An Informatics View of the World

An HSE Moscow Seminar, Wednesday April 22, 2015

Dines Bjørner

March 25, 2015: 11:33 am

Summary

- This talk is for beginners in the serious study of computer science[1].
- Behind the presentation of this talk
  - lies the attitude that **software**,
  - including **programmes**,
  - denote **mathematical objects**.

[1]— or for that matter: software engineering, informatics, or the like!

- Through three examples
  - I provide a glimpse into a universe of domains
  - for which software is developed every day
  - but for which, in most instances,
  - there are no formal, i.e., mathematical understanding.

- Three examples are:
  - ⟐ **road transport and platooning** systems,
  - ⟐ **oil/gas pipeline** systems, and
  - ⟐ **stock exchange** sell offer/buy bid matching.

- The objective of this talk is to show you
  - what it means to develop software using mathematics,
    - albeit it new kind of mathematics,
    - not differential equations, nor integrals, nor statistics, etc.,
    - but **mathematical logic** and **modern algebra**,
  - so that software can be shown
    - **correct**, i.e., no bugs, no **blue screen**, and
    - to **met customers' expectations**!

Opening

- In this talk I show the listener
  a number of examples related to software development:
  - from the **transport** domain: road networks,
    I move into an example of a "future" domain of platoons;
  - then an example of oil or gas **pipelines**; and finally
  - an example domain of securities trading,
    more specifically: **stock exchanges**.

- The objective of this talk is to show you
  - what it means to develop software using mathematics,
    - albeit it new kind of mathematics,
    - not differential equations, nor integrals, nor statistics, etc.,
    - but mathematical logic and modern algebra,
  - so that software can be shown
    - correct, i.e., no bugs, no **blue screen**, and
    - to met customers' expectations!

A First Discourse: Road Nets and Platooning

- In the first example we show a way of describing,
  - informally, in natural, but precise language, and
  - formally, in some form of mathematics

  software.
- The example is that of
  - road nets — such that enables
  - the transport of people and goods.
  - The example is extended into sketching a domain of platooning!

2.1. Hubs, Links and Nets

1 There are hubs, h:H, and there are links, l:L.

```
type
  H, L
```

2 From a net, n:N, one can **obs**erve finite sets of one or more hubs, hs:Hs, and zero, one or more links, ls:Ls.

type
  N
  Hs = H-**set**
  Ls = L-**set**
value
  obs_Hs: N → Hs
  obs_Ls: N → Ls
axiom
  $\forall$ n:N · **card**(obs_Hs(n))≥1 ∧ **card**(obs_Ls(n))>0

3 So, to express, that is, to describe a domain we need both

a. an informal language, as here English, and

b. a formal language, as here some abstract "programming language"-like mathematical notation[2].

[2]That abstract "programming language"-like mathematical notation is here the `RAISE` [25] `S`ecification `L`anguage [24].

2.2. Unique Identifiers and Mereology

4 Hubs and links have unique identifiers:

type
  HI, LI
value
  uid_H: H $\rightarrow$ HI
  uid_L: L $\rightarrow$ LI
axiom
  HI $\bigcap$ LI $= \{\}$

5 Hubs of a net are connected[3] to (a finite set of) zero, one or more links ***of the net***:

value

mereo_H: H → LI-**set**

[3] The relation of connectedness of parts, such as hubs and links, is an aspect of the **mereology** of such parts.

6 Links of a net are connected to exactly two hubs ***of the net***:

value
  mereo_L: L $\rightarrow$ HI-**set**
axiom
  $\forall$ l:L$\cdot$**card**(mereo_L(l)=2)

7 The *"of the net"* constraints are axiomatised:

axiom
$\forall$ n:N, hs:Hs, ls:Ls $\cdot$
hs = obs_Hs(n) $\wedge$ ls = obs_Ls(n) $\Rightarrow$
let his = hub_ids(hs), lis = link_ids(ls) **in**
$\forall$ h:H $\cdot$ h $\in$ hs $\Rightarrow$ mereo_H(h)$\subseteq$lis $\wedge$
$\forall$ l:L $\cdot$ l $\in$ ls $\Rightarrow$ mereo_L(l)$\subseteq$his
end

8 We used two auxiliary functions above:

value

hub_ids: Hs → HI-**set**

hub_ids(hs) ≡ ∪ {mereo_H(h)|h:H•h ∈ hs}

link_ids: Ls → LI-**set**

link_ids(ls) ≡ ∪ {mereo_L(l)|l:L•l ∈ ls}

9 From a proper hub identifier and a net we can **retr**ieve the identified hub; and

10 From a proper link identifier and a net we can **retr**ieve the identified link.

value

$\mathsf{retr\_H}$: $\mathsf{HI} \rightarrow \mathsf{N} \xrightarrow{\sim} \mathsf{H}$

$\mathsf{retr\_H(hi)(n)} \equiv \textbf{let}\ \mathsf{h{:}H} \cdot \mathsf{h} \in \mathsf{obs\_Hs(n)} \cdot \mathsf{uid\_H(h){=}hi}\ \textbf{in}\ \mathsf{h}\ \textbf{end}$

  pre: $\mathsf{hi} \in \mathsf{his(n)}$

$\mathsf{retr\_L}$: $\mathsf{LI} \rightarrow \mathsf{N} \xrightarrow{\sim} \mathsf{L}$

$\mathsf{retr\_L(li)(n)} \equiv \textbf{let}\ \mathsf{l{:}L} \cdot \mathsf{l} \in \mathsf{obs\_Ls(n)} \cdot \mathsf{uid\_L(l){=}li}\ \textbf{in}\ \mathsf{l}\ \textbf{end}$

  pre: $\mathsf{li} \in \mathsf{lis(n)}$

2.3. Routes

11 By a route we shall understand
an alternating sequence of hub and link identifiers:

type
  R = {| r:(HI|LI)* | wf_R(r) |}
value
  wf_R: (HI|LI)* → **Bool**
  wf_R(r) ≡
    ∀ i:**Nat** · {i,i+1}⊆**inds** r ⇒
      is_LI(r(i))∧is_HI(r(i+1)) ∨ is_HI(r(i))∧is_LI(r(i+1))

12 A pair $\langle$hi,li$\rangle$ are **neighbours** in a net if

type
  neighbours: $(\mathsf{HI}\times\mathsf{LI}) \to \mathsf{N} \to \mathbf{Bool}$
  neighbours(hi,li)(n) $\equiv$ li $\in$ mereo(retr_H(hi)(n))
    pre: hi $\in$ his(n)
  neighbours: $(\mathsf{LI}\times\mathsf{HI}) \to \mathsf{N} \to \mathbf{Bool}$
  neighbours(li,hi)(n) $\equiv$ hi $\in$ mereo(retr_L(li)(n))
    pre: li $\in$ lis(n)

13 Any net of, n:N, defines a possibly infinite set of finite routes:

```
value
  routes: N → R-infset
  routes(n) ≡
    let hs = obs_Hs(n),
        ls = obs_Ls(n) in
    let rs = {⟨⟩}
          ∪ {⟨obs_HI(h)⟩|h:H•h ∈ hs}
          ∪ {⟨obs_LI(h)⟩|l:L•l ∈ ls}
          ∪ {⟨hi,li⟩|hi:HI,li:LI•neighbours(hi,li)(n)}
          ∪ {⟨Li,Hi⟩|Li:lI,Hi:hI•neighbours(Li,Hi)(n)}
          ∪ {r⌢r′|r,r′:R•{r,r′}⊆rs} in
    rs end end
```

2.4. Discussion

- And so forth.
- Nets, hubs, links and mereologies are manifest entities.
- Identifiers and routes are abstract concepts.
- We have presented an abstract description of manifest nets and their hubs and links.
- We could go on describing actions, event and behaviours of nets.
- But we leave that for now.

2.5. Vehicles

14 There are vehicles, v:V.

15 Vehicles have unique identifiers, vi:VI.

16 Vehicles have positions, vp:VP, *on the road net.*

type
  V
  VI
  VP == atH(li:LI,hi:HI,li:LI) | onL(hi:HI,li:LI,r:**Real**,hi:HI)
value
  uid_V: V → VI
  attr_VP: V → VP
axiom
  ∀ onL(_,_,r,_):VP · 0≤r≤1

17 The *on the road net* constraint is axiomatised:

$$
\begin{array}{l}
\textbf{axiom} \\
\quad \forall\ \mathsf{n{:}N,vp{:}VP} \cdot \\
\quad\quad \textbf{case}\ \mathsf{vp}\ \textbf{of}: \\
\quad\quad\quad \mathsf{atH(fli,hi,tli)} \rightarrow \{\mathsf{fli,tli}\} \subseteq \mathsf{mereo\_H(retr\_H(hi)(n))}, \\
\quad\quad\quad \mathsf{onL(fhi,li,r,thi)} \rightarrow \{\mathsf{fli,tli}\} = \mathsf{mereo\_L(retr\_L(li)(n))} \\
\quad\quad \textbf{end}
\end{array}
$$

18 Vehicles move:

- veh(vi,attrs)(vp:atH(hi,fli,tli)) expresses that vehicle **vi** is at a hub: vp:atH(hi,fli,tli).

value
  veh: (VI × ATTRS) → VP → **Unit**
  veh(vi,attrs)(vp:atH(hi,fli,tli)) ≡
    wait ; veh(vi,attrs)(vp)
  ⨅
    let {hi′,thi} = mereo_L(retr_L(tli)(n)) **assert:** hi=hi′ **in**
    veh(vi,attrs)(onL(tli,hi,thi,0)) **end**
  ⨅
    stop

where attrs:ATTRS are some vehicle attributes not mentioned.

- veh(vi,attrs)(vp:onL(li,fhi,thi,r) expresses that vehicle **vi** is on a link at position vp:onL(li,fhi,thi,r).

```
value
  veh: (VI × ATTRS) → VP → Unit
  veh(vi,attrs)(vp:onL(li,fhi,thi,r)) ≡
     veh(vi,attrs)(vp)
   ⌈⌉
     if r + ℓϵ≤1
       then veh(vi,attrs)(onL(li,fhi,thi,r+ℓϵ))
       else let li′:LI•li′ ∈ mereo_H(retr_H(thi)(n)) in
             veh(vi,attrs)(atH(li,thi,li′)) end end
   ⌈⌉
     stop
```

where ℓ_ϵ is some very small real close to 0.

2.6. Platooning

- A platoon (or a convoy) of vehicles
  is a "somehow tightly connected" set of vehicles.
- On a road net vehicles can travel at their own leisure,
  that is, at different speeds,
  weaving in and out of road lanes,
  overtaking one another, etc.
- A "somehow tightly connected" platoon,
  when traveling along a route,
  moves at one speed, with one ac/deceleration,
  thus making all its vehicles travel identically.
- This may allow platoons, and hence their vehicles
  to travel at overall higher average speeds.

- Platoons can be formed and be reformed, dynamically, when traveling along routes:
  - (i) two vehicles can **form** a platoon,
  - (ii) a vehicle can **join** a platoon,
  - (iii) a vehicle can **leave** a platoon,
  - (iv) two platoons can be conjoined,
  - (v) a platoon can be decomposed into two platoons, and
  - (vi) a platoon can be dissolved.
- We shall describe platoons and the beginnings of an algebra of platoons.

2.7. Platoons

19 A platoon is a collection of at least two vehicles.

type
  P
value
  obs_S: P $\rightarrow$ V-**set**
axiom
  $\forall$ p:P $\cdot$ **card** obs_S(s) $\geq$ 2

20 Platoons have unique identifiers:

type
  PI
value
  uid_P: P $\rightarrow$ PI

21 Platoons travel along a route.

22 The platoon leader has a position at the head of the platoon route.

value
  attr_Ldr: P $\rightarrow$ V
  attr_VP: (P|V) $\rightarrow$ VP
  attr_R: P $\rightarrow$ R
axiom
  $\forall$ p:P $\cdot$
      attr_VP(attr_Ldr(p)) = attr_VP(p)
    $\wedge$ **hd**(attr_R(p))=
      case attr_VP(p) **of**:
       atH(_,hi,) $\rightarrow$ hi,
       onL(_,li,_,_)$\rightarrow$li
      end

23 At any one time there are a finite number of zero, one or more uniquely identified platoons in existence, that is, in the platoon state, $\sigma : \Sigma$:

type
  Σ
value
  obs_Ps: $\Sigma \rightarrow$ P-**set**
axiom
  $\forall\ \sigma{:}\Sigma \cdot$
    $\forall$ p,p′:P $\cdot$ p$\neq$p′$\wedge${p,p′}$\subseteq$obs_Ps(σ)
      $\Rightarrow$ uid_P(p) $\neq$ uid_P(p′)

24 We define some auxiliary (overloaded) functions:

value

πs: $\Sigma \rightarrow$ PI-**set**
πs(σ) $\equiv$ {uid_P(p)|p:P•p $\in$ obs_Ps(σ)}

pl: PI $\rightarrow \Sigma \rightarrow$ P
pl(π)(σ) $\equiv$
ι p:P • p $\in$ obs_Ps(σ) $\wedge$ uid_P(p)=π
pre: $\exists$ p:P • p $\in$ obs_Ps(σ) $\wedge$ uid_P(p)=π

vs: PI $\rightarrow \Sigma \rightarrow$ V-**set**
vs(π)(σ) $\equiv$ obs_S(pl(π)(σ))
pre: p:P • p $\in$ obs_Ps(σ) $\wedge$ uid_P(p)=π

vs: $\Sigma \rightarrow$ V-**set**
vs(σ) $\equiv \cup$ { vs(π)(σ) | π:PI • $\exists$ p:P • p $\in$ obs_Ps(σ) $\wedge$ uid_P(p)=π}

vis: $\Sigma \rightarrow$ VI-**set**
vis(σ) $\equiv$ {uid_V(v)|v:V•v $\in$ vs(σ)}

25 Two vehicles not in a platoon can **form** a platoon:

value

$$
\begin{array}{l}
\mathsf{form}\colon \mathsf{V} \times \mathsf{V} \to \Sigma \xrightarrow{\sim} \Sigma \times \mathsf{PI} \\
\mathsf{form}(\mathsf{v},\mathsf{v}')(\sigma)\ \mathbf{as}\ (\sigma',\pi) \\
\quad \mathbf{pre}\colon\ \mathsf{v} \neq \mathsf{v}' \\
\qquad \wedge\ \{\mathsf{v},\mathsf{v}'\} \cap \mathsf{vis}(\sigma) = \{\}, \\
\quad \mathbf{post}\colon \exists\ \pi{:}\mathsf{PI} \cdot \pi \notin \pi\mathsf{s}(\sigma) \\
\qquad \wedge\ \pi\mathsf{s}(\sigma') = \pi\mathsf{s}(\sigma) \cup \{\pi\} \\
\qquad \wedge\ \exists\ \mathsf{p}{:}\mathsf{P} \cdot \mathsf{uid\_P}(\mathsf{p}){=}\pi \Rightarrow \mathsf{p} \in \mathsf{obs\_Ps}(\sigma') \\
\qquad \wedge\ \{\mathsf{v},\mathsf{v}'\}{=}\mathsf{obs\_Vs}(\mathsf{p}) \\
\qquad \wedge\ \mathsf{ps}(\sigma') = \mathsf{ps}(\sigma) \cup \{\mathsf{p}\}
\end{array}
$$

26 A vehicle can **join** a platoon:

$$
\begin{array}{l}
\textbf{value} \\
\quad \mathsf{join}\colon \mathsf{V} \times \mathsf{PI} \to \Sigma \stackrel{\sim}{\to} \mathsf{Sigma} \\
\quad \mathsf{join}(\mathsf{v},\pi)(\sigma)\ \textbf{as}\ \sigma' \\
\qquad \textbf{pre}\colon\ \mathsf{v} \notin \mathsf{vs}(\sigma) \wedge \pi \in \pi\mathsf{s}(\sigma) \\
\qquad \textbf{post}\colon\ \textbf{let}\ \mathsf{p}{=}\mathsf{pl}(\pi)(\sigma)\ \textbf{in} \\
\qquad\qquad \textbf{let}\ \mathsf{p}'{:}\mathsf{P}{\cdot}\mathsf{uid\_P}(\mathsf{p}){=}\mathsf{uid\_P}(\mathsf{p}')\wedge\mathsf{obs\_Vs}(\mathsf{p}){=}\mathsf{obs\_Vs}(\mathsf{p})\cup\{\mathsf{v}\}\ \textbf{in} \\
\qquad\qquad \pi \in \pi\mathsf{s}(\sigma') \\
\qquad\quad \wedge\ \mathsf{ps}(\sigma'){=}\mathsf{ps}(\sigma)\setminus\{\mathsf{pl}(\pi)(\sigma)\}\cup\{\mathsf{pl}(\pi)(\sigma')\} \\
\qquad\qquad \textbf{end end}
\end{array}
$$

2.8. Discussion

- And so forth.
- Platooning is unlike trains.
  - Train com- and decomposition occurs while trains are not running.
  - Platoon com- and decomposition occurs only while platoons are running.
- Platooning requires
  - that participating vehicles are
    - are electronically and electron-mechanically instrumented
    - for self-drive,
    - for co-operating, via a “platooning cloud”, with platoons,
    - and so forth.

- Platooning also requires
  - that there is some GPSS-supported "cloud" that
  - can monitor & control platoon traffic.
- The example of platooning
  - is "new", free for you to "hijack",
  - and is one that you might
    earn a fortune researching, developing and marketing.
  - I invite you to do that !

A Second Discourse: Pipelines
3.1. Pipeline Structures

27 A **pipeline** consists of an indefinite number of pipeline **units**.

28 A pipeline unit is either a **well**, or a **pipe**, or a **pump**, or a **valve**, or a **fork**, or a **join**, or a **sink**.

29 All these unit sorts are atomic and disjoint.

type
27. PL, U, We, Pi, Pu, Va, Fo, Jo, Si
27. Well, Pipe, Pump, Valv, Fork, Join, Sink
value
27. **obs_part**_Us: PL → U-**set**
type
28. U == We | Pi | Pu | Va | Fo | Jo | Si
29. We::Well, Pi::Pipe, Pu::Pump, Va::Valv, Fo:Fork, Jo::Join, Si::Sink

3.2. Pipeline Well-formedness

- Pipeline units serve to conduct fluid or gaseous material.
- The flow of these occur in only one direction: from so-called input to so-called output.

30 Wells have exactly one connection to an output unit.

31 Pipes, pumps and valves have exactly one connection from an input unit and one connection to an output unit.

32 Forks have exactly one connection from an input unit and exactly two connections to distinct output units.

33 Joins have exactly one two connection from distinct input units and one connection to an output unit.

34 Sinks have exactly one connection from an input unit.

35 Thus we model the mereology of a pipeline unit as a pair of disjoint sets of unique pipeline unit identifiers.

type
35. $\mathsf{UM}'=(\mathsf{UI}\textbf{-set}\times\mathsf{UI}\textbf{-set})$
35. $\mathsf{UM}=\{|(\mathsf{iuis},\mathsf{ouis}):\mathsf{UI}\textbf{-set}\times\mathsf{UI}\textbf{-set}\cdot\mathsf{iuis}\cap\mathsf{ouis}=\{\}|\}$
value
35. **obs_mereo**_U: UM
axiom [Well−formedness of Pipeline Systems, PLS (0)]
 $\forall$ pl:PL,u:U $\cdot$ u $\in$ **obs_part**_Us(pl) $\Rightarrow$
  let (iuis,ouis)=**obs_mereo**_U(u) **in**
  case (**card** iuis,**card** ouis) **of**
30.   (0,1) $\rightarrow$ **is**_We(u),
31.   (1,1) $\rightarrow$ **is**_Pi(u)$\vee$**is**_Pu(u)$\vee$**is**_Va(u),
32.   (1,2) $\rightarrow$ **is**_Fo(u),
33.   (2,1) $\rightarrow$ **is**_Jo(u),
34.   (1,0) $\rightarrow$ **is**_Si(u)
  end end

3.3. Pipeline Flow

- Let us postulate a[n attribute] sort Flow.
- We now wish to examine the flow of liquid (or gaseous) material in pipeline units.
- We use two types

  36 F for “productive” flow, and L for wasteful leak.
- Flow and leak is measured, for example, in terms of volume of material per second.
- We then postulate the following unit attributes
  - “measured” at the point of in- or out-flow
  - or in the interior of a unit.

37 current flow of material into a unit input connector,

38 maximum flow of material into a unit input connector while maintaining laminar flow,

39 current flow of material out of a unit output connector,

40 maximum flow of material out of a unit output connector while maintaining laminar flow,

41 current leak of material at a unit input connector,

42 maximum guaranteed leak of material at a unit input connector,

43 current leak of material at a unit input connector,

44 maximum guaranteed leak of material at a unit input connector,

45 current leak of material from “within” a unit, and

46 maximum guaranteed leak of material from “within” a unit.

type

36. F, L

value

37. **attr**_cur_iF: U → UI → F
38. **attr**_max_iF: U → UI → F
39. **attr**_cur_oF: U → UI → F
40. **attr**_max_oF: U → UI → F
41. **attr**_cur_iL: U → UI → L
42. **attr**_max_iL: U → UI → L
43. **attr**_cur_oL: U → UI → L
44. **attr**_max_oL: U → UI → L
45. **attr**_cur_L: U → L
46. **attr**_max_L: U → L

- It may be difficult or costly, or both,
  - to ascertain flows and leaks in materials-based domains.
  - But one can certainly speak of these concepts.
  - This casts new light on **domain modeling**.
  - That is in contrast to
    - incorporating such notions of flows and leaks
    - in **requirements modeling**
  - where one has to show implement-ability.

47 For every unit of a pipeline system, except the well and the sink units, the following law apply.

48 The flows into a unit equal

a. the leak at the inputs

b. plus the leak within the unit

c. plus the flows out of the unit

d. plus the leaks at the outputs.

axiom [Well−formedness of Pipeline Systems, PLS (1)]

```
47. ∀ pls:PLS,b:B\We\Si,u:U ·
47.      b ∈ obs_part_Bs(pls)∧u=obs_part_U(b)⇒
47.      let (iuis,ouis) = obs_mereo_U(u) in
48.      sum_cur_iF(iuis)(u) =
48a..       sum_cur_iL(iuis)(u)
48b..    ⊕ attr_cur_L(u)
48c..    ⊕ sum_cur_oF(ouis)(u)
48d..    ⊕ sum_cur_oL(ouis)(u)
47.      end
```

49 The sum_cur_iF
(cf. Item 48) sums current input flows over all input connectors.

50 The sum_cur_iL
(cf. Item 48a.) sums current input leaks over all input connectors.

51 The sum_cur_oF
(cf. Item 48c.) sums current output flows over all output connectors.

52 The sum_cur_oL
(cf. Item 48d.) sums current output leaks over all output connectors.

49. sum_cur_iF: UI-**set** $\rightarrow$ U $\rightarrow$ F
49. sum_cur_iF(iuis)(u) $\equiv$ $\oplus$ {**attr**_cur_iF(ui)(u)|ui:UI·ui $\in$ iuis}
50. sum_cur_iL: UI-**set** $\rightarrow$ U $\rightarrow$ L
50. sum_cur_iL(iuis)(u) $\equiv$ $\oplus$ {**attr**_cur_iL(ui)(u)|ui:UI·ui $\in$ iuis}
51. sum_cur_oF: UI-**set** $\rightarrow$ U $\rightarrow$ F
51. sum_cur_oF(ouis)(u) $\equiv$ $\oplus$ {**attr**_cur_iF(ui)(u)|ui:UI·ui $\in$ ouis}
52. sum_cur_oL: UI-**set** $\rightarrow$ U $\rightarrow$ L
52. sum_cur_oL(ouis)(u) $\equiv$ $\oplus$ {**attr**_cur_iL(ui)(u)|ui:UI·ui $\in$ ouis}

$\oplus$: (F|L) $\times$ (F|L) $\rightarrow$ F

53 For every pair of connected units of a pipeline system the following **law** apply:

a. the flow out of a unit directed at another unit minus the leak at that output connector

b. equals the flow into that other unit at the connector from the given unit plus the leak at that connector.

axiom [Well−formedness of Pipeline Systems, PLS (2)]

53. $\forall$ pls:PLS,b,b′:B,u,u′:U•
53. {b,b′}⊆**obs_part**_Bs(pls)∧b≠b′∧u′=**obs_part**_U(b′)
53. ∧ **let** (iuis,ouis)=**obs_mereo**_U(u),(iuis′,ouis′)=**obs_mereo**_U(u′)
53. ui=**uid**_U(u),ui′=**uid**_U(u′) **in**
53. ui ∈ iuis ∧ ui′ ∈ ouis′ ⇒
53a.. **attr**_cur_oF(u′)(ui′) − **attr**_leak_oF(u′)(ui′)
53b.. = **attr**_cur_iF(u)(ui) + **attr**_leak_iF(u)(ui)
53. **end**
53. **comment:** b′ precedes b

3.4. Discussion

- Pipelines, whether oil, gas, water, or other are of increasing importance.
- Pipelines need be computer monitored & controlled.
- The sketched description need be further researched & developed:
  - The model, as presented, basically models discrete properties.
  - A fluid dynamics model is needed.
  - The two models need be formally related.
  - To do so is a serious research issue.
- I invite you to work on such problems.

A Third Discourse: Stock Exchanges

- The market of financial instruments is only partially understood.
- Here we present a model of some crucial properties of selling and buying stocks.
- The model, although that of the *Tokyo Stock Exchange, TSE,* can be simply modified to model other stock exchanges.
- For example, the *New York Stock Exchange, NYSE,* or other.
- Such models need be integrated with models of ("high street") banking, mortgaging, portfolio management, etc.
- By my guest !

4.1. Market and Limit Offers and Bids

54 A **market sell offer** or **buy bid** specifies

a. the unique identification of the stock,

b. the number of stocks to be sold or bought, and

c. the unique name of the seller.

55 A limit sell offer or buy bid specifies the same information as a market sell offer or buy bid (i.e., Items 54a.–54c.), and

d. the price at which the identified stock is to be sold or bought.

56 A trade order is either a (**mkMkt** marked) market order or (**mkLim** marked) a limit order.

57 A trading command is either a sell order or a buy bid.

58 The sell orders are made unique by the **mkSell** "make" function.

59 The buy orders are made unique by the **mkBuy** "make" function.

type

54 Market = Stock_id × Number_of_Stocks × Name_of_Customer

54a. Stock_id

54b. Number_of_Stocks = $\{|$n·n:**Nat**$\wedge$n>1$|\}$

54c. Name_of_Customer

55 Limit = Market × Price

55d. Price = $\{|$n·n:**Nat**$\wedge$n>1$|\}$

56 Trade == mkMkt(m:Market) | mkLim(l:Limit)

57 Trading_Command = Sell_Order | Buy_Bid

58 Sell_Order == mkSell(t:Trade)

59 Buy_Bid == mkBuy(t:Trade)

4.2. Order Books

60 We introduce a concept of linear, discrete time, T.

61 We also introduce a concept of Order number.

62 For each stock the stock exchange keeps an Order Book.

63 An order book for stock s_{id}:SI keeps track of limit buy bids and limit sell offers (for the *id*entified *s*tock, s_{id}:SI), as well as the market buy bids and sell offers; that is, for each price

d. the number stocks, by unique order number, offered for sale at that price, that is, limit Sell Orders, and

e. the number of stocks, by unique order number, bid for buying at that price, that is, limit Buy Bid orders.

f. If an offer is a market sell offer, then the number of stocks to be sold is recorded, and if an offer is a market buy bid (also an offer), then the number of stocks to be bought is recorded,

64 Over time the (Tokyo) Stock Exchange (TSE) displays series of full order books.

65 A Trade Unit, tu:TU, is a pair of a unique order number and an amount (a number larger than 0) of stocks.

66 An Amount designates a number of one or more stocks.

type

60 T

61 On

62 All_Stocks_Order_Book = Stock_Id $\overrightarrow{m}$ Stock_Order_Book

63 Stock_Order_Book = (Price $\overrightarrow{m}$ Orders) × Market_Offers

63 Orders:: so:Sell_Orders × bo:Buy_Bids

63d. Sell_Orders = On $\overrightarrow{m}$ Amount

63e. Buy_Bids = On $\overrightarrow{m}$ Amount

63f. Market_Offers :: mkSell(n:**Nat**) × mkBuy(n:**Nat**)

64 TSE = T $\overrightarrow{m}$ All_Stocks_Order_Book

65 TU = On × Amount

66 Amount = {|n·**Nat**∧n≥1|}

4.3. Aggregate Offers

67 We introduce the concepts of aggregate sell and buy orders for a given stock at a given price (and at a given time).

68 The aggregate sell orders for a given stock at a given price is

g. the stocks being market sell offered and

h. the number of stocks being limit offered for sale at that price or lower

value

68 aggr_sell: All_Stocks_Order_Book $\times$ Stock_Id $\times$ Price $\rightarrow$ **Nat**
68 aggr_sell(asob,sid,p) $\equiv$
68 **let** ((sos,_),(mkSell(ns),_)) = asob(sid) **in**
68g. ns +
68h. all_sell_summation(sos,p) **end**

all_sell_summation: Sell_Orders $\times$ Price $\rightarrow$ **Nat**
all_sell_summation(sos,p) $\equiv$
let ps = $\{$p′|p′:Prices $\cdot$ p′ $\in$ **dom** sos $\wedge$ p′ $\geq$ p$\}$ **in** accumulate(sos,ps)(0) **end**

69 The aggregate bur bids for a given stock at a given price is

a. including the stocks being market bid offered and

b. the number of stocks being limit bid for buying at that price or higher

value

69 aggr_buy: All_Stocks_Order_Book $\times$ Stock_Id $\times$ Price $\rightarrow$ **Nat**
69 aggr_buy(asob,sid,p) $\equiv$
69 **let** ((_,bbs),(_,mkBuy(nb))) = asob(sid) **in**
69a. nb +
69b. nb + all_buy_summation(bbs,p) **end**

all_buy_summation: Buy_Bids $\times$ Price $\rightarrow$ **Nat**
all_buy_summation(bbs,p) $\equiv$
let ps = $\{$p$'$|p$'$:Prices $\cdot$ p$'$ $\in$ **dom** bos $\wedge$ p$'$ $\leq$ p$\}$ **in** accumulate(bbs,ps)(0) **e**

- The auxiliary **accumulate** function is shared between the **all_sell_summation** and the **all_buy_summation** functions. It sums the amounts of limit stocks in the price range of the **accumulate** function argument **ps**.
- The auxiliary **sum** function sums the amounts of limit stocks — "pealing off" the their unique order numbers.

value

accumulate: (Price $\overrightarrow{m}$ Orders) × Price-**set** → **Nat** → **Nat**
accumulate(pos,ps)(n) ≡
  case ps **of**
    {} → n,
    {p}∪ ps′ → accumulate(pos,ps′)(n+sum(pos(p)){**dom** pos(p)}) **end**

sum: (Sell_Orders|Buy_Bids) → On-**set** → **Nat**
sum(ords)(ns) ≡
  case ns **of**
    {} → 0,
    {n}∪ ns′ → ords(n)+sum(ords)(ns′) **end**

- To handle the sub_limit_sells and sub_limit_buys indicated by Item 71c. on the facing slide of the Itayose "algorithm" we need the corresponding sub_sell_summation and sub_buy_summation functions:

value

$$\mathsf{sub\_sell\_summation}: \mathsf{Stock\_Order\_Book} \times \mathsf{Price} \to \mathbf{Nat}$$
$$\mathsf{sub\_sell\_summation(((sos,\_),(ns,\_)),p)} \equiv \mathsf{ns} +$$
$$\quad \mathbf{let}\ \mathsf{ps} = \{\mathsf{p}'|\mathsf{p}':\mathsf{Prices} \cdot \mathsf{p}' \in \mathbf{dom}\ \mathsf{sos} \wedge \mathsf{p}' > \mathsf{p}\}\ \mathbf{in}$$
$$\quad \mathsf{accumulate(sos,ps)(0)}\ \mathbf{end}$$

$$\mathsf{sub\_buy\_summation}: \mathsf{Stock\_Order\_Book} \times \mathsf{Price} \to \mathbf{Nat}$$
$$\mathsf{sub\_buy\_summation(((\_,bbs),(\_,nb)),p)} \equiv \mathsf{nb} +$$
$$\quad \mathbf{let}\ \mathsf{ps} = \{\mathsf{p}'|\mathsf{p}':\mathsf{Prices} \cdot \mathsf{p}' \in \mathbf{dom}\ \mathsf{bos} \wedge \mathsf{p}' < \mathsf{p}\}\ \mathbf{in}$$
$$\quad \mathsf{accumulate(bbs,ps)(0)}\ \mathbf{end}$$

4.4. The TSE Itayose "Algorithm"

70 The TSE practices the so-called Itayose "algorithm" to decide on opening and closing prices[4]. That is, the Itayose "algorithm" determines a single so-called 'execution' price, one that matches sell and buy orders[5]:

71 The "matching sell and buy orders" rules:

a. *All market orders must be 'executed'*[6].

b. *All limit orders to sell/buy at prices lower/higher than the 'execution price'*[7] *must be executed.*

c. *The following amount of limit orders to sell or buy at the execution prices must be executed: the entire amount of either all sell or all buy orders, and at least one 'trading unit'*[8] *from 'the opposite side of the order book'*[9].

[4][26, pp 59, col. 1, lines 4-3 from bottom, cf. Page ??]

[5][26, pp 59, col. 2, lines 1–3 and Items 1.–3. after yellow, four line 'insert', cf. Page ??] These items 1.–3. are reproduced as "our" Items 71a.–71c..

[6]To execute an order:

[7]Execution price:

[8]Trading unit:

[9]The opposite side of the order book:

value

```
71 match: All_Stocks_Order_Book × Stock_Id → Price-set
71 match(asob,sid) as ps
71     pre: sid ∈ dom asob
71     post: ∀ p′:Price • p′ ∈ ps ⇒
71′            ∃ os:On-set •
71a.′                    market_buys(asob(sid))
71b.′                  + sub_limit_buys(asob(sid))(p′)
71c.′                  + all_priced_buys(asob(sid))(p′)
71a.′                =   market_sells(asob(sid))
71b.′                  + sub_limit_sells(asob(sid))(p′)
71c.′                  + some_priced_buys(asob(sid))(p′)(os) ∨
71″            ∃ os:On-set •
71a.″                    market_buys(asob(sid))
71b.″                  + sub_limit_buys(asob(sid))(p′)
71c.″                  + some_priced_buys(asob(sid))(p′)(os)
71a.″                =   market_sells(asob(sid))
71b.″                  + sub_limit_sells(asob(sid))(p′)
71c.″                  + all_priced_buys(asob(sid))(p′) ∨
```

- The **match** function calculates a set of prices for each of which a match can be made.
- The set may be empty: there is no price which satisfies the match rules (cf. Items 71a.–71c. below).
- The set may be a singleton set: there is a unique price which satisfies match rules Items 71a.–71c..
- The set may contain more than one price: there is not a unique price which satisfies match rules Items 71a.–71c..
- The single ($'$) and the double ($''$) quoted (71a.–71c.) group of lines, in the **match** formulas above, correspond to the **Itayose** "algorithm"s Item 71c. *'opposite sides of the order book'* description.
- The existential quantification of a set of order numbers of lines $71'$ and $71''$ correspond to that "algorithms" (still Item 71c.) point of *at least one 'trading unit'*. It may be that the **post** condition predicate is only fulfilled for all trading units – so be it.

```
value
  market_buys: Stock_Order_Book → Amount
  market_buys((_,(_,mkBuys(nb))),p) ≡ nb

  market_sells: Stock_Order_Book → Amount
  market_sells((_,(mkSells(ns),_)),p) ≡ ns

  sub_limit_buys: Stock_Order_Book → Price → Amount
  sub_limit_buys(((,bbs),_))(p) ≡ sub_buy_summation(bbs,p)

  sub_limit_sells: Stock_Order_Book → Price → Amount
  sub_limit_sells((sos,_))(p) ≡ sub_sell_summation(sos,p)

  all_priced_buys: Stock_Order_Book → Price → Amount
  all_priced_buys((_,bbs),_)(p) ≡ sum(bbs(p))

  all_priced_sells: Stock_Order_Book → Price → Amount
  all_priced_sells((sos,_),_)(p) ≡ sum(sos(p))

  some_priced_buys: Stock_Order_Book → Price → On-set → Amount
  some_priced_buys((_,bbs),_)(p)(os) ≡
    let tbs = bbs(p) in if {}≠os∧os⊆dom tbs then sum(tbs)(os) else 0 end end

  some_priced_sells: Stock_Order_Book → Price → On-set → Amount
  some_priced_sells((sos,_),_)(p)(os) ≡
    let tss = sos(p) in if {}≠os∧os⊆dom tss then sum(tss)(os) else 0 end end
```

4.5. Discussion

- The *TSE* is further detailed in [9, 18].
- It would be interesting to compare the *TSE* model, to that of similar models for the *Frankfurt, Hong Kong, Moscow, NYSE, NASDAQ, Paris, Shanghai* and other stock exchanges.
- Similar models for commodity exchanges (grain (Chicago), metals (London), etc.) ought be researched & developed.
- Perhaps a generic model of financial instrument and commodity exchanges can be researched & developed.
- And a family of strongly related (*"product line"*) software ought be derivable from this generic domain description.
- By my guest !

Conclusion

5.1. What Have We Learned ? — Hopefully

- You have seen informal and formal models of domains such as:
  - road net, vehicles and platooning,
  - pipelines, and
  - stock matching.
- You have therefore seen that
  - one can talk of large systems
  - very precisely and very comprehensively
  - using natural language and mathematics.
- That should give you, I hope, the desire to do likewise!

5.2. A Context for Domain Engineering

- *Before software can be developed,*
- *we must have a reasonable grasp of its requirements.*
- *Before requirements can be understood,*
- *we must have a reasonable grasp of the domain in which they “reside”.*

5.3. A Triptych Of Software Engineering

- As a result we see software development ("ideally") proceeding as follows:
  - (i) first we research & develop a description of the domain;
  - (ii) then we research & develop a prescription of the requirements by *"deriving"* these (more-or-less) from the domain description;
  - (iii) finally we design the $\mathcal{S}$oftware
    - from the $\mathcal{R}$equirements
    - such that we can prove ($\models$) the $\mathcal{S}$oftware correct
    - with respect to the $\mathcal{R}$equirements
    - and in the context of the $\mathcal{D}$omain model:

$$\mathcal{D}, \mathcal{S} \models \mathcal{R}$$

- That is:
  - **Software Engineering** =
    - **Domain Engineering**
    - $\oplus$ **Requirements Engineering**
    - $\oplus$ **Software Design**

- *Tupolev* would not hire an aircraft design engineer
  - unless that person was well educated in aeronautics,
  - fluid dynamics, etc.
- L.M. Ericsson would not hire a radio antenna design engineer
  - unless that person was well educated in electro-magnetic field theory
  - and knows how to interpret Maxwell's Equations.

- So it will be, in future, for software engineers, sales and marketeers:
  - they must know how to read and some even to write domain models,
  - and they must know the basics of how to turn these into software.
- So better get started.
- To start up your own software company you must specialise in a domain:
  - that means, you must make sure
    that your corporate asset is an appropriate domain model,
  - and that you continue to research, develop and adapt your domain model(s) to a competitive market.
  - Your highly tuned domain model(s) make you stay ahead of the market.

5.4. Tony Hoare's Summary on 'Domain Modeling'

"There are many unique contributions that can be made by domain modeling[10].

1 The models describe all aspects of the real world that are relevant for any good software design in the area. They describe possible places to define the system boundary for any particular project.

2 They make explicit the preconditions about the real world that have to be made in any embedded software design, especially one that is going to be formally proved.

3 They describe the whole range of possible designs for the software, and the whole range of technologies available for its realisation.

4 They provide a framework for a full analysis of requirements, which is wholly independent of the technology of implementation.

5 They enumerate and analyse the decisions that must be taken earlier or later in any design project, and identify those that are independent and those that conflict. Late discovery of feature interactions can be avoided."

[10] E-Mail to Dines Bjørner, July 19, 2006

5.5. Acknowledgements

- I thank Academician *Victor Petrovich Ivannikov*
  for inviting me to Moscow —
- in particular challenging me to write this talk.
- I thank his assistant, *Ms Darya*, for arranging all practical details.

Bibliography
6.1. Published Papers

- I have thought about domain engineering for more than 20 years.
- But serious, focused writing only started to appear since [3, Part IV] — with [2, 1] being exceptions:
  - [4] suggests a number of domain science and engineering research topics;
  - [7] covers a concept of domain facets;
  - [22] explores compositionality and Galois connections;
  - [5, 21] show how to systematically, but, of course, not automatically, “derive” requirements prescriptions from domain descriptions;
  - [10] takes the triptych software development as a basis for outlining principles for believable software management;

- [6, 14] presents a model for Stanisław Leśniewski's [23] concept of mereology;
- [8, 11] present an extensive example and is otherwise a precursor for the present paper;
- [12] presents, based on the **TripTych** view of software development as ideally proceeding from domain description via requirements prescription to software design, concepts such as software demos and simulators;
- [13] analyses the **TripTych**, especially its domain engineering approach, with respect to Maslow's [11] and Peterson's and Seligman's [12] notions of humanity: how can computing relate to notions of humanity;

[11] *Theory of Human Motivation.* Psychological Review 50(4) (1943):370-96; and *Motivation and Personality*, Third Edition, Harper and Row Publishers, 1954.
[12] *Character strengths and virtues: A handbook and classification.* Oxford University Press, 2004

- the first part of [15] is a precursor for the present paper with its second part presenting a first formal model of the elicitation process of analysis and description based on the prompts more definitively presented in the current paper;
- [16] focus on domain safety criticality;
- [17] is the definitive paper on *manifest domains: analysis & description.*

6.2. Reports

1 *A Railway Systems Domain:*
http://euler.fd.cvut.cz/railwaydomain/ (2003)

2 *Models of IT Security. Security Rules & Regulations:*
it-security.pdf (2006)

3 *A Container Line Industry Domain:* container-paper.pdf (2007)

4 *The "Market": Consumers, Retailers, Wholesalers, Producers:*
themarket.pdf (2007)

5 *What is Logistics ?:* logistics.pdf (2009)

6 *A Domain Model of Oil Pipelines:* `pipeline.pdf` (2009)

7 *Transport Systems:* `comet/comet1.pdf` (2010)

8 *The Tokyo Stock Exchange:* `todai/tse-1.pdf` and `todai/tse-2.pdf` (2010)

9 *On Development of Web-based Software. A Divertimento:* `wfdftp.pdf` (2010)

10 *Documents (incomplete draft):* `doc-p.pdf` (2013)

6.3. References

[1] D. Bjørner. Michael Jackson's Problem Frames: Domains, Requirements and Design. In L. ShaoYang and M. Hinchley, editors, *ICFEM'97: International Conference on Formal Engineering Methods*, Los Alamitos, November 12–14 1997. IEEE Computer Society. Final Version.

[2] D. Bjørner. Domain Engineering: A "Radical Innovation" for Systems and Software Engineering ? In *Verification: Theory and Practice*, volume 2772 of *Lecture Notes in Computer Science*, Heidelberg, October 7–11 2003. Springer–Verlag. The Zohar Manna International Conference, Taormina, Sicily 29 June – 4 July 2003. Final draft version.

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

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

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

[6] D. Bjørner. On Mereologies in Computing Science. In *Festschrift: Reflections on the Work of C.A.R. Hoare*, History of Computing (eds. Cliff B. Jones, A.W. Roscoe and Kenneth R. Wood), pages 47–70, London, UK, 2009. Springer.

[7] D. Bjørner. Domain Engineering. In P. Boca and J. Bowen, editors, *Formal Methods: State of the Art and New Directions*,

Eds. Paul Boca and Jonathan Bowen, pages 1–42, London, UK, 2010. Springer.

[8] D. Bjørner. Domain Science & Engineering – From Computer Science to The Sciences of Informatics, Part I of II: The Engineering Part. *Kibernetika i sistemny analiz*, (4):100–116, May 2010.

[9] D. Bjørner. The Tokyo Stock Exchange Trading Rules. R&D Experiment, Fredsvej 11, DK-2840 Holte, Denmark, January, February 2010.

[10] D. Bjørner. Believable Software Management. *Encyclopedia of Software Engineering*, 1(1):1–32, 2011.

[11] D. Bjørner. Domain Science & Engineering – From Computer Science to The Sciences of Informatics Part II of II: The Science Part. *Kibernetika i sistemny analiz*, (2):100–120, May 2011.

[12] D. Bjørner. Domains: Their Simulation, Monitoring and Control – A Divertimento of Ideas and Suggestions. In *Rainbow of*

Computer Science, Festschrift for Hermann Maurer on the Occasion of His 70th Anniversary., Festschrift (eds. C. Calude, G. Rozenberg and A. Saloma), pages 167–183. Springer, Heidelberg, Germany, January 2011.

[13] D. Bjørner. *Domain Science and Engineering as a Foundation for Computation for Humanity*, chapter 7, pages 159–177. Computational Analysis, Synthesis, and Design of Dynamic Systems. CRC [Francis & Taylor], 2013. (eds.: Justyna Zander and Pieter J. Mosterman).

[14] D. Bjørner. *A Rôle for Mereology in Domain Science and Engineering*. Synthese Library (eds. Claudio Calosi and Pierluigi Graziani). Springer, Amsterdam, The Netherlands, October 2014.

[15] D. Bjørner. Domain Analysis: Endurants – An Analysis & Description Process Model. In S. Iida, J. Meseguer, and K. Ogata,

editors, *Specification, Algebra, and Software: A Festschrift Symposium in Honor of Kokichi Futatsugi*. Springer, May 2014.

[16] D. Bjørner. Domain Engineering – A Basis for Safety Critical Software. Invited Keynote, ASSC2014: Australian System Safety Conference, Melbourne, 26–28 May, December 2014.

[17] D. Bjørner. Manifest Domains: Analysis & Description. Research Report, 2014. Part of a series of research reports: [19, 20], Being submitted.

[18] D. Bjørner. The Tokyo Stock Exchange Trading Rules. R&D Experiment, Fredsvej 11, DK-2840 Holte, Denmark, January and February, 2010. Version 1, 78 pages: many auxiliary appendices, Version 2, 23 pages: omits many appendices and corrects some errors..

[19] D. Bjørner. Domain Analysis & Description: Models of Processes and Prompts. Research Report, To be completed early 2014. Part

of a series of research reports: [17, 20].

[20] D. Bjørner. From Domains to Requirements. Research Report, To be completed early 2015. Part of a series of research reports: [17, 19].

[21] D. Bjørner. The Role of Domain Engineering in Software Development. Why Current Requirements Engineering Seems Flawed! In *Perspectives of Systems Informatics*, volume 5947 of *Lecture Notes in Computer Science*, pages 2–34, Heidelberg, Wednesday, January 27, 2010. Springer.

[22] D. Bjørner and A. Eir. Compositionality: Ontology and Mereology of Domains. Some Clarifying Observations in the Context of Software Engineering in July 2008, eds. Martin Steffen, Dennis Dams and Ulrich Hannemann. In *Festschrift for Prof. Willem Paul de Roever Concurrency, Compositionality, and*

Correctness, volume 5930 of *Lecture Notes in Computer Science*, pages 22–59, Heidelberg, July 2010. Springer.

[23] R. Casati and A. Varzi. *Parts and Places: the structures of spatial representation*. MIT Press, 1999.

[24] C. W. George, P. Haff, K. Havelund, A. E. Haxthausen, R. Milne, C. B. Nielsen, S. Prehn, and K. R. Wagner. *The RAISE Specification Language*. The BCS Practitioner Series. Prentice-Hall, Hemel Hampstead, England, 1992.

[25] C. W. George, A. E. Haxthausen, S. Hughes, R. Milne, S. Prehn, and J. S. Pedersen. *The RAISE Development Method*. The BCS Practitioner Series. Prentice-Hall, Hemel Hampstead, England, 1995.

[26] T. Tamai. Social Impact of Information System Failures. *Computer, IEEE Computer Society Journal*, 42(6):58–65, June 2009.