DOMAIN SCIENCE & ENGINEERING

The TU Wien Lectures, Fall 2022

Dines Bjørner

Technical University of Denmark

The Triptych Dogma

In order to *specify* **software**,
we must understand its requirements.

In order to *prescribe* **requirements**
we must understand the **domain**.

So we must **study, analyse** and **describe** domains.

Day #2: External Qualities, Analysis (I)

- This is the first, properly systematic treatment lecture of some of the
  - principles,
  - procedures,
  - techniques and
  - tools

  of the *Domain Engineering* method.
- In this lecture we cover those of
  - analysing endurants.

- This, the present chapter
  - is based on Chapter 4 of [18].
  - You may wish to study that chapter for more detail.

4.1 Universe of Discourse

Definition 28 . *Universe of Discourse, UoD:*

- By a *universe of discourse* we shall understand
  - the same as the *domain of interest*,
  - that is, the *domain* to be *analysed & described* ■

4.1.1 Identification

- The **first task** of a domain analyser cum describer
  - is to settle upon the domain to be analysed and described.
  - That domain has first to be given a *name*.

4.1.2 Naming

- A **first decision** is to give a name to the overall domain sort,
  * that is, the type of the domain seen as an endurant,
  * with that sort, or type, name
    being freely chosen by the analyser cum describer –
  * with no such sort names having been chosen so far !

4.1.3 Examples

- Examples of UoDs

- *railways* [2, 22, 4],
- *"The Market"* [3],
- *container shipping* [5],
- *Web systems* [6],
- *stock exchange* [7],
- *oil pipelines* [8],
- *credit card systems* [10],
- *weather information* [11],
- *swarms of drones* [12],
- *document systems* [13],
- *container terminals* [14],
- *retail systems* [16],
- *assembly plants* [17],
- *waterway systems* [19],
- *shipping* [20],
- *urban planning* [24].

4.1.4 Sketching

- The **second task** of a domain analyser cum describer is to develop a *rough sketch narrative* of the domain.
- The rough-sketching of [what] a domain [is,] is not a trivial matter.
  - It is not done by a committee !
  - It usually requires repeated "trial sketches".
  - To carry it out, i.e., the sketching, normally requires a combination of
    - physical visits to domain examples, if possible;
    - talking with domain professionals, at all levels; and
    - reading relevant literature.
    - It also includes searching the Internet for information.
- We shall show an example next.

Example 21 . Sketch of a Road Transport System UoD:

- The road transport system that we have in mind consists of
  - a road net and
  - a set of automobiles (private, trucks, buses, etc.)
  - such that the road net serves to convey automobiles.
- We consider the road net to consist of
  - hubs, i.e., street intersections, including street segment connection points, and
  - links, i.e., street segments between adjacent hubs[1] ■

[1] This "rough" narrative fails to narrate ...

4.1.5 Universe of Discourse Description

The general universe of discourse, i.e., domain, description prompt can be expressed as follows:

Domain Description Prompt 1 . *calc_Universe_of_Discourse:*

calc_Universe_of_Discourse describer

❝ Naming:
  type UoD
Rough Sketch:
  Text ❞

The above ❝ RSL-**Text** ❞ expresses that the `calc_Universe_of_Discourse()` domain describer generates RSL-**Text**.

Here is another example rough sketch:

Example 22 . A Rough Sketch Domain Description:

- The example is that of the production of rum, say of a **Rum Production** domain.
- From

1. the sowing, watering, and tending to of sugar cane plants;
2. via the "burning" of these prior to harvest;
3. the harvest;
4. the collection of harvest from sugar cane fields to
5. the chopping, crushing, (and sometimes repeated) boiling, cooling and centrifuging of sugar cane when making sugar and molasses (into A, B, and low grade batches);
6. the fermentation, with water and yeast, producing a 'wash';

7. the (pot still or column still) distilling of the wash into rum;
8. the aging of rum in oak barrels;
9. the charcoal filtration of rum;
10. the blending of rum;
11. the bottling of rum;
12. the preparation of cases of rum for sales/export; and
13. the transportation away from the rum distiller of the rum ■

Some comments on this example:

- Each of the enumerated items above is phrased in terms of perdurants.
  - Behind each such perdurant lies some endurant.
  - That is, in English, *"every noun can be verbed", and vice-versa.*
  - So we anticipate the transcendence, from endurants to perdurants.

• • •

Method Principle 1 . *From the "Overall" to The Details:*

- *Our first principle, as the first task in any new domain modelling project, is*
  - *to "focus" on the "overall",*
  - *that is, on the "entire",*
  - *generic domain* ■

4.2 Entities

A core concept of domain modelling is that of an *entity*.

Definition 29 . ***Entity:***

- By an ***entity*** we shall understand a ***phenomenon***, i.e., something
  - that can be ***observe***d, i.e., be
    - seen or touched by humans,
    - *or* that can be ***conceive***d
    - as an ***abstraction*** of an entity;
  - alternatively,
    - a phenomenon is an entity, *if it exists, it is* ***"being"***,
    - *it is that which makes a* ***"thing"*** *what it is:*
      essence, essential nature.
- If a phenomenon cannot be so **observed and described** then it is not en entity ■

Analysis Predicate Prompt 1 . `is_entity:`

- *The domain analyser analyses "things" (θ) into either entities or non-entities.*
- *The method provides the* ***domain analysis prompt****:*
  - `is_entity` – *where* `is_entity(θ)` *holds if θ is an entity* ■ [2]

- `is_entity` is said to be
  - a *prerequisite prompt*
  - for all other prompts.
- `is_entity` is a method tool.

[2] ■ marks the end of an analysis prompt definition.

On Analysis Prompts: The `is_entity` predicate function represents the first of a number of analysis prompts.

- They are "applied" by the domain analyser to phenomena of domains.
- They yield truth values, true or false, "left" in the mind of the domain analyser ■

• • •

- We have just shown how the `is_entity` predicate prompt can be applied to a universe of discourse.
- From now on we shall see prompts being applicable to successively more analysed entities.
- Figure 4.1 [Page 64][3] diagrams a **domain description ontology** of entities.
- That ontology indicates the sub-classes of endurants for which we shall motivate and for which we shall introduce
  - prompts,
  - predicates and
  - functions.

[3] This ontology was first shown, as Fig. **??** [Page **??**]

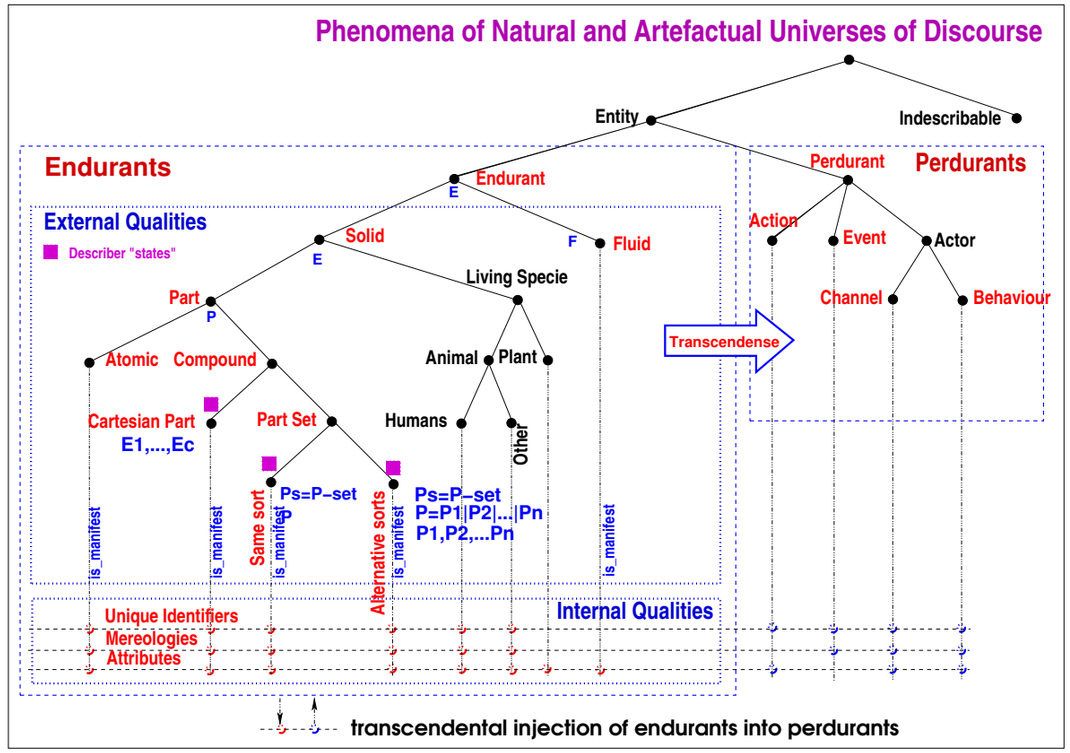

Figure 4.1: The Upper Ontology

- The present chapter shall focus only
  - on the external qualities,
  - that is, on the "contents" of the leftmost dotted box.

• • •

Method Principle 2 . ***Justifying Analysis along Philosophical Lines:***

- *The concept of* entities *as a main focal point*
  - *is justified in* Kai Sørlander*'s philosophy.*
  - *Entities are in that philosophy referred to as* primary objects.
  - *They are the ones about which we express predicates* ■

4.3 Endurants and Perdurants

Method Principle 3 . *Separation of Endurants and Perdurants:*

- *As we shall see in this primer,*
  the domain analysis & description method calls
  for the separation of
  - *first considering*
    - *the careful analysis & description*
    - *of endurants,*
  - *then considering*
    - *perdurants.*
- *This principle is based on*
  - *the transcendental deduction*
  - *of the latter from the former* ■

4.3.1 Endurants

Definition 30 . *Endurant:*

- By an ***endurant***, to repeat, we shall understand an entity
  - that can be observed, or conceived and described,
    as a "complete thing"
    at no matter which given snapshot of time;
  - alternatively an entity is endurant
    if it is capable of *enduring*, that is *persist*, *"hold out"*.

Were we to "freeze" time

- we would still be able to observe the entire endurant ■

Example 23 . Natural and Artefactual Endurants:

Geography Endurants:

- fields,
- lakes,
- forests,
- mountains,
- meadows,
- rivers,
- hills,
- et cetera.

Railway Track Endurants:

- a railway track,
- its net,
- its individual tracks,
- switch points,
- trains,
- their individual locomotives,
- signals,
- et cetera.

Road Transport System Endurants:

- the transport system,
- its road net aggregate and the aggregate of automobiles,
- the set of links (road segments) and hubs (road intersections) of the road net aggregate,
- these links and hubs, and
- the automobiles.

Analysis Predicate Prompt 2 . `is_endurant:`

- *The domain analyser analyses an entity, ϕ,*
  into an endurant
  as prompted by the ***domain analysis prompt****:*
  - `is_endurant` – *ϕ is an endurant if* `is_endurant(`ϕ`)` *holds* ■

- `is_entity` is a *prerequisite prompt* for `is_endurant`.
- `is_endurant` is a method tool.

4.3.2 Perdurants

Definition 31 . *Perdurant:*

- By a ***perdurant*** we shall understand an entity
  - for which only a fragment exists
    if we look at or touch them
    at any given snapshot in time.
  - Were we to freeze time we would only see or touch
    a fragment of the perdurant ■

Example 24 . Perdurants:

Geography Perdurants:

- *the continuous changing of the weather (meteorology);*
- *the erosion of coastlines;*
- *the rising of some land area and the "sinking" of other land area;*
- *volcanic eruptions;*
- *earthquakes;*
- *et cetera.*

Railway System Perdurants:

- *the ride of a train from one railway station to another; and*
- *the stop of a train at a railway station from some arrival time to some departure time* ■

Analysis Predicate Prompt 3 . `is_perdurant:`

- *The domain analyser analyses an entity e*
  into perdurants
  as prompted by the **domain analysis prompt**:
  - `is_perdurant`– *e is a perdurant if* `is_perdurant(e)` *holds.*
- `is_entity` *is a prerequisite prompt for* `is_perdurant` ■
- `is_perdurant` is a method tool.

• • •

- We repeat method principle 3 on Slide 67:

Method Principle 4. ***Separation of Endurants and Perdurants:***

- *First domain analyse & describe endurants;*
- *then domain analyse & describe perdurants* ■

4.4 Solids and Fluids

- For *pragmatic* reasons we distinguish between
  - solids and
  - fluids.

Method Principle 5 . *Abstraction, I:*

- *The principle of abstraction is now brought into "full play":*
  - *In analysing & describing entities the domain analyser cum describer*
  - *is "free" to not consider all facets of entities,*
  - *that is, to abstract.*

4.4.1 Solids

Definition 32 . *Solid Endurant::*

- By a ***solid endurant*** we shall understand an endurant
- which is
  - separate,
  - individual or
  - distinct in form or concept,
- or, rephrasing:
  - a body
  - or magnitude

  of three-dimensions,
  having length, breadth and thickness [34, Vol. II, pg. 2046] ■

Analysis Predicate Prompt 4. `is_solid:`

- *The domain analyser analyses endurants, e,*
  into solid entities
  as prompted by the ***domain analysis prompt****:*
  - `is_solid` – *e is solid if* `is_solid(e)` *holds* ■

- To simplify matters we shall allow
  separate elements of a solid endurant to be fluid !
- That is, a solid endurant, i.e., a part,
  may be conjoined with a fluid endurant, a fluid.
- `is_solid` is a method tool.

Example 25 . Artefactual Solid Endurants:

- The individual endurants of the above example of **railway system** endurants, Example 23 on Slide 69, were all solid.
- Here are examples of solid endurants of **pipeline systems**.
  - A pipeline and
  - its individual units:
    - ∗ wells,
    - ∗ pipes,
    - ∗ valves,
    - ∗ pumps,
    - ∗ forks,
    - ∗ joins,
    - ∗ regulator, and
    - ∗ sinks.

4.4.2 Fluids

Definition 33 . *Fluid Endurant:*

- By a ***fluid endurant*** we shall understand an endurant which is
  - prolonged, without interruption, in an unbroken series or pattern;

  or, rephrasing:

  - a substance (liquid, gas or plasma) having the property of flowing, consisting of particles that move among themselves [34, Vol. I, pg. 774] ■

Analysis Predicate Prompt 5 . `is_fluid:`

- *The domain analyser analyses endurants e into fluid entities as prompted by the* **domain analysis prompt**:
  - `is_fluid` – *e is fluid if* `is_fluid(e)` *holds* ■
- `is_fluid` is a method tool.

- Fluids are otherwise
  - liquid, or
  - gaseous, or
  - plasmatic, or
  - granular[4], or
  - plant products[5],
  - et cetera.

[4] This is a purely pragmatic decision.
"Of course" sand, gravel, soil, etc., are not fluids,
but for our modelling purposes it is convenient
to "compartmentalise" them as fluids !

[5] i.e., chopped sugar cane, threshed, or otherwise. See footnote 4.

Example 26 . Fluids:

- Specific examples of fluids are:
  - water, oil, gas, compressed air, etc.
- A container, which we consider a solid endurant,
  - may be *conjoined* with another, a fluid,
  - like a gas pipeline unit may "contain" gas ■

4.5 Parts and Living Species

- We analyse endurants into either of two kinds:
  - *parts* and
  - *living species*.
- The distinction between *parts* and *living species* is motivated in Kai Sørlander's Philosphy [46, 47, 48, 49, 50].

4.5.1 Parts

Definition 34. *Parts:*

- By a *part* we shall understand
  - a solid endurant existing in time and
  - subject to laws of physics,
  - including the *causality principle* and
  - *gravitational pull*[6] ■

[6] This characterisation is the result of our study of relations between philosophy and computing science, notably influenced by Kai Sørlander's Philosphy [46, 47, 48, 49, 50]

Analysis Predicate Prompt 6 . `is_part:`

- *The domain analyser analyses "things" (e) into part.*
- *The method can thus be said to provide the* ***domain analysis prompt****:*
  - `is_part` – *where* `is_part(e)` *holds if e is a part* ■

`is_part` is a method tool.

- *Parts* are
  - either *natural* parts, or are
  - *artefactual* parts, i.e. man-made.
- Natural and man-made parts are either
  - *atomic* or
  - *compound*.

4.5.1.1 Atomic Parts

- The term 'atomic' is, perhaps, misleading.
  - It is not used in order to refer to nuclear physics.
  - It is, however, chosen in relation to the notion of *atomism:*
    - *a doctrine that the physical or physical and mental universe*
    - *is composed of simple indivisible minute particles* [Merriam Webster].

Definition 35 . ***Atomic Part, II:***

- By an *atomic part* we shall understand a part
  - which the domain analyser considers to be indivisible
  - in the sense of not meaningfully,
  - for the purposes of the domain under consideration,
  - that is, to not meaningfully consist of sub-parts ■

Example 27 . Atomic Parts:

- We refer to Example 25 on Slide 79: pipeline systems. The
  - wells,
  - pumps,
  - valves,
  - pipes,
  - forks,
  - joins and
  - sinks

can be considered atomic ■

Analysis Predicate Prompt 7 . `is_atomic:`

- *The domain analyser analyses "things" (e) into atomic part.*
- *The method can thus be said to provide the* **domain analysis prompt**:
  - `is_atomic` – *where* `is_atomic(e)` *holds if e is an atomic part* ■

`is_atomic` is a method tool.

4.5.1.2 Compound Parts, II

- We, pragmatically, distinguish between
  - Cartesian-product-, and
  - set-

  oriented parts.
- That is, if Cartesian-product-oriented,
  to consist of two or more
  distinctly sort-named endurants (solids or fluids),
- or, if set-oriented,
  to consist of an indefinite number of zero, one or more
  identically sort-named parts.

Definition 36 . *Compound Part:*

- *Compound part*s are those which are
  - either Cartesian-product-
  - or are set-
- oriented parts ■

Analysis Predicate Prompt 8 . `is_compound:`

- *The domain analyser analyses "things" (e) into compound part.*
- *The method can thus be said to provide the* **domain analysis prompt**:
  - `is_compound` – *where* `is_compound(e)` *holds if e is a compound part* ■

`is_compound` is a method tool.

4.5.1.2.1 Cartesian Parts

Definition 37 . ***Cartesian Part:***

- ***Cartesian part***s are those (compound parts)
  - which consists of an "indefinite number"
  - of two or more parts
  - of distinctly named sorts ■

Example 28 . Cartesian Automobiles:

- We refer to Example 23 on Slide 70, the **transport system** sub-example.
- We there viewed (hubs, links and) automobiles as atomic parts.
- From another point of view we shall here understand automobiles as Cartesian parts:
  - the engine train,
  - the chassis,
  - the car body,
  - four doors (left front, left rear, right front, right rear), and
  - the wheels.
- These may again be considered Cartesian parts.

Analysis Predicate Prompt 9 . `is_Cartesian:`

- *The domain analyser analyses "things" (e) into Cartesian part.*
- *The method can thus be said to provide the* **domain analysis prompt**:
  - `is_Cartesian` – *where* `is_Cartesian(e)` *holds if e is a Cartesian part* ■

`is_Cartesian` is a method tool.

4.5.1.2.2 Calculating Cartesian Part Sorts

- The above analysis amounts to the analyser
  - first "applying" the *domain analysis* prompt
  - `is_compound`(e) to a solid endurant, e,
  - where we now assume that the obtained truth value is **true**.
  - Let us assume that endurants e:E consist of sub-endurants of sorts
    $\{E_1, E_2, \ldots, E_m\}$.
  - Since we cannot automatically guarantee that our domain descriptions secure that
  - E and each E_i $(1 \leq i \leq m)$
  - denotes disjoint sets of entities **we must prove so!**

• • •

On Determination Functions:

- Determination functions
  - apply to compound parts
  - and yield their sub-parts and the sorts of these.
- *That is,*
  - *we observe the domain*
  - *and our observation results*
  - *in a focus on a subset of that domain*
  - *and sort information about that subset.*

An RSL Extension:

- The determine_··· functions below are expressed as follows:

  value determine_···(e) **as** (parts,sorts)

  – where we focus here on the sorts clause.
  – Typically that clause is of the form ηA,ηB,...,ηC.[7]
  – That is, a "pattern" of sort names: A,B,...,C.
- These sort names are provided by the domain analyser cum describer.
- They are chosen as "full names", or as mnemonics, to capture an essence of the (to be) described sort.
- Repeated invocations, by the domain analyser cum describer, of these (...,sorts) analysis functions normally lead to new sort names distinct from previously chosen such names.

4.5.1.2.2.1 Cartesian Part Determination

Observer Function Prompt 1 . *determine_Cartesian_parts:*

- *The domain analyser analyses a part into a Cartesian part.*
- *The method provides the* ***domain observer prompt****:*
  - `determine_Cartesian_parts` — *it directs the domain analyser to determine the definite number of values and corresponding distinct sorts of the part.*

    value

    determine_Cartesian_parts: E → (E1×E2×...×En) × (ηE1×ηE2×...×ηEn)[8]

    determine_Cartesian_parts(e) **as** ((e1,...,en),(ηE1,...,ηEn))

    where by E, Ei we mean endurants, i.e., part values, and by ηEi we mean the names of the corresponding types.

`determine_Cartesian_parts` is a method tool.

[7] ηA,ηB,...,ηC are **names** of types. $\eta\theta$ is the type of all type names !
[8] The ordering, ((e1,...,en),(ηE1,...,ηEn)), is pairwise arbitrary.

On Calculate Prompts:

- Calculation prompts
  - apply to compound parts: Cartesians and sets,
  - and yield an RSL-**Text** description.

Domain Description Prompt 2 . `calc_Cartesian_parts`:

- *If* `is_Cartesian`*(e) holds, then the analyser "applies" the* **domain description prompt**
  - `calc_Cartesian_parts`*(e)*

  resulting in the analyser writing down
  the endurant sorts and endurant sort observers
  domain description text
  according to the following schema:

calc_Cartesian_parts describer

let (_[9],(ηE$_1$,...,ηE$_m$)) = determine_Cartesian_parts_sorts(e)[10] **in**
❝ **Narration:**
[s] ... narrative text on sorts ...
[o] ... narrative text on sort observers ...
[p] ... narrative text on proof obligations ...
Formalisation:
type
[s] E$_1$, ❞...❝, E$_m$
value
[o] obs_E$_1$: E → E$_1$, ❞...❝, obs_E$_m$: E → E$_m$
proof obligation
[p] [Disjointness of endurant sorts]❞
end

`calc_Cartesian_parts` is a method tool.

Elaboration 1 ***Type, Values and Type Names:***

- *Note the use of quotes above.*
- *Please observe that when we write* obs_E *then* obs_E *is the name of a function.*
- *The* E*, when juxtaposed to* obs_ *is now a name* ■

Observer Function Prompt 2 . ***type_name, type_of:***
The definition of **type_name, type_of**
implies the informal definition of

$$\mathsf{obs\_E}_i(\mathsf{e})=\mathsf{e}_i \equiv \mathsf{type\_name}(\mathsf{e}_i)=\text{“}\,\mathsf{E}_i\,\text{”}\wedge$$
$$\mathsf{type\_of}(\mathsf{e}_i) \equiv \mathsf{E}_i \wedge$$
$$\mathsf{is\_E}_i(\mathsf{e}_i)$$

[9] The use of the underscore, _ , shall inform the reader that there is no need, here, for naming a value.

[10] For `determine_composite_parts` see Sect. 4.5.1.2.2.1 on Slide 100

Example 29 . A Road Transport System Domain: Cartesians:

14. There is the *universe of discourse*, RTS.

It is composed from

15. a *road net*, RN, and
16. an *aggregate of automobiles*, AA.

type
14 RTS
15 RN
16 AA
value
15 obs_RN: RTS → RN
16 obs_AA: RTS → AA ■

- We continue the analysis & description of "our" road transport system:

17. The road net consists of
    (a) an aggregate, AH, of hubs and
    (b) an aggregate, AL, of links.

type
17a AH
17b AL
value
17a obs_AH: RN → AH
17b obs_AL: RN → AL

4.5.1.2.3 Part Sets

Definition 38 . ***Part Sets:***

- ***Part set***s are those which,
  - in a given context,
  - are deemed to *meaningfully* consist of separately observable
    - a ["root"] part and
    - an indefinite number of proper ["sibling"] *'sub-part*s ■

- For pragmatic reasons we distinguish between parts sets all of whose parts are
  - of the same, single, further un-analysed sort, and
  - of two or more distinct atomic sorts.

Definition 39 . ***Single Sort Part Sets:***

- ***Single sort part set***s are those which,
  - in a given context,
  - are deemed to *meaningfully* consist of separately observable
    - a ["root"] part and
    - an indefinite number of proper ["sibling"] *'sub-part*s of the same, i.e., single sort ■

Analysis Predicate Prompt 10 . `is_single_sort_set:`

- *The domain analyser analyses a solid endurant, i.e., a part p into a set endurant:*
  - `is_single_sort_set`: *p is a composite endurant if* `is_single_sort_set(p)` *holds* ■

`is_single_sort_set` is a method tool.

- The `is_single_sort_set` predicate is informal.
- So are all the domain analysis predicates (and functions).
- That is,
  - Their values are “calculated” by a human, the domain analyser.
  - That person observes parts in the “real world”.
  - The determination of the predicate values, hence, are subjective.

Definition 40 . ***Alternative Atomic Part Sets:***

- ***Alternative sorts part set***s are those which,
  - in a given context,
  - are deemed to *meaningfully* consist of separately observable
    - a ["root"] part and
    - an indefinite number of proper ["sibling"] *'sub-part*s of two or more atomic parts of distinct sorts ■

Analysis Predicate Prompt 11 . *is_alternative_sorts_set:*

- *The domain analyser analyses a solid endurant, i.e., a part p into a set endurant:*
  - *is_alternative_sorts_set: p is a composite endurant if* `is_alternative_sorts_set(p)` *holds* ■

`is_alternative_sorts_set` is a method tool.

4.5.1.2.3.1 Determine Same Sort Part Sets

Observer Function Prompt 3 . `determine_same_sort_parts_set:`

- *The domain analyser observes parts into same sorts part sets.*
- *The method provides the* ***domain observer prompt****:*
  - `determine_alternative_sorts_part_set`
    directs the domain analyser to determine the values and corresponding sorts of the part.

```
value
  determine_same_sort_part_set: E → (P-set×θP)
  determine_same_sort_part_set(e) as (ps,ηPn)
```

`determine_same_sort_part_set` is a method tool.

4.5.1.2.3.2 Determine Alternative Sorts Part Sets

Observer Function Prompt 4.
determine_alternative_sorts_part_set:

- *The domain analyser observes parts into alternative sorts part sets.*
- *The method provides the* ***domain observer prompt****:*
  - *determine_alternative_sorts_part_set*
    directs the domain analyser to determine the values and corresponding sorts of the part.

    value
    determine_alternative_sorts_part_set: E → ((P1×θP1)×...×(Pn,θPn))
    determine_alternative_sorts_part_set(e) **as** ((p1,ηp1),...,(pn,ηPn))

  - *The set of parts, of different sorts, may have more than one element, p, p′,..., p″ being of the same sort Ei.*

determine_alternative_sorts_part_set is a method tool.

4.5.1.2.3.3 Calculating Single Sort Part Sets

Domain Description Prompt 3 . `calc_single_sort_parts_sort:`

- *If* `is_single_set_sort_parts(e)` *holds, then the analyser "applies" the* domain description prompt
  - `calc_single_sort_parts_sort(e)`

  resulting in the analyser writing down the single set sort and sort observers *domain description text according to the following schema:*

calculate_single_sort_parts_sort(e) Describer

let (_,ηP) = determine_single_sort_part(e)[11] **in**
❝ **Narration:**
[s] ... narrative text on sort ...
[o] ... narrative text on sort observer ...
[p] ... narrative text on proof obligation ...
Formalisation:
type
[s] P
[s] Ps = P-**set**
value
[o] obs_Ps: E → Ps ❞
proof obligation
[p] [Single"sortness" of Ps]❞
end

`calculate_single_sort_parts_sort` is a method tool.

Elaboration 2 *Type, Values and Type Names:*

- *Note the use of quotes above.*
- *Please observe that when we write* obs_Ps *then* obs_Ps *is the name of a function.*
- *The* Ps*, when juxtaposed to* obs_ *is now a name* ■

[11] For `determine_single_sort_part` see Defn. 39 on Slide 108.

Example 30 . Road Transport System: Sets of Hubs, Links and Automobiles: We refer to Example 29 on Slide 105.

18. The road net aggregate of road net hubs consists of a set of [atomic] hubs,
19. The road net aggregate of road net links consists of a set of [atomic] links,
20. The road net aggregate of automobiles consists of a set of [atomic] automobiles.

type
18. Hs = H-**set**, H
18. Ls – L-**set**, L
18. As = A-**set**, A

value
18. obs_Hs: AH → Hs
18. obs_Ls: AL → Ls
18. obs_As: AA → As ■

4.5.1.2.3.4 Calculating Alternative Sort Part Sets

- We leave it to the reader to decipher the `calculate_alternative_sort_part_sorts` prompt.

Domain Description Prompt 4 .
`calculate_alternative_sort_part_sorts`:

- *If `is_alternative_sort_parts_sorts(e)` holds, then the analyser "applies" the domain description prompt*
  - `calculate_alternative_sort_part_sorts(e)`

resulting in the analyser writing down
the alternative sort and sort observers
domain description text
according to the following schema:

calculate_alternative_sort_part_sorts(e) Describer

let ((p1,ηE_1),...,(pn,ηE_n)) = determine_alternative_sorts_part_set_sorts(e)[12] **in**
❝ **Narration:**
[s] ... narrative text on alternative sorts ...
[o] ... narrative text on sort observers ...
[p] ... narrative text on proof obligations ...
Formalisation:
type
[s] Ea = E_1 | ... | E_n
[s] E_1 :: End_1, ..., E_n :: End_n
value
[o] obs_Ea: E → Ea
proof obligation
[p] [disjointness **of** alternative sorts] E_1, ..., E_n ❞
end

- The set of parts, of different sorts, may have more than one element, say $p, p', ..., p''$ being of the same sort E_i.
  - Since parts are not mentioned in the sort description above, cf., –,
  - only the distinct alternative sort observers appear in that description.

`calculate_alternative_sort_part_sorts` is a method tool.

[12] For `determine_alternative_sort_part_sorts` see Defn. 40 on Slide 111.

Example 31 . Alternative Rail Units:

21. The example is that of a railway system.
22. We focus on railway nets. They can be observed from the railway system.
23. The railway net embodies a set of [railway] net units.
24. A net unit is either a
    - straight or curved **linear** unit, or a
    - simple switch, i.e., a **turnout**, unit[13] or
    - a simple cross-over, i.e., a **rigid** crossing unit, or a
    - single switched cross-over, i.e., a **single** slip unit, or a
    - double switched cross-over, i.e., a **double** slip unit, or a
    - **terminal** unit.
25. As a formal specification language technicality disjointness of the respective rail unit types is afforded by RSL's :: type definition construct.

- We refer to Figure 4.2 on the next slide.

type
21. RS
22. RN

value
22. obs_RN: RS → RN

type
23. NUs = NU-**set**
24. NU = LU|PU|RU|SU|DU|TU
25. LU :: LinU
25. PU :: PntU
25. SU :: SwiU
25. DU :: DblU
25. TU :: TerU

value
23. obs_NUs: RN → NUs

[13] https://en.wikipedia.org/wiki/Railroad_switch

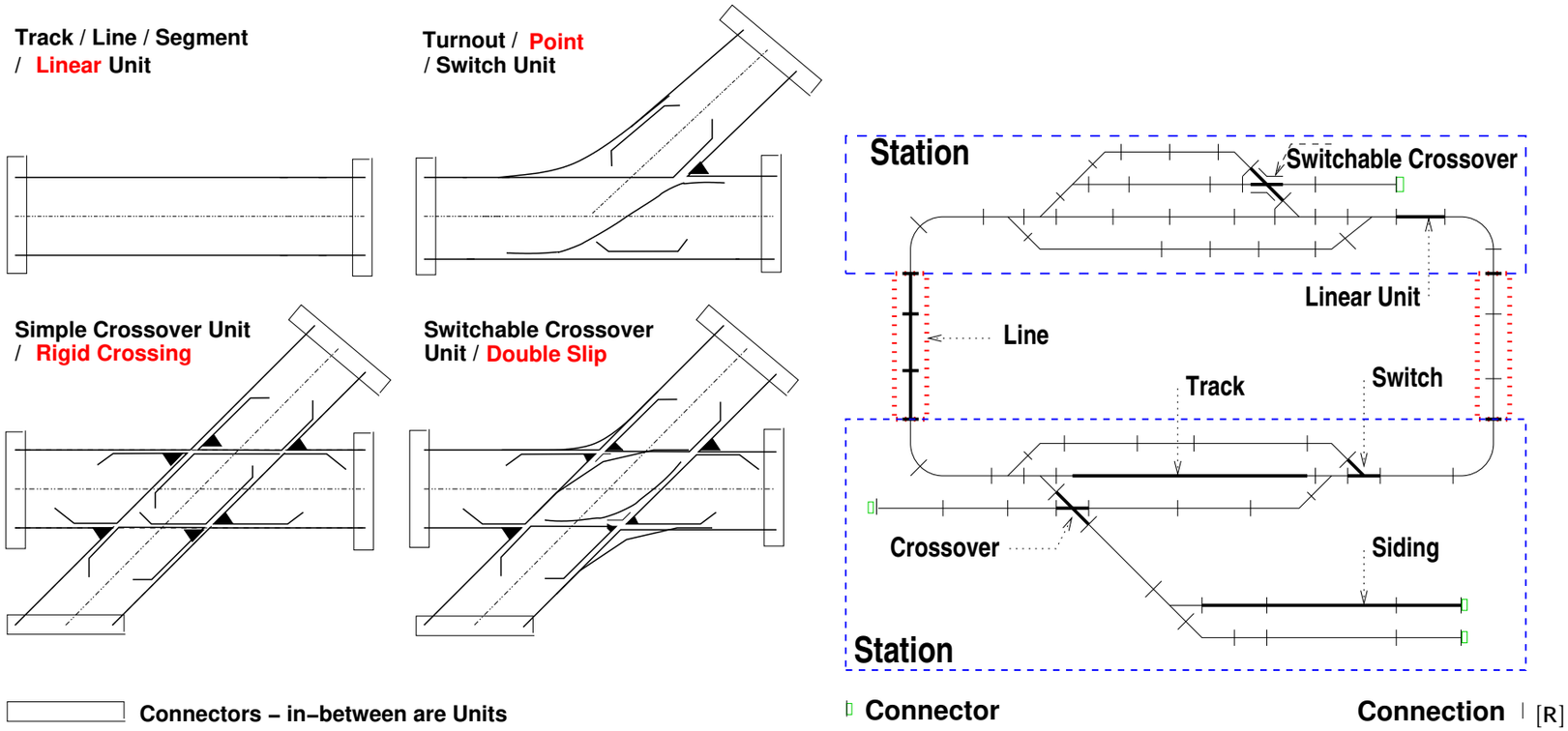

Figure 4.2: Left: Four net units (LU, PU, SU, DU); Right: A railway net

• • •

Method Principle 6 . *Pedantic Steps of Development:*

- *This section, i.e., Sect. 4.5.1, has illustrated a principle of "small, pedantic" analysis & description steps.*
  - *You could also call it a principle of separation of concerns* ■

Day #2: External Qualities, Synthesis (II)

4.5.1.3 Ontology and Taxonomy

- We can speak of two kinds of ontologies:
  - the general ontologies of domain analysis & description, cf. Fig. 4.1 on Slide 64, and
  - a specific domain's possible endurant ontologies.
  - We shall here focus on a ["restricted"] concept of taxonomies[14]

[14] By taxonomy (or taxonomical classification) we shall here understand a scheme of classification, especially a hierarchical classification, in which things are organized into groups [Wikipedia].

Definition 41 . *Domain Taxonomy:* By a domain taxonomy we shall understand

- a hierarchical structure,
  usually depicted as a(n "upside-down") tree,
- whose "root" designates a compound part
- and whose "siblings" (proper sub-trees)
  designate parts or fluids ■

- The 'restriction' amounts to considering only endurants.
- That is, not considering perdurants.
- Taxonomy is a method technique.

Example 32 . The Road Transport System Taxonomy:

- Figure 4.3 shows a schematised, i.e., the . . . , taxonomy for the *Road Transport System* domain of Example 4.1 on Slide 64.

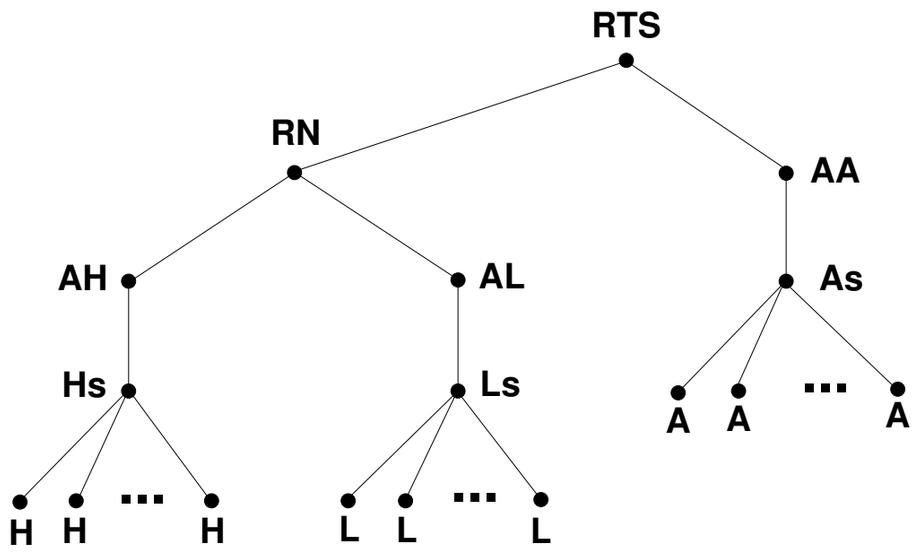

Figure 4.3: A Road Transport System Ontology ■

4.5.1.4 "Root" and "Sibling" Parts

- For compound parts, cf. Definition 36 on Slide 92,
  - we introduce the specific domain taxonomy concepts of "root" and "sibling" parts.
  - (We also refer to Fig. 4.3 on the facing slide.)
- When observing, as a human, a compound part one may ask the question
  - *"a tree consisting of a specific domain taxonomy node labelled, e.g., X*
  - *and the sub-trees labelled, e.g., $Y_1, Y_2, \ldots, Y_n$*
  - *does that tree designate one "indivisible" part*
  - *or does it designate $n+1$ parts ?"*
  - We shall, in general, consider the answer to be the latter: $n+1$!

- We shall, in general, consider compound parts to consist of
  - a "root" parts
  - and n "sibling parts and fluids".
- What the analyser cum describer observes
  - appears as one part, "the whole",
  - with n "embedded" sub-parts.
- What the analyser cum describer is asked to model is
  - 1, the root part, and
  - n, the sibling, parts and fluids.

- The fact that the root part is separately modelled from the sibling parts,
- may seem to disappear in this separate modelling —
- but, as You shall see, in the next chapter,
  - their relation: the siblings to "the whole", i.e., the root,
  - will be modelled, specifically through their mereologies,
  - as will be covered in Sect. 5.3,
  - but also through their respective attributes, Sect. 5.4.
- We shall see this non-embbedness of root and sibling parts
  - further accentuated in the modelling of their transcendentally deduced
  - respective (perdurant) behaviours as distinct concurrent behaviours
  - in Chapter 6.

4.5.2 Living Species

- *Living Species* are
  - either *plants*
  - or *animals*.
- Among animals we have the *humans*.

Definition 42 . *Living Species:*

- By a *living species* we shall understand
  - a solid endurant,
  - subject to laws of physics, and
  - additionally subject to *causality of purpose*.
- Living species
  - must have some *form they can be developed to reach*;
  - a form they must be *causally determined to maintain*.
  - This *development and maintenance* must further engage in *exchanges of matter with an environment*.
- It must be possible that living species occur in two forms:
  - plants, respectively animals,
  - forms which are characterised by *development, form and exchange*,
  - which, additionally, can be characterised by the *ability of purposeful movement* ■

Analysis Predicate Prompt 12 . `is_living_species:`

- *The domain analyser analyses "things" (e) into living species.*
- *The method can thus be said to provide the* ***domain analysis prompt****:*
  - `is_living_species` *– where* `is_living_species(e)` *holds if e is a living species* ■

`is_living_species` is a method tool.

- It is appropriate here to mention **Carl Linnaeus** (1707–1778).
  - He was a Swedish botanist, zoologist, and physician
  - who formalised, in the form of a binomial nomenclature,
  - the modern system of naming organisms.
  - He is known as the “father of modern taxonomy”.
  - We refer to his ‘Species Plantarum’ gutenberg.org/files/20771/20771-h/20771-h.htm.

4.5.2.1 Plants

Example 33 . Plants:

- Although we have not yet come across domains for which the need to model the living species of plants were needed, we give some examples anyway:
  - grass,
  - tulip,
  - rhododendron,
  - oak tree.

Analysis Predicate Prompt 13 . `is_plant:`

- *The domain analyser analyses "things" (ℓ) into a plant.*
- *The method can thus be said to provide the* ***domain analysis prompt****:*
  - `is_plant` *– where* `is_plant(`ℓ`)` *holds if* ℓ *is a plant* ■

- `is_plant` is a method tool.
- The predicate `is_living_species(`ℓ`)` is a prerequisite for `is_plant(`ℓ`)`.

4.5.2.2 Animals

Definition 43 . ***Animal:*** We refer to the initial definition of *living species* above – while emphasizing the following traits:

- (i) a *form that animals can be developed to reach* and
- (ii) *causally determined to maintain* through
- (iii) *development and maintenance*
  in an *exchange of matter with an environment*, and
- (iv) *ability to purposeful movement* ■

Analysis Predicate Prompt 14. `is_animal:`

- *The domain analyser analyses "things" (ℓ) into an animal.*
- *The method can thus be said to provide the* ***domain analysis prompt****:*
  - `is_animal` *– where* `is_animal(`ℓ`)` *holds if* ℓ *is an animal* ■

- `is_animal` is a method tool.
- The predicate `is_living_species(`ℓ`)` is a prerequisite for `is_animal(`ℓ`)`.
- We distinguish, motivated by [49], between
  - humans and
  - other.

4.5.2.2.1 Humans

Definition 44. *Human:*

- A *human* (a *person*) is an *animal*,
  cf. Definition 43 on Slide 138,
  with the additional properties of having
  - *language*,
  - being *conscious* of *having knowledge* (of its own situation), and
  - *responsibility* ■

Analysis Predicate Prompt 15 . `is_human:`

- *The domain analyser analyses "things" (ℓ) into a human.*
- *The method can thus be said to provide the* ***domain analysis prompt****:*
  - `is_human` – *where* `is_human(`ℓ`)` *holds if* ℓ *is a human* ■

- `is_human` is a method tool.
- The predicate `is_animal`(ℓ) is a prerequisite for `is_human`(ℓ).

- We have not,
  in our many experimental domain modelling efforts
  - had occasion to model humans;
  - or rather:
    - we have modelled, for example, automobiles
      - as possessing human qualities,
      - i.e., “subsuming humans”.

- We have found,
  in these experimental domain modelling efforts
  - that we often confer anthropomorphic qualities on artefacts,
  - that is, that these artefacts have human characteristics.
- You, the listeners, are reminded
  - that when some programmers try to explain their programs
  - they do so using such phrases as
  - *and here the program does ...* so-and-so !

4.5.2.2.2 Other

- We shall skip any treatment of other than human animals !

• • •

External Quality Analysis & Description First is a method procedure.

4.6 Some Observations

- Two observations must be made.
  - (i) The domain analyser cum describer procedures
    - illustrated by the analysis functions
    - `determine_Cartesian_parts`,
    - `determine_same_sort_part_set` and
    - `determine_alternative_sorts_part_set`
    - yield names of endurant sorts.
    - Some of these names may have already been encountered, i.e., discovered.
    - That is, the domain analyser cum describer must carefully consider such possibilities.

– (ii) Endurants are **not recursively definable** !
  * This appears to come as a surprise
    to many computer scientists.
  * Immediately many suggest that "tree-like" endurants
    like a river,
  * or, indeed, a tree,
  * should be defined recursively.
  * But we posit that that is not the case.
  * A river, for example, has a delta, its "root" so-to-speak,
  * but the sub-trees of a recursively defined river endurant
  * has no such "deltas" !
  * Instead we define such "tree-like" endurants
    as graphs with appropriate mereologies.

4.7 States

- In our continued modelling
  - we shall make good use of a concept of states.

Definition 45 . ***State:*** By a *state* we shall understand

- any collection of one or more parts ■

- In Chapter 5 Sect. 5.4
  we introduce the notion of *attributes.*
  - Among attributes there are the *dynamic attributes*.
  - They model that internal part quality values
    may change dynamically.
  - So we may wish, on occasion, to 'refine' our notion of state
    to be just those parts which have dynamic attributes.

4.7.1 State Calculation

- Given any universe of discourse, uod:UoD, we can recursively calculate its "full" state, calc_parts({uod}).

26. Let e be any endurant. Let arg_parts be the parts to be calculated. Let res_parts be the parts calculated. Initialise the calculator with arg_parts={e} and res_parts={}. Calculation stops with arg_parts empty and res_parts the result.
27. If `is_Cartesian`(e)
28. then we obtain its immediate parts, determine_composite_part(e)
29. add them, as a set, to arg_parts, e removed from arg_parts and added to res_parts calculating the parts from that.
30. If `is_single_sort_part_set`(e)
31. then the parts, ps, of the single sort set are determined,
32. added to arg_parts and e removed from arg_parts and added to res_parts calculating the parts from that.
33. If `is_alternative_sorts_part_set`(e) then the parts, ((p1,_),(p2,_),...,(pn,_)), of the alternative sorts set are determined, added to arg_parts and e removed from arg_parts and added to res_parts calculating the parts from that.

```
value
26.  calc_parts: E-set → E-set → E-set
26.  calc_parts(arg_parts)(res_parts) ≡
26.    if arg_parts = {} then res_parts else
26.    let e · e ∈ arg_parts in
27.    is_Cartesian(e) →
28.      let ((e1,e2,...,en),_) = observe_Cartesian_part(e) in
29.      calc_parts(arg_parts\{e} ∪ {e1,e2,...,en})(res_parts ∪ {e}) end
30.    is_single_sort_part_set(e) →
31.      let ps = observe_single_sort_part_set(e) in
32.      calc_parts(arg_parts\{e}∪ ps)(res_parts ∪ {e}) end
33.    is_alternative_sort_part_set(e) →
33.      let ((p1,_),(p2,_),...,(pn,_)) = observe_alternative_sorts_part_set(e) in
33.      calc_parts(arg_parts\{e}∪{p1,p2,...,pn})(res_parts ∪ {e}) end
26.    end end
```

`calc_parts` is a method tool.

Method Principle 7 . *Domain State:*

- *We have found, once all the state components,*
  i.e., the endurant parts,
  have had their external qualities analysed, that
  - *it is then expedient to define the domain state.*
  - *It can then be the basis for several concepts*
  - *of internal qualities.*

Example 34 . Constants and States:

34. Let there be given a universe of discourse, *rts*.
    The set {*rts*} is an example of a state.

From that state we can calculate other states.

35. The set of all hubs, *hs*.
36. The set of all links, *ls*.
37. The set of all hubs and links, *hls*.
38. The set of all automobiles, *as*.
39. The set of all parts, *ps*.

value

34 rts:UoD [34]

35 hs:H-**set** ≡ obs_sH(obs_SH(obs_RN(rts)))

36 ls:L-**set** ≡ obs_sL(obs_SL(obs_RN(rts)))

37 hls:(H|L)-**set** ≡ $hs \cup ls$

38 as:A-**set** ≡ obs_As(obs_AA(obs_RN(rts)))

39 ps:(UoB|H|L|A)-**set** ≡ $rts \cup hls \cup as$

4.7.2 Update-able States

- We shall, in Sect. 5.4, introduce the notion of parts,
  - having dynamic attributes,
  - that is, having internal qualities that may change.
- To cope with the modelling,
- in particular of so-called *monitor-able* attributes,
- we present the *state* as a global variable:

variable σ := calc_parts({uod})

4.8 An External Analysis and Description Procedure

- We have covered
  - the individual analysis and description steps
  - of our approach to the external qualities modelling
  - of domain endurants.
- We now suggest
  - a ‘formal’ description of the process
  - of linking all these analysis and description steps.

4.8.1 An Analysis & Description State

- Common to all the discovery processes is an idea of a *notice board*.
- A notice board, at any time in the development of a domain description, is a repository of the analysis and description process.
- We suggest to model the notice board in terms of three global variables.
  - The new variable holds the parts yet to be described,
  - The ans variable holds the sort name of parts that have so far been described,
  - the gen variable holds the parts that have so far been described, and
  - the txt variable holds the RSL-Text so far generated.
    - We model the txt variable as a map
    - from endurant identifier names to RSL-Text.

A Domain Discovery Notice Board

variable
  new := {uod} ,
  asn := { “UoD ”}
  gen := {} ,
  txt:RSL-**Text** := [uid_UoD(uod) ↦ ⟨“**type** UoD ”⟩]

4.8.2 A Domain Discovery Procedure, I

- The discover_sorts pseudo program
  - suggests a systematic way of proceeding
  - through analysis, manifested by the is_⋯ predicates,
  - to (→) description.
- Some comments are in order.
  - The e-set$_a$⊎e-set$_b$ expression
  - yields a set of endurants that are either in e-set$_a$, or in e-set$_a$, or in both,
  - but such that two endurants, e$_x$ and e$_y$
  - which are of the same endurants type, say E,
  - and are in respective sets is only represented once in the result;
  - that is, if they are type-wise the same, but value-wise different
  - they will only be included once in the result.
- As this is the first time RSL-**Text** is put on the notice board
  we express this as:
  - txt := txt ∪ [type_name(v) ↦ ⟨RSL-**Text**⟩]
- Subsequent insertion of RSL-**Text**
  for internal quality descriptions and perdurants
  is then concatenated to the end of previously uploaded RSL-**Text**.

An External Qualities Domain Analysis and Description Process

```
value
discover_sorts: Unit → Unit
discover_sorts() ≡ while new ≠ {} do
  let v • v ∈ new in (new := new \ {v} || gen := gen ∪ {v} || ans := ans \ {type_of(v)}) ;
  is_atomic(v) → skip ,
  is_compound(v) →
    is_Cartesian(v) →
     let ((e1,...,en),(ηE1,...,ηEn))=analyse_composite_parts(v) in
     (ans := ans ∪ {ηE1,...,ηEn} || new := new ⊎ {e1,...,en}
       || txt := txt ∪ [ type_name(v) ↦ ⟨calculate_composite_part_sorts(v)⟩ ]) end,
    is_part_set(v) →
     (is_single_sort_set(v) →
       let ({p1,...,pn},ηP)=analyse_single_sort_parts_set(v) in
       (ans := ans ∪ {ηP} || new := new ⊎ {p1,...,pn} ||
         txt := txt ∪ [ type_name(v) ↦ calculate_single_sort_part_sort(v) ]) end,
      is_alternative_sorts_set(v) →
       let ((p1,ηE1),...,(pn,ηEn))= observe_alternative_sorts_part_set(v) in
       (ans := ans ∪ {ηE1,...,En} || new := new ⊎ {p1,...,pn} ||
         txt := txt ∪ [ type_name(v) ↦ calculate_alternative_sorts_part_sort(v) ]) end)
end end
```

`discover_sorts` is a method procedure.

4.9 Summary

- We briefly summarise the main findings of this chapter.
- These are the main
  - analysis predicates and functions and
  - the main description functions.
- These, to remind the student, are
  - the *analysis*, the **is_···**, *predicates*,
  - the *analysis*, the **determine_···**, *functions*,
  - the *state calculation* function,
  - the *description* functions, and
  - the *domain discovery* procedure.

- They are summarised in this table:

#	Name	Introduced
	Analysis Predicates	
1	is_entity	page 61
2	is_endurant	page 71
3	is_perdurant	page 74
4	is_solid	page 78
5	is_fluid	page 81
6	is_part	page 86
7	is_atomic	page 90
8	is_compound	page 93
9	is_Cartesian	page 96
10	is_single_sort_set	page 109
11	is_alternative_sorts_set	page 112
12	is_living_species	page 134
13	is_plant	page 137
14	is_animal	page 139
15	is_human	page 141
	Analysis Functions	
1	determine_Cartesian_parts	page 100
3	determine_same_sort_part_set	page 113
4	determine_alternative_sorts_part_set	page 114
	State Calculation	
	calc_parts	page 149
	Description Functions	
1	calc_Universe_of_Discourse	page 56
2	calc_Cartesian_parts	page 102
3	calc_single_sort_parts_sort	page 115
4	calc_alternative_sort_part_sorts	page 116
	Domain Discovery	
	discover_sorts	page 159

• • •

- Please consider Fig. 4.1 on Slide 64.
  - This chapter has covered the tree-like structure to the left in Fig. 4.1.
  - The next chapter covers the horisontal and vertical lines, also to the left in Fig. 4.1.