Start of Lecture 3: DOMAINS: Intrinsics – Rules & Regulations

3. An Ontology of Domain Facets

3.0.1. Definitions

Definition: Domain. *An area of activity which some software is to support (or supports) or partially or fully automate (resp. automates).*

- The term 'application domain' is considered synonymous with the term 'domain'.

Definition: Domain Description. *A textual, informal or formal document which describes a domain* ***as it is****.*

Usually a domain description is a set of documents with many parts recording many facets of the domain: The

- *business processes*,
- *intrinsics*,
- *support technology*,
- *rules and regulations*,
- *management and organisation*, and the
- *human behaviours*.

Definition: Domain Engineering.

- *The engineering of the development of a domain description, from*
  - *identification of domain stakeholders, via*
  - *domain acquisition,*
  - *domain analysis,*
  - *terminologisation,*

  and
  - *domain description*

  to
  - *domain validation and*
  - *domain verification.*

Definition: Domain Facet.

- *By a domain facet we understand*
  - *one amongst a finite set of generic ways of analysing a domain:*
  - *A view of the domain, such that the different facets cover conceptually different views,*
  - *and such that these views together cover the domain.*
- We consider here the following domain facets:
  - *business process*es,
  - *intrinsics*,
  - *support technology*,
  - *rules and regulations*,
  - *management and organisation*, and
  - *human behaviour*.

3.0.2. What Can Be Observed

- "Whether you can observe a thing or not depends on the theory which you use. It is the theory which decides what can be observed."
- Albert Einstein objecting to the placing of observables at the heart of the new quantum mechanics, during Heisenberg's 1926 lecture at Berlin; related by Heisenberg, quoted in *Unification of Fundamental Forces* (1990) by Abdus Salam ISBN 0521371406.

3.0.3. Business Processes
3.0.3.1. A Characterisation

- By a business process we shall understand
  - a *behaviour*
  - of an enterprise, a business, an institution, a factory.

3.0.3.2. An Example

- The business processes of transportationevolves around
  - freights or passengers
  - being transported along routes
  - by a vehicle (car, train, aircraft, ship)
  - "propelled" by some locomotive force.

3.1. Intrinsics

Definition: Intrinsics.

- *By the intrinsics of a domain we shall understand*
  - *those phenomena and concepts of a domain*
  - *which are basic to any of the other facets,*
  - *with such a domain intrinsics initially covering at least one stakeholder view.*

3.1.1. Net Topology Descriptors

Instead of dealing with the entire phenomenon of a net, that is, the real, physical, geographic "thing", we can describe essentials of a net, for example how its hub and links are connected.

56. One way of abstractly modelling a net descriptor is as a map, nd, from hub identifiers to simple maps, lihis, from link identifiers to hub identifiers,
57. such that
    (a) for all hi in (the definition set of) nd it is the case that
    (b) if hi maps to lihi,
    (c) and in that link identifier to hub identifier map, li maps to hi′,
    (d) then hi′ is different from hi and
    (e) hi′ maps to an lihi′ in which li is defined and maps to hi.
    (f) And there are only such pairings.

type
56. ND′ = HI $\overrightarrow{m}$ (LI $\overrightarrow{m}$ HI)
56. ND = {|nd′:ND•wf_ND(nd′)|}
value
57. wf_ND: ND′ → **Bool**
57. wf_ND(nd) ≡
57(a). ∀ hi:HI•hi ∈ **dom** nd ⇒
57(b). **let** lihi = nd(hi) **in**
57(c). ∀ li:LI • li ∈ **dom** lihi ⇒
57(c). **let** hi′ = (nd(hi))(li) **in**
57(d). hi ≠ hi′ ∧
57(e). hi′ ∈ **dom** nd ∧ li ∈ **dom**(nd(hi′)) ∧ hi=(nd(hi′))(li)
57(f). **end end**

From a net one can construct its net descriptor:

value
conND: N → ND
conND(n) ≡
[hi↦[li↦hi′|li:LI,hi′:HI•li ∈ obs_LIs(getH(hi,n))∧{hi,hi′}=obs_HIs(getL(li,n))]|
hi:HI•hi ∈ xtrHIs(n)]

3.1.2. Link States and Link State Spaces

- We introduce the notions of
  – the state of a link,
  – the state of a hub,
  – the state space of a link and
  – the state space of a hub.
- States abstract directions of movement.
- Links are, by our previous definitions, bi-directional:
  – from one of the connected hubs to the other,
  – and vice versa.
- And hubs are multi-directional:
  – from potentially any link via the hub to potentially any link.

- Let
  - the observed hub identifiers of a link ℓ be $\{h_j, h_k\}$,
  - then link ℓ can potentially be in any one of the four link states:
  - $\{\{(h_j, h_k), (h_k, h_j)\}, \{(h_j, h_k)\}, \{(h_k, h_j)\}$ and $\{\{\}\}\}$.
- Any one particular link may
  - always remain in one and the same state,
  - or it may from time to time undergo transitions between any subset of the potential link state space.

58. Link states, $l\sigma{:}L\Sigma$, are set of pairs of hub identifiers.
59. Link state spaces are set of link states.
60. From a link one can generate the link state space of all potential link states.
61. From a link one can observe the current link state $l\sigma{:}L\Sigma$.
62. From a link one can observe the link state space $l\omega{:}L\Omega$.

type
58. LΣ = (HI×HI)**-set**
59. LΩ = LΣ**-set**

value
60. generate_full_LΣ: L → LΣ
60. generate_full_LΣ(l) ≡
60. {}∪{(hi′,hi″)|hi′,hi″:HI·hi′≠hi″∧{hi′,hi″}=obs_HIs(l)}

60. generate_LΩ: L → LΩ
60. **let** fullLσ = generate_full_LΣ(l) **in**
60. {{},∪{σ|σ:LΣ·$\sigma\subseteq$fullLσ}} **end**

61. obs_LΣ: L → LΣ
62. obs_LΩ: L → LΣ**-set**

3.1.3. Hub States and Hub State Spaces

63. Hub states, $h\sigma{:}H\Sigma$, are sets of pairs of link identifiers $((l_i, l_k))$, designating that if (l_i, l_k) is in the current hub state then movement can take place from the link designated by l_i (via hub h) to the link designated by l_k.
64. Hub state spaces are set of hub states.
65. From a hub one can generate the hub state space of all potential hub states.
66. From a hub one can observe the current hub state $h\sigma{:}H\Sigma$.
67. From a hub one can observe the hub state space $h\omega{:}H\Omega$.

type
63. HΣ = (LI×LI)-**set**
64. HΩ = HΣ-**set**

value
65. generate_full_HΣ: H → HΣ
65. generate_full_HΣ(h) ≡
65. {}∪{(li′,li″)|li′,li″:LI·{li′,li″}⊆obs_LIs(h)}

60. generate_HΩ: H → HΩ
60. **let** fullHσ = generate_full_HΣ(h) **in**
60. {{}∪{σ|σ:HΣ·σ⊆fullHσ}} **end**

66. obs_HΣ: H → HΣ
66. obs_HΩ: H → HΣ-**set**

3.1.4. State and State Space Wellformedness

68. States must be in appropriate state spaces.
69. State spaces must be subsets of all potential appropriate states.

axiom
∀ n:N,l:L,h:H · l ∈ obs_Ls(n) ∧ h ∈ obs_Hs(n) ⇒
58. obs_LΣ(l) ∈ obs_LΩ(l) ∧
59. obs_LΩ(l) ⊆ generate_full_LΣ(l) ∧
58. obs_HΣ(h) ∈ obs_HΩ(h) ∧
59. obs_HΩ(h) ⊆ generate_full_HΣ(h)

theorems:
∀ n:N,l:L,h:H · l ∈ obs_Ls(n) ∧ h ∈ obs_Hs(n) ⇒
obs_LΣ(l) ⊆ {(hi′,hi″)|hi′,hi″:H·{hi′,hi″}⊆obs_HIs(l)} ∧
obs_HΣ(h) ⊆ {(li′,li″)|li′,li″:L·{li′,li″}⊆obs_LIs(h)}

3.1.5. Concrete Types for Simple Entities

- As an alternative for, or as a step of refinement from the earlier sorts of nets, hubs and links
- one can simplify matters by concrete types for these simple entities.

70. Nets are Cartesians of sets of hubs and links.
71. A link is a Cartesian of a link identifier, a set of exactly two hub identifiers, a link state, a link state space, and a number of presently further unspecified link attributes.
72. A hub is a Cartesian of a hub identifier, a set of zero, one or more link identifiers, a hub state, a hub state space, and a number of presently further unspecified hub attributes.

type
70. N = H-**set** × L-**set**
71. L :: obs_LI:LI × obs_HIs:HI-**set** × LΣ × LΩ × LAtrs
72. H :: obs_HI:HI × obs_LIs:LI-**set** × HΣ × HΩ × HAtrs

We leave it to the reader to narrate the wellformedness constraints.

axiom

∀ (hs,ls):N · ls≠{} ⇒ **card** hs ≥ 2 ∧
  ∀ l′,l″:L · {l′,l″}⊆ls ∧ l′≠l″ ⇒ obs_LI(l′)≠obs_LI(l″) ∧
  ∀ h′,h″:H · {h′,h″}⊆hs ∧ h′≠h″ ⇒ obs_HI(h′)≠obs_HI(h″) ∧
  ∀ l:(li,his,lσ,lω,latrs):L · l ∈ ls ⇒
    card his=2 ∧ his⊆{obs_HI(h″)|h‴:H · h‴ ∈ hs} ∧
    lσ ∈ generate_full_LΣ(l) ∧
    lσ ∈ lω ⊆ generate_full_LΣ(l) ∧
  ∀ h:(hi,lis,hσ,hω,hatrs):H · h ∈ hs ⇒
    lis⊆{obs_LI(l‴)|l‴:L · l‴ ∈ ls} ∧
    hσ ∈ generate_full_HΣ(h) ∧
    hσ ∈ hω ⊆ generate_full_HΣ(h)

3.1.6. Example Hub Crossings

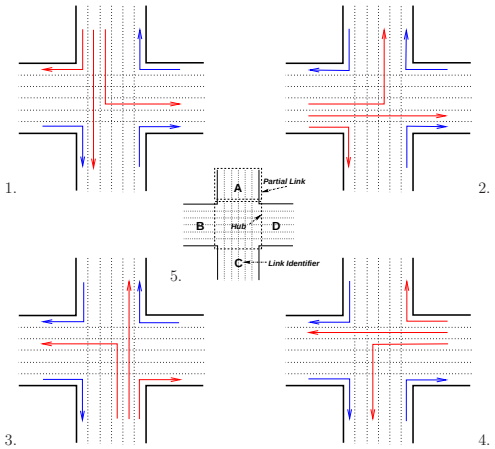

Figure 1: Four "Safe" Flows

The top left hub/link diagram (1.) thus can be claimed to depict hub state {(A, B), (A, C), (A, D), (B, C), (C, D), (D, A)}.

Photo 2 shows a semaphore which seems to be able to display all kinds of states.

Figure 2: A General Purpose Traffic Light

The point of this example is to show that a hub may take on many states, that not all hub states may be desirable (viz., lead to crossing traffic if so interpreted), and that to reach from one hub state to another one must change the state.

3.1.7. Actions Continued

73. The action change_HΣ takes a hub, h, in some state, and a desired next state, $h\sigma'$, and results in a hub, h', which
    (a) has the same hub identifier as h,
        is connected to the same links as h,
        has the same hub state space as h,
        has the same attributes (names and values) as h,
    (b) but whose state may have changed.

73(b). The new state of h' ought be $h\sigma'$, but electro-mechanical or other failures in setting the state may set the new state to any state of the potential states of h (i.e., h'), not just to any state in the hub state space of h.

value
73. change_HΣ: H × HΣ → H
73. change_HΣ((hi,lis,hσ,hω,hatrs),hσ′) ≡
73(b). **let** hσ‴ ∈ generate_full_HΣs **in**
73(a). (hi,lis,hσ‴,hω,hatrs) **end**

- Had we specified that the resulting state must be $h\sigma'$
- then we had prescribed a requirements to a change operation.
- As it is now we have described a domain phenomenon, namely that operations may fail.

3.2. Support Technologies

Definition: Support Technology. *By a support technology we understand*

- *a facet of a domain,*
- *one which reflects its (current) dependency on*
  - *human,*
  - *mechanical,*
  - *electro-mechanical,*
  - *electronic and/or*
  - *other technologies*

(i.e., tools) in order to carry out its business processes.

3.2.1. Traffic Signals

A traffic signal represents a technology in support of visualising hub states and in effecting state changes.

74. A hub state is now modelled as a triple: the link identifier l_i ("coming from"), a colour (**red**, **yellow**, and **green**), and another the link identifier l_j ("going to").
75. Signalling is now a sequence of one or more pairs of next hub states and time intervals:

$$< (h\sigma_1, ti_1), (h\sigma_2, ti_2), ..., (h\sigma_{n-1}, ti_{n-1}), (h\sigma_n, ti_n) >, n > 0$$

- The idea of a signalling is
  - to first change the designated hub to state $h\sigma_1$,
  - then wait ti_1 time units,
  - then set the designated hub to state $h\sigma_2$,
  - then wait ti_2 time units,
  - etcetera, ending with final state σ_n
  - and a (supposedly) long time interval ti_n
  - before any decisions are to be made as to another signalling.
- The set of hub states $\{h\sigma_1, h\sigma_2, ..., h\sigma_{n-1}\}$ of

$$< (h\sigma_1, ti_1), (h\sigma_2, ti_2), ..., (h\sigma_{n-1}, ti_{n-1}), (h\sigma_n, ti_n) >, n > 0$$

  are called intermediate states.
- Their purpose is to secure an orderly vehicle-wise safe signal transitions from **red** to **green** etc.

76. A street signal (a semaphore) is now abstracted as a map from pairs of hub states to signalling sequences.

    The idea is that given a hub one can observe its semaphore, and given the state, $h\sigma$ (not in the above set), of the hub "to be signalled" and the state $h\sigma_n$ into which that hub is to be signalled "one looks up" under that pair in the semaphore and obtains the desired signalling.

type
74. HΣ = LI × Colour × LI
74. Colour == red | yellow | green
75. Signalling = (HΣ × TI)*
75. TI
76. Sempahore = (HΣ×HΣ) $\xrightarrow{m}$ Signalling

value
76. obs_Semaphore: H → Sempahore

77. A hub semaphore, sema, contains only such hub states as are observed in the hub state space.
    (a) Let hsps be the set of "from/to" hub state pairs in sema.
    (b) Then hs is the set of all hub states mentioned in hsps.
    (c) To hs join all the hub states mentioned in any signalling, sg, of sema.

77. hub_state_space: Sempahore → HΣ-**set**
77. hub_state_space(sema) ≡
77(a). **let** hsps={hsp|hsp:(HΣ×HΣ)·hsp ∈ **dom** sema} **in**
77(b). **let** hs={hσ′,hσ″|hσ′,hσ″:HΣ·(hσ′,hσ″)∈ hsps} **in**
77(c). hs ∪ ∪{{hσ|(hσ,ti):(HΣ×TI)·(hσ,ti)∈ **elems** sg}|sg:Signalling·sg ∈
77. **end end**

axiom
77. ∀ h:H · ∪ obs_HΩ(h) = hub_state_space(obs_Semaphore(h))

3.2.2. Traffic "Control"

78. Given two hub states, $h\sigma_{\text{init}}$ and $h\sigma_{\text{end}}$, where $h\sigma_{\text{init}}$ designates a present hub state and $h\sigma_{\text{end}}$ designates a desired next hub state after signalling.

79. Now signalling is a sequence of one or more successful hub state changes.

value
78. signalling: HΣ × HΣ → H → H
79. signalling(hσ$_{\text{init}}$,hσ$_{\text{end}}$)(h) ≡
79. **let** sema = obs_Semaphore(h) **in**
79. **let** sg = sema(hσ$_{\text{init}}$,hσ$_{\text{end}}$) **in**
79. signal_sequence(sg)(h) **end end**
79. **pre** (hσ$_{\text{init}}$,hσ$_{\text{end}}$) ∈ **dom** obs_Semaphore(h)

79. signal_sequence(⟨⟩)(h) ≡ h
79. signal_sequence(⟨(hσ,ti)⟩⌢sg)(h) ≡
79. **let** hσ′ = change_HΣ(h)(hσ) **in**
79. **if** hσ′ ≠ hσ **then chaos**
79. **else wait**(ti); signal_sequence(sg)(h) **end end**

- If a desired hub state change fails (**chaos**) then we do not define the outcome of signalling.

3.3. Rules and Regulations

Definition: Rule. *A rule stipulates a regulating principle.*

- *In the context of modelling domain rules we shall understand a domain rule*
  - *as some* ***text***
  - *whose meaning is a* ***predicate***
  - *over a pair of suitably chosen domain* ***state****s.*
- *We may assume that*
  - *a domain* ***action*** *or a domain* ***event***
  - *takes place in the* ***first*** *of these* ***state****s and*
  - *results in the* ***second*** *of these* ***state****s.*
- *If the* ***predicate*** *is true*
  - *then we say that the rule has been* ***obeyed****,*

  - *otherwise that it has been* ***violated****.*

Usually a domain rule is paired with a possibly remedying regulation.

Definition: Regulation.

- *A regulation stipulates that*
  - *an* ***action*** *be taken*
  - *in order to remedy a previous action which violated a* ***rule****.*
- *That is,*
  - *a regulation is some* ***text***
  - *which designates a possibly composite* ***action****,*
  - *that is, a* ***state-to-state change***
  - *which ostensibly results in a state*
  - *in which the* ***rule****, "attached" to the regulation,* ***now holds****.*

3.3.1. Vehicles

80. Vehicles are further undefined quantities except that
    (a) vehicles have unique identifiers,
    (b) vehicles are either positioned
        i. at/in hubs
        ii. or on links, in some fractional (non-zero) distance from a hub toward the connecting hub.
81. From a net (sort) one can observe all the vehicles of the net.[1]
82. No two vehicles so observed have the same identifier.

[1] Thus a concrete net type, in addition to hubs and links (now) also contains vehicles.

type
80. V
80(a). VI
80(b). VP = HP | LP
80((b))i. HP == atH(hi:HI)
80((b))ii. LP == onL(li:LI,fhi:HI,f:F,thi:HI)
80((b))ii. F = {|f:F•0<f<1|}
value
80(a). obs_VI: V → VI
80(b). obs_VP: V → VP
81. obs_Vs: N → V-**set**
axiom
82. ∀ v:V • v ∈ obs_Vs(n) ⇒
82. ∃ onL(li,fhi,f,thi):VP • onL(li,fhi,f,thi)=obs_VP(v) ⇒
82. ∃ l:L•l ∈ obs_Ls(n)∧li=obs_LI(l)∧{fhi,thi}=obs_HIs(l) ∨
82. ∃ atH(hi):VP • atH(hi)=obs_VP(v) ⇒
82. ∃ h:H•h ∈ obs_Hs(n)∧hi=obs_HI(h)

3.3.2. Traffic

83. By traffic we understand a continuous function from time to a pair of nets and position of vehicles.
84. By time we understand a dense set of points with dense and points being mathematical concepts [wayne.d.blizard.90,J.van.Benthem.Logic.Time91].

type
83. TF = T → (sel_net:N × sel_veh_pos:(V $\overrightarrow{m}$ VP))
84. T

3.3.2.1. Wellformedness of Traffic

- Expressing the wellformedness of traffic is not a simple matter.
- We shall approach this task in a number of "small steps".

3.3.2.1.1. • *Static Wellformedness*•

85. We define a predicate over vehicle positions.
   (a) Every vehicle in the traffic has a proper position on the net, either at a hub or along a link.
   (b) No two vehicles of the traffic can occupy exactly the same link position. (That is, the link positions onL(li,hi,f,hi′) and onL(li,hi,f′,hi′) must have the two fractions (f, f') differ – be it ever so "minutely").

We first define two auxiliary functions:[2]

value
obs_HIs: N → HI-**set**
obs_HIs(n) ≡ {obs_HI(h)|h:H•h ∈ obs_Hs(n)}
obs_LIs: N → LI-**set**
obs_LIs(n) ≡ {obs_LI(h)|l:L•l ∈ obs_Ls(n)}

[2] They really ought to have been defined much earlier!

85. proper_vehicle_positions: TF → **Bool**
85. proper_vehicle_positions(tf) ≡
85. ∀ t:T • t ∈ 𝔻𝕆𝕄𝔸𝕀𝕟 tf •
85. **let** (n,vps) = tf(t) **in**
85(a). ∀ v:V•v ∈ **dom** vp•is_net_position(vps(v))(n)
85(b). ∀ v′:V•v′ ∈ **dom** vp ∧ v≠v′⇒diff_net_pos(vps(v),vps(v′))
85. **end**
85(a). is_net_position: VP → N → **Bool**
85(a). is_net_position(vp)(n) ≡
85(a). **case** vp **of**
85(a). atH(hi) → hi ∈ obs_HIs(n),
85(a). onL(li,fhi,f,thi) → li ∈ obs_LIs(n)∧{fhi,thi}⊆obs_HIs(n)
85(a). **end**
85(b). diff_net_pos: VP × VP → **Bool**
85(b). diff_net_pos(vp,vp′) ≡
85(b). **case** (vp,vp′) **of**
85(b). (atH(hi),atH(hi)) → **true**,
85(b). (onL(li,fhi,f,thi),onL(li,fhi,f′,thi)) → f≠f′,
85(b). _ → **true**
85(b). **end**

3.3.2.1.2. • *Dynamic Wellformedness*•

86. Vehicles, when moving, move monotonically, that is,
    (a) if a vehicle, at some time, t, is at a link position onL(li,hi,f,hi′) where f is not infinitesimally close to 1, then that vehicle will, at some later time t', infinitesimally close to t, be at link position onL(li,hi,f′,hi′) where f' is infinitesimally close to f;
    (b) if the vehicle, at some time, t, is at a link position onL(li,hi,f,hi′) where f is indeed infinitesimally close to 1, then that vehicle will, at some infinitesimally later time t', be at hub position atH(hi′);
    (c) and if the vehicle, at some time, t, is at a hub position atHP(hi) then the vehicle will at some infinitesimally later time t' either be at hub position atHP(hi) or at some link position onL(li,hi,f,hi′) where f is infinitesimally close to 0.


```
value
86. monotonic: TF → Bool
86. monotonic(tf) ≡
86.   ∀ t,t′:T · {t,t′}⊆DOMAIN tf ·
86.     let (n,vps) = tf(t),(n′,vps′)=tf(t′) in
86.     INFINITESIMALLY CLOSE (t,t′)∧t<t′⇒
86.       ∀ v:V·v ∈ dom vps ∩ dom vps′ ·
86.         case (vps(v),vps′(v)) of
86(a).           (onL(li,fhi,f,thi),onL(li,fhi,f′,thi)) →
86(a).               f<f′ ∧ INFINITESIMALLY CLOSE (f,f′),
86(b).           (onL(li,fhi,f,thi),atH(thi)) →
86(b).               INFINITESIMALLY CLOSE (f,1),
86(c).           (atH(hi),atH(hi)) → true,
86(c).           (atH(hi),onL(li,hi,f,thi)) →
86(c).               INFINITESIMALLY CLOSE (0,f),
86.            _ → true
86.     end   end
```


87. If a vehicle is (has been) moving along a link l_i and is now,
    - at time t, at position onL(l_i, h_j, f, h_k), that is, moving from h_j to h_k,
    - then it cannot at a subsequent, infinitesimally close time, t', be at a position
    - onL(l_i, h_k, f', h_j), that is, moving in the opposite direction, h_k to h_j.


```
value
87.  God_does_not_play_dice[3]: TF → Bool
87.  God_does_not_play_dice(tf) ≡
87.    ∀ t,t′:T · {t,t′}⊆DOMAIN tf ∧ t<t′ ∧ INFINITESIMALLY CLOSE (t,t′)⇒
87.      let (n,vps) = tf(t),(n′,vps′)=tf(t′) in
87.      ∀ v:V · v ∈ dom vps ∩ dom vps′ ⇒
87.        case (vps(v),vps′(v)) of
87.          (onL(li,fhi,_,thi),onL(li,thi,_,fhi))→false,
87.          _ → true
87.      end end
```

[3] Albert Einstein: "I, at any rate, am convinced that He does not throw dice." Letter to Max Born (4 December 1926); The Born-Einstein Letters (translated by Irene Born) (Walker and Company, New York, 1971) ISBN 0-8027-0326-7. Reflects Einstein's view of Quantum Mechanics at the time.

88. If a vehicle is (has been) moving along and has,
    - at time t, been at some position p, and
    - at time t', later than t, is at some position p',
    - then it must at all times t'' between t and t' have been somewhere on the net.

```
value
88.  no_ghost_vehicles: TF → Bool
88.  no_ghost_vehicles(tf) ≡
88.    ∀ t,t':T · {t,t'}⊆DOMAIN tf ∧ t<t' ⇒
88.      let (n,vps) = tf(t),(n',vps')=tf(t') in
88.      ∀ v:V·v ∈ dom vps ∩ dom vps' ⇒
88.        ∀ t'':T · t<t''<t' ⇒
88.          let (n'',vps'') = tf(t'') in v ∈ dom vps'' end
88.      end
```


3.3.3. Traffic Rules (I of II)

89. A vehicle must not move from a hub, h_i, into a link ℓ (from hub (identified by) h_i to hub (identified by) h_j) which is closed in direction (h_i, h_j), that is, where (h_i, h_j) is not in the current state of link.

rule:

```
89.  ∀ tf:TF,t:T · t ∈ DOMAIN(tf) ⇒
89.    let (n,tp) = tf(t) in
89.    ∀ v:V · v ∈ dom tp ⇒
89.      case tp(v) of
89.        atH(hi) →
89.          let t':T · t'>t ∧ t' ∈ DOMAIN(tr') ∧ INFINITESIMALLY_CLOSE(t,t') in
89.          let (n',tp') = tf(t') in
89.          ∃ li:LI,hi':HI,f:F,hi'':HI ·
89.            hi'=hi ∧ INFINITEIMALLY_CLOSE(f,0) ∧
89.            tp'(v) = onL(li,hi',f',hi'') ∧(hi,hi'') ∉ obs_LΣ(getL(li,n'))
89.        _ → ...
89.    end end end end
```


3.3.4. Another Traffic Regulator

- We present an abstraction of a more conventional traffic signal than modelled in Items 74 on page 78 to 77 on page 81.

90. A traffic signal now simply shows an entry permit: either **red**, **yellow** or **green** at the hub when "leaving" any link, i.e., at the entry to a hub from any link.

```
type
90.  EP == red | yellow | green
90.  HΣ = LI ~m~> EP
axiom
90.  ∀ h:H · obs_LIs(h)=dom obs_HΣ(h)
```

- We leave it to the reader to express a constraint over hub state spaces as to how there must be hub states such that entry from any link is possible.

3.3.5. Traffic Rules (II of II)

91. Vehicles must not enter a hub if entry permission is not **green**.

rule:

```
91.  ∀ tf:TF,t:T : t ∈ DOMAIN(tf) ⇒
91.    let (n,vps) = tf(t) in
91.    ∀ v:V · v ∈ dom vps ⇒
91.      case vps(v) of
91.        onL(li,hi,f,hi') →
91.          INFINITESIMALLY_CLOSE(f,1) ∧
91.          let hσ = obs_HΣ(getH(hi',n)),
91.              t':T · t'>t ∧ INFINITESIMALLY_CLOSE(t,t') in
91.          let (n',vps') = vps(t') in
91.          hσ(li) ≠ green ∧ vps'(v) ≠ atH(hi')  assert: vps'(v) = onL(li,hi,f,hi')
91.          end end
91.        _ → ...
91.    end end
```

End of Lecture 3: DOMAINS: Intrinsics – Rules & Regulations

B Slide Table-of-Contents

Contents