Double-entry Bookkeeping

Dines Bjørner[*]
Technical University of Denmark
Fredsvej 11, DK-2840 Holte, Danmark
E–Mail: bjorner@gmail.com, URL: www.imm.dtu.dk/˜db

April 16, 2024, 09:13

Abstract

We stepwise unfold a formal model of a double-entry bookkeeping "system".

Contents

[*]This report is under copyright protection: © Dines Bjørner, April 5, 2024

1 Introduction

1.1 What is This All About

We shall present a description of certain aspects of double-entry bookkeeping.[1] The description, in Sects. 3 and 3, focus on the "classical" issue of single- and double-entry bookkeeping. Whereas the description, in Sect. 5, focus on the domain modelling, that is, of embedding bookkeeping in models of such domains as road-pricing, shipping, retailing, manufacturing, etc. We refer to the books [6, 11] for introductions to domain modelling, and to the `Internet` document [9, *Domain Models – A Compendium*] for a compendium on some 15 [such] domains.[2] Double-entry bookkeeping, per se, is not [really] a domain issue. But its relation to domains is obvious !

1.2 A Background – A Context

There are two issues at play here.

1.2.1 Background

The working-out of this model of *double-entry bookkeeping* takes place on/in the following background.

1.2.1.1 Actual Double-entry Bookkeeping Systems.

Having first learned basic skills of *double-entry bookkeeping* and passed an examn during my MSc studies, 1956–1962. Having realized that *double-entry bookkeeping* represents an example of *intentional pull*, in recent years, cf. Sect. 5.6 of [11], 2020. Having a neighbour, "up the road", who has made a first fortune on *double-entry bookkeeping* software. But, having "studied" commercial, on the market *double-entry bookkeeping* software packages[3], never been quiet content with their explanation of these software systems.

1.2.1.2 Formal Software Development.

Since 1973, i.e., since my work at the IBM Vienna Laboratory, Austria, it has been clear to me that programs, their specification, and hence also now, domain descriptions and requirements prescriptions are mathematical object. And that the development of software can, and, to me, thus should be orderly developed: in phases from domain descriptions via requirements prescriptions to software and its code. All this is presented in [5, 6, 11].

1.2.2 Context

The Triptych Dogma

In order to *specify* **software**,
we must understand its requirements.
In order to *prescribe* **requirements**,
we must understand the **domain**.
So we must **study, analyze** and **describe** domains.

[1] https://en.wikipedia.org/wiki/Double-entry_bookkeeping
[2] It is the intention, eventually, to include this document's model of double-entry bookkeeping into that compendium.
[3] – but, it must be said: never personally used such software

The specific context in which this report, on what may seem a rather "low-level" topic, is then conceived is this. First: the above, the **The Triptych Dogma**. Then fact that each human artifact domain — such as those described in [9] — somehow or other include a [double-entry] bookkeeping element.

1.3 A Caveat

The present, Spring 2024, report is a torso. It sketches while also presenting the essential facets: the updating of double-entry bookkeeping debit/credit and asset/liability accounts. We leave it to the reader to complete possibly "dangling" descriptions: narratives and formalizations; to tie the various description elements together, and "embed" the result in a specific [road pricing, container shipping, retailer, banking, or pipeline domain.

1.4 Structure of Report

- In Sect. 2 we present, mostly from/courtesy `Wikipadia`, a vocabulary of terms relevant to bookkeeping.
- Section 3 then presents, in the style of [5, *Software Engineering, vols. 1–3* 2005/2006] a series of from very simple to reasonably realistic single-entry bookkeeping models.
- Section 4 then "generalizes" this to a double-entry bookkeeping model.
- Section 5 finally "embeds" the double-entry bookkeeping model into a model f the domain of accountancy.

2 Vocabulary

I expect to insert more term explanations.

- **Account:** In bookkeeping, an account refers to assets, liabilities, income, expenses, and equity, as represented by individual ledger pages, to which changes in value are chronologically recorded with debit and credit entries. These entries, referred to as postings, become part of a book of final entry or ledger. Examples of common financial accounts are sales, accounts receivable, mortgages, loans, PP&E (Property, Plant, and Equipment), common stock, sales, services, wages and payroll.

  A chart of accounts provides a listing of all financial accounts used by particular business, organization, or government agency.

  The system of recording, verifying, and reporting such information is called accounting. Practitioners of accounting are called accountants.

- **Asset:** An asset is any resource owned or controlled by a business or an economic entity. It is anything (tangible or intangible) that can be used to produce positive economic value. Assets represent value of ownership that can be converted into cash (although cash itself is also considered an asset). The balance sheet of a firm records the monetary value of the assets owned by that firm. It covers money and other valuables belonging to an individual or to a business.[

  Assets can be grouped into two major classes: tangible assets and intangible assets. Tangible assets contain various subclasses, including current assets and fixed assets. Current assets include cash, inventory, accounts receivable, while fixed assets include land, buildings and equipment. Intangible assets are non-physical resources and rights that have a value to the firm because they give the firm an advantage in the marketplace. Intangible assets include goodwill, intellectual property (such as copyrights, trademarks, patents, computer programs), and financial assets, including financial investments, bonds, and companies' shares.

IFRS (International Financial Reporting Standards), the most widely used financial reporting system, defines: "An asset is a present economic resource controlled by the entity as a result of past events. An economic resource is a right that has the potential to produce economic benefits."

- **Audit:** An audit is an *independent examination of financial information of any entity, whether profit oriented or not, irrespective of its size or legal form when such an examination is conducted with a view to express an opinion thereon*. Auditing also attempts to ensure that the books of accounts are properly maintained by the concern as required by law. Auditors consider the propositions before them, obtain evidence, roll forward prior year working papers, and evaluate the propositions in their auditing report.

  Audits provide third-party assurance to various stakeholders that the subject matter is free from material misstatement The term is most frequently applied to audits of the financial information relating to a legal person. Other commonly audited areas include: secretarial and compliance, internal controls, quality management, project management, water management, and energy conservation. As a result of an audit, stakeholders may evaluate and improve the effectiveness of risk management, control, and governance over the subject matter.

- **Auditor:** An auditor is a person or a firm appointed by a company to execute an audit. To act as an auditor, a person should be certified by the regulatory authority of accounting and auditing or possess certain specified qualifications. Generally, to act as an external auditor of the company, a person should have a certificate of practice from the regulatory authority.

- **Balance:** In banking and accounting, the balance is the amount of money owed (or due) on an account.

  In bookkeeping, "balance" is the difference between the sum of debit entries and the sum of credit entries entered into an account during a financial period. When total debits exceed the total credits, the account indicates a debit balance. The opposite is true when the total credit exceeds total debits, the account indicates a credit balance. If the debit/credit totals are equal, the balances are considered zeroed out. In an accounting period, "balance" reflects the net value of assets and liabilities to better understand balance in the accounting equation.

- **Credits and Debits:** Credits and debits in double-entry bookkeeping are entries made in account ledgers to record changes in value resulting from business transactions. A debit entry in an account represents a transfer of value to that account, and a credit entry represents a transfer from the account. Each transaction transfers value from credited accounts to debited accounts. For example, a tenant who writes a rent cheque to a landlord would enter a credit for the bank account on which the cheque is drawn, and a debit in a rent expense account. Similarly, the landlord would enter a credit in the rent income account associated with the tenant and a debit for the bank account where the cheque is deposited.

  Debits and credits are traditionally distinguished by writing the transfer amounts in separate columns of an account book. This practice simplified the manual calculation of net balances before the introduction of computers; each column was added separately, and then the smaller total was subtracted from the larger. Alternately, debits and credits can be listed in one column, indicating debits with the suffix ”Dr” or writing them plain, and indicating credits with the suffix ”Cr” or a minus sign. Debits and credits do not, however, correspond in a fixed way to positive and negative numbers. Instead the correspondence depends on the normal balance convention of the particular account.

- **Double-entry accounting:** See Double-entry bookkeeping.
- **Double-entry bookkeeping:** Double-entry bookkeeping, also known as double-entry accounting, is a method of bookkeeping that relies on a two-sided accounting entry to maintain financial information. Every entry to an account requires a corresponding and opposite entry to a different account. The double-entry system has two equal and corresponding sides, known as debit and credit; this is based on the fundamental accounting principle that for every debit, there must be an equal and opposite credit. A transaction in double-entry bookkeeping always affects at least two accounts, always includes at least one debit and one credit, and always has total debits and total credits that are equal.

  A Complete Transaction: In our model “the two sides” are instead modelled as a pair of pairs: A *debit/credit* pair and an *asset/liability* pair. Thus a “completed” transaction[4] in our double-entry bookkeeping should always affects at least two accounts, always includes a *debit/credit* and an *asset/liability*, and always has total *debit/credits* and total *asset/liability* that should be equal.

- **Equity:** Ownership of assets that have liabilities attached to them:
  - **Stock:** equity based on original contributions of cash or other value to a business.
  - **Home equity:** the difference between the market value and unpaid mortgage balance on a home.
  - **Private equity:** stock in a privately held company.
  - **Equity Method:** Equity method in accounting is the process of treating investments in associate companies. Equity accounting is usually applied where an investor entity holds 2050% of the voting stock of the associate company, and therefore has significant influence on the latter’s management. Under International Financial Reporting Standards, equity method is also required in accounting for joint ventures.[1] The investor records such investments as an asset on its balance sheet. The investor’s proportional share of the associate company’s net income increases the investment (and a net loss decreases the investment), and proportional payments of dividends decrease it. In the investors income statement Equity accounting may also be appropriate where the investor has a smaller interest, depending on the nature of the actual relationship between the investor and investee. Control of the investee, usually through ownership of more than 50% of voting stock, results in recognition of a subsidiary, whose financial statements must be consolidated with the parent’s. The ownership of less than 20% creates an investment position, carried at historic book or fair market value (if available for sale or held for trading) in the investor’s balance sheet.[5]

[4]By a “complete” transaction we shall understand a set of two or more *writes* (*updates*): a *debit/credit* account update and one or more *asset/liability* account updates – cf. Sect. 5.2.1 on page 24.

[5]https://ifrscommunity.com/knowledge-base/equity-method/

- **Ledger:**A ledger[1] is a book or collection of accounts in which accounting transactions are recorded. Each account has: (1) an opening or brought-forward balance; (2) a list of transactions, each recorded as either a debit or credit in separate columns (usually with a counter-entry on another page) and (3) an ending or closing, or carry-forward, balance.

- **Liability:** Liability: a current obligation of an entity arising from past transactions or events.

  In accounting, **contingent liabilities** are liabilities that may be incurred by an entity depending on the outcome of an uncertain future event[1] such as the outcome of a pending lawsuit. These liabilities are not recorded in a company's accounts and shown in the balance sheet when both probable and reasonably estimable as 'contingency' or 'worst case' financial outcome. A footnote to the balance sheet may describe the nature and extent of the contingent liabilities. The likelihood of loss is described as probable, reasonably possible, or remote. The ability to estimate a loss is described as known, reasonably estimable, or not reasonably estimable. It may or may not occur.

  Current liability, or **short-term liabilities** are obligations that will be settled by current assets or by the creation of new current liabilities.

  Non-current, or **Long-term liabilities**, are liabilities with a future benefit over a certain period of time (e.g. longer than one year)

3 A Sequence of Models of Single-entry Bookkeeping

3.1 A Simplest Single Entry Model

The simplest possible accounting just records the *budget* and the *debit/credit balance*. There is no recording of the earnings and expenditure transactions.

3.1.1 A Formal Type Model

1. An simplest account is just a pair of a *budget* and what has been accumulated: *debit [income]* and *credit [expense]*.

2. The budget is a natural number of [currency] units allocated.

3. *Debit [income]* & *Credit [expense]* entry is an integer number of [currency] units that has been *earned* or *spent*.

type
1. ACCOUNT_0 = BUDGET_0 $\times$ DEB_CRE_0
2. BUDGET_0 = **Nat**
3. DEB_CRE_0 = **Int**

3.1.2 A Formal 'Semantics' Model.

There is, basically, no bookkeeping to be associated with this model. Expenses result in the debit/credit being lowered. Income result in the debit/credit being lowered. No record is made (i.e., "written down") of these "transactions.

4. There is an *account* value.
5. There are two kinds of transactions: expenses and incomes.
6. It's debit/credit element is being decreased by expenses.
7. And increased by income.

value
4. (budget,deb_cre):ACCOUNT_0
type
5. Transaction = Expense | Income
6. Expense = **Nat**
7. Income = **Nat**
value
6. expense: Expense → ACCOUNT_0 → ACCOUNT_0
6. expense(n)(budget)(deb_cre) ≡ (budget,deb_cre − n)
7. income: Income → ACCOUNT_0 → ACCOUNT_0
7. income(n)(budget)(deb_cre) ≡ (budget,deb_cre + n)

3.2 Two Simple Single Entry Semantic Type Models

3.2.1 Simple Account Lists

3.2.1.1 A Formal Model.

8. A simple *account* is a pair of an *debit [income]* and *credit [expense] accounts*.
9. *Debit [income] accounts* are *account triplets*
10. *Credit [expense] accounts* are *account triplets*
11. *Account triplets* are triplets of a *budget*, an *entry list* and the sum total of what has been *earned* or *spent*.[6]
12. A *budget* is as defined in Item 2 on the previous page.
13. An *entry list* is a list of entries.
14. An *entry*[7] is a triplet of a time-stamp, some [explanatory] text, and an *amount earned* or *spent*.
15. A *time* stamp is further unspecified.
16. The explanatory *text* is further unspecified.
17. The *amount* is a natural number of [currency] units that has been *earned* or *spent*.

The *simple account lists* model thus has both the income and the expense accounts being lists of time-stamped, text-explained transactions.

[6]The term 'earned' is used in connection with *income accounts*, and the term 'spent' in connection with *expense accounts*.

[7]An *entry* is the recorded evidence of a *transaction*. A *transaction* is an action, i.e., something that changes a state.

type
8. ACCOUNT_1 $=$ DEBIT_ACCOUNT_1 $\times$ CREDIT_ACCOUNT_1
9. DEBIT_ACCOUNT_1 $=$ ACCOUNT_TRIPLE_1
10. CREDIT_ACCOUNT_1 $=$ ACCOUNT_TRIPLE_1
11. ACCOUNT_TRIPLE_1 $=$ BUDGET_1 $\times$ s_entries:ENTRY_LIST_1 $\times$ s_amount:AMOUNT_1
12. BUDGET_1 $=$ BUDGET_0
13. ENTRY_LIST_1 $=$ ENTRY_1*
14. ENTRY_1 $=$ s_time:$\mathbb{TIME}$ $\times$ s_text:E_Text_1 $\times$ s_amount:AMOUNT_1
15. $\mathbb{TIME}$ $= ...$
16. E_Text_1 $= ...$
17. AMOUNT_1 $=$ **Nat**

3.2.1.2 Wellformedness.

18. *Entries* in a *list of entries* are ordered *time*-wise in ascending order – with adjacent entries possibly have same time stamps.

19. The sum total of all *amounts* in an *account entry list* must equal the *spent* entry of the *account*.

axiom
18. $\forall$ el:ENTRIES_1 $\bullet$ $\forall$ i,j:**Nat** $\bullet$ $\{$i,j$\}\subseteq$**inds** el $\wedge$ i$<$j $\equiv$ s_time(el(i))$\leq$s_time(el(j))

19. $\forall$ (inc_acct_1,eps_acct_1):ACCOUNT_1 $\bullet$
19. **let** total_inc $=$ s_amount(inc_acct_1), total_exp $=$ s_amount(exp_acct_1) **in**
19. **let** income $=$ sum(s_entries(inc_acct_1)), expenses $=$ sum(s_entries(exp_acct_1)) **in**
19. total_inc$=$ income $\wedge$ total_exp $=$ expenses **end end**

value
19.‘ sum: ENTRIES_1 $\rightarrow$ Amount_1
19.$'$ sum(el) $\equiv$ **case** el **of** $\langle\rangle \rightarrow 0$, $\langle$(_,_,a)$\rangle$^el$'$ $\rightarrow$ a $+$ sum_amounts(el$'$) **end**

3.2.2 Simple Account Maps.

3.2.2.1 A Formal Model.

The *simple account map* model introduces separate account name lists of time-stamped, text-explained transactions.

20. [ι 8 π 8] A simple *account* is a pair of an *debit* and *credit accounts*.

21. [ι 9 π 8] *Debit accounts* are *account triplets*

22. [ι 10 π 8] *credit accounts* are *account triplets*

23. [*] *Account triplets* are triplets of a *budget*, an *entry map* and the sum total of what has been *earned* or *spent*.

24. [ι 12 π 8] A *budget* is as defined in Item 2 on page 7.

25. [*] An *entry map* is a map of *account named entry lists*.
26. [ι 8 π 8] An *entry list* is a list of entries.
27. [ι 13 π 8] An *entry* is a triplet of a time-stamp, some [explanatory] text, and an *amount earned* or *spent*.
28. [ι 14 π 8] A *time* stamp is further unspecified.
29. [ι 15 π 8] The explanatory *text* is further unspecified.
30. [ι 16 π 8] The *amount* is a natural number of [currency] units that has been *earned* or *spent*.

The [ι#π#] refers to ι*tem*/π*age* entries. The [*]-marked items represent the changes wrt. the simple account lists model 3.2.1 on page 8.

type
20. ACCOUNT_1 = DEBIT_ACCOUNT_1 × CREDIT_ACCOUNT_1
21. DEBIT_ACCOUNT_1 = ACCOUNT_TRIPLE_1
22. CREDIT_ACCOUNT_1 = ACCOUNT_TRIPLE_1
23. ACCOUNT_TRIPLE_1 = BUDGET_1 × s_entries:ENTRY_MAP_1 × s_amount:AMOUNT_1
24. BUDGET_1 = BUDGET_0
25. ENTRY_MAP_1 = Acc_Name $\overrightarrow{m}$ ENTRY_LIST_1
26. ENTRY_LIST_1 = ENTRY_1*
27. ENTRY_1 = s_time:$\mathbb{TIME}$ × s_text:E_Text_1 × s_amount:AMOUNT_1
28. $\mathbb{TIME}$ = ...
29. E_Text_1 = ...
30. AMOUNT_1 = **Int**

3.2.2.2 Wellformedness.

31. [ι 18 π 9] *Entries* in a *list of entries* are ordered *time*-wise in ascending order – with adjacent entries possibly have same time stamps.
32. [ι 19 π 9] The sum total of all *amounts* in an *account entry list* must equal the *earned* or *spent* entry of the *account.*

axiom
18. ∀ el:ENTRIY_LIST_1 • ∀ i,j:**Nat** • {i,j}⊆**inds** el ∧ i<j ≡ s_time(el(i))≤s_time(el(j))

19. ∀ (inc_acct_1,eps_acct_1):ACCOUNT_1 •
19. **let** total_inc = s_amount(inc_acct_1), total_exp = s_amount(exp_acct_1) **in**
19. **let** income = sum(s_entries(inc_acct_1)), expenses = sum(s_entries(exp_acct_1)) **in**
19. total_inc= income ∧ total_exp = expenses **end end**

value
sum: ENTRIES_1 → Amount_1
sum(el) ≡ **case** el **of** ⟨⟩ → 0, ⟨(_,_,a)⟩⌢el′ → a + sum_amounts(el′) **end**

3.3 A General Single Entry Model

3.3.1 A Formal Model

3.3.1.1 A Type Model.

33. [ι 8 π 8] An *account* is a pair of an *debit [income]* and *credit [expense] accounts*.
34. [ι 9 π 8] *Debit [Income] accounts* are *account triplets*
35. [ι 10 π 8] *Debit [Expense accounts* are *account triplets*
36. [ι 23 π 9] *Account triplets* are triplets of a *budget*, an *entries component* and a sum total of what has been *earned* or *spent.*
37. [ι 12 π 8] A *budget* is as defined in Item 2 on page 7.
38. [*] An *entries component* is a map from [sub-]*account names* to either an *entry list* or an *entry map.*
39. [ι 8 π 8] An *entry list* is a triple of a *budget*, a list of simple entries, and a sum total of what has been *earned* or *spent.*
40. [*] An *entry map* is a triplet of a *budget*, a map, and a sum total of what has been *earned* or *spent.*
41. The *map* is from [sub]*account names* to *account triplets*
42. [ι 8 π 8] A *simple entry* is a triplet of a time-stamp, some [explanatory] text, and an *amount spent*
43. [ι 15 π 8] A *time* stamp is further unspecified.
44. [ι 16 π 8] The explanatory *text* is further unspecified.
45. [ι 17 π 8] The *amount* is a natural number of [currency] units that has been *earned* or *spent.*

The [*]-marked items represent the changes wrt. the simple account lists model 3.2.2 on page 9.

type
33. ACCOUNT_2 = DEBIT_ACCOUNT_2 $\times$ CREDIT_ACCOUNT_2
34. DEBIT_ACCOUNT_2 = ACCOUNT_TRIPLET_2
35. CREDIT_ACCOUNT_2 = ACCOUNT_TRIPLET_2
36. ACCOUNT_TRIPLET_2 = BUDGET_2 $\times$ s_entries:ENTRIES_2 $\times$ s_total:AMOUNT_2
37. BUDGET_2 = BUDGET_0
38. [*] ENTRIES_2 = Account_Name $\overrightarrow{m}$ s_entries:(ENTRY_LIST_2 | ENTRY_MAP_2
39. ENTRY_LIST_2 = BUDGET_2 $\times$ s_entry_list:ENTRY_2* $\times$ s_sub_total:AMOUNT_2
40. [*] ENTRY_MAP_2 = s_budget:BUDGET_2 $\times$ MAP_2 $\times$ s_total:AMOUNT_2
41. [*] MAP_2 = Acc_Name $\overrightarrow{m}$ ACCOUNT_TRIPLET_2
42. ENTRY_2 = s_time:$\mathbb{TIME}$ $\times$ s_text:E_Text_2 $\times$ s_amount:AMOUNT_2
43. $\mathbb{TIME}$ = ...
44. E_Text_2 = ...
45. AMOUNT_2 = **Nat**

Figure 1 intends to graphically + textually illustrate a specific [ACCOUNT_2] account.

```
        ┌                                                                   ┐
        │ _->(#,<(_,_,*),(_,_,*),...,(_,_,*)>,*),                           │   [1]
        │ _->(#,<(_,_,*),(_,_,*),...,(_,_,*)>,*),                           │   [2]
        │        ┌ _->(#,<(_,_,*),...,(_,_,*)>,*),┐                         │   [3]
  (#,   │ _->(#, │ ... ,                           │  ,*),                  │ ,*)  [4]
        │        └ _->(#,<(_,_,*),...,(_,_,*)>,*) ┘                         │   [5]
        │ ... ,                                                             │
        │ _->(#,<(_,_,*),(_,_,*),...,(_,_,*)>,*)                            │   [6]
        └                                                                   ┘
```

Figure 1: **An Account.**

0. Slanted, bracketed numerals, e.g., *[1,3,5]*, refer to text lines of the figure.
1. Texts after →s in lines *[1,2,6]* stand 2. for ENTRY_LIST_2s.
3. Text of leftmost →s in line *[4]* stands for an ENTRY_MAP_2.
4. Text lines *[3,5]* within the ENTRY_MAP_2 stand for ENTRY_LIST_2s.
5. The '#'s *[1,2,3,4,4,5,6]* stands for a *budget*s.
6. The '_'s immediately to the right of the opening square brackets *[1,2,3,4,5,6]* stand for *account names*.
7. The '_'s right after the '→ parentheses '_'s *[1,2,3,4,4,6]* stands for *budgets*.
8. All other '_'s stands for *time*, resp. *texts*.
9.The '*'s stands for *amounts*.

Constraints:

10. The first three '*'s in the first two and the last text lines *[1,2,6]* must sum up to the last '*' in those lines.
11. Similarly for the first two '*'s in the 3rd and the 4th *[3,4]* text lines: they must sum up to the last '*' in those lines.
12. The last '*'s in the arrowed (→) lines *[1,2,3,5,6]* must sum up to the last two '*', respectively, in the rightmost text lines *[4]*.
13. The above and below constraints are formalized in Sect. **??** on page ??.
14. Similar constraints apply to *budget* entries:
    15. The sum of the first #s in line *[1,2,6]* and the second # in line *[4]* must equal the first # in line *[1]*.
    16. The sum of the first #s in lines *[3,5]* must equal the second # in line *[4]*.

3.3.1.2 Access Paths.

We define an auxiliary function: access_paths. An *access path* is a sequence of *account names* such that the first element of the path applies to a [root] account and selects either an entry list or an entry map of either an income or an expense account. And the first of a possible tail of the path accesses an entry list or an entry map of the selected former such entry. Et cetera. Thus *debit* and *credit accounts* define each their sets of *access paths*.

46. An *access path* is a sequence of *account names*.

47. access_paths applies to either *debit* and *credit accounts* and yields a set of *access path*s.

48. Since *debit* and *credit accounts* are *account triplets* one can select their *entries* component.

    [38. An *entries component* is a map to either *entry_lists* or *entry_maps*.]

49. If *entry_lists*, then a set of *singleton access paths*, ⟨an⟩, for each of the account names of the *entry_lists* is yielded.

50. If *entry_maps*, then a set, map_access_paths(map), of all the *access paths* reachable from, and including the *map access path*, ⟨an⟩ is yielded.

```
type
46. Acces_Path = Account_Name*
value
47. access_paths: (DEBIT_ACCOUNT_2|CREDIT_ACCOUNT_2) → Access_Path-set
47. access_paths(acc_trip) ≡
48.    let entries = s_entries(acc_trip) in
49.    is_ENTRY_LIST_2(entries(an))
49.        → { ⟨an⟩ | an:Acc_Name • an ∈ dom entries }
50.    is_ENTRY_MAP_2(entries(an))
50.        → ∪ {map_access_paths(entries(an)) | an:Acc_Name • an ∈ dom entries }
47.    end
```

51. The map_access_paths function applies to map:ENTRY_MAP_2s and yields a set of *access paths*.

    [36 [ι 23 π 9]. Each map range element is an *account triplet* and these are triplets of a *budget*, an *entries component* and the sum total of what has been *earned* or *spent*.]

52. So for each *account name*, an, in the map that *account name* is prefixed each of the the *access paths* from that *account triple*.

```
value
51. map_access_paths: ENTRY_MAP_2 → Access_Path-set
51. map_access_paths(entry_map) ≡
52.    { ⟨an⟩⌢ap | an:Account_Name • an ∈ dom entry_map,
52.                 ap:Access_Path • ap ∈ access_paths(entry_map(an)) }
```

3.3.1.3 Well-formed Access Paths.

We aim at expressing that all *account names* are distinct. To to so we build up that well-formedness criterion in two stages.

53. The *account names* of any access path are distinct.

```
axiom
53. ∀ ap:Access_Path • card elems ap = le ap
```

54. Any two distinct *access paths*, if they share an *account name* then it is the first element of these *access paths*.

axiom

54. $\forall$ ap,ap$'$:Access_Path •
54. **elems** ap $\cap$ **elems elems** ap$'$ $\neq$ {}
54. $\Rightarrow$ **hd** ap=**hd** ap$'$ $\wedge$ {**hd** ap}=**elems** ap $\cap$ **elems elems** ap$'$

We can choose to let the *debit* and the *credit accounts* be "identically structured", that is have exactly the same access paths:

55. The *debit* and the *credit accounts* have exactly the same *access paths:*

55. $\forall$ (inc_acc,exp_acc):ACCOUNT_2 • access_paths(inc_acc) = access_paths(exp_acc)

Or we could choose otherwise, cf. Sect. 3.3.1.5. That should suffice. [Prove that !]

3.3.1.4 Account Access.

An *access path* "points" to an *entry list*. A proper prefix, i.e., if the *access path* is of **len**gth 2 or more, "points" to an *entry map*.

56. The function access takes as `argument` an *access path* or a proper prefix thereof and `applies` to either an *debit* or an *credit* account and `yields` either an *entry list* or an *entry map*.
57. If the *access path*
58. is of length 1, i.e., $\langle$an$\rangle$, then *select* the *entries* of the *account* as the result.
59. If the *access path* is of length more than 1, i.e., $\langle$an$\rangle$$\widehat{\ }ap'$, then access the account obtained from *access path* $\langle$an$\rangle$ with *access path* ap$'$.

value

```
56. access: Access_Path×(DEBIT_ACCOUNT_2|CREDIT_ACCOUNT_2)
56.     → (ENTRY_LIST_2|ENTRY_MAP_2)
56. access(ap,account) ≡
57.     case ap of
58.         ⟨an⟩ → s_entries(account),
59.         ⟨an⟩⁀ap′ → access(ap′,s_entries(account))
56.     end
56.     pre ap ∈ access_paths(account) ∨ ∃ ap′• ∈ access_paths(account) ∧ ap ∈ prefix_paths(ap′)

    prefix_paths: Access_Path → Access_Path-set
    prefix_paths(ap) ≡ { ⟨an(i)| i:Nat • 1≤i≤len ap ⟩ }
```

3.3.1.5 Summary Expense Accounts.

There are at least two other possibilities of distinguishing between income and expenses.

3.3.1.5.1 Paired Debit/Credit Entries

- The ACCOUNT_2 model, cf. Item 33 on page 11,
  - has the ENTRY_LIST_2s cf. Item 39 on page 11,
  - be simple triplets:
  - ENTRY_LIST_2 = BUDGET_2 × s_entry_list:ENTRY_2* × s_total:AMOUNT_2.
- Instead we could avoid the distinction
  - at the top level of the ACCOUNT_2 model
  - between INCOME_ACCOUNT_2s and EXPENSE_ACCOUNT_2s.
    - Instead ACCOUNT_2s are now just ACCOUNT_TRIPLEs.
    - But ENTRY_LIST_2s now make the distinction between *debit* and *credit*:
    - BUDGET_2×s_entry_lists(s_inc:ENTRY_2*,s_exp:ENTRY_2*)×s_total:AMOUNT_2.

type
33. ACCOUNT_2 = ACCOUNT_TRIPLET_2
36. ACCOUNT_TRIPLET_2 = BUDGET_2×s_entries:ENTRIES_2×s_total:AMOUNT_2
37. BUDGET_2 = BUDGET_0
38. ENTRIES_2 = Account_Name $\overrightarrow{m}$ (ENTRY_LIST_2|ENTRY_MAP_2)
39. [*] ENTRY_LIST_2 = BUDGET_2×s_debit:ENTRY_2*,s_credit:ENTRY_2*×s_total:AMOUNT_2
40. [*] ENTRY_MAP_2 = s_budget:BUDGET_2 × MAP_2 × s_total:AMOUNT_2
41. [*] MAP_2 = Acc_Name $\overrightarrow{m}$ ACCOUNT_TRIPLET_2
42. ENTRY_2 = s_time:$\mathbb{TIME}$×s_text:E_Text_2×s_amount:AMOUNT_2
43. $\mathbb{TIME}$ = ...
44. E_Text_2 = ...
45. AMOUNT_2 = **Nat**

3.3.1.5.2 Summary Credit Entries

Instead of pairing, as in Sect. 3.3.1.5.1, *debit* and *credit* entries, one could summarize *expenses* in "earlier" entries, that is, in entries with whose *access path* is a prefix of the the *access path*, ap, to the *debit* entry, however with an account name ⟨an⟩, suffixed to ap,

We leave the formalization to the reader !

4 A Double-entry Bookkeeping Model

We present the *double-entry bookkeeping* as a pair of pairs ! That is: a pair of *debit/credit accounts* and a pair of *asset/liability accounts.*

4.1 A Type Model

Each of the pairs are type-structured as were the accounts in Sect. 3.3.1.1 on page 11.

4.1.1 Types

We repeat most of the type formulas from Sect. 3.3.1.1 on page 11.

60. *Double-entry Bookkeeping Accounts* are pairs of *debit/credit accounts* and *asset/liability accounts*.
61. *Asset/Liability Accounts* are pairs of *Asset Accounts* and *Liability Accounts*.
62. *Asset Accounts* are *account triplets*.
63. *Liability Accounts* are *account triplets*.

type
60. DBL_ENTRY_ACCOUNT = DC_ACCOUNT $\times$ AL_ACCOUNT
[ι 33 π 11]. DC_ACCOUNT = DEBIT_ACCOUNT $\times$ CREDIT_ACCOUNT
[ι 34 π 11]. DEBIT_ACCOUNT = ACCOUNT_TRIPLET
[ι 35 π 11]. CREDIT_ACCOUNT = ACCOUNT_TRIPLET
61. AL_ACCOUNT = ASSET_ACCOUNT $\times$ LIABILITY_ACCOUNT
62. ASSET_ACCOUNT = ACCOUNT_TRIPLET
63. LIABILITY_ACCOUNT = ACCOUNT_TRIPLET
[ι 36 π 11]. ACCOUNT_TRIPLET = BUDGET $\times$ s_entries:ENTRIES $\times$ s_total:AMOUNT
[ι 37 π 11]. BUDGET = **Nat**
[ι 38 π 11]. ENTRIES = Account_Name $\overrightarrow{m}$ s_entries:(ENTRY_LIST | ENTRY_MAP)
[ι 39 π 11]. ENTRY_LIST = BUDGET $\times$ s_entry_list:ENTRY* $\times$ s_sub_total:AMOUNT
[ι 40 π 11]. ENTRY_MAP = s_budget:BUDGET $\times$ MAP $\times$ s_total:AMOUNT
[ι 40 π 11]. MAP = Acc_Name $\overrightarrow{m}$ ACCOUNT_TRIPLET
[ι 42 π 11]. ENTRY = s_time:$\mathbb{TIME}$ $\times$ s_text:E_Text $\times$ s_amount:AMOUNT
[ι 43 π 11]. $\mathbb{TIME}$ = ...
[ι 44 π 11]. E_Text = ...
[ι 45 π 11]. AMOUNT = **Nat**

Please observe the recursion in formula [ι 41 π 11] "back to" formula [ι 36 π 11] above.

4.1.2 Wellformedness

We refer to Sects. 3.3.1.2 on page 12 and 3.3.1.3 on page 13 The signature of the function *access paths* need be adjusted:

4.1.2.1 Access Paths

4.1.2.1.1 Common Constraints

value
47.′ access_paths:
47.′ (DEBIT_ACCOUNT|CREDIT_ACCOUNT|ASSET_|LIABILITY_ACCOUNT)
47.′ $\rightarrow$ Access_Path-**set**

[ι 53 π 13] The *account names* of any access path are distinct.

axiom
[ι 53 π 13]. $\forall$ ap:Access_Path • **card elems** ap = **len** ap

[ι 54 π 14] Any two distinct *access paths*, if they share an *account name* then it is the first element of these *access paths*.

```
axiom
[ι 54 π 14].  ∀ ap,ap′:Access_Path •
[ι 54 π 14].        elems ap ∩ elems elems ap′ ≠ {}
[ι 54 π 14].            ⇒ hd ap=hd ap′ ∧ {hd ap}=elems ap ∩ elems elems ap′
```

[ι 55 π 14] The *debit* and the *credit accounts* have exactly the same *access paths*, informally:

```
[ι 55 π 14].  ∀ debit/asset_acc,credit/liability_acc:ACCOUNT
[ι 55 π 14].     • access_paths(deb_acc) = access_paths(cre_acc)
[ι 55 π 14].       ∧ access_paths(ass_acc) = access_paths(lia_acc)
```

4.1.2.1.2 Double-entry Constraints

64. The *access paths* of *debit/credit* and of *asset/liability* accounts are identical.[8]

```
64.  ∀ ((deb_acc,cre_acc),(ass_acc,lai_acc)):
64.        ((DEBET_ACCOUNT×CREDIT_ACCOUNT)×(ASSET_ACCOUNT×LIABILIY_ACCOUNT))
64.           • access_paths(deb_acc) = access_paths(cre_acc)
64.             ∧ access_paths(ass_acc) = access_paths(lia_acc)
```

65. The set of *account names* of *debit/credit* and of *asset/liability* accounts are distinct.[9]

66. We define the auxiliary function: account_names.

```
value
66.  account_names: ACCOUNT_TRIPLET → Acc_Name-set
66.  account_names(acc_trip) ≡
66.     let acc_pths = access_paths(s_entries(acc_trip)) in
66.     ∪ { ∪ { elems pth | pth:Acc_Path • pth ∈ acc_pths } }
66.     end
axiom
65.  ∀ ((deb_acc_trip,cre_acc_trip))((ass_acc_trip,lia_acc_trip))
65.     • (DEBIT_ACCOUNT×DEBIT_ACCOUNT)×(DEBIT_ACCOUNT×DEBIT_ACCOUNT)
65.          (account_names(deb_acc_trip) = account_names(cre_acc_trip)
65.         ∧  account_names(ass_acc_trip) = account_names(lia_acc_trip))
65.         ∧ (account_names(deb_acc_trip) ∪ account_names(cre_acc_trip))
65.           ∩ (account_names(ass_acc_trip) ∪ account_names(lia_acc_trip)) = {}
```

[8] Cf. 3.3.1.3 on page 13
[9] Cf. 3.3.1.3 on page 13

4.1.2.2 Budgets

We refer to lines [10–16] of the caption of Fig. 1 on page 12.

67. The *budget* of an ACCOUNT_TRIPLET must equal the summation of the *budget*s of the BUDGETs of the ENTRY_LIST or the *ENTRY_MAP*.

This constraint looks “innocent”, at first. But since it applies to recursively embedded ACCOUNT_TRIPLETs it is quite powerful. So we express it as a universal predicate over ACCOUNT_TRIPLETs rather than trying to figure out a recursively, first descending, then ascending, re-tracking, function. [Try formulate such a function !]

```
axiom [Budgets]
67. ∀ (b,e,_):ACCOUNT_TRIPLET • b = budget_sum(e)
value
67. budget_sum: ENTRIES → AMOUNT
67. budget_sum(e) ≡
67.     case e of
67.         [] → 0,
67.         [a↦elom]∪ e‘ → entry_sum(elom)+budget_sum(e‘)
67.     end

67.‘ entry_sum: (ENTRY_LIST|ENTRY_MAP) → AMOUNT
67.‘ entry_sum(elom) ≡
67.‘     is_ENTRY_LIST(elom) → list_sum(s_entry_list(elom)),
67.‘     is_ENTRY_MAP(elom) → map_sum(s_entry_list(elom))

67.“ list_sum: ENTRY_LIST → AMOUNT
67.“ list_sum(el) ≡ sum(el) [cf.[ι 19 π 9].′]

67.“‘ map_sum: ENTRY_MAP → AMOUNT
67.“‘ map_sum(em) ≡
67.“‘     case em of
67.“‘         [] → 0,
67.“‘         [a↦(b,_,_)]∪ em‘ → b + map_sum(em‘)
67.“‘     end
```

4.1.2.3 Amounts

68. The *amount* of an ACCOUNT_TRIPLET must equal the summation of the *amounts*s of the BUDGETs of the ENTRY_LIST or the *ENTRY_MAP*.

```
axiom [Amounts]
68. ∀ (_,e,a):ACCOUNT_TRIPLET • a = amount_sum(e)
value
68. amount_sum: ENTRIES → AMOUNT
68. amount_sum(e) ≡
```

```
68.     case e of
68.       [ ] → 0,
68.       [a↦elom]∪ e‘ → amount_entry_sum(elom)+amount_sum(e‘)
68.     end

68.‘  amount_entry_sum: (ENTRY_LIST|ENTRY_MAP) → AMOUNT
68.‘  amount_entry_sum(elom) ≡
68.‘      is_ENTRY_LIST(elom) → amount_list_sum(s_entry_list(elom)),
68.‘      is_ENTRY_MAP(elom) → amount_map_sum(s_entry_list(elom))

68.“  amount_list_sum: ENTRY_LIST → AMOUNT
68.“  amount_list_sum(el) ≡ sum(el) [cf.[ι 19 π 9].′]

68.“‘  amount_map_sum: ENTRY_MAP → AMOUNT
68.“‘  amount_map_sum(em) ≡
68.“‘      case em of
68.“‘         [ ] → 0,
68.“‘         [a↦(_,_,a)]∪ em‘ → a + amount_map_sum(em‘)
68.“‘      end
```

4.1.2.4 Balance

69. By a *balance* of DC_ACCOUNT or a AL_ACCOUNT

70. we shall mean the difference between their *budgets* and *amounts*.

```
value
69. balance: ACCOUNT_TRIPLET → Int
70. balance(budget,_,amount) ≡ budget − amount
```

4.1.2.5 Intentional Pull

71. The balances of the DC_ACCOUNT and the AL_ACCOUNT of a *double-entry bookkeeping* system must equal !

Well, there is no guarantee that the accounts balance ! Only *proper accountancy* and *audit* might secure that !

```
value
71. proper_accountancy: ENTRY_ACCOUNT → Bool
71. proper_accountancy(dc_acc,al_acc) ≡ balance(dc_acc)=balance(al_acc)
```

This constraint is the “hall-mark” of *double-entry bookkeeping* systems !

4.2 Transactions

4.2.1 View

72. To *view*, is to [screen] "display" an *account entry* of a *double-entry bookkeeping* system - given an *access path* to either a *debit/credit account* or an *access/liability account* for that system.

```
value
72.  view: DBL_ENTRY_ACCOUNT × (DCorAL × Access_Path) → (ENTRY_LIST | ENTRY_MAP)
type
72.  DCorAL = ″dc″ | ″al″
value
72.  view((dca,ala),(dcoral,ap)) ≡
72.      case dcoral of
72.          ″dc″→access(ap,dca), cf. [ι 56 π 14]
72.          ″al″→access(ap,ala)  cf. [ι 56 π 14]
72.      end
72.  pre: dcoral=″dc″→ap ∈ access_paths(dca),_→ap ∈ access_paths(ala)
```

4.2.2 Write

To *write* is to *insert a new entry* is an ENTRY_LIST, that is, at the end of the *view*ed entry.

Writes can occur to either a *debit/credit account* or to an *asset/liability* account. *Updating* a *debit/credit account* usually requires a corresponding one or more *updates* to the *asset/liability account*.

This is required in order to maintain the *intentional pull* of the *double-entry bookkeeping* system. Cf. Sect. 4.1.2.5 on the preceding page.

We model *writes* follows:

73. To *write* syntactically takes (i) an indication as to whether the update is to that of a debit/credit account or to an asset/liability account, (ii) an access path and (iii) the text and (iv) amount with which to update the accessed entry.

74. Semantically the *write* occurs in the context of a *double-entry bookkeeping* system and yields such a system.

75. We express the effect of a *write* to a *double-entry bookkeeping* system (dca,ala) **as** that of yielding a changed *double-entry bookkeeping* system (dca′,ala′).

76. The "difference" between (dca,ala) and (dca′,ala′) is expressed in the **where** predicate.

77. The *access paths* are unchanged.

78. A time, τ, is recorded.[10]

79. Either the *write* is to a *debit/credit account* or it is to an *asset/liability account*.

[10] **record**_$\mathbb{TIME}$() is a "built-in" primitive of the description language.

(a) If to a *debit/credit account* then the *asset/liability account* is unchanged.
(b) For all *accesses*, ap′,
(c) to the *debit/credit account* other than the prescribed (to be updated) entry,
(d) the entries are unchanged.
(e) For the accessed *entry list* their sub-entries differ as follows:
(f)
- the budget is unchanged;
- the entry list extended with suffix triplet of
  - the time stamp;
  - a text; and
  - an amount;
- and the entire entry list amount is adjusted accordingly.

80. A similar [**where**] predicate applies to *asset/liability accounts*

```
type
73.  Write :: mkWrite(DCorAL,Access_Path,E_Text,AMOUNT)
value
74.  write: Write → DBL_ENTRY_ACCOUNT → DBL_ENTRY_ACCOUNT
75.  write(dcoral,ap,txt,a)(dca,ala) as (dca′,ala′)
76.    where
77.      access_paths(dca)=access_paths(dca′) ∧ access_paths(ala)=access_paths(ala′)
78.    ∧ let τ = record_TIME() in
79.      case dcoral of
79a.       "dc" → ala′ = ala ∧
79b.         ∀ ap′ •
79c.           ap′ ∈ Access_Path(view((dca,_),("dc",ap)))\{ap}
79d.             ⇒ view((dca,_),("dc",ap′)) = view((dca,_),("dc",ap))
79e.         ∧ let (b,el,am) =view((dca,_),("dc",ap)), (b′,el′,am′) = view((dca,_),(dcoral,ap)) in
79f.           b=b′ ∧ el′=el⌢⟨(τ,txt,a)⟩ ∧ am′ = am + a end
80.       "al" → [ similarly ! ]
79.      end
78.      end
74.  pre: dcoral="dc"→ap ∈ access_paths(dcacc),_→ap ∈ access_paths(alacc)
```

The above model is inspired by the storage model – for such languages as `PL/I`, `Algol 68`, `CHILL` and `Ada` [13, 3, 1, 4] – put forward by Hans Bekič and Kurt Walk [2].

5 A Financial Management Sub-Domain

We refer to [10, *Domain Modelling*].

5.1 Endurants

5.1.1 External Qualities

81. From any domain, cf. [9, *Domain Models A Compendium*], we can, besides the "core" of the domain, observe:

82. its *management*.

From this *management* we can observe:

83. the double-entry bookkeeping system,
84. its *accountancy* and
85. its *audit[or]*.

From the *accountancy* we can observe:

86. its *chief accountant* and
87. a set of zero, one or more *accountant*s.

We leave the *audit[or]* further undefined.

```
type
81.  DOMAIN
82.  MGT
83.  DEBK = DBL_ENTRY_ACCOUNT
84.  ACCOUNTANCY
85.  AUDIT
86.  ACCOUNTANT
87.  CHIEF_ACCOUNTANT  = ACCOUNTANT
value
82.  obs_MGT: DOMAIN → MGT
83.  obs_DEBK: MGT → DEBK
84.  obs_ACCOUNTANCY: MGT → ACCOUNTANCY
85.  obs_AUDIT: MGT → AUDIT
86.  obs_CHIEF_ACCOUNTANT:  ACCOUNTANCY → CHIEF_ACCOUNTANT
87.  obs_ACCOUNTANTs:  ACCOUNTANCY → ACCOUNTANT-set
```

5.1.1.1 Endurant Values

For use in later descriptions we introduce some relevant endurant values.

88. There is given a *domain*.
89. From its *management*, *mgt*, we observe its *double-entry bookkeeping* system as a global value, *debk*.
90. From the *management* we can observe an *attribute*, the *debit/credit to asset/liability relation*, DB_AL_REL.
91. From the *management* we can observe observe the *accountancy*.
92. From the *accountancy* we can observe *chief_accountant*, and
93. the set of *accountants*.

```
value
88.  domain:DOMAIN
89.  mgt:MGT = obs_MGT(domain)
89.  (dc_acc,al_acc):DEBK = obs_DEBK(mgt)
91.  accountancy:ACCOUNTANCY = obs_ACCOUNTANCY(mgt)
92.  chief_accountant:CHIEF_ACCOUNTANT = obs_CHIEF_ACCOUNTANT(accountancy)
93.  accountants:ACCOUNTANTs = obs_ACCOUNTANTs(obs_(accountancy))
type
94.  DB_AL_REL = Access_Path →ₘ REL
```

5.1.2 Internal Qualities

5.1.2.1 Unique Identifiers

5.1.2.2 Mereologies

5.1.2.3 Attributes

5.1.2.3.1 Debit/Credit Accounts

pp:Debit Credit Accounts

5.1.2.3.2 Asset/Liability Accounts

5.1.2.3.3 Accountants

5.1.2.3.4 Attribute Constraints

94. The *debit/credit to asset/liability relation*, DB_AL_REL, maps *debit/credit access paths* to

95. a map, REL, from *access/liability access paths* to a rational lying properly between 0 and 1,

96. and such that these sum up to 1 !

```
type
95.  REL = Access_Path →ₘ Rat
value
90.  attr_DB_AL_REL: MGT → DB_AL_REL
90.  db_al_rel = attr_DB_AL_REL(mgt)
axiom [Proper Manageemnt]
94.  ∀ db_al_rel:DB_AL_REL • dom db_al_rel = ...
96.  ∀ rel:REL • dom rel = ... ∧ rng_rel_sum(rel)=1
value
96.  rng_rel_sum: REL → Rat
96.  rng_rel_sum(rel) ≡
96.     case rel of
96.        [] → 0,
96.        [ap↦r]+rel′ → r + rng_rel_sum(rel′)
96.     end
96.  pre: ∀ r:Rat • r ∈ rng rel ⇒ 0<r≤1
```

The idea behind the DB_AL_REL is explained in Sect. 5.2.1.

5.2 Perdurants

5.2.1 A Complete Transaction.

We refer to the *A Complete Transaction* comment on Page 6.

The idea behind the DB_AL_REL is the following: When a *debit* [or *credit*] entry is posted, for a certain amount, it should be followed by one (or more) *liability* [resp., *asset*] posting(s). For any given *debit* [or *credit*] posting there is one or more specific *liability* [resp., *asset*] posting(s) to be made, each such posting being in the amount of a fraction of the *debit* [or *credit*] posting, with their sum being equal to the *debit* [or *credit*] posting amount. The rational number fractions do not necessarily result in a natural number liability [resp., asset] posting. Hence these must be suitably *"rounded"*.

5.2.2 Transaction Syntax, Semantic Types.

97. The *actual posting* is thus a map from *debit* [or *credit*] *access path*s to maps from *liability* respectively [*asset*] *access path*s to natural number *amounts*.

type
97. ACT_A_POST = Deb_AccessPath $\overrightarrow{m}$ Lia_A_A_REL
97. ACT_L_POST = Cre_AccessPath $\overrightarrow{m}$ Ass_A_L_REL
97. Lia_A_A_REL = Lia_AccessPath $\overrightarrow{m}$ Amount
97. Ass_A_L_REL = Ass_AccessPath $\overrightarrow{m}$ Amount
97. Deb_AccessPath,Cre_AccessPath,Lia_AccessPath,Ass_AccessPath = AccessPath

5.2.3 Transaction Syntax, Syntactic Types.

98. A *transaction* is a pair commands: an *debit/credit enter* and a of set of *liability/asset enter* one or more commands —
99. such that these latter conform to the constraints expresses in [ι 97 π 24].

type
98. Transaction = DC_Enter × LA−Enter-**set**
axiom [Well-formed Transaction]
99. [ι 97 π 24] ...

5.2.4 Bookkeeping Behaviours.

5.2.4.1 Bookkeeping Perdurants.

We refer to Sect. **??**. We shall consider the following domain perdurants to be transcendentally deduced into domain behaviours.

100. a *double-entry debit/credit bookkeeping account* behaviour,
101. a *double-entry asset/liability bookkeeping account* behaviour, and

102. a set of *accountant* behaviours.

100. dbl_dc_book [based on] DC_ACCOUNT [i.e.,] dc_acc
101. dbl_al_book: [basd on] AL_ACCOUNT [i.e.,] al_acc
102. accountant: [basd on] ACCOUNTANTs [i.e.,]accountants

5.2.4.2 Bookkeeping Domain Behaviour Signatures.

We shall not follow the *'doctrine"* of expressing the *domain behaviour* signatures strictly according to [6]. That is: We omit a "full treatment" of all attributes. But to remind you:

- Endurants morph into behaviours.
- Behaviours are uniquely distinguished by he Unique Identifiers of "their parts": $p : P$: **uid**_$P(p)$. So the unique identifier π:UI of p is a static, constant, argument of behaviour behaviour$_P$.
- Behaviours communicate with other behaviours. So the mereology of part p indicates with which other behaviours behaviour p interacts. So the mereology **mereo**_P(p), usually modelled as a set of unique identifies, is a [usually] static argument of behaviour behaviour$_P$.
- We shall focus on a very few endurant attributes. Attributes [also] become behaviour arguments.
  - Some are *static*, cannot change value.
  - Others are *programmable*, does, indeed, change value.

103. The *double-entry debit/credit bookkeeping* behaviour, dbl_dc_book, communicates with a the set of all accountants [a *mereology* argument], and has the *debit/credit account*, dc_acc, as its *programmable* argument.
104. The *double-entry asset/liability bookkeeping* behaviour, dbl_al_book, communicates with just the *asset/liability account*, al_acc [a *mereology* argument], and has the *asset/liability account*, al_acc, as its *programmable* argument.
105. The accountant behaviour communicates with just the *double-entry asset/liability bookkeeping* behaviour [a *mereology* argument], dbl_al_book.

value
103. dbl_dc_book: UID → ACC_UI-**set** → ... → DC_ACCOUNT ... **Unit**
104. dbl_al_book: UID → ACC_UI-**set** → ... → AL_ACCOUNT ... **Unit**
105. accountant: UID → UI → (Acces_Path-**set** × ...) → ... **Unit**

5.2.4.3 Bookkeeping Behaviour Definitions.

[ι 90 π 22]. We remind the reader of the **value** definition of db_al_rel, [ι 90 π 22],

[ι 75 π 20]. and the definition of the write function, [ι 75 π 20]

106. The dbl_dc_book *behaviour* is here defined without detailing possible [*static* and *monitorable*] arguments.

107. The dbl_dc_book *behaviour* external non-deterministically, $\lceil\rceil$, awaits write commands from either of the accountant behaviours (cf. [ι 111 π 26]).

108. It then performs the write function on the *double-entry bookkeeping's debit/credit account* dc_acc.

109. After which it then performs the "corresponding" updates, at least one, possible ["a few"] more, on the *double-entry bookkeeping's asset/liability account* "al".

110. After which it "reverts" to being the The dbl_dc_book *behaviour* –

111. [with this external non-deterministic actions "ranging" over all accounts]

```
value
90.  db_al_rel = attr_DB_AL_REL(mgt) Cf. [ι 5.1.2.3.4 π 23]
106. dbl_dc_book(dci)(auis)(...)(dc_acc) ≡
107.   ⌈⌉ { let (mkWrite("dc",ap,txt,a),al_enters_set) = ch[{dci,aui}] in
108.        let dci_acc′ = write("dc",ap,txt,a)(dc_acc) in
109.        update_asset_liability_accounts(ap,txt,a);
110.        dbl_dc_book(dci)(auis)(...)(dc_acc′)
111.        end end | aui:Acc_UI • aui ∈ auis }

109. update_asset_liability_accounts: Access_Path×Txt×Amount
109. update_asset_liability_accounts(ap,txt,a) ≡ upd_ass_lia_acc(db_al_rel(ap))(ap,txt,a)

109. upd_ass_lia_acc: REL → Access_Path×Txt×Amount → Unit
109. upd_ass_lia_acc(rel)(dc_ap,txt,a) ≡
109.   case rel of
109.     [] → skip,
109.     [ac_ap↦f] ∪ rel′ →
109.       ch[{aci,aui}] ! mkWrite("al",ac_ap,txt,amount(f,a)) ;
109.       upd_ass_lia_acc(rel′)(dc_ap,txt,a)
109.   end
```

- The dbl_al_book behaviour is much like the dbl_dc_book behaviour.
- A few renamings and item [ι 109 π 26] omitted !

6 Summing Up

7 Bibliography

References

[1] Anon. *C.C.I.T.T. High Level Language (CHILL), Recommendation Z.200, Red Book Fascicle VI.12*. See [12]. ITU (Intl. Telecmm. Union), Geneva, Switzerland, 1980 – 1985.

[2] Hans Bekič and Kurt Walk. Formalization of Storage Properties. In *Symposium on Semantics of Algorithmic Languages*, volume LNM 188. Springer, 1971.

[3] B.J. Mailloux and J.E.L Peck and C.H.A. Koster and Aad van Wijngaarden. *Report on the Algorithmic Language ALGOL 68*. Springer, Berlin, Heidelberg, 1969.

[4] D. Bjørner and O. Oest. *Towards a Formal Description of Ada*, volume 98 of *LNCS*. Springer-Verlag, 1980.

[5] 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, Heidelberg, Germany, 2006.

[6] Dines Bjørner. *Domain Science & Engineering – A Foundation for Software Development*. EATCS Monographs in Theoretical Computer Science. Springer, Heidelberg, Germany, 2021. A revised version of this book is [8].

[7] Dines Bjørner. Domain Modelling – A Primer. A short version of [8]. xii+202 pages[11], May 2023.

[8] Dines Bjørner. Domain Science & Engineering – A Foundation for Software Development. Revised edition of [6]. xii+346 pages[12], January 2023.

[9] Dines Bjørner. Domain Models – A Compendium. Internet: `http://www.imm.dtu.dk/~dibj/2024/models/domain-models.pdf`, March 2024. This is a very early draft. 19 domain models are presented.

[10] Dines Bjørner. Domain Models – A Compendium. Internet: `http://www.imm.dtu.dk/~dibj/2024/models/domain-models.pdf`, March 2024. This is a very early draft. 19 domain models are presented.

[11] Dines Bjørner and Yang ShaoFa. Domain Modelling. Technical University of Denmark. Revised edition of [10]. xii+208 pages. https://www.imm.dtu.dk/ dibj/2023/dommod/dommod.pdf, May 2023.

[12] P.L. Haff, editor. *The Formal Definition of CHILL*. ITU (Intl. Telecmm. Union), Geneva, Switzerland, 1981.

[13] ANSI X3.53-1976. The PL/I programming language. Technical report, American National Standards Institute, Standards on Computers and Information Processing, 1976.

[11]This book is currently being translated into Chinese by Dr. Yang ShaoFa, IoS/CAS, Beijing and into Russian by Dr. Mikhail Chupilko, ISP/RAS, Moscow

[12]Due to copyright reasons no URL is given to this document's possible Internet location. A primer version, omitting certain chapters, is [7]