What Is Logistics ?
A Domain Analysis*

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

Document started: May 13, 2009. Compiled May 27, 2009: 20:21

Abstract

We examine the concept of logistics, exemplify it by some *"use cases"*, bring a definition of the term 'logistics' from Wikipedia (Sect. 2: *What is Logistics*), and then we rigorously and stepwise unravel the constituent concepts of *Transport Networks* (Sect. 3), *Containers and Freight Items* (Sect. 4), *Transport Companies, Vehicles and Timetables* (Sect. 5), *Handling* (Sect. 6), *Logistics Traffic* (Sect. 7) and *Senders and Receivers* (Sect. 8). In Sects. 9–10, *Model Extensions*, we discuss possible additional phenomena and concepts of logistics. s1

The document presents a domain model (in the form of a both English narrative and a formal RSL description), that is, it does not present requirements to a computerised logistics system, let alone software for such systems.

A concluding section, *Logistics System Functions* (Sect. 11) — to be written — surveys some standard software and hardware support for logistics. s2

We constrain the treatment of logistics to that of shipping companies handling (optimal) freight consignments (cf. waybills and bill of ladings) involving possibly multiple vehicles from possibly multiple transport companies.

Thus we do not cover the logistics of, say, container stowage aboard container vessels. In **http://www2.imm.dtu.dk/~db/container-paper.pdf** we cover that aspect.

Methodology

This document applies the domain engineering principles of [5–9] to the domain of logistics. The specification language used is `RSL` of the `RAISE` method [3–5,34,35,37]. The three volume [3–5] gives an overall, 2400 page introduction to software engineering, the `RAISE` specification language `RSL`, to abstraction and and modelling principles and techniques, and to the triptych of software engineering: domain engineering as a basis for requirements engineering and the latter as a basis for software design. Included in [4] are introductions to Automata and Machines, Modules and Class Diagrams, Petri Nets [54, 65, 67–69], Message Sequence Charts [50–52], State Charts [40–43,45] and Temporal Logic (in the form of DC for Duration Calculus, [78, 79]). In the present document we shall not tackle problems that cannot be expressed in `RSL`. A most recent and comprehensive intriduction to domain engineering is the less than 200 page document: http://www2.imm.dtu.dk/~db/de+re-p.pdf.

---

[*] "Inspired" by Fabio Rosetti, 14 May 2009

Contents

Methodology 1

A Series of Domain Descriptions 3

Obviously Missing Diagrams &c. 3

1 Why This Document ? 4
- 1.1 Facts 4
- 1.2 Aims & Objectives 4
  - 1.2.1 Aims 4
  - 1.2.2 Objectives 4

2 What is Logistics 4
- 2.1 The "Players" 4
- 2.2 Some Use Cases 5
  - 2.2.1 Consignment and Transport 5
  - 2.2.2 Inquiry 7
  - 2.2.3 Tracing 7
- 2.3 A Wikipedia Definition of 'Logistics' 7
- 2.4 A Definition of 'Transport' 7
- 2.5 Structure of Report 8

3 Transport Networks 8
- 3.1 Nets, Hubs and Links 9
  - 3.1.1 Mereology of Nets 9
  - 3.1.2 Reference Nets 10
  - 3.1.3 Attributes of Hubs and Links 11
- 3.2 Routes 12
  - 3.2.1 Hub Traversals, Entries and Exits 12
  - 3.2.2 Link Traversals, Entries and Exits 13
  - 3.2.3 First and Last Hubs of Link Traversals 14
  - 3.2.4 Routes 14
- 3.3 Connected and Disconnected Nets 15
- 3.4 Subnets 16
- 3.5 Route Attributes 16
- 3.6 Link, Hub, Route and Net Modalities 17
  - 3.6.1 Link and Hub Modalities 17
  - 3.6.2 Route Modalities 18
  - 3.6.3 Net Modalities 18

4 Containers and Freight Items 19
- 4.1 Containers 19
- 4.2 Freight Items 19

5 Transport Companies, Vehicles and Timetables 19
- 5.1 Transport Companies 19
- 5.2 Vehicles 20
- 5.3 Timetables 21
  - 5.3.1 Timed Link Traversals 23

6 Handling 25
- 6.1 Shipping Requests and Responses 25
  - 6.1.1 Shipping Requests 25
  - 6.1.2 Positive Shipping Request Responses: Waybills 26
  - 6.1.3 Waybill Wellformedness 27
- 6.2 Generation of Waybills 30

7 Logistics Traffic 31

8 Senders and Receivers 32
- 8.1 Senders 32
- 8.2 Receivers 32

9 Miscellaneous 32

10 Model Extensions 33

11 Logistics System Computing Functions 33

12 Conclusion 33

13 Bibliographical Notes 33

A An RSL Primer 40
- A.1 Types 40
  - A.1.1 Type Expressions 40
  - A.1.2 Type Definitions 42
- A.2 The RSL Predicate Calculus 43
  - A.2.1 Propositional Expressions 43
  - A.2.2 Simple Predicate Expressions 43
- A.3 Quantified Expressions 44
- A.4 Concrete RSL Types: Values and Operations 44
  - A.4.1 Arithmetic 44
  - A.4.2 Set Expressions 44
  - A.4.3 Cartesian Expressions 45
  - A.4.4 List Expressions 45
  - A.4.5 Map Expressions 46
  - A.4.6 Set Operations 46
- A.5 Cartesian Operations 48
  - A.5.1 List Operations 49
  - A.5.2 Map Operations 50
- A.6 λ-Calculus + Functions 52
  - A.6.1 The λ-Calculus Syntax 52
  - A.6.2 Free and Bound Variables 53
  - A.6.3 Substitution 53
  - A.6.4 α-Renaming and β-Reduction 53
  - A.6.5 Function Signatures 54
  - A.6.6 Function Definitions 54
- A.7 Other Applicative Expressions 55
  - A.7.1 Simple let Expressions 55
  - A.7.2 Recursive let Expressions 55
  - A.7.3 Predicative let Expressions 55
  - A.7.4 Pattern and "Wild Card" let Expressions 56
  - A.7.5 Conditionals 56
  - A.7.6 Operator/Operand Expressions 57
- A.8 Imperative Constructs 57
  - A.8.1 Statements and State Changes 57
  - A.8.2 Variables and Assignment 58
  - A.8.3 Statement Sequences and skip 58
  - A.8.4 Imperative Conditionals 58
  - A.8.5 Iterative Conditionals 58
  - A.8.6 Iterative Sequencing 58
- A.9 Process Constructs 59
  - A.9.1 Process Channels 59
  - A.9.2 Process Composition 59
  - A.9.3 Input/Output Events 59
  - A.9.4 Process Definitions 60
- A.10 Simple RSL Specifications 60

B Indexes 61

A Series of Domain Descriptions

This document is one in an emerging series of documents that describe indidual domains: a financial service industry (banks, securities trading, etc.), a container line industry[1], pipe line systems[2], railways[3], etc.

Obviously Missing Diagrams *&c.*

The current version is relative complete: In Sect. 6.2 on page 30 we reach a "current" high in expressing the generation of waybills from requests for consignment and optimal transport wrt. different criteria. But what is missing for the lay reader is: (i) diagrams to easen the intuitive understanding of text and formulas and (ii) explanations of the formula.

---

[1] http://www2.imm.dtu.dk/ db/container-paper.pdf
[2] http://www2.imm.dtu.dk/˜db/de+re-p.pdf
[3] http://www.railwaydomain.org/PDF/tb.pdf

s3 1 Why This Document ?

1.1 Facts

There is no document which describes logistics in a precise manner. Thus there is no student text from which one can learn about logistics in a professionally responsible way.

1.2 Aims & Objectives

By *aims* we mean: *what is being covered in this document ?* By *objectives* we mean: *what do we wish to achieve by presenting this document ?*

s4 1.2.1 Aims

We aim to cover all facets of logistics: a detailed description of the multi-modal transport nets along which suitable vehicles transport freight, from initial hub or link position origins of the net along routes of the net to hubs or link positions of the net to final hub or link position destinations of the net possibly changing from vehicles to vehicles of same or different modalities (trucks, trains, air-cargo or vessels) while possibly being temporarily warehouse
s5 stored for further shipment; a detailed description of the functions of senders, shipping companies and receivers: senders making inquiries, placing requests for transportation, accepting
s6 shipper proposed routes and fares, etc.; shipping companies finding optimal freight routes with respect to any one or a composition of requirements, and with respect to transport company time– and fare tables; and accepting responsibility for shipments, providing senders and receivers with regular information as to the whereabouts of the consigned freight, etc.; a description of those aspects of transport companies, their vehicles the timetables according to which vehicles perform transport; etc., etc.

s7 1.2.2 Objectives

It is our objective to achieve the following with this document: (i) to show that one can indeed provide a concise English narrative as well as a precise mathematical formalisation
s8 of all of the above-mentioned and many more aspects of logistics; (ii) to implicitly convince the reader that no software development ought begin without a clear, consistent and relative
s9 complete domain description of 'logistics' — including that it can be done; and (iii) to suggest that education and training, of students of shipping, and research into logistics be based on domain descriptions like the one of this document.

s10 2 What is Logistics

2.1 The "Players"

Figure 1 on the next page indicates the five major "players" on the 'logistics' scene, from left to right: the senders and receivers of freight, the shipping companies, the transport companies and their vehicles, and the transport net.

The reader may observe that we have not indicated, by any symbol, the "real" object of
s11 logistics, namely the freight items !

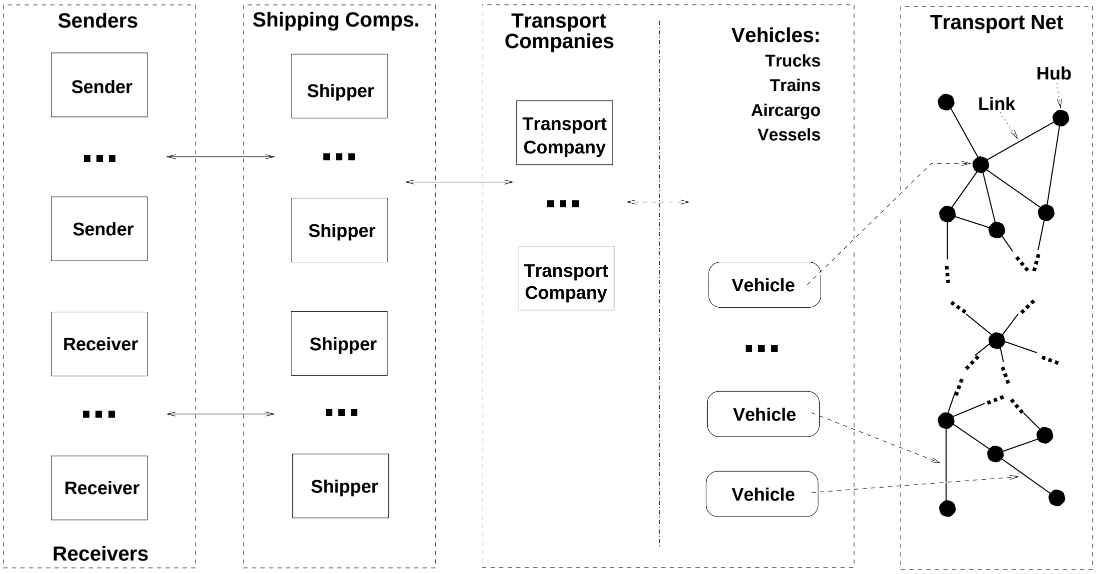

Figure 1: The Logistics “Players”

2.2 Some Use Cases s12

We present three use cases.

2.2.1 Consignment and Transport s13

1. You are a **sender**[4]: a person who, or a company which, wishes to send a consignment of a number of one or more pieces of freight from location O (origin), say in Asia, to location D (destination), say in Europe.
2. So you contact a **shipper**, that is, a **shipping company**. s14
3. You inform them of
   (a) number of pieces of freight, the individual measures (height, width, breadth and weight) of this freight,
   (b) from whom, i.e., the **sender**, name, etc., when (date and time) and where (address, **hub** or **link**[5] position) it is to be fetched,
   (c) to whom, i.e., the **receiver**, name, etc., and where (address, **hub** or **link** position) it to be delivered,
   (d) whether the freight items are already packed,
   (e) whether the freight is fragile
   (f) and/or flammable,
   (g) value of each freight item,

---

[4] The **bold face** terms appear on Fig. 1.

[5] Items 3(a)–3(b), when specifying link positions assume truck fetch or delivery — as trains, aircraft and vessels can only pause at hubs.

    (h) et cetera. s15

4. The **shipping company**,
    (a) based on knowledge about **transport companies**,
    (b) the timetables of their **vehicles** and
    (c) the **transport net** of these vehicles,
5. suggests a route of transport
    (a) with this route usually composed from several transport segments:
    (b) **truck, train, air-cargo** or **vessel**, etc., ending possibly with train and truck delivery. s16
6. The **shipping company** informs the **sender** of
    (a) transportation price,
    (b) whether **receiver** pays for local delivery or you do;
    (c) transportation dates and times:
        i. initial fetch (from a **link** position),
        ii. intermediate transfers and possible warehousing (at **hubs**),
        iii. and final delivery (from a **link** position). s17
7. You agree,
    (a) after some negotiation
    (b) that might involve alternative routes (et cetera),
8. and sign appropriate papers
    (a) bill of lading[6]
    (b) and waybills[7].
9. Your freight is fetched (from a **link** position).
10. You are — perhaps — regularly or irregularly informed of status of transport.
11. Finally freight arrives and is delivered to receiver (at a **link** position).

s18

---

[6] Wikipedia: A bill of lading (sometimes referred to as a BOL, or B/L) is a document issued by a carrier to a shipper, acknowledging that specified goods have been received on board as cargo for conveyance to a named place for delivery to the consignee who is usually identified. A through bill of lading involves the use of at least two different modes of transport from road, rail, air, and sea. The term derives from the noun "bill", a schedule of costs for services supplied or to be supplied, and from the verb "to lade" which means to load a cargo onto a ship or other form of transport.

[7] Wikipedia: A waybill is a document issued by a carrier giving details and instructions relating to the shipment of a consignment of goods. Typically it will show the names of the consignor and consignee, the point of origin of the consignment, its destination, route, and method of shipment, and the amount charged for carriage. Unlike a bill of lading, which includes much of the same information, a waybill is not a document of title.

Discussion Items 9–11 are not logistics actions. They are not performed by the shipper, maybe except for cases of Item 10. Instead they are performed by the transport company and its vehicles. Thus you see that the rôle of a shipper is to arrange, to accommodate — i.e., to manage ! The management of overall vehicle coordination with respect to (wrt.) senders, shippers and receivers is done by the transport companies and is not considered an issue of logistics. The management individual vehicles is done by the truck driver, the train engine man, the aircraft captain (pilot), respectively the ship captain and is likewise not considered an issue of logistics.

2.2.2 Inquiry

s19

You are a person who, or a company which, wishes to send a consignment of a number of one or more pieces of freight from location O (origin), say in Asia, to location D (destination), say in Europe. You are wondering about costs, transportation times, etc. So you "shop around": inquiring with a number of (one or more) shipping companies as for shipping route, times, costs, packaging, insurance, et cetera.

Therefore several of the actions mentioned above take place.

2.2.3 Tracing

s20

You are a person who, or a company which, has commits the consignment of a number of one or more pieces of freight from location O (origin), say in Asia, to location D (destination), say in Europe. There is therefore a set of bill of ladings and a waybill — all with appropriate reference identifications. Now, after initial send-off of freight, you wish to know the status of the ongoing transport, or why it appears that there is a delay in shipping. Tracing therefore takes place: the shipping company via the transport companies, finding out about the whereabouts of the freight. Et cetera,

2.3 A Wikipedia Definition of 'Logistics'

s21

According to Wikipedia (http://en.wikipedia.org/wiki/Logistics):

> *"Logistics is the management of the transport of goods, information and other resources, including energy and people, between the point of origin and the point of destination in order to meet the requirements of consumers (frequently, and originally, military organizations). Logistics involves the integration of information, transportation, inventory, warehousing, material-handling, packaging, and occasionally security[8]. Logistics is a channel of the supply chain which adds the the value of time and place utility."*

2.4 A Definition of 'Transport'

s22

By transport[ation] we shall mean

---

[8] We have covered one facet of security extensively elsewhere [21, in [9]] and shall therefore not cover this aspect in this report.

> (i) the movement (ii) of goods (iii) on a vehicle (iv) along a route of a network of hubs and [two way] links[9] (v) from a source (point of origin) to a sink (a point of destination).

s23 (i) Movement is a behaviour, that is, a function over time. (ii) Goods are items of freight that have value, volume, maybe perishable (that is, whose value diminishes rapidly with excess transportation time). (iii) Vehicles are like actors: they convey freight, they can accommodate a maximum of freight volume and weight, they can move at certain velocities within a specified range of distances — along roads, rails, or air or sea lanes. (iv) Routes are sequences of hub visits "infixed" with travels along links, that is, a sequence staring with a s24 hub (of origin), then a link, then a hub, etc., and ending with a (destination) hub. Hubs are like road intersections, train stations, airports and harbours, including production centers, warehouses, distribution centers and customer locations. Links are like road segments, rail tracks (between train stations), air lanes or sea lanes. (v) Sources and sinks are hubs.

s25 2.5 Structure of Report

We shall therefore focus on the following concepts — some of which are *highlighted in this type font* above: Sect. 3: *Transport Networks* of *hubs* and *links* (incl. *origins*, *destinations*) — covering both road, rail, air and sea transport nets; Sect. 4: *Containers and Freight Items*; Sect. 5: *Transport Companies, Vehicles and Timetables* (trucks, busses, trains, aircraft and sea vessels) and *timetables*; Sect. 6: *Handling* (consignments, bill of ladings, waybills, et cetera); Sect. 7: *Logistics Traffic*; Sect. 8: *Senders and Receivers* (temporary storage before, during and after transport); and Sect. 9: various miscellaneous issues (*packaging*, *tracing*, *notifications* et cetera).

s26 3 Transport Networks

1. We shall introduce the notions of (transport) nets, hubs and links.

Sub-sets of a transport net may be road, rail, air traffic or sea vessel nets.

2. A transport net contains two or more hubs
3. and one or more links

Examples of hubs are: street intersections of road net, train stations of a rail net, airports of an air traffic net and harbours of a sea vessel net. Examples of links are: street segments between two intersections of road net, tracks between two train stations of a rail net, air lanes between two airports of an air traffic net and sea lanes between two harbours of a sea vessel net.

```
type
  1 N, H, L
value
  2 obs_Hs: N → H-set
axiom
  2 ∀ n:N • card obs_Hs(n) ≥ 2
value
  3 obs_Ls: N → L-set
```

---

[9] A network is a graph: hubs are nodes and links are edges.

```
axiom
   3  ∀ n:N • card obs_Ls(n) ≥ 1
```

3.1 Nets, Hubs and Links

3.1.1 Mereology of Nets

We wish to express how hubs and links are connected.

4. To express how hubs and links are connected we need identify hubs and links uniquely.
5. From a hub we can observe its unique hub identifier.
6. From a link we can observe its unique link identifier.

```
type
   4  HI, LI
value
   5  obs_HI: H → HI
   6  obs_LI: L → LI
axiom
      ∀ n:N,h,h′:H,l,l′:L •
   5     {h,h′}⊆obs_Hs(n) ⇒ (h≠h′ ⇒ obs_HI(h) ≠ obs_HI(h′)) ∧
   6     {l,l′}⊆obs_Ls(n) ⇒ (l≠l′ ⇒ obs_LI(l) ≠ obs_LI(l′))
```

Axioms 5–6 express uniqueness of identifiers.

7. From a hub we can observe the link identifiers of all the links connected to the hub. s27
8. From a link we can observe the hub identifiers of the two distinct hubs to which the link is connected. s28 s29

```
value
   7  obs_LIs: H → LI-set
   8  obs_HIs: L → HI-set
axiom
        ∀ n:N, h:H, l:L • h ∈ obs_Hs(n) ∧ l ∈ obs_Ls(n) ⇒
   7       ∀ li:LI • li ∈ obs_LIs(h) ⇒ ∃ l′:L • l′ ∈ obs_Ls(n) ∧ obs_LI(l′)=li
   8       ∀ hi:HI • hi ∈ obs_HIs(l) ⇒ ∃ h′:H • h′ ∈ obs_Hs(n) ∧ obs_HI(h′)=hi
```

s30

9. Given a net one can obtain all it link and all its hub identifiers.
10. Given a net and a link identifier of that net one can obtain the so-identified link.
11. Given a net and a hub identifier of that net one can obtain the so-identified hub.

s31

```
value
   9  xtr_LIs: N → LI-set, xtr_HIs: N → HI-set
   10  xtr_L: N → LI ⥲ L
   11  xtr_H: N → HI ⥲ H

   9  xtr_LIs(n) ≡ {obs_LI(l)|l:L•l ∈ obs_Ls(n)}
   9  xtr_HIs(n) ≡ {obs_HI(h)|h:H•h ∈ obs_Hs(n)}

   10  xtr_L(n)(li) ≡ let l:L•l ∈ obs_Ls(n)∧li=obs_LI(l) in l end
                      pre li ∈ xtr_LIs(n)
   11  xtr_H(n)(hi) ≡ let h:H•h ∈ obs_Hs(n)∧hi=obs_HI(h) in h end
                      pre hi ∈ xtr_HIs(n)
```

s32

3.1.2 Reference Nets

12. A net defines a reference net.

12. A reference net maps hub identifiers to sets of one or more link identifiers.

12. Thus from a net one can calculate its reference net: For every hub its identifier is mapped into the link identifiers observable from that hub.

```
type
   12   RN = HI →m (HI −m> LI-set)
value
   12.1   calc_RN: N → RN
   12.2   calc_RN(n) ≡
   12.3     [ hi ↦ [ hi′ ↦ {obs_LI(l)
   12.4          | l:L•l ∈ obs_Ls(n)∧hi ∈ obs_HIs(l)∧hi′ ∈ obs_HIs(l)\{hi}} ]
   12.5          | h:H•h ∈ obs_Hs(n)∧hi=obs_HI(h) ]
```

s33

- We refer to
  - the hi definition set elements (leftmost hi of 12.3) of the reference net as the *origin hub* identifier;
  - the rightmost hi′ of 12.3 as a *target hub* identifier, and
  - the range set of link identifiers as ‘the range set of link identifiers’ !

s34

13. A reference net, n_{s_r}, is a sub-reference net, n_r, if
    (a) the origin hub identifiers, hi, of n_{s_r}, form a subset of the origin hub identifiers of n_r;
    (b) the set of target hub identifiers, hi′, for origin hub identifier hi, of n_{s_r}, form a subset of those of n_r; and

(c) the range set of link identifiers in n_{s_r} is a subset of those of the corresponding range set of link identifiers in n_r.

s35

value

13 is_sub_ref_net: RN × RN → **Bool**
13 is_sub_ref_net(rn′,rn) ≡
13(a) **dom** rn′ ⊆ **dom** rn ∧
13(b) ∀ hi:HI • hi ∈ **dom** rn ⇒ **dom** rn′(hi) ⊆ **dom** rn(hi) ∧
13(c) ∀ hi′:HI • hi′ ∈ **dom** rn′(hi) ⇒ (rn′(hi))(hi′)⊆(rn(hi))(hi′)

3.1.3 Attributes of Hubs and Links

s36

14. Hubs have a number of attributes:
    - (a) spatial (i.e., geographic) location which, since we simply hubs a points, can be represented by three coordinates: longitude, latitude and altitude;
    - (b) duration (time) of
      - i. entering,
      - ii. traversing and
      - iii. leaving

      a hub[10];
    - (c) et cetera.

s37

15. Links have a number of attributes:
    - (a) spatial (i.e., geographic) location which, since we simply links as lines that can be described in the way that we describe Bezier curves[11];
    - (b) length;
    - (c) cost of transporting a unit of freight volume per unit of length along the link;
    - (d) duration (time) of
      - i. entering,
      - ii. traversing and
      - iii. leaving

      a link[12];
    - (e) et cetera.

s38

---

[10] The time intervals are specific to each hub and depends on direction of traversal, type of vehicle and its load status

[11] http://en.wikipedia.org/wiki/Bézier_curve

[12] We disregard the possibility that traversing a link in one direction may take longer time than traversing it in the opposite direction.

type
  14(a) HLoc
  14(b) TimDur
  14(c) ...
  15(a) Bezier
  15(b) Length
  15(c) Cost
value
  14(a) obs_HLoc: H → HLoc
  14(b) obs_InTime, obs_TravTime, obs_OutTime: H×... → TimDur
  15(a) obs_LLoc: L → Bezier
  15(b) obs_Length: L → Length
  15(c) obs_Cost: L → Cost
  15(d) obs_InTime, obs_TravTime, obs_OutTime: L×... → TimDur
  15(e) ...

s39

3.2 Routes

s40

3.2.1 Hub Traversals, Entries and Exits

16. A hub traversal is here represented by a triple
    (a) a(n input) link identifier, ili,
    (b) a hub identifier, hi and
    (c) a(n output) link identifier, oli,

    such that

    (d) the identifiers are those of links and hubs of the network,
    (e) the two link identifiers are observable from the hub identified by hi.
17. A hub “entry” is here represented by the pair of the first two elements of a hub traversal.
18. A hub “exit” is here represented by the pair of the two two elements of a hub traversal.

s41

type
  16 HubTrav = LI × HI × LI
  17 HubEntry = LI × HI
  18 HubExit = HI × LI
axiom
  16(d) ∀ n:N, (ili,hi,oli):HubTrav • (ili,hi,oli) ∈ HubTraversals(n)
  16(b) ∀ n:N, (ili,hi):HubEntry • (ili,hi) ∈ HubEntries(n)
  16(c) ∀ n:N, (oli):HubExit • (hi,oli) ∈ HubExits(n)
  ... et cetera
value
  HubTraversals: N → HubTrav-**set**
  HubTraversals(n) ≡

{(ili,hi,oli)|(ili,hi,oli):HubTrav, h:H • hi=obs_HI(h)∧{ili,oli}⊆obs_LIs(h)}
HubEntries: N → HubEntry-**set**
HubEntries(n) ≡ {(li,hi)|(ili,hi):HubEntry, h:H • hi=obs_HI(h)∧li ∈ obs_LIs(h)}
HubExits: N → HubExit -**set**
HubExits(n) ≡ {(hi,li)|(hi,oli):HubExit, h:H • hi=obs_HI(h)∧li ∈ obs_LIs(h)}

3.2.2 Link Traversals, Entries and Exits

s42

19. A link traversal is here represented by a triple
    (a) a(n input) hub identifier, ihi,
    (b) a link identifier, li and
    (c) a(n output) hub identifier, ohi,

    such that

    (d) the identifiers are those of links and hubs of the network,
    (e) the two hub identifiers are observable from the link identified by hi.
20. A link “entry” is here represented by the pair of the first two elements of a link traversal.
21. A link “exit” is here represented by the pair of the two two elements of a link traversal.

s43

type
19 LinkTrav = HI × LI × HI
20 LinkEntry = HI × LI
21 LinkExit = LI × HI
axiom
19(d) ∀ n:Nii, (ihi,li,oli):HubTrav • (ihi,li,ohi) ∈ LinkTraversals(n)
19(b) ∀ n:N, (ihi,li):HubEntry • (ihi,li) ∈ LinkEntries(n)
19(c) ∀ n:N, (li,ohi):HubExit • (li,ohi) ∈ LinkExits(n)
... et cetera
value
LinkTraversals: N → LinkTrav-**set**
LinkTraversals(n) ≡
{(ihi,li,ohi)|(ihi,li,ohi):LinkTrav, l:L • li=obs_LI(h)∧{ihi,ohi}=obs_HIs(l)}
LinkEntries: N → LinkEntry-**set**
LinkEntries(n) ≡ {(ihi,li)|(ihi,li):LinkEntry, l:L • li=obs_LI(l)∧hi ∈ obs_HIs(l)}
LinkExits: N → HubExit -**set**
LinkExits(n) ≡ {(li,ohi)|(li,ohi):LinkExit, l:L • li=obs_HI(l)∧hi ∈ obs_HIs(l)}
axiom
...

3.2.3 First and Last Hubs of Link Traversals s44

22. If (hi,li,hi′) is a link traversal then
    (a) hi identifies the *first* hub of that traversal, and
    (b) hi′ identifies the *last* hub of that traversal

```
value
  22(a)  fstHI: LinkTrav → HI
  22(a)  fstHI(hi,li,hi′) ≡ hi
  22(b)  lstHI: LinkTrav → HI
  22(b)  lstHI(hi,li,hi′) ≡ hi′
```

s45 3.2.4 Routes

23. Routes are sequences of one or more link traversals and defined as follows:
    (a) **Basis Clause:** A sequence of one link traversal is a route.
    (b) **Induction Clause:** If r and r' are routes such that the
        i. last hub identifier of the last traversal of r
        ii. is the same as the first hub identifier of the first traversal of r'
        iii. then $r^\frown r'$ is a route.
    (c) **Extremal Clause:** Only sequences of link traversals that can be formed from a finite number of uses of the basis and the induction clauses are routes.

s46

```
type
  23  Route′,R′ = LinkTrav*
  23  Route,R = {|r:R′ • len r≥1 ∧ wf_R(r)|}
value
  23  wf_R: R′ → Bool
  23  wf_R(r) ≡
        case r of
  23(a)   ⟨⟩ → true,
  23(a)   ⟨(hi,li,hi′)⟩ → true,
  23(b)   r⌢⟨(hi,li,hi′)⟩⌢⟨(hi″,li′,hi‴)⟩⌢r′ → wf_R(r)∧wf_R(r′)∧hi′=hi″
        end
  gen_Rs: N → R-infset
  gen_Rs(n) ≡
  23(a) let rs = {⟨lt⟩|lt:LinkTrav•lt ∈ LinkTraversals(n)}
  23(b)          ∪ {r⌢r′|r,r′:R • {r,r′}⊆rs∧lstHI(r(len r))=fstHI(r′(1))} in
        rs end
```

s47 The gen_Rs function generates all routes of a network. For technical reasons we have defined the well-formedness of routes predicate, wf_R, to also apply to empty sequences of link traversals although they are not (proper) routes. Whereas the definition of routes did not refer to the net whereby well-formedness of routes was just a "syntactic" matter, the function that generates routes (from a net) secures "semantic" well-formedness of routes. s48

24. Given a net and two distinct hub identifiers (of that net)
    (a) one can calculate whether there is a route from the one identified hub to the other (and, since all links are two way links, vice versa);
    (b) and, if there is such a route then one can calculate the set of all such routes.

```
value
  24(a)  is_route: N × (HI×HI) → Bool
  24(a)  is_route(n,(fhi,thi)) ≡ {r|r:R•fstHI(r(1))=fhi∧lstHI(r(len r))=thi}≠{}
  24(b)  routes: N × (HI×HI) → R-set
  24(b)  routes(n,(fhi,thi)) ≡ {r|r:R•fstHI(r(1))=fhi∧lstHI(r(len r))=thi}
```

s49

25. Since all links are two-way links one can speak of reverse links.

```
value
  25  reverse_route: R → R
  25  reverse_route(r) ≡
  25    case r of
  25      ⟨⟩ → ⟨⟩,
  25      ⟨(hi,li,hi′)⟩⌒r′ → reverse_route(r′)⌒⟨(hi′,li,hi)⟩
  25    end
```

3.3 Connected and Disconnected Nets

s50

We assume, throughout, that all links can be traversed in both directions, that is, there are no *cul de sacs* (*sackgasse*, “blind” streets).

26. A net is said to be connected if for every pair of distinct hubs of the net there is a route that connects them, i.e., from the one hub to the other.

27. Two otherwise, i.e., respectively connected nets, n_i, n_j, are said to be disconnected if they share no hubs and links.

28. A net defines a set of one or more disconnected nets.

s51

```
value
  26  is_connected: N → Bool
  26  is_connected(n) ≡
  26    ∀ h,h′:H •{h,h′}⊆obs_Hs(n)⇒is_route(n,(obs_HI(h),obs_HI(h′)))

  27  are_disjoint: N×N → Bool
  27  are_disjoint(n,n′) ≡
  27    obs_Hs(n)∩ obs_Hs(n′)={}∧obs_Ls(n)∩ obs_Ls(n′)={}

  28  disconnected_nets: N → N-set
  28  disconnected_nets(n) as ns
  28    post ∪{n|n:N•n ∈ ns}=n
```

3.4 Subnets

s52

29. A given net, n, defines a set of one or more subnets $\{n_1, n_2, \ldots, n_m\}$.
30. A net, n_s, is a subnet of another net, n,
    (a) if the reference net, nr_s, of n_s
    (b) is a sub-reference-net, rn, of n.

```
29 subnets: N → N-set
29 subnets(n) as ns
29   post ∀ n′:N • n′ ∈ ns ⇒ sub_ref_net(calc_RN(n′),calc_RN(n))
30 is_subnet: N × N → Bool
30 is_subnet(ns,n) ≡ ns ∈ subnets(n)
```

s53

3.5 Route Attributes

31. Routes have lengths — “measured” as the sum of the lengths of all the links denoted by link traversal link identifiers.
    (a) Thus a route from a first hub h to a last hub h'
    (b) has same length as the reverse route (from a first hub h' to a last hub h).
32. Routes have travel times — “measured” as the sum of the travel times of all the links denoted by link traversal link identifiers.
33. Given two distinct hubs (say, by their hub identifiers) one can calculate
    (a) the shortest route(s) between these two hubs; and
    (b) the fastest route(s) between these two hubs given the attributes of the vehicle which is supposed to travel the route.

s54

```
value
  31 length: R × N → Length
  31 +: Length × Length → Length
  31 length(r,n) ≡
  31   case r of
  31     ⟨⟩ → 0,
  31     ⟨(_,li,_)⟩⌒r′ → obs_Length(xtr_L(n)(li)) + length(r′,n)
  31   end
  32 travel_time: R × N → Time
  32 +: Length × Length → Length
  32 travel_time(r,n) ≡
  32   case r of
  32     ⟨⟩ → 0,
  32     ⟨(_,li,_)⟩⌒r′ → obs_TravTime(xtr_L(n)(li)) + travel_time(r′,n)
  32   end
```

s55 One can prove:

```
lemma:
    ∀ n:N,r:R • r ∈ routes(n) ⇒
        length(r)(n) = length(reverse_route(r))(n)
        travel_time(r)(n) = travel_time(reverse\_route(r))(n)
```

Some “interesting” functions: s56

```
value
    33(a)  shortest_route: N × (HI×HI) → R×Length
    33(a)  shortest_route(n,(fhi,thi)) ≡
    33(a)    let rs = routes(n,(fhi,thi)) in
    33(a)    {r|r:R•r ∈ rs∧∼∃ r′:R•r′isin rs∧length(r′)<length(r)}
    33(a)    end

    33(b)  fastest_route: N × (HI×HI) → R×Days
    33(b)  fastest_route(n,(fhi,thi)) ≡
    33(b)    let rs = routes(n,(fhi,thi)) in
    33(b)    {r|r:R•r ∈ rs∧∼∃ r′:R•r′isin rs∧travel_time(r′)<travel_time(r)}
    33(b)    end

    33(b)  least_costly_route: N × (HI×HI) → R×Cost
    33(b)  least_costly_route(n,(fhi,thi)) ≡
    33(b)    let rs = routes(n,(fhi,thi)) in
    33(b)    {r|r:R•r ∈ rs∧∼∃ r′:R•r′isin rs∧cost(r′)<cost(r)}
    33(b)    end
```

3.6 Link, Hub, Route and Net Modalities s57

3.6.1 Link and Hub Modalities

34. With a link we now associate a further attribute: that of is transport modality which is either that of `road`, `rail`, `air`, or `sea`.
35. To provide for “smooth” transfer of freight from respective vehicle modalities (`truck`, `train`, `air-cargo`, respectively `vessel`),
36. we expect hubs connected to n links to have up to four hub modalities, that is, any subset of the set {`truck`,`train`,`air-cargo`,`vessel`}.

s58

```
type
    34  TM == road | rail | air | sear
    35  VM == truck | train | aircargo | vessel
value
    34  obs_TM: Link → TM
    35  obs_VM: Vehicle → VM
    36  obs_TMs: Hub → TM-set
```

s59 where we presuppose the vehicle phenomenon.

37. Links incident upon a hub in a net must be of a modality also represented by that hub, and for all links and hubs.
38. A hub of a net must have exactly the modalities of the links connected to that hub.

```
axiom
   ∀ n:N, l:L, h:H •
      l ∈ obs_Ls(h)∧h ∈ obs_Hs(h)∧obs_LI(l) ∈ obs_LIs(h)∧obs_HI(h) ∈ obs_HIs(l)⇒
37       obs_TM(l) ∈ obs_TMs(h) ∧
38       ∀ li:LI • li ∈ obs_LIs(h) ⇒
38          obs_TM(xtr_LI(li)(n))∈ obs_TMs(h)
```

s60

3.6.2 Route Modalities

39. A route is said to be a single modularity route if all its links are of the same modality.
40. A route is said to have the set of 1, 2, 3 or 4 modalities that are those of its links.

```
value
   39  is_sgl_TM: Route → N → Bool
   40  route_TMs: Route → N → RM-set

   39  is_sgl_TM(r)(n) ≡
   39    ∀ i,j:Nat • {i,j}⊆indes(r)
   39      let (_,li,_)=r(i),(_,lj,_)=r(j) in
   39      obs_TM(xtr_L(n)(li))=obs_TM(xtr_L(n)(lj)) end

   40  route_TMs(r)(n) ≡
   40    {obs_TM(xtr_L(n)(li))|(_,li,_):LTrav • (_,li,_)∈ elems r}
```

s61

3.6.3 Net Modalities

41. A net is said to be a single modality net if all its routes are of the same modality.
42. The modality of a net is the set of modalities of its routes.

```
value
   41   is_sgl_TM: N → Bool
   41   is_sgl_TM(n) ≡
   41     ∀ r,r′:R • {r,r′}⊆routes(n) ⇒
   41       route_TMs(r)=route_TMs(r′)∧card route_TMs(r)=1

   42   net_modalities: N → TM-set
   42   net_modalities(n) ≡
   42     ∪{route_TMs(r)(n)|r:R • r ∈ routes(n)}
```

s62 4 Containers and Freight Items

4.1 Containers

43.

44.

45.

46.

43
44
45
46

4.2 Freight Items s63

47.

48.

49.

50.

47
48
49
50

5 Transport Companies, Vehicles and Timetables s64

5.1 Transport Companies

For simplicity, but with no loss of generality, we assume that each company is “mono-modal”, that is offering either

truck, train, aircargo, or vessel

transport; and we assume that all such transport is line transport, that is, freight can be carried, without reloading, along either of a standard set of routes. For each such line there is a timetable which repeats itself at regular intervals. s65

More precisely:

51. A transport company operates

  - (a) a finite number of one or more vessels, identified by their unique vessel identifiers, and
  - (b) is focused on a a finite number of one or more timetables. and
  - (c) has a unique (transport company) identification.

s66

```
type
   51  TransComp
   51(a)  Vid
   51(b)  Timetable, TT
   51(c)  TCId
value
   51(a)  obs_VIds: TransComp → VId-set
   51(b)  obs_Timetable, obs_TT: TransComp → Timetable-set
   51(c)  obs_TCId: TransComp → TCId
```

s67 5.2 Vehicles

Without loss of generality we assume all vessels to be container vessels.

52. There are vehicles.
53. Vehicles have unique vehicle identification
    - (a) from which one can observe the identification of the transport company which operates the vehicle.
54. A vehicle is either a truck, a train, an aircargo (aircraft, aircargo for short) or a vessel.
55. A vehicle location is either at
    - (a) at a hub, identified by that hub's unique identifier, or
    - (b) or along a link (identified by that link's unique identifier), from some hub (identified by that hub's unique identifier)
    - (c) a fraction, f, of the distance to another hub (identified by that hub's unique identifier).

s68

56. From a vehicle one can observe which freight the vehicle is conveying (at the moment, the time, of being observed), where we simplify the freight observation to
    - (a) observing the set of the bill-of-ladings for each freight item and
    - (b) the identification of the container in which it is packed.
57. One might wish to add such possibly observable information as:
    - (a) expected arrival (date and time) at next hub,
    - (b) velocity,

etc.

s69

```
type
   52       Vehicle, CId, Velocity
   53       VId
   53(a)   TCId
   54       Vehicle_type == truck | train | aircargo | vessel
   55       VLoc =  VHLoc | VLLoc
   55(a)   VHLoc == atH(hi:HI)
   55(b)   VLLoc == onL(thi:HI,li:LI,f:Frac,thi:HI)
   55(c)    Frac = {|r:Real•0<r<1|}
   56(a)    BoL
value
   53        obs_VId: Vehicle → VId
   53(a)    obs_TCId: VId → TCId
   54        obs_Vehicle_type: Vehicle → Vehicle_type
   55        obs_VLoc: Vehicle → VLoc
   56(a)    obs_BoLs: Vehicle → BoL-set
   56(b)    obs_Cid: Vehicle × BoL ⥲ CId
   57(a)    obs_Arrival: Vehicle → (Date × Time)
   57(b)    obs_Velocity: Vehicle → Velocity
```

5.3 Timetables

s70

58. Timetables are wellformed relative to a net.[13]
59. There is a concept of timetable identifiers.
60. A timetable
    (a) has a timetable identifier;
    (b) features a reference net; and finally the timetable also
    (c) lists a sequence of timed *link traversals*
61. From a timetable identifier one may observe the identifier of the transport company which operates a freight service according to that timetable.
62. From a timetable identifier one may observe the identification of the vehicle that has been allocated to serve the timetabled schedule.

s71

Two or more timetables of different names may feature identical timetables — in which case only the observable transport company identifiers are different[14].

---

[13] When in formula line 58 we postulate a net: **value** n:N, then that value declaration should be seen as ranging over any net.

[14] that is: “competition to the line”

```
value
   58        n:N
type
   59        TTId
   60        TT′ =
   60(a)          TTId
   60(b)            × RN
   60(c)              × TLT*
value
   61        obs_TCId: TTId → TCId
   62        obs_VId: TTId → Vid
```

s72

63. Timetables must be well-formed, that is, the link traversals of a timetable
    (a) must visit exactly $m+1$ hubs where m is the length of the list of link traversals;
    (b) must be commensurate with the timetable reference net ('commensurability' is expressed by the tt_is_ref_net_commensurable predicate below),
    (c) the timetable link traversal list must be well-formed, and,
    (d) given a net, n, and a timetable, tt, the timetable reference net, rn, must be commensurate with the net n (that is, refnet_is_tt_commensurable(rn,n)).

s73

```
type
   63        TT = {|tt:TT′•wf_TT(tt)(n)|}
value
   63        wf_TT: TT′ → N → Bool
   63        wf_TT(tt:(_,rn,tltl))(n) ≡
   63(a)         card{hi|(hi,li,hi′):LTrav•(_,(hi,li,hi′),_)∈ elems tltl}=len tltl+1 ∧
   63(b)         tt_is_refnet_commensurable(tt) ∧
   63(c)         wf_TLT*(tltl) ∧
   63(d)         refnet_is_net_commensurable(rn,n)
```

s74

64. (cf. Item 63(b).) Commensurability of a timetable's lists of link traversals with respect to that timetable's reference net is defined as follows:
    (a)
    (b)
    (c)

65. (cf. Item 63(d).) Commensurability of a timetable's reference net with respect to the (global) net is defined as follows:
    (a)
    (b)
    (c)

s75

```
64      tt_is_refnet_commensurable: TT → Bool
64(a)   tt_is_refnet_commensurable(_,rn,tltl) ≡
64(b)
64(c)

65      refnet_is_net_commensurable: RN × N → Bool
65      refnet_is_net_commensurable(rn,n) ≡
```

s76

66. Instead of representing a set of timetables as a set of the timetables as defined above we may represent them as a map from timetable identifiers to pairs of reference net and lists of timed link traversals.

67. Such maps must be well-formed.

68. The well-formedness conditions can be referred back to well-formednes of the previously defined timetables.

s77

```
type
   66  TTs′ = TTId →m RN × TLT*
   67  TTs = {|tts:TTs′ • wf_TTs(tts)(n)|}
value
   68  wf_TTs: TTs′ → N → Bool
   68  wf_TTs(tts)(n) ≡
   68    ∀ ttid:TTId • ttid ∈ dom tts ⇒
   68      let (rn.tltl) = tts(ttid) in wf_TT(ttid,rn,tltl)(n) end
```

5.3.1 Timed Link Traversals

s78

69. Timed link traversals, besides the link traversal, contains the date/times of entering and leaving the link and the

70. cost to the user (sender/receiver) per unit of freight volume for getting such a unit of freight volume transported along the identified link.

71. Well-formed timed link traversals must be understood in the context of the global net[15] in which transport takes place.

```
value
   fn:15   n:Net
type
   69   TLT′ = (Date×Time)×LinkTrav×Cost×(Date×Time)
   70   Cost −−− see also Item 15(c) on page 11
   71   TLT = {|tlt:TLT′•wf_TLT(tlt)(n)|}
```

s79

72. For each timed link traversal the date/time of entering the link must precede the date/time of leaving the link;
73. the interval, TI, between these date/times must be commensurate with the length and “normative” velocity of the identified link; and
74. the user cost of transporting a unit of freight along the link must be commensurate with the normative cost of moving a vehicle along that link.

s80

```
value
  71   wf_TLT: TLT′ → N → Bool
  71   wf_TLT(tlt:((d,t),(hi,li,hi′),c,(d′,t′)))(n) ≡
  72     precede((d,t),(d′,t′)) ∧
  73     commensurate_time(interval((d,t),(d′,t′)),obs_TravTime(xtr_L(n)(li))) ∧
  74     commensurate_cost(c,xtr_L(n)(li))
  72   precede: (Date×Time)×(Date×Time) → Bool
type
  73   TI[16]
value
  73   commensurate_time: TI × TI → Bool
  73   interval: (Date×Time) × (Date×Time) → TI
  74   commensurate_cost: Cost × L → Bool
  74   commensurate_cos(c,l) ≡
  74     ... c = f(obs_Length(l),obs_Cost(l),...) ...
  74     [ where f is a real valued function over two arguments:  ]
  74     [ length and cost typically yielding a value larger than 1 ]
```

s81

75. Lists of timed link traversals must be time-wise ordered:
    (a) for all adjacent positions, i and i+1, in the list
    (b) the ith departure date/time and the i+1st arrival time
    (c) most have the former precede the latter.
    (d) the reference net (implicitly) expressed by the list of timed link traversals must be a sub reference net of the timetable reference net.

s82

```
value
  75      wf_TLT*: TLT* → Bool
  75      wf_TLT*(tltl) ≡
  75(a)     ∀ i:Nat•{i,i+1}⊆inds tltl⇒
```

---

[15] That is why we bring the **value** declaration n:Net in formula line fn:15 Page 23.

[16] Time intervals arise when one date/time is subtracted from another date/time. One can add time intervalsto get a time interval ; one can add a time interval to a date/time to obtain a date/time; one can multiply a time interval with a number (whether natual or real; etc.

```
75(b)      let (_,_,_,(d,t))=tltl(i),((d′,t′),_,_,_)=tltl(i+1) in
75(c)      precede((d,t),(d′,t′)) end ∧

75(d)    is_sub_refnet(xtr_RN(tltl),rn)

75(d)    xtr_RN: TLT* → RN
75(d)    xtr_RN(tltl) ≡ [hi↦[hi′↦{li}]|(hi,li,hi′):LTrav•(hi,li,hi′)∈ elems tltl][17]
```

6 Handling

s83

We shall look at only a single aspect of handling, namely that of responding to a request from sender c: provide an optimal shipping, s_o, of such-and-such, a, freight, f, from origin h to receiver c', destination h' at this time, t, or at some earliste time, t', thereafter; a stands for attributes of freight f.

6.1 Shipping Requests and Responses

s84

6.1.1 Shipping Requests

76. A shipping request contains the following information:
    (a) Name, c, of sender;
    (b) origin, h_i, of freight, i.e., where to be sent from;
    (c) destination, h'_j, of freight, i.e., where to be sent to;
    (d) attributes, a, of freight;
    (e) Name, c', of receiver;
    (f) some optimality criterion: *"fastest"* route, *"least costly"* route, or *"earliest arrival date"*, or other; and
    (g) the date/time of submission of the request.
77. A negative response to a shipping request has the form of a `‘‘request is not feasible’’`.

s85

```
type
          SndrId, RcvrId, FreightAttrs, Neg_Resp
  76      Ship_Req′ =
  76(a)       SndrId
  76(b)       × HI                       [from]
  76(c)       × HI                       [to]
  76(d)       × RcvrId
  76(e)       × Freight_Attrs
  76(f)       × Optimality
  76(g)       × (Date × Time)            [earliest send date]
  76(f)     Optimality == fastest|cheapest|earliest_arrival|...
  77        Neg_Resp        × TT*
```

s86

78. For a shipping request, shipreq:Ship_Req′, to be well-formed
    (a) the sender and receiver identifiers must be different and
    (b) the origin and destination hubs must be different.

value

```
78      wf_Ship_Req: Ship_Req → Bool
78      wf_Ship_Req(sid,hi,hi′,rid,fas,o,dt) ≡
78(a)       sid ≠ rid
78(b)       hi ≠ hi′
```

s87

6.1.2 Positive Shipping Request Responses: Waybills

79. A positive response to a shipping request has the form of a waybill, WB, which contains the following information:
    (a) sender's identification, *c*;
    (b) from where, hi:HI, freight is to originate (fetched);
    (c) to where, hi′:HI, freight is to be destined (delivered);
    (d) the receiver's identification, c';
    (e) attributes, a, of the freight;
    (f) the list of one or more timetables, i.e., the possibly optimal shipping;
    (g) the total cost of shpping;
    (h) the date/time of start of transport;
    (i) the date/time of earliest delivery of freight; and
    (j) the total elapsed time interval of transport, measured in number of days.

s88

type

```
79(j)    Days
79       WB =
79(a)        SndrId
79(b)        × HI                [from]
79(c)        × HI                [to]
79(d)        × RcvrId
79(e)        × Freight_Attrs
79(f)        × TT*
79(g)        × Cost
79(h)        × (Date × Time)    [send date]
79(i)        × (Date × Time)    [receipt date]
79(j)        × Days              [duration]
```

s89 6.1.3 Waybill Wellformedness

Well-formedness of waybills must be expressed in terms of the global transportation net and the set of timetables available to the shipping company which produces the waybill.

80. The waybill is well-formed in the context of the net and a set of shipping agent timetables
    (a) waybill sender and receiver identifications must be different;
    (b) waybill from and to hub identifications must be different;
    (c) waybill timetable list must not be empty;
    (d) if the timetable list of the waybill is well-formed with respect to the set of shipping agent timetables;
    (e) if the first hub identifier of the timetable list of the way bill equals the ‘from’ hub identifier of the waybill and the last hub identifier of the timetable list of the way bill equals the ‘to’ hub identifier of the waybill;
    (f) waybill specified cost must be commensurate with the costs of each of the transports stated in the waybill timetable list;
    (g) freight departure date/time must precede freight arrival date/time; and
    (h) the total elapsed time interval of transport must be commensurate with the interval between the freight departure date/time and freight arrival date/time.

s90

value
```
80     wf_WB: WB → (N × TTs) → Bool
80     wf_WB(sid,fhi,thi,rid,fas,ttl,c,sdt,rdt,dur)(n,tts) ≡
80(a)      sid ≠ rid ∧
80(b)      fhi ≠ thi ∧
80(c)      ttl ≠ ⟨⟩ ∧
80(d)      wf_tt_arguments(ttl,tts) ∧
80(e)      from_to((fhi,thi),ttl) ∧
80(f)      commensurate_costs(c,ttl) ∧
80(g)      precede(sdt,rdt) ∧
80(h)      commensurate_duration((sdt,rdt),duration(ttl))
```

s91

81. (80(e)) The timetable arguments (contained in ttl and tts) are well-formed
    (a) if the timetables mentioned in ttl all have distinct timetable identifiers;
    (b) if the timetables mentioned in ttl are defined in tts;
    (c) if the list of timed link traversals contained in the time table named ttid in ttl is a sublist of the time table named ttid in tts;
    (d) if the list of timed link traversal lists are connected;
    (e) if the sublists do not specify the revisit hubs.

s92

---

[17] The constraint expressed in Item and formula line 63(a) secures that there is only one link in the list of link traversals, hence $\{li\}$, between hub identifiers hi and hi'.

```
value
  81     wf_tt_arguments: TT* × TTs → Bool
  81     wf_tt_arguments(ttl,tts)
  81(a)     let ttids = {ttid|i:Nat•i ∈ inds ttl⇒(ttid,_,_)=ttl(i)} in card ttids = len ttl ∧
  81(b)     ttids ⊆ dom tts end ∧
  81(c)     ∀ i:Nat•i ∈ inds ttl ⇒
  81(c)       let (ttid,rn,tltl)=ttl(i) in let (rn′,tltl′)=tts(ttid) in is_sublist(tltl,tltl′) end end ∧
  81(d)     ∀ i:Nat•{i,i+1}⊆inds ttl ⇒ lstHI((ttl(i))(len ttl(i)))=fstHI((ttl(i+1))(1)) ∧
  81(e)     no_hub_revisits(ttl)
```

s93

82. (83) A timed link traversal list, tltl, is a sublist, is_sublist(tltl,tltl′), of another timed link traversal list, tltl′,
    (a) if there are two indices into tltl′
    (b) such that the elements in tltl′ between and including these index positions equals tltl.

```
value
  82       is_sublist: TLT* × TLT* → Book
  82       is_sublist(tltl,tltl′) ≡
  82(a)        ∃ i,j:Nat • i≤j ∧ {i,j}⊆inds tltl′ ⇒
  82(b)          tltl = ⟨tltl′(k)|i≤k≤j⟩
```

s94

83. The no_hub_revisits predicate[18] is specified as follows:
    (a) first a single list, ltlt, of time link traversals is constructed from the **conc**atenation of the list of time link traversals contained in each of the timetables of the waybill;
    (b) then the set, his, of distinct hub identifiers of ltlt is constructed;
    (c) the number of hub identifiers in that set, that is, **card** his, must be equal to one plus the length of the consolidated list ltlt — a larger number would mean that the individual lists of time link traversals contained in each of the timetables of the waybill were not connected, and if it was smaller then there would be revisits.

s95

```
value
  83     no_hub_revisits: TT* → Bool
  83     no_hub_revisits(ttl) ≡
  83(a)      let ltlt = conc⟨tlti|i:[1..len ttl]•let (_,_,tlti′)=ttl(i) in tlti=tlti′ end⟩ in
  83(b)      let his = {hi,hi′|hi:HI•(hi″,_,hi‴):LinkTrav•(hi,_,hi′)∈ elems ltlt∧hi=hi″∧hi′=hi‴} in
  83(c)      card his = len ltlt+1 end end
```

s96

84. (80(e)) The predicate from_to expresses

---

[18] The no_hub_revisits predicate tests that the sublists of timed link traversal lists contained in its single ttl argument do not describe the revisit hubs

   (a) that the first hub identifier of the timetable list of the way bill equals the 'from' hub identifier of the waybill, and
   (b) that the last hub identifier of the timetable list of the way bill equals the 'to' hub identifier of the waybill;

value

```
84      from_to: (HI×HI) × TT* → Bool
84      from_to((fhi,thi),ttl) ≡
84(a)      fhi = fstHI((ttl(1))(len ttl(1))) ∧
84(b)      thi = lstHI((ttl(len ttl))(len ttl(len ttl)))
```

s97

85. The `commensurate_costs(c,accumulated_cost(ttl))` (80(f)) predicate
   (a) sums the costs of the summing of costs of each individual list of timed (and costed) link traversals given in each of the waybill timetables
   (b) and compares that to the cost directly described in the waybill; the comparison is non-determinate, that is, we do not describe precise means of comparing these costs.

value

```
85    commensurate_costs: Cost × Cost → Bool
85    commensurate_costs(c,ttl) ≡
85(a)     let costs = sum_of_sums_of_costs(ttl) in
85(b)     costs ≃ cost end

    ≃: Cost × Cost → Bool
```

s98

86. The `sum_of_sums_of_costs` function calculates its cost result by recursion:
   (a) if the argument list is empty the cost is zero (0),
   (b) else the cost is the sum of the cost described in the first link traversal and the `sum_of_sums_of_costs` of the rest of the argument list.

value

```
86   sum_of_sums_of_costs: TT* → Cost
86   sum_of_sums_of_costs(ttl) ≡
86(a)     if ttl = ⟨⟩ then 0 else
86(b)     let (_,_,c,_) = hd ttl in c ⊕ sum_of_sums_of_costs(tl ttl) end end

⊕: Cost × Cost → Cost
```

s99

87. The `precede(sdt,rdt)` (80(g)) predicate is left undefined.

   Once a specific representation of dates and time has been decided upon one can then easily define this function.

```
value
  87   precede: (Date×Time) × (Date×Time) → Bool
  87   precede(sdt,rdt) ≡ sdt ≪ rdt

  ≪: (Date×Time) × (Date×Time) → Bool
```

s100

88. The commensurate_duration((sdt,rdt),duration(ttl)) (wfwbi) predicate also requires a specific representation of dates and time in order to be calculated, that is:
    (a) one must somehow subtract sdt from rdt
    (b) and then perform the commensurateness test.

s101

```
value
  88      commensurate_duration: ((Date×Time)×(Date×Time))×Days → Bool
  88      commensurate_duration((sdt,rdt),duration(ttl)) ≡
  88(a)       let dur = rdt − sdt in
  88(b)       dur ≃ duration(ttl) end

      duration: TT* → Days
      duration(ttl) ≡
          if ttl = ⟨⟩ then 0
          else let (dt,_,_,dt′) = hd ttl in (dt′ ⊖ dt) ⊕ duration(tl ttl) end end

      ⊖: (Date×Time)×(Date×Time) → Days
      ⊕: Days × Days → Days
```

s102

6.2 Generation of Waybills

89. A well-formed shipping request (sid,fhi,thi,rid,fas,o,dt) in the context of a net, n,

90. and a set of transport companies' timetables, tts, now denotes, $\mathcal{M}$, a set, wbs, of n waybills: { wb_1, wb_2, ..., wb_i, ..., wb_n } where individual wb_is are of the form (sid,fhi,thi,rid,fas,ttl_i,c_i,sdt_i,rdt_i,dur_i)

91. which all satisfy wf_WB(sid,fhi,thi,rid,fas,ttl,c,sdt,rdt,dur)(n,tts).

```
  89   𝓜: Ship_Req → (Net × TTs) → WB-set
  90   𝓜(sid,fhi,thi,rid,fas,o,dt)(n,tts) as wbs
  89     pre: wf_Ship_Req(sid,fhi,thi,rid,fas,o,dt)(n)
  91     post: ∀ wb:WB • wb ∈ wbs ⇒ wf_WB(wb)(n,tts)
```

s103

92. The set of optimal waybills depend on the optimality criterion, o:
    (a) if o=`fastest` then the set of waybills with the same smallest duration, dur is chosen;

(b) if `o=cheapest` then the set of waybills with the same lowest cost, `c` is chosen; and
(c) if `o=earliest_arrival` then the set of waybills with the same earliest arrival date/time, `rdt` is chosen.

s104

```
92    optimal_WBs: WB-set → Optimality → WB-set
92    optimal_WBs(wbs)(o) ≡
92      {wb|wb:WB • wb ∈ wbs ⇒
92          let (sid,fhi,thi,rid,fas,ttl,c,sdt,rdt,dur) = wb in
92          ∼∃ wb′:(sid,fhi,thi,rid,fas,ttl,c′,sdt,rdt′,dur′):WB•wb′ ∈ wbs ∧
92            case o of
92(a)              fastest → dur′ ≺ dur,
92(b)              cheapest → c′ ≺ c,
92(c)              earliest_arrival → precede(rdt,rdt′)
92          end end}

    ≺: (Days×Days)|(Cost×Cost) → Bool
```

7 Logistics Traffic

s105

93. By logistics traffic, traf:TRAFFIC, we mean a continuous function from time to pairs of nets and vehicle positions.
94. That continuous function must satisfy some well-formedness conditions.

```
value
    n:N
type
    93  TRAFFIC′ = T → (N × (Vehicle ⇀ VLoc))
    94  TRAFFIC = {|tra:TRAFFIC′•wf_TRAFFIC(tra)(n)|}
```

s106

95. The well-formedness conditions for logistics traffics are:
    (a) If at two times, close to one another, a vehicle is in the traffic — at both of these times — then that vehicle is in the traffic at any time beween the two times.
    (b) At no time can two or more vehicles occupy the same location.
    (c) Et cetera.

s107

```
value
    95     wf_TRAFFIC: TRAFFIC → N → Bool
    95     wf_TRAFFIC(tra)(n) ≡
    95(a)    ∀ t,t′:T • {t,t′}⊆dom tra ∧ 0<t′−t<δ_T ⇒
    95(a)      ∀ v:Vehicle • v ∈ dom(tra(t))∩ dom(tra(t′)) ⇒
    95(a)        ∀ t″:T • t<t″<t′ • v ∈ dom(tra(t″)) ∧
    95(b)        ∀ v′:Vehicle • v≠v′ ∧ v′ ∈ dom(tra(t)) ⇒ (tra(t))(v)≠(tra(t))(v′) ∧
    95(c)    et cetera.
```

s108

8 Senders and Receivers

8.1 Senders

96.

97.

98.

99.

96
97
98
99

s109

8.2 Receivers

100.

101.

102.

103.

100
101
102
103

s110

9 Miscellaneous

104.

105.

106.

107.

s111

10 Model Extensions

s112

11 Logistics System Computing Functions

s113

12 Conclusion

s114

13 Bibliographical Notes

Specification languages, techniques and tools, that cover the spectrum of domain and requirements specification, refinement and verification, are dealt with in Alloy: [53], ASM: [70, 71], B/event B: [1,16], CafeOBJ: [18,19,32,33], CSP [48,49,73,74], DC [78,79] (Duration Calculus), Live Sequence Charts [17, 44, 55], Message Sequence Charts [50–52], RAISE [3–5, 34, 35, 37] (RSL), Petri nets [54, 65, 67–69], Statecharts [40–43, 45], Temporal Logic of Reactive Systems [58, 59, 64, 66], TLA+ [56, 57, 60, 61] (Temporal Logic of Actions), VDM [11, 12, 30, 31], and Z [46, 47, 75–77]. Techniques for integrating “different” formal techniques are covered in [2, 13, 14, 38, 72]. The recent book on Logics of Specification Languages [10] covers ASM, B/event B, CafeObj, CASL, DC, RAISE, TLA+, VDM and Z.

References

[1] Jean-Raymond Abrial. *The B Book: Assigning Programs to Meanings*. Tracts in Theoretical Computer Science. Cambridge University Press, Cambridge, England, 1996.

[2] Keijiro Araki, Andy Galloway, and Kenji Taguchi, editors. *IFM 1999: Integrated Formal Methods*, volume 1945 of *Lecture Notes in Computer Science*, York, UK, June 1999. Springer. Proceedings of 1st Intl. Conf. on IFM.

[3] Dines Bjørner. *Software Engineering, Vol. 1: Abstraction and Modelling*. Texts in Theoretical Computer Science, the EATCS Series. Springer, 2006.

[4] Dines Bjørner. *Software Engineering, Vol. 2: Specification of Systems and Languages*. Texts in Theoretical Computer Science, the EATCS Series. Springer, 2006. Chapters 12–14 are primarily authored by Christian Krog Madsen.

[5] Dines Bjørner. *Software Engineering, Vol. 3: Domains, Requirements and Software Design*. Texts in Theoretical Computer Science, the EATCS Series. Springer, 2006.

[6] Dines Bjørner. Domain Theory: Practice and Theories, Discussion of Possible Research Topics. In *ICTAC'2007*, volume 4701 of *Lecture Notes in Computer Science (eds. J.C.P. Woodcock et al.)*, pages 1–17, Heidelberg, September 2007. Springer.

[7] Dines Bjørner. From Domains to Requirements. In *Montanari Festschrift*, volume 5065 of *Lecture Notes in Computer Science (eds. Pierpaolo Degano, Rocco De Nicola and José Meseguer)*, pages 1–30, Heidelberg, May 2008. Springer.

[8] Dines Bjørner. Domain Engineering. In *BCS FACS Seminars*, Lecture Notes in Computer Science, the BCS FAC Series (eds. Paul Boca and Jonathan Bowen), pages 1–42, London, UK, 2009. Springer.

[9] Dines Bjørner. *Domain Engineering: Technology Management, Research and Engineering*. JAIST Press, March 2009. The monograph contains the following chapters: [20–29].

[10] Dines Bjørner and Martin C. Henson, editors. *Logics of Specification Languages — see [16, 19, 30, 34, 39, 46, 61, 62, 71]*. EATCS Monograph in Theoretical Computer Science. Springer, Heidelberg, Germany, 2008.

[11] Dines Bjørner and Cliff B. Jones, editors. *The Vienna Development Method: The Meta-Language*, volume 61 of *LNCS*. Springer–Verlag, 1978. This was the first monograph on *Meta-IV*. .

[12] Dines Bjørner and Cliff B. Jones, editors. *Formal Specification and Software Development*. Prentice-Hall, 1982.

[13] Eerke A. Boiten, John Derrick, and Graeme Smith, editors. *IFM 2004: Integrated Formal Methods*, volume 2999 of *Lecture Notes in Computer Science*, London, England, April 4-7 2004. Springer. Proceedings of 4th Intl. Conf. on IFM. ISBN 3-540-21377-5.

[14] Michael J. Butler, Luigia Petre, and Kaisa Sere, editors. *IFM 2002: Integrated Formal Methods*, volume 2335 of *Lecture Notes in Computer Science*, Turku, Finland, May 15-18 2002. Springer. Proceedings of 3rd Intl. Conf. on IFM. ISBN 3-540-43703-7.

[15] Dominique Cansell and Dominique Méry. Logical Foundations of the B Method. *Computing and Informatics*, 22(1–2), 2003. This paper is one of a series: [18, 36, 47, 60, 63, 70] appearing in a double issue of the same journal: *Logics of Specification Languages* — edited by Dines Bjørner.

[16] Dominique Cansell and Dominique Méry. *Logics of Specification Languages*, chapter The event-B Modelling Method: Concepts and Case Studies, pages 47–152 in [10]. Springer, 2008.

[17] Werner Damm and David Harel. LSCs: Breathing life into Message Sequence Charts. *Formal Methods in System Design*, 19:45–80, 2001. Early version appeared as Weizmann Institute Tech. Report CS98-09, April 1998. An abridged version appeared in *Proc. 3rd IFIP Int. Conf. on Formal Methods for Open Object-based Distributed Systems* (FMOODS'99), Kluwer, 1999, pp. 293–312.

[18] Ražvan Diaconescu, Kokichi Futatsugi, and Kazuhiro Ogata. CafeOBJ: Logical Foundations and Methodology. *Computing and Informatics*, 22(1–2), 2003. This paper is one of a series: [15, 36, 47, 60, 63, 70] appearing in a double issue of the same journal: *Logics of Specification Languages* — edited by Dines Bjørner.

[19] Răzvan Diaconescu. *Logics of Specification Languages*, chapter A Methodological Guide to the CafeOBJ Logic, pages 153–240 in [10]. Springer, 2008.

[20] Dines Bjørner. *Domain Engineering: Technology Management, Research and Engineering*, chapter 5: The Triptych Process Model – Process Assessment and Improvement, pages 107–138. JAIST Press, March 2009.

[21] Dines Bjørner. *Domain Engineering: Technology Management, Research and Engineering [9]*, chapter 9: Towards a Model of IT Security — – The ISO Information Security

Code of Practice – An Incomplete Rough Sketch Analysis, pages 223–282. JAIST Press, March 2009.

[22] Dines Bjørner. *Domain Engineering: Technology Management, Research and Engineering [9]*, chapter 1: On Domains and On Domain Engineering – Prerequisites for Trustworthy Software – A Necessity for Believable Management, pages 3–38. JAIST Press, March 2009.

[23] Dines Bjørner. *Domain Engineering: Technology Management, Research and Engineering [9]*, chapter 2: Possible Collaborative Domain Projects – A Management Brief, pages 39–56. JAIST Press, March 2009.

[24] Dines Bjørner. *Domain Engineering: Technology Management, Research and Engineering [9]*, chapter 3: The Rôle of Domain Engineering in Software Development, pages 57–72. JAIST Press, March 2009.

[25] Dines Bjørner. *Domain Engineering: Technology Management, Research and Engineering [9]*, chapter 4: Verified Software for Ubiquitous Computing – A VSTTE Ubiquitous Computing Project Proposal, pages 73–106. JAIST Press, March 2009.

[26] Dines Bjørner. *Domain Engineering: Technology Management, Research and Engineering [9]*, chapter 6: Domains and Problem Frames – The Triptych Dogma and M.A.Jackson's PF Paradigm, pages 139–175. JAIST Press, March 2009.

[27] Dines Bjørner. *Domain Engineering: Technology Management, Research and Engineering [9]*, chapter 7: Documents – A Rough Sketch Domain Analysis, pages 179–200. JAIST Press, March 2009.

[28] Dines Bjørner. *Domain Engineering: Technology Management, Research and Engineering [9]*, chapter 8: Public Government – A Rough Sketch Domain Analysis, pages 201–222. JAIST Press, March 2009.

[29] Dines Bjørner. *Domain Engineering: Technology Management, Research and Engineering [9]*, chapter 10: Towards a Family of Script Languages – – Licenses and Contracts – Incomplete Sketch, pages 283–328. JAIST Press, March 2009.

[30] John S. Fitzgerald. *Logics of Specification Languages*, chapter The Typed Logic of Partial Functions and the Vienna Development Method, pages 453–487 in [10]. Springer, 2008.

[31] John S. Fitzgerald and Peter Gorm Larsen. *Developing Software using `VDM-SL`*. Cambridge University Press, The Edinburgh Building, Cambridge CB2 1RU, England, 1997.

[32] K. Futatsugi, A.T. Nakagawa, and T. Tamai, editors. *CAFE: An Industrial–Strength Algebraic Formal Method*, Sara Burgerhartstraat 25, P.O. Box 211, NL–1000 AE Amsterdam, The Netherlands, 2000. Elsevier. Proceedings from an April 1998 Symposium, Numazu, Japan.

[33] Kokichi Futatsugi and Razvan Diaconescu. *CafeOBJ Report The Language, Proof Techniques, and Methodologies for Object-Oriented Algebraic Specification*. AMAST Series in Computing – Vol. 6. World Scientific Publishing Co. Pte. Ltd., 5 Toh Tuck Link, SINGAPORE 596224. Tel: 65-6466-5775, Fax: 65-6467-7667, E-mail: wspc@wspc.com.sg, 1998.

[34] Chris George and Anne E. Haxthausen. *Logics of Specification Languages*, chapter The Logic of the RAISE Specification Language, pages 349–399 in [10]. Springer, 2008.

[35] 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.

[36] Chris W. George and Anne E. Haxthausen. The Logic of the RAISE Specification Language. *Computing and Informatics*, 22(1–2), 2003. This paper is one of a series: [15, 18, 47, 60, 63, 70] appearing in a double issue of the same journal: *Logics of Specification Languages* — edited by Dines Bjørner.

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

[38] Wolfgang Grieskamp, Thomas Santen, and Bill Stoddart, editors. *IFM 2000: Integrated Formal Methods*, volume of *Lecture Notes in Computer Science*, Schloss Dagstuhl, Germany, November 1-3 2000. Springer. Proceedings of 2nd Intl. Conf. on IFM.

[39] Michael R. Hansen. *Logics of Specification Languages*, chapter Duration Calculus, pages 299–347 in [10]. Springer, 2008.

[40] David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, 1987.

[41] David Harel. On visual formalisms. *Communications of the ACM*, 33(5), 514–530 1988.

[42] David Harel and Eran Gery. Executable object modeling with Statecharts. *IEEE Computer*, 30(7):31–42, 1997.

[43] David Harel, Hagi Lachover, Amnon Naamad, Amir Pnueli, Michal Politi, Rivi Sherman, Aharon Shtull-Trauring, and Mark B. Trakhtenbrot. STATEMATE: A working environment for the development of complex reactive systems. *Software Engineering*, 16(4):403–414, 1990.

[44] David Harel and Rami Marelly. *Come, Let's Play – Scenario-Based Programming Using LSCs and the Play-Engine*. Springer-Verlag, 2003.

[45] David Harel and Amnon Naamad. The STATEMATE semantics of Statecharts. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 5(4):293–333, 1996.

[46] Martin C. Henson, Moshe Deutsch, and Steve Reeves. *Logics of Specification Languages*, chapter Z Logic and Its Applications, pages 489–596 in [10]. Springer, 2008.

[47] Martin C. Henson, Steve Reeves, and Jonathan P. Bowen. Z Logic and its Consequences. *Computing and Informatics*, 22(1–2), 2003. This paper is one of a series: [15, 18, 36, 60, 63, 70] appearing in a double issue of the same journal: *Logics of Specification Languages* — edited by Dines Bjørner.

[48] Tony Hoare. *Communicating Sequential Processes*. C.A.R. Hoare Series in Computer Science. Prentice-Hall International, 1985.

[49] Tony Hoare. Communicating Sequential Processes. Published electronically: `http://www.usingcsp.com/cspbook.pdf`, 2004. Second edition of [48]. See also `http://www.usingcsp.com/`.

[50] ITU-T. CCITT Recommendation Z.120: Message Sequence Chart (MSC), 1992.

[51] ITU-T. ITU-T Recommendation Z.120: Message Sequence Chart (MSC), 1996.

[52] ITU-T. ITU-T Recommendation Z.120: Message Sequence Chart (MSC), 1999.

[53] Daniel Jackson. *Software Abstractions Logic, Language, and Analysis*. The MIT Press, Cambridge, Mass., USA, April 2006. ISBN 0-262-10114-9.

[54] Kurt Jensen. *Coloured Petri Nets*, volume 1: Basic Concepts (234 pages + xii), Vol. 2: Analysis Methods (174 pages + x), Vol. 3: Practical Use (265 pages + xi) of *EATCS Monographs in Theoretical Computer Science*. Springer–Verlag, Heidelberg, 1985, revised and corrected second version: 1997.

[55] Jochen Klose and Hartmut Wittke. An automata based interpretation of Live Sequence Charts. In T. Margaria and W. Yi, editors, *TACAS 2001*, LNCS 2031, pages 512–527. Springer-Verlag, 2001.

[56] Leslie Lamport. The Temporal Logic of Actions. *Transactions on Programming Languages and Systems*, 16(3):872–923, 1995.

[57] Leslie Lamport. *Specifying Systems*. Addison–Wesley, Boston, Mass., USA, 2002.

[58] Zohar Manna and Amir Pnueli. *The Temporal Logic of Reactive Systems: Specifications*. Addison Wesley, 1991.

[59] Zohar Manna and Amir Pnueli. *The Temporal Logic of Reactive Systems: Safety*. Addison Wesley, 1995.

[60] Stephan Merz. On the Logic of TLA+. *Computing and Informatics*, 22(1–2), 2003. This paper is one of a series: [15, 18, 36, 47, 63, 70] appearing in a double issue of the same journal: *Logics of Specification Languages* — edited by Dines Bjørner.

[61] Stephan Merz. *Logics of Specification Languages*, chapter The Specification Language TLA$^+$, pages 401–451 in [10]. Springer, 2008.

[62] T. Mossakowski, A. Haxthausen, D. Sannella, and A. Tarlecki. *Logics of Specification Languages*, chapter CASL – the Common Algebraic Specification Language, pages 241–298 in [10]. Springer, 2008.

[63] Till Mossakowski, Anne E. Haxthausen, Don Sanella, and Andrzej Tarlecki. CASL — The Common Algebraic Specification Language: Semantics and Proof Theory. *Computing and Informatics*, 22(1–2), 2003. This paper is one of a series: [15, 18, 36, 47, 60, 70] appearing in a double issue of the same journal: *Logics of Specification Languages* — edited by Dines Bjørner.

[64] Ben C. Moszkowski. *Executing Temporal Logic Programs*. Cambridge University Press, Cambridge, England, 1986.

[65] Carl Adam Petri. *Kommunikation mit Automaten*. Bonn: Institut für Instrumentelle Mathematik, Schriften des IIM Nr. 2, 1962.

[66] Amir Pnueli. The Temporal Logic of Programs. In *Proceedings of the 18th IEEE Symposium on Foundations of Computer Science*, IEEE CS FoCS, pages 46–57. Providence, Rhode Island, IEEE CS, 1977. .

[67] Wolfang Reisig. *A Primer in Petri Net Design*. Springer Verlag, March 1992. 120 pages.

[68] Wolfgang Reisig. *Petri Nets: An Introduction*, volume 4 of *EATCS Monographs in Theoretical Computer Science*. Springer Verlag, May 1985.

[69] Wolfgang Reisig. *Elements of Distributed Algorithms: Modelling and Analysis with Petri Nets*. Springer Verlag, December 1998. xi + 302 pages.

[70] Wolfgang Reisig. The Expressive Power of Abstract State Machines. *Computing and Informatics*, 22(1–2), 2003. This paper is one of a series: [15, 18, 36, 47, 60, 63] appearing in a double issue of the same journal: *Logics of Specification Languages* — edited by Dines Bjørner.

[71] Wolfgang Reisig. *Logics of Specification Languages*, chapter Abstract State Machines for the Classroom, pages 15–46 in [10]. Springer, 2008.

[72] Judi M.T. Romijn, Graeme P. Smith, and Jaco C. van de Pol, editors. *IFM 2005: Integrated Formal Methods*, volume 3771 of *Lecture Notes in Computer Science*, Eindhoven, The Netherlands, December 2005. Springer. Proceedings of 5th Intl. Conf. on IFM. ISBN 3-540-30492-4.

[73] A. W. Roscoe. *Theory and Practice of Concurrency*. C.A.R. Hoare Series in Computer Science. Prentice-Hall, 1997. Now available on the net: http://www.comlab.ox.ac.uk/-people/bill.roscoe/publications/68b.pdf.

[74] Steve Schneider. *Concurrent and Real-time Systems — The CSP Approach*. Worldwide Series in Computer Science. John Wiley & Sons, Ltd., Baffins Lane, Chichester, West Sussex PO19 1UD, England, January 2000.

[75] J. M. Spivey. *Understanding Z: A Specification Language and its Formal Semantics*, volume 3 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, January 1988.

[76] J. M. Spivey. *The Z Notation: A Reference Manual*. Prentice Hall International Series in Computer Science, 2nd edition, 1992.

[77] J. C. P. Woodcock and J. Davies. *Using Z: Specification, Proof and Refinement*. Prentice Hall International Series in Computer Science, 1996.

[78] Chao Chen Zhou and Michael R. Hansen. *Duration Calculus: A Formal Approach to Real–time Systems*. Monographs in Theoretical Computer Science. An EATCS Series. Springer–Verlag, 2004.

[79] Chao Chen Zhou, Charles Anthony Richard Hoare, and Anders P. Ravn. A Calculus of Durations. *Information Proc. Letters*, 40(5), 1992.

A An RSL Primer s115

This is an ultra-short introduction to the `RAISE` Specification Language, `RSL`.

A.1 Types

The reader is kindly asked to study first the decomposition of this section into its sub-parts and sub-sub-parts.

A.1.1 Type Expressions

Type expressions are expressions whose value are type, that is, possibly infinite sets of values (of "that" type).

Atomic Types Atomic types have (atomic) values. That is, values which we consider to have no proper constituent (sub-)values, i.e., cannot, to us, be meaningfully "taken apart".

`RSL` has a number of *built-in* atomic types. There are the Booleans, integers, natural numbers, reals, characters, and texts. s116

Basic Types

type
[1] **Bool**
[2] **Int**
[3] **Nat**
[4] **Real**
[5] **Char**
[6] **Text**

Composite Types Composite types have composite values. That is, values which we consider to have proper constituent (sub-)values, i.e., can, to us, be meaningfully "taken apart". s117

From these one can form type expressions: finite sets, infinite sets, Cartesian products, lists, maps, etc.

Let A, B and C be any type names or type expressions, then:

Composite Type Expressions

[7] A-**set**
[8] A-**infset**
[9] $A \times B \times \ldots \times C$
[10] A^*
[11] A^ω
[12] $A \underset{m}{\rightarrow} B$
[13] $A \rightarrow B$
[14] $A \overset{\sim}{\rightarrow} B$
[15] (A)
[16] A | B | ... | C

```
[17] mk_id(sel_a:A,...,sel_b:B)
[18] sel_a:A ... sel_b:B
```

The following are generic type expressions:

1. The Boolean type of truth values **false** and **true**.
2. The integer type on integers ..., –2, –1, 0, 1, 2, ...
3. The natural number type of positive integer values 0, 1, 2, ...
4. The real number type of real values, i.e., values whose numerals can be written as an integer, followed by a period ("."), followed by a natural number (the fraction).
5. The character type of character values ″a″, ″b″, ...
6. The text type of character string values ″aa″, ″aaa″, ..., ″abc″, ...
7. The set type of finite cardinality set values.
8. The set type of infinite and finite cardinality set values.
9. The Cartesian type of Cartesian values.
10. The list type of finite length list values.
11. The list type of infinite and finite length list values.
12. The map type of finite definition set map values.
13. The function type of total function values.
14. The function type of partial function values.
15. In (A) A is constrained to be:
    - either a Cartesian B × C × ... × D, in which case it is identical to type expression kind 9,
    - or not to be the name of a built-in type (cf., 1–6) or of a type, in which case the parentheses serve as simple delimiters, e.g., (A $\overrightarrow{m}$ B), or (A*)-**set**, or (A-**set**)list, or (A|B) $\overrightarrow{m}$ (C|D|(E $\overrightarrow{m}$ F)), etc.
16. The postulated disjoint union of types A, B, . . . , and C.
17. The record type of mk_id-named record values mk_id(av,...,bv), where av, . . . , bv, are values of respective types. The distinct identifiers sel_a, etc., designate selector functions.
18. The record type of unnamed record values (av,...,bv), where av, . . . , bv, are values of respective types. The distinct identifiers sel_a, etc., designate selector functions.

A.1.2 Type Definitions s118

Concrete Types Types can be concrete in which case the structure of the type is specified by type expressions:

Type Definition

```
type
   A = Type_expr
```

s119 Some schematic type definitions are:

Variety of Type Definitions

```
[1]  Type_name = Type_expr /∗ without |s or subtypes ∗/
[2]  Type_name = Type_expr_1 | Type_expr_2 | ... | Type_expr_n
[3]  Type_name ==
          mk_id_1(s_a1:Type_name_a1,...,s_ai:Type_name_ai) |
          ... |
          mk_id_n(s_z1:Type_name_z1,...,s_zk:Type_name_zk)
[4]  Type_name :: sel_a:Type_name_a  ...  sel_z:Type_name_z
[5]  Type_name = {| v:Type_name′ • 𝒫(v) |}
```

s120 where a form of [2–3] is provided by combining the types:

Record Types

```
   Type_name = A | B | ... | Z
   A == mk_id_1(s_a1:A_1,...,s_ai:A_i)
   B == mk_id_2(s_b1:B_1,...,s_bj:B_j)
   ...
   Z == mk_id_n(s_z1:Z_1,...,s_zk:Z_k)
```

Types A, B, ..., Z are disjoint, i.e., shares no values, provided all mk_id_k are distinct and due to the use of the disjoint record type constructor ==.

```
axiom
   ∀ a1:A_1, a2:A_2, ..., ai:Ai •
      s_a1(mk_id_1(a1,a2,...,ai))=a1 ∧ s_a2(mk_id_1(a1,a2,...,ai))=a2 ∧
      ... ∧ s_ai(mk_id_1(a1,a2,...,ai))=ai ∧
   ∀ a:A • let mk_id_1(a1′,a2′,...,ai′) = a in
      a1′ = s_a1(a) ∧ a2′ = s_a2(a) ∧ ... ∧ ai′ = s_ai(a) end
```

s121

Subtypes In `RSL`, each type represents a set of values. Such a set can be delimited by means of predicates. The set of values b which have type B and which satisfy the predicate $\mathcal{P}$, constitute the subtype A:

Subtypes

```
type
   A = {| b:B • 𝒫(b) |}
```

s122

Sorts — Abstract Types Types can be (abstract) sorts in which case their structure is not specified:

Sorts

```
type
   A, B, ..., C
```

A.2 The RSL Predicate Calculus s123

A.2.1 Propositional Expressions

Let identifiers (or propositional expressions) a, b, ..., c designate Boolean values (**true** or **false** [or **chaos**]). Then:

Propositional Expressions

```
   false, true
   a, b, ..., c ∼a, a∧b, a∨b, a⇒b, a=b, a≠b
```

are propositional expressions having Boolean values. ∼, ∧, ∨, ⇒, = and ≠ are Boolean connectives (i.e., operators). They can be read as: *not*, *and*, *or*, *if then* (or *implies*), *equal* and *not equal*. s124

A.2.2 Simple Predicate Expressions

Let identifiers (or propositional expressions) a, b, ..., c designate Boolean values, let x, y, ..., z (or term expressions) designate non-Boolean values and let i, j, . . ., k designate number values, then:

Simple Predicate Expressions

```
   false, true
   a, b, ..., c
   ∼a, a∧b, a∨b, a⇒b, a=b, a≠b
   x=y, x≠y,
   i<j, i≤j, i≥j, i≠j, i≥j, i>j
```

s125 are simple predicate expressions.

A.3 Quantified Expressions

Let X, Y, …, C be type names or type expressions, and let $\mathcal{P}(x)$, $\mathcal{Q}(y)$ and $\mathcal{R}(z)$ designate predicate expressions in which x, y and z are free. Then:

Quantified Expressions

$\forall$ x:X • $\mathcal{P}(x)$
$\exists$ y:Y • $\mathcal{Q}(y)$
$\exists$! z:Z • $\mathcal{R}(z)$

are quantified expressions — also being predicate expressions.

They are "read" as: For all x (values in type X) the predicate $\mathcal{P}(x)$ holds; there exists (at least) one y (value in type Y) such that the predicate $\mathcal{Q}(y)$ holds; and there exists a unique z (value in type Z) such that the predicate $\mathcal{R}(z)$ holds.

s126 A.4 Concrete `RSL` Types: Values and Operations

A.4.1 Arithmetic

Arithmetic

type
  Nat, **Int**, **Real**
value
  $+,-,*$: **Nat**×**Nat**→**Nat** | **Int**×**Int**→**Int** | **Real**×**Real**→**Real**
  /: **Nat**×**Nat**$\overset{\sim}{\to}$**Nat** | **Int**×**Int**$\overset{\sim}{\to}$**Int** | **Real**×**Real**$\overset{\sim}{\to}$**Real**
  $<,\leq,=,\neq,\geq,>$ (**Nat**|**Int**|**Real**) → (**Nat**|**Int**|**Real**)

s127

A.4.2 Set Expressions

Set Enumerations Let the below a's denote values of type A, then the below designate simple set enumerations:

Set Enumerations

{{}, {a}, {e_1,e_2,...,e_n}, ...} ∈ A-**set**
{{}, {a}, {e_1,e_2,...,e_n}, ..., {e_1,e_2,...}} ∈ A-**infset**

s128

Set Comprehension The expression, last line below, to the right of the ≡, expresses set comprehension. The expression "builds" the set of values satisfying the given predicate. It is abstract in the sense that it does not do so by following a concrete algorithm.

Set Comprehension

```
type
    A, B
    P = A → Bool
    Q = A →̃ B
value
    comprehend: A-infset × P × Q → B-infset
    comprehend(s,P,Q) ≡ { Q(a) | a:A • a ∈ s ∧ P(a)}
```

s129

A.4.3 Cartesian Expressions

Cartesian Enumerations Let e range over values of Cartesian types involving A, B, $\ldots$, C, then the below expressions are simple Cartesian enumerations:

Cartesian Enumerations

```
type
    A, B, ..., C
    A × B × ... × C
value
    (e1,e2,...,en)
```

s130

A.4.4 List Expressions

List Enumerations Let a range over values of type A, then the below expressions are simple list enumerations:

List Enumerations

```
    {⟨⟩, ⟨e⟩, ..., ⟨e1,e2,...,en⟩, ...} ∈ A*
    {⟨⟩, ⟨e⟩, ..., ⟨e1,e2,...,en⟩, ..., ⟨e1,e2,...,en,... ⟩, ...} ∈ Aω

    ⟨ a_i .. a_j ⟩
```

The last line above assumes a_i and a_j to be integer-valued expressions. It then expresses the set of integers from the value of e_i to and including the value of e_j. If the latter is smaller than the former, then the list is empty. s131

List Comprehension The last line below expresses list comprehension.

List Comprehension

```
type
```

```
    A, B, P = A → Bool, Q = A →̃ B
value
    comprehend: A^ω × P × Q →̃ B^ω
    comprehend(l,P,Q) ≡
        ⟨ Q(l(i)) | i in ⟨1..len l⟩ • P(l(i))⟩
```

s132

A.4.5 Map Expressions

Map Enumerations Let (possibly indexed) u and v range over values of type $T1$ and $T2$, respectively, then the below expressions are simple map enumerations:

Map Enumerations

```
type
    T1, T2
    M = T1 →ₘ T2
value
    u,u1,u2,...,un:T1, v,v1,v2,...,vn:T2
    [ ], [u↦v], ..., [u1↦v1,u2↦v2,...,un↦vn] ∀ ∈ M
```

s133

Map Comprehension The last line below expresses map comprehension:

Map Comprehension

```
type
    U, V, X, Y
    M = U →ₘ V
    F = U →̃ X
    G = V →̃ Y
    P = U → Bool
value
    comprehend: M×F×G×P → (X →ₘ Y)
    comprehend(m,F,G,P) ≡
        [ F(u) ↦ G(m(u)) | u:U • u ∈ dom m ∧ P(u) ]
```

s134

A.4.6 Set Operations

Set Operator Signatures Quite a set !

Set Operations

```
value
    19 ∈: A × A-infset → Bool
    20 ∉: A × A-infset → Bool
```

21 $\cup$: A-**infset** $\times$ A-**infset** $\rightarrow$ A-**infset**
22 $\cup$: (A-**infset**)-**infset** $\rightarrow$ A-**infset**
23 $\cap$: A-**infset** $\times$ A-**infset** $\rightarrow$ A-**infset**
24 $\cap$: (A-**infset**)-**infset** $\rightarrow$ A-**infset**
25 $\setminus$: A-**infset** $\times$ A-**infset** $\rightarrow$ A-**infset**
26 $\subset$: A-**infset** $\times$ A-**infset** $\rightarrow$ **Bool**
27 $\subseteq$: A-**infset** $\times$ A-**infset** $\rightarrow$ **Bool**
28 $=$: A-**infset** $\times$ A-**infset** $\rightarrow$ **Bool**
29 $\neq$: A-**infset** $\times$ A-**infset** $\rightarrow$ **Bool**
30 **card**: A-**infset** $\overset{\sim}{\rightarrow}$ **Nat**

s135

Set Examples

For your enlightment !

Set Examples

examples
$a \in \{a,b,c\}$
$a \notin \{\}$, $a \notin \{b,c\}$
$\{a,b,c\} \cup \{a,b,d,e\} = \{a,b,c,d,e\}$
$\cup\{\{a\},\{a,b\},\{a,d\}\} = \{a,b,d\}$
$\{a,b,c\} \cap \{c,d,e\} = \{c\}$
$\cap\{\{a\},\{a,b\},\{a,d\}\} = \{a\}$
$\{a,b,c\} \setminus \{c,d\} = \{a,b\}$
$\{a,b\} \subset \{a,b,c\}$
$\{a,b,c\} \subseteq \{a,b,c\}$
$\{a,b,c\} = \{a,b,c\}$
$\{a,b,c\} \neq \{a,b\}$
card $\{\} = 0$, **card** $\{a,b,c\} = 3$

s136

Informal Explication

19. $\in$: The membership operator expresses that an element is a member of a set.
20. $\notin$: The nonmembership operator expresses that an element is not a member of a set.
21. $\cup$: The infix union operator. When applied to two sets, the operator gives the set whose members are in either or both of the two operand sets.
22. $\cup$: The distributed prefix union operator. When applied to a set of sets, the operator gives the set whose members are in some of the operand sets.
23. $\cap$: The infix intersection operator. When applied to two sets, the operator gives the set whose members are in both of the two operand sets.
24. $\cap$: The prefix distributed intersection operator. When applied to a set of sets, the operator gives the set whose members are in some of the operand sets. s137
25. $\setminus$: The set complement (or set subtraction) operator. When applied to two sets, the operator gives the set whose members are those of the left operand set which are not in the right operand set.
26. $\subseteq$: The proper subset operator expresses that all members of the left operand set are also in the right operand set.

27. ⊂: The proper subset operator expresses that all members of the left operand set are also in the right operand set, and that the two sets are not identical.
28. =: The equal operator expresses that the two operand sets are identical.
29. ≠: The nonequal operator expresses that the two operand sets are *not* identical.
30. **card**: The cardinality operator gives the number of elements in a finite set.

s138

Set Operator Definitions The operations can be defined as follows (≡ is the definition symbol):

Set Operation Definitions

```
value
   s′ ∪ s″ ≡ { a | a:A • a ∈ s′ ∨ a ∈ s″ }
   s′ ∩ s″ ≡ { a | a:A • a ∈ s′ ∧ a ∈ s″ }
   s′ \ s″ ≡ { a | a:A • a ∈ s′ ∧ a ∉ s″ }
   s′ ⊆ s″ ≡ ∀ a:A • a ∈ s′ ⇒ a ∈ s″
   s′ ⊂ s″ ≡ s′ ⊆ s″ ∧ ∃ a:A • a ∈ s″ ∧ a ∉ s′
   s′ = s″ ≡ ∀ a:A • a ∈ s′ ≡ a ∈ s″ ≡ s⊆s′ ∧ s′⊆s
   s′ ≠ s″ ≡ s′ ∩ s″ ≠ {}
   card s ≡
      if s = {} then 0 else
      let a:A • a ∈ s in 1 + card (s \ {a}) end end
      pre s /∗ is a finite set ∗/
   card s ≡ chaos /∗ tests for infinity of s ∗/
```

s139

A.5 Cartesian Operations

Cartesian Operations

```
type
   A, B, C
   g0: G0 = A × B × C
   g1: G1 = ( A × B × C )
   g2: G2 = ( A × B ) × C
   g3: G3 = A × ( B × C )

value
   va:A, vb:B, vc:C, vd:D
   (va,vb,vc):G0,
   (va,vb,vc):G1
   ((va,vb),vc):G2
   (va3,(vb3,vc3)):G3

decomposition expressions
   let (a1,b1,c1) = g0,
         (a1′,b1′,c1′) = g1 in .. end
   let ((a2,b2),c2) = g2 in .. end
   let (a3,(b3,c3)) = g3 in .. end
```

s140

A.5.1 List Operations

List Operator Signatures Also quite a few:

List Operations

value
  hd: $A^\omega \overset{\sim}{\rightarrow} A$
  tl: $A^\omega \overset{\sim}{\rightarrow} A^\omega$
  len: $A^\omega \overset{\sim}{\rightarrow}$ **Nat**
  inds: $A^\omega \rightarrow$ **Nat-infset**
  elems: $A^\omega \rightarrow$ A-**infset**
  .(.): $A^\omega \times$ **Nat** $\overset{\sim}{\rightarrow} A$
  $\widehat{\ }$: $A^* \times A^\omega \rightarrow A^\omega$
  $=$: $A^\omega \times A^\omega \rightarrow$ **Bool**
  $\neq$: $A^\omega \times A^\omega \rightarrow$ **Bool**

s141

List Operation Examples We continue:

List Examples

examples
  hd$\langle a1,a2,...,am\rangle = a1$
  tl$\langle a1,a2,...,am\rangle = \langle a2,...,am\rangle$
  len$\langle a1,a2,...,am\rangle = m$
  inds$\langle a1,a2,...,am\rangle = \{1,2,...,m\}$
  elems$\langle a1,a2,...,am\rangle = \{a1,a2,...,am\}$
  $\langle a1,a2,...,am\rangle(i) = ai$
  $\langle a,b,c\rangle \widehat{\ } \langle a,b,d\rangle = \langle a,b,c,a,b,d\rangle$
  $\langle a,b,c\rangle = \langle a,b,c\rangle$
  $\langle a,b,c\rangle \neq \langle a,b,d\rangle$

s142

Informal Explication

- **hd**: Head gives the first element in a nonempty list.
- **tl**: Tail gives the remaining list of a nonempty list when Head is removed.
- **len**: Length gives the number of elements in a finite list.
- **inds**: Indices give the set of indices from 1 to the length of a nonempty list. For empty lists, this set is the empty set as well.
- **elems**: Elements gives the possibly infinite set of all distinct elements in a list.
- $\ell(i)$: Indexing with a natural number, i larger than 0, into a list ℓ having a number of elements larger than or equal to i, gives the ith element of the list. s143
- $\widehat{\ }$: Concatenates two operand lists into one. The elements of the left operand list are followed by the elements of the right. The order with respect to each list is maintained.
- $=$: The equal operator expresses that the two operand lists are identical.

- ≠: The nonequal operator expresses that the two operand lists are *not* identical.

s144 The operations can also be defined as follows:

List Operator Definitions These are informal definitions !

List Operator Definitions

```
value
   is_finite_list: A^ω → Bool

   len q ≡
      case is_finite_list(q) of
         true → if q = ⟨⟩ then 0 else 1 + len tl q end,
         false → chaos end

   inds q ≡
      case is_finite_list(q) of
         true → { i | i:Nat • 1 ≤ i ≤ len q },
         false → { i | i:Nat • i≠0 } end

elems q ≡ { q(i) | i:Nat • i ∈ inds q }

q(i) ≡
      if i=1
         then
            if q≠⟨⟩
               then let a:A,q′:Q • q=⟨a⟩⌒q′ in a end
               else chaos end
         else q(i−1) end

fq ⌒ iq ≡
         ⟨ if 1 ≤ i ≤ len fq then fq(i) else iq(i − len fq) end
          | i:Nat • if len iq≠chaos then i ≤ len fq+len end ⟩
      pre is_finite_list(fq)

   iq′ = iq″ ≡
      inds iq′ = inds iq″ ∧ ∀ i:Nat • i ∈ inds iq′ ⇒ iq′(i) = iq″(i)

   iq′ ≠ iq″ ≡ ∼(iq′ = iq″)
```

s145

A.5.2 Map Operations

Map Operator Signatures and Map Operation Examples This time we combine the two.

Map Operations and Examples

```
value
```

$m(a)\colon\ M \rightarrow A \overset{\sim}{\rightarrow} B,\ m(a) = b$

$\textbf{dom}\colon\ M \rightarrow A\text{-}\textbf{infset}\ [\text{domain of map}]$
$\quad \textbf{dom}\ [a1 \mapsto b1, a2 \mapsto b2, \ldots, an \mapsto bn] = \{a1, a2, \ldots, an\}$

$\textbf{rng}\colon\ M \rightarrow B\text{-}\textbf{infset}\ [\text{range of map}]$
$\quad \textbf{rng}\ [a1 \mapsto b1, a2 \mapsto b2, \ldots, an \mapsto bn] = \{b1, b2, \ldots, bn\}$

$\dagger\colon\ M \times M \rightarrow M\ [\text{override extension}]$
$\quad [a \mapsto b, a' \mapsto b', a'' \mapsto b''] \dagger [a' \mapsto b'', a'' \mapsto b'] = [a \mapsto b, a' \mapsto b'', a'' \mapsto b']$

$\cup\colon\ M \times M \rightarrow M\ [\text{merge } \cup]$
$\quad [a \mapsto b, a' \mapsto b', a'' \mapsto b''] \cup [a''' \mapsto b'''] = [a \mapsto b, a' \mapsto b', a'' \mapsto b'', a''' \mapsto b''']$

$\backslash\colon\ M \times A\text{-}\textbf{infset} \rightarrow M\ [\text{restriction by}]$
$\quad [a \mapsto b, a' \mapsto b', a'' \mapsto b''] \backslash \{a\} = [a' \mapsto b', a'' \mapsto b'']$

$/\colon\ M \times A\text{-}\textbf{infset} \rightarrow M\ [\text{restriction to}]$
$\quad [a \mapsto b, a' \mapsto b', a'' \mapsto b''] / \{a', a''\} = [a' \mapsto b', a'' \mapsto b'']$

$=, \neq\colon\ M \times M \rightarrow \textbf{Bool}$

$^\circ\colon\ (A \overrightarrow{_m} B) \times (B \overrightarrow{_m} C) \rightarrow (A \overrightarrow{_m} C)\ [\text{composition}]$
$\quad [a \mapsto b, a' \mapsto b'] \circ [b \mapsto c, b' \mapsto c', b'' \mapsto c''] = [a \mapsto c, a' \mapsto c']$

s146

Map Operation Explication

- $m(a)$: Application gives the element that a maps to in the map m.
- **dom**: Domain/Definition Set gives the set of values which *maps to* in a map.
- **rng**: Range/Image Set gives the set of values which *are mapped to* in a map.
- †: Override/Extend. When applied to two operand maps, it gives the map which is like an override of the left operand map by all or some "pairings" of the right operand map.
- ∪: Merge. When applied to two operand maps, it gives a merge of these maps. s147
- \: Restriction. When applied to two operand maps, it gives the map which is a restriction of the left operand map to the elements that are not in the right operand set.
- /: Restriction. When applied to two operand maps, it gives the map which is a restriction of the left operand map to the elements of the right operand set.
- =: The equal operator expresses that the two operand maps are identical.
- ≠: The nonequal operator expresses that the two operand maps are *not* identical.

- °: Composition. When applied to two operand maps, it gives the map from definition set elements of the left operand map, m_1, to the range elements of the right operand map, m_2, such that if a is in the definition set of m_1 and maps into b, and if b is in the definition set of m_2 and maps into c, then a, in the composition, maps into c.

s148

Map Operation Redefinitions The map operations can also be defined as follows:

Map Operation Redefinitions

```
value
   rng m ≡ { m(a) | a:A • a ∈ dom m }

   m1 † m2 ≡
      [ a↦b | a:A,b:B •
         a ∈ dom m1 \ dom m2 ∧ b=m1(a) ∨ a ∈ dom m2 ∧ b=m2(a) ]

   m1 ∪ m2 ≡ [ a↦b | a:A,b:B •
                 a ∈ dom m1 ∧ b=m1(a) ∨ a ∈ dom m2 ∧ b=m2(a) ]

   m \ s ≡ [ a↦m(a) | a:A • a ∈ dom m \ s ]
   m / s ≡ [ a↦m(a) | a:A • a ∈ dom m ∩ s ]

   m1 = m2 ≡
      dom m1 = dom m2 ∧ ∀ a:A • a ∈ dom m1 ⇒ m1(a) = m2(a)
   m1 ≠ m2 ≡ ∼(m1 = m2)

   m°n ≡
      [ a↦c | a:A,c:C • a ∈ dom m ∧ c = n(m(a)) ]
      pre rng m ⊆ dom n
```

s149 A.6 λ-Calculus + Functions

A.6.1 The λ-Calculus Syntax

λ-Calculus Syntax

```
type /* A BNF Syntax: */
    ⟨L⟩ ::= ⟨V⟩ | ⟨F⟩ | ⟨A⟩ | ( ⟨A⟩ )
    ⟨V⟩ ::= /* variables, i.e. identifiers */
    ⟨F⟩ ::= λ⟨V⟩ • ⟨L⟩
    ⟨A⟩ ::= ( ⟨L⟩⟨L⟩ )
value /* Examples */
    ⟨L⟩: e, f, a, ...
    ⟨V⟩: x, ...
```

$\langle$F$\rangle$: λ x • e, ...
$\langle$A$\rangle$: f a, (f a), f(a), (f)(a), ...

s150

A.6.2 Free and Bound Variables

Free and Bound Variables

Let x, y be variable names and e, f be λ-expressions.

- $\langle$V$\rangle$: Variable x is free in x.
- $\langle$F$\rangle$: x is free in $\lambda y \bullet e$ if $x \neq y$ and x is free in e.
- $\langle$A$\rangle$: x is free in $f(e)$ if it is free in either f or e (i.e., also in both).

s151

A.6.3 Substitution

In `RSL`, the following rules for substitution apply:

Substitution

- **subst**([N/x]x) $\equiv$ N;
- **subst**([N/x]a) $\equiv$ a,

  for all variables a$\neq$ x;
- **subst**([N/x](P Q)) $\equiv$ (**subst**([N/x]P) **subst**([N/x]Q));
- **subst**([N/x]($\lambda x \bullet P$)) $\equiv$ λ y•P;
- **subst**([N/x](λ y•P)) $\equiv$ $\lambda y \bullet$ **subst**([N/x]P),

  if x$\neq$y and y is not free in N or x is not free in P;
- **subst**([N/x](λy•P)) $\equiv$ λz•**subst**([N/z]**subst**([z/y]P)),

  if y$\neq$x and y is free in N and x is free in P
  (where z is not free in (N P)).

s152

A.6.4 α-Renaming and β-Reduction

α and β Conversions

- α-renaming: λx•M

  If x, y are distinct variables then replacing x by y in λx•M results in λy•**subst**([y/x]M). We can rename the formal parameter of a λ-function expression provided that no free variables of its body M thereby become bound.

- β-reduction: (λx•M)(N)

  All free occurrences of x in M are replaced by the expression N provided that no free variables of N thereby become bound in the result. (λx•M)(N) $\equiv$ **subst**([N/x]M)

s153

A.6.5 Function Signatures

For sorts we may want to postulate some functions:

Sorts and Function Signatures

```
type
    A, B, C
value
    obs_B: A → B,
    obs_C: A → C,
    gen_A: B×C → A
```

s154

A.6.6 Function Definitions

Functions can be defined explicitly:

Explicit Function Definitions

```
value
    f: Arguments → Result
    f(args) ≡ DValueExpr

    g: Arguments ⥲ Result
    g(args) ≡ ValueAndStateChangeClause
    pre P(args)
```

s155 Or functions can be defined implicitly:

Implicit Function Definitions

```
value
    f: Arguments → Result
    f(args) as result
    post P1(args,result)

    g: Arguments ⥲ Result
    g(args) as result
    pre P2(args)
    post P3(args,result)
```

The symbol $\overset{\sim}{\rightarrow}$ indicates that the function is partial and thus not defined for all arguments. Partial functions should be assisted by preconditions stating the criteria for arguments to be meaningful to the function.

A.7 Other Applicative Expressions

s156

A.7.1 Simple let Expressions

Simple (i.e., nonrecursive) **let** expressions:

Let Expressions

let a = $\mathcal{E}_d$ **in** $\mathcal{E}_b$(a) **end**

is an “expanded” form of:

$(\lambda a.\mathcal{E}_b(a))(\mathcal{E}_d)$

s157

A.7.2 Recursive let Expressions

Recursive **let** expressions are written as:

Recursive **let** Expressions

let f = λa:A • E(f) **in** B(f,a) **end**

is “the same” as:

let f = **Y**F **in** B(f,a) **end**

where:

F ≡ λg•λa•(E(g)) and YF = F(YF)

s158

A.7.3 Predicative let Expressions

Predicative **let** expressions:

Predicative **let** Expressions

let a:A • $\mathcal{P}$(a) **in** $\mathcal{B}$(a) **end**

express the selection of a value a of type A which satisfies a predicate $\mathcal{P}$(a) for evaluation in the body $\mathcal{B}$(a).

s159

A.7.4 Pattern and “Wild Card” let Expressions

Patterns and *wild cards* can be used:

Patterns

```
let {a} ∪ s = set in ... end
let {a,_} ∪ s = set in ... end

let (a,b,...,c) = cart in ... end
let (a,_,...,c) = cart in ... end

let ⟨a⟩⁀ℓ = list in ... end
let ⟨a,_,b⟩⁀ℓ = list in ... end

let [a↦b] ∪ m = map in ... end
let [a↦b,_] ∪ m = map in ... end
```

s160

A.7.5 Conditionals

Various kinds of conditional expressions are offered by `RSL`:

Conditionals

```
if b_expr then c_expr else a_expr end

if b_expr then c_expr end ≡ if b_expr then c_expr else skip end

if b_expr_1 then c_expr_1
elsif b_expr_2 then c_expr_2
...
elsif b_expr_n then c_expr_n end

case expr of
   choice_pattern_1 → expr_1,
   choice_pattern_2 → expr_2,
   ...
   choice_pattern_n_or_wild_card → expr_n
end
```

s161

A.7.6 Operator/Operand Expressions

Operator/Operand Expressions

⟨Expr⟩ ::=
    ⟨Prefix_Op⟩ ⟨Expr⟩
  | ⟨Expr⟩ ⟨Infix_Op⟩ ⟨Expr⟩
  | ⟨Expr⟩ ⟨Suffix_Op⟩
  | ...
⟨Prefix_Op⟩ ::=
  − | ∼ | ∪ | ∩ | **card** | **len** | **inds** | **elems** | **hd** | **tl** | **dom** | **rng**
⟨Infix_Op⟩ ::=
    = | ≠ | ≡ | + | − | ∗ | ↑ | / | < | ≤ | ≥ | > | ∧ | ∨ | ⇒
  | ∈ | ∉ | ∪ | ∩ | \ | ⊂ | ⊆ | ⊇ | ⊃ | ^ | † | ◦
⟨Suffix_Op⟩ ::= !

A.8 Imperative Constructs

s162

A.8.1 Statements and State Changes

Often, following the RAISE method, software development starts with highly abstract-applicative constructs which, through stages of refinements, are turned into concrete and imperative constructs. Imperative constructs are thus inevitable in RSL.

Statements and State Change

type
  Unit

value
  stmt: **Unit** → **Unit**
  stmt()

- Statements accept no arguments.
- Statement execution changes the state (of declared variables).
- **Unit** → **Unit** designates a function from states to states.
- Statements, **stmt**, denote state-to-state changing functions.
- Writing () as “only” arguments to a function “means” that () is an argument of type **Unit**.

A.8.2 Variables and Assignment

s163

Variables and Assignment

0. **variable** v:Type := expression
1. v := expr

A.8.3 Statement Sequences and skip

Sequencing is expressed using the ';' operator. **skip** is the empty statement having no value or side-effect.

Statement Sequences and **skip**

2. **skip**
3. stm_1;stm_2;...;stm_n

A.8.4 Imperative Conditionals

Imperative Conditionals

4. **if** expr **then** stm_c **else** stm_a **end**
5. **case** e **of**: p_1→S_1(p_1),...,p_n→S_n(p_n) **end**

s164

A.8.5 Iterative Conditionals

Iterative Conditionals

6. **while** expr **do** stm **end**
7. **do** stmt **until** expr **end**

A.8.6 Iterative Sequencing

Iterative Sequencing

8. **for** e **in** list_expr • P(b) **do** S(b) **end**

s165

A.9 Process Constructs

A.9.1 Process Channels

Let A and B stand for two types of (channel) messages and i:KIdx for channel array indexes, then:

Process Channels

```
channel c:A
channel { k[i]:B • i:KIdx }
```

declare a channel, c, and a set (an array) of channels, k[i], capable of communicating values of the designated types (A and B). s166

A.9.2 Process Composition

Let P and Q stand for names of process functions, i.e., of functions which express willingness to engage in input and/or output events, thereby communicating over declared channels. Let P() and Q stand for process expressions, then:

Process Composition

```
P ‖ Q    Parallel composition
P [] Q   Nondeterministic external choice (either/or)
P ⌈⌉ Q   Nondeterministic internal choice (either/or)
P ╫ Q    Interlock parallel composition
```

express the parallel (‖) of two processes, or the nondeterministic choice between two processes: either external ([]) or internal (⌈⌉). The interlock (╫) composition expresses that the two processes are forced to communicate only with one another, until one of them terminates. s167

A.9.3 Input/Output Events

Let c, k[i] and e designate channels of type A and B, then:

Input/Output Events

```
c ?, k[i] ?       Input
c ! e, k[i] ! e   Output
```

expresses the willingness of a process to engage in an event that “reads” an input, respectively “writes” an output. s168

A.9.4 Process Definitions

The below signatures are just examples. They emphasise that process functions must somehow express, in their signature, via which channels they wish to engage in input and output events.

Process Definitions

```
value
    P: Unit → in c out k[i]
    Unit
    Q: i:KIdx →  out c in k[i] Unit

    P() ≡ ... c ? ... k[i] ! e ...
    Q(i) ≡ ... k[i] ? ... c ! e ...
```

The process function definitions (i.e., their bodies) express possible events.

s169

A.10 Simple RSL Specifications

Often, we do not want to encapsulate small specifications in schemes, classes, and objects, as is often done in RSL. An RSL specification is simply a sequence of one or more types, values (including functions), variables, channels and axioms:

Simple RSL Specifications

```
    type
        ...
    variable
        ...
    channel
        ...
    value
        ...
    axiom
        ...
```

B Indexes

- The [#i] which adorn most ‘Type and Function Index’ entries refer to enumerated narrative items and the formula lines.

Types, 8–32
Bezier [15(a)], 12
BoL [56(a)], 21
CId, 21
Cost [15(c)], 12
Cost [70], 23
Days [79(j)], 26
Frac [55(c)], 21
Freight_Attrs [76(e)], 25
Freight_Attrs [79(e)], 26
H [1], 8
HI [4], 9
HLoc [14(a)], 12
HubEntry [17], 12
HubExit [18], 12
HubTrav [16], 12
L [1], 8
Length [15(b)], 12
LI [4], 9
LinkEntry [20], 13
LinkExit [21], 13
LinkTrav [19], 13
N [1], 8
Neg_Resp [77], 25
Optimality [76(f)], 25
R′≡Route′ [23], 14
RcvrId [76(d)], 25
RcvrId [79(d)], 26
RN [12], 10
Route′≡R′ [23], 14
Route≡R [23], 14
R≡Route [23], 14
Ship_Req′ [76], 25
SndrId [76(a)], 25
SndrId [79(a)], 26
TCId [51(c)], 20
TCId [53(a)], 21
TI [73], 24
TimDur [14(b)], 12
Timetable ≡ TT [51(b)], 20
TLT [71], 23
TLT′ [69], 23
TM [34], 17
TRAFFIC [94], 31
TRAFFIC′ [93], 31
TransComp [51], 20
TT [67], 23
TT [63], 22
TT′ [60], 22
TTId [59], 22
TTs′ [66], 23
Vehicle [52], 21
Vehicle_type [54], 21
Velocity [52], 21
VHLoc [55(a)], 21
Vid [51(a)], 20
VId [53], 21
VLLoc [55(b)], 21
VLoc [55], 21
VM [35], 17
WB [79], 26

Functions, 8–32
$\ll$ (date/times) [87], 30
$\ominus$ (date/times) [88], 30
$\oplus$ (costs) [86(b)], 29
$\oplus$ (days) [88], 30
$\prec$ (days or costs) [92], 31
$\simeq$ (costs) [85(b)], 29
are_disjoint [27], 15
before [72], 24
calc_RN [12], 10
commensurate_cost [74], 24
commensurate_costs [85], 29
commensurate_duration [88], 30
commensurate_time [73], 24
disconnected_nets [28], 15
fastest_route [33(b)], 17
fstHI [22(a)], 14
gen_Rs, 14
HubEntries, 13
HubExits, 13

HubTraversals, 12, 13
interval [73], 24
is_connected [26], 15
is_route [24(a)], 15
is_sgl_RM [39], 18
is_sgl_TM [41], 18
is_sgl_TM [39], 18
is_sub_ref_net [13], 11
is_sub_refnet [73], 25
is_sublist [82], 28
is_subnet [30], 16
least_costly_route [33(b)], 17
length [31], 16
LinkEntries, 13
LinkExits, 13
LinkTraversals, 13
lstHI [22(b)], 14
net_modalities [42], 18
no_hub_revisits [83], 28
obs_Arrival [57(a)], 21
obs_BoLs [56(a)], 21
obs_Cid [56(b)], 21
obs_Cost [15(c)], 12
obs_HI [5], 9
obs_HIs [8], 9
obs_HLoc [14(a)], 12
obs_Hs [2], 8
obs_InTime [14(b)], 12
obs_InTime [15(d)], 12
obs_Length [15(b)], 12
obs_LI [6], 9
obs_LIs [7], 9
obs_LLoc [15(a)], 12
obs_Ls [3], 8
obs_OutTime [14(b)], 12
obs_OutTime [15(d)], 12
obs_TCId [51(c)], 20
obs_TCId [61], 22
obs_TCId [53(a)], 21
obs_Timetable ≡ obs_TT [51(b)], 20
obs_TM [34], 17
obs_TMs [36], 17
obs_TravTime [14(b)], 12
obs_TravTime [15(d)], 12
obs_Vehicle_type [54], 21
obs_Velocity [57(b)], 21
obs_VId [62], 22
obs_VId [53], 21
obs_VIds [51(a)], 20
obs_VLoc [55], 21
obs_VM [35], 17
optimal_WBs [92], 31
precede [87], 30
refnet_is_net_commensurable [65], 23
reverse_route [25], 15
route_TMs [40], 18
routes [24(b)], 15
shortest_route [33(a)], 17
subnets [29], 16
sum_of_sums_of_costs [86], 29
travel_time [32], 16
tt_is_refnet_commensurable [64], 23
wf_R [23], 14
wf_TLT [71], 24
wf_TLT* [75], 24, 25
wf_TRAFFIC [95], 31
wf_TT [60], 22
wf_TT [63], 22
wf_tt_arguments [81], 28
wf_TTs [68], 23
wf_WB [81(a)], 27
xtr_H [11], 10
xtr_HIs [9], 10
xtr_L [10], 10
xtr_LIs [9], 10
xtr_RN [75(d)], 25