star \tau_j, \tau_{i_2} \rangle, & if \quad j = 0 \quad or \quad \mathcal{COMM}(\tau_{i_1}) \ominus \mathcal{COMM}(\tau_j) \quad and \\ & \quad \mathcal{COMM}(\tau_{i_2}) \not\ominus \mathcal{COMM}(\tau_j) \\ \langle \tau_{i_1}, \tau_{i_2} \star \tau_j \rangle, & if \quad j = 0 \quad or \quad \mathcal{COMM}(\tau_{i_1}) \not\ominus \mathcal{COMM}(\tau_j) \quad and \\ & \quad \mathcal{COMM}(\tau_{i_2}) \ominus \mathcal{COMM}(\tau_j) \\ (\tau_{i_1} \star \tau_j) \star \tau_{i_2}, & if \quad j \neq 0, \quad \mathcal{COMM}(\tau_{i_1}) \ominus \mathcal{COMM}(\tau_j) \quad and \\ & \quad \mathcal{COMM}(\tau_{i_2}) \ominus \mathcal{COMM}(\tau_j) \\ \langle \tau_{i_1}, \tau_{i_2}, \tau_j \rangle, & if \quad j \neq 0, \quad \mathcal{COMM}(\tau_{i_1}) \not\ominus \mathcal{COMM}(\tau_j) \quad and \\ & \quad \mathcal{COMM}(\tau_{i_2}) \not\ominus \mathcal{COMM}(\tau_j) \end{cases}$$

Definition 2.12 defines the combination of a pseudo tree with a basic tree. The next definition, 2.13 is very similar to 2.12 except that the order of the trees is reversed, a basic tree is combined with a pseudo tree. Definition 2.13 is necessary since combining trees (or processes) should be commutative [HOAR85] but the property cannot be derived from previous definitions.

Definition 2.13. *Let τ_0 be the null tree, $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$, a set of basic trees, and $\mathcal{P} = \{\rho_1, \ldots, \rho_m\}$, a set of pseudo trees. $\forall \rho_i = \langle \tau_{i_1}, \tau_{i_2} \rangle \in \mathcal{P}$ and $\tau_j \in \mathcal{T}$,*

$$\tau_j{\star\star}\rho_i = \begin{cases} \langle \tau_j \star \tau_{i_1}, \tau_{i_2} \rangle, & if \quad j = 0 \quad or \quad \mathcal{COMM}(\tau_{i_1}) \text{⊜} \mathcal{COMM}(\tau_j) \quad and \\ & \quad \mathcal{COMM}(\tau_{i_2}) \not\text{⊜} \mathcal{COMM}(\tau_j) \\ \langle \tau_{i_1}, \tau_j \star \tau_{i_2} \rangle, & if \quad j = 0 \quad or \quad \mathcal{COMM}(\tau_{i_1}) \not\text{⊜} \mathcal{COMM}(\tau_j) \quad and \\ & \quad \mathcal{COMM}(\tau_{i_2}) \text{⊜} \mathcal{COMM}(\tau_j) \\ (\tau_{i_1} \star \tau_j) \star \tau_{i_2}, & if \quad j \neq 0, \quad \mathcal{COMM}(\tau_{i_1}) \text{⊜} \mathcal{COMM}(\tau_j) \quad and \\ & \quad \mathcal{COMM}(\tau_{i_2}) \text{⊜} \mathcal{COMM}(\tau_j) \\ \langle \tau_j, \tau_{i_1}, \tau_{i_2} \rangle, & if \quad j \neq 0, \quad \mathcal{COMM}(\tau_{i_1}) \not\text{⊜} \mathcal{COMM}(\tau_j) \quad and \\ & \quad \mathcal{COMM}(\tau_{i_2}) \not\text{⊜} \mathcal{COMM}(\tau_j) \end{cases}$$

The combination of two pseudo trees is now defined. Two pseudo trees are combined by taking each component of the second pseudo tree one at a time and combining it with the first pseudo tree. A different operation, $\star$ rather than $\star\star$, is required depending on the result of combining the first component with the first pseudo tree. If the first component of the second pseudo tree has matching communication events, with all the components of the first pseudo tree, then the result will be a basic tree. Since the second component of the second pseudo tree is also a basic tree the $\star$ must be used. If they don't all have matching communication events, then the result will be a pseudo tree and the $\star\star$ operation will be used to combine the intermediate result with the second component of the first pseudo tree.

Definition 2.14. *Let τ_0 be the null tree, $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$, a set of basic trees, and $\mathcal{P} = \{\rho_1, \ldots, \rho_m\}$, a set of pseudo trees. $\forall \rho_i = \langle \tau_{i_1}, \tau_{i_2} \rangle, \rho_j = \langle \tau_{j_1}, \tau_{j_2} \rangle \in \mathcal{P}$,*

$$\rho_i{\star\star\star}\rho_j = \begin{cases} (\rho_i{\star\star}\tau_{j_1}) \star \tau_{j_2}, & if \quad \mathcal{COMM}(\tau_{i_1}) \text{⊜} \mathcal{COMM}(\tau_{j_1}) \quad and \\ & \mathcal{COMM}(\tau_{i_2}) \text{⊜} \mathcal{COMM}(\tau_{j_1}) \\ (\rho_i{\star\star}\tau_{j_1}){\star\star}\tau_{j_2}, & otherwise \end{cases}$$

Finally, the $\circledast$ operation defines for any two trees, basic or pseudo, how to combine them. Note that $\mathcal{EDT}$ is defined differently in Definition 2.15 from the previous definitions (where it denoted only a set of basic trees).

Definition 2.15. *Let* τ_0 *be the null tree,* $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$, *a set of basic trees, and* $\mathcal{P} = \{\rho_1, \ldots, \rho_m\}$, *a set of pseudo trees. Let* $\mathcal{EDT} = \mathcal{T} \bigcup \mathcal{P}$ *and let* $\tau_i, \tau_j \in \mathcal{EDT}$.

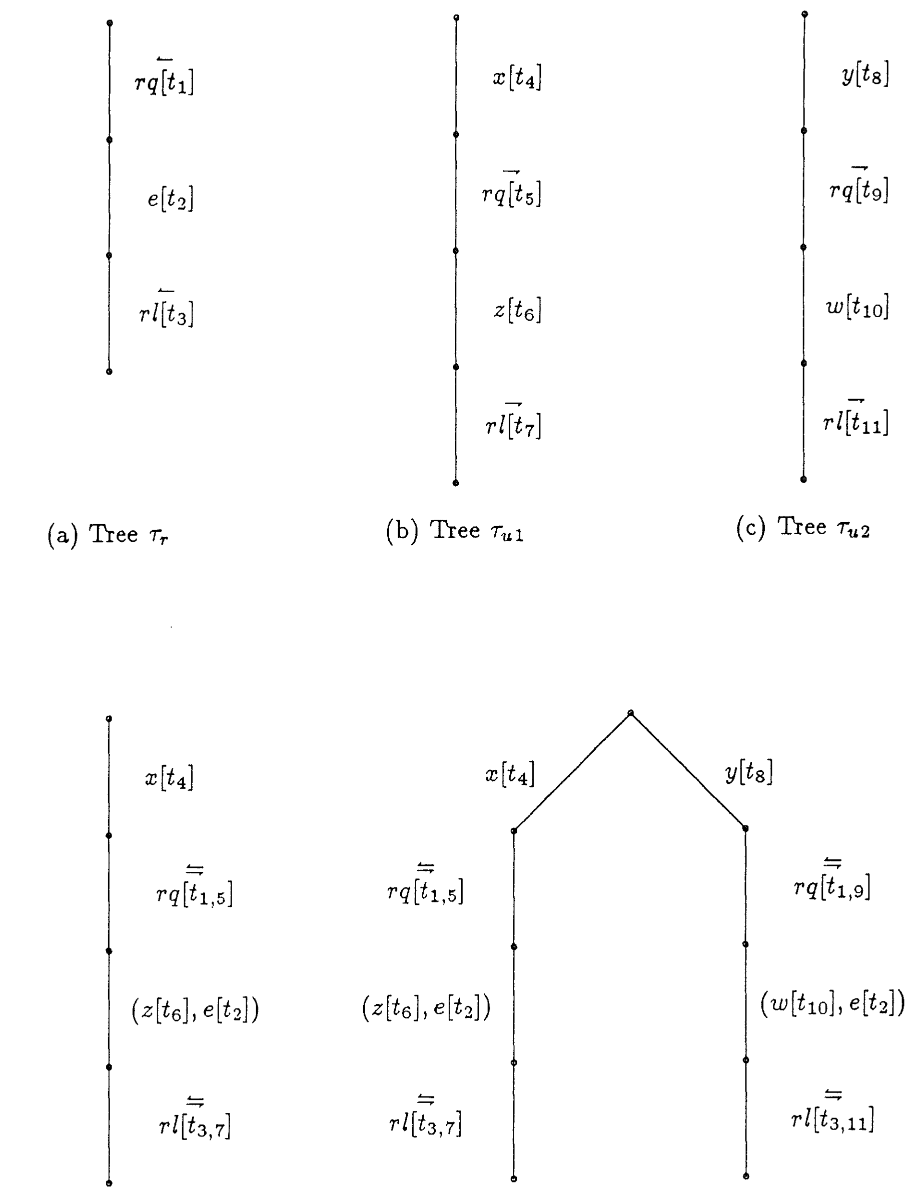
$$\tau_i \circledast \tau_j = \begin{cases} \tau_i \star \tau_j & if \quad \tau_i, \tau_j \in \mathcal{T} \\ \tau_i {\star\star} \tau_j & if \quad \tau_i \in \mathcal{T} \quad and \quad \tau_j \in \mathcal{P} \quad or, \\ & \qquad \tau_i \in \mathcal{P} \quad and \quad \tau_j \in \mathcal{T} \\ \tau_i {\star\star\star} \tau_j & if \quad \tau_i, \tau_j \in \mathcal{P} \end{cases}$$

An example is presented that demonstrates the operations defined above. The two types of trees (basic and pseudo) and the multiple combine operations are necessary for the $\circledast$ operation to be associative.

Resource Manager

A simple resource manager and two user processes comprising a program with three concurrent processes will be used to demonstrate how the tree combination operation represents event conflicts. The resource manager has three events: (1) receive a request for the resource, (2) grant the resource, and (3) receive notification to release the resource. Each of the user processes has four events: (1) perform some calculation, (2) request the resource, (3) use the resource, and (4) release the resource.

The tree in Figure 5 part (a) represents the resource manager process, and parts (b) and (c) the two user processes. As shown in (d) $\tau_r \circledast \tau_{u1}$ is a tree with a single path. It contains two synchronized communication events, one execution event, $x[t_4]$, and a tuple of two execution events. The tuple in some sense indicates that the events are or could be concurrent. The complete program of three processes is represented by the tree in (e). The second user process interacts with


$\overleftarrow{rq}[t_1]$
$e[t_2]$
$\overleftarrow{rl}[t_3]$
$x[t_4]$
$\overrightarrow{rq}[t_5]$
$z[t_6]$
$\overrightarrow{rl}[t_7]$
$y[t_8]$
$\overrightarrow{rq}[t_9]$
$w[t_{10}]$
$\overrightarrow{rl}[t_{11}]$
$x[t_4]$
$\overleftrightarrows{rq}[t_{1,5}]$
$(z[t_6], e[t_2])$
$\overleftrightarrows{rl}[t_{3,7}]$
$x[t_4]$
$y[t_8]$
$\overleftrightarrows{rq}[t_{1,5}]$
$(z[t_6], e[t_2])$
$\overleftrightarrows{rl}[t_{3,7}]$
$\overleftrightarrows{rq}[t_{1,9}]$
$(w[t_{10}], e[t_2])$
$\overleftrightarrows{rl}[t_{3,11}]$

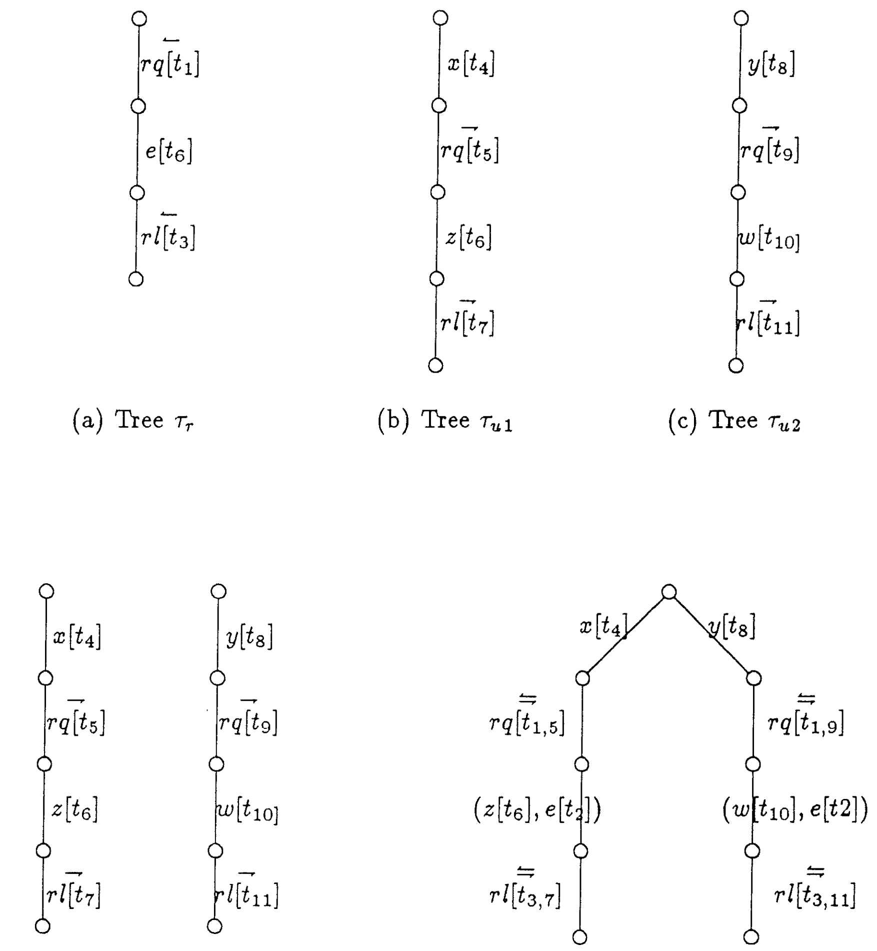
(a) Tree τ_r (b) Tree τ_{u1} (c) Tree τ_{u2}

(d) $\tau_{r,u1} = \tau_r \circledast \tau_{u1}$ (e) $\tau_{r,u1,u2} = (\tau_r \circledast \tau_{u1}) \circledast \tau_{u2}$

Figure 5

A Simple Resource Manager

the resource manager in much the same way as the first user process. The tree $(\tau_r \circledast \tau_{u1}) \circledast \tau_{u2}$ has two branches. The branch taken depends on which of the two events, x or y, is quicker.



$\overleftarrow{rq}[t_1]$
$e[t_6]$
$\overleftarrow{rl}[t_3]$
$x[t_4]$
$\overrightarrow{rq}[t_5]$
$z[t_6]$
$\overrightarrow{rl}[t_7]$
$y[t_8]$
$\overrightarrow{rq}[t_9]$
$w[t_{10}]$
$\overrightarrow{rl}[t_{11}]$
$x[t_4]$
$\overrightarrow{rq}[t_5]$
$z[t_6]$
$\overrightarrow{rl}[t_7]$
$y[t_8]$
$\overrightarrow{rq}[t_9]$
$w[t_{10}]$
$\overrightarrow{rl}[t_{11}]$
$x[t_4]$
$y[t_8]$
$\overleftrightarrows{rq}[t_{1,5}]$
$\overleftrightarrows{rq}[t_{1,9}]$
$(z[t_6], e[t_2])$
$(w[t_{10}], e[t2])$
$\overleftrightarrows{rl}[t_{3,7}]$
$\overleftrightarrows{rl}[t_{3,11}]$

(a) Tree τ_r (b) Tree τ_{u1} (c) Tree τ_{u2}

(d) Tree $< \tau_{u1}, \tau_{u2} >$ (e) Tree $\tau_{u1,u2,r}$

Figure 6

Combining Trees in Different Orders

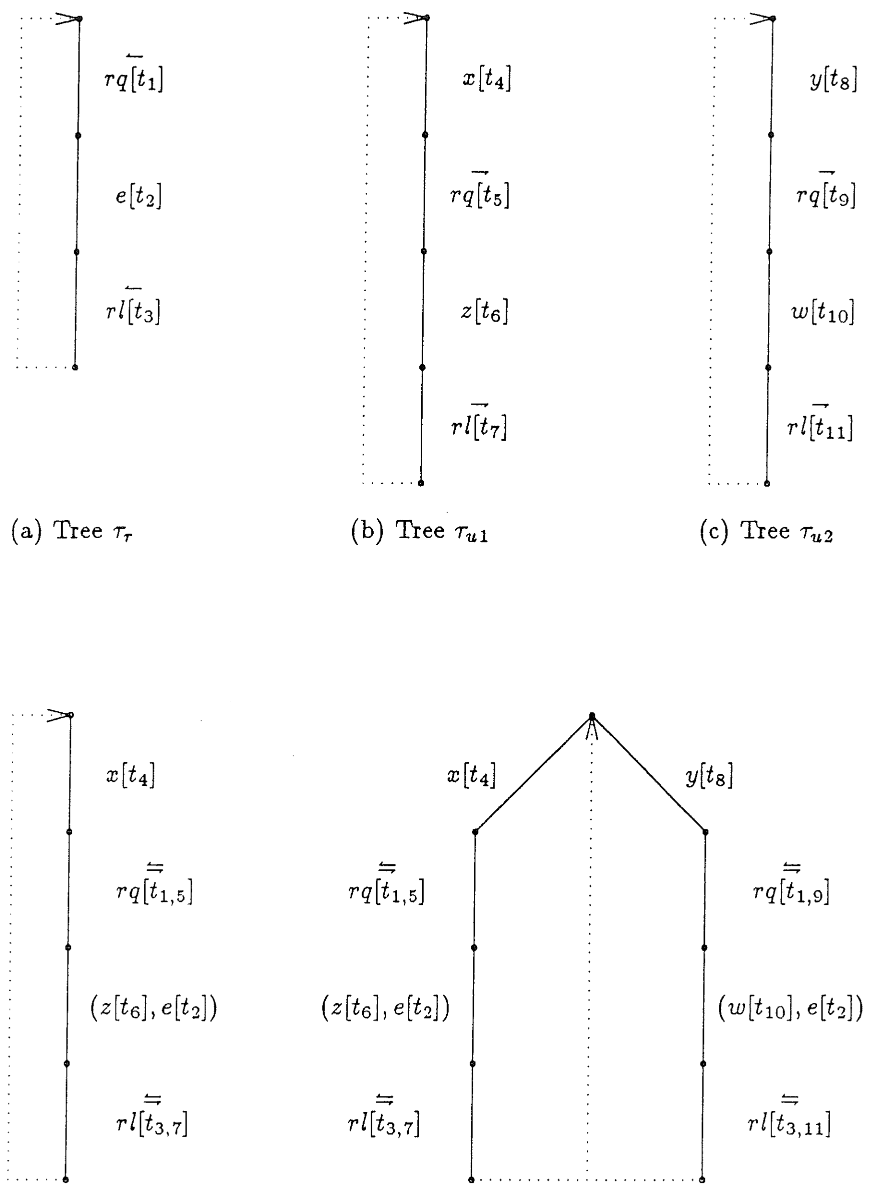
Now consider combining the trees in a different order, $(\tau_{u1} \circledast \tau_{u2}) \circledast \tau_r$.

$$\tau_{u1} \circledast \tau_{u2} = \langle \tau_{u1}, \tau_{u2} \rangle$$

$$\begin{aligned} \langle \tau_{u1}, \tau_{u2} \rangle \circledast \tau_r &= (\tau_{u1} \star \tau_r) \star \tau_{u2} \\ &= \tau_{u1,r} \star \tau_{u2} \\ &= \tau_{u1,r,u2} \end{aligned}$$

Note that $\tau_{u1,r,u2} = \tau_{r,u1,u2}$. See Figure 6.

In the previous example, the EDTs did not represent infinite processes. The EDT that represents an infinite process will be infinite. An infinite tree can be



$rq[\overleftarrow{t_1}]$
$e[t_2]$
$rl[\overleftarrow{t_3}]$
$x[t_4]$
$rq[\overrightarrow{t_5}]$
$z[t_6]$
$rl[\overrightarrow{t_7}]$
$y[t_8]$
$rq[\overrightarrow{t_9}]$
$w[t_{10}]$
$rl[\overrightarrow{t_{11}}]$
$x[t_4]$
$rq[\overleftrightarrows{t_{1,5}}]$
$(z[t_6], e[t_2])$
$rl[\overleftrightarrows{t_{3,7}}]$
$x[t_4]$
$y[t_8]$
$rq[\overleftrightarrows{t_{1,5}}]$
$(z[t_6], e[t_2])$
$rl[\overleftrightarrows{t_{3,7}}]$
$rq[\overleftrightarrows{t_{1,9}}]$
$(w[t_{10}], e[t_2])$
$rl[\overleftrightarrows{t_{3,11}}]$

(a) Tree τ_r (b) Tree τ_{u1} (c) Tree τ_{u2}

(d) $\tau_{r,u1} = \tau_r \circledast \tau_{u1}$ (e) $\tau_{r,u1,u2} = (\tau_r \circledast \tau_{u1}) \circledast \tau_{u2}$

Figure 7

Representing Infinite Processes

represented in two ways: (1) replicating the events that occur over and over again using "...", or (2) indicating which event occurs next by connecting two nodes with a dotted arc. In Figure 7 the resource manager is represented as the combination of three infinite processes.

Summary

This paper defines a new representation of communicating systems called Event Dependency Trees (EDT). In EDT processes are represented as trees where the nodes of a tree represent system states and the arcs represent the execution of system events. An event is one of three types: (1) execution: represents the execution of a sequential piece of code (with no communication constructs), (2) communication: represents the execution of a message passing construct, or (3) the null event. Communication events are further subdivided into send, receive, and synchronized communication events. In addition, each event has an associated time delay, represented by some variable such as t.

EDT is a formal model of distributed or communicating systems that predicts how CSP–type processes will interact. Although it appears that EDT is a model of software, assumptions about how the system impacts the execution of the software is a crucial aspect of the model, the primary assumption being that events take time that could differ from execution to execution. From an EDT model of software one can identify each execution path by its unique event ordering. This provides some insight as to how one might reason about whether certain events and ultimately execution paths can occur. The model supplies potentially important information for the design and construction of concurrent software systems.

References

[Broo84] Brookes S.D., Hoare C.A.R., and Roscoe A.W. A Theory of Communicating Sequential Processes. *Journal of the ACM 31*, 3 (July, 1984), 560–599.

[Henn85b] Hennessy M. Acceptance Trees. *CACM 32*, 4 (October, 1985), 896–928.

[Hoar85] Hoare C.A.R. *Communicating Sequential Processes*, Prentice–Hall, 1985.

[Lane87] Lane, D.S. Representing Communicating Software to Derive System Behavior and Deadlock-Free Software. Technical Report No. 87-27. University of California, Irvine (October, 1987).

[Miln80] Milner R. *A Calculus of Communicating Systems*, Goos G., and Hartmanis J., Ed., Springer–Verlag, Berlin, 1980.

[Wins84] Winskel G. Synchronization Trees. *Theoretical Computer Science 34* (1984), 33–82.