Dines Bjørner's MAP-i Lecture # 4

Components, Materials – and Discussion of Endurants

Monday, 25 May 2015: 16:45–17:30

1.2.10. Components

- Components are discrete endurants which are not considered parts.
  - $\texttt{is\_component}(k) \equiv \texttt{is\_endurant}(k) \wedge \sim\texttt{is\_part}(k)$

Example 45 . Parts and Components:

- We observe components as associated with atomic parts:
  - The contents, that is, the collection of zero, one or more boxes, of a container is the components of the container part.
  - Conveyor belts transport machine assembly units and are thus considered the components of the conveyor belt.

- We now complement the `observe_part_sorts` (of earlier).
- We assume, without loss of generality, that only atomic parts may contain components.
- Let p:P be some atomic part.

Analysis Prompt 15 . *has_components:*

- *The* **domain analysis prompt***:*
  - ◈ ***has_components***(p)
- *yields* **true** *if atomic part p potentially contains components otherwise false* ■

- Let us assume that parts $p{:}P$ embodies components of sorts $\{K_1,K_2,\ldots,K_n\}$.
- Since we cannot automatically guarantee that our domain descriptions secure that
  - each K_i ([$1 \le i \le$n])
  - denotes disjoint sets of entities

  we must prove it.

Domain Description Prompt 6 . *observe_component_sorts*:

- *The* **domain description prompt***:*
  - ***observe_component_sorts***(e)

  yields the component sorts and component sort observers *domain description text according to the following schema:*

6. observe_component_sorts schema

Narration:

[s] ... narrative text on component sorts ...
[o] ... narrative text on component sort observers ...
[i] ... narrative text on component sort recognisers ...
[p] ... narrative text on component sort proof obligations ...

Formalisation:

type
[s] K1, K2, ..., Kn
[s] KS = (K1|K2|...|Kn)-**set**
value
[o] **components**: P → KS
[i] **is**_K$_i$: K → **Bool** [$1 \leq i \leq n$]

Proof Obligation:

[Disjointness of Component Sorts]
[p] $\forall\, m_i$:(K$_1$|K$_2$|...|K$_n$) •
[p] $\bigwedge$ {**is**_K$_i$(m_i) $\equiv$ $\bigvee$∼{**is**_K$_j$(m_i)|j ∈ {1..m}\{i}}|i ∈ {1..m}}

Example 46 . Container Components: We continue Example 22 on Slide 135.

60 When we apply `obs_component_sorts_C` to any container c:C we obtain

a. a type clause stating the sorts of the various components of a container,

b. a union type clause over these component sorts, and

c. the component observer function signature.

type
60a. K1, K2, ..., Kn
60b. KS = (K1|K2|...|Kn)-**set**
value
60c. **obs_comp**_KS: C → KS ■

- We have presented one way of tackling the issue of describing components.
  - There are other ways.
  - We leave those ‘other ways’ to the reader.
- We are not going to suggest techniques and tools for analysing, let alone describing qualities of components.
  - We suggest that conventional abstraction of modeling techniques and tools be applied.

1.2.11. Materials

- Continuous endurants (i.e., **materials**) are entities, m, which satisfy:
  - $\texttt{is\_material}(m) \equiv \texttt{is\_endurant}(m) \wedge \texttt{is\_continuous}(m)$

Example 47 . Parts and Materials:

- We observe materials as associated with atomic parts:
  - Thus liquid or gaseous materials are observed in pipeline units

- We shall in this seminar not cover
  the case of parts being immersed in materials.

- We assume, without loss of generality, that only atomic parts may contain materials.
- Let $p{:}P$ be some atomic part.

Analysis Prompt 16 . *has_materials:*

- *The* **domain analysis prompt***:*

  ⟐ ***has_materials***(p)

- *yields* **true** *if the atomic part p:P potentially contains materials otherwise false* ■

- Let us assume that parts p:P embodies materials of sorts $\{M_1,M_2,\ldots,M_n\}$.
- Since we cannot automatically guarantee that our domain descriptions secure that
  - each M_i ($[\,1\leq i\leq$n$\,]$)
  - denotes disjoint sets of entities

  we must prove it.

Domain Description Prompt 7 . *observe_material_sorts***:**

- *The* **domain description prompt***:*
  - ***observe_material_sorts***(e)

  yields the ***material sorts and material sort observers*** *domain description text according to the following schema:*

7. observe_material_sorts schema

Narration:

[s] ... narrative text on material sorts ...

[o] ... narrative text on material sort observers ...

[i] ... narrative text on material sort recognisers ...

[p] ... narrative text on material sort proof obligations ...

Formalisation:

type

[s] M_i [$1 \leq i \leq n$]

[s] MS = M1 M2 ... Mn

value

[o] **obs_mat**_M_i: P → M_i [$1 \leq i \leq n$]

[o] **materials**: P → MS

[i] **is**_M_i: M → **Bool** [$1 \leq i \leq n$]

proof obligation [Disjointness of Material Sorts]

[p] $\forall\, m_i$:(M_1|M_2|...|M_n) •

[p] $\bigwedge$ {**is**_$\mathsf{M}_i(m_i) \equiv \bigvee\sim${**is**_$\mathsf{M}_j(m_i)$|j ∈ {1..m}\{i}}|i ∈ {1..m}}

- *The M_i are all distinct* ■

Example 48 . Pipeline Material: We continue Example 27 on Slide 140 and Example 33 on Slide 162.

61 When we apply `obs_material_sorts_U` to any unit **u**:U we obtain

a. a type clause stating the material sort **LoG** for some further undefined liquid or gaseous material, and

b. a material observer function signature.

type
61a. LoG
value
61b. **obs_mat**_LoG: U $\rightarrow$ LoG ■

1.2.11.1. Materials-related Part Attributes

- It seems that the "interplay" between parts and materials
  - is an area where domain analysis
  - in the sense of this seminar
  - is relevant.

Example 49 . Pipeline Material Flow: We continue Examples 27, 33 and 48.

- 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

62 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.

63 current flow of material into a unit input connector,

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

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

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

67 current leak of material at a unit input connector,

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

69 current leak of material at a unit input connector,

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

71 current leak of material from "within" a unit, and

72 maximum guaranteed leak of material from "within" a unit.

type
62. F, L
value
63. **attr**_cur_iF: U → UI → F
64. **attr**_max_iF: U → UI → F
65. **attr**_cur_oF: U → UI → F
66. **attr**_max_oF: U → UI → F
67. **attr**_cur_iL: U → UI → L
68. **attr**_max_iL: U → UI → L
69. **attr**_cur_oL: U → UI → L
70. **attr**_max_oL: U → UI → L
71. **attr**_cur_L: U → L
72. **attr**_max_L: U → L

- The maximum flow attributes are static attributes and are typically provided by the manufacturer as indicators of flows below which laminar flow can be expected.
- The current flow attributes are dynamic attributes

1.2.11.2. Laws of Material Flows and Leaks

- 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.
- Modeling flows and leaks is important to the modeling of materials-based domains.

Example 50 . Pipelines: Intra Unit Flow and Leak Law:

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

74 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)]
73. ∀ pls:PLS,b:B\We\Si,u:U ·
73. b ∈ **obs_part**_Bs(pls)∧u=**obs_part**_U(b)⇒
73. **let** (iuis,ouis) = **obs_mereo**_U(u) **in**
74. sum_cur_iF(iuis)(u) =
74a.. sum_cur_iL(iuis)(u)
74b.. ⊕ **attr**_cur_L(u)
74c.. ⊕ sum_cur_oF(ouis)(u)
74d.. ⊕ sum_cur_oL(ouis)(u)
73. **end**

75 The **sum_cur_iF** (cf. Item 74) sums current input flows over all input connectors.

76 The **sum_cur_iL** (cf. Item 74a.) sums current input leaks over all input connectors.

77 The **sum_cur_oF** (cf. Item 74c.) sums current output flows over all output connectors.

78 The **sum_cur_oL** (cf. Item 74d.) sums current output leaks over all output connectors.

```
75.  sum_cur_iF: UI-set → U → F
75.  sum_cur_iF(iuis)(u) ≡ ⊕ {attr_cur_iF(ui)(u)|ui:UI•ui ∈ iuis}
76.  sum_cur_iL: UI-set → U → L
76.  sum_cur_iL(iuis)(u) ≡ ⊕ {attr_cur_iL(ui)(u)|ui:UI•ui ∈ iuis}
77.  sum_cur_oF: UI-set → U → F
77.  sum_cur_oF(ouis)(u) ≡ ⊕ {attr_cur_iF(ui)(u)|ui:UI•ui ∈ ouis}
78.  sum_cur_oL: UI-set → U → L
78.  sum_cur_oL(ouis)(u) ≡ ⊕ {attr_cur_iL(ui)(u)|ui:UI•ui ∈ ouis}
     ⊕: (F|L) × (F|L) → F
```

Example 51 . Pipelines: Inter Unit Flow and Leak Law:

79 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)]

79. ∀ pls:PLS,b,b′:B,u,u′:U•

79. {b,b′}⊆**obs_part**_Bs(pls)∧b≠b′∧u′=**obs_part**_U(b′)

79. ∧ **let** (iuis,ouis)=**obs_mereo**_U(u),(iuis′,ouis′)=**obs_mereo**_U(u′),

79. ui=**uid**_U(u),ui′=**uid**_U(u′) **in**

79. ui ∈ iuis ∧ ui′ ∈ ouis′ ⇒

79a.. **attr**_cur_oF(u′)(ui′) − **attr**_leak_oF(u′)(ui′)

79b.. = **attr**_cur_iF(u)(ui) + **attr**_leak_iF(u)(ui)

79. **end**

79. **comment:** b′ precedes b ■

- From the above two laws one can prove the **theorem:**
  - what is pumped from the wells equals
  - what is leaked from the systems plus what is output to the sinks.
- We need formalising the flow and leak summation functions.

1.2.12. "No Junk, No Confusion"

- Domain descriptions are, as we have already shown, formulated,
  - both informally
  - and formally,

  by means of abstract types,
  - that is, by sorts
  - for which no concrete models are usually given.
- Sorts are made to denote
  - possibly empty,
  - possibly infinite,
  - rarely singleton,
  - sets of entities on the basis of the qualities defined for these sorts, whether external or internal.

- By **junk** we shall understand
  - that the domain description
  - unintentionally denotes undesired entities.
- By **confusion** we shall understand
  - that the domain description
  - unintentionally have two or more identifications
  - of the same entity or type.
- The question is
  - *can we formulate a [formal] domain description*
  - *such that it does not denote junk or confusion*?
- The short answer to this is no!

- So, since one naturally wishes "no junk, no confusion" what does one do ?
- The answer to that is
  - *one proceeds with great care !*
- To avoid **junk** we have stated a number of **sort well-formedness axioms**, for example:
  - Slide 151 for *Well-formedness of Links, L, and Hubs, H*,
  - Slide 158 for *Well-formedness of Domain Mereologies*,
  - Slide 161 for *Well-formedness of Road Nets, N*,
  - Slide 163 for *Well-formedness of Pipeline Systems, PLS (0)*,
  - Slide 182 for *Well-formedness of Hub States, HΣ*,
  - Slide 220 for *Well-formedness of Pipeline Systems, PLS (1)*,
  - Slide 222 for *Well-formedness of Pipeline Systems, PLS (2)*,
  - Slide 229 for *Well-formedness of Pipeline Route Descriptors* and
  - Slide 233 for *Well-formedness of Pipeline Systems, PLS (3)*.

- To avoid confusion we have stated a number of proof obligations:
  - Slide 122 for *Disjointness of Part Sorts*,
  - Slide 178 for *Disjointness of Attribute Types* and
  - Slide 212 for *Disjointness of Material Sorts*.

Example 52 . No Pipeline Junk:

- We continue Example 27 on Slide 140 and Example 33 on Slide 162.

80 We define a proper pipeline route to be a sequence of pipeline units.

a. such that the i^{th} and $i{+}1^{\text{st}}$ units in sequences longer than 1 are (forward) adjacent, in the sense defined below, and

b. such that the route is acyclic, in the sense also defined below.

To formalise the above we describe some auxiliary notions.

1.2.12.0.1 Pipe Routes

81 A route descriptor is the sequence of unit identifiers of the units of a route (of a pipeline system).

type

80. $\mathsf{R}' = \mathsf{U}^\omega$

80. $\mathsf{R} = \{|\ \mathsf{r:Route'}\cdot\mathsf{wf\_Route(r)}\ |\}$

81. $\mathsf{RD} = \mathsf{UI}^\omega$

axiom [Well−formedness of Pipeline Route Descriptors, RD]

81. $\forall\ \mathsf{rd:RD} \cdot \exists\ \mathsf{r:R}\cdot\mathsf{rd{=}descriptor(r)}$

value

81. $\mathsf{descriptor:\ R} \to \mathsf{RD}$

81. $\mathsf{descriptor(r)} \equiv \langle \mathbf{uid}\_\mathsf{UI(r[\,i\,])}|\mathsf{i}:\mathbf{Nat}\cdot 1 \leq \mathsf{i} \leq \mathbf{len}\ \mathsf{r}\rangle$

82 Two units are (forward) adjacent if the output unit identifiers of one shares a unique unit identifier with the input identifiers of the other.

value

```
82.   adjacent: U × U → Bool
82.   adjacent(u,u′) ≡
82.     let (,ouis)=obs_mereo_U(u),
82.         (iuis,)=obs_mereo_U(u′) in
82.     ouis ∩ iuis ≠ {} end
```

83 Given a pipeline system, pls, one can identify the (possibly infinite) set of (possibly infinite) routes of that pipeline system.

a. The empty sequence, $\langle\rangle$, is a route of pls.

b. Let u be a unit of pls, then $\langle u\rangle$ is a route of pls.

c. Let u, u' be adjacent units of pls then $\langle u, u'\rangle$ is a route of pls.

d. If r and r' are routes of pls such that the last element of r is the same as the first element of r', then $r^\frown\mathbf{tl}\,r'$ is a route of pls.

e. No sequence of units is a route unless it follows from a finite number of applications of the basis and induction clauses of Items 83a.–83d..

```
value
83.    Routes: PLS → R-infset
83.    Routes(pls) ≡
83a..      let rs = ⟨⟩
83b..          ∪ {⟨u⟩|u:U•u ∈ obs_part_Us(pls)}
83c..          ∪ {⟨u,u′⟩|u,u′:U•{u,u′}⊆obs_part_Us(pls) ∧ adjacent(u,u′)}
83d..          ∪ {r⌢tl r′|r,r′:R•{r,r′}⊆rs∧r[len r]=hd r′}
83e..      in rs end
```

1.2.12.0.2 Well-formed Routes

84 A route is acyclic if no two route positions reveal the same unique unit identifier.

value

84. acyclic_Route: R $\rightarrow$ **Bool**
84. acyclic_Route(r) $\equiv$ $\sim\exists$ i,j:**Nat**$\cdot$\{i,j\}$\subseteq$**inds** r $\wedge$ i$\neq$j $\wedge$ r[i]=r[j]

1.2.12.0.3 Well-formed Pipeline Systems

85 A pipeline system is well-formed if

a. none of its routes are circular and

b. all of its routes are embedded in well-to-sink routes.

axiom [Well−formedness of Pipeline Systems, PLS (3)]

```
85.    ∀ pls:PLS ·
85a..        non_circular(pls)
85b..      ∧ are_embedded_in_well_to_sink_Routes(pls)
value
85.    non_circular_PLS: PLS → Bool
85.    non_circular_PLS(pls) ≡
85.        ∀ r:R·r ∈ routes(p)∧acyclic_Route(r)
```

86 We define well-formedness in terms of well-to-sink routes, i.e., routes which start with a well unit and end with a sink unit.

value

```
86. well_to_sink_Routes: PLS → R-set
86. well_to_sink_Routes(pls) ≡
86.     let rs = Routes(pls) in
86.     {r|r:R·r ∈ rs ∧ is_We(r[1]) ∧ is_Si(r[len r])} end
```

87 A pipeline system is well-formed if all of its routes are embedded in well-to-sink routes.

87. are_embedded_in_well_to_sink_Routes: PLS $\rightarrow$ **Bool**
87. are_embedded_in_well_to_sink_Routes(pls) $\equiv$
87. **let** wsrs = well_to_sink_Routes(pls) **in**
87. $\forall$ r:R $\cdot$ r $\in$ Routes(pls) $\Rightarrow$
87. $\exists$ r′:R,i,j:**Nat** $\cdot$
87. r′ $\in$ wsrs
87. $\wedge$ {i,j}$\subseteq$**inds** r′$\wedge$i$\leq$j
87. $\wedge$ r = $\langle$r′[k]|k:**Nat**$\cdot$i$\leq$k$\leq$j$\rangle$ **end**

1.2.12.0.4 Embedded Routes

88 For every route we can define the set of all its embedded routes.

value

88. embedded_Routes: R → R-**set**
88. embedded_Routes(r) ≡
88. {⟨r[k]|k:**Nat**·i≤k≤j⟩ | i,j:**Nat**· i {i,j}⊆**inds**(r) ∧ i≤j}

1.2.12.0.5 A Theorem

89 The following theorem is conjectured:

a. the set of all routes (of the pipeline system)

b. is the set of all well-to-sink routes (of a pipeline system) and

c. all their embedded routes

theorem:

89. ∀ pls:PLS ·
89. **let** rs = Routes(pls),
89. wsrs = well_to_sink_Routes(pls) **in**
89a.. rs =
89b.. wsrs ∪
89c.. ∪ {{r′|r′:R · r′ ∈ embedded_Routes(r″)} | r″:R · r″ ∈ wsrs}
88. **end** ■

- The above example,
  - besides illustrating one way of coping with “junk”,
  - also illustrated the need for introducing a number of auxiliary notions:
    - types,
    - functions,
    - axioms and
    - theorems.

1.2.13. Discussion of Endurants

- In Sect. 4.2.2 a “depth-first” search for part sorts was hinted at.
- It essentially expressed
  - that we discover domains epistemologically[16]
  - but understand them ontologically.[17]
- The Danish philosopher Søren Kirkegaard (1813–1855) expressed it this way:
  - *Life is lived forwards,*
  - *but is understood backwards.*
- The presentation of the of the **domain analysis prompt**s and the **domain description prompt**s results in domain descriptions which are ontological.
- The “depth-first” search recognizes the epistemological nature of bringing about understanding.

[16] **Epistemology**: the theory of knowledge, especially with regard to its methods, validity, and scope. Epistemology is the investigation of what distinguishes justified belief from opinion.

[17] **Ontology**: the branch of metaphysics dealing with the nature of being.

- This “depth-first” search
  - that ends with the analysis of atomic part sorts
  - can be guided, i.e., hastened (shortened),
  - by postulating composite sorts
  - that “correspond” to vernacular nouns:
  - everyday nouns that stand for classes of endurants.

- We could have chosen our **domain analysis prompt**s and **domain description prompt**s to reflect
  - a "bottom-up" epistemology,
  - one that reflected how we composed composite understandings
  - from initially atomic parts.
  - We leave such a collection of **domain analysis prompt**s and **domain description prompt**s to the student.

Dines Bjørner's MAP-i Lecture # 4

End of MAP-i Lecture # 4: Components, Materials – and Discussion of Endurants

Monday, 25 May 2015: 16:45–17:30