Abstract

We present a recording of stages and steps of a development of a domain analysis & description of an answer to the question: what, mathematically, is a container terminal?

Caveats: The present, November 22, 2018: 11:05 am, version of this document is “vastly” incomplete. It is being distributed only to a limited circle of people so that they can see that my ECNU course project proposal is one of substance. Being incomplete, it is rich on incomplete formulas and poor on explanatory text; and can only be understood, i.e., appreciated, that is, requires non-trivial knowledge of:

Dines Bjørner.
Manifest Domains: Analysis & Description.
Online: July 2016. DOI 10.1007/s00165-016-0385-z
Springer, BCS copyright 2016

Limited Circulation:
This document constitutes my preparation for a possible student project at ECNU in November 2018. The students are themselves to analyse & describe a domain of container terminals. Therefore, please, do not circulate this document!
## Contents

1 **Introduction**  
    1.1 Survey of Literature on Container-related Matters ............................................. 7

2 **Some Pictures**  
    2.1 Terminal Port Container Stowage Area ................................................................. 8
    2.2 Container Stowage Area and Quay Cranes ............................................................. 9
    2.3 Container Vessel Routes ......................................................................................... 9
    2.4 Containers ............................................................................................................... 10
        2.4.1 40 and 20 Feet Containers ................................................................................ 10
        2.4.2 Container Markings ............................................................................................ 10
    2.5 Container Vessels ..................................................................................................... 10
    2.6 Container Stowage Area: Bays Rows, Stacks and Tier ............................................... 11
    2.7 Stowage Software ..................................................................................................... 12
    2.8 Quay Cranes ............................................................................................................. 12
    2.9 Container Stowage Area and Stack Cranes ............................................................. 13
    2.10 Container Stowage Area .......................................................................................... 13
    2.11 Quay Trucks .......................................................................................................... 13
    2.12 Map of Shanghai and YangShan ............................................................................. 13

3 **SECT** ......................................................................................................................... 13

4 **Main Behaviours**  
    4.1 A Diagram ............................................................................................................... 15
    4.2 Terminology - a Caveat ............................................................................................ 16
    4.3 Assumptions ............................................................................................................. 17

5 **Endurants** .................................................................................................................. 18
    5.1 Parts .......................................................................................................................... 18
        5.1.1 Terminal Ports .................................................................................................... 19
        5.1.2 Quays ................................................................................................................ 20
        5.1.3 Container Stowage Areas: Bays, Rows and Stacks ................................. 20
        5.1.4 Vessels ............................................................................................................. 21
        5.1.5 Functions Concerning Container Stowage Areas ......................................... 21
        5.1.6 Axioms Concerning Container Stowage Areas ............................................. 22
        5.1.7 Stacks ............................................................................................................... 23
    5.2 Terminal Port Command Centers ....................................................................... 23
        5.2.1 Discussion ......................................................................................................... 23
        5.2.2 Justification ....................................................................................................... 23
    5.3 Unique Identifications ............................................................................................. 24
        5.3.1 Unique Identifiers: Distinctness of Parts ......................................................... 25
        5.3.2 Unique Identifiers: Two Useful Abbreviations .............................................. 25
        5.3.3 Unique Identifiers: Some Useful Index Set Selection Functions ............... 25
        5.3.4 Unique Identifiers: Ordering of Bays, Rows and Stacks ............................... 26
    5.4 States, Global Values and Constraints ................................................................. 26
        5.4.1 States ................................................................................................................. 26
        5.4.2 Unique Identifiers ............................................................................................ 27
        5.4.3 Some Axioms on Uniqueness ......................................................................... 29
    5.5 Mereology .................................................................................................................. 29
        5.5.1 Physical versus Conceptual Mereology ......................................................... 29
        5.5.2 Vessels ............................................................................................................... 29
            Physical Mereology: Conceptual Mereology: ................................................. 30
        5.5.3 Quay Cranes ...................................................................................................... 30
            Physical Mereology: Conceptual Mereology: ................................................. 31
        5.5.4 Quay Trucks ...................................................................................................... 31
            Physical Mereology: Conceptual Mereology: ................................................. 31
        5.5.5 Stack Cranes ...................................................................................................... 31
            Physical Mereology: Conceptual Mereology: ................................................. 31
Container Stowage Areas

Bays, Rows and Stacks

5.5.7 Bay Mereology

Physical Vessel Bay Mereology:

Conceptual Vessel Bay Mereology:

Physical Terminal Port Bay (cum Stack) Mereology:

Conceptual Terminal Port Bay (cum Stack) Mereology:

5.5.8 Land Trucks

Physical Mereology:

Conceptual Mereology:

5.5.9 Command Center

5.5.10 Conceptual Mereology of Containers

5.6 Attributes

5.6.1 States

5.6.2 Actions

5.6.3 Attributes: Quays

5.6.4 Attributes: Vessels

5.6.5 Attributes: Quay Cranes

5.6.6 Attributes: Quay Trucks

5.6.7 Attributes: Terminal Stack Cranes

5.6.8 Attributes: Container Stowage Areas

5.6.9 Attributes: Land Trucks

5.6.10 Attributes: Command Center

5.6.11 Attributes: Containers

6 Perdurants

6.1 A Modelling Decision

6.2 Virtual Container Storage Areas

6.3 Changes to The Parts Model

6.4 Basic Model Parts

6.5 Actions, Events, Channels and Behaviours

6.6 Actions

6.6.1 Command Center Actions

Motivating the Command Center Concept:

Calculate Next Transaction:

Command Center Action [A]: update_position_from_vessel:

Command Center Action [B]: calc_ves_pos:

Command Center Action [C-D-E]: calc_ves_map:

Command Center Action [F-G-H]: calc_stack:

Command Center Action [I-J-K]: calc_port:

Command Center Action [L-M-N]: calc_stack:

Command Center Action [N-M-L]: calc_stack:

Command Center Action [O-P-Q]: calc_stack:

Command Center Action [Q-P-O]: calc_stack:

Command Center: Further Observations

6.6.2 Container Storage Area Actions

The Load Pre-/Post-Conditions

The Unload Pre-/Post-Conditions

6.6.3 Vessel Actions

Action [A]: calc_next_port:

Vessel Action [B]: calc_vessel_map:

6.6.4 Land Truck Actions

Land Truck Action [R]: calc_truck_delivery:

Land Truck Action [S]: calc_truck_avail:

6.7 Events

6.7.1 Active Part Initiation Events

6.7.2 Active Part Completion Events

6.8 Channels

6.8.1 Channel Declarations

6.8.2 Channel Messages

A,B,X,Y,C: Vessel Messages
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.5.6</td>
<td>Function Definitions</td>
<td>99</td>
</tr>
<tr>
<td></td>
<td>Explicit Function Definitions</td>
<td>99</td>
</tr>
<tr>
<td></td>
<td>Implicit Function Definitions</td>
<td>99</td>
</tr>
<tr>
<td>10.6</td>
<td>Other Applicative Expressions</td>
<td>100</td>
</tr>
<tr>
<td>10.6.1</td>
<td>Simple let Expressions</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>Let Expressions</td>
<td>100</td>
</tr>
<tr>
<td>10.6.2</td>
<td>Recursive let Expressions</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>Recursive let Expressions</td>
<td>100</td>
</tr>
<tr>
<td>10.6.3</td>
<td>Predicative let Expressions</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>Predicative let Expressions</td>
<td>100</td>
</tr>
<tr>
<td>10.6.4</td>
<td>Pattern and &quot;Wild Card&quot; let Expressions</td>
<td>101</td>
</tr>
<tr>
<td></td>
<td>Patterns</td>
<td>101</td>
</tr>
<tr>
<td>10.6.5</td>
<td>Conditionals</td>
<td>101</td>
</tr>
<tr>
<td></td>
<td>Conditionals</td>
<td>101</td>
</tr>
<tr>
<td>10.6.6</td>
<td>Operator/Operand Expressions</td>
<td>102</td>
</tr>
<tr>
<td></td>
<td>Operator/Operand Expressions</td>
<td>102</td>
</tr>
<tr>
<td>10.7</td>
<td>Imperative Constructs</td>
<td>102</td>
</tr>
<tr>
<td>10.7.1</td>
<td>Statements and State Changes</td>
<td>102</td>
</tr>
<tr>
<td></td>
<td>Statements and State Changes</td>
<td>102</td>
</tr>
<tr>
<td>10.7.2</td>
<td>Variables and Assignment</td>
<td>103</td>
</tr>
<tr>
<td></td>
<td>Variables and Assignment</td>
<td>103</td>
</tr>
<tr>
<td>10.7.3</td>
<td>Statement Sequences and skip</td>
<td>103</td>
</tr>
<tr>
<td></td>
<td>Statement Sequences and skip</td>
<td>103</td>
</tr>
<tr>
<td>10.7.4</td>
<td>Imperative Conditionals</td>
<td>103</td>
</tr>
<tr>
<td></td>
<td>Imperative Conditionals</td>
<td>103</td>
</tr>
<tr>
<td>10.7.5</td>
<td>Iterative Conditionals</td>
<td>103</td>
</tr>
<tr>
<td></td>
<td>Iterative Conditionals</td>
<td>103</td>
</tr>
<tr>
<td>10.7.6</td>
<td>Iterative Sequencing</td>
<td>103</td>
</tr>
<tr>
<td></td>
<td>Iterative Sequencing</td>
<td>103</td>
</tr>
<tr>
<td>10.8</td>
<td>Process Constructs</td>
<td>104</td>
</tr>
<tr>
<td>10.8.1</td>
<td>Process Channels</td>
<td>104</td>
</tr>
<tr>
<td></td>
<td>Process Channels</td>
<td>104</td>
</tr>
<tr>
<td>10.8.2</td>
<td>Process Composition</td>
<td>104</td>
</tr>
<tr>
<td>10.8.3</td>
<td>Input/Output Events</td>
<td>104</td>
</tr>
<tr>
<td></td>
<td>Input/Output Events</td>
<td>104</td>
</tr>
<tr>
<td>10.8.4</td>
<td>Process Definitions</td>
<td>105</td>
</tr>
<tr>
<td></td>
<td>Process Definitions</td>
<td>105</td>
</tr>
<tr>
<td>10.9</td>
<td>Simple RSL Specifications</td>
<td>105</td>
</tr>
<tr>
<td></td>
<td>Simple RSL Specifications</td>
<td>105</td>
</tr>
<tr>
<td>10.10</td>
<td>RSL Index</td>
<td>106</td>
</tr>
</tbody>
</table>
Abstract

This is a report on an experiment. At any stage of development, and the present draft stage is judged 2/3 “completed” it reflects how I view an answer to the question what is a container terminal port? mathematically speaking.

1 Introduction

TO BE WRITTEN

1.1 Survey of Literature on Container-related Matters

[1, A Container Line Industry Domain, 2007]
[2, A-Z Dictionary of Export, Trade and Shipping Terms]
[3, Portworker Development Programme: PDP Units]
[4, An interactive simulation model for the logistics planning of container operations in seaports, 1996]
[5, Stowage planning for container ships to reduce the number of shifts, 1998]
[6, Container stowage planning: a methodology for generating computerised solutions, 2000]
[7, Container ship stowage problem: complexity and connection to the coloring of circle graphs, 2000]
[8, Container stowage pre-planning: using search to generate solutions, a case study, 2001]
[9, A genetic algorithm with a compact solution encoding for the container ship stowage problem, 2002]
[10, Multi-objective ... stowage and load planning for a container ship with container rehandle ..., 2004]
[11, Container terminal operation and operations research - a classification and literature review, 2004]
[12, Online rules for container stacking, 2010]
2 Some Pictures

2.1 Terminal Port Container Stowage Area

Analysis of the above picture:

- The picture shows a terminal.
- At bottom we are hinted (through shadows) at quay cranes serving (unshown) vessels.
- Most of the picture shows a container stowage area, here organised as a series of columns, from one side of the picture to the other side, e.g., left-to-right, sequences (top-to-bottom) of [blue] bays with rows of stacks of containers.
- Almost all columns show just one bay.
- Three “rightmost” columns show many [non-blue] bays.
- Most of the column “tops” and “bottoms” show stack cranes.
- The four leftmost columns show stack cranes at bays “somewhere in the middle” of a column.
2.2 Container Stowage Area and Quay Cranes

2.3 Container Vessel Routes
2.4 Containers

2.4.1 40 and 20 Feet Containers

2.4.2 Container Markings

2.5 Container Vessels
Quay cranes and vessel showing row of aft (rear) bay.

2.6 Container Stowage Area: Bays Rows, Stacks and Tier

Bay, Row, Tier Numbers. Row Numbers

Cross section of a Bay. Tier Numbers.

Bay Numbering
2.7 Stowage Software

2.8 Quay Cranes

2.9 Container Stowage Area and Stack Cranes
### 2.10 Container Stowage Area

![Container Stowage Area](image1.jpg)

### 2.11 Quay Trucks

![Quay Trucks](image2.jpg)

### 2.12 Map of Shanghai and YangShan

![Map of Shanghai and YangShan](image3.jpg)

### 3 SECT

- *Shanghai East Container Terminal*
is the joint venture terminal of
APM Terminals and
Shanghai International Port Group
in Wai Gao Qiao port area of Shanghai.

• No.1 Gangjian Road, Pudong New District, Shanghai, China
4 Main Behaviours

- From consumer/origin to consumer/final destination:
  - container loads onto land truck;
  - land truck travels to terminal stack;
  - container unloads by means of terminal stack crane from land truck onto terminal stack.
Container moves from stack to vessel:
- terminal stack crane moves container from terminal stack to quay truck,
- quay truck moves container from terminal stack to quay,
- quay crane moves container to top of a vessel stack;

Container moves on vessel from terminal to terminal:
- Either container is unloaded at a next terminal port to a stack and from there to a container truck
- or: container is unloaded at a next terminal port to a stack and from there to a next container vessel.

4.1 A Diagram

![Diagram of container terminal activities]

Fig. 1: Container Terminal Ports, I
A “from the side” snapshot of terminal port activities

4.2 Terminology - a Caveat

Bay: contains indexed set of rows (of stacks of containers).
Container: smallest unit of central (i.e., huge) concern!

Container Stowage Area: An area of a vessel or a terminal where containers are stored, during voyage, respectively awaiting to be either brought out to shippers or onto vessels.

1The terms introduced in this section are mine. They are most likely not the correct technical terms of the container shipping and stowage trade. I expect to revise this section, etc.
**Crane:**

- **Stack Crane**: moves *containers* between *land* or *terminal trucks* and *terminal stacks*.
- **Quay Crane**: moves *containers* between *[land or]* *terminal trucks* and *vessels*.

**Land**: ... as you know it ...

**Ocean**: ... as you know it ...

**Shipper**: arranges shipment of containers with container lines

**Quay**: area of terminal next to vessels (hence water).

**Row**: contains indexed set of *stacks* (of containers).

**Stack**: contains indexed set of *containers*.

We shall also, perhaps confusingly, use the term stack referring to the land-based bays of a terminal.

**Terminal**: area of land and water between land and ocean equipped with container stowage area, and stack and quay cranes, etc.

**Truck**:  

- **Land Truck**: privately operated truck transport *containers* between *shippers* and *stack cranes*.
- **Quay Truck**: terminal operated special truck transport *containers* between *stack cranes* and *quay cranes*.

**Tier**: index of *container* in *stack*.

**Vessel**: contains a *container stowage area*.

### 4.3 Assumptions

Without loss of generality we can assume that there is exactly one stack crane per land-based terminal stack; quay cranes each serve exactly one bay on a vessel; there are enough quay cranes to serve all bays of any berthed vessel; quay trucks may serve any (quay and stack) crane; land trucks may serve more than one terminal; et cetera.
We refer to [13, Sects. 3., 4., and 5.].

Our model focuses initially on parts, that is, manifest, observable phenomena. Our choice of these is expected to be subject to serious revision once we... More to come ...

5.1 Parts

We refer to [13, Sect. 3.3].

Our model has, perhaps arbitrarily, focused on just some of the manifest, i.e., observable parts of a domain of container terminal ports. We shall invariable refer to container terminal ports as either container terminals, or terminal ports, tp:TP, or just terminals. We expect revisions to the decomposition as shown as we learn more from professional stakeholders, e.g., APM Terminals/SECT, Shanghai.

1 In the container line industry, CLI, we can observe

2 a structure, TPS, of all terminal ports, and from each such structure, an indexed set, TPs, of two or more container terminal ports, TP;

3 a structure, VS, of all container vessels, and from each such structure, an indexed set, Vs, of one or more container vessels, V; and
4 a structure, LTS, of all land trucks, and from each such structure, a non-empty, indexed set, LTs of land trucks, LT;

type
1 CLI
2 STPs, TP = TP-set, TP
3 SVs, V = V-set, V
9 SLTs, LT = LT-set, LT

value
2 obs_STPs: CLI → STPs, obs_TPs: STPs → TPs
3 obs_SVs: CLI → SVs, obs_Vs: SVs → Vs
9 obs_SLTs: CLI → SLTs, obs_LTs: SLTs → LTs

axiom
2 ∀ cli: CLI • card(obs_STPs(obs_STPs(cli))) ≥ 2
3 ∧ card(obs_Vs(obs_SVs(cli))) ≥ 1
9 ∧ card(obs_LTs(obs_SLTs(cli))) ≥ 1

5.1.1 Terminal Ports

In a terminal port, tp: TP, one can observe

5 a [composite] container stowage area, csa: CSA;

6 a structure, sqc: SQC, of quay cranes, and from that, a non-empty, indexed set, qcs: QCs, of one or more quay cranes, qc: QC;

7 structure, sqt: SQT, of quay trucks, and from that a non-empty, indexed set, qts: QTs, of quay trucks, qt: QT;

8 a structure, Scs: SCS, of stack cranes, and from that a non-empty, indexed set, scs: SCs, of one or more stack cranes, sc: SC;

9 a[n atomic] quay, q: Q;

10 a[n atomic] terminal port monitoring and control center, mcc: MCC.

We can, without loss of generality, describe a terminal as having exactly one quay (!) – just as we, again without any loss of generality, describe it as having exactly one container stowage area.

Quay: a long structure, usually built of stone, where boats can be tied up to take on and off their goods.

Pronunciation: key.

Thesaurus: berth, jetty, key, landing, levy, slip, wharf
type
5 CSA
6 SQC, QCs = QC-set, QC
7 SQT, QTs = QT-set, QT
8 SCS, SCs = SC-set, SC
9 Q
10 MCC

value
5 obs_{CSA}: TP → CSA
6 obs_{SQC}: TP → SQC, obs_{QCs}: SQC → QCs
7 obs_{SQT}: TP → SQT, obs_{QTs}: SQT → QTs
8 obs_{SCS}: TP → SCS, obs_{SCs}: SCS → SCs
9 obs_{Q}: TP → Q
10 obs_{MCC}: TP → MCC

axiom
6 ∀ sqc:SQC • \text{card}\ obs_{QCs}(sqc) ≥ 1
7 ∀ sqt:SQT • \text{card}\ obs_{QTs}(sqt) ≥ 1
8 ∀ scs:SCS • \text{card}\ obs_{SCs}(scs) ≥ 1

5.1.2 Quays

Although container terminal port quays can be modelled as composite parts we have chosen to describe them as atomic. We shall subsequently endow the single terminal port quay with such attributes as quay segments, quay positions and berthing\(^4\).

5.1.3 Container Stowage Areas: Bays, Rows and Stacks

11 From a container stowage area one can observe a non-empty indexed set of bays,
12 From a bay we can observe a non-empty indexed set of rows,
13 From a row we can observe a non-empty indexed set of stacks,
14 From a stack we can observe a possibly empty indexed set of containers.

type
11 BAYS, BAYs = BAY-set, BAY
12 ROWS, ROWs = ROW-set, ROW

\(^4\)Berth: Sufficient space for a vessel to maneuver; a space for a vessel to dock or anchor; (whether occupied by vessels or not). Berthing: To bring (a vessel) to a berth; to provide with a berth.
STKS, STKs = STK-set, STK
CONS, CONs = CON-set, CON

value
obs_BAYS: CSA → BAYS, obs_BAYs: BAYS → BAYs
obs_ROWS: BAY → ROWS, obs_ROWS: ROWS → ROWs
obs_STKS: ROW → STKS, obs_STKs: STKS → STKs
obs_CONS: STK → CONS, obs_CONs: CONS → CONs

axiom
∀ bays:BAYs • card bays > 0
∀ rows:ROWs • card rows > 0
∀ stks:STKs • card stks > 0

5.1.4 Vessels
From (or in) a vessel one can observe

[5] a container stowage area

and some other parts.

type
5 CSA
16 ...

value
5 obs_CSA: V → CSA
16 ...

5.1.5 Functions Concerning Container Stowage Areas

One can calculate

the set of all container storage areas:

of all terminal ports together with those

of all container lines.

value
cont_stow_areas: CLI → CSA-set
cont_stow_areas(cli) ≡
{obs_CSA(tp)|tp:TP•tp ∈ obs_TPs(obs_TPS(cli))}
∪ {obs_CSA(cl)|cl:CL•cl ∈ obs_CLs(obs_CLS(cli))}
One can calculate the containers of

21 a stack,
22 a row,
23 a bay, and
24 a container stowage area.

5.1.6 Axioms Concerning Container Stowage Areas

25 All rows contain different, i.e. distinct containers.

26 All bays contain different, i.e. distinct containers.

27 All container stowage areas contain different, i.e. distinct containers.
5.1.7 Stacks

An aside: We shall use the term ‘stack’ in two senses: (i) as a component of container storage area bays; and (ii) to refer to the collection of stacks in a bay of a terminal container storage area.

28 Stacks are created empty, and hence stacks can be empty.

29 One can push a container onto a stack and obtain a non-empty stack.

30 One can pop a container from a non-empty stack and obtain a pair of a container and a possibly empty stack.

value
28 empty: () → STK, is_empty: STK → Bool
29 push: CON × STK → STK
30 pop: STK → (CON × STK)

axiom
28 is_empty(empty()), ~is_empty(push(c,stk))
29 pop(push(c,stk)) = (c,stk)
30 pre pop(stk), pop(push(c,stk)): ~is_empty(stk)
30 pop(empty()) = chaos

5.2 Terminal Port Command Centers

5.2.1 Discussion

We consider terminal port monitoring & control command centers to be atomic parts. The purpose of a terminal port command center is to monitor and control the allocation and servicing (berthing) of any visiting vessel to quay positions and by quay cranes, the allocation and servicing of vessels by quay cranes, the allocation and servicing of quay cranes by quay trucks, the allocation and servicing of quay trucks to quay cranes, containers and terminal stacks, the allocation and servicing of land trucks to containers and terminal stacks, This implies that there are means for communication between a terminal command center and vessels, quay cranes, stack cranes, quay trucks, land trucks, terminal stacks and containers.

5.2.2 Justification

We shall justify the concept of terminal monitoring & control, i.e., command centers. First, using the domain analysis & description approach of [13], we know that we are going, through a transcendental deduction, to model certain parts as behaviours. These
parts, we decide, after some analysis that we forego, to be vessels, quay cranes, quay trucks, stack cranes stacks, land trucks, and containers. Behaviours are usually like actors: they can instigate actions. But we decide, in our analysis, that some of these behaviours, quay cranes, quay trucks, stack cranes and stacks, are “passive” actors: are behaviourally not endowed with being able to initiate “own” actions. Instead, therefore, of all these behaviours, being able to communicate directly, pairwise, as loosely indicated by the figures of Pages 16 and 18, we model them to communicate via their terminal command centers.

This is how we justify the introduction of the concept of terminal command centers. They are an abstraction. In “ye olde days” you could observe, not one, but, perhaps, a hierarchy of terminal port offices, staffed by people, [each office, each group of staff] with its set of duties: communicating (by radio-phone) with approaching [and departing] vessels; scheduling quay positions, quay cranes and quay trucks; managing the operation of cranes and trucks; and, on a large scale, calculating stowage: on vessels and in terminals. Today, “an age of ubiquitous computing”, most of these offices and their staff are replaced by electronics: sensors, actuators, communication and computing, and with massive stowage data processing: where should containers be stowed on board vessels and in terminals so as to near-optimise all operations.

5.3 Unique Identifications

We refer to [13, Sect. 5.1].

| 31 | Vessels have unique identifiers. |
| 32 | Quay cranes have unique identifiers. |
| 33 | Quay trucks have unique identifiers. |
| 34 | Stack cranes have unique identifiers. |
| 35 | Bays (“Stacks”) of terminal container stowage areas have unique identifiers, cf. Item 39. |
| 36 | Land trucks have unique identifiers. |
| 37 | Terminal port command centers have unique identifiers. |
| 38 | Containers have unique identifiers. |
| 39 | Bays of container stowage areas have unique identifiers. |
| 40 | Rows of a bay have unique identifiers. |
| 41 | Stacks of a row have unique identifiers. |
| 42 | The part unique identifier types are mutually disjoint. |

| type | 33 QTI |
| 31 VI | 34 SCI |
| 32 QCI | 35 TBI |
5.3.1 Unique Identifiers: Distinctness of Parts

If two containers are different then their unique identifiers must be different.

\[ \forall \text{con}, \text{con}': \text{CON} \cdot \text{con} \neq \text{con}' \Rightarrow \text{uid}_\text{CON}(\text{con}) \neq \text{uid}_\text{CON}(\text{con}') \]

The same distinctness criterion applies to stacks, rows, bays, container storage areas, terminal ports, cranes, vessels, etc.

5.3.2 Unique Identifiers: Two Useful Abbreviations

Container positions within a container stowage area can be represented in two ways:

- by a triple of a bay identifier, a row identifier and a stack identifier, and
- by these three elements and a tier position (i.e., position within a stack).

\[ \text{BRS} = \text{BI} \times \text{RI} \times \text{SI} \]
\[ \text{BRSP} = \text{BI} \times \text{RI} \times \text{SI} \times \text{Nat} \]

5.3.3 Unique Identifiers: Some Useful Index Set Selection Functions

- From a container stowage area once can observe all bay identifiers.
- From a bay once can observe all row identifiers.
48 From a row once can observe all stack identifiers.

49 From a virtual container storage area, i.e., an iCSA, one can extract all the unique container identifiers.

\[
xtr_{\text{BI}s}(\text{csa}) \equiv \{ \text{uid}_{\text{BAY}}(\text{bay}) | \text{bay} : \text{BAY} \cdot \text{bay} \in xtr_{\text{BAY}s}(\text{csa}) \}
\]

\[
xtr_{\text{RI}s}(\text{bay}) \equiv \{ \text{uid}_{\text{ROW}}(\text{row}) | \text{row} : \text{ROW} \cdot \text{row} \in \text{obs}_{\text{ROWS}}(\text{bay}) \}
\]

\[
xtr_{\text{SI}s}(\text{row}) \equiv \{ \text{uid}_{\text{STK}}(\text{stk}) | \text{stk} : \text{STK} \cdot \text{stk} \in \text{obs}_{\text{STK}s}(\text{row}) \}
\]

\[
xtr_{\text{CI}s}(\text{icsa}) \equiv \ldots \ [\text{to come}] \ldots
\]

5.3.4 Unique Identifiers: Ordering of Bays, Rows and Stacks

The bays of a container stowage area are usually ordered. So are the rows of bays, and stacks of rows. Ordering is here treated as attributes of container stowage areas, bays and stacks. We shall treat attributes further on.

5.4 States, Global Values and Constraints

5.4.1 States

50 We postulate a container line industry \( \text{cli} : \text{CLI} \).

From that we observe, successively, all parts:

51 the set, \( \text{cs} : \text{C-set} \), of all containers;

52 the set, \( \text{tps} : \text{TPs} \), of all terminal ports;

53 the set, \( \text{vs} : \text{Vs} \), of all vessels; and

54 the set, \( \text{fts} : \text{LTs} \), of all land trucks.
value
50  cli:CLI
51  cs:C-set = obs_Cs(obs_CS(cli))
52  tps:TP-set = obs_TPs(obs_TPS(cli))
53  vs:V-set = obs_Vs(obs_VS(cli))
54  lts:Lts = obs_LTs(obs_LTS(cli))

We can observe
55  csas:CSA-set, the set of all terminal port container stowage areas of all terminal ports;
56  bays:BAY-set, the terminal port bays of all terminals;
57  the set, qcs:QC-set, of all quay cranes of all terminals;
58  the set, qts:QT-set, of all quay trucks of all terminal ports; and
59  the set, scs:SC-set, of all terminal (i.e., stack) cranes of all terminal ports.

value
55  csas:CSA-set = \{obs_CSA(tp)|tp:TP • tp ∈ tps\}
55  bays:BAY-set = \{obs_BAY(csa)|csa:CSA • csa ∈ csas\}
57  qcs:QC-set = \{obs_QCs(obs_QCS(tp))|tp:TP • tp ∈ tps\}
58  qts:QT-set = \{obs_QTs(obs_QTS(tp))|tp:TP • tp ∈ tps\}
59  scs:SC-set = \{obs_SCs(obs_SCS(tp))|tp:TP • tp ∈ tps\}

5.4.2 Unique Identifiers

Given the generic parts outlined in Sect. 5.4.1 we can similarly define generic sets of unique identifiers.

60  There is the set, c_uis, of all container identifiers;
61  the set, tp_uis, of all terminal port identifiers;
62  the set, mcc_uis, of all terminal port command center identifiers;
63  the set, v_uis, of all vessel identifiers;
64  the set, qc_uis, of quay crane identifiers of all terminal ports;
65  the set, qt_uis, of quay truck identifiers of all terminal ports;
the set, \( sc_{uis} \), of stack crane identifiers of all terminal ports;

the set, \( stk_{uis} \), of stack identifiers of all terminal ports;

the set, \( lt_{uis} \), of all land truck identifiers; and

the set, \( uis \), of all vessel, crane and truck identifiers.

\[
\begin{align*}
\text{value} \quad & c_{uis}: \text{Cl-set} = \{ \text{uid}_C(c) | \text{c}: C \in cs \} \\
& tp_{uis}: \text{TPI-set} = \{ \text{uid}_TP(tp) | \text{tp}: TP \in tps \} \\
& mcc_{uis}: \text{TPI-set} = \{ \text{uid}_{\text{MCC}}(\text{obs}_{\text{MCC}}(tp)) | \text{tp}: TP \in tps \} \\
& v_{uis}: \text{VI-set} = \{ \text{uid}_V(v) | v: V \in vs \} \\
& qc_{uis}: \text{QCI-set} = \{ \text{uid}_{QC}(qc) | \text{qc}: QC \in qcs \} \\
& qt_{uis}: \text{QTI-set} = \{ \text{uid}_{\text{QT}}(qt) | \text{qt}: QT \in qts \} \\
& sc_{uis}: \text{SCI-set} = \{ \text{uid}_{\text{SC}}(sc) | \text{sc}: SC \in scs \} \\
& stk_{uis}: \text{BI-set} = \{ \text{uid}_{\text{BAY}}(stk) | \text{stk}: BAY \in stks \} \\
& lt_{uis}: \text{LTI-set} = \{ \text{uid}_{\text{LL}}(lt) | \text{lt}: LT \in lts \} \\
& uis: \{ \text{VI} | \text{QCI} | \text{QTI} | \text{SCI} | \text{BI} | \text{LTI} \}-\text{set} = v_{uis} \cup qc_{uis} \cup qt_{uis} \cup sc_{uis} \cup stk_{uis} \cup lt_{uis}
\end{align*}
\]

\( \text{tpmcc}_{\text{idm}} \), from terminal port identifiers into the identifiers of respective command centers;

\( \text{mccqc}_{\text{idsm}} \), from command center identifiers into the set of quay crane identifiers of respective ports;

\( \text{mccqt}_{\text{idsm}} \), from command center identifiers into the identifiers of quay trucks of respective ports;

\( \text{mccsc}_{\text{idsm}} \), from command center identifiers into the identifiers of quay trucks of respective ports; and

\( \text{mccbays}_{\text{idsm}} \), from command center identifiers into the set of bay identifiers (i.e., “stacks”) of respective ports.
5.4.3 Some Axioms on Uniqueness

5.5 Mereology

We refer to [13, Sect. 5.2].

5.5.1 Physical versus Conceptual Mereology

We briefly discuss a distinction that was not made in [13]: whether to base a mereology on physical connections or on functional or, as we shall call it, conceptual relations. We shall, for this domain model, choose the conceptual view. The physical mereology view can be motivated, i.e. justified, from the figures on pages 16 and 18. The conceptual view is chosen on the basis of the justification of the terminal command centers, cf. Sect. 5.2 on Page 23. We shall model physical mereology as attributes.\(^5\)

5.5.2 Vessels

Physical Mereology:

Vessels are physically “connectable” to quay cranes of any terminal port.

type

\[ \text{Phys}_V\text{Mer} = \text{QCl-set} \]

value

\[ \text{attr}\_\text{Phys}_V\text{Mer}: V \to \text{Phys}_V\text{mer} \]

\(^5\)Editorial note: Names of physical and of conceptual mereologies have to be “streamlined”. As now, they are a “mess”!

\[ 5 \text{.}4.3 \text{ } \text{S}o\text{me} \text{Axioms} \text{on} \text{Uniqueness} \]

\[ 5 \text{.}5 \text{ } \text{M}erology \]

\[ \text{We refer to} [13, \text{Sect.} \text{5.2}] . \]

\[ 5 \text{.}5.1 \text{ } \text{P}hysical \text{v}ersus \text{C}onceptual \text{M}erology \]

\[ \text{We briefly discuss a distinction that was not made in} [13]: \text{whether to base a mereology on physical connections or on functional or, as we shall call it, conceptual relations. We shall, for this domain model, choose the conceptual view. The physical mereology view can be motivated, i.e. justified, from the figures on pages 16 and 18. The conceptual view is chosen on the basis of the justification of the terminal command centers, cf. Sect. 5.2 on Page 23. We shall model physical mereology as attributes.}^5 \]

\[ 5 \text{.}5.2 \text{ } \text{V}essels \]

\[ \text{Physical Mereology:} \]

\[ 75 \text{ Vessels are physically “connectable” to quay cranes of any terminal port.} \]

\[ \text{type} \]

\[ 75 \text{ Phys}_V\text{Mer} = \text{QCl-set} \]

\[ \text{value} \]

\[ 75 \text{ attr}\_\text{Phys}_V\text{Mer}: V \to \text{Phys}_V\text{mer} \]

\[ ^5 \text{Editorial note: Names of physical and of conceptual mereologies have to be “streamlined”. As now, they are a “mess”!} \]
Conceptual Mereology:

76 Container vessels can potentially visit any container terminal port, hence have as [part of] their mereology, a set of terminal port command center identifiers.

\[
\text{type} \\
V_{\text{Mer}} = \text{MCCI-set} \\
\text{value} \\
mereo_V: V \rightarrow V_{\text{Mer}} \\
\text{axiom} \\
\forall v: V \cdot v \in vs \Rightarrow \text{mereo}_V(v) \subseteq mcc_uis
\]

5.5.3 Quay Cranes

Physical Mereology: In modelling the physical mereology, though as an attribute, of quay cranes, we need the notion of quay positions.

77 Quay cranes are, at any time, positioned at one or more adjacent quay positions of an identified segment of such.

\[
\text{type} \\
\text{Phys}_Q\text{C}_{\text{Mereo}} = \text{QPSId} \times \text{QP}^* \\
\text{value} \\
\text{attr}_{\text{Phys}_Q\text{C}}: \text{QC} \rightarrow \text{Phys}_Q\text{C}_{\text{Mereo}}
\]

78 The quay positions, \( q_{\text{mereo}} = (qpsid, qpl): \text{QCMereo} \), must be proper quay positions of the terminal,

79 that is, the segment identifier, \( qpsid \), must be one of the terminal,

80 and the list, \( qpl \), must be contiguously contained within the so identifier segment.

\[
\text{axiom} \forall \text{tp}: \text{TP}, \\
\text{let} q = \text{obs}_Q(tp), qcs = \text{obs}_Q\text{Cs}(\text{obs}_Q\text{CS}(tp)) \text{ in} \\
\forall q: Q \cdot q \in qcs \Rightarrow \\
\text{let} (qpsid, qpl) = \text{obs}_\text{Mereo}(q), qps = \text{attr}_\text{QPSs}(q) \text{ in} \\
qpsid \in \text{dom} qps \\
\exists i, j: \text{Nat} \cdot \{i, j\} \in \text{inds} qpl \land ((qps(qpsi))[k]_{i \leq k \leq j} = qpl \\
\text{end end}
\]
**Conceptual Mereology:** The conceptual mereology is simpler.

81 Quay cranes are conceptually related to the command center of the terminal in which they are located.

\[ \text{type } QC\_\text{Mer} = MCCI \]
\[ \text{value } \text{mero}_{QC}: QC \rightarrow QC\_\text{Mer} \]

### 5.5.4 Quay Trucks

**Physical Mereology:**
82 Quay trucks are physically “connectable” to quay and stack cranes.

\[ \text{type } \text{Phys}_{QT}\_\text{Mer} = QCI\text{-set} \times QCI\text{-set} \]
\[ \text{value } \text{attr}_{\text{Phys}_{QT}\_\text{Mer}}: QT \rightarrow \text{Phys}_{QT}\_\text{Mer} \]

**Conceptual Mereology:**
83 Quay trucks are conceptually connected to the command center of the terminal port of which they are a part.

\[ \text{type } QT\_\text{Mer} = MCCI \]
\[ \text{value } \text{mero}_{QT}: QT \rightarrow QT\_\text{Mer} \]

### 5.5.5 Stack Cranes

**Physical Mereology:**
84 Terminal stack cranes are positioned to serve one or more terminal area bays, one or more quay trucks and one or more land trucks.

85 The terminal stack crane positions are indeed positions of their terminal
86 and no two of them share bays.
type
84 \( \text{Phys}_{\text{SCmereo}} = s_{\text{bis}}:\text{BI-set} \times s_{\text{qtis}}:\text{QTI-set} \times s_{\text{ltis}}:\text{LTI-set} \)

axiom
84 \( \forall (\text{bis},\text{qtis},\text{ltis}):\text{Phys}_{\text{SCmereo}}: \text{bis} \not= {} \land \text{qtis} \not= {} \land \text{ltis} \not= {} \)

value
84 \( \text{Phys}_{\text{SCmereo}}: \text{SC} \rightarrow \text{Phys}_{\text{SCmereo}} \)

axiom
84 \( \forall \text{tp}:\text{TP} \cdot \)
84 \( \text{let } \text{csa}=\text{obs}_{\text{CSA}}(\text{tp}), \text{bays}=\text{obs}_{\text{BAYS}}(\text{obs}_{\text{BAYS}}(\text{csa})), \text{scs}=\text{obs}_{\text{SCS}}(\text{obs}_{\text{SCS}}(\text{tp})) \text{ in} \)
85 \( \forall \text{sc}:\text{SC} \cdot \text{sc} \in \text{scs} \Rightarrow \text{Phys}_{\text{SCmereo}}(\text{sc}) \subseteq \text{xtr}_{\text{BIs}}(\text{csa}) \)
86 \( \land \forall \text{tp}',\text{tp}'':\text{TP} \cdot \{\text{tc}',\text{tc}'''\} \subseteq \text{tcs} \land \text{tc}' \not= \text{tc}''' \)
86 \( \Rightarrow s_{\text{bis}}(\text{Phys}_{\text{SCmereo}}(\text{tc}')) \cap s_{\text{bis}}(\text{Phys}_{\text{SCmereo}}(\text{tc}'')) = {} \) end

Conceptual Mereology: The conceptual stack crane mereology is simple:

87 Each stack is conceptually related to the command center of the terminal at which it is located.

type
87 \( \text{SC}_{\text{Mer}} = \text{MCCI} \)

value
87 \( \text{mereo}_{\text{SC}}: \text{SC} \rightarrow \text{SC}_{\text{Mer}} \)

5.5.6 Container Stowage Areas

Bays, Rows and Stacks: The following are some comments related to, but not defining a mereology for container stowage areas.

88 A bay of a container stowage area

a. has either a predecessor
b. or a successor,
c. or both (and then distinct).
d. No row cannot have neither a predecessor nor a successor.

89 A row of a bay has a predecessor and a successor, the first stack has no predecessor and the last stack has no successor.

90 A stack of a row has a predecessor and a successor, the first stack has no predecessor, and the last stack has no successor.
value
88 \( \text{BAY} \_\text{Mer}: \text{BAY} \to (\{'\text{nil}'\}|\text{BI}) \times (\text{BI}|\{'\text{nil}'\}) \)
89 \( \text{ROW} \_\text{Mer}: \text{ROW} \to (\{'\text{nil}'\}|\text{RI}) \times (\text{RI}|\{'\text{nil}'\}) \)
90 \( \text{STK} \_\text{Mer}: \text{STK} \to (\{'\text{nil}'\}|\text{SI}) \times (\text{SI}|\{'\text{nil}'\}) \)

axiom
88 \( \forall \text{csa} : \text{CSA} \cdot \text{let } bs = \text{obs} \_\text{BAYs}(\text{obs} \_\text{BAYS}(\text{csa})) \text{ in} \)
88 \( \forall \text{ b : BAY} \cdot \text{ b } \in \text{ bs } \Rightarrow \)
88 \( \text{ let } (nb,nb') = \text{ mereo} \_\text{BAY}(b) \text{ in} \)
88 \( \text{ case } (nb,nb') \text{ of} \)
88a. \( ('\text{nil}',bi) \to bi \in \text{xtr} \_\text{BIs}(\text{csa}), \)
88b. \( (bi,'\text{nil}') \to bi \in \text{xtr} \_\text{BIs}(\text{csa}), \)
88d. \( ('\text{nil}','\text{nil}') \to \text{ chaos}, \)
88c. \( (bi,bi') \to \{bi,bi'\} \subseteq \text{xtr} \_\text{BIs}(\text{csa}) \land bi \neq bi' \)
88 \( \text{ end end end} \)
89 as for rows
90 as for stacks

5.5.7 Bay Mereology

Physical Vessel Bay Mereology:

91 A vessel bay is topologically related to the vessel on board of which it is placed and to the set of all quay cranes of all terminal ports.

type
91 \( \text{Phys} \_\text{VES} \_\text{BAY} \_\text{Mer} = \text{VI} \times \text{QCI-set} \)

Conceptual Vessel Bay Mereology:

92 A vessel bay is conceptually related to the set of all command centers of all terminal ports.

type
92 \( V \_\text{BAY} \_\text{Mer} = \text{MCCI-set} \)
Physical Terminal Port Bay (cum Stack) Mereology:

93 A terminal bay (cum stack) is topologically related to the stack cranes of a given terminal port and all land trucks.

type
93 Phys_{STK,Mer} = SCI-set × LTI-set

Conceptual Terminal Port Bay (cum Stack) Mereology:

94 A terminal port bay is conceptually related to the command center of its port.

type
94 T_{BAY,Mer} = MCCI

5.5.8 Land Trucks

Physical Mereology:

95 Land trucks are physically “connectable” to stack cranes – of any port.

type
95 Phys_{LT,Mer} = SCI-set
value
95 attr_{Phys_{LT,Mer}}: LT → Phys_{LT,Mer}

Conceptual Mereology:

96 Land trucks are conceptually connected to the command centers of any terminal port.

type
96 LT_{Mer} = MCCI-set
value
96 mereo_{LT}: LT → LT_{Mer}
5.5.9 **Command Center**

Command centers are basically conceptual quantities. Hence we can expect the physical mereology to be the conceptual mereology.

Command centers are physically and conceptually connected to all vessels, all cranes of the terminal port of the command center, all quay trucks of the terminal port of the command center, all stacks (i.e., bays) of the terminal port of the command center, and all land trucks, and all containers.

\[
\text{type}\quad \text{MCC\_Mer} = \text{VI\_set} \times \text{QCI\_set} \times \text{QTI\_set} \times \text{SCI\_set} \times \text{BI\_set} \times \text{LTI\_set} \times \text{CI\_set}
\]

\[
\text{value}\quad \text{mereo\_MCC}: \text{MCC} \rightarrow \text{MCC\_Mer}
\]

\[
\text{axiom}\quad \forall \text{tp:TP} \cdot \text{tp} \in \text{tps} \cdot
\]

\[
\text{let}\quad \text{qcs:QC\_set} \cdot \text{qcs} = \text{obs\_QCs} (\text{obs\_QCS(tp)}),
\]

\[
\text{qts:QT\_set} \cdot \text{qts} = \text{obs\_QTS} (\text{obs\_QTS(tp)}),
\]

\[
\text{scs:SC\_set} \cdot \text{scs} = \text{obs\_SCs} (\text{obs\_SCS(tp)}),
\]

\[
\text{bs:iBAY\_set} \cdot \text{bs} = \text{obs\_Bs} (\text{obs\_BS(\text{obs\_CSA(tp)})}) \text{ in}
\]

\[
\text{let}\quad \text{vis:VI\_set} \cdot \text{vis} = \{ \text{uid\_VI(v)} | v:V \cdot v \in \text{vs} \},
\]

\[
\text{qcis:QCI\_set} \cdot \text{qcis} = \{ \text{uid\_QCI(qc)} | \text{qc:QC} \cdot \text{qc} \in \text{qcs} \},
\]

\[
\text{qtis:QTI\_set} \cdot \text{qtis} = \{ \text{uid\_QTI(qt)} | \text{qt:QT} \cdot \text{qt} \in \text{qts} \},
\]

\[
\text{scis:SCI\_set} \cdot \text{scis} = \{ \text{uid\_SCI(sc)} | \text{sc:SC} \cdot \text{sc} \in \text{scs} \},
\]

\[
\text{bis:iBAY\_set} \cdot \text{bis} = \{ \text{uid\_Bi(b)} | \text{bi:BAY} \cdot \text{bi} \in \text{bs} \},
\]

\[
\text{ltis:LTI\_set} \cdot \text{ltis} = \{ \text{uid\_LTI(lt)} | \text{lt:LT} \cdot \text{lt} \in \text{ltis} \},
\]

\[
\text{cis:SCI\_set} \cdot \text{cis} = \{ \text{uid\_Ci(c)} | \text{c:C} \cdot \text{c} \in \text{cis} \} \text{ in}
\]

\[
\text{mereo\_MCC} (\text{obs\_MCC(tp)}) = (\text{vis, qcis, scis, bis, ltis, cis}) \text{ end end}
\]

5.5.10 **Conceptual Mereology of Containers**

The physical mereology of any container is modelled as a container attribute.

The conceptual mereology is modelled by containers being connected to all terminal command centers.

\[
\text{type}\quad \text{C\_Mer} = \text{MCCI\_set}
\]

\[
\text{value}\quad \text{mereo\_C}: \text{C} \rightarrow \text{C\_Mer}
\]

\[
\text{axiom}\quad \forall \text{c:C} \cdot \text{mereo\_C(c)} = \text{mcc\_uis}
\]
5.6 **Attributes**

We refer to [13, Sect. 5.3].

5.6.1 **States**

By a state we shall mean one or more parts such that these parts have *dynamic* attributes, in our case typically *programmable* attributes.

5.6.2 **Actions**

Actions apply to states and yield possibly updated states and, usually, some result values.

We shall in this section, Sect. 5.6, on attributes, outline a number of *simple* (usually called *primitive*) actions of states. These actions are invoked by some behaviours either at their own volition, or in response to events occurring in other behaviours. The action outcomes are simple enough, but calculations resulting in these outcomes are not. Together the totality of the actions performed by the terminal’s monitoring & control of vessels, cranes, trucks and the container stowage area, reflect the complexity of stowage handling.

5.6.3 **Attributes: Quays**

99 Quays are segmented into one or more quay segments, qs:QS, each with a sequence of one or more crane positions, cp:CP.

100 Quay segments and

101 crane positions are further unspecified.

\[
\text{type} \quad QPOS = QS \times CP^* \quad \text{axiom} \quad \forall (\_cpl): QPOS \cdot cpl \neq (\_)
\]

5.6.4 **Attributes: Vessels**

102 A vessel is

a. either at sea, at some *programmable* geographical location (longitude and latitude),
b. or in some *programmable* terminal port – designated by the identifier of its command center and its quay position.

103 We consider the “remainder” of the vessel state as a programmable attribute – which we do not further define. The remainder includes all information about all containers, their bay/row/stack/tier positions, their bill-of-ladings, etc.

104 There may be other vessel attributes.

\[
\text{type} \\
V_{\text{Pos}} == \text{AtSea} \ | \ \text{InPort} \\
102a. \ \text{Longitude, Latitude} \\
102b. \ \text{InPort} :: \ \text{MCCI} \times \text{QPOS} \\
103 \ \forall \Sigma \\
104 \ ... \\
\text{value} \\
102 \ \text{attr}_{V_{\text{Pos}}} : V \rightarrow V_{\text{Pos}} \\
104 \ \text{attr}_{\forall \Sigma} : V \rightarrow \forall \Sigma \\
104 \ \text{attr}_{...} : V \rightarrow ...
\]

\text{axiom} \\
102b. \ \forall \ \text{mkInPort} (ti) : \text{InPort} \cdot ti \in t_{p\text{uis}}

5.6.5 Attributes: Quay Cranes

105 At any one time a quay crane may *programmably* hold a container or may not. We model the container held by a crane by the container identifier.

106 At any one time a quay crane is *programmably* positioned in a quay position within a quay segment.

107 Quay cranes may have other attributes.

\[
\text{type} \\
Q_{\text{Chold}} == \text{mkNil('nil')} \ | \ \text{mkCon(ci:CI)} \\
105 \ \text{QCPos} = \text{QSID} \times \text{QP} \\
107 \ ... \\
\text{value} \\
105 \ \text{attr}_{Q_{\text{Chold}}} : QC \rightarrow Q_{\text{Chold}} \\
106 \ \text{attr}_{Q_{\text{CPos}}} : QC \rightarrow Q_{\text{CPos}} \\
107 \ ...
\]
5.6.6 Attributes: Quay Trucks

At any one time a land truck may programmably hold a container or may not. We model the container held by a quay truck by the container identifier.

Quay trucks may have other attributes.

Note that we do not here model the position of quay trucks.

type
108 QTHold == mkNil('nil') | mkCon(ci:CI)
109 ...

value
108 attr_QTHold: QT → QTHold
109 ...

5.6.7 Attributes: Terminal Stack Cranes

At any one time a stack crane may programmably hold a container or may not. We model the container held by a crane by the container identifier.

Stack cranes are programmably positioned at a terminal bay.

Stack cranes may have other attributes.

type
110 SCHold == mkNil('nil') | mkCon(ci:CI)
111 SCPos = BI
111 ...

value
110 attr_SCHold: SC → SCHold
111 attr_SCPos: SC → SCPos
112 ...

5.6.8 Attributes: Container Stowage Areas

Bays of container storage areas statically have total order.

Rows of bays statically have total order.

Stacks of rows statically have total order.
We abstract orderings in two ways.

**type**

113 \( \text{BOm} = BI \rightarrow \text{Nat} \), \( \text{BOl} = BI^* \)

114 \( \text{ROm} = RI \rightarrow \text{Nat} \), \( \text{ROl} = RI^* \)

115 \( \text{SOm} = SI \rightarrow \text{Nat} \), \( \text{SOL} = SI^* \)

**axiom**

113 \( \forall \ b_{\text{om}}: \text{BOm} \cdot \text{rng} b_{\text{om}} = \{1: \text{card dom} b_{\text{om}}\} \), \( \forall \ b_{\text{ol}}: \text{BOl} \cdot \text{inds} b_{\text{ol}} = \{1: \text{len} b_{\text{ol}}\} \)

114 \( \forall \ r_{\text{om}}: \text{ROm} \cdot \text{rng} r_{\text{om}} = \{1: \text{card dom} r_{\text{om}}\} \), \( \forall \ r_{\text{ol}}: \text{ROl} \cdot \text{inds} r_{\text{ol}} = \{1: \text{len} r_{\text{ol}}\} \)

115 \( \forall \ s_{\text{om}}: \text{SOm} \cdot \text{rng} s_{\text{om}} = \{1: \text{card dom} s_{\text{om}}\} \), \( \forall \ s_{\text{ol}}: \text{SOL} \cdot \text{inds} s_{\text{ol}} = \{1: \text{len} s_{\text{ol}}\} \)

**value**

113 \( \text{attr.}_{\text{BOm}}: \text{CSA} \rightarrow \text{BOm} \), \( \text{attr.}_{\text{BOl}}: \text{CSA} \rightarrow \text{BOl} \)

114 \( \text{attr.}_{\text{ROm}}: \text{BAY} \rightarrow \text{ROm} \), \( \text{attr.}_{\text{ROl}}: \text{BAY} \rightarrow \text{ROl} \)

115 \( \text{attr.}_{\text{SOm}}: \text{ROW} \rightarrow \text{SOm} \), \( \text{attr.}_{\text{SOL}}: \text{ROW} \rightarrow \text{SOL} \)

CSAs, BAYs, ROWs and STKs have (presently further) static descriptions\(^6\) and terminal and vessel container stowage areas have definite numbers

116 of bays,

117 and any one such bay a definite number of rows,

118 and any one such row a definite number of stacks,

119 and any one such stack a maximum loading of containers.

**type**

116 \( \text{CASd} \)

117 \( \text{BAYd} \)

118 \( \text{ROWd} \)

119 \( \text{STKd} \)

**value**

116 \( \text{attr.}_{\text{CSAd}}: \text{CSA} \rightarrow BI \rightarrow \text{BI} \rightarrow \text{BI} \) \( \text{CSAd} \)

117 \( \text{attr.}_{\text{BAYd}}: \text{BAY} \rightarrow RI \rightarrow \text{RI} \rightarrow \text{RI} \) \( \text{BAYd} \)

118 \( \text{attr.}_{\text{ROWd}}: \text{ROW} \rightarrow SI \rightarrow \text{SI} \rightarrow \text{SI} \) \( \text{ROWd} \)

119 \( \text{attr.}_{\text{STKd}}: \text{STK} \rightarrow (\text{Nat} \times \text{STKd}) \)

---

\(^6\)Such descriptions include descriptions of for what kind of containers a container stowage area, a bay, a row and a stack is suitable: flammable, explosives, etc.
5.6.9 Attributes: Land Trucks

At any one time a land truck may *programmably* hold a container or may not. We model the container held by a land truck by the container identifier.

Land trucks also possess a further undefined *programmable* land truck state.

Land trucks may have other attributes.

Note that we do not here model the position of land trucks.

\[
\text{type} \quad \text{LTHold} \equiv \text{mkNil('nil')} \mid \text{mkCon(ci:CI)}
\]

\[
\text{value} \quad \text{attr}_\text{LTHold}: \text{LT} \to \text{LTHold}
\]

5.6.10 Attributes: Command Center

The *syntactic description*\(^7\) of the spatial positions of quays, cranes and the container storage area of a terminal, \text{TopLogDescr}, is a *static* attribute.

The *syntactic description*\(^8\) of the terminal state, i.e., the actual positions and deployment of vessels at quays, quay and stack cranes, quay and land trucks, and the actual container "contents" of these, \text{Term}\(\Sigma\)Descr, is a *programmable* attribute.

\[
\text{type} \quad \text{TopLogDescr}
\]

\[
\text{value} \quad \text{attr}_\text{TopLogDescr}: \text{MCC} \to \text{TopLogDescr}
\]

\[
\text{value} \quad \text{attr}_\text{Term}\(\Sigma\)Descr: \text{MCC} \to \text{Term}\(\Sigma\)Descr
\]

\(^7\) A *syntactic description* describes something, i.e., has some *semantics*, from which it is, of course, different.

\(^8\) The *syntactic description* of the terminal state is, of course, not that state, but only its description. The terminal state is the combined states of all cranes, trucks and the container storage area.
5.6.11 Attributes: Containers

A Bill-of-Lading⁹ is a static container attribute.¹⁰

<table>
<thead>
<tr>
<th>type</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BoL</td>
<td>attr_BoL: C → BoL</td>
</tr>
</tbody>
</table>

At any one time a container is positioned either

- in a stack on a vessel: at sea or in a terminal, or
- on a quay crane in a terminal port, being either unloaded from or loaded onto a vessel, or
- on a quay truck to or from a quay crane, i.e., from or to a stack crane, in a terminal port, or
- on a stack crane in a terminal port, being either unloaded from a quay truck onto a terminal stack or loaded from a terminal stack onto a quay truck, or
- on a stack in a terminal port, or
- on a land truck, or
- idle.

A container position is a programmable attribute.

There are other container attributes. For convenience we introduce an aggregate attribute: CAttrs for all attributes.

---

⁹https://en.wikipedia.org/wiki/Bill_of_lading: A bill of lading (sometimes abbreviated as B/L or BoL) is a document issued by a carrier (or their agent) to acknowledge receipt of cargo for shipment. In British English, the term relates to ship transport only, and in American English, to any type of transportation of goods. A bill of Lading must be transferable, and serves three main functions: it is a conclusive receipt, i.e. an acknowledgment that the goods have been loaded; and it contains or evidences the terms of the contract of carriage; and it serves as a document of title to the goods, subject to the nemo dat rule. Bills of lading are one of three crucial documents used in international trade to ensure that exporters receive payment and importers receive the merchandise. The other two documents are a policy of insurance and an invoice. Whereas a bill of lading is negotiable, both a policy and an invoice are assignables.

In international trade outside of the USA, Bills of lading are distinct from waybills in that they are not negotiable and do not confer title. The nemo dat rule: that states that the purchase of a possession from someone who has no ownership right to it also denies the purchaser any ownership title.

¹⁰For waybills see https://en.wikipedia.org/wiki/Waybill: A waybill (UIC) 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, and route. Most freight forwarders and trucking companies use an in-house waybill called a house bill. These typically contain "conditions of contract of carriage" terms on the back of the form. These terms cover limits to liability and other terms and conditions.
6 Perdurants

We refer to [13, Sect. 7].

6.1 A Modelling Decision

In the transcendental interpretation of parts into behaviours we make the following modelling decisions: All atomic and all composite parts become separate behaviours. But there is a twist. Vessels and terminal stacks are now treated as “atomic” behaviours. Containers that up till now were parts of container stowage areas on vessels and in terminal stacks are not behaviours embedded in the behaviours of vessels and terminal stacks, but are “factored” out as separate, atomic behaviours.

This modelling decision entails that container stowage areas, CSA\(_s\), of vessels and terminal stacks are modelled by replacing the [physical] containers of these CSA\(_s\) with virtual container stowage areas, vir\(_{CSA}\)s. Where there “before” were containers there are now, instead, descriptions of these: their unique identifiers, their mereology, and their attributes.

6.2 Virtual Container Storage Areas

In our transition from endurants to perdurants we shall thus need a notion of container stowage areas which, for want of a better word, we shall call virtual CSAs. Instead of stacks embodying containers, they embody...
128 container information: their unique identifier, mereology and attributes.

We must secure that no container is referenced more than once across the revised-model;

129 that is, that all ci:CIs are distinct.

**type**

5' vir CSA

11' vir BAY_s = vir BAY-set, vir BAY

12' vir ROW_s = vir ROW-set, vir ROW = vir STK-set

13' vir STK = vir STK-set, vir STK

14' vir STK = ClInfo*

128 ClInfo = CI × CMereo × CAttrs

**value**

5' attr_vir CSA: TP → vir CSA

11' attr_vir BAY_s: vir CSA → vir BAY_s, vir BAY_s = vir BAY-set, vir BAY

11' uid_vir BAY: vir BAY → BI

12' attr_vir ROW_s: vir BAY → vir ROW_s

**axiom**

129 [ all CIs of all vir CSAs are distinct ]

### 6.3 Changes to The Parts Model

We revise the parts model of earlier:

**type**

2 STPs, TPs = TP-set, TP

3 SVs, Vs = V-set, V

**value**

2 obs STPs: CLI → STPs, obs TPs: STPs → TPs

3 obs SVs: CLI → SVs, obs Vs: SVs → Vs

We treat the former CSAs of terminal ports as a composite, concrete part, vir BAY_m consisting of a set of atomic virtual bays, vir BAY.

**type**

11' vir BAY_s = vir BAY-set, vir BAY

**value**

5 obs BAY_s: TP → vir BAY_s

5 uid BAY: vir BAY → BI
And we treat the former CSAs of vessels as a programmable attribute of vessels:

\[
\text{attr\_vir\_CSA}: V \rightarrow \text{vir\_CSA}
\]

### 6.4 Basic Model Parts

There are \(c_n\) container behaviours, where \(c_n\) is the number of all containers of the system we are modelling. For each terminal port there is 1 controller behaviour, \(v_n\) vessel behaviours, where \(v_n\) is the number of vessels visiting that terminal port, \(qc_n\) quay crane behaviours, where \(qc_n\) is the number of quay cranes of that terminal port, \(qt_n\) quay truck behaviours, where \(qt_n\) is the number of quay trucks of that terminal port, \(lt_n\) land truck behaviours, where \(lt_n\) is the number of land trucks (of that terminal port), and \(tb_n\) terminal stack behaviours, where \(ts_n\) is the number of terminal bays of that terminal port.

The vessel, the land truck and the terminal monitoring & control [command] center behaviours are pro-active: At their own initiative (volition), they may decide to communicate with other behaviours. The crane, quay truck, stack and container behaviours are passive: They respond to interactions with other behaviours.

### 6.5 Actions, Events, Channels and Behaviours

We refer to [13, Sect. 7.1].

In building up to the behavioral analysis & description of the terminal container domain we first analyse the actions and events of that domain. These actions and events are the building blocks of behaviours.

---

11 The labeling \(A, B, C, D, ..., X, Y\) may seem arbitrary, but isn’t!
**Actions**, to remind the reader, are explicitly performed by an actor, i.e., a behaviour, calculates some values and, usually, effect a state change.

**Events** “occur to” actors (behaviours), that is, are not initiated by these, but usually effect state changes.

### 6.6 Actions

We refer to [13, Sects. 7.1.5, 7.3.1].

The unloading of containers from and the loading of container onto container stowage areas are modelled by corresponding actions on virtual container stowage areas. Vessels, land trucks and terminal monitoring & control centers, i.e., command centers, are here modelled as the only entities that can *initiate* actions.

#### 6.6.1 Command Center Actions

**Motivating the Command Center Concept:** We refer to the \[[A,B,...,U]\] labeled arrows of the figure on Page 44.

Imagine a terminal port. It has several vessels berthed along quays. It also has quay space, i.e., positions, for more vessels to berth. Berthed vessels are being serviced by several, perhaps many quay cranes. The totality of quay cranes are being serviced by [many more] quay trucks. The many quay trucks service several terminal bays, i.e., *stacks*. Land trucks are arriving, attending *stacks* and leaving. Quite a “busy scene”. So is the case for all container terminal ports.

The concept of a *monitoring & control*, i.e., a *command center*, is an abstract one; the figure on Page 44 does not show a part with a *center* label. The *actions* of vessels and trucks, and the *events* of cranes, terminal stacks and trucks are either hap-hazard, no-one interferes, they somehow “just happen”, or they are somehow co-ordinated.

Whether “free-wheeling” or “more-or-less coordinated” we can think of a *command center* as somehow *monitoring and controlling actions and events*.

Terminal *monitoring & control centers*, also interchangeably referred to as *command centers*, are thus where the logistics of container handling takes place.

You may think of this command center as receiving notices from vessels and land trucks as to their arrival and with information about their containers; thus building up awareness, i.e., a state, of the containers of all incoming and arrived vessels and land trucks, the layout of the terminal and the state of its container stowage area, the current whereabouts of vessels, cranes and trucks. Quite a formidable “state”.

We shall therefore model the “comings” and “goings” of vessels, trucks, cranes and stacks as if they were monitored and controlled by a command center, In our modelling we are not assuming any form of efficiency; there is, as yet no notion of optimality, nor of freedom from mistakes and errors. Our modelling – along these
Calculate Next Transaction: The core action of the command center is `calc_nxt_transaction`. We shall define `calc_nxt_transaction` only by its signature and a pair of pre/post conditions. In this way we do not have to consider efficiency, security, safety, etc., issues. These, i.e., the efficiency, security, safety, etc., issues can “always” be included in an requirements engineering implementation of `calc_nxt_transaction`. Basically the `calc_nxt_transaction` has to consider which of a non-trivially large number of possible actions have to be invoked. They are listed in Items 131 to 137 below. The `calc_nxt_transaction` occurs in time, and occur repeatedly, endlessly, i.e., “ad-infinitum”, At any time that `calc_nxt_transaction` is invoked the monitoring and control command center (`mcc`) is in some state. That state changes as the result of both monitoring actions and control actions. The `calc_nxt_transaction` therefore non-deterministically-internally chooses one among several possible alternatives. If there is no alternative, then a skip action is performed.

The command center, `mcc`, models the following actions and events: [A] the update of the `mcc` state, `mccσ`, in response to the vessel action that inform the `mcc` of the vessel arrival.

130 The result of a `calc_nxt_transaction` is an transaction designator, `MCCTrans` and a state change. There are several alternative designators. We mention some:

131 [B]: the calculation of vessel positions for [their] arrivals;

132 [CDE]: the calculation of vessel to quay crane container transfers;

133 [FGH]: the calculation of quay crane to quay truck container transfers;

134 [IJK]: the calculation of quay truck to stack crane container transfers;

135 [LMN]: the calculation of stack crane to stack container transfers;

136 [OPQ]: the calculation of land truck to stack crane container transfers;

137 [X]: the calculation that stowage, for a given vessel, has completed; and

138 the calculation that there is no next transaction that can be commenced.

139 The signature of the `calc_nxt_transaction` involves the unique identifier, mereology, static and programmable attributes, i.e., the state of the command center, and indicates that a command center transaction results and a next state “entered”.

© Dines Bjørner. 2018, Fredsvæj 11, DK–2840 Holte, Denmark – November 22, 2018: 11:05 am
For this, the perhaps most significant action of the entire container terminal port operation, we “skirt” the definition and leave to a pair of pre/post conditions that of characterising the result and next state.

**type**

\[
\begin{align*}
\text{MCCTrans} & = \text{QayPos} | \text{VSQC}_Xfer | \text{QCQT}_Xfer | \text{QTSC}_Xfer \\
\text{SCSTK}_Xfer | \text{SCLT}_Xfer | \text{LT}_\text{Dept} | \text{VS}_\text{Dept} | \text{Skip}
\end{align*}
\]

- **[B]:** \(\text{QuayPos}::\text{VI} \times \text{QPos}\)
- **[CDE]:** \(\text{VSQC}_Xfer::\text{VI} \times \text{BRS} \times \text{CI} \times \text{QCI}\)
- **[FGH]:** \(\text{QCQT}_Xfer::\text{QCI} \times \text{CI} \times \text{QTI}\)
- **[IJK]:** \(\text{QTSC}_Xfer::\text{QTI} \times \text{CI} \times \text{SCI}\)
- **[LMN]:** \(\text{SCSTK}_Xfer::\text{SCI} \times \text{CI} \times \text{BRS}\)
- **[OPQ]:** \(\text{SCLT}_Xfer::\text{SCI} \times \text{CI} \times \text{LTI}\)
- **[X]:** \(\text{VS}_\text{Dept}::\text{VI}\)
- **[Z]:** \(\text{Skip}::\text{nil}\)

**value**

\[
\begin{align*}
\text{calc}_\text{nxt}_\text{transaction}: \text{MCCI} \times \text{mereoMCC} \times \text{statMCC} \rightarrow \text{MCC}_\Sigma \rightarrow \text{MCCTrans} \times \text{MCC}_\Sigma
\end{align*}
\]

\[
\begin{align*}
\text{calc}_\text{nxt}_\text{transaction}(\text{mcci},\text{mccmereo},\text{mmstat})(\text{mcc}_\sigma) & \text{ as } (\text{mcctrans},\text{mcc}_\sigma') \\
\text{pre}: & \quad P_{\text{calc}_\text{nxt}_\text{trans}}((\text{mcci},\text{mccmereo},\text{mmccstat})(\text{mcc}_\sigma)) \\
\text{post}: & \quad Q_{\text{calc}_\text{nxt}_\text{trans}}((\text{mcci},\text{mccmereo},\text{mcctstat})(\text{mcc}_\sigma))(\text{mcctrans},\text{mcc}_\sigma')
\end{align*}
\]

The above mentioned actions are invoked by the command center in its endeavor to see containers moved from vessels to customers. A similar set of actions affording movement of containers customers to vessels, i.e., in the reverse direction: from land trucks to stack cranes, from stacks to quay trucks, from quay trucks to quay cranes, and from quay cranes to vessels, round off the full picture of all command center actions.

**Command Center Action [A]: update_mcc_from_vessel:**

141 Command centers

142 upon receiving arrival information, \(v\_\text{info}\), from arriving vessels, \(v\_i\), can update their state “accordingly”.

143 We leave undefined the pre- and post-conditions.

**value**

\[
\begin{align*}
\text{update_mcc_from_vessel}: \text{VSMCC}\_\text{MSG} \times \text{MCC}_\Sigma \rightarrow \text{MCC}_\Sigma
\end{align*}
\]

\[
\begin{align*}
\text{update_mcc_from_vessel}(\text{vs}_i,\text{vir}_\text{csa},\text{vs}_i\_\text{info},\text{mcc}_\sigma) & \text{ as } \text{mcc}_\sigma' \\
\text{pre}: & \quad P_{\text{update_mcc}_f}(\text{vs}_i,\text{vir}_\text{csa},\text{vs}_i\_\text{info},\text{mcc}_\sigma) \\
\text{post}: & \quad Q_{\text{update_mcc}_f}(\text{vs}_i,\text{vir}_\text{csa},\text{vs}_i\_\text{info},\text{mcc}_\sigma)(\text{mcc}_\sigma')
\end{align*}
\]
Command Center Action [B]: calc_ves_pos:

144 Command centers
145 can calculate, \( q_{\text{pos}} \), the quay segment and quay positions for an arriving vessel, \( v_i \).
146 We leave undefined the pre- and post-conditions.

value
144 \( \text{calc}_\text{ves}_\text{pos} : \text{MCCI} \times \text{MCC}_\text{mero} \times \text{TopLog} \times \text{MCC}_\Sigma \times \text{VI} \rightarrow (\text{QSId} \times \text{QP}^*) \times \text{MCC}_\Sigma \)
145 \( \text{calc}_\text{ves}_\text{pos}(\text{mcc}_i, \text{mcc}_\text{mero}, \text{toplog}, \text{mcc}_\sigma, v_j) \text{ as } (q_{\text{pos}}, \text{mcc}_\sigma') \)
146 \( \text{pre: } P_{\text{calc}_\text{ves}_\text{pos}}(\text{mcc}_i, \text{mcc}_\text{mero}, \text{toplog}, \text{mcc}_\sigma, v_j) \)
146 \( \text{post: } Q_{\text{calc}_\text{ves}_\text{pos}}(\text{mcc}_i, \text{mcc}_\text{mero}, \text{toplog}, \text{mcc}_\sigma, v_j)(q_{\text{pos}}, \text{mcc}_\sigma') \)

Command Center Action [C-D-E]: calc_ves_qc

147 The command center non-deterministically internally calculates
148 a pair of a triplet: the bay-row-stack coordinates, \( \text{brs} \), from which a top container, supposedly \( \text{ci} \), is to be removed by quay crane \( q_{\text{ci}} \), and a next command center state reflecting that calculation (and that the identified quay crane is being so alerted).
149 We leave undefined the relevant pre- and post-conditions

value
147 \( \text{calc}_\text{ves}_\text{qc} : \text{MCC}_\Sigma \rightarrow (\text{BRS} \times \text{CI} \times \text{QCI}) \times \text{MCC}_\Sigma \)
148 \( \text{calc}_\text{ves}_\text{qc}(\text{mcc}\sigma) \text{ as } ((\text{brs}, \text{ci}, q_{\text{ci}}), \text{mcc}\sigma') \)
149 \( \text{pre: } P_{\text{calc}_\text{ves}_\text{qc}}(\text{mcc}\sigma) \)
149 \( \text{post: } Q_{\text{calc}_\text{ves}_\text{qc}}(\text{mcc}\sigma)((\text{brs}, \text{ci}, q_{\text{ci}}), \text{mcc}\sigma') \)

Command Center Action [F-G-H]: calc_qc_qt

150 The command center non-deterministically internally
151 calculates a pair of a triplet: the identities of the quay crane from which and the quay truck to which the quay crane is to transfer a container, and an update command center state reflecting that calculation (and that the identified quay crane, container and truck are being so alerted).
We leave undefined the relevant pre- and post-conditions

table

\[ \text{calc_qc_qt: } \text{MCC} \Sigma \rightarrow (\text{QCI} \times \text{CI} \times \text{QTI}) \times \text{MCC} \Sigma \]

\[ \text{calc_qc_qt(mcc_\sigma) as ((qci,ci,qti),mcc_\sigma')} \]

\[ \text{pre: } P_{\text{calc_qc_qt}(mcc_\sigma)} \]

\[ \text{post: } Q_{\text{calc_qc_qt}(mcc_\sigma)}((qci,ci,qti),mcc_\sigma') \]

Command Center Action [I-J-K]: \text{calc_qt_sc}

The command center non-deterministically internally calculates a pair of a triplet: the identities of a quay truck, a container, and a stack crane, and an update command center state reflecting that calculation (and that the identified quay truck, container and stack crane are being so alerted).

We leave undefined the relevant pre- and post-conditions

\[ \text{calc_qt_sc: } \text{MCC} \Sigma \rightarrow (\text{QTI} \times \text{CI} \times \text{SCI}) \times \text{MCC} \Sigma \]

\[ \text{calc_qt_sc(mcc_\sigma) as ((qti,ci,sci),mcc_\sigma')} \]

\[ \text{pre: } P_{\text{calc_qt_sc}(mcc_\sigma)} \]

\[ \text{post: } Q_{\text{calc_qt_sc}(mcc_\sigma)}((qti,ci,sci),mcc_\sigma') \]

Command Center Action [L-M-N]: \text{calc_sc_stack}

The command center non-deterministically internally calculates a pair:

a triplet of the identities of a stack crane, a container and a terminal bay/row/stack triplet and a new state that reflects this action.

We leave undefined the relevant pre- and post-conditions

\[ \text{calc_sc_stack: } \text{MCC} \Sigma \rightarrow (\text{SCI} \times \text{CI} \times \text{BRS}) \times \text{MCC} \Sigma \]

\[ \text{calc_sc_stack(mcc_\sigma) as ((sci,ci,brs),mcc_\sigma')} \]

\[ \text{pre: } P_{\text{calc_sc_stack}(mcc_\sigma)} \]

\[ \text{post: } Q_{\text{calc_sc_stack}(mcc_\sigma)}((sci,ci,brs),mcc_\sigma') \]
Command Center Action [N-M-L]: \texttt{calc\_stack\_sc}

159 The command center non-deterministically internally calculates a pair:

160 a triplet of a terminal bay/row/stack triplet and the identities of a container and
a stack crane, and a new state that reflects this action.

161 We leave undefined the relevant pre- and post-conditions

\begin{align*}
\text{value} \\
\text{calc\_stack\_sc}: \text{MCC} & \to (\text{BRS} \times \text{CI} \times \text{SCI}) \times \text{MCC} \\
\text{calc\_stack\_sc}(\text{mcc}\sigma) & \text{ as } ((\text{brs},\text{ci},\text{sci}),\text{mcc}\sigma') \\
\text{pre} & : P_{\text{calc\_stack\_sc}}(\text{mcc}\sigma) \\
\text{post} & : Q_{\text{calc\_stack\_sc}}(\text{mcc}\sigma)((\text{brs},\text{ci},\text{sci}),\text{mcc}\sigma')
\end{align*}

Command Center Action [O-P-Q]: \texttt{calc\_sc\_lt}

162 The command center non-deterministically internally calculates a pair:

163 a triplet of the identities of a stack crane, a container and a land truck, and a
new state that reflects this action.

164 We leave undefined the relevant pre- and post-conditions.

\begin{align*}
\text{value} \\
\text{calc\_sc\_lt}: \text{MCC} & \to (\text{BRS} \times \text{CI} \times \text{SCI}) \times \text{MCC} \\
\text{calc\_sc\_lt}(\text{mcc}\sigma) & \text{ as } ((\text{sci},\text{ci},\text{lti}),\text{mcc}\sigma') \\
\text{pre} & : P_{\text{calc\_sc\_lt}}(\text{mcc}\sigma) \\
\text{post} & : Q_{\text{calc\_sc\_lt}}(\text{mcc}\sigma)((\text{sci},\text{ci},\text{lti}),\text{mcc}\sigma')
\end{align*}

Command Center Action [Q-P-O]: \texttt{calc\_lt\_sc}

165 The command center non-deterministically internally calculates a pair:

166 a triplet of the identities of a land truck, a container and a stack crane, and a
new state that reflects this action.

167 We leave undefined the relevant pre- and post-conditions.

\begin{align*}
\text{value} \\
\text{calc\_lt\_sc}: \text{MCC} & \to (\text{BRS} \times \text{CI} \times \text{SCI}) \times \text{MCC} \\
\text{calc\_lt\_sc}(\text{mcc}\sigma) & \text{ as } ((\text{lti},\text{ci},\text{sci}),\text{mcc}\sigma') \\
\text{pre} & : P_{\text{calc\_lt\_sc}}(\text{mcc}\sigma) \\
\text{post} & : Q_{\text{calc\_lt\_sc}}(\text{mcc}\sigma)((\text{lti},\text{ci},\text{sci}),\text{mcc}\sigma')
\end{align*}
**Command Center: Further Observations**  Please observe the following: any terminal command center repeatedly and non-deterministically alternates between any and all of these actions. Observe further that: The intention of the pre- and post-conditions [Items 143, 146, 149, 152, 155, 158, 161, 167, and 164], express requirements to the command center states, $mcc_\sigma: mcc_\Sigma$, w.r.t. the information it must handle. Quite a complex state.

### 6.6.2 Container Storage Area Actions

We define two operations on virtual CSAs:

168 one of stacking (loading) a container, referred to by its unique identifier in a virtual CSA,

169 and one of unstacking (unloading) a container;

170 both operations involving bay/row/stack references.

```plaintext
type
170 $BRS = BI \times RI \times SI$

value
168 $load_{\text{Cl}}: \text{vir}_{\text{CSA}} \times BRS \times Cl \rightarrow \text{vir}_{\text{CSA}}$
168 $load_{\text{Cl}}(\text{vir}_{\text{csa}},(bi,ri,si),ci) \text{ as } \text{vir}_{\text{csa}}'$
168 $\text{pre: } P_{load}(\text{vir}_{\text{csa}},(bi,ri,si),ci)$
168 $\text{post: } Q_{load}(\text{vir}_{\text{csa}},(bi,ri,si),ci)(\text{vir}_{\text{csa}}')$
169 $unload_{\text{Cl}}: \text{vir}_{\text{CSA}} \times BRS \rightarrow Cl \times \text{vir}_{\text{CSA}}$
169 $unload_{\text{Cl}}(\text{vir}_{\text{csa}},(bi,ri,si)) \text{ as } (ci,\text{vir}_{\text{csa}}')$
169 $\text{pre: } P_{unload}(\text{vir}_{\text{csa}},(bi,ri,si))$
169 $\text{post: } Q_{unload}(\text{vir}_{\text{csa}},(bi,ri,si))(ci,\text{vir}_{\text{csa}}')$
```

### The Load Pre-/Post-Conditions

171 The virtual $\text{vir}_{\text{CSA}}$, i.e., $\text{vir}_{\text{csa}}$, must be well-formed;

172 the $ci$ must not be embodied in that $\text{vir}_{\text{csa}}$; and

173 the bay/row/stack reference, $(bi,ri,si)$ must be one of the [virtual] container stowage area.
value
168 \( P_{load}(\text{vir}_\text{csa},(\text{bi},\text{ri},\text{si}),\text{ci}) \equiv \)
171 well_formed(\text{vir}_\text{csa}) \quad \text{cf. 25–27 on Page 22}
172 \land \text{ci} \not\in \text{xtr}_\text{Cls}(\text{vir}_\text{csa}) \quad \text{cf. 49 on Page 26}
174 \land \text{valid}_{\text{BRS}}(\text{bi},\text{ri},\text{si})(\text{vir}_\text{csa})

174 valid_{\text{BRS}}: \text{BRS} \rightarrow \text{iCSA} \rightarrow \text{Bool}
174 valid_{\text{BRS}}(\text{bi},\text{ri},\text{si})(\text{vir}_\text{csa}) \equiv
174 \text{bi}\in \text{dom}_\text{vir}_\text{csa}\land\text{ri}\in \text{dom}_\text{vir}_\text{csa}(\text{bi})\land\text{si}\in \text{dom}(\text{vir}_\text{csa}(\text{bi}))(\text{ri})

174 The resulting \text{vir}_\text{CSA}, i.e., \text{vir}_\text{csa}', must have the same bay, row and stack
identifications, and
175 except for the designated bay, row and stack, must be unchanged.
176 The designated “before”, i.e., the stack before loading, must equal the tail of the
“after”, i.e., the loaded stack, and
177 the top of the “after” stack must equal the “input” argument container identifier.,

value
169 Q_{load}(\text{vir}_\text{csa},(\text{bi},\text{ri},\text{si}),\text{ci})(\text{vir}_\text{csa}') \equiv
174 \text{dom}_\text{vir}_\text{csa} = \text{dom}_\text{vir}_\text{csa}'
174 \land \forall \text{bi}\':\text{BI}\cdot\text{bi}'\in \text{dom}_\text{vir}_\text{csa}(\text{bi}')
174 \Rightarrow \text{dom}_\text{vir}_\text{csa}(\text{bi}')=\text{dom}_\text{vir}_\text{csa}'(\text{bi}')
174 \land \forall \text{ri}':\text{RI}\cdot\text{ri}'\in \text{dom}_\text{vir}_\text{csa}(\text{bi})
174 \Rightarrow \text{dom}_\text{vir}_\text{csa}(\text{bi})(\text{ri}')=(\text{dom}_\text{vir}_\text{csa}'(\text{bi}'))(\text{ri}')
174 \land \forall \text{si}':\text{SI}\cdot\text{si}'\in \text{dom}_\text{vir}_\text{csa}(\text{ri}')
174 \Rightarrow (\text{vir}_\text{csa}(\text{bi}))(\text{ri}')=(\text{vir}_\text{csa}'(\text{bi}'))(\text{ri}')
174 \land \forall \text{bi}':\text{BI}\cdot\text{bi}'\in \text{dom}_\text{vir}_\text{csa} \setminus \{\text{bi}\}
175 \Rightarrow \text{vir}_\text{csa} \setminus \{\text{bi}\}=\text{vir}_\text{csa}' \setminus \{\text{bi}'\}
175 \land \forall \text{bi}'\':\text{BI}\cdot \text{bi}'\in \text{dom}_\text{vir}_\text{csa}(\text{bi}) \setminus \{\text{ri}\}
175 \Rightarrow ((\text{vir}_\text{csa}(\text{bi}))(\text{ri}'))=((\text{vir}_\text{csa}'(\text{bi}'))(\text{ri}'))
175 \land \forall \text{si}':\text{SI}\cdot \text{si}'\in \text{dom}_\text{vir}_\text{csa}(\text{ri}') \setminus \{\text{si}\}
175 \Rightarrow ((\text{vir}_\text{csa}(\text{bi}'))(\text{si}'))=((\text{vir}_\text{csa}'(\text{bi}'))(\text{si}'))
176 \land \text{tl}((\text{vir}_\text{csa}'(\text{bi}'))(\text{si}'))=((\text{vir}_\text{csa}'(\text{bi}'))(\text{si}'))
177 \land \text{hd}((\text{vir}_\text{csa}'(\text{bi}'))(\text{si}'))=\text{ci}
The Unload Pre-/Post-Conditions

178 The virtual \(\text{vir\_csa}\), i.e., \(\text{vir\_csa}\),
179 must be wellformed; and
180 the bay/row/stack reference, \((bi,ri,si)\) must be one of the [virtual] container stowage area.

\[\begin{align*}
\text{value} \\
178 & \ P_{\text{unload}}(\text{vir\_csa},(bi,ri,si)) \equiv \\
179 & \ \text{well\_formed}(\text{vir\_csa}) \\
180 & \ \land \ \text{valid\_BRS}(bi,ri,si)(\text{vir\_csa})
\end{align*}\]

\[\begin{align*}
\text{value} \\
169 & \ Q_{\text{unload}}(\text{vir\_csa},(bi,ri,si))(ci,\text{vir\_csa'}) \equiv \\
181 & \ \text{dom} \ \text{vir\_csa} = \text{dom} \ \text{vir\_csa'} \\
182 & \ \land \ \forall \ bi':BI \cdot bi' \in \text{dom} \ \text{vir\_csa} \setminus \{bi\} \\
183 & \ \Rightarrow \ \text{vir\_csa} \setminus \{bi\} = \text{vir\_csa'} \setminus \{bi\} \\
184 & \ \land \ \forall \ ri':RI \cdot ri' \in \text{dom} \ \text{vir\_csa}(bi) \setminus \{ri\} \\
185 & \ \Rightarrow \ ((\text{vir\_csa}(bi))(ri')) = ((\text{vir\_csa'}(bi))(ri')) \\
186 & \ \land \ \forall \ si':SI \cdot si' = \text{dom} \ (\text{vir\_csa})(ri') \setminus \{si\} \\
187 & \ \Rightarrow \ ((\text{vir\_csa}(bi'))(si')) = ((\text{vir\_csa'}(bi'))(si')) \\
188 & \ \land \ ((\text{vir\_csa'}(bi'))(si') = \text{tl}((\text{vir\_csa'}(bi'))(si')) \\
189 & \ \land \ \text{hd}((\text{vir\_csa}(bi'))(si') = ci
\end{align*}\]

6.6.3 Vessel Actions

Vessels (and land trucks) are in a sense, the primary movers in understanding the terminal container domain. Containers are, of course, at the very heart of this domain. But without container vessels (and land trucks) arriving at ports \textit{nothing would happen!} So the actions of vessels are those of actively announcing their arrivals at and departures from ports, and participating, more passively, in the unloading and loading of containers.
**Action [A]: calc_next_port:**

186 Vessels can calculate, `calc_next_port`, the unique identifier, `mcc_i`, of that ports’ monitoring & control center.

187 We do not further define the pre- and post-conditions of the `calc_next_port` action.

**Value**

186 \[\text{calc}_{\text{next-port}}: \text{VI} \times \text{VS}_{\text{Mereo}} \times \text{VS}_{\text{Stat}} \rightarrow \text{vir}_{\text{CSA}} \times \text{VS}_{\sigma} \rightarrow \text{MCC} \times \text{VS}_{\Sigma}\]

186 \[\text{calc}_{\text{next-port}}(\text{vs}_i, \text{vs}_{\text{Mereo}}, \text{vs}_{\text{Stat}})(\text{vir}_{\text{CSA}}, \text{vs}_{\sigma}) \text{ ia } (\text{mcc}_i, \text{vs}_{\sigma}')\]

187 \[\text{pre: } \mathcal{P}_{\text{calc}_{\text{next-port}}}(\text{vs}_i, \text{vs}_{\text{Mereo}}, \text{vs}_{\text{Stat}})\]

187 \[\text{post: } \mathcal{Q}_{\text{calc}_{\text{next-port}}}(\text{vs}_i, \text{vs}_{\text{Mereo}}, \text{vs}_{\text{Stat}})(\text{mcc}_i, \text{vs}_{\sigma}')\]

**Vessel Action [B]: calc_ves_msg:**

188 Vessels can calculate, `calc_ves_info`, the vessel information, `vs_info:VS_{Info}`, to be handed to the next ports’ command center.

189 This information is combined with the vessel identifier and its virtual CSA,

190 We leave undefined the pre- and post-conditions over vessel states and vessel information.

**Type**

188 `VS_{Info}`

189 `VS_{MCC_{MSG}} :: \text{VI} \times \text{vir}_{\text{CSA}} \times \text{VS}_{\text{Info}}`

**Value**

188 \[\text{calc}_{\text{ves-msg}}: \text{VI} \times \text{VMereo} \times \text{VStat} \rightarrow \text{VS}_{\text{Pos}} \times \text{vir}_{\text{CSA}} \times \text{VS}_{\Sigma} \rightarrow \text{VS}_{\text{MCC}_{\text{MSG}}} \times \text{VS}_{\Sigma}\]

188 \[\text{calc}_{\text{ves-msg}}(\text{vs}_i, \text{vs}_{\text{Mereo}}, \text{vs}_{\text{Stat}})(\text{vpos}, \text{vir}_{\text{CSA}}, \text{vs}_{\sigma}) \text{ as } (\text{vs}_{\text{mcc_{msg}}}, \text{vs}_{\sigma}')\]

190 \[\text{pre: } \mathcal{P}_{\text{calc}_{\text{ves-msg}}}(\text{vs}_i, \text{vs}_{\text{Mereo}}, \text{vs}_{\text{Stat}})(\text{vpos}, \text{vir}_{\text{CSA}}, \text{vs}_{\sigma})\]

190 \[\text{post: } \mathcal{Q}_{\text{calc}_{\text{ves-msg}}}(\text{vs}_i, \text{vs}_{\text{Mereo}}, \text{vs}_{\text{Stat}})(\text{vpos}, \text{vir}_{\text{CSA}}, \text{vs}_{\sigma})(\text{vs}_{\text{mcc_{msg}}}, \text{vs}_{\sigma}')\]

### 6.6.4 Land Truck Actions

Land trucks can initiate the following actions vis-a-vis a targeted terminal port command center: announce, to a terminal command center, its arrival with a container; announce, to a terminal command center, its readiness to haul a container. Land trucks furthermore interacts with stack cranes – as so directed by terminal command centers.
Land Truck Action [R]: calc_truck_delivery:

191 Land trucks, upon approaching, from an outside, terminal ports, calculate
192 the identifier of the next port’s command center and a next land truck state.
We do not define the
193 pre- and
194 post conditions of this calculation.

value
191 calc_truck_delivery: CI × TRUCKΣ → MCCI × LTΣ
192 calc_truck_delivery(ci,ltσ) as (mcci,ltσ’)
193 pre: P_{calc_truck_delivery}(ci,ltσ)
194 post: Q_{calc_truck_delivery}(ci,ltσ)(mcci,ltσ’)

Land Truck Action [S]: calc_truck_avail:

195 Land trucks, when free, i.e., available for a next haul, calculate
196 the identifier of a suitable port’s command center and a next land truck state.
We do not define the
197 pre- and
198 post conditions of this calculation.

value
195 calc_truck_avail: LTI × LTΣ → MCCI × LTΣ
196 calc_truck_avail(lti,ltσ) as (mcci,ltσ’)
197 pre: P_{calc_truck_avail}(lti,ltσ)
198 post: Q_{calc_truck_avail}(lti,ltσ)(mcci,ltσ’)

6.7 Events

We refer to [13, Sect. 7.1.6 and 7.3.2]. Events occur to all entities. For reasons purely
of presentation we separate events into active part initiation events and active part
completion events. Active part initiation events are those events that signal the initi-
ation of actions. (Let [Θ] designate an action, then [Θ’] designates the completion of
that action.) Active part completion events are those events that signal the completion
of actions. We do not show the lower case [d, f, g, h, i, j, k, l, m, n, o] in Fig. 3.
6.7.1 Active Part Initiation Events

**Vessels:**

199 $[\alpha_{vessel}]$ approaching terminal port; ladings – and these actual unloads/ladings;

200 $[A]$ informing the command center, $mcc$, of a terminal port, of arrival;

201 $[B]$ receiving from an $mcc$ directions as to quay berth positions;

202 $[C]$ receiving from an $mcc$, for each container to be unloaded or loaded, directions as to these unloads and

203 $[X]$ receiving from an $mcc$ directions of completion of stowage (no more unloads/loads);

204 $[Y]$ informing the $mcc$ of its departure from terminal port; or

205 $[\omega_{vessel}]$ leaving a terminal port.

**Land Trucks:**

206 $[\alpha_{land\_truck}]$ approaching a terminal port; of a container;

207 $[W]$ informing its $mcc$ of its arrival;

208 $[V]$ being directed, by an $mcc$, as to the stack (crane) of destination;

209 $[S]$ the unloading, to a stack crane, 210 $[T]$ the loading of a container from a stack crane;

211 $[R]$ informing its $mcc$ of its departure; or

212 $[\omega_{land\_truck}]$ leaving a terminal port.

**Containers:** the transfers from

213 $[D]$ vessel to quay crane;

214 $[d]$ quay crane to vessel;

215 $[G]$ quay crane to quay truck;

216 $[g]$ quay truck to quay crane;

217 $[J]$ quay truck to stack crane;

218 $[j]$ stack crane to quay truck;

219 $[M]$ stack crane to stack;

220 $[m]$ stack to stack crane;

221 $[P]$ stack crane to land truck; or from

222 $[p]$ land truck to stack crane.

**Quay Cranes:** being informed, by the command center, $mcc$, of a container to be

223 $[E]$ picked-up from a vessel;

224 $[e]$ set-down on a vessel;

225 $[F]$ set-down on a quay truck; or

226 $[f]$ picked-up from a quay truck.
**Quay Trucks:** being informed, by the command center, *mcc*, of a container to be

227 [H] loaded from a quay crane; 229 [t] picked-up by a stack crane; or

228 [h] picked-up by a quay crane; 230 [i] loaded from a stack crane.

**[Terminal] Stack Cranes:** being informed, by the command center, *mcc*, of a container to be

231 [K] picked-up from a quay truck; 234 [l] loaded on to a stack;

232 [k] loaded on to a quay truck; 235 [O] picked-up from a land truck; or

233 [L] picked-up from a stack; 236 [o] loaded on to a land truck.

**[Terminal Bay] Stacks:** being informed, by the command center, *mcc*, of a container to be

237 [N] set-down, of a container, from a stack crane; or

238 [n] picked-up, of a container, by a stack crane.

These events, in most cases, prompt interaction with the terminal command center.

### 6.7.2 Active Part Completion Events:

We do not show, in Fig. 3, the c', e', h', o', q', t' events.

239 [C']

240 [E']

241 [H']

242 [O']

243 [Q']

244 [T']

### 6.8 Channels

We refer to [13, Sect. 7.2], and we refer to Sect. 5.2 and to Fig. 2 on Page 44.
6.8.1 Channel Declarations

There are channels between terminal port monitoring & control command center (mcci) and that command centers and that terminal port’s

245 all the containers (ci), that might visit the terminal port; ch\_mcc\_con[mcci,ci]^{12};

246 vessels (vi) that might visit that port, ch\_mcc[mcci,vi]^{13};

247 quay cranes (qci) of that port, ch\_mcc[mcci,qci]^{14};

248 quay trucks (qti) of that port, ch\_mcc[mcci,qti]^{15};

249 stack cranes (sci) of that port, ch\_mcc[mcci,sci]^{16};

250 stacks [bays] (stki) of that port, ch\_mcc[mcci,stki]^{17}; and

251 land trucks (lti) of, in principle, any port, ch\_mcc[mcci,lti]^{18}.

252 We shall define the concrete types of messages communicated by these channels subsequently (Sect. 6.8.2).

channel
245 \{ch\_mcc\_con[mcci,ci]|mcci:MCCI,ci:CI\in mcci\_uis\cap ci\in c\_uis\}:MCC\_Con\_Cmd

246-251 \{ch\_mcc[mcci,ui]|mcci:MCCI,ui:(VI|QCI|QTI|SCI|STKI|LTI)\in mcci\_uis\cap ui\in uis\}:MCC\_Msg

type
252 MCC\_Con\_Msg, MCC\_Msg

6.8.2 Channel Messages

We present a careful analysis description, for the channels declared above, of the rather rich variety of messages communicated over channels. All messages “goes to” (a few) or “comes from” (the rest) the command center. Messages from quay cranes, quay trucks, stack cranes, and land trucks – directed at the command center – are all in response to the events of their being loaded or unloaded.

\begin{footnotesize}
\begin{itemize}
\item[12] cf. Item 98 on Page 35
\item[13] cf. Item 76 on Page 30
\item[14] cf. Item 81 on Page 31
\item[15] cf. Item 83 on Page 31
\item[16] cf. Item 87 on Page 32
\item[17] cf. Item 94 on Page 34
\item[18] cf. Item 96 on Page 34
\end{itemize}
\end{footnotesize}
**A,B,X,Y,C': Vessel Messages**

253 There are a number command center – vessel and vice-versa messages:

a. **A**: Vessels announce their (forthcoming) arrival to the next destination terminal by sending such information, \( VSArrv \), to its monitoring & control (also referred to as command) center, that enables it to handle those vessels' berthing, unloading and loading (of container stowage).\(^1\)

b. **B**: The terminal command center informs such arriving vessels of their quay segment positions, \( VSQPos \).

c. **X**: The terminal command center informs vessels of completion of stowage handling, \( VSComp \).

d. **Y**: Vessels inform the terminal of their departure, \( VesDept \).

\[ type \]

253  \[ MCC\_Cmd \equiv VSArrv|VSQPos|VSComp|VesDept|... \]

253a. **A**: \( VSArrv :: VI \times vir\_CSA \)

253b. **B**: \( VSQPos :: VI \times (QSId \times QP^+) \)

253c. **X**: \( VSComp :: MCCI \times VI \)

253d. **Y**: \( VesDept :: MCCI \times VI \)

**C,D,E,E': Vessel/Container/Quay Crane Messages**

254 The terminal command center, at a time it so decides, “triggers” the simultaneous transitions, **C,D,E**, of

a. **C**: unloading (loading) from (to) a vessel stack position of a container (surrogate), \( VSQC\_Xfer, QC\_VS\_Xfer \),

b. **D**: notifying the physical, i.e., the actual container that it is being unloaded (loaded), \( C\_VS\_to\_QC, C\_QC\_to\_VS \), and

c. **E**: loading (unloading) the container (surrogate) onto (from) a quay crane, \( VStoQC, QCtoVS \).

255 **C',E':** The vessel and the quay crane, in response to their being unloaded, respectively loaded with a container “moves” that load, from its top vessel bay/row/stack position to the quay crane and notifies the terminal command center of the completion of that move, \( VSQC\_Compl \).

\(^{19}\)What exactly that information is, i.e., any more concrete type model of \( Ves\_Info \) cannot be given at this early stage in our development of **what a terminal is**.
F,G,H,H': Quay Crane/Container/Quay Truck Messages

The terminal command center, at a time it so decides “triggers” the simultaneous transitions, **F,G,H**: QCtoQT, of

a. **F**: the removal of the container from the quay crane,

b. **G**: the notification of the physical container that it is now being transferred to a quay truck, and

c. **H**: the loading of that container to a quay truck.

d. **H'**: The quay truck, in response to it being loaded notifies the terminal command center of the completion of that move.

I,J,K,K': Quay Truck/Container/Stack Crane Messages

The terminal command center, at a time it so decides “triggers” the simultaneous transitions, **I,J,K**: QTtoSC, of

a. **I**: the removal of a container from a quay truck,

b. **J**: the notification of the physical container that it is now being transferred to a stack crane, and

c. **K**: the loading of that container to a stack crane.
K': The stack crane, in response to it being loaded notifies the terminal command center of the completion of that move.

\[
\begin{align*}
\text{type} & \quad \text{MCC_Cmd} = \ldots \mid \text{QTtoSC} \mid \ldots \\
& \quad \text{QTtoSC} = \text{UnLoadCQT} \mid \text{NowConSC} \mid \text{QCQTCompl} \\
& \quad \text{UnLoadCQT} :: \text{CI} \times \text{QRI} \\
& \quad \text{NowConSC} :: \text{CI} \times \text{SCI} \\
& \quad \text{LoadCSC} :: \text{CI} \times \text{SCI} \\
& \quad \text{QCSCCompl} :: \ldots
\end{align*}
\]

L,M,N,N': Stack Crane/Container/Stack Messages

The terminal command center, at a time it so decides “triggers” the simultaneous transitions, \(L,M,N\): \(\text{SCtoStack}\), of

a. \(L\): the unloading of the container from a stack crane;

b. \(M\): the notification of the physical container that it is now being transferred to a stack, and

c. \(N\): the loading of that container to a stack.

N': The stack, in response to it being loaded, notifies the terminal command center of the completion of that move.

\[
\begin{align*}
\text{type} & \quad \text{MCC_Cmd} = \ldots \mid \text{SCtoStack} \mid \ldots \\
& \quad \text{SCtoStack} = \text{UnLoadCSC} \mid \text{NowConSTK} \mid \text{LoadConSTK} \mid \text{SCStkCompl} \\
& \quad \text{UnLoadCSC} :: \text{CI} \times \text{SCI} \\
& \quad \text{NowConSTK} :: \text{CI} \times \text{BRS} \\
& \quad \text{LoadConSTK} :: \text{CI} \times \text{BRS} \\
& \quad \text{SCStkCompl} :: \ldots
\end{align*}
\]

O,P,Q,Q': Land Truck/Container/Stack Crane Messages

The terminal command center, at a time it so decides “triggers” the simultaneous transitions, \(O,P,Q\): \(\text{LTtoSC}\), of

a. \(Q\): the unloading of the container from a land truck to a stack crane;

b. \(P\): the notification of the physical container that it is now being transferred to a stack crane, and
c. **O**: the loading of that container to a stack crane.

d. **O’**: The stack crane, in response to it being loaded, notifies the terminal command center of the completion of that move.²⁰

<table>
<thead>
<tr>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>261</td>
</tr>
<tr>
<td>261a.</td>
</tr>
<tr>
<td>261b.</td>
</tr>
<tr>
<td>261c.</td>
</tr>
<tr>
<td>261d.</td>
</tr>
</tbody>
</table>

**R,S,T,U,Q,V**: Land Truck Messages

262 These are the messages that are communicated either from land trucks to command centers or vice versa:

a. **R**: Land trucks, when approaching a terminal port, informs that port of its offer to deliver an identified container to stowage.

b. **S**: Land trucks, when approaching a terminal port, informs that port of its offer to accept (load) an identified container from stowage.

c. **T**: Land trucks, at a terminal, are informed by the terminal of the stack crane at which to deliver (unload) an identified container.

d. **U**: Land trucks, at a terminal, are informed by the terminal of the stack crane from which to accept an identified container.

e. **Q**: Land trucks, at a terminal, are informed by the terminal of the stack crane at which to unload (deliver) an identified container.

f. **q**: Land trucks, at a terminal, are informed by the terminal of the stack crane at which to load (accept) an identified container.

g. **V**: Land trucks, at a terminal, inform the terminal of their departure.

<table>
<thead>
<tr>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>262</td>
</tr>
<tr>
<td>262a.</td>
</tr>
</tbody>
</table>

²⁰The **O’** event is “the same” as the **K’** event.
6.9 Behaviours

We refer to [13, Sects. 7.1.7, 7.3.3-4-5, and 7.4].

To every part of the domain we associate a behaviour. Parts are in space: there are the manifest parts, and there are the notion of their corresponding behaviours. Behaviours are in space and time. We model behaviours as processes defined in \( \text{RSL}^+ \). We cannot see these processes. We can, however, define their effects.

Parts may move in space: vessels, cranes, trucks and containers certainly do move in space; processes have no notion of spatial location. So we must “fake” the movements of movable parts. We do so as follows: We associate with containers the programmable attribute of location, as outlined in Items 126–126g. on Page 41. We omit, for this model, the more explicit modelling of vessels, cranes and trucks but refer to their physical mereologies.

In the model of endurants, cf. Page 20, we modelled vessel and terminal container stowage areas as physically embodying containers, and we could move containers: push and pop them onto, respectively from bay stacks. This model must now, with containers being processes, be changed. The stacks, \( \text{STACK} \), of container stowage areas, \( \text{CAS} \), now embody unique container identifiers! We rename these stacks into \( \text{cistack:C} \)

6.9.1 Terminal Command Center

The terminal command center is at the core of activities of a terminal port. We refer to the figure on Page 44. “Reading” that figure left-to-right illustrates the movements of containers from \([C-D-E]\) vessels to quay cranes, \([F-G-H]\) quay cranes to quay trucks, \([I-J-K]\) quay trucks to stack cranes, \([L-M-N]\) stack cranes to stacks, and from \([O-P-Q]\) land truck to stack cranes. A similar “reading” of that figure from right-to-left would illustrate the movements of containers from \([q-p-o]\) stack cranes to land trucks; \([n-m-l]\) stacks to stack cranes; \([k-j-i]\) stack cranes to quay trucks; \([h-g-f]\) quay trucks to quay cranes; and from \([e-d-c]\) quay cranes to vessels. We have not show the \([c-d-e-f-g-h-i-j-k-l-m-n-o-p-q]\) labels, but their points should be obvious (!).

The Command Center Behaviour: We distinguish between the command center behaviour offering to monitor primarily vessels and land trucks, secondarily cranes, quay cranes and stacks, and offering to control vessels, cranes, trucks and containers.
The signature of the `command center` behaviour is a triple of the command center identifier, the conceptual command center mereology and the static command center attributes (i.e., the topological description of the terminal); the programmable command center attributes (i.e., the command center state); and the input/output channels for the command center.

The command center behaviour non-deterministically (externally) chooses between either monitoring inputs from vessels, cranes, trucks, stacks and containers.

The Command Center Monitor Behaviours: The command center monitors the behaviours of vessels, cranes and trucks: \([A, Y', C', E', F', H', I', K', L', N', O', Q']\). The input message thus received is typed:

\[
\text{type} \\
VCT\_\text{Info} = ... \\
\]

That information is used by the command center to update its state:

\[
\text{value} \\
\text{update}_\Sigma: \ VCT\_\text{Info} \rightarrow \Sigma \rightarrow \Sigma \\
\]

The definition of `monitoring` is simple.
The signature of the monitoring behaviour is the same as the command center behaviour.

The monitor non-deterministically externally ([]) offers to accept any input, \texttt{vct\_info}, message from any vessel, any land truck and from local terminal port quay trucks and cranes.

That input, \texttt{vct\_info}, enters the update of the command center state, from \texttt{mcc\sigma} to \texttt{mcc\sigma'}.

Whereupon the monitoring behaviour resumes being the command center behaviour with an updated state.

\begin{verbatim}
value
266  monitoring: mcci:MCCI × mis:MCC, Mereo × MCC, Stat

266  → MCCΣ

266  → in,out {chan.mcci[i] | i ∈ mis} Unit

266  monitoring(mcci, mis, mcc, stat)(mccσ) ≡

267  let vct_info = [] { chan.mcci[i] ? | i ∈ mis } in

268  let mccσ' = update_MCCΣ((vct_info, ui))(mccσ) in

269  command_center(mcci, mis, mcc, stat)(mccσ') end end
\end{verbatim}

\textbf{The Command Center Control Behaviours:}

The command center control behaviour has the same signature as the command center behaviour (formula Items 263).

In each iteration of the command center behaviour in which it chooses the control alternative it calculates\textsuperscript{21} a next [output] transaction. This calculation is at the very core of the overall terminal port. We shall have more to say about this in Sect. 7.1 on Page 79.

Items, 272a–272j, represent 10 alternative transactions.

They are “selected” by the case clause (Item 272).

So for each of these 10 alternatives there the command center offers a communication. For the [\texttt{CDE, FGH, IJK, LMN, OPQ, opq}] cases there is the same triple of concurrently synchronised events. For the [\texttt{B, T, X}] clauses there are only a single synchronisation effort. The command center events communicates:

\textsuperscript{21}For calc\_nxt\_transaction see Items 130 – 140 on Page 47
a. [B] the quay positions to arriving vessels, the transfer of containers
b. [CDE] from vessel stacks to quay cranes,
c. [FGH] quay cranes to quay trucks,
d. [IJK] quay trucks to stack cranes,
e. [LMN] stack cranes to stacks,
f. [OPQ] stack cranes to land trucks, and
g. [opq] land trucks to stack cranes.

We also illustrate h. [T] the bays to which a land truck is to deliver, or fetch a container, and i. [X] the "signing off" of a vessel by the command center.

j. For the case that the next transaction cannot be determined [at any given point in time] there is nothing to act upon.

273 After any of these alternatives the command center control behaviour resumes being the command center behaviour with the state updated from the next transaction calculation.
6.9.2 Vessels

The signature of the vessel behaviour is a triple of the vessel identifier, the conceptual vessel mereology, the static vessel attributes, and the programmable vessel attributes. [We presently leave static attributes unspecified: ...]

Nondeterministically externally, [], the vessel decides between

275 [A] either approaching a port,

276 [], or [subsequently] arriving at that port,

or [subsequently] participating in the

277 [], unloading and

278 [], loading of containers of containers,

279 [], or [finally] departing from that port.

value

274 vessel: vi:VI × mccis:V_Mereo × V_Sta_Attrs → (V_Pos × vir_CSA × VΣ)
274 → in,out {ch_mcc[mcci,vi] ∣ mcci:MCCI ⊔ mcci ∈ mccis} Unit
274 vessel(vi,mccis,...)(vpos,vir_csa,vσ) ≡
275 port_approach(vi,mccis,...)(vpos,vir_csa,vσ)
276 port_arrival(vi,mccis,...)(vpos,vir_csa,vσ)
277 unload_container(vi,mccis,...)(vpos,vir_csa,vσ)
278 load_container(vi,mccis,...)(vpos,vir_csa,vσ)
279 port_departure(vi,mccis,...)(vpos,vir_csa,vσ)
Port Approach

280 The signature of \texttt{port\_approach} behaviour is identical to that of \texttt{vessel} behaviour.

281 On approaching any port the vessel calculates the identity of that port’s command center.

282 Then, with an updated state, it calculates the information to be handed over to the designated terminal –

283 \[A\] which is then communicated from the vessel to the command center;

284 whereupon the vessel resumes being a vessel albeit with a doubly updated state.

value

280 \texttt{port\_approach}: \texttt{vi:VI × vs\_mer:VS\_Mereo × VS\_Stat} \rightarrow (VS\_Pos × vir\_CSA × VΣ)

281 \texttt{port\_approach(vi,vs\_mer,vs\_stat)}(vpos,vir\_csa,vs\_info) \equiv

282 \{ let (mcci,vs\_info) = \texttt{calc\_next\_port(vi,vs\_mer,vs\_stat)}(vpos,vir\_csa,vs\_info) \texttt{ in}

283 \} ch\_mcc[mcci,vi]! mkVS\_Info(vi,vir\_csa,vs\_info)

284 \texttt{vessel(vi,vs\_mer,vs\_stat)}(mkInPort(mcci,mkVS\_Pos(vi),(qs,cpl)),vir\_csa,vs\_info) \texttt{ end end}

Port Arrival

285 The signature of \texttt{port\_arrival} behaviour is identical to that of \texttt{vessel} behaviour.

286 \[B\] Non-deterministically externally the vessel offers to accept a terminal port quay position from any terminal port’s command center.

287 The vessel state is updated accordingly.

288 Whereupon the vessel resumes being a vessel albeit with a state updated with awareness of its quay position.

289 The vessel is ready to receive such quay position from any terminal port.

value

285 \texttt{port\_arrival}: \texttt{vi:VI × mccis:V\_Mereo × V\_Sta\_Attrs} \rightarrow (V\_Pos × vir\_CSA × VΣ)

286 \texttt{port\_arrival(vi,mccis,\ldots)}(vpos,vir\_csa,vs\_info) \equiv

287 \{ let mkVS\_Pos(vi,(qs,cpl)) = ch\_mcc[mcci,vi] ? in

288 \} ch\_mcc[mcci,vi]! mkVS\_Info(vi,vir\_csa,vs\_info)

289 \texttt{vessel(vi,mccis,\ldots)}(mkInPort(mcci,mkVS\_Pos(vi),(qs,cpl)),vir\_csa,vs\_info) \texttt{ end end}
Unloading of Containers

The signature of \texttt{port arriv} behaviour is identical to that of \texttt{vessel} behaviour.

The vessel offers to accept, \texttt{ch\_mcc\_v[mcci,vi] ?}, a directive from the command center of the terminal port at which it is berthed, to unload, \texttt{mkUn\_load((bi,ri,si),ci)}. a container, identified by \texttt{ci}, at some container stowage area location \((\texttt{bi,ri,si})\).

The vessel unloads the container – identified by \texttt{ci’}.

If the unloaded container identifier is different from the expected \texttt{chaos} erupts!

The vessel state, \texttt{v\_\sigma’}, is updated accordingly.

“Some time has elapsed since the unload directive, modelling” the completion, from the point of view of the vessel, of the unload operation – whereupon the command center is informed of this completion \((\texttt{mkCompl(mkV Un\_Load((bi,ri,si),ci))})\).

The vessel resumes being the vessel in a state reflecting the unload.

Loading of Containers

The signature of \texttt{load\_container} behaviour is identical to that of \texttt{vessel} behaviour.

The vessel offers to accept, \texttt{ch\_mcc\_v[mcci,vi] ?}, a directive from the command center of the terminal port at which it is berthed, to load, \texttt{mkLoad((bi,ri,si),ci)}. a container, identified by \texttt{ci}, at some container stowage area location \((\texttt{bi,ri,si})\).
The vessel (in co-operation with a quay crane, see later) then unloads the container – identified by $c_i$.

The vessel state, $v_{\sigma'}$, is updated accordingly.

“Some time has elapsed since the unload directive, modelling” the completion, from the point of view of the vessel, of the unload operation – whereupon the command center is informed of this completion ($'c'$).

and the vessels resumes being the vessel in a state reflecting the load.

The signature of port departure behaviour is identical to that of vessel behaviour.

At some time some command center informs a vessel that stowage, i.e., the unloading and loading of containers has ended.

Vessels update their states accordingly.

Vessels respond by informing the command center of their departure.

Whereupon vessels resume being vessels.
The next three behaviours: quay_cane, quay_truck and stack_cane, are very similar. One substitutes, line-by-line, command center/quay crane, quay crane/quay truck, quay truck/stack crane et cetera!

6.9.3 Quay Cranes

309 The signature of the quay_cane behaviour is a triple of the quay crane identifier, the conceptual quay crane mereology, the static quay crane attributes, the programmable quay crane attributes – and the 'command center'/'quay crane' channel.

310 The quay crane offers, non-deterministically externally, to

311 either, [E], accept a directive of a ‘container transfer from vessel to quay crane’.

a. The quay crane then resumes being a quay crane now holding (a surrogate of) the transferred container.

312 or, [F] accept a directive of a transfer ‘container from quay crane to quay truck’.

a. The quay crane then resumes being a quay crane now holding (a surrogate of) the transferred container.

value
309 quay_cane: qci:QCI × mcci:QC_Mer × QC_Sta → (QCHold×QCPos)
309 → ch_mcc[ mcci,qci ] Unit
309 quay_cane(qci,mcci,qc_staq)(qchold,qcpos) ≡
311 let mkVSQC(ci) = ch_mcc[ mcci,qci ] ? in
311a. quay_cane(qci,mcci,qc_staq)(mkCon(ci),qcpos) end
310 []
312 let mkQCVS(ci) = ch_mcc[ mcci,qci ] ? in
312a. quay_cane(qci,mcci,qc_staq)(mkCon(ci),qcpos) end
6.9.4 Quay Trucks

The signature of the quay truck behaviour is a triple of the quay truck identifier, the conceptual quay truck mereology, the static quay truck attributes, the programmable quay truck attributes – and the ‘command center’/‘quay truck’ channel.

The quay truck offers, non-deterministically externally, to either, [H], accept a directive of a ‘container transfer from quay crane to quay truck’:

a. The quay truck then resumes being a quay truck now holding (a surrogate of) the transferred container.

or, [I], accept a directive of a ‘container transfer from quay truck to quay crane’.

a. The quay truck then resumes being a quay truck now holding (a surrogate of) the transferred container.

6.9.5 Stack Crane

The signature of the stack crane behaviour is a triple of the stack crane stack crane identifier, the conceptual mereology, the static stack crane attributes, the programmable stack crane attributes – and the ‘command center’/‘stack crane’ channel.

The stack crane offers, non-deterministically externally, to either, [K], accept a directive of a ‘container transfer from quay truck to stack crane’.
a. The stack crane then resumes being a stack crane now holding (a surrogate of) the transferred container.

320 or, [L], accept a directive of a ‘container transfer from stack crane to quay truck’.

a. The stack crane then resumes being a stack crane now holding (a surrogate of) the transferred container.

value
317 stack_crate: sci:SCI × mcci:SC_Mer × SC_Sta → (SCHold×SCPos)
317 \rightarrow \text{Unit}
317 stack_crate(sci,mcci,sc_sta)(schold,scpos) \equiv
319a. \text{let mkQTSC(ci) = } ch_mcc\[mcci,sci]\ ? \text{ in}
319a. stack_crate(sci,mcci,sc_sta)(mkCon(ci),scpos) \text{ end}
318 \]
320a. \text{let mkSCQT(ci) = } ch_mcc\[mcci,sci]\ ? \text{ in}
320a. stack_crate(sci,mcci,sc_sta)(mkCon(ci),scpos) \text{ end}

6.9.6 Stacks

The stack behaviour is very much like the unload container container behaviour of the vessel, cf. Items 290 – 294 on Page 69.

321 The signature of the stack behaviour is a triple of the stack, i.e. terminal port bay identifier, the conceptual bay mereology, the static bay attributes, the programmable bay attributes and the ‘command center’/’stack’ channel.

322 The stack offers, [N], to accept directive of a ‘container transfer from stack crane to stack’.

a. The stack behaviour loads the container, identified by ci’, to the bay/row/stack top, identified by \( (bi,ri,si) \).

b. If the unloaded container identifier is different from the expected chaos erupts!

c. The stack state, bay’, is updated accordingly.

d. [N’] “Some time has elapsed since the load directive, modelling” the completion, from the point of view of the vessel, of the unload operation –

e. whereupon the command center is informed of this completion ([’]).

f. The stack then resumes being a stack now holding (a surrogate of) the transferred container.
The signature of the land truck behaviour is a triple of the land truck identifier, the conceptual land truck mereology and the static land truck attributes, and the programmable land truck attributes.

R

a. The land truck calculates the identifier of the next port’s command center
b. and communicates with this center as to its intent to deliver a container identified by ci,
c. whereupon the land truck resumes being that.

T

a. The command center informs the land truck of the bay (‘stack’), brs, at which to deliver the container,
b. whereupon the land truck resumes being that.

Q

a. The command center informs the land truck of the delivery of a container from a stack crane,
b. ..., c. whereupon the land truck resumes being that.
a. The land truck informs the command center of its intent to depart from the terminal port,
b. whereupon the land truck resumes by leaving the terminal port.
6.9.8 Containers

In RSL, as with all formal specification languages one cannot “move” values. So we model containers of vessels and of terminal port stacks as separate behaviours and replace their “values”, C in vessel and terminal port stacks by their unique identifications, CI.

328 The signature of the container behaviour is simple: the container identifier, its mereology, its static values, its position and state, and its input channels.

329 [D,G,J,M,P] The container is here simplified to just, at any moment, accepting a new position from any terminal ports command center;

330 whereupon the container resumes being that with that new position.

\[
\text{container}(ci, mcci\_uis,...)(pos, s_\sigma) \equiv \\
\text{let mkNewPos(p) = \{ ch\_mcc\_con [mcci, ci] ?} \\
\text{mcci: MCCI \& mcci\_uis \in mcci\_uis \} in} \\
\text{container}(ci, mcci\_uis,...)(mkNewPos(p), s_\sigma) \text{ end}
\]

6.10 Initial System

6.10.1 The Distributed System

We remind ourselves that the container line industry includes a set of vessels, a set of land trucks, a set of containers and a set of terminal ports. We rely on the states expounded in Sect. 5.4.1’s Items 50 on Page 26 – 54 on Page 26.

331 The signature of \( \tau_{initial\_system} \) is that of a function from an endurant endurant container line industry to its perdurant behaviour, i.e., Unit.

This behaviour is expressed as

332 the distributed composition of all vessel behaviours in parallel with

333 the distributed composition of all land truck behaviours in parallel with

334 the distributed composition of all container behaviours in parallel with

335 the distributed composition of all terminal port behaviours.
The signature of the \( \tau_{\text{vessel}} \) translation function is simple: a \( \tau \) translator from endurant vessel parts \( v \) to perdurant vessel behaviours, i.e., \( \text{Unit} \).

The transcendental deduction then consists of obtaining the proper arguments for the vessel behaviour –

\[
\begin{align*}
\text{vessel}(v_{ui}, v_{mer}, v_{sta})(v_{pos}, v_{csa}, v_{\Sigma}) & \text{ end}
\end{align*}
\]

Similarly:

\[
\begin{align*}
\tau_{\text{land\_truck}}: & \ \text{LT} \rightarrow \text{Unit} \\
\tau_{\text{land\_truck}}(lt) & \equiv \\
& \text{let } lt_{ui} = \text{uid}_{\text{LT}}(lt), lt_{mer} = \text{mereo}_{\text{LT}}(lt), \\
& \quad lt_{sta} = \text{attr}_{\text{LT\_Sta}}(lt), lt_{pos} = \text{attr}_{\text{LT\_Pos}}(lt), \\
& \quad lt_{hold} = \text{attr}_{\text{LT\_Hold}}(lt), lt_{\Sigma} = \text{attr}_{\text{LT\_\Sigma}}(lt) \text{ in} \\
& \quad \text{vessel}(lt_{ui}, lt_{mer}, lt_{sta})(lt_{pos}, lt_{hold}, lt_{\Sigma}) \text{ end}
\end{align*}
\]

\footnote{As for state: I need to update the container attribute section, Sect. 5.6.11 on Page 41 to reflect a state (for example: the component contents of a container)}
### 6.10.4 Initial Containers

Similarly:

\[
\tau_{\text{container}}: \text{CON} \rightarrow \text{Unit}
\]

\[
\tau_{\text{container}}(\text{con}) \equiv \text{let } c_{\text{ui}} = \text{uid}_\text{CON}(\text{con}), c_{\text{mer}} = \text{mero}_\text{CON}(\text{con}),
\]

\[
c_{\text{sta}} = \text{attr}_\text{CSta}(\text{con}), c_{\text{pos}} = \text{attr}_\text{CPos}(\text{con}),
\]

\[
c_{\sigma} = \text{attr}_\text{CON}@\Sigma(\text{lt}) \text{ in}
\]

\[
\text{container}(c_{\text{ui}}, c_{\text{mer}}, c_{\text{sta}})(c_{\text{pos}}, c_{\sigma}) \text{ end}
\]

### 6.10.5 Initial Terminal Ports

Terminal ports consists of a set of quay cranes, a set of quay trucks a set of stack cranes, and a set of stacks. They translate accordingly:

\[
\tau_{\text{terminal port}}: \text{TP} \rightarrow \text{Unit}
\]

\[
\tau_{\text{terminal port}}(\text{tp}) \equiv \text{let } qcs = \text{obs}_\text{QCs}(\text{obs}_\text{QCS}(\text{tp})),
\]

\[
qts = \text{obs}_\text{QTs}(\text{obs}_\text{QTS}(\text{tp})),
\]

\[
scs = \text{obs}_\text{SCs}(\text{obs}_\text{SCS}(\text{tp})),
\]

\[
stks = \text{obs}_\text{STKs}(\text{obs}_\text{STKS}(\text{tp})) \text{ in}
\]

\[
\| \{ \tau_{\text{quay crane}}(\text{qc}) \mid \text{qc:QC} \cdot \text{qc} \in qcs \} \| \]

\[
\| \{ \tau_{\text{quay truck}}(\text{qt}) \mid \text{qt:QT} \cdot \text{qt} \in qts \} \| \]

\[
\| \{ \tau_{\text{stack crane}}(\text{sc}) \mid \text{sc:SC} \cdot \text{sc} \in scs \} \| \]

\[
\| \{ \tau_{\text{stack}}(\text{stk}) \mid \text{stk:STK} \cdot \text{stk} \in stks \} \text{ end}
\]

### 6.10.6 Initial Quay Cranes

\[
\tau_{\text{quay crane}}: \text{QC} \rightarrow \text{Unit}
\]

\[
\tau_{\text{quay crane}}(\text{qc}) \equiv \text{let } qc_{\text{ui}} = \text{uid}_\text{QC}(\text{qc}), qc_{\text{mer}} = \text{mero}_\text{QC}(\text{qc}),
\]

\[
qc_{\text{sta}} = \text{attr}_\text{QCSta}(\text{qc}), qc_{\text{pos}} = \text{attr}_\text{QCPos}(\text{qc}),
\]

\[
qc_{\sigma} = \text{attr}_\text{QC}@\Sigma(qc) \text{ in}
\]

\[
\text{quay_crate}(qc_{\text{ui}}, qc_{\text{mer}}, qc_{\text{sta}})(qc_{\text{pos}}, qc_{\sigma}) \text{ end}
\]
6.10.7  **Initial Quay Trucks**  
\[ \tau_{\text{quay\_truck}} : \text{QT} \to \text{Unit} \]
\[ \tau_{\text{quay\_truck}}(qt) \equiv \]
\[ \quad \text{let } qt\_ui = \text{uid\_QT}(qt), qt\_mer = \text{mero\_QT}(qt), \]
\[ \quad \quad \text{qt\_sta} = \text{attr\_QT\_Sta}(qt), qt\_pos = \text{attr\_QT\_Pos}(qt), \]
\[ \quad \quad \text{qt}\sigma = \text{attr\_QT}\Sigma(qt) \text{ in} \]
\[ \quad \text{quay\_truck}(qt\_ui, qt\_mer, qt\_sta)(qt\_pos, qt\sigma) \text{ end} \]

6.10.8  **Initial Stack Cranes**  
\[ \tau_{\text{stack\_crane}} : \text{SC} \to \text{Unit} \]
\[ \tau_{\text{stack\_crane}}(sc) \equiv \]
\[ \quad \text{let } sc\_ui = \text{uid\_SC}(sc), sc\_mer = \text{mero\_SC}(sc), \]
\[ \quad \quad \text{sc\_sta} = \text{attr\_SC\_Sta}(sc), sc\_pos = \text{attr\_SC\_Pos}(sc), \]
\[ \quad \quad sc\sigma = \text{attr\_SC}\Sigma(sc) \text{ in} \]
\[ \quad \text{container}(sc\_ui, sc\_mer, sc\_sta)(sc\_pos, sc\sigma) \text{ end} \]

6.10.9  **Initial Stacks**  
\[ \tau_{\text{stack}} : \text{STK} \to \text{Unit} \]
\[ \tau_{\text{stack}}(stk) \equiv \]
\[ \quad \text{let } stk\_ui = \text{uid\_STK}(stk), stk\_mer = \text{mero\_STK}(stk), \]
\[ \quad \quad \text{stk\_sta} = \text{attr\_STK\_Sta}(stk), \]
\[ \quad \quad \text{stk}\sigma = \text{attr\_STK}\Sigma(stk) \text{ in} \]
\[ \quad \text{stack}(stk\_ui, stk\_mer, stk\_sta)(stk\sigma) \text{ end} \]

7  **Conclusion**  

TO BE WRITTEN

7.1  **An Interpretation of the Behavioural Description**  

TO BE WRITTEN

7.2  **What Has Been Done**  

TO BE WRITTEN
7.3 What To Do Next

TO BE WRITTEN

7.4 Acknowledgements

This report was begun when I was first invited to lecture, for three weeks in November 2018, at ECNU\textsuperscript{23}, Shanghai, China. For this and for my actual stay at ECNU, I gratefully acknowledge Profs. He JiFeng, Zhu HuiBiao, Wang XiaoLing and Min Zhang. I chose at the time of the invitation to lead the course students through a major, non-trivial example. Since Shanghai is also one of the major container shipping ports of the world, and since the Danish company \textit{Maersk}, through its subsidiary, \textit{APMTerminals}, operates a major container terminal port, I decided on the subject for this experimental report. I gratefully acknowledge the support the ECNU course received from \textit{APMTerminals}, through its staff, Messrs Henry Bai and Niels Roed.

8 Bibliography

8.1 References


\textsuperscript{23}ECNU: East China Normal University

\textsuperscript{24}http://www2.imm.dtu.dk/˜db/container-paper.pdf


9 Summary of Internal Types

9.1 Unique Identifiers

```
32 pp.24 QCI
33 pp.24 QTI
34 pp.24 SCI
35 pp.24 TBI
36 pp.24 LTI
37 pp.24 MCCI
38 pp.24 CI
39 pp.24 BI
40 pp.24 RI
41 pp.24 SI
```

9.2 Mereologies

```
76 pp.30 V_Mer = MCCI-set
81 pp.31 QC_Mer = MCCI
83 pp.31 QT_Mer = MCCI
87 pp.32 SC_Mer = MCCI
94 pp.34 T_BAY_Mer = MCCI
```

9.3 Attributes

```
type Vessels:
102 pp.36 V_Pos == AtSea | InPort
102a. pp.36 Longitude, Latitude
102a. pp.36 AtSea :: Longitude x Latitude
102b. pp.37 InPort :: MCCI x QPOS
103 pp.37 VΣ

type Quay Cranes:
105 pp.37 QCHold == mkNil('nil')
106 pp.37 QCPos = QSid x QP

type Quay Trucks:
108 pp.38 QTHold == mkNil('nil')
108 pp.38 | mkCon(ci:CI)

type Stack Cranes:
110 pp.38 SCHold == mkNil('nil')
110 pp.38 | mkCon(ci:CI)
111 pp.38 SCPos = BI

type Terminal [i.e., Bay] Stacks
113 pp.38 BOM = BI \* Nat, BOI = BI* 
114 pp.38 ROM = RI \* Nat, ROI = RI* 
115 pp.38 SOM = SI \* Nat, SOL = SI* 
```

```
type Land Trucks:
120 pp.40 LTHold == mkNil('nil')
120 pp.40 | mkCon(ci:CI)
121 pp.40 LTΣ

type Command Centers:
123 pp.40 TopLogDescr
124 pp.40 MCCΣDescr

type Containers:
125 pp.41 BoL
126 pp.41 CPos == onV | onQC | onQT
126 pp.41 | onSC | onStk | onLT | Idle
```

Concrete types of onV, onQC, onQT, onSC, onStk, onLT and Idle

```
126a. pp.41 onV :: VI x BRSP x VPos
126a. pp.41 VPos == AtSea | InTer
126a. pp.41 AtSea :: Geo
126a. pp.41 InTer :: QPSid x QP^*
126b. pp.41 onQC :: MCCI x QCI
126c. pp.41 onQT :: MCCI x QTI
126d. pp.41 onSC :: MCCI x SCI
126e. pp.41 onStk :: MCCI x BRSP
126f. pp.41 onLT :: MCCI x LTI
126g. pp.41 Idle :: {"idle"}
10 RSL: The RAISE Specification Language – A Primer

10.1 Type Expressions

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

10.1.1 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.

Basic Types::

\[
\begin{align*}
\text{type} & \quad \text{Bool} & \text{true}, \text{false} \\
& \quad \text{Int} & ... , -2, -2, 0, 1, 2, ... \\
& \quad \text{Nat} & 0, 1, 2, ... \\
& \quad \text{Real} & ..., -5.43, -1.0, 0.0, 1.23, ..., 2,7182\cdot \cdot \cdot, 3,1415\cdot \cdot \cdot, 4.56, ... \\
& \quad \text{Char} & "a", "b", ..., "0", ... \\
& \quad \text{Text} & "abracadabra" \\
\end{align*}
\]

10.1.2 Composite Types

Composite types have composite values. That is, values which we consider to have proper constituent (sub-)values, i.e., can be meaningfully “taken apart”. There are two ways of expressing composite types: either explicitly, using concrete type expressions, or implicitly, using sorts (i.e., abstract types) and observer functions.

Concrete Composite Types 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 the following are type expressions:

Composite Type Expressions::

\[
\begin{align*}
& \quad \text{A-set} \\
& \quad \text{A-infset} \\
& \quad A \times B \times ... \times C \\
& \quad A^* \\
& \quad A^{\omega} \\
\end{align*}
\]
The following the meaning of the atomic and the composite type expressions:

1. The Boolean type of truth values $\text{false}$ and $\text{true}$.
2. The integer type on integers $\ldots, -2, -1, 0, 1, 2, \ldots$.
3. The natural number type of positive integer values $0, 1, 2, \ldots$
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", "bb", ...
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 $B \times C \times \ldots \times D$, 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., $(A \rightarrow B)$, or $(A^\ast)$-set, or $(A\text{-set})\text{list}$, or $(A|B) \rightarrow (C|D|(E \rightarrow F))$, etc.
16. The postulated disjoint union of types $A, B, \ldots, C$. 
The record type of *mk*<sub>id</sub>-named record values *mk*<sub>id</sub>(av,...,bv), where av, ..., bv, are values of respective types. The distinct identifiers sel<sub>a</sub>, etc., designate selector functions.

The record type of unnamed record values (av,...,bv), where av, ..., bv, are values of respective types. The distinct identifiers sel<sub>a</sub>, etc., designate selector functions.

**Sorts and Observer Functions**

\[ \text{type } \ A, \ B, \ C, \ ..., \ D \]

\[ \text{value } \ \text{obs}_B : A \to B, \ \text{obs}_C : A \to C, \ ..., \ \text{obs}_D : A \to D \]

The above expresses that values of type A are composed from at least three values — and these are of type B, C, ..., and D. A concrete type definition corresponding to the above presupposing material of the next section

\[ \text{type } \ B, \ C, \ ... , D \]

\[ A = B \times C \times ... \times D \]

**10.2 Type Definitions**

**10.2.1 Concrete Types**

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

\[ \text{Type Definition::} \]

\[ \text{type } \ A = \text{Type_expr} \]

Some schematic type definitions are:
**Variety of Type Definitions::**

[19] \[ \text{Type}\_\text{name} = \text{Type}\_\text{expr} */ \text{without } | \text{s or subtypes } */ \\
[20] \text{Type}\_\text{name} = \text{Type}\_\text{expr} \_1 | \text{Type}\_\text{expr} \_2 | \ldots | \text{Type}\_\text{expr} \_n \\
[21] \text{Type}\_\text{name} == \\
\quad \text{mk}\_\text{id}\_1(\text{s}\_\text{a}1:\text{Type}\_\text{name}\_\text{a}1,\ldots,\text{s}\_\text{ai}:\text{Type}\_\text{name}\_\text{ai}) | \\
\quad \ldots | \\
\quad \text{mk}\_\text{id}\_n(\text{s}\_\text{z}1:\text{Type}\_\text{name}\_\text{z1},\ldots,\text{s}\_\text{zk}:\text{Type}\_\text{name}\_\text{zk}) \\
[22] \text{Type}\_\text{name} :: \text{sel}\_\text{a}:\text{Type}\_\text{name}\_\text{a} \ldots \text{sel}\_\text{z}:\text{Type}\_\text{name}\_\text{z} \\
[23] \text{Type}\_\text{name} = \{ | \text{v:Type}\_\text{name}' \cdot \mathcal{P}(\text{v}) | \}

where a form of [20]–[21] is provided by combining the types:

**Record Types::**

\[ \text{Type}\_\text{name} = A | B | \ldots | Z \]
\[ A == \text{mk}\_\text{id}\_1(\text{s}\_\text{a}1:A\_1,\ldots,\text{s}\_\text{ai}:A\_i) \]
\[ B == \text{mk}\_\text{id}\_2(\text{s}\_\text{b}1:B\_1,\ldots,\text{s}\_\text{bj}:B\_j) \]
\[ \ldots \]
\[ Z == \text{mk}\_\text{id}\_n(\text{s}\_\text{z}1:Z\_1,\ldots,\text{s}\_\text{zk}:Z\_k) \]

Types A, B, ..., Z are disjoint, i.e., shares no values, provided all \text{mk}\_\text{id}\_k are distinct and due to the use of the disjoint record type constructor ==.

**axiom**

\[ \forall \text{a}1:A\_1, \text{a}2:A\_2, \ldots, \text{ai}:A\_i \cdot \\
\quad \text{s}\_\text{a}1(\text{mk}\_\text{id}\_1(\text{a}1,\text{a}2,\ldots,\text{ai}))=\text{a}1 \land \text{s}\_\text{a}2(\text{mk}\_\text{id}\_1(\text{a}1,\text{a}2,\ldots,\text{ai}))=\text{a}2 \land \\
\quad \ldots \land \text{s}\_\text{ai}(\text{mk}\_\text{id}\_1(\text{a}1,\text{a}2,\ldots,\text{ai}))=\text{ai} \land \\
\quad \forall \text{a}:A \cdot \text{let} \:\text{mk}\_\text{id}\_1(\text{a}1',\text{a}2',\ldots,\text{ai}') = \text{a} \text{ in} \\
\quad \text{a}1' = \text{s}\_\text{a}1(\text{a}) \land \text{a}2' = \text{s}\_\text{a}2(\text{a}) \land \ldots \land \text{ai}' = \text{s}\_\text{ai}(\text{a}) \text{ end} \]

**10.2.2 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::**

\[ \text{type} \\
\quad A = \{ | \text{b:B } \cdot \mathcal{P}(\text{b}) | \} \]
10.2.3  **Sorts — Abstract Types**

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

\[
\text{Sorts::}
\]

\[
\text{type} A, B, ..., C
\]

10.3  **The RSL Predicate Calculus**

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

\[
\text{Propositional Expressions::}
\]

\[
\text{false, true, } a, b, ..., c \sim a, a \land b, a \lor b, a \Rightarrow b, a =b, a \neq b
\]

are propositional expressions having Boolean values. \(\sim, \land, \lor, \Rightarrow, =\) and \(\neq\) are Boolean connectives (i.e., operators). They can be read as: \textit{not}, \textit{and}, \textit{or}, \textit{if then} (or \textit{implies}), \textit{equal} and \textit{not equal}.

10.3.1  **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 \(i, j, ...\) designate number values, then:

\[
\text{Simple Predicate Expressions::}
\]

\[
\text{false, true, } a, b, ..., c
\]

\[
\sim a, a \land b, a \lor b, a \Rightarrow b, a =b, a \neq b
\]

\[
x =y, x \neq y,
\]

\[
i < j, i \leq j, i \geq j, i \neq j, i \geq j, i > j
\]

are simple predicate expressions.

10.3.2  **Quantified Expressions**

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

\[ \forall x:X \cdot P(x) \]
\[ \exists y:Y \cdot Q(y) \]
\[ \exists ! z:Z \cdot R(z) \]

are quantified expressions — also being predicate expressions.

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

10.4 RSL Values and Operations

10.4.1 Arithmetic

Arithmetic:

\[
\begin{align*}
\text{type} & \quad \text{Nat, Int, Real} \\
\text{value} & \quad +, -, *, : \quad \text{Nat} \times \text{Nat} \to \text{Nat} | \text{Int} \times \text{Int} \to \text{Int} | \text{Real} \times \text{Real} \to \text{Real} \\
\text{}/: & \quad \text{Nat} \times \text{Nat} \to \text{Nat} | \text{Int} \times \text{Int} \to \text{Int} | \text{Real} \times \text{Real} \to \text{Real} \\
\langle , \leq , = , \neq , \geq , \rangle & \quad (\text{Nat} | \text{Int} | \text{Real}) \to (\text{Nat} | \text{Int} | \text{Real})
\end{align*}
\]

10.4.2 Set Expressions

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

Set Enumerations:

\[
\begin{align*}
\{\}, \{a\}, \{e_1, e_2, \ldots, e_n\}, \ldots \in A\text{-set} \\
\{\}, \{a\}, \{e_1, e_2, \ldots, e_n\}, \ldots, \{e_1, e_2, \ldots\} \in A\text{-infset}
\end{align*}
\]

Set Comprehension  The expression, last line below, to the right of the \( \equiv \), 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) }

10.4.3 Cartesian Expressions

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

Cartesian Enumerations::

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

10.4.4 List Expressions

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

List Comprehension The last line below expresses list comprehension.
**List Comprehension::**

*type*

\[ A, B, P = A \rightarrow \text{Bool}, \ Q = A \sim B \]*

*value*

comprehend: \( A^\omega \times P \times Q \sim B^\omega \)

\[ \text{comprehend}(l, P, Q) \equiv \langle Q(l(i)) \mid i \in \langle 1..\text{len} \ l \rangle \cdot P(l(i)) \rangle \]

**10.4.5 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*

\[ T1, T2 \]

\[ M = T1 \rightarrow T2 \]

*value*

\[ u, u1, u2, \ldots, un : T1, \ v, v1, v2, \ldots, vn : T2 \]

\[ \[, \[ u \mapsto v \] \], \ldots, \[ u1 \mapsto v1, u2 \mapsto v2, \ldots, un \mapsto vn \] \text{ all } \in M \]

**Map Comprehension** The last line below expresses map comprehension:

**Map Comprehension::**

*type*

\[ U, V, X, Y \]

\[ M = U \rightarrow V \]

\[ F = U \sim X \]

\[ G = V \sim Y \]

\[ P = U \rightarrow \text{Bool} \]

*value*

comprehend: \( M \times F \times G \times P \rightarrow (X \rightarrow Y) \)

\[ \text{comprehend}(m, F, G, P) \equiv \[ F(u) \mapsto G(m(u)) \mid u : U \in \text{dom} \ m \land P(u) \] \]

**10.4.6 Set Operations**

**Set Operator Signatures**
Set Operations::

value
19 \in: A \times A\text{-}infset \to \text{Bool}
20 \notin: A \times A\text{-}infset \to \text{Bool}
21 \cup: A\text{-}infset \times A\text{-}infset \to A\text{-}infset
22 \cup: (A\text{-}infset)\text{-}infset \to A\text{-}infset
23 \cap: A\text{-}infset \times A\text{-}infset \to A\text{-}infset
24 \cap: (A\text{-}infset)\text{-}infset \to A\text{-}infset
25 \setminus: A\text{-}infset \times A\text{-}infset \to A\text{-}infset
26 \subset: A\text{-}infset \times A\text{-}infset \to \text{Bool}
27 \subseteq: A\text{-}infset \times A\text{-}infset \to \text{Bool}
28 =: A\text{-}infset \times A\text{-}infset \to \text{Bool}
29 \neq: A\text{-}infset \times A\text{-}infset \to \text{Bool}
30 \text{card}: A\text{-}infset \sim \text{Nat}

Set Examples

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,bb\}, \{a,d\}\} = \{a,b,d\}
\{a,b,c\} \cap \{c,d,e\} = \{c\}
\cap\{\{a\}\}, \{a,bb\}, \{a,d\}\} = \{a\}
\{a,b,c\} \setminus \{c,d\} = \{a,bb\}
\{a,bb\} \subset \{a,b,c\}
\{a,b,c\} \subseteq \{a,b,c\}
\{a,b,c\} = \{a,b,c\}
\{a,b,c\} \neq \{a,bb\}
\text{card}\ \{\} = 0, \text{card}\ \{a,b,c\} = 3

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.

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 $\subset$: 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 $=:\text{The equal operator expresses that the two operand sets are identical.}$

29 $\neq:\text{The nonequal operator expresses that the two operand sets are not identical.}$

30 $\text{card: The cardinality operator gives the number of elements in a finite set.}$

**Set Operator Definitions**  
The operations can be defined as follows ($\equiv$ is the definition symbol):  

**Set Operation Definitions::**

**value**

\[
\begin{align*}
\text{s'} \cup \text{s''} & \equiv \{ a \mid a: A \land a \in \text{s'} \lor a \in \text{s''} \} \\
\text{s'} \cap \text{s''} & \equiv \{ a \mid a: A \land a \in \text{s'} \land a \in \text{s''} \} \\
\text{s'} \setminus \text{s''} & \equiv \{ a \mid a: A \land a \in \text{s'} \land a \not\in \text{s''} \} \\
\text{s'} \subseteq \text{s''} & \equiv \forall a: A \cdot a \in \text{s'} \Rightarrow a \in \text{s''} \\
\text{s'} \subset \text{s''} & \equiv \text{s'} \subseteq \text{s''} \land \exists a: A \cdot a \in \text{s''} \land a \not\in \text{s'} \\
\text{s'} = \text{s''} & \equiv \forall a: A \cdot a \in \text{s'} \equiv a \in \text{s''} \equiv \text{s} \subseteq \text{s'} \land \text{s'} \subseteq \text{s''} \\
\text{s'} \neq \text{s''} & \equiv \text{s'} \cap \text{s''} \neq \{\} \\
\text{card} \; s & \equiv \\
\text{if} \; s = \{\} \text{ then } 0 \text{ else} \\
\text{let} \; a: A \cdot a \in s \text{ in } 1 + \text{card} \; (s \setminus \{a\}) \end{align*}
\]
93

pre s /* is a finite set */
\text{card} s \equiv \text{chaos} /* tests for infinity of s */

10.4.7 Cartesian Operations

\textbf{Cartesian Operations::}

\textbf{type}

\begin{align*}
A, B, C \\
g0: G0 &= A \times B \times C \\
g1: G1 &= ( A \times B \times C ) \\
g2: G2 &= ( A \times B ) \times C \\
g3: G3 &= A \times ( B \times C )
\end{align*}

\textbf{value}

\begin{align*}
va:A, vb:B, vc:C, vd:D \\
(va,vb,vc):G0, \\
((va,vb),vc):G2 \\
(va3,(vb3,vc3)):G3
\end{align*}

\textbf{decomposition expressions}

\begin{align*}
\text{let} \ (a1,b1,c1) = g0, \\
\text{let} \ (a1',b1',c1') = g1 \text{ in } .. \text{ end} \\
\text{let} \ ((a2,b2),c2) = g2 \text{ in } .. \text{ end} \\
\text{let} \ (a3,(b3,c3)) = g3 \text{ in } .. \text{ end}
\end{align*}

10.4.8 List Operations

\textbf{List Operator Signatures}

\textbf{List Operations::}

\textbf{value}

\begin{align*}
\text{hd}: A^\omega \rightarrow A \\
\text{tl}: A^\omega \rightarrow A^\omega \\
\text{len}: A^\omega \rightarrow \text{Nat} \\
\text{inds}: A^\omega \rightarrow \text{Nat-infset} \\
\text{elems}: A^\omega \rightarrow A^\text{infset} \\
(.): A^\omega \times \text{Nat} \rightarrow A \\
\sim: A^* A^* A^* A^* A^* \rightarrow \text{bool}
\end{align*}

\textbf{List Operation Examples}

\textbf{List Examples::}

\begin{align*}
\text{examples}
\end{align*}

\begin{align*}
\text{hd}(a1,a2,\ldots,am) &= a1 \\
\text{tl}(a1,a2,\ldots,am) &= \langle a2,\ldots,am \rangle \\
\text{len}(a1,a2,\ldots,am) &= m
\end{align*}
\( \text{inds}(a_1,a_2,\ldots,a_m) = \{1,2,\ldots,m\} \)
\( \text{elems}(a_1,a_2,\ldots,a_m) = \{a_1,a_2,\ldots,a_m\} \)
\( (a_1,a_2,\ldots,a_m)(i) = a_i \)
\( (a,b,c) \cdot (a,b,d) = (a,b,c,a,b,d) \)
\( (a,b,c) = (a,b,c) \)
\( (a,b,c) \neq (a,b,d) \)

**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 \(\ell\) having a number of elements larger than or equal to \(i\), gives the \(i\)th element of the list.
- **\(\cdot\)**: 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.
- **\(\neq\)**: The nonequal operator expresses that the two operand lists are *not* identical.

The operations can also be defined as follows:

**List Operator Definitions**

\[
\text{value} \\
\text{is\_finite\_list}: A^{\omega} \rightarrow \text{Bool}
\]
\[
\text{len } q \equiv \\
\hspace{1em} \text{case } \text{is\_finite\_list}(q) \text{ of} \\
\hspace{2em} \text{true} \rightarrow \text{if } q = \langle \rangle \text{ then } 0 \text{ else } 1 + \text{len } \text{tl } q \text{ end}, \\
\hspace{2em} \text{false} \rightarrow \text{chaos end}
\]
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
    end
  else q(i-1) end

fq ∪ iq ≡
⟨ if 1 ≤ i ≤ len fq then fq(i) else iq(i) 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″)

10.4.9 Map Operations

Map Operator Signatures and Map Operation Examples

value
m(a): M → A → B, m(a) = b

dom: M → A-infset [domain of map]
dom [a1→b1,a2→b2,...,an→bn] = {a1,a2,...,an}

rng: M → B-infset [range of map]
rng [a1→b1,a2→b2,...,an→bn] = {b1,b2,...,bn}

↑: M × M → M [override extension]
[a→b,a'→bb',a''→bb''] ↳ [a'→bb'',a''→bb'] = [a→b,a'→bb'',a''→bb']
\[ \cup: M \times M \to M \ [\text{merge } \cup] \]
\[ [a\mapsto b, a'\mapsto bb', a''\mapsto bb''] \cup [a''\mapsto bb''] = [a\mapsto b, a'\mapsto bb', a''\mapsto bb'', a''\mapsto bb''] \]

\[ \setminus: M \times A\text{-inset} \to M \ [\text{restriction by}] \]
\[ [a\mapsto b, a'\mapsto bb', a''\mapsto bb''] \setminus \{a\} = [a'\mapsto bb', a''\mapsto bb''] \]

\[ \div: M \times A\text{-inset} \to M \ [\text{restriction to}] \]
\[ [a\mapsto b, a'\mapsto bb', a''\mapsto bb''] \div \{a', a''\} = [a'\mapsto bb', a''\mapsto bb''] \]

\[ =, \neq: M \times M \to \text{Bool} \]

\[ \circ: (A \to m B) \times (B \to m C) \to (A \to m C) \ [\text{composition}] \]
\[ [a\mapsto b, a'\mapsto bb'] \circ [bb'\mapsto c, bb''\mapsto c'] = [a\mapsto c, a'\mapsto c'] \]

---

**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.
- **\ː** 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.
- **\ː** 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 \(c\).
Map Operation Redefinitions  The map operations can also be defined as follows:

Map Operation Redefinitions:

\[
\text{value} \\
\text{rng } m \equiv \{ m(a) \mid a:A \cdot a \in \text{dom } m \} \\
\]

\[
m_1 \overset{1}{\to} m_2 \equiv \\
[ a \mapsto b \mid a:A,b:B \cdot \\
a \in \text{dom } m_1 \setminus \text{dom } m_2 \land bb=m_1(a) \lor a \in \text{dom } m_2 \land bb=m_2(a) ] \\
\]

\[
m_1 \cup m_2 \equiv [ a \mapsto b \mid a:A,b:B \cdot \\
a \in \text{dom } m_1 \land bb=m_1(a) \lor a \in \text{dom } m_2 \land bb=m_2(a) ] \\
\]

\[
m \setminus s \equiv [ a \mapsto m(a) \mid a:A \cdot a \in \text{dom } m \setminus s ] \\
m \div s \equiv [ a \mapsto m(a) \mid a:A \cdot a \in \text{dom } m \cap s ] \\
\]

\[
m_1 = m_2 \equiv \\
\text{dom } m_1 = \text{dom } m_2 \land \forall a:A \cdot a \in \text{dom } m_1 \Rightarrow m_1(a) = m_2(a) \\
m_1 \neq m_2 \equiv \sim(m_1 = m_2) \\
\]

\[
m^n \equiv \\
[ a \mapsto c \mid a:A,c:C \cdot a \in \text{dom } m \land c = n(m(a)) ] \\
\text{pre } \text{rng } m \subseteq \text{dom } n \\
\]

10.5  \(\lambda\)-Calculus + Functions

10.5.1  The \(\lambda\)-Calculus Syntax

\(\lambda\)-Calculus Syntax::

type /* A BNF Syntax: */  
\langle L \rangle ::= \langle V \rangle \mid \langle F \rangle \mid \langle A \rangle \mid ( \langle A \rangle ) \\
\langle V \rangle ::= /* variables, i.e. identifiers */ \\
\langle F \rangle ::= \lambda \langle V \rangle \cdot \langle L \rangle \\
\langle A \rangle ::= ( \langle L \rangle(\langle L \rangle) ) \\
value /* Examples */  
\langle L \rangle: e, f, a, ... \\
\langle V \rangle: x, ... \\
\langle F \rangle: \lambda x \cdot e, ... \\
\langle A \rangle: f a, (f a), f(a), (f)(a), ...
10.5.2 Free and Bound Variables

Free and Bound Variables:: Let \( x, y \) be variable names and \( e, f \) be \( \lambda \)-expressions.

- \( (V) \): Variable \( x \) is free in \( x \).
- \( (F) \): \( x \) is free in \( \lambda y \cdot e \) if \( x \not= y \) and \( x \) is free in \( e \).
- \( (A) \): \( x \) is free in \( f(e) \) if it is free in either \( f \) or \( e \) (i.e., also in both).

10.5.3 Substitution

In RSL, the following rules for substitution apply:

Substitution::

- \( \text{subst}([N/x]x) \equiv N \);
- \( \text{subst}([N/x]a) \equiv a \),
  for all variables \( a \not= x \);
- \( \text{subst}([N/x](P \ Q)) \equiv (\text{subst}([N/x]P) \ \text{subst}([N/x]Q)) \);
- \( \text{subst}([N/x](\lambda x \cdot P)) \equiv \lambda y \cdot \text{subst}([N/x]P), \)
  if \( x \not= y \) and \( y \) is not free in \( N \) or \( x \) is not free in \( P \);
- \( \text{subst}([N/x](\lambda y \cdot P)) \equiv \lambda z \cdot \text{subst}([N/z]\text{subst}([z/y]P)), \)
  if \( y \not= x \) and \( y \) is free in \( N \) and \( x \) is free in \( P \)
  (where \( z \) is not free in \( (N \ P) \)).

10.5.4 \( \alpha \)-Renaming and \( \beta \)-Reduction

\( \alpha \) and \( \beta \) Conversions::

- \( \alpha \)-renaming: \( \lambda x \cdot M \)

  If \( x, y \) are distinct variables then replacing \( x \) by \( y \) in \( \lambda x \cdot M \) results in \( \lambda y \cdot \text{subst}([y/x]M) \).

  We can rename the formal parameter of a \( \lambda \)-function expression provided that no free variables of its body \( M \) thereby become bound.

- \( \beta \)-reduction: \( (\lambda x \cdot M)(N) \)

  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. \( (\lambda x \cdot M)(N) \equiv \text{subst}([N/x]M) \).
10.5.5 Function Signatures

For sorts we may want to postulate some functions:

Sorts and Function Signatures::

\begin{itemize}
\item \textbf{type}
  \begin{itemize}
  \item A, B, C
  \end{itemize}
\item \textbf{value}
  \begin{itemize}
  \item \texttt{obs}_B: A \rightarrow B,
  \item \texttt{obs}_C: A \rightarrow C,
  \item \texttt{gen}_A: BB \times C \rightarrow A
  \end{itemize}
\end{itemize}

10.5.6 Function Definitions

Functions can be defined explicitly:

Explicit Function Definitions::

\begin{itemize}
\item \textbf{value}
  \begin{itemize}
  \item \texttt{f}: \text{Arguments} \rightarrow \text{Result}
  \item \texttt{f}(\text{args}) \equiv \text{DValueExpr}
  \end{itemize}
\item \textbf{value}
  \begin{itemize}
  \item \texttt{g}: \text{Arguments} \xrightarrow{\sim} \text{Result}
  \item \texttt{g}(\text{args}) \equiv \text{ValueAndStateChangeClause}
  \item \texttt{pre} \texttt{P}(\text{args})
  \end{itemize}
\end{itemize}

Or functions can be defined implicitly:

Implicit Function Definitions::

\begin{itemize}
\item \textbf{value}
  \begin{itemize}
  \item \texttt{f}: \text{Arguments} \rightarrow \text{Result}
  \item \texttt{f}(\text{args}) \texttt{as} \texttt{result}
  \item \texttt{post} \texttt{P1}(\text{args,result})
  \end{itemize}
\item \textbf{value}
  \begin{itemize}
  \item \texttt{g}: \text{Arguments} \xrightarrow{\sim} \text{Result}
  \item \texttt{g}(\text{args}) \texttt{as} \texttt{result}
  \item \texttt{pre} \texttt{P2}(\text{args})
  \item \texttt{post} \texttt{P3}(\text{args,result})
  \end{itemize}
\end{itemize}

The symbol $\xrightarrow{\sim}$ 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.
10.6 Other Applicative Expressions

10.6.1 Simple let Expressions

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

Let Expressions::

\[
\text{let } a = \mathcal{E}_d \text{ in } \mathcal{E}_b(a) \text{ end}
\]

is an “expanded” form of:

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

10.6.2 Recursive let Expressions

Recursive let expressions are written as:

Recursive let Expressions::

\[
\text{let } f = \lambda a:A \cdot \mathcal{E}(f) \text{ in } \mathcal{B}(f,a) \text{ end}
\]

is “the same” as:

\[
\text{let } f = \mathcal{YF} \text{ in } \mathcal{B}(f,a) \text{ end}
\]

where:

\[
\mathcal{F} \equiv \lambda g: \lambda a: \mathcal{E}(g) \text{ and } \mathcal{YF} = \mathcal{F}(\mathcal{YF})
\]

10.6.3 Predicative let Expressions

Predicative let expressions:

Predicative let Expressions::

\[
\text{let } a:A \cdot \mathcal{P}(a) \text{ in } \mathcal{B}(a) \text{ end}
\]

express the selection of a value \( a \) of type \( A \) which satisfies a predicate \( \mathcal{P}(a) \) for evaluation in the body \( \mathcal{B}(a) \).
10.6.4 Pattern and “Wild Card” let Expressions

Patterns and wild cards can be used:

Patterns::

let \{a\} \cup s = set in ... end
let \{a\_\} \cup s = set in ... end

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

let \(\langle a\rangle^\ell\) = list in ... end
let \(\langle a\_,bb\rangle^\ell\) = list in ... end

let \[a\mapsto bb\] \cup m = map in ... end
let \[a\mapsto b\_] \cup m = map in ... end

10.6.5 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 \equiv /* same as: */
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_3 then c_expr_3
...
ejsif b_expr_n then c_expr_n end

case expr of
choice_pattern_1 \rightarrow expr_1,
choice_pattern_2 \rightarrow expr_2,
...
choice_pattern_n_or_wild_card \rightarrow expr_n
end
10.6.6 Operator/Operand Expressions

Operator/Operand Expressions::

\[ \langle \text{Expr} \rangle ::= \]
\[ \langle \text{Prefix\_Op} \rangle \langle \text{Expr} \rangle \]
\[ \mid \langle \text{Expr} \rangle \langle \text{Infix\_Op} \rangle \langle \text{Expr} \rangle \]
\[ \mid \langle \text{Expr} \rangle \langle \text{Suffix\_Op} \rangle \]
\[ \mid \ldots \]

\[ \langle \text{Prefix\_Op} \rangle ::= - \mid \sim \mid \cup \mid \cap \mid \text{card} \mid \text{len} \mid \text{inds} \mid \text{elems} \mid \text{hd} \mid \text{tl} \mid \text{dom} \mid \text{rng} \]

\[ \langle \text{Infix\_Op} \rangle ::= = \mid \neq \mid \equiv \mid + \mid - \mid \ast \mid \uparrow \mid / \mid < \mid \leq \mid \geq \mid > \mid \land \mid \lor \mid \Rightarrow \]
\[ \mid \in \mid \notin \mid \cup \mid \cap \mid \setminus \mid \subset \mid \subseteq \mid \supset \mid \supseteq \mid \approx \mid \hat{\ast} \mid \hat{\dagger} \mid \hat{\circ} \]

\[ \langle \text{Suffix\_Op} \rangle ::= ! \]

10.7 Imperative Constructs

10.7.1 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::

Unit

value

stmt: Unit \rightarrow Unit

stmt()

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

© Dines Bjørner. 2018, Fredsvej 11, DK–2840 Holte, Denmark – November 22, 2018: 11:05 am
10.7.2 Variables and Assignment

Variables and Assignment::

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

10.7.3 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

10.7.4 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

10.7.5 Iterative Conditionals

Iterative Conditionals::

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

10.7.6 Iterative Sequencing

Iterative Sequencing::

8. for e in list_expr • P(b) do S(b) end
10.8 Process Constructs

10.8.1 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).

10.8.2 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:

\[ \begin{align*}
  P \parallel Q & \quad \text{Parallel composition} \\
  P \lceil \parallel Q & \quad \text{Nondeterministic external choice (either/or)} \\
  P \lceil \cdot \parallel Q & \quad \text{Nondeterministic internal choice (either/or)} \\
  P - \parallel Q & \quad \text{Interlock parallel composition}
\end{align*} \]

express the parallel (\(\parallel\)) of two processes, or the nondeterministic choice between two processes: either external (\(\lceil \parallel \rceil\)) or internal (\(\lceil \cdot \parallel \rceil\)). The interlock (\(-\parallel\)) composition expresses that the two processes are forced to communicate only with one another, until one of them terminates.

10.8.3 Input/Output Events

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

**Input/Output Events::**

\[ \begin{align*}
  c \ ? , k[i] \ ? & \quad \text{Input} \\
  c \ ! e , k[i] \ ! e & \quad \text{Output}
\end{align*} \]

expresses the willingness of a process to engage in an event that “reads” an input, respectively “writes” an output.
10.8.4 **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::**

\[
\text{value} \\
\begin{align*}
P &: \text{Unit} \to \text{in } c \text{ out } k[i] \\
Q &: \text{i:Kldx} \to \text{out } c \text{ in } k[i] \text{ Unit}
\end{align*}
\]

\[
P() \equiv ... \ c \ ? \ ... \ k[i] ! e ... \\
Q(i) \equiv ... k[i] ? ... \ c ! e ...
\]

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

### 10.9 **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::**

\[
\text{type} \\
\text{variable} \\
\text{channel} \\
\text{value} \\
\text{axiom}
\]

In practice a full specification repeats the above listings many times, once for each “module” (i.e., aspect, facet, view) of specification. Each of these modules may be “wrapped” into scheme, class or object definitions.\(^{26}\)

\(^{26}\)For schemes, classes and objects we refer to [14, Chap. 10]
10.10 **RSL Index**
Arithmetics

...-2,-1,0,1,2,...  \[ a_i-a_j \] , 94
\[ a_i+a_j \] , 94
\[ a_i/a_j \] , 93
\[ a_i\geq a_j \] , 93
\[ a_i>a_j \] , 93
\[ a_i\leq a_j \] , 93
\[ a_i<a_j \] , 93
\[ a_i\neq a_j \] , 93
\[ a_i-a_j \] , 94

Cartesians

\[ (e_1,e_2,...,e_n) \] , 95

Chaos

chaos, 99–101

Clauses

... elsif ... , 107
\[ \text{case } b_e \text{ of } p a_1 \rightarrow c_1, ... \ p a_n \rightarrow c_n \text{ end} \] , 107
\[ \text{if } b_e \text{ then } c_e \text{ else } c_a \text{ end} \] , 107

Combinators

let \( a:A \bullet P(a) \) in \( c \) end , 106
let \( p a = e \) in \( c \) end , 106

Functions

post \( P(\text{args},\text{result}) \) , 105
pre \( P(\text{args}) \) , 105
\( f(\text{args}) \) as \( \text{result} \) , 105
\( f(a) \) , 103
\( f(\text{args}) \equiv \text{expr} \) , 105

Imperative

\[ \text{case } b_e \text{ of } p a_1 \rightarrow c_1, ... \ p a_n \rightarrow c_n \text{ end} \] , 109
\[ \text{do } \text{stmt} \text{ until } b_e \text{ end} \] , 109
\[ \text{for } e \text{ in list}\_\text{expr} \bullet P(b) \text{ do } \text{stm(e)} \text{ end} \] , 109
\[ \text{if } b_e \text{ then } c_e \text{ else } c_a \text{ end} \] , 109
\[ \text{skip} \] , 109
variable \( v:Type := \text{expression} \) , 109
while \( b_e \text{ do } \text{stmt} \) end , 109
\( f() \) , 108
\( \text{stm_1;stm_2;...;stm_n} \) , 109
\( v := \text{expression} \) , 109

Lists

\[ <Q(1(i))|i \in <1..len|> \bullet P(a) > \] , 96
\( hA,B \) , 95
\( \ell(i) \) , 99
\( \langle e_1, e_2, ..., e_n, B \rangle \) , 96
\( \text{elems } \ell \) , 99
hd \( \ell \) , 99
\( \text{inds } \ell \) , 99
\( \text{len } \ell \) , 99
\( \text{tl } \ell \) , 99

Logics

\( b_i \lor b_j \) , 93
\( \forall \ a:A \bullet P(a) \) , 94
\( \exists \ a:A \bullet P(a) \) , 94
\( \exists a:A \bullet P(a) \) , 94
\( \sim b \) , 93
false, 90, 93
true, 90, 93
\( a_i=a_j \) , 94
\( a_i\geq a_j \) , 94
\( a_i>a_j \) , 94
\( a_i\leq a_j \) , 94
\( a_i<a_j \) , 94
\( a_i\neq a_j \) , 94
\( b_i \Rightarrow b_j \) , 93
\( b_i \land b_j \) , 93

Maps

\[ [F(e)\mapsto G(m(e))|e: E \in \text{dom } m \land P(e)] \] , 96
\[ [ ] \] , 96
\[ [u_1 \mapsto v_1, u_2 \mapsto v_2, ..., u_n \mapsto v_n] \] , 96
\( m_i \setminus m_j \) , 102
\( m_i \circ m_j \) , 102
\( m_i / m_j \) , 102
\( \text{dom } m \) , 101
\( \text{rng } m \) , 101
\( m_i = m_j \) , 102
\( m_i \cup m_j \) , 102
\( m_i \uparrow m_j \) , 101
\( m_i \neq m_j \) , 102
\( m(e) \) , 101

Processes
channel $c : T$, 110
channel $\{k[i]: T \star i : \text{KIdx}\}$, 110
$c ! e$, 110
$c ?$, 110
$k[i] ! e$, 110
$k[i] ?$, 110
$P \parallel Q$, 110
$P \mid Q$, 110
$P : \text{Unit} \rightarrow \text{in} \ c \ \text{out} \ k[i] \ \text{Unit}$, 111
$P \parallel Q$, 110
$Q : i : \text{KIdx} \rightarrow \text{out} \ c \ \text{in} \ k[i] \ \text{Unit}$, 111

Sets
$\{Q(a)|a : A \bullet a \in \mathcal{S} \land P(a)\}$, 95
$\{\}$, 94
$\{e_1, e_2, ..., e_n\}$, 94
$\cap\{s_1, s_2, ..., s_n\}$, 97
$\cup\{s_1, s_2, ..., s_n\}$, 97
$\text{card} \ s$, 97
e$\in \mathcal{S}$, 97
e$\notin \mathcal{S}$, 97
$s_i = s_j$, 97
$s_i \cap s_j$, 97
$s_i \cup s_j$, 97
$s_i \subseteq s_j$, 97
$s_i \neq s_j$, 97
$s_i \setminus s_j$, 97

Types
$\text{Bool}$, 89
$\text{Char}$, 89
$\text{Int}$, 89
$\text{Nat}$, 89
$\text{Real}$, 89
$\text{Text}$, 89
$\text{Unit}$, 108, 111
$(T_1 \times T_2 \times \cdots \times T_n)$, 90
$T^*$, 89
$T^\omega$, 89
$T_1 \times T_2 \times \cdots \times T_n$, 89
$\text{mk id}(s_1 : T_1, s_2 : T_2, ..., s_n : T_n)$, 90
$s_1 : T_1 \ s_2 : T_2 \ ... \ s_n : T_n$, 90
$T = \text{Type} \ast \text{Expr}$, 91
$T = | \ T_1 \ | \ T_2 \ | \ ... \ | \ T_1 \ | \ T_n$, 90
$T = \{v : T \ast P(v)\}$, 92
$T = \{T_1 \ | \ T_2 \ | \ ... \ | \ T_1 \ | \ T_n$, 90
$T - \text{infset}$, 89
$T - \text{set}$, 89
$T_1 \tilde{\rightarrow} T_j$, 90
$T_i \rightarrow T_j$, 90