

## What Are Documents?

The concept of document is all pervasive. It occurs in as widely different contexts as scientific, technical, business or tourism papers or reports, as sales offers, marketing brochures and personal letters, as inter- and intra-departmental memos in public administration, as legal entities such as law texts, juris prudence and law court verdicts, as e-mails and web pages, as credit card payment slips, as music and movie recordings (subject, possibly, to DRM (digital rights management), etcetera. A patient medical report is a dossier of documents: anamnese, analysis reports, diagnostics and treatment plans. In the Danish Board of National Health report on patient medical reports (as a basis for electronic patient journals) the term document seems to take on a bewildering and not insignificantly large set of different meanings.

### 1.1 Varieties of Documents

The above enumeration of examples of documents shall serve as an admittedly rather loose delineation of what we mean by a document: something visual or audial or audio-visual (intellectual, artistic, legal, technical, scientific) that can be created, can be read (i.e., rendered; listened to or viewed), can (in some cases) be edited, can be copied, can be moved from place to place, and can be destroyed (shredded, deleted).

One of the fundamental ideas of computer science is that “data”, like documents, can best be understood in terms of the operations on the data, i.e., the documents. So that is what we shall do. Not the operations on personal letters, or a poem, or a collection of poems; not the operations on a credit slip, or on a CD ROM of music, or a DVD recorded movie; but those operations that seem to be generic to all documents: create, render, edit, copy, move and destroy.

## 1.2 On the Domain of Documents

When, as we shall now do in this book, and as from the next chapter, namely describe a domain of documents, then the reader will, at time, perhaps object, saying “*but that’s not feasible in the domain!*”.

We shall, of course, at almost “every turn of the road”, defend our position: namely that if you can think it in the domain, then it must be described.

Thus a domain description describes **what there is**. Not how you would like it to be (that may imply either some business process re-engineering or requirements for software). In describing **what there is** we have to accept that we must describe aspects of the domain that we may not like, or that we may not think possible.

Let us taken some examples — some claims about what it is possible to describe — in our domain: We claim, as you shall soon see, that from every document,  $d$ , we can observe its entire past history: If it is an unedited copy of a prior document,  $d_p$ , then we can, from that copy,  $d$ , observe the document,  $d_p$ , from which it was copied. If  $d$  is an edited version of a document,  $d_p$ , then we claim that we can observe the un-edited version of  $d$ , i.e.,  $d_p$ . We claim, as we shall later substantiate, that we can observe the time and location of when and where operations were performed on documents.

Now that’s a “pretty tall story!” you should say. Can one really, from an edited document “see” its un-edited form. Yes we claim. Suppose it was hand-written. Then also editing was hand-written. And we claim that it is possible to talk about which of the hand-writing was there first, which was added as editing changes. Can one really from a copy “see” the document from which it was copied. Yes we claim. Certainly, if  $d_c$  is a copy of  $d$ , then whatever information  $d$  had  $d_c$  must have. In addition it must be possible to claim that  $d_c$  is not the master (i.e., the basis for the copy), but that  $d$  is. There are some subtle distinctions here. The main gist of these distinctions is that *it is in principle possible* to make these claims, i.e., to talk about these properties of documents, their copies, their edited versions, etc. Whether it is in practice always possible to identify these matters is another issue.

Now, whether what we claim can be observed are really the actual things, or surrogates thereof, is another matter which we shall, at the moment not touch upon. And: whether we get toe correct time and location, when we, for example, say that we can observe time and location of when and where an operation was performed on a document, is another matter also: the domain is fallible. We may get a wrong answer, but an answer we get! Similarly, if we claim to be able to observe the predecessor of a copy, i.e., the master from from which the copy was made, than what we may see may not be that master, but a corrupted version of it. So be it. We see something claimed to be “such and such”!

We shall elaborate further on the above when we, in Chap. 2, go through various aspects of documents.

## 1.3 Semiotics of Documents

Semiotics, as used by us, is the study and knowledge of pragmatics, semantics and syntax of language(s).

### 1.3.1 Pragmatics of Documents

Pragmatics is the study of, the knowledge about and the and practice of the factors that govern our choice of language in social interaction and the effects of our choice on others.

Pragmatics determine, it appears, when we decided to create, render, edit, copy, move and destroy documents.

We shall not be bothered by formalising pragmatic issues of documents.

### 1.3.2 Semantics of a Document Concept

Semantics is the study and knowledge, incl. specification of meaning in language.

Semantics determine, it appears, how we edit documents. In this book we shall not be concerned about the contents of document, i.e., the information to be edited.

But we shall be very much interested in the semantics of the interrelated sets of operations performed on documents. We claim that the semantics of these interrelated sets of operations is an integral part of the semantics of documents. We leave it to other books to tackle the semantics of document edits.

With computing and communication now supporting the creation, editing, sending/receiving (including copying) and destruction of documents there are “rich”, but dangerous grounds for misinterpretations as to what constitutes a document.

### 1.3.3 Syntax of Documents

By syntax we mean the ways in which words are arranged to show meaning (cf. semantics) within and between sentences, and the rules for forming syntactically correct sentences.

Since we are not, in this book, interested in the syntax of document information — that part of a document which is of interest to the casual user of documents — we shall not be dealing with syntaxes for document contents. But we shall, over the next many pages, be building up an implicitly expressed abstract syntax for a number of document attributes. These attributes have to do with such things as: when and where (time and location) was an operation performed on the document, was the document copied, the pre-history of the document — since creation, etc. Together this ensemble of attributes imply an abstract syntax of document, not of its contents, but of its management.

## 1.4 Structure of This Report

Chapter 2 presents a conventional model of documents such as we have tried to encircle that phenomenon (“documents”) above. That is, we shall model the create, copy, edit and move operations.

Chapter 3 presents a model of documents that change locations over time: dynamics and mobility. Doing so raises some technical issues of — it is believed — any formal specification language.

Chapter 4 tries to tackle some of these issues. It does so by allowing us to “extend” any formal specification with a mathematical first order theory. That is, a theory in which we can set aside a number of variables whose types are dense point sets (such as time and locations) and where we are then allowed to assume continuity and express first order differential equations.

Chapter 5 along another line of exploration, introduces the notions of authorization and partial view of documents. A partial view is a rendering function: applied to a pair, an authorization and a document, it yields those parts of a document contents for which the agent (issuing the rendering function) has rendering rights. This part may be all, no, or proper subparts.

Chapter 6 along a related line of exploration, introduces the notion of document license. A license is also a rendering function: applied to a pair, a license and a document, it yields those parts of a document contents for which the agent (issuing the rendering function) has rendering rights — together, possibly, with an updated document (one that perhaps changes future license-based rendering rights on that document).

## 1.5 On Reading This Report

In the first releases of this document the formalisations are expressed in terms of the RAISE specification language RSL.

That language, besides [19, 21], is also introduced in [8–10]. Those three volumes, besides, gives an extensive introduction to abstraction and modeling.

For readers not familiar with the tradition of formal specification languages whose origins goes back to John McCarthy [28–31] and Peter Landin [23–26], and which took its first form in the VDM meta (or specification) language [5, 11], now ISO standardised into VDM-SL [2, 3, 27, 34] [and of which RSL [19] is a derivate], we provide annotations that explain the notation. As the book “wears on” these annotations “peters off”.

---

## A Simplistic Model of Documents

### 2.1 First Model of Document Intrinsic

#### 2.1.1 Originals, Copies and Versions

There are documents. Documents are either created, edited or copied. One can claim that a document is either an original, or an edited version, for short, a version, of a document, or a copy of a document. One can claim that a document can either only (say: “most recently”) be an original, or only (say: “most recently”) be an edited document (i.e., a version), or only (say: “most recently”) be a copy of a document. The pragmatic intention of documents is to embody document content. We leave the notion of document content undefined. There is information. Information is either document content, or the absence of such. We use the special literal `void` to designate absence of content. To create a document needs no document content. From a document one can observe its most recent information.

#### type

D, oD, eD, cD, C, E  
 I == void | C

#### value

create: **Unit** → oD  
 edit: E × D → eD  
 copy: D → cD

is\_oD: D → **Bool**

is\_eD: D → **Bool**

is\_cD: D → **Bool**

#### axiom

∀ d:D •  
 is\_oD(d) ∨ is\_eD(d) ∨ is\_cD(d) ∧  
 is\_oD(d) ⇒ ¬is\_eD(d) ∧ ¬is\_cD(d) ∧

$$\begin{aligned} & \text{is\_eD}(d) \Rightarrow \sim \text{is\_oD}(d) \wedge \sim \text{is\_cD}(d) \wedge \\ & \text{is\_cD}(d) \Rightarrow \sim \text{is\_oD}(d) \wedge \sim \text{is\_eD}(d) \wedge \dots \text{ or, which is the same:} \\ \forall d:D, e:E \bullet \\ & \text{is\_oD}(\text{create}()) \wedge \sim \text{is\_eD}(\text{create}()) \wedge \sim \text{is\_cD}(\text{create}()) \wedge \\ & \text{is\_cD}(\text{copy}(d)) \wedge \sim \text{is\_oD}(\text{copy}(d)) \wedge \sim \text{is\_cD}(\text{edit}(e,d)) \\ & \text{is\_eD}(\text{edit}(e,d)) \wedge \sim \text{is\_oD}(\text{edit}(e,d)) \wedge \sim \text{is\_cD}(\text{edit}(e,d)) \end{aligned}$$
**value**

$$\text{obs\_I}: D \rightarrow I$$
**axiom**

$$\begin{aligned} & \text{obs\_I}(\text{create}()) = \text{void} \wedge \\ & \forall d:D \bullet \text{is\_oD}(d) \Rightarrow \text{obs\_I}(\text{copy}(d)) = \text{void} \end{aligned}$$
**Annotations:**

- The sectioning literal **type** designates that the following text (up to a next sectioning literal) introduces abstract and concrete type definitions. An abstract type definition is like a sort.
  - ★  $D$ ,  $\text{oD}$ ,  $\text{eD}$ ,  $\text{cD}$ ,  $C$  and  $E$  introduces the sorts of documents, original documents, edited documents, document copies, document contents and document editing. (We shall not elaborate further on  $E$  till Sect. 2.1.2 on the facing page.)
  - ★ The equation  $I == \text{void} \mid C$  defines document information as either being  $\text{void}$  or  $C$ . (We are not here telling you what  $\text{void}$  means.) The alternatives of  $U == V \mid W \mid X \mid Y \dots$  are, by the  $==$  constructor, being defined as disjoint types.
- The sectioning literal **value** designates that the following text (up to a next sectioning literal) introduces values of defined types. Six such values are introduced. We see from their types ( $\dots \rightarrow \dots$ ) that they are all function values.
  - ★  $\text{create}$  designates the create function. It is a type  $\mathbf{Unit} \rightarrow \text{oD}$ . Thus it takes no arguments (designated by the value literal  $\mathbf{Unit}$ ) and yields an original document.
  - ★  $\text{edit}$  designates the editing function. It is a type  $E \times D \rightarrow \text{eD}$ . Thus it takes two arguments: some editing value and a document and yields an edited document.
  - ★  $\text{copy}$  designates the copy function. It is a type  $D \rightarrow \text{cD}$ . Thus it takes one argument, a document and yields a document: the copied document. The function signature says nothing about “what happened” to the input argument. As we shall see, it is still there, “somewhere”.<sup>1</sup>
  - ★  $\text{is\_oD}$  designates a predicate observer function. It is a type  $D \rightarrow \mathbf{Bool}$ . If the document is an original then truth is yielded, otherwise falsity.
  - ★  $\text{is\_eD}$  designates a predicate observer function. It is a type  $D \rightarrow \mathbf{Bool}$ . If the document is an edited version of a document then truth is yielded, otherwise falsity.

<sup>1</sup>Adding 3 and 7, yielding 10, does not, in any way, destroy or influence 3 and 7.

- ★ `is_cD` designates a predicate observer function. It is a type  $D \rightarrow \mathbf{Bool}$ .  
If the document is a copy then truth is yielded, otherwise falsity.  
The functions are all postulated. They are claimed to exist. They are not defined. Instead their properties will be revealed through axioms.
- The sectioning literal **axiom** designates that the following text (up to a next sectioning literal) introduces a number of properties — typically over the types and values introduced before these axioms.
  - ★ The clause  $\forall a:A \bullet$  “reads”: *for all values a of type A it is the case that*.  
In RSL all quantifications are typed.
  - ★ The proposition  $\text{is\_oD}(d) \vee \text{is\_eD}(d) \vee \text{is\_cD}(d) \wedge$  “reads” a document *d* is either an original or an edited version or a copy, and ... .
  - ★ The proposition  $\text{is\_oD}(d) \Rightarrow \sim \text{is\_eD}(d) \wedge \sim \text{is\_cD}(d)$  “reads” if a document is an original then it is neither an edited version or a copy, and ... .
  - ★ The proposition  $\text{is\_oD}(\text{create}()) \wedge \sim \text{is\_eD}(\text{create}()) \wedge \sim \text{is\_cD}(\text{create}())$  “reads” for all documents and editing values, the value resulting from a proper create operation is an original and is not a edited version and is not a copy.
  - ★ The predicate  $\forall d:D, e:E \bullet \text{is\_eD}(\text{edit}(e,d)) \wedge \sim \text{is\_oD}(\text{edit}(e,d))$  etcetera “reads” a document that has been properly edited is an edited version and is not an original and is not a copy.
- The signature  $\text{obs\_I}: D \rightarrow I$  expresses a an observer function which from a document observes its information.
- The axioms  $\text{obs\_I}(\text{create}()) = \text{void}$  and  $\forall d:D \bullet \text{is\_oD}(d) \Rightarrow \text{obs\_I}(\text{copy}(d)) = \text{void}$  expresses that copies of copies of ... of copies of originals still have no proper information content.

### 2.1.2 Editing and Versions

Editing a document modifies its information. An edited document is a version of the document from which it was edited. Editing a document does not amount to establishing a new document. From an edited document one can observe the information immediately before it was most recently edited, and how that information was edited, i.e., the resulting content. One way of modelling the edit function is by means of two functions: a forward editing function and a backward, “undo” editing function. The forward editing function takes an information argument and delivers an information result. The backward editing function takes an information argument and delivers an information results. The backward editing function is the inverse of the forward editing function.

**type**

$$E' = FE \times BE$$

$$FE, BE = I \rightarrow I$$

$$E = \{ \{(fe, be):E \bullet \forall i:I \bullet be(fe(i))=i \} \}$$

**value**

$$\text{obs\_E}: D \xrightarrow{\sim} E$$
**axiom**

$$\begin{aligned} \text{obs\_E}(\text{create}()) &= \mathbf{chaos} \wedge \\ \forall d:D, e, (fe, be):E & \\ \text{obs\_E}(\text{edit}(e, d)) &= e \wedge \\ \text{obs\_I}(\text{edit}((fe, be), d)) &= fe(\text{obs\_I}(d)) \wedge \\ \text{obs\_I}(d) &= be(\text{obs\_I}(\text{edit}((fe, be), d))) \wedge \\ \text{obs\_E}(\text{copy}(\text{edit}(e, d))) &= e \wedge \dots /* \text{induction} */ \end{aligned}$$
**Annotations:**

- $E'$  is a concrete type. It is defined as the Cartesian of two types: FE and BE.
- Both FE and BE are total functions from information to information.
- $E$  is the subtype of  $E'$  which constrains the backward editing function  $be:BE$  to be an inverse of the forward editing function  $fe:FE$ .
- $\text{obs\_E}$  is a partial observer function. It applies to documents.
- From an original document one cannot observe any editing functions:  $\text{obs\_E}(\text{create}()) = \mathbf{chaos}$ .
- From edited documents (whether since copied) one can (still) observe the editing functions.
- The parenthesised clauses: (whether since copied) and (still) are not expressed by  $\text{obs\_E}(\text{copy}(\text{edit}(e, d))) = e$ , but intimated by the ellipses clause  $\dots$  — to be formalised below.

**2.1.3 Document Traces**

From a document one can observe its immediate predecessor document. An original document has no predecessor. A copy,  $d_c$  of a document,  $d$ , had  $d$  as its immediate predecessor document. An edited document, also called a version,  $d_e$  of a document,  $d$ , had  $d$  as its immediate predecessor document. And so on, “ad finitum”, till the original document is encountered.

Let us call the document from which an edited version arises for the master document. And let us call document from which a copy is made also for the master document. Thus the predecessor documents are masters wrt. the successors.

**value**

$$\text{obs\_Pre}: D \xrightarrow{\sim} D$$
**axiom**

$$\begin{aligned} \text{obs\_Pre}(\text{create}()) &= \mathbf{chaos} \wedge \\ \forall d:D, e:E \cdot \text{obs\_Pre}(\text{copy}(d)) &= d = \text{obs\_Pre}(\text{edit}(e, d)) \end{aligned}$$
**Annotations:**

- From any document other than an original one can observe,  $\text{obs\_Pre}$ , its predecessor.

- Thus  $\text{obs\_Pre}(\text{create}())$  is not defined, that is, is **chaos**.
- For all documents and editing functions the predecessor of a copy of  $d$ , i.e.,  $\text{copy}(d)$ , is  $d$ , and the predecessor of the  $e$  edited version,  $\text{edit}(e,d)$  of  $d$  is also  $d$ .

### Observations:

- We could decide, instead of making  $\text{obs\_Pre}$  a partial function, to let  $\text{obs\_Pre}(\text{create}())$  yield  $\text{create}()$ .
- Then  $\text{obs\_Pre}$  would be a total function.
- And then  $\text{obs\_Pre}(\text{copy}(\text{create}()))$  would be “the same” as  $\text{create}()$ .
- We shall review and modify our predecessor function,  $\text{obs\_Pre}$ , later in this book.

A document trace is a history trail, i.e., a sequence of documents, from an original to the present document, whether a copy or a version such that the first document of the sequence is the document, the  $i$ th document in the sequence is the predecessor of the  $i - 1$ st document in the sequence, and hence such that the last document in the sequence is the original. Thus one can establish the full history that any document has undergone since the creation of its “ultimate predecessor”.

### value

```

obs_doc_trace: D → D*
obs_doc_trace(d) ≡
  if is_oD(d) then ⟨d⟩ else ⟨d⟩^obs_doc_trace(obs_Pre(d)) end

```

### Annotations:

- We name the document trace function  $\text{obs\_doc\_trace}$  since it is really an observer function (it is being “defined” solely in terms of, in this case one observer function).
- The document trace of an original document is the singleton sequence of that document.
- The document trace of a copy or an edited version ( $d$ ) is the prefix concatenation of the singleton sequence of that document ( $d$ ) with the document trace of the predecessor document of  $d$ .
- Termination is guaranteed since only a finite number of copies and edits can have taken place on any document.

We can now complete the induction part of the axiom above  $\text{obs\_E}(\text{copy}(\text{edit}(e,d))) = e \wedge \dots$

### axiom

$$\forall d:D \bullet$$

$$\forall i:\text{Nat} \bullet i \in \text{inds } \text{obs\_doc\_trace}(d) \Rightarrow$$

$$\text{is\_eD}(\text{obs\_doc\_trace}(d)(i)) \Rightarrow$$

$$\forall j:\text{Nat} \bullet j \in \text{inds } \text{obs\_doc\_trace}(d) \wedge j < i \wedge$$

$$\forall k:\mathbf{Nat} \bullet k \in \{j, i-1\} \wedge \text{is\_cD}(\text{obs\_doc\_trace}(d)(k)) \Rightarrow \\ \text{obs\_E}(\text{obs\_doc\_trace}(d)(i)) = \text{obs\_E}(\text{obs\_doc\_trace}(d)(k))$$

**Annotations:**

- For all documents
- and for all indices,  $i$ , into the trace of such documents
- if the  $i$ 'th document of that trace is an edited version
- then for all lower indices  $j$ , before  $i$ ,
- if all documents  $(\text{obs\_doc\_trace}(d)(k))$  of the trace properly  $j$  and  $i-1$  are copies,
- then we can observe in these copies the same editing value.

**2.1.4 Annotated Original Documents**

We modify the `copy` function and the notion of an original document, `od:oD`. We now annotate original document by a trace of “has been copied” markers. The document resulting from `create()` has an empty such trace. The document resulting from `copy(create())` has a singleton trace of one “has been copied” marker. Each additional copying of a marked original adds one “has been copied” marker to the trace.

Two original documents which differ only in number of “has been copied” markers are otherwise considered the same original.

**type**

`hbc_Mark == hbc`

**value**

`obs_hbc_Marks: oD → hbc_Mark*`

**axiom**

`obs_hbc_Marks(create()) ≡ ⟨⟩ ∧`

`∀ od:oD • obs_hbc_Marks(copy(od)) = ⟨hbc⟩^obs_hbc_Marks(od)`

**value**

`disregard_Marks: D → D`

`disregard_Marks(d) as d'`

`obs_hbc_Marks(d') = ⟨⟩ ∧ obs_Pre(d) = obs_Pre(d')`

`differ_by_1_Mark: D × D → Bool`

`differ_by_1_Mark(d, d') ≡`

`obs_hbc_Marks(d) = tl obs_hbc_Marks(d') ∧`

`disregard_Marks(d) = disregard_Marks(d')`

**Annotations:**

- `hbc_Mark` names a concrete type. Its only value is `hbc`. `hbc` is not further defined.
- `obs_hbc_Marks` is an observer function. It applies to original documents and yields a possibly empty list of `hbc:hbc_Mark*` of `hbc` markers.

- The list of `hbc` markers of a fresh, “virgin” original is empty.
- The list of `hbc` markers of any original that has been copied (once or more) has one more `hbc` marker than the original from which it was copied.
- We can view a document without its “has been copied” marks. That is the function of the `disregard_Marks` function.
- Two documents are, in a sense, the same if they differ only by one or more marks.

We now redefine the predecessor observer function.

**value**

`obs_Pre`:  $D \rightarrow D$

**axiom**

`obs_Pre(copy(d)) = d`  $\wedge$

$\forall d:D, e:E \bullet$

`obs_Pre(edit(e,d)) = d`  $\wedge$

$\forall od:oD \bullet$

`obs_hbc_Marks(od) =  $\langle \rangle$   $\Rightarrow$  obs_Pre(od) = chaos  $\wedge$`

*/\* the above is the same as \*/ `obs_Pre(create()) = chaos`  $\wedge$*

`obs_Pre(copy(od)) = od`

Later we shall augment the “has been copied” marker with location and time of copying.

### 2.1.5 Document Family Trees

Each document creation may give rise to a whole set of documents: copies of documents (for each copy a new document arises while the document from which it was copied basically remains), and edited versions of documents (for each version the number of documents remain the same). Given an original document one can establish the family tree of documents descending from the originally created document.

A document family tree consists of nodes and stems (i.e., branches). Nodes, other than the root node, designate operations performed on documents. The root node designates the “moment” before “creation”! Stems designate documents. A node, other than the root node, has one input stem and, for any node, one or two output stems. The input stem of a node is (also) said to be incident upon that node, and to designate the predecessor document of the new document resulting from the node operation. The output stem is, or the output stems are said to emanate from the node. The root node designates the create operation. Any other node designates either an edit or a copy operation. If a node designates an edit operation then it has one output stem and that stem designates the edited version of the document designated by the stem incident upon the edit node. If a node designates a copy operation then it has two output stems: one of these stems designate the (input) document designated by the stem incident upon the copy node while the other stem

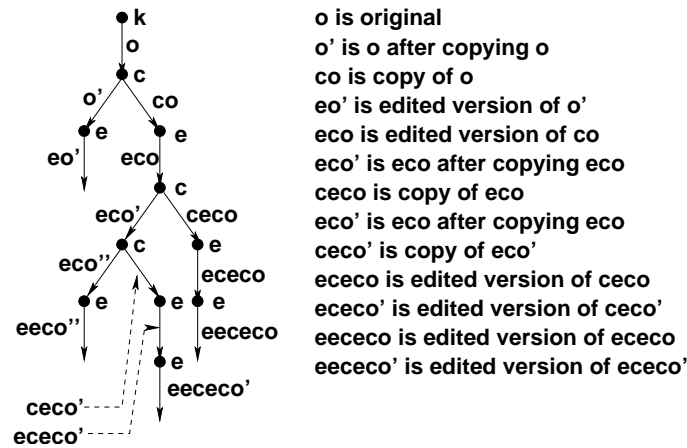


Fig. 2.1. A document family tree

designates the copy of that (input) document. Finally a document family tree ends in leaves which are stems, i.e., documents. From any stem in a document tree one can establish the unique path of stems from that stem back to the original document designated by the stem emanating from the root node. Such a path is a document trace. As for the general, i.e., abstract concept of trees one can speak of subtrees. If a stem is incident upon a node, then that node is the sub-root of a subtree which we shall here call a document tree (as distinguished from a document family tree). A (sub-)root of a document [family]tree<sup>2</sup> may have one or two subtrees, i.e., document trees: one of the (sub-)root designates the [create] (edit)<sup>3</sup> operation, two if it designates the copy operation.

**type**

```

DFT' = mkCreate() × oD × DT
DFT = { |dft:DFT' • wfDFT(dft)| }
DT == nil | ET | CT
ET = mkET(mkEdit(efns:(fe:FE,be:BE)),(ed:eD,dt:DT))
CT = mkCT(mkCopy(),(d:D,dt:DT),(cd:cD,dt':DT))

```

**value**

```

wfDFT: DFT' → Bool
wfDFT(⊥,od,dt) ≡
  case dt of
    nil → true,

```

<sup>2</sup>The phrase: (sub-)root of a document [family]tree reads as follows: root of a document family tree or sub-root of a document tree.

<sup>3</sup>The phrase: (sub-)root designates the [create] (edit) reads as follows: root designates the create or sub-root designates the edit.

```

    _ → wfDT(dt)(od)
end

```

```

wfDT: DT → D → Bool
wfDT(dt)(d) ≡
  case dt of
    nil → true,
    mkET((fe,be),(ed,dt'))
      → preEpost((fe,be),d,ed) ∧ wfDT(dt')(ed),
    mkCT(mkCopy(),(d',dt'),(cd,dt''))
      → preCpost(d,d') ∧ wfDT(dt')(d') ∧ wfDT(dt'')(cd)
  end

```

```

preEpost: E × D × eD → Bool
preEpost((fe,be),d,ed) ≡ ...
/* see postcondition of the edit function on page 14 */

```

```

preCpost: D × D → Bool
preCpost(d,d') ≡ disregard_Marks(d')=d

```

### Annotations:

- DFT' defines the Cartesian of not necessarily well-formed document tree.
- mkCreate(), oD and DT are the types of the components of the document tree.
- mkCreate() is strictly speaking not necessary, but is introduced so that all nodes possess an operation designator.
- oD designates the stem emanating from the mkCreate() node.
- DT designates the possibly empty sub-tree “attached” to the stem, i.e., upon which the stem may be incident.
- DT is thus either nil (i.e., the stem is a leaf) or is an edit tree et:ET or a copy tree ct:CT.
- An edit tree mkET(mkEdit(efns:(fe:FE,be:BE)),(ed:eD,dt:DT)) has sub-root node mkEdit(efns:(fe:FE,be:BE)) and one sub-tree (ed:eD,dt:DT).
  - ★ The sub-root node designates the editing functions mkEdit(efns:(fe:FE,be:BE)).
  - ★ The forward editing function fe “works” on the document of the stem incident upon this sub-root node.
  - ★ The backward editing function be “works” on the document of [the edited version stem ed:eD] emanating from this sub-root node.
  - ★ dt:DT designates a possible sub-tree of the stem emanating from this sub-root node.
- A copy tree mkCT(mkCopy(),(d:D,dt:DT),(cd:cD,dt':DT)) has sub-root node mkCopy() and two sub-trees (d:D,dt:DT) and (cd:cD,dt':DT).
  - ★ The sub-root node designates the copy function mkCopy().

- ★ One (here shown as “the left”) sub-tree  $(d:D, dt:DT)$  designates the document  $d:D$  being copied, hence “carried” forward, and its sub-tree  $dt:DT$ .
- ★ One (here shown as “the right”) sub-tree  $(cd:cD, dt':DT)$  designates the document copy  $cd:cD$ , and its sub-tree  $dt':DT$ .
- A number of constraints must be satisfied for a document history tree,  $dft$ , to be proper, i.e., to be well-formed  $wfDFT(dft)$ .
  - ★ We can ignore the Cartesian  $mkCopy()$  component of  $dft$ .
  - ★ If the sub-tree component  $dt$  is nil then the whole document history tree is well-formed.
  - ★ Otherwise the well-formedness of  $dft$  is the well-formedness of  $dt$  in the context of the incident document  $od$ .
- The well-formedness  $wfDT(dt)(d)$  of a sub-tree  $dt$  in the context of an incident document  $d$  is likewise defined by cases:
  - ★ If  $dt$  is nil then well-formedness is guaranteed.
  - ★ If  $dt$  is an edit sub-tree  $mkET((fe,be),(ed,dt'))$  then well-formedness is a conjunction of
    - the edit pre/post condition  $preEpost((fe,be),d,ed)$  explained earlier, and
    - the well-formedness of the version document sub-tree  $dt'$ .
  - ★ If  $dt$  is a copy sub-tree  $mkCT(mkCopy(),(d',dt'),(cd,dt''))$  then well-formedness is a conjunction of
    - the copy pre/post condition  $preCpost(d,d')$  where  $d'$  is the document being copied — and after copying,
    - the well-formedness of the master<sup>4</sup> document sub-tree  $wfDT(dt')(d')$ , and
    - the well-formedness of the copied document sub-tree  $wfDT(dt'')(cd)$ .

### 2.1.6 Document Family States

A state of a document family tree is a breadth-first set of stems of the tree. A breadth-first set of stems of a document family tree is one whose stems belong to distinct paths. Fig. 2.2 shows 11 states of a document family tree.

The idea is that there is an initial state, here  $s0$ , of the tree, and that there is a final state, here  $s10$ , of the tree. The initial state, here  $s0$ , designates the initial, i.e., the original document. The final state, here  $s10$ , designates a notion of final documents. A final state means that no further operations are to be performed on members of a set of documents. (“Case closed.”) Please note that the final state of any document family tree is unique — as is the initial state. Please also note that a void document, i.e., a copy of a copy of ... a

---

<sup>4</sup>We shall move this notion way back, towards the front of this book: the master document is the document being copied.

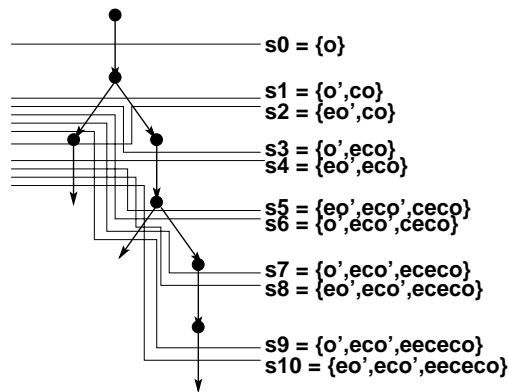


Fig. 2.2. Document family states

copy of an original document may be a final document.<sup>5</sup> Intermediate states designate possible collections of non-final documents. Thus a non-final state has one or more successor states. Usually there may be several ways of making state transitions from the initial state to the final state. Possible sequences of states are indicated by:

$$\begin{aligned}
 s_0 &\mapsto s_1 \mapsto s_3 \mapsto s_6 \mapsto s_7 \mapsto s_9, \\
 s_0 &\mapsto s_1 \mapsto s_2 \mapsto s_4 \mapsto s_8 \mapsto s_{10}.
 \end{aligned}$$

From a document family tree we can compute all states and all possible initial to final state sequences.

#### type

$$\Sigma = \{|\sigma:D\text{-set}\cdot\sigma\neq\{\}\}$$

#### value

$$\text{States: DFT} \rightarrow \Sigma\text{-set}$$

$$\text{Traversal: DFT} \rightarrow \Sigma^*$$

$$\text{States(dft)} \equiv \dots$$

$$\text{Traversal(dft)} \equiv \dots$$

#### Annotations:

- 
- 
- 

<sup>5</sup>The reader may feel uncomfortable having such void copies “floating” around, seemingly to no effect. But that is the cost of not imposing constraints that would otherwise impose what we consider unnatural limitations on what can be done to documents.

- 
- 
- 

### 2.1.7 Document Community

By a document community we mean a set of uniquely identified document family trees.

#### type

Did  
 $\text{DoCo} = \text{Did} \xrightarrow{m} \text{DFT}$

#### Annotations:

- 
- 
- 
- 
- 
- 

No two states of (two) distinctly named document family trees share states, i.e., have one or more documents in common.

#### value

$\text{wfDoCo}: \text{DoCo} \rightarrow \mathbf{Bool}$   
 $\text{wfDoCo}(\text{doco}) \equiv \dots$

#### Annotations:

- 
- 
- 
- 
- 
- 

## 2.2 Discussion of First Model of Document Intrinsic

There seems to be a number of problems with the model so far: Documents, whether manifest by humans senses (such as paper documents) or by technical/scientific apparatus (such as MS Word, L<sup>A</sup>T<sub>E</sub>X (.tex) files, portable document format (.pdf) files or postscript (.ps)files) always have a unique location in space. Operations on documents occur at certain times and these operations may, or may not “take time to perform”. Finally we did not mention

any notion of document identity: two documents which differ in some way (location, time of application of, say, most recent operation, content, etc.) can be claimed to have unique, i.e., distinct identities. We will, in the next two sections propose concrete models of locations and time of operation invocation.

### 2.3 A Concrete Model of Locations

We introduce a spatial notion of location. Mathematically we consider a location to be a dense point set equipped with some “neighbourhood” (or “infinitesimally close” predicate). No two otherwise distinct documents can occupy overlapping locations. Thus all distinct documents of a document family state occupy distinct, non-overlapping locations. And similarly for document communities.

We now extend our simplistic model of document intrinsics. From documents we can now observe their location. When creating or copying a document a single location is provided. The original document being created “receives” the given location. The document copy being established likewise “receives” the given location. The document from which the copy was made retains its location. The document resulting from an edit retains the location of the document being edited. We finally add a new operation on documents: Moving a document from one location to another, therefrom distinct location. The move shall result in the location of the moved document changing from what it was before the move to the given location. We shall, when now considering the create, copy, edit and move operations not consider whether the implied locations may interfere with locations of other documents of a family or community.

**type**

L

**value**

=:  $L \times L \rightarrow \mathbf{Bool}$

infinitesimally\_close:  $L \times L \rightarrow \mathbf{Bool}$

**axiom**

$\forall l, l': L \bullet \text{infinitesimally\_close}(l, l') \Rightarrow l \neq l'$

$\forall l, l', l'': L \bullet l' \neq l'' \wedge$

$\text{infinitesimally\_close}(l, l') \wedge \text{infinitesimally\_close}(l, l'') \Rightarrow$

$\text{infinitesimally\_close}(l', l'') \dots$

**value**

create:  $L \rightarrow \text{oD}$

copy:  $D \times L \rightarrow D \times \text{cD}$

edit:  $E \times D \rightarrow \text{eD}$

move:  $D \times L \xrightarrow{\sim} D$ , **pre**:  $\text{move}(d, l): \text{obs\_L}(d) \neq l$

obs\_L:  $D \rightarrow L$

**axiom**

$$\forall l:L, e:E \bullet$$

$$\text{obs\_L}(\text{create}(l)) = l \wedge$$

$$\text{let } (d', cd) = \text{copy}(d, l) \text{ in}$$

$$\text{preCpost}'(d, d') \wedge \text{obs\_L}(cd) = l \text{ end } \wedge$$

$$\text{let } ed = \text{edit}(e, d) \text{ in } \text{obs\_L}(ed) = \text{obs\_L}(d) \text{ end } \wedge$$

$$\text{obs\_L}(\text{move}(d, l)) = l$$
**Annotations:**

- 
- 
- 
- 
- 
- 

**2.4 A Basic Concrete Model of Time****2.4.1 Time**

We introduce a temporal notion of time. Mathematically we consider time to be a linear dense point ordering. Each document operation: create, copy, edit and move occurs at a specific time (and lasts no time).

We now extend our simplistic model of document intrinsics. From documents we can now observe the time of their last operation. When creating, copying, editing and moving a document a single time is provided. The original document being created “receives” the given time. The document copy being established likewise “receives” the given time. The document from which the copy was made retains its time. The document resulting from an edit “receives” the given time. The move shall result in a moved document marked with the given time. We shall, when now considering the create, copy, edit and move operations not consider whether the implied times are coincident with times of other documents of other the same family or other families. Previous documents of any documents retain their times of operation applications.

**type**

$$T$$
**value**

$$\text{obs\_T}: D \rightarrow T$$

$$\text{create}: L \times T \rightarrow oD$$

$$\text{copy}: D \times L \times T \rightarrow D \times cD$$

$$\text{edit}: E \times D \times T \rightarrow eD$$

$$\text{move}: D \times L \times T \xrightarrow{\sim} D, \text{pre: } \text{move}(d, l): \text{obs\_L}(d) \neq l$$
**axiom**

$$\forall t:T, e:E \bullet$$

$$\text{obs\_T}(\text{create}(\_,t)) = t \wedge$$

$$\mathbf{let} (d',cd) = \text{copy}(d,\_,t) \mathbf{in}$$

$$\text{preCpost}'(d,d') \wedge \text{obs\_T}(cd)=t \mathbf{end} \wedge$$

$$\mathbf{let} ed = \text{edit}(e,d,t) \mathbf{in} \text{obs\_T}(ed)=t \mathbf{end} \wedge$$

$$\text{obs\_T}(\text{move}(d,l,t)) = t$$
**Annotations:**

- 
- 
- 
- 
- 
- 

**2.5 Located and Timed Documents**

We wish to record that for every document that has been copied the fact that it has been copied: tie and place.

**value**

$$\text{has\_been\_copied}: D \rightarrow \mathbf{Bool}$$

$$\text{when\_where\_copied}: D \xrightarrow{\sim} L \times T$$

$$\text{otherwise\_the\_same}: D \times D \rightarrow \mathbf{Bool}$$
**axiom**

$$\forall d:D \bullet$$

$$\sim \text{has\_been\_copied}(d) \equiv \text{when\_where\_copied}(d)=\mathbf{chaos} \wedge$$

$$\forall l:L, t:T \bullet$$

$$\mathbf{let} (c',cd) = \text{copy}(d,l,t) \mathbf{in}$$

$$\text{has\_been\_copied}(d') \wedge \text{when\_where\_copied}(d')=(l,t) \wedge$$

$$\text{otherwise\_the\_same}(d,d') \mathbf{end}$$
**Annotations:**

- 
- 
- 
- 
- 
-

### 2.5.1 Time and Space (Locations)

No document can at the same time be in two different locations:

```

axiom
   $\forall d, d': D \bullet$ 
    let  $(l, t) = (\text{obs\_L}(d), \text{obs\_T}(d)), (l', t') = (\text{obs\_L}(d'), \text{obs\_T}(d'))$  in
       $(t=t' \wedge l=l' \equiv d=d') \wedge (t=t' \wedge l \neq l' \equiv d \neq d')$  end

```

**Annotations:**

- 
- 
- 
- 
- 
- 

## 2.6 Unique Document Identifiers

Given that creation and copying times and locations are unique we can introduce a notion of unique document identifications. For a document family that is simple enough:

```

type
  Uid
value
  obs_Uid: D → Uid
axiom
  let  $od = \text{create}()$  in
    let  $uid = \text{obs\_Uid}(d)$  in
       $\forall d: D, e: E, t: T, l: L \bullet$ 
        let  $dt = \text{doc\_trace}(d)$  in  $dt(\text{len } dt) = od \Rightarrow$ 
           $\forall i: \text{Nat} \bullet i \in \text{inds } dt \Rightarrow$ 
             $\text{is\_oD}(dt(i)) \Rightarrow \text{obs\_Uid}(dt(i)) = uid \wedge$ 
             $\text{is\_eD}(dt(i)) \Rightarrow \text{obs\_Uid}(dt(i)) \neq uid \wedge$ 
             $\text{is\_cD}(dt(i)) \Rightarrow \text{obs\_Uid}(dt(i)) \neq uid$ 
        end end end

```

**Annotations:**

- 
- 
-

- 
- 
- 

For a community we need to distinguish two originals by their distinct document family identifiers. That is: The Uid is some concoction of a document family identifier, a copy or edit operation time and a copy or edit operation location.

## 2.7 Discussion of the Simplistic Model of Documents