Double-Entry Bookkeeping

Dines Bjørner
Fredsvej 11, DK-2840 Holte, Danmark
E–Mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

September 2, 2023

Abstract

We present a model of the domain of double-entry bookkeeping.

Contents

1 Introduction

The concept of double-entry bookkeeping is assumed known[1]. The model that we shall present is a convention model, and is expressed in RSL [9], the RAISE[2], [10] Specification Language. We refer to [1] for a general introduction to rigorous software engineering. And we refer to [2] for introductions to the domain modelling approach that – generally – lies behind our modeling of this, the double-entry bookkeeping domain. Recent papers summarizing domain moelling are [3, 4, 7].

You may wish to first study the simple (UNIX-like) file system model of Appendix A.

more to come

2 A Formal Model

We present a model of the domain of double-entry bookkeeping. The description alternates beween numbered

- **narratives** and
- **formulas**.

The formula definitions are indexed, cf. Appendix B.

2.1 Double-Entry Account Types

1. A double-entry bookkeeping account, traditionally, consists of two accounts, an income/outcome (earnings/expenditure) account, IOA, and an assets/liabilities account, ALA.[3]
2. In addition a double-entry bookkeeping account relates to, is for, a year – further undefined.
3. Let deba be a DEBA value – one that we shall refer to as a "global" example value – one whose components are ioa, ala and y.

type
1. DEBA = $\mathsf{IOA}_0 \times \mathsf{ALA}_0 \times$ Year
2. Year

value
3. deba:(ioa,ala,year):DEBA = ...

[1] We refer to the Wikipedia article on double-entry bookkeeping (or ...accounting): https://en.wikipedia.org/wiki/Double-entry_bookkeeping

[2] RAISE: Rigorous Approach to Industrial Software Engineering

[3] Pls., for a moment, till after the presentation of the formal model, Sect. 2.3 on page 5, disregard the type name suffixes $_{i,i-1}$ and $_{i+1}$.

4. An income/outcome account consists of zero, one or more elements of two kinds:
   - immediate income/outcome accounts, I_IOAs, and
   - embedded income/outcome accounts, E_IOAs;
   - and a sum.

   Both are maps, merged into one map using the map type union constructor, $\Cup$.

   type
   4. $\mathsf{IOA}_i = (\mathsf{I\_IOAs}_i \Cup \mathsf{E\_IOAs}_i) \times \mathsf{Sum}_{i+1}$

5. Immediate income/outcome accounts map income/outcome entry names, IOENm, into audit trails, Audit_Trail.

   type
   5. $\mathsf{I\_IOAs}_i = \mathsf{IOENm}_i \xrightarrow[m]{} \mathsf{Audit\_Trail}_i$

6. Embedded income/outcome accounts map income/outcome directory names, IODNm, into income/outcome audits.

   type
   6. $\mathsf{E\_IOAs}_i = \mathsf{IODNm}_i \xrightarrow[m]{} \mathsf{IOA}_{i-1}$

7. Income/outcome entry and directory names are disjoint types of further undefined identifiers, ID.

   type
   7. IOENm_i :: ID, IODNm_i :: ID ,

8. An assets/liabilities account consists of zero, one or more elements of two kinds:
   - immediate assets/liabilities accounts, I_IOAs, and
   - embedded assets/liabilities accounts, E_IOAs;
   - and a sum.

   Both are maps, merged into one map using the map type union constructor, $\Cup$.

   type
   8. $\mathsf{ALA}_i = (\mathsf{I\_ALAs}_i \Cup \mathsf{E\_ALAs}_i) \times \mathsf{Sum}_{i+1}$
   8. $\mathsf{I\_ALAs}_i = \mathsf{ALENm}_i \xrightarrow[m]{} \mathsf{Audit\_Trail}_i$
   8. $\mathsf{E\_ALAs}_i = \mathsf{ALDNm}_i \xrightarrow[m]{} \mathsf{ALA}_{i-1}$

9. Asset/liability entry and directory names are disjoint types of further undefined identifiers, ID.

type
9. ALENm_i :: ID, ALDNm_i :: ID

10. An audit trail entry is a triplet[4]: an audit information element, AI_{i+1} (further undefined), a list of timed audit records, $\mathsf{TAR}_i{}^*$, and a sum, Sum_i[5]

type
10. $\mathsf{Audit\_Trail}_i = \mathsf{ATI}_i \times \mathsf{TAR}_i{}^* \times \mathsf{Sum}_i$

11. Audit trail information will presently be left further undefined.

type
11. ATI_i

12. Sums are the same as amounts.

type
12. $\mathsf{Sum}_i = \mathsf{Amount}_i$

13. **type**
13. $\mathsf{TAR}_i = \mathbb{TIME}_i \times \mathsf{Ref}_i \times \mathsf{Amount}_i$

14. References are further unexplained references to documents evidencing the audit entry.

type
14. Ref_i

15. Amounts are real numbers with a currency designator[6].

15. Amount = **Real US $**

[4] An audit trail is a step-by-step record by which accounting, trade details, or other financial data can be traced to their source. Audit trails are used to verify and track many types of transactions, including accounting transactions and trades in company accounts `https://www.investopedia.com/terms/a/audittrail.asp`

[5] Suffixes, $_{i,i+1}$, will be explained right after this narrative+formalization text, Sect. 2.3 on the next page.

[6] Hee we have, arbitrarily (!), chosen **US $**

2.2 Summation

16. The *sum* element of the audit triplets must be the sum of the time-stamped value entries for that income/outcome or assets/liabilities account.

axiom [Sums add up]
16. ∀ (_,tarl,sum$_i$):(AI$_i$×TAR$_i$*×Sum$_i$)
16. • summation(tarl)=sum$_i$
value
16. summation: TAR* → **Real US $**
16. summation(tarl) ≡ **case** tarl **of** ⟨⟩ → 0, (_,_,v)^tarl′→ v+summation(tarl′) **end**

2.3 Suffix $_{0,i,i-1}$ and $_{i+1}$ Types

We now explain the rôle of the $_{0,i,i-1}$ and $_{i+1}$ type suffixes.

First we summarize their occurrences.

As suffix $_0$ they are affixed the IOA and ALA when they first appear, Item 1 on page 2. They are then, in subsequent type descriptions, affixed the IOA and ALA type names. From there they are carried forward to the ALENm, ALDNm type names. They are decreased from their left-hand side type definition occurrence in definition Item 5 on page 3. They are increased from their left-hand side type definition occurrence in definition Item 10 on the facing page.

Then we explain their deployment.

Initially we let the 'base' type by suffixed $_0$. In subsequent type definitions we let the relevant left-hand side type identifiers be suffixed by $_i$. The specific value of i may be 0, or -1, or -2, etc. Occurrences of suffixed type name on the right-hand side of a type description "inherits" the right-hand side suffix. Embedded book-keeping directories are thus ascribed decreasing suffixes.

Then we explain their purpose.

Lean back and reflect ! Formalization here means several things. One of these is that we abstract from physical representation of the defined type values. In the case of this example we can think of two representations. There is the abstract representation, as mathematical quantities. And there is the concrete representation of double-entry accounts: either as maintained in computer storage, or as displayed on a computer screen, or as printed on a sheet of paper. The computer storage representation we shall not consider here. The computer display representation has, amongst others, the following characteristics: it is represented, line-wise, from the top of a possibly scrolled screen, to the bottom – as far down as is necessary, with each screen-line extending from a fixed left-most position to anywhere – as far to the right on a left-to-right "scrolled" screen as is necessary, ! The printed paper representation has, amongst others, the following characteristics: It is constrained by page size. Limited page width and height (actually depth). So unless one uses "fold-out" and "fold-down" sheets of paper for the physical paper representation of accounts, one is constrained. We shall in the following "assume" the presence of such "fold" characteristics.

Now we can tackle the issue of the indices.

The issue, really, is this: We wish to indicate, by the suffixes, the amount of line space, right-to-left that the audit trail: 'audit information$_i$', 'audit record triplet$_i$', 'sum$_i$', and 'sum$_{i+1}$' texts might conveniently be displayed on a computer screen or on (possibly"folded")

paper – right-to-left indented – with index $_0$ entries furthest to the right, and with index $_{i-1}$ entries to the left of $_i$ entries. We leave it to the reader to interpret Fig. 1 with respect to the above explication!

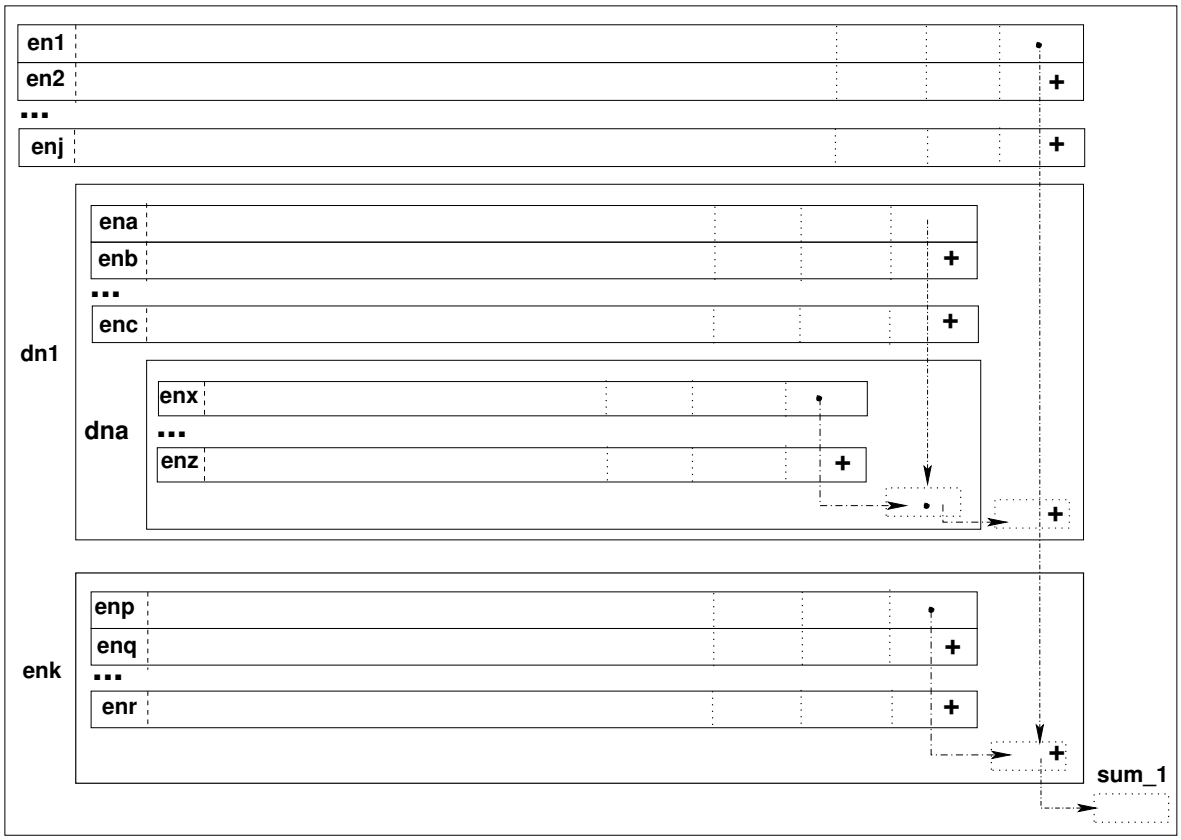

Figure 1: A Fragment Double-entry Bookkeeping Account

2.4 Account Identifiers

17. An account identifier is an account entry and directory name list, i.e., a sequence of one or more account names, either of income/outcome or asset/liability class names

18. – such that the last name in the list is an entry name, the others being directory names.

type
17. AI = IOCNm* | ALCNm*
17. IOCNm = IOENm | IODNm, ALCNm = ALENm | ALDNm

axiom [well-formed paths]
17. ∀ ai:IOCNm* • ai≠⟨⟩ ∧ is_IOENm(ai[**len** ai])
17. ∧ ∀ i:**Nat**•i ∈ **inds** ai \ {**len** ai} ⇒ is_IODNm(ai(i))
17. ∀ ai:ALCNm* • ai≠⟨⟩ ∧ is_ALENm(ai[**len** ai])
17. ∧ ∀ i:**Nat**•i ∈ **inds** ai \ {**len** ai} ⇒ is_ALDNm(ai(i))

19. From an account we can extract the set of all its account identifiers. We refer to Item 64 on page 18.

```
type
19. Path = IOCNm*
value
19. end_paths: IOA → P-set
19. end_paths(ioa) ≡
19.     case ioa of
19.         [ ] → {⟨⟩},
19.         [ mkIOENm(id)↦e ] ∪ ioa′ → {⟨mkIOENm(id)⟩} ∪ end_paths(ioa′),
19.         [ mkIODNm(id)↦d ] ∪ ioa′ → {⟨mkIODNm(id)⟩⌢p:P•p ∈ end_paths(d)} ∪ end_paths(ioa′)
19.     end
```

Similar for end_paths of ALA.

2.5 Account Trails

20. An account identifier designates an audit trail.

```
value
20.  entry: Path → IOA → Audit_Trail
20.  entry(p)(ioa) ≡
20.      case p of
20.          ⟨mkIOENm(id)⟩ → ioa(mkIOENm(id)),
20.          ⟨mkIODNm(id)⟩⌢p′ → entry(p′)(ioa(mkIODNm(id)))
20.      end
20.      pre: p ∈ end_paths(ioa)
```

Similar for audit trails of ALA.

2.6 Account Class Types

21. An account class type[7] is either **"nominal"**, **"tangible_real"**, **"intangible_real"**, **"artificial_personal"**, **"natural_personal"**, or **"representative_personal"** –

[7] Accounts can be partitioned into the following types:

- **Nominal:** A nominal account is a general ledger containing the transactions of a business, namely expenses, incomes, profits and losses. It contains all the transactions that occur in one fiscal year. Furthermore, it resets to zero and starts afresh when the next fiscal year begins.

  Examples of nominal accounts are Commission Received, Salary Account, Rent Account and Interest Account.
- **Real:**
  - **Tangible:** Tangible assets include land, buildings, machinery, furniture, etc.
  - **Intangible:** Intangible assets include goodwill, patents, copyrights, etc.
- **Personal:** relates to people, associations and companies.
  - **Artificial:** An artificial personal account represents bodies which are not human beings but act as separate legal entities according to the law. For example, government bodies, hospitals, banks, companies, cooperatives, partnerships, etc.

22. corresponding to appropriate predicates.

type
21. AType = ″nominal″
21. | ″tangible_real″ | ″intangible_real″
21. | ″artificial_personal″ | ″natural_personal″ | ″representative_personal″
value
22. account_type: AI → AType
22. is_nominal: AI → **Bool**
22. is_tangible_real: AI → **Bool**
22. is_intangible_real: AI → **Bool**
22. is_artificial_personal: AI → **Bool**
22. is_natural_personal: AI → **Bool**
22. is_representative_personal: AI → **Bool**

2.7 Double-entry Bookkeeping Type Constraints, I

23. For audit trails, the time stamped income/outcome and asset/liability entries are chronologically ordered – earlier times precede later times.

axiom [Chronologically ordered Audits]
23. ∀ tar:TAR•∀ i:**Nat**•{i,i+1}⊆**inds** tra⇒**let** ((t,_,_),(t′,_,_))=(tar(i),tar(i+1)) **in** t<t′ **end**

2.8 Income/Outcome and Asset/Liability Class Name Relations

The basic idea of double-entry bookkeeping is that the two kinds of accounts somehow "balance" one-another – according to the equation:

- **if** revenue (income) = expenses (outcome) **then** assets = liabilities + equity.

This implies that the two accounts, IOA (income/outcome) and ALA (assets/liabilities) have their entry names "somehow" relate. We shall here simplify that "somehow" relation into two user-defined maps:

24. To each income/outcome account identifier there corresponds, for that double-entry book keeping account, one or more asset/liability account identifiers; and

25. To each assets/liabilities account identifier there corresponds, for that double-entry book keeping account, one or more income/outcome account identifiers.

– **Natural:** A natural personal account represents human beingsfor example, a Capital account, a Drawings account, Creditors, Debtors, etc.
– **Representative:** This type of personal account represents the accounts of natural or artificial entities. However, the transactions in this type of account either belong to the previous or the coming year. For example, a representative personal account can contain information on an employees due salary from last year. Also, it can represent the amount of rent a company paid in advance for the coming year.

We leave unspecified, ..., how that association comes about!

26. Let ioal_map be the asscociation for a givn double-entry account.

27. The income/outcome to asset/liability associations must cover all incomes/outcomes and assets/liabilities, and vice versa.

type
24. Assoc_IOAL_Map: IONm* −m> (ALNm*)-**set**
25. Assoc_ALIO_Map: ALNm* −m> (IONm*)-**set**
value
26. ioal_map:Assoc_IOAL_Map = ... ; alio_map:Assoc_ALIO_Map = ... ;
axiom [Income/Outcome–Asset/Liability Association]
27. **dom** ioal_map = end_parths(ioa) ∧ ∪ **rng** ioalmap = end_paths(ala) •

One can have other relations, but this one suffice for the illustration of the concept of double -entry bookkeeping.

With this relation in hand we can always know which, thus "mutual", accounts to update – but the amounts with which respective accounts are updated is left to the accountant!

Example 1 Mutual Account Updates: Let ai stand for the income/outcome account of *payment for furniture*, then the corresponding asset/liability account could be that of *furniture*.

Similar for the income/outcome account of *depreciation/appreciation of furniture* – the same asset/liability furniture account maps into (at least) the two income/outcome accounts: payment for furniture and payment for furniture

more to come

■

2.9 Account Structures

28. By an account structure we mean "almost the same" as an account! Where an income/outcome or an assets/liabilities account – during the accounting year – reflects time-stamped entries an account structure does not contain the time-stamped entries.

type
28. DEBAS = IOAS × ALAS
28. IOAS = (IOENm $\overrightarrow{m}$ IOAI) ⩊ (IODNm $\overrightarrow{m}$ IOAS)
28. ALAS = (ALENm $\overrightarrow{m}$ ALAI) ⩊ (ALDNm $\overrightarrow{m}$ ALAS)

29. From an income/outcome account we can observe an income/outcome account structure, and the same for asset/liability accounts.

value
29. observe: (IOA → IOAS) | (ALA → ALAS)
29. observe(a) **as** as; **post**: end_paths(a) = end_paths(as)

2.10 Establish Initial Accounts

Initial accounts are accounts that have not been completed wrt. real[8] assets.

30. From an income/outcome account structure we can establish an initial income/outcome account , and the same for assets/liability accounts.

value
29. establish: (IOAS → IOA) | (ALAS → ALA)
29. establish(as) **as** a
29. **post**: end_paths(as) = end_paths(a)
30. ∧ ∀ nm:(IOCNm|ALCNm)•nm ∈ end_paths(a) ⇒ entry(nm)(a) ∈ {[ ],⟨⟩}

Real assets need be initialized. That can be done by commands mentioned below.

2.11 Double-Entry Account Command Types

New accounts, i.e., accounts deba:DEBA where no such exists for that enterprise, can be established through these commands:

31. **new_DEBA:** This command assumes that those responsible for the double-entry bookkeeping accounts have decided upon
    (a) an account structure, debas = (ioas,alas),
    (b) on an association relation, assoc_id,
    (c) on the amounts with with which real assets are to be initialized, and
    (d) an accounting year.

value
31a. debas:DEBAS = ...
31b. Assoc_Maps:(Assoc_IOAL_Map × Assoc_ALIO_Map) = ...
31c. Real_Assets:(AI $\xrightarrow[m]{}$ Amount) = ...
31d. year:Year = ...
31. new_DEBA: DEBAS×Assoc_Maps×Real_Assets×Year → DEBA
31. new_DEBA(debas,assoc_maps,real_assets,year) **as** deba
31. **pre**: $\mathcal{P}$(debas,assoc_maps,real_assets,year)
31. **post**: $\mathcal{Q}$(debas,assoc_maps,real_assets,year)(deba)

We leave it to the reader to fill in the **pre/post**, i.e., the $\mathcal{P}$, $\mathcal{Q}$, conditions!

2.11.1 Operating Commands

We shall illustrate hust one double-entry bookkeeping "command": that of entering a combined income/outcome – assets/libailities entry. We shall simplify that command. We shall simply "balance" the earnings/expenditue entry with a single assets/libailities entry.

32. **enter:** To post, mkEnter(...), a transaction the following information is required:

[8] Cf. footnote 7 on page 7

- an account identifier, ai,
- the time, τ,
- a reference, ref, to some transaction evidence, i.e., a document,
- and an amount, amount.

The time will be "generated" when acting upon the posting.

33. To enter a transaction, mkEnter(ai,ref,amount) into a double-entry bookkeeping account, deba=(ioa,ala,year),
34. at a recorded time, **record**_𝕋𝕀𝕄𝔼(),
35. and selecting, non-determinstically ($\prod$) one assets/liabilities path, ap,
36. results in a new double-entry bookkeeping account, deba=(ioa',ala',year'),
37. – provided the access identifier (path, ai) is of the accounts – and where the
38. end paths are the same,
39. years are the same,
40. income/outcome entries, except for the argument access identifier, are the same,
41. assets/liabilities entries are all the same, except for the argument access identifier, where
42. old and the new access path entries have
43. the same account information, ainfo, but the audit trail has been extended with the appropriate triplet, and where
44. all the intermediate sums have been adjusted – as have the full account sums!

```
type
32.  Enter :: P × Ref × Amount
value
32.  enter: Enter → DEBA ~→ DEBA
33.  enter(mkEnter(ip,ref,amount))(((ioas,isum),(alas,asum),year)) ≡
34.     let τ = record_𝕋𝕀𝕄𝔼()
35.         ap = ∏ ioal_map(ip) in
36.     ((ioas′,isum′),(alas′,asum′),year′) end
37.     pre: ip ∈ end_paths(ioas) ∧ ap ∈ end_paths(alas)
38.     post:  end_paths(ioas)=end_paths(ioas′) ∧ end_paths(alas)=end_paths(alas′)
39.         ∧ year=year′
40.         ∧ ∀ p:(IOCNm*)-set•p ∈ end_paths(ioas)\{ip} • entry(p)(ioas)=entry(p)(ioas′)
41.         ∧ ∀ p:(ALCNm*)-set•p ∈ end_paths(alas)\{ap} • entry(p)(alas)=entry(p)(alas′)
42.         ∧ let (info,tar,isum,isum′)=entry(ip)(ioas),(info′,tar′,isum″,isum‴)=entry(ip)(ioas′) in
43.           info=info′ ∧ tar′=tar⌢⟨(τ,ref,amount,sum+amount)⟩ end
42.         ∧ let (linfo,tar,sum,sum′)=entry(ap)(ala),(info′,tar′,sum″,sum‴)=entry(ap)(ala′) in
43.           info=info′ ∧ tar′=tar⌢⟨(τ,ref,amount,sum+amount)⟩ end
44.         ∧ adjusted_summations((ioas,isum),(alas,asum),(ioas,isum),(alas,asum))
```

Ler us recall where intermediate amounts, sums and summations appear in the double-entry bookkeeping accounts (ιitem,πage: $[\iota 4\pi 3]$, $[\iota 8\pi 3]$, $[\iota 5\pi 3]$, $[\iota 8\pi 3]$, $[\iota 10\pi 4]$, $[\iota 13\pi 4]$):

4. $\mathsf{IOA}_i = (\mathsf{I\_IOAs}_i \Cup \mathsf{E\_IOAs}_i) \times \mathsf{Sum}_{i+1}$
8. $\mathsf{ALA}_i = (\mathsf{I\_ALAs}_i \Cup \mathsf{E\_ALAs}_i) \times \mathsf{Sum}_{i+1}$
5. $\mathsf{I\_IOAs}_i = \mathsf{IOENm}_i \xrightarrow[m]{} \mathsf{Audit\_Trail}_i$
8. $\mathsf{I\_ALAs}_i = \mathsf{ALENm}_i \xrightarrow[m]{} \mathsf{Audit\_Trail}_i$
10. $\mathsf{Audit\_Trail}_i = \mathsf{ATI}_i \times \mathsf{TAR}_i^* \times \mathsf{Sum}_i$
13. $\mathsf{TAR}_i = \mathbb{TIME}_i \times \mathsf{Ref}_i \times \mathsf{Amount}_i$

45. The sum_i (intermediate sums) appearing (Item 10) in an Audit_trail's third (i.e., last) element is to be the summation of all the amounts of that audit's trail elements.

46. The sums_{i+1} appearing (Items 4, 8) as the last elements in IOA_i and ALA_i is to be the summation of all the intermediate sums.

value
45.
45.
45.
45.
45.
46.
46.
46.
46.
46.

2.11.2 A Double-entry Accounting Behaviour

The above domain description focuses on just the double-entry bookkeeping accounts and a few operations on such accounts. Now we "widen" our domain scope. The double-entry bookkeeping accounts are but one element, one beaviour, in any enterprise's bookkeeping. Other elements are the bookkeeper (accountants), the (human) staff, whose kookkeeping (accounting) behaviours perform the accounts. That is: a "wider" context places the above domain description as one behaviour, the double-entry account behaviour, and the set of one or more bookkeepers as a concurrently interacting set of other behaviours. We refer to the context of the bookkeeper behaviours as the *mereology* of the double-entry account behaviour.

47. That mereology can be abstracted as the set, bis:BI-**set**, of *unique identifiers*, bi:BI, of bookkeeper behaviours:

type
47. DEBA_Mer = BI-**set**, BI

whee BI is the type of the unique identifiers of bookkeepes.

48. We define a double-entry account behaviour. It is based on that behaviour having, as arguments, the
    - *unique identifier* (i:DEABI),
    - *mereology* (bis:BI-**set**) of
    - *static attribute* (year:Year), and the
    - *programmable attribute* (income/outcome and asssets/liabilities ioa:IOA,ala:ALA).
49. Besides other ... tasks the double-entry accounting behaviour non-deterministically externally, [],
50. offers to accept mkEnter(...) requests from accountant behaviours identifified by their unique identifier j.
51. Having accepted such a request the double-entry accounting behaviour honors that request, resutinng in an "update" ioa':IOA,ala':ALA (ignoring the year), and
52. resumes being a double-entry accounting behaviour –
53. offering, perhaps, ..., to accept other double-entry accounting requests.

```
type
48. DEABI
value
48. double_entry_account: DBEAI → Year → DEBA_Mer → (IOA×ALA) → ... Unit
48. double_entry_account(dbeai)(y)(bis)(ioa,ala) ≡
49.     ...
50.     [] let mkEnter(ip,ref,amount) = { ch[ {dbeai,bi} ] ? | bi:BI • bi ∈ bis } in
51.       let (ioa′,ala′,_) = enter(mkEnter(ip,ref,amount))(ioa,ala,y) in
52.       double_entry_account(dbeai)(y)(bis)(ioa′,ala′) end end
53.     ...
```

2.11.3 Reusing Existing Accounts

Resetting accounts, i.e., initial accounts deba:DEBA where such exists for that enterprise, can be established through these commands:

54. :
55. :
56. :
57. :
58. :

type
54.
55.
56.
57.
58.

3 Discussion

3.1 A Dichotomy

There is, however, a *dichotomy*.[9] The two contrasting things are:

- the *enterprise* and
- its *double-entry bookkeeping accounts*.

We have made no reference, at all (!), to the enterprise in the previous section. Yet, we know that we can (likewise, as we have now done it for "its" double-entry bookkeeping accounting) refer to — i.e., make a domain model for — the domain of the enterprise. We refer to [8]. That reference records domain models for

- *Nuclear Power Plants*,
- *Shipping*,
- *Rivers and Canals*,
- *A Retailer Market*,
- *Container Terminals*,
- *Documents*,
- *Urban Planning*,
- *Swarms of Drones*,
- *Credit Cards*,
- *Weather Information*,
- *Web-based Transaction Processing*,
- *The Tokyo Stock Exchange*,
- *Pipelines*,
- *Container Lines*,
- *The Market*,
- *Railways*,
- etcetera!

3.2 Unified Enterprise Domain Models

Here is how to do it "better"! – i.e., in a "coordinated" manner: While modeling the domain of relevant enterprises — i.e., such for which double-entry bookkeeping is relevant — model, at the same time, "its" double-entry bookkeeping accounts],! That is, when an action in the enterprise warrants a double-entry posting, then the combined, i.e., unified model, models that posting.

4 Conclusion

to be written

5 Bibliography

5.1 Bibliographical Notes

to come

[9] **Dichotomy:** A division or contrast between two things that are or are represented as being opposed or entirely different.

5.2 References

[1] Dines Bjørner. *Software Engineering, Vol. 1: Abstraction and Modelling; Vol. 2: Specification of Systems and Languages; Vol. 3: Domains, Requirements and Software Design*. Texts in Theoretical Computer Science, the EATCS Series. Springer, 2006.

[2] Dines Bjørner. *Domain Science & Engineering – A Foundation for Software Development*. EATCS Monographs in Theoretical Computer Science. Springer, 2021. A revised version of this book is [6].

[3] Dines Bjørner. *Domain Modelling*. Technical report, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, Denmark, December 2022. Chapter 3 of [**?**].

[4] Dines Bjørner. Domain Modelling. In Jonathan Bowen et al., editor, *Theories of Programming and Formal Methods: Essays Dedicated to Jifeng He on the Occasion of His 80th Birthday*, Lecture Notes in Computer Science, Festschrift. Spriger, August 2023.

[5] Dines Bjørner. Domain Modelling – A Primer. A short version of [6]. xii+202 pages[10], May 2023.

[6] Dines Bjørner. Domain Science & Engineering – A Foundation for Software Development. Revised edition of [2]. xii+346 pages[11], January 2023.

[7] Dines Bjørner. Domain Science: Modelling. *To be submitted*, August 2023. Institute of Mathematics and Computer Science. Technical University of Denmark.

[8] Dines Bjørner. Domain Case Studies:

- 2023: *Nuclear Power Plants, A Domain Sketch*, 21 July, 2023 `www.imm.dtu.dk/~dibj/2023/nupopl/nupopl.pdf`
- 2021: *Shipping*, April 2021. `www.imm.dtu.dk/~dibj/2021/ral/ral.pdf`
- 2021: *Rivers and Canals – Endurants – A Technical Note*, March 2021. `www.imm.dtu.dk/~dibj/2021/Graphs/Rivers-and-Canals.pdf`
- 2021: *A Retailer Market*, January 2021. `www.imm.dtu.dk/~dibj/2021/Retailer/BjornerHeraklit27January2021.pdf`
- 2019: *Container Terminals*, ECNU, Shanghai, China `www.imm.dtu.dk/~dibj/2018/yangshan/maersk-pa.pdf`
- 2018: *Documents*, TongJi Univ., Shanghai, China `www.imm.dtu.dk/~dibj/2017/docs/docs.pdf`
- 2017: *Urban Planning*, TongJi Univ., Shanghai, China `www.imm.dtu.dk/~dibj/2018/BjornerUrbanPlanning24Jan2018.pdf`
- 2017: *Swarms of Drones*, Inst. of Softw., Chinese Acad. of Sci., Peking, China `www.imm.dtu.dk/~dibj/2017/swarms/swarm-paper.pdf`

[10] This book is currently being trasnlated into Chinese by Dr. Yang ShaoFa, IoSCAS, Beijing and into Russian by Dr. Mikhail Chupilko, ISP/RAS, Moscow

[11] Due to copyright reasons no URL is given to this document's possible Internet location. A primer version, omitting certain chapters, is [5]

- 2013: *Road Transport*, Techn. Univ. of Denmark `www.imm.dtu.dk/~dibj/road-p.pdf`
- 2012: *Credit Cards*, Uppsala, Sweden `www.imm.dtu.dk/~dibj/2016/credit/accs.pdf`
- 2012: *Weather Information*, Bergen, Norway `www.imm.dtu.dk/~dibj/2016/wis/wis-p.pdf`
- 2010: *Web-based Transaction Processing*, Techn. Univ. of Vienna, Austria `www.imm.dtu.dk/~dibj/wfdftp.pdf`
- 2010: *The Tokyo Stock Exchange*, Tokyo Univ., Japan `www.imm.dtu.dk/~db/todai/tse-1.pdf`, `www.imm.dtu.dk/~db/todai/tse-2.pdf`
- 2009: *Pipelines*, Techn. Univ. of Graz, Austria `www.imm.dtu.dk/~dibj/pipe-p.pdf`
- 2007: *A Container Line Industry Domain*, Techn. Univ. of Denmark `www.imm.dtu.dk/~dibj/container-paper.pdf`
- 2002: *The Market*, Techn. Univ. of Denmark `www.imm.dtu.dk/~dibj/themarket.pdf`
- 1995–2004: *Railways*, Techn. Univ. of Denmark - a compendium `www.imm.dtu.dk/~dibj/train-book.pdf`

Experimental research reports carried out to "discover", try-out and refine method principles, techniques and tools, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, Denmark.

[9] Chris W. George, Peter Haff, Klaus Havelund, Anne Elisabeth Haxthausen, Robert Milne, Claus Bendix Nielsen, Søren Prehn, and Kim Ritter Wagner. *The RAISE Specification Language*. The BCS Practitioner Series. Prentice-Hall, Hemel Hampstead, England, 1992.

[10] Chris W. George, Anne Elisabeth Haxthausen, Steven Hughes, Robert Milne, Søren Prehn, and Jan Storbank Pedersen. *The RAISE Development Method*. The BCS Practitioner Series. Prentice-Hall, Hemel Hampstead, England, 1995.

A A Simple File System

As a preamble (!) for studying the main example of this report, i.e., the double-entry bookkeeping example, we "first" bring You this "teaser". It models the file system used on LINUX (Ubuntu) systems!

59. A file directory is a directory.
60. With respect to directories there are the (abstract) notions of files, f:F, (abstract) file and directory identifiers, fi:FI, di:DI, and directories, d:D[12].
61. In a (classical) file directory directories map these identifiers into respectively files or directories.
62.

type
59. FD = D
60. F, I, FI, DI
60. FI :: I, DI :: I
61. D = (FI $\xrightarrow[m]{}$ F) ⨄ (DI $\xrightarrow[m]{}$ D)

An example:

value
```
fd:FD = [ f0 ↦ file0,
          f1 ↦ file1,
          ...
          fn ↦ filen,
          d1 ↦ [ f11 ↦ file11,
                 f12 ↦ file12,
                 ...,
                 f1n ↦ file1n,
                 d11 ↦ [ f111 ↦ file111,
                         f112 ↦ file112,
                         ...,
                         f11n ↦ file11m,
                         d111 ↦ dir111,
                         d112 ↦ dir112,
                         ...,
                         d11n ↦ dir11n ]
                 d12 ↦ dir12,
                 d13 ↦ dir13,
                 ...,
                 d1o ↦ dir1o
                ],
```

[12] In writing 'abstract' we mean to say that we do not here bother about what a file nor what an identifier is.

```
  d2 ↦ dir1,
  d3 ↦ dir3,
  ...,
  dp ↦ dirp
]
```

63. A file or directory path name is a sequence of zero, one or more, i.e., n, identifiers of which the firs $n-1$ are directory identifiers and the last is either a directory or a file identifier.

The sequences

⟨d1,d11,f111⟩ and ⟨d1,d11,d112⟩

are path names of the directory example above and identify file file111, respectively directory dir112.

```
type
63.  DPN = (DI|FI)*
axiom
63.  ∀ dpn:DPN • len dpn > 0 ⇒ ∀ i:Nat • i ∈ inds dpn ∧ i<len dpn  ⇒ is_DI(dp(i))
```

64. The function end_paths generates all the paths (to empty sub-directories and files) of a directory.
65. If the directory is empty the singleton set of one empty path is yielded.
66. If the directory immediately contains a file, then the singleton path of that file's identifier is yielded together with the paths of the remaining directory entries.
67. If the directory immediately contains a sub-directory then the singleton path of that directory's identifier concatenated to all the paths of the sub-directory is yielded together with the paths of remaining directory.

```
value
64.  end_paths: D → DPN-set
64.  end_paths(d) ≡
64.    case d of
65.      [ ] → {⟨⟩},
66.      [ mkFI(i)↦f ] ∪ d′ → { ⟨mkFI(i)⟩ } ∪ end_paths(d′),
67.      [ mkDI(i)↦d ] ∪ d′ → { ⟨mkDI(i)⟩⌢p | p:P • p ∈ end_paths(d) } ∪ end_paths(d′)
64.    end
```

68. The function paths generates all the paths (to all sub-directories and files) of a directory.

    We express that by applying the function end_paths and "taking" all prefixes of the yielded paths.

```
value
68. paths: D → DPN-set
68. paths(d) ≡ { prefix(p)(i) | p>:P • p ∈ end_paths(d) ∧ i:Nat • i ∈ inds(p) }

68. prefix: P × Nat → P-set
68. prefix(p,n) ≡ { ⟨ p(i) | i:Nat • 1 ≤ i < j ⟩ | j:Nat • 1≤j≤n } pre: n ≤ len p
```

69. Given an end path of a directory and that directory, read_entry yields the designated entry, whether a file or an empty directory.
70. If the path
71. is a singleton path of just a file or an empty directory identifier then that file, respectively empty sub-directory is yielded,
72. else the path has a directory identifier head and and tail path and the desired entry is yielded from sub-directory of the head identifier with the tail path.
73. The "empty directory" requirement is expressed in the pre condition that the argument path must be an end path.

```
69. read_entry: P → D → (D|F)
69. read_entry(p)(d) ≡
70.     case p of
71.         ⟨i⟩ → d(i),
72.         ⟨mkDI(i)⟩^p′ → read_entry(p′)(d(mkDI(i)))
69.     end
73. pre: p ∈ end_paths(d)
```

Let d denote a directory, p a path in d, di a directory identifier, and fi and fi' file identifiers.

File directories, d, are subject to the following additional operations:

74. create_file_dir_sys(),
75. insert_dir(p,di)(d),
79. insert_file(p,(fi,file))(d),
82. copy_file(p,fi)(p′,fi′)(d),
87. update_file(p)(file), and
91. delete_entry(p)(d)

Observe that the operations deal only with "smallest" file system entities: empty directories and files.

74. Creating a file directory system yields an empty directory, [].

```
value
74. create_file_dir_sys: Unit → D
74. create_file_dir_sys() ≡ [ ]
```

75. **Inserting an empty directory**, named di, in a file directory system d, at position (i.e., path) p,

76. yields a new file directory system d',
77. all of whose paths, except for the new path: p'=p⌢⟨di⟩, are those of d with path p' being a new path designating [] !
78. This latter is expressed by the predicate +++.

value
75. insert_dir: Path × DI → D → D
76. insert_dir(p,di)(d) **as** d′
75. **pre**: p⌢⟨di⟩ ∉ end_paths(d)
77. **post**: unchanged(p,di,d) ∧ changed(p,di,d)

77. unchanged: P × DI × D → **Bool**
77. unchanged(p,di,d) ≡
77. ∀ p:P • p ∈ end_paths(d) ⇒ read_entry(p)(d) = read_entry(p)(d′)
77. changed: P × DI × D → **Bool**
77. changed(p,di,d) ≡
77. paths(d′) = paths(d) ∪ {p⌢⟨di⟩} ∧ read_entry(p⌢⟨di⟩)(d′)=[ ]

79. **Inserting a file**, f, named fi, in a file directory system d, at position (i.e., path) p,
80. yields a new file directory system d',
81. which is unchanged wrt. paths of directory d, but changed (only) wrt. the new path p concatenated with ⟨fi⟩

value
79. insert_file: P × (FI×F) → D → D
79. insert_file(p,(fi,file))(d) **as** d′
80. **pre**: p⌢⟨fi⟩ ∉ end_paths(d)
81. **post**: unchanged(p,fi,d) ∧ changed(p,di,d)

81. unchanged: P × FI × D × D → **Bool**
81. unchanged(p,fi,d,d′) ≡
81. ∀ p′:P • p′ ∈ end_paths(d) • read_entry(p′)(d) = read_entry(p′)(d′)
81. changed: P × (FI × F) × D × D → **Bool**
81. changed(p,(fi,f),d,d′) ≡
81. paths(d′) = paths(d) ∪ {p⌢⟨fi⟩} ∧ read_entry(p⌢⟨fi⟩)(d′)=f

Note that the unchanged and changed predicates are so-called "overloaded", i.e., same name but two distinct function types.

82. **Copying** a file from one "end location" in a file directory, d, to another such yields a changed file directory, d',
83. provided the argument "from" file is an end path of d, the directory path of the new "location" is also an end path of d, and the suggested end path to the copied file is not an end path of d,

84. where all end paths of d and d' identify the same files, respectively empty directories, and
85. the paths of p' are exactly those of p augmented by just the new end path to fi', and
86. where the files to be copied and copied are the same.

value
82. copy_file: (P×FI)×(P×FI) → D → D
82. copy_file((p,fi),(p′,fi′))(d) **as** d′
83. **pre**: p⌢⟨fi⟩ ∈ end_paths(d) ∧ p′ ∈ end_paths(d) ∧ p′⌢⟨fi⟩ ∉ end_paths(d)
84. **post**: unchanged(p,fi,d,d′) ∧
85. changed(p,(fi′,read_entry(p⌢⟨fi′⟩)(d))d,d′) ∧
86. read_entry(p⌢⟨fi⟩)(d) = read_entry(p⌢⟨fi⟩)(d′) = read_entry(p′⌢⟨fi′⟩)(d′)

87. **Updating** a file is to replace an identified file with a given, i.e., an argument file, thus resulting in a new file directory,
88. provided that the argument path is an end path of d and designates a file, and
89. where all the paths of the argument and the result file directory are the same, but
90. the file at designation p is now the argument, i.e., new file.

value
87. update_file: P × F → D → D
87. update_file(p,f)(d) **as** d′
88. **pre**: p ∈ end_paths(d) ∧ is_FI(p(**len** p))
89. **post**: paths(d) = paths(d′)
90. ∧ read_entry(p)(d′) = f

91. **Deleting** an entry, a file or an empty directory from a file directory, results in a new file directory,
92. provided the argument end path is in the argument file directory,
93. but not in the result file directory.

value
91. delete_entry: P → D → D
91. delete_entry(p)(d) **as** d′
92. **pre**: p ∈ end_paths(d)
93. **post**: end_paths(d′) = end_paths(d) \ {p}

• • •

We kindly ask the reader to consider the following:

- What we have modeled here can be seen as a language ! Yes, a "programming language" with which file directories can be established and manipulated. For this "systems programming language" fragment we have defined the syntax of "data types" and commands, and the defined the semantics of the "system commands" whose interpretation effects changes.
- Yes, the reader is kindly asked to observe how a "whole little theory", i.e., an *algebra*, of file directories have been carefully "unfolded:": That algebra consists of the *values* whose types are defined in Items 59– 63 on page 18, and the *operations* defined in Items 64 on page 18 through 93 on the preceding page.
- Finally, to reemphasize from the above, please observe two "technical issues:
  - That the development of this algebra has been in small steps.
  - And that the file directory operations have been defined in terms of their **pre/post** conditions, i.e., properties.

B Index

There are 59 indexed terms.