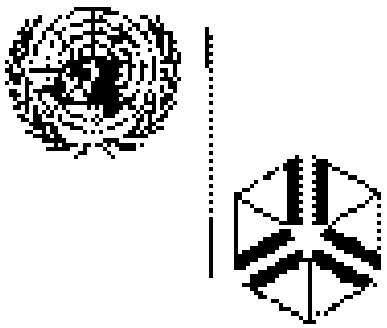
Formal Calculi Applied to Software Component Knowledge Elicitation



UNU/IIST — Seminar n.º 3
5–16 May 1997
Macau

J. Nuno Oliveira
University of Minho / INESC (Group 2361)
Braga

email: jno@di.uminho.pt
http://www.di.uminho.pt/~jno

Abstract

- For many years, industrial software development has been based on informal, graphical notations such as E-R diagrams, DFDs, flowcharts etc.
- Because the (formal) semantics of such drawings is not taken into account by commercially available CASE systems, software companies end up with massive repositories of ”drawings” which cannot be searched for and therefore are hardly re-used.
- This talk reports on a formal-method-based strategy devised for such large ”drawing” repository re-engineering. The work was carried out by INESC group 2361 for an Italian software company.
- Our experiment was focussed on E-R diagrams and was based on a SETS calculus semantics for such drawings.
- This made it possible to reason about such informal pieces of information and, in particular, to introduce hierarchical classification into E-R diagram repositories, supported by a software design calculus.
- Some effort was also put on developing tools for semi-automatic assistance in such a process which, however, could never be tested on a large scale basis.

Project Info

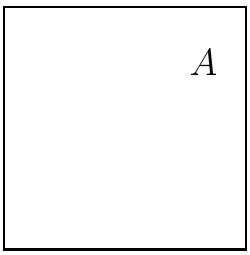
- **Partner Company**: OIS Ricerca S.c.p.A., Via Alberotanza 5, 70125 Bari, Italy
- **R&D Contract**: C.1.9. Sviluppo di Metodologie, Sistemi e Servizi Innovativi in Rete
- **CASE environment**: ENVISION ™

OIS Ricerca Motivation

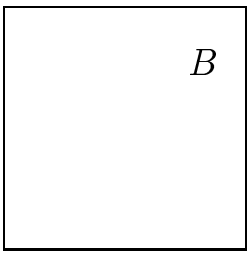
- The company has a huge legacy of code and documentation about banking software.
- Disks are full of large “drawings” (typically ER or DFD diagrams) produced a long time ago by several software designers (some of these have left the company).
- Lack of productivity: people feel lost and seem to be re-doing the past over and over again — *“I have a feeling that I’ve modelled a company like this before...”*
- Can we hierarchically organize these company-model diagrams in a repository which can be easily searched for?
- (Tools are welcome).

The Re-Use “Mitology”

My “new” (software) problem is. . .



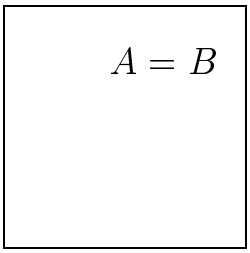
Do you know of any. . .



in your library which I could *reuse*?

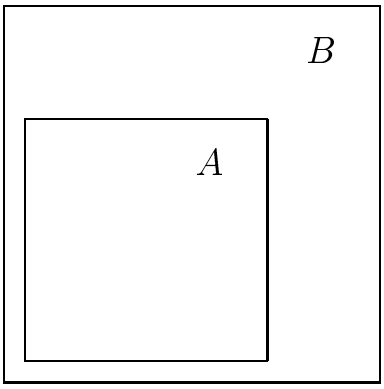
And what can I expect of B, if a particular one is found?

- Perfect match:



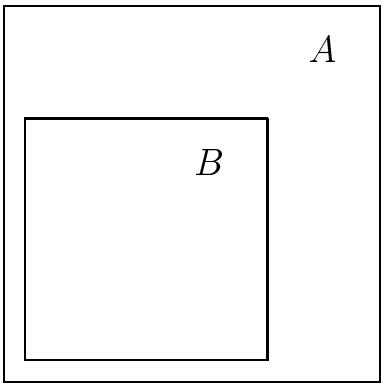
  Nothing left to do, my problem has been solved before;

- A is a special case of B:



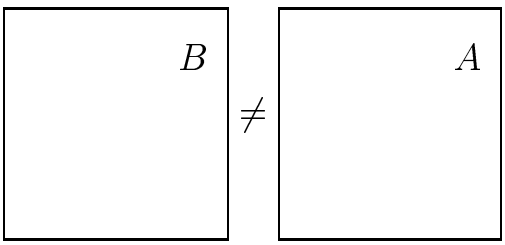
Little to do, my problem's class has already been treated already — just instantiate B and that's all;

- A generalizes B:



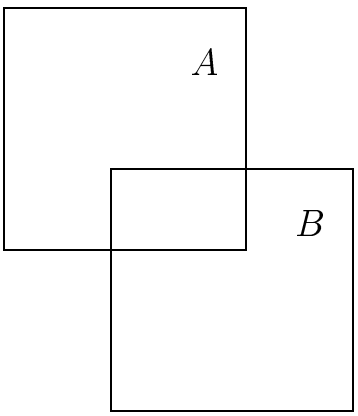
Ok—generalize B and save it as A; restore B as an instantiation of A.

- Worst case:

for all B in library, 

No help! A is really "new" and must be built from scratch.

- In general, A partially matches B:

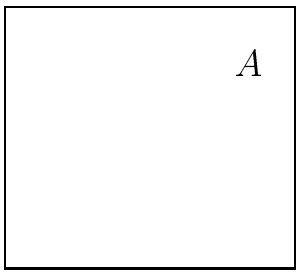


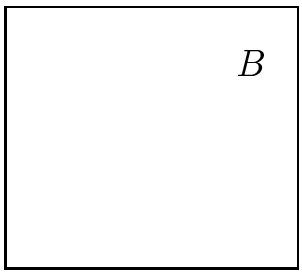
Something to do — the "common" part of A and B must be identified and A and B must be restored "sharing" such an intersection.

What About our Software "Legacy"?

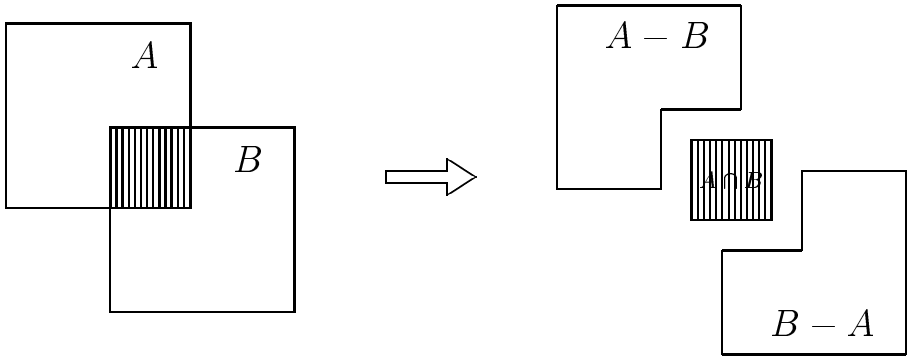
It's probably the case that we will need a kind of *software library "slimming diet"* algorithm, an iteration of:

{

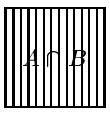
for all 

and 

in library
do



od
}

For every 

: a reusable component has been identified.

Prieto-Diaz "Re-use Algorithm"

From references [PDF87] and [PD91]:

```
begin search library;
   if identical match
   then terminate
   else begin collect similar components;
            for each component in library
                compute degree of match
            end;
            rank and select best;
            modify selected component
        end
end;
```

Questions

What does it mean...

...degree of match?

...best?

...similar?

⇒ Need for reliable software component comparison metrics

...what is the "semantics" of a software component?

...what is the best framework for capturing the "knowledge" implicit in arbitrary 'software' packages?

Approach

Map software component (code/documentation) into some kind of "metric space".

Alternatives

- **Generic but loose**:
  - attach OO-attributive profile to every piece of code or documentation, irrespectively of its "type", and
  - "classify by profile".

$\Rightarrow$ Cf. the SOUR approach (EU379).

- **Specific but accurate**:
  - define a formal semantics for each type of document/code and
  - "classify by semantics".

In any case...

- Comparison metrics unlikely to be linear such as *e.g.*

$$1 < 2 < 3 < \ldots < i < i + 1 < \ldots < max$$

- Thus

No best component

- This work adopts the second approach above in the context of the diagrammatic flavour of the ENVISION ™CASE system.
- An approach is put forward for comparing and classifying such graphical information and illustrated in the case of *Entity-Relationship* (ER) diagrams.

Work Plan:

- Address software comparison metrics in general + metrics-oriented repositories.

- Define a SETS formal semantics for ER diagrams.
- Formalize a comparison metrics for SETS expressions (entailing a classification for ER-digrams).

What is SETS? Recall on-going course on SETS — *A Data Structuring Calculus and Its Application to Program Development*.

Software Comparison Metrics in General

- Comparison in mathematics is usually expressed by an ordering on the specific domain of reasoning.
- concerning software descriptions (SDs), one may expect a *partial ordering* here, because not every two descriptions will be linearly comparable.
- Should two given descriptions be incomparable, a *least upper-bound* is expected to exist — the least abstract description which contains the "common information" of (*i.e.* which generalizes) the given descriptions.

Software description comparison can be expressed in terms of a function

$$comp : SD \times SD \longrightarrow SD \tag{1}$$

— the cornerstone of every comparison algebra:

- the comparison of an SD with itself should yield itself:

$$comp(x, x) = x$$

- SD-comparison should not depend on argument-order:

$$comp(x, y) = comp(y, x)$$

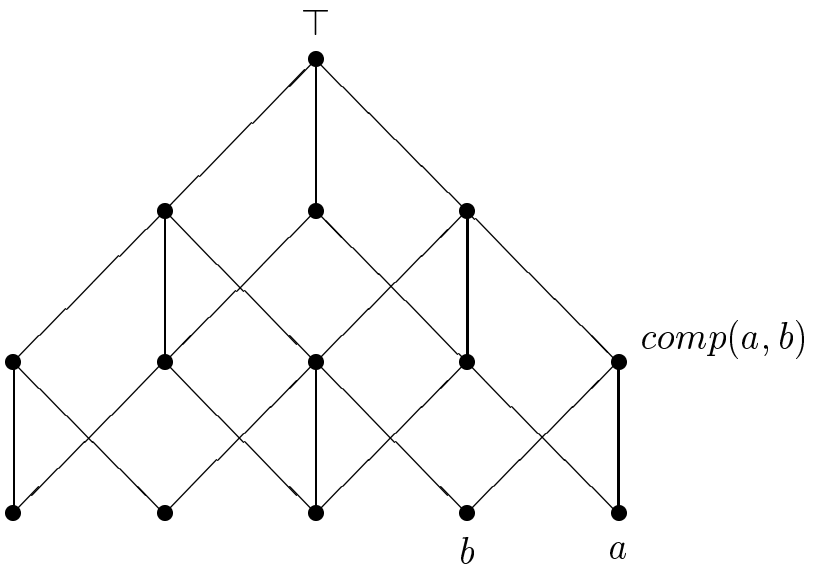
- SD-comparison should be compositional and associative:

$$comp(x, comp(y, z)) = comp(comp(x, y), z)$$

- a "most abstract" SD, $\top$, generalizes any other SD:

$$comp(\top, x) = comp(x, \top) = \top$$

Then $comp$ will induce a *specialize/generalize* hierarchy, with $\top$ at the top:



where

$a \leq b$ means $b = comp(a, b)$

Survey of Common Comparison Metrics

1. Keyword-based borrowed from classical library theory:

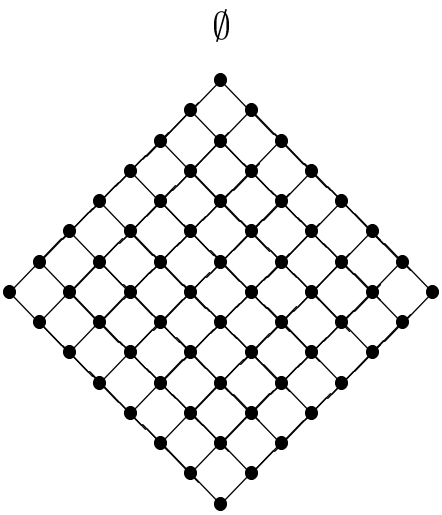
$$SD = 2^{Keyword} \tag{2}$$

such that

$$comp(x, y) = x \cap y$$

yields a "measure" of what x and y have in common.

In a picture:



$Keyword$

So $\top = \emptyset$.

1.1. Dewey-decimal-system specializes the keyword-approach:

$$Keyword = \overline{n}^{+}$$

e.g. $x = \{518.12.7, 509.5.12\}$. Based on the *longest common prefix* between two keywords, comparison requires an invariant defining a sub-lattice of (2).

2. Attributive-metrics generalizes the keyword approach:

$$SD = Keyword \hookrightarrow Value \tag{3}$$

SDs are attribute valuations, *i.e.* mappings from keywords (= *attribute names*) to a co-domain of *values* or *terms*, and

$$comp(x, y) = \left(\begin{array}{c} a \\ x(a) \end{array} \right)_{a \in dom(x) \cap dom(y) \wedge x(a) = y(a)} \tag{4}$$

and most abstract SD is the totally undefined function,

$$\top = \left(\quad \right)$$

Generalization is obvious in SETS:

$$2^{Keyword} \cong Keyword \hookrightarrow 1$$

so $Keyword \hookrightarrow Value$ generalizes $Keyword \hookrightarrow 1$

Increase in discriminant power is illustrated in the following example:

$$x = \{Programming, Operating\ System\}$$

becomes

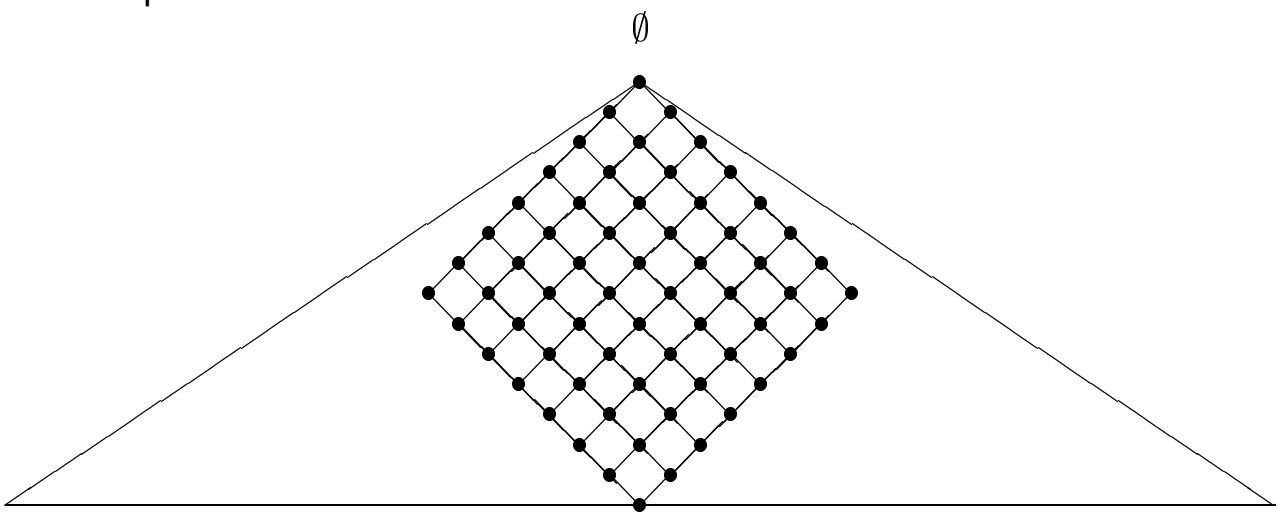
$$x = \begin{pmatrix} Programming & Operating\ System \\ C & UNIX \end{pmatrix}$$

or

$$x = \begin{pmatrix} Programming & Operating\ System \\ Dephi & Win95 \end{pmatrix}$$

etc.

In a picture:



3. Faceted-metrics generalizes the attributive-approach: equality in (4) is too "strict" — faceted metrics replace $x(a) = y(a)$ by $x(a) \simeq_f y(a)$ where $\simeq_f$ denotes "equal at fuzzyness level f".

Faceted classification was implemented in the SOUR [Oli95] repository;
ER diagrams and other pieces of information were given a fuzzy "conceptualization" and archived in the SOUR hierarchical repository — recall 1st seminar.

Here we want to proceed in another direction, that of basing classification on a SETS formal semantics for ER-diagrams.

An ER-diagram Semantics Cookbook

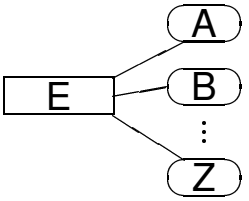
A quick reference list of rules for converting ER-diagram to the SETS notation follows:

In the ER-diagrams which follow, symbols E, F denote entities; symbols $\#E$, $\#F$ denote the relevant key attributes; and A, B, C, D, G, $\cdots$, Z denote attribute domains. Each rule is of the form

$$d = e$$

where d is a ER-diagram and e is the corresponding SETS denotation.

Entities



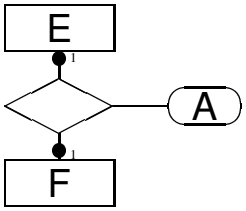
$$= \quad \#E \hookrightarrow A \times B \times \cdots \times Z \tag{5}$$

Relationships

In general we assume that an abstract attribute A is associated to each relationship, which in each case may actually be a Cartesian product of concrete attributes.

1:1 Relationships Several cases to be considered depending on the optionality of the involved entities E or F:

1. E and F both optional:



$$= \quad ((\#F \hookrightarrow \#E \times A)\times \\ (\#E \hookrightarrow \cdots)\times \\ (\#F \hookrightarrow \cdots))_{\phi} \tag{6}$$

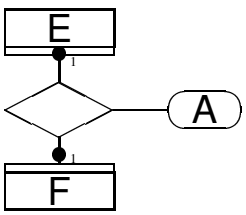
Local invariant:

$$\phi(\rho, \sigma, \sigma') \stackrel{\text{def}}{=} \\ \forall x, x' \in dom(\rho) : x \neq x' \Rightarrow \pi_1(\rho(x)) \neq \pi_1(\rho(x'))\wedge \\ dom(\rho) \subseteq dom(\sigma')\wedge \\ 2^{\pi_1}(rng(\rho)) \subseteq dom(\sigma)$$

In case of no attribute A in the relationship ($A = 1$):

$$\#F \hookrightarrow \#E \times 1 \cong \#F \hookrightarrow \#E$$

2. E compulsory and F optional:

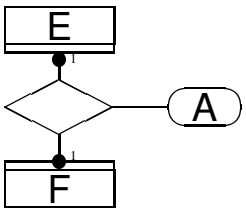


$$= \quad ((\#F \hookrightarrow \#E \times A)\times \\ (\#E \hookrightarrow \cdots)\times \\ (\#F \hookrightarrow \cdots))_{\phi} \tag{7}$$

Local invariant:

$$\phi(\rho, \sigma, \sigma') \stackrel{\text{def}}{=} \\ \forall x, x' \in dom(\rho) : x \neq x' \Rightarrow \pi_1(\rho(x)) \neq \pi_1(\rho(x'))\wedge \\ dom(\rho) \subseteq dom(\sigma')\wedge \\ 2^{\pi_1}(rng(\rho)) = dom(\sigma)$$

3. F compulsory and E optional:

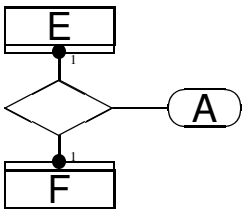


$$= ((\#F \hookrightarrow \#E \times A)\times (\#E \hookrightarrow \cdots)\times (\#F \hookrightarrow \cdots))_{\phi} \quad (8)$$

Local invariant:

$$\phi(\rho,\sigma,\sigma') \stackrel{\text{def}}{=} \forall x,x' \in dom(\rho) : x \neq x' \Rightarrow \pi_1(\rho(x)) \neq \pi_1(\rho(x'))\wedge dom(\rho) = dom(\sigma')\wedge 2^{\pi_1}(rng(\rho)) \subseteq dom(\sigma)$$

4. E and F both compulsory:



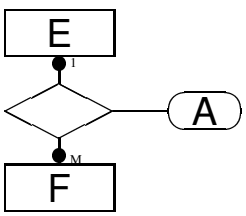
$$= ((\#F \hookrightarrow \#E \times A)\times (\#E \hookrightarrow \cdots)\times (\#F \hookrightarrow \cdots))_{\phi} \quad (9)$$

Local invariant:

$$\phi(\rho,\sigma,\sigma') \stackrel{\text{def}}{=} \forall x,x' \in dom(\rho) : x \neq x' \Rightarrow \pi_1(\rho(x)) \neq \pi_1(\rho(x'))\wedge dom(\rho) = dom(\sigma')\wedge 2^{\pi_1}(rng(\rho)) = dom(\sigma)$$

M:1 Relationships Again 4 rules, in the same order:

1.

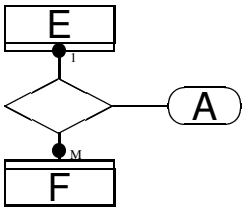


$$= ((\#F \hookrightarrow \#E \times A)\times (\#E \hookrightarrow \cdots)\times (\#F \hookrightarrow \cdots))_{\gamma} \quad (10)$$

Local invariant:

$$\gamma(\rho,\sigma,\sigma') \stackrel{\text{def}}{=} dom(\rho) \subseteq dom(\sigma')\wedge 2^{\pi_1}(rng(\rho)) \subseteq dom(\sigma)$$

2.

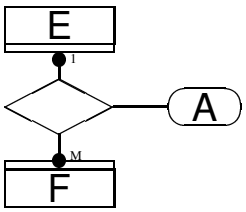


$$= \quad ((\#F \hookrightarrow \#E \times A)\times \quad (\#E \hookrightarrow \cdots)\times \quad (\#F \hookrightarrow \cdots))_{\gamma} \tag{11}$$

Local invariant:

$$\gamma(\rho, \sigma, \sigma') \stackrel{\text{def}}{=} dom(\rho) \subseteq dom(\sigma') \wedge 2^{\pi_1}(rng(\rho)) = dom(\sigma)$$

3.

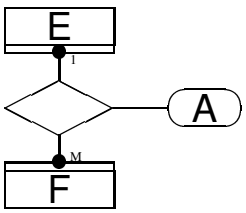


$$= \quad ((\#F \hookrightarrow \#E \times A)\times \quad (\#E \hookrightarrow \cdots)\times \quad (\#F \hookrightarrow \cdots))_{\gamma} \tag{12}$$

Local invariant:

$$\gamma(\rho, \sigma, \sigma') \stackrel{\text{def}}{=} dom(\rho) = dom(\sigma') \wedge 2^{\pi_1}(rng(\rho)) \subseteq dom(\sigma)$$

4.



$$= \quad ((\#F \hookrightarrow \#E \times A)\times \quad (\#E \hookrightarrow \cdots)\times \quad (\#F \hookrightarrow \cdots))_{\gamma} \tag{13}$$

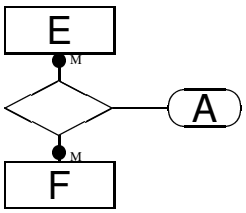
Local invariant:

$$\gamma(\rho, \sigma, \sigma') \stackrel{\text{def}}{=} dom(\rho) = dom(\sigma') \wedge 2^{\pi_1}(rng(\rho)) = dom(\sigma)$$

Note that $1 : 1$ relationships are particular cases of $M : 1$ relationships, only imposing an extra injectivity constraint.

M:M Relationships (in the same order):

1.



$$= \quad ((\#E \times \#F \hookrightarrow A)\times \\ (\#E \hookrightarrow \cdots)\times \\ (\#F \hookrightarrow \cdots))_{\varphi} \tag{14}$$

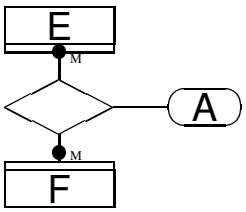
Local invariant:

$$\varphi(\rho,\sigma,\sigma') \stackrel{\text{def}}{=} \quad 2^{\pi_1}(dom(\rho)) \subseteq dom(\sigma) \wedge \\ 2^{\pi_2}(dom(\rho)) \subseteq dom(\sigma')$$

Again, for $A = 1$ (no attributes in relationship) we have

$$\begin{aligned} \#E \times \#F \hookrightarrow 1 &\cong (1+1)^{\#E\times\#F} \\ &\cong 2^{\#E\times\#F} \end{aligned}$$

2.

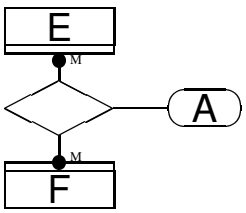


$$= \quad ((\#E \times \#F \hookrightarrow A)\times \\ (\#E \hookrightarrow \cdots)\times \\ (\#F \hookrightarrow \cdots))_{\varphi} \tag{15}$$

Local invariant:

$$\varphi(\rho,\sigma,\sigma') \stackrel{\text{def}}{=} \quad 2^{\pi_1}(dom(\rho)) = dom(\sigma) \wedge \\ 2^{\pi_2}(dom(\rho)) \subseteq dom(\sigma')$$

3.



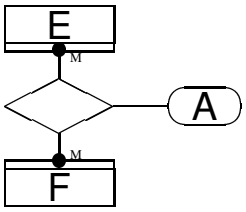
$$= \quad ((\#E \times \#F \hookrightarrow A)\times \\ (\#E \hookrightarrow \cdots)\times \\ (\#F \hookrightarrow \cdots))_{\varphi} \tag{16}$$

Local invariant:

$$\varphi(\rho,\sigma,\sigma') \stackrel{\text{def}}{=} \quad 2^{\pi_1}(dom(\rho)) \subseteq dom(\sigma) \wedge \\ 2^{\pi_2}(dom(\rho)) = dom(\sigma')$$

(notice that this diagram is symmetric wrt. the previous diagram)

4.



$$= \quad ((\#E \times \#F \hookrightarrow A)\times \quad (\#E \hookrightarrow \cdots)\times \quad (\#F \hookrightarrow \cdots))_{\varphi} \tag{17}$$

Local invariant:

$$\varphi(\rho,\sigma,\sigma') \stackrel{\text{def}}{=} \quad 2^{\pi_1}(dom(\rho)) = dom(\sigma) \wedge \quad 2^{\pi_2}(dom(\rho)) = dom(\sigma')$$

An Example: "Toy Factory" Production Database

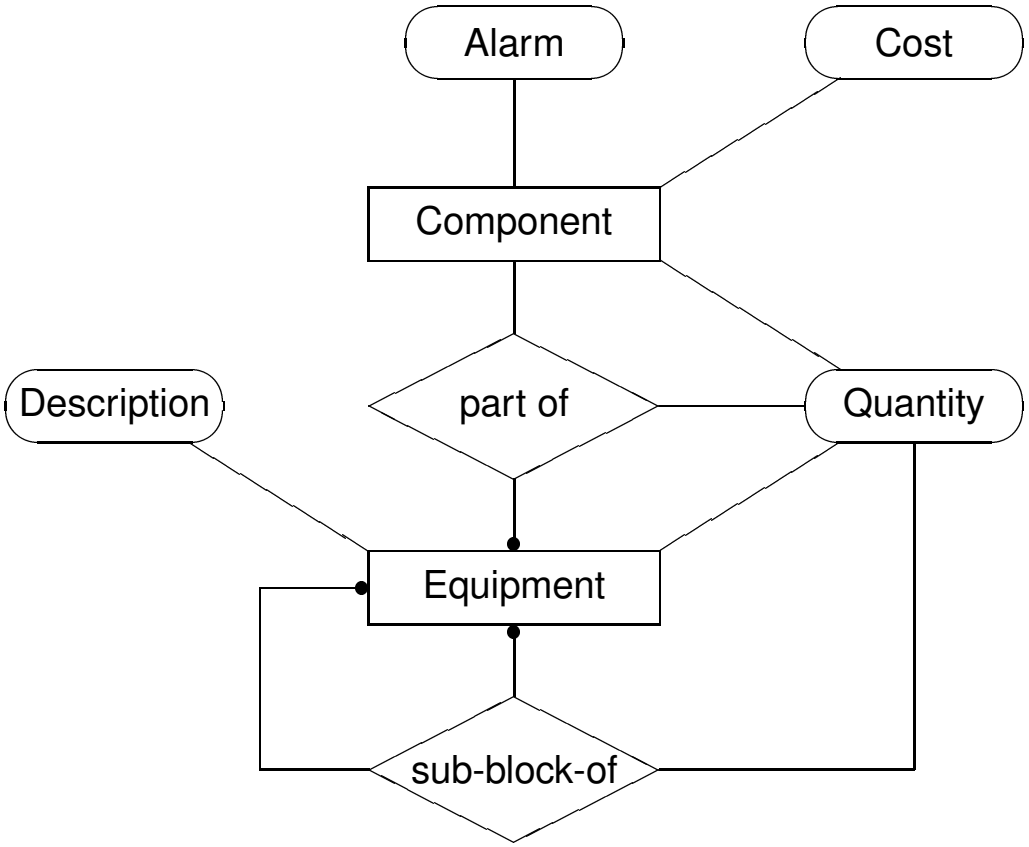
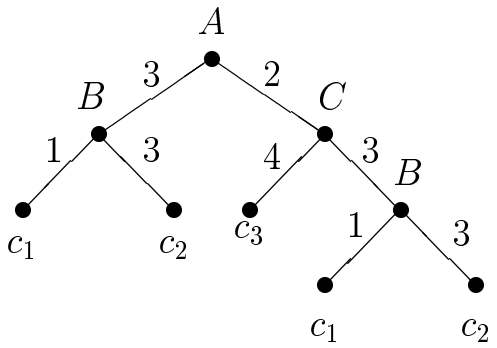


Diagram analysis:

- this factory builds some kind of *equipment* whose production involves individual *components* obtained from a second source;

- each individual component has a *cost* and is stocked in some *quantity* controlled with respect to an arbitrated minimum *alarm* level;
- every component is, in a fixed quantity, *part of* at least one *equipment*;
- equipments may contain, in a fixed quantity, other equipments as *sub-blocks*.

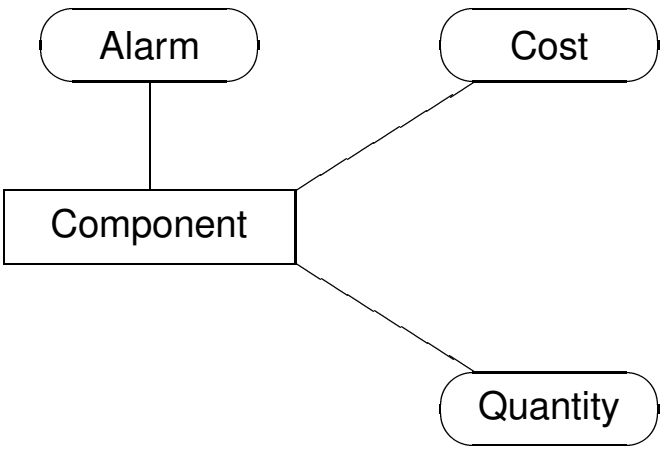
A production tree must exist for each equipment, which may involve individual components and/or other production sub-trees, *e.g.*



There are only two entities in the diagram, $Component$ and $Equipment$. Using rule (5) one obtains:

$$\#Component \hookrightarrow Alarm \times Cost \times Quantity$$

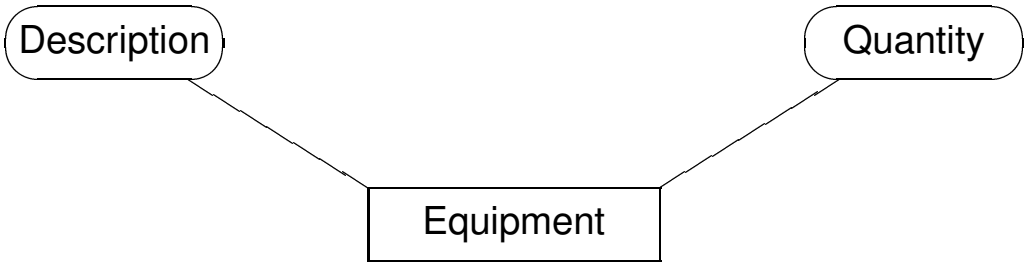
for



and

$$\#Equipment \hookrightarrow Description \times Quantity$$

for



The *part of* relationship can be taken into account by adding, via rule (15), an extra finite mapping:

$$(\#Component \times \#Equipment \hookrightarrow Quantity)$$

Finally, relationship *sub-block-of* is incorporated using rule (17). Final expression is

$$\begin{array}{ll} (& (\#Equipment \times \#Equipment \hookrightarrow Quantity)\times \\ & (\#Component \times \#Equipment \hookrightarrow Quantity)\times \\ & (\#Component \hookrightarrow Alarm \times Cost \times Quantity)\times \\ & (\#Equipment \hookrightarrow Description \times Quantity) \\ )_{\Phi} & \end{array} \tag{18}$$

subject to an overall invariant Φ induced by the rules used above:

$$\begin{array}{rcl} \Phi(r_1, r_2, e_1, e_2) & \stackrel{\text{def}}{=} & 2^{\pi_1}(dom(r_1)) \subseteq dom(e_2) \\ & & \wedge \\ & & 2^{\pi_2}(dom(r_1)) \subseteq dom(e_2) \\ & & \wedge \\ & & 2^{\pi_1}(dom(r_2)) = dom(e_1) \\ & & \wedge \\ & & 2^{\pi_2}(dom(r_2)) \subseteq dom(e_2) \end{array}$$

Independently of this invariant, via the SETS calculus expression (18) can be transformed into

$$\begin{array}{l} ((\#Equipment \times (\#Equipment + \#Component)) \hookrightarrow Quantity)\times \\ (\#Equipment \hookrightarrow Description \times Quantity)\times \\ (\#Component \hookrightarrow Alarm \times Cost \times Quantity) \end{array}$$

The same invariant can be partly lifted via another SETS law finally leading to:

$$
\begin{array}{lll}
(& (\#Equipment \hookrightarrow & Description \times Quantity \times \qquad\qquad\qquad\qquad)\times \\
 & & ((\#Equipment + \#Component) \hookrightarrow Quantity) \\
 & (\#Component \hookrightarrow Alarm \times Cost \times Quantity) & \\
)_{\varphi} & &
\end{array}
\tag{19}
$$

where the remaining invariant is

$$
\begin{array}{rcll}
\varphi(e,c) & \stackrel{\text{def}}{=} & let & X = \bigcup_{k \in dom(e)} dom(\pi_3(e(k))) \\
 & & & E = \{x \in \#Equipment \mid \ i_1(x) \in X\} \\
 & & & C = \{x \in \#Component \mid \ i_2(x) \in X\} \\
 & & in & dom(c) = C \wedge E \subseteq dom(e)
\end{array}
\tag{20}
$$

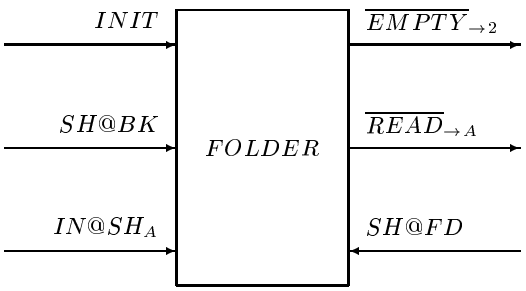
- the formal semantics of given ER-diagram shows it to be, on formal specification grounds, a poor specification of what the specifier might have had in mind but could not express in the diagram.
- For instance, where in (20) is a clause enforcing that *no equipment can recursively be a sub-block of itself*?

$\Rightarrow$ ER (even hierarchical ER) is a relatively "poor" specification device.

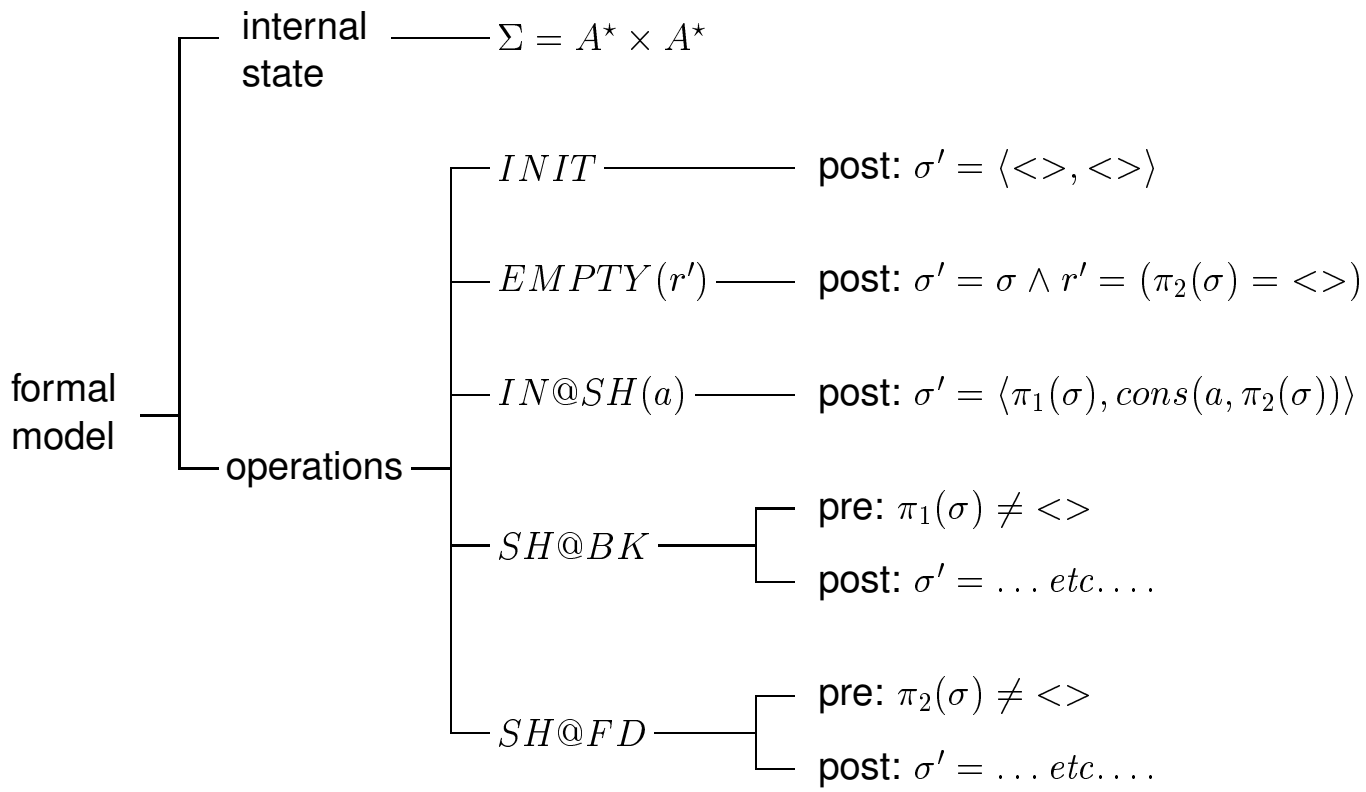
SETS-based Software Formal Comparison Metrics

Formal Software Component

SYNTAX:



SEMANTICS:



Informally:

$INIT$ — initialization;

$IN@SH$ — sheet insertion;

$SH@FD$ — sheet forward;

$SH@BK$ — sheet backwards;

$\overline{READ}$ — show current sheet;

$\overline{EMPTY}$ — check for empty folder.

Two Classification Axes, then:

Classify by . . . —┬— . . . by state + invariant (if any)
                   └— . . . by the operations

- The structure of the internal state of a given model X, denoted by Σ_X, is always a Set expression, *e.g.*

$$\Sigma_{FOLDER} = A^\star \times A^\star$$

- Our attention in this talk will be paid to the "classify by state + invariant" axis, bearing the SETS calculus in mind.

Inheritance versus Classification

We want

$$FOLDER\_OF\_NAT < FOLDER$$

to hold for

$$\Sigma_{FOLDER\_OF\_NAT} = \mathbf{N}^\star \times \mathbf{N}^\star$$

cf. "operation inheritance" (and recall OO stuff . . .):

- Every operation defined on top of an A we don't know anything about will remain mathematically valid once we know that $A = \mathbf{N}$;
- But we can write operations for $FOLDER\_OF\_NAT$ which have no meaning in $FOLDER$, *e.g.*

$$SUM(r') \stackrel{\text{def}}{=} r' = \sum elems(\pi_1(\sigma))$$

In summary, $FOLDER\_OF\_NAT$ *specializes* $FOLDER$.

Formally:

Given two components X and Y, we have

$$Y \leq X$$

(read every "Y *is a* X") if and only if

$$\Sigma_Y = \rho(\Sigma_X) \tag{21}$$

where ρ is a *substitution* of variable symbols by Set-(sub)expressions.

Example

Given

$$\Sigma_Y = (A \times \mathbf{N}) \times B + A$$

and

$$\Sigma_X = B \times A + C$$

we have $Y \leq X$ under substitution

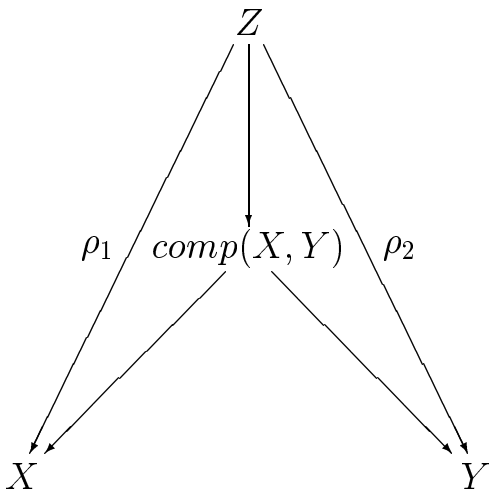
$$\rho = \begin{pmatrix} B & A & C \\ A \times \mathbf{N} & B & A \end{pmatrix}$$

THE COMPARISON FUNCTION:

$comp(X, Y)$ of two given formal models X and Y "along the state-axis" simply is the *unification* of formal expressions Σ_X and Σ_Y.

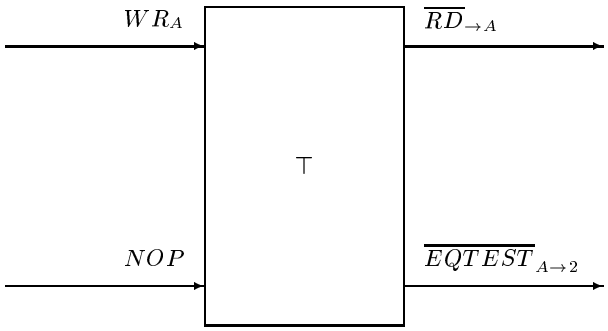
That is, $\Sigma_{comp(X,Y)}$ is the "greatest SETS expression" from which both Σ_X and Σ_Y can be derived by substitutions, *cf.* ρ_1 and ρ_2 (respectively) in the next diagram.

By "greatest" one means that any other generalization Z common to both X and Y will also be a generalization of $comp(X, Y)$.



ABOUT "TOP"

$\top$, the most abstract component, is such that $\Sigma_\top = A$, where A is any SETS variable. $\top$ is a formal model in which very few operations can be specified — its specifies just a (parameterized) "memory cell" where one can but *write*, *read*, *test for equality* or *do nothing*:



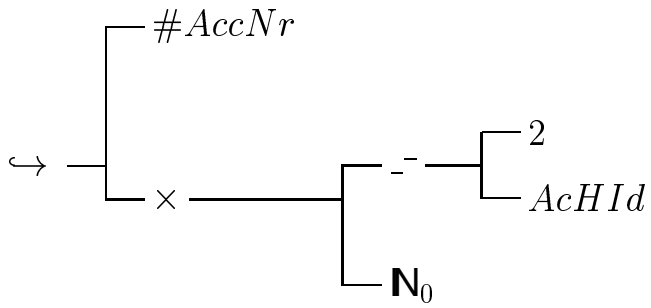
Component State Comparison Examples

Consider the following pair of "toy" components:

- $BAMS$ (*bank account management system*):

$$\Sigma_{BAMS} = \#AccNr \hookrightarrow \begin{array}{ll} 2^{AcHId} & \times \quad \text{/*set of account holders */} \\ \mathbf{N}_0 & \quad \text{/*account balance */} \end{array}$$

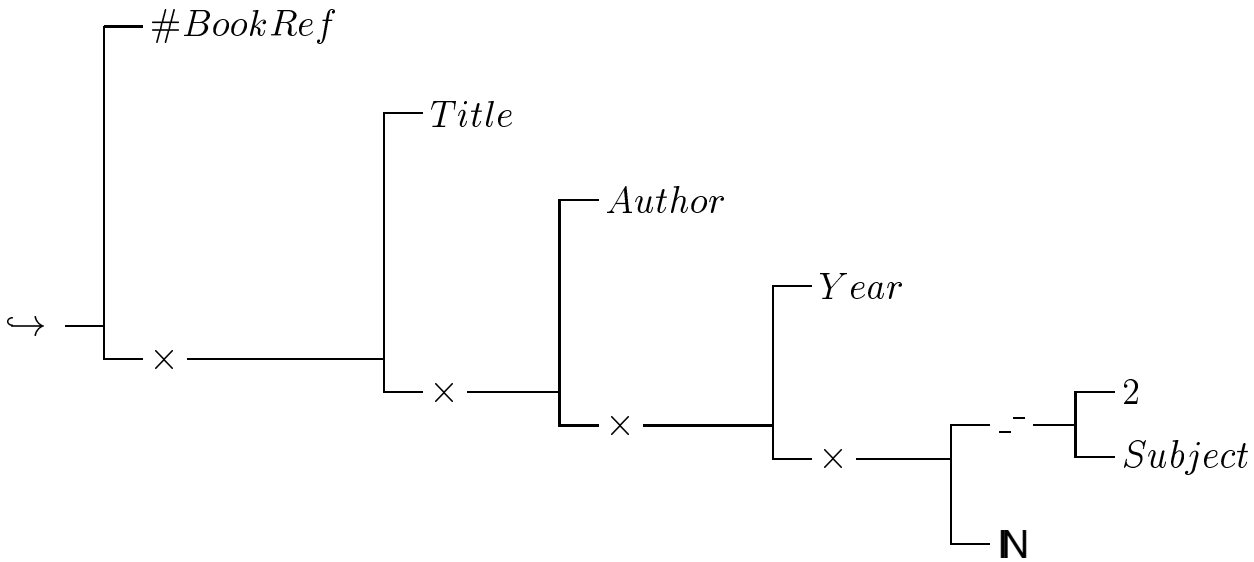
that is, Σ_{BAMS} is SETS expression:



- $LCAT$ (*library catalogue*):

$$\begin{array}{lllll} \Sigma_{LCAT} = \#BookRef \hookrightarrow & Title & \times & \text{/*book title */} \\ & Author & \times & \text{/*book author */} \\ & Year & \times & \text{/*book date */} \\ & 2^{Subject} & \times & \text{/*classification */} \\ & \mathbf{N} & & \text{/*book ISBN */} \end{array} \tag{22}$$

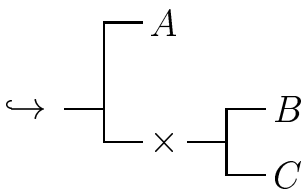
that is, its internal state class is SETS expression:



Then:

$$\Sigma_{comp(BAMS,LCAT)} = A \hookrightarrow B \times C \tag{23}$$

that is,



for substitutions

$$\rho_{BAMS} = \begin{pmatrix} A & B & C \\ \#AccNr & 2^{AcHId} & \mathbf{N}_0 \end{pmatrix}$$

$$\rho_{LCAT} = \begin{pmatrix} A & B & C \\ \#BookRef & Title & Author \times Year \times 2^{Subject} \times \mathbf{N} \end{pmatrix}$$

Of course, the choice of variables A, B and C is arbitrary. Writing $X \hookrightarrow Y \times Z$ would have the same effect as writing $A \hookrightarrow B \times C$. So, strictly speaking, in this approach $\leq$ is a pre-order and not a partial-order.

Is this Enough? No, Bring SETS in...

- The specialization ordering and comparison function of the previous section are too strict and not sufficiently powerful.
- Many components are distinguished which should be found equivalent.
- We don't want to distinguish between two components whose data are "equivalent",

```
struct X {                          struct X {
    A first;                            struct Y {
    struct Y {                              A first;
        B first;        equivalent to       B second;
        C second;                       } first;
    } second;                           C second;
};                                  };
```

(24)

that is:

$$A \times (B \times C) \text{ equivalent to } (A \times B) \times C$$

Therefore, two seemingly different components may actually be "the same" component up to some "valid" transformations.

In the example above, and because $\times$ is not only associative but also commutative, we may subject part of (22) to the following transformations,

$$\begin{aligned}(Title \times Author \times Year) \times 2^{Subject} \times \mathbf{N} &= \\ 2^{Subject} \times (Title \times Author \times Year) \times \mathbf{N} &= \\ 2^{Subject} \times (Title \times Author \times Year \times \mathbf{N})\end{aligned}$$

obtaining

$$\Sigma_{LCAT} = \#BookRef \hookrightarrow 2^{Subject} \times (Title \times Author \times Year \times \mathbf{N})$$

By unifying this version of Σ_{LCAT} with Σ_{BAMS} we obtain

$$\Sigma_{comp(BAMS,LCAT)} = A \hookrightarrow 2^B \times C$$

that is, an intersection "larger" than (23), which captures more of their similarity.

Another Example

Recall, from on-going SETS course, recursive specifications

$$DecTree \cong What \times (Answer \hookrightarrow DecTree) \tag{25}$$

and

$$GenDia \cong Ind \times (GenDia + 1) \times (GenDia + 1) \tag{26}$$

Comments:

- $DecTree$ and $GenDia$ look fairly incomparable data models at first sight;

- They belong to unrelated application domains (*e.g. UI design* versus *demographic software*).

In fact

$$comp(DecTree, GenDia) = \top$$

if criterion (21) is adopted.

But consider the following reasoning in the SETS calculus, where variable symbols T, I, A in

$$T \cong I \times (A \hookrightarrow T) \tag{27}$$

abbreviate the obvious identifiers of (25).

Let $A = 2$. Then

$$\begin{aligned} I \times (A \hookrightarrow T) &\cong I \times (2 \hookrightarrow T) \\ &\cong I \times (T+1)^2 \\ &\cong I \times (T+1) \times (T+1) \end{aligned}$$

— *cf.* Set-laws studied in the companion SETS course.

So, (27) becomes

$$T \cong I \times (T+1) \times (T+1) \tag{28}$$

Now compare (28) against (26): clearly, they are the "same" data model up to variable renaming.

CONCLUSION

We conclude that *genealogical diagrams* "are" specializations of *decision trees*,

$$GenDia \leq DecTree$$

that is, if a package implementing the latter is available, then it may be *re-used* to implement the former.

Of course, a full comparison between both models should examine the operational structure as well — *cf.* observational equivalence, *etc.*— but here we are concerned with classification "along the state-axis" only.

A Broader Specialization (Pre) Ordering

All this leads to a broader specialization (pre) ordering among components which takes the laws of the SETS calculus into account:

$$Y \leq X \quad \text{iff} \quad \exists \rho : \Sigma_Y \cong \rho(\Sigma_X)$$

and, in a formal sense, defines a basic *similarity* relation between formal models able to support their analogical reuse...

...across unrelated domain areas — an important point, recall the purpose of Generic Programming as a whole.

Scaling up the Idea

Many examples of the usefulness of $\cong$-based comparison can be given, providing evidence of considerable overlapping among software data models.

For instance, just let $I = 1$ in the decision tree model (27). Since 1 is the identity of $\times$ we obtain

$$\begin{aligned} T &\cong 1 \times (A \hookrightarrow T) \\ &\cong A \hookrightarrow T \end{aligned}$$

which may be regarded as an abstract data model for subject taxonomies:

$$Tax \cong Subject \hookrightarrow Tax \tag{29}$$

This means that Tax is another specialization of $DecTree$ — a taxonomy is just decision tree with a NIL question at each node.

Also suggestive is the fact that Tax also specializes another general

purpose data model, that of hierarchical *file systems*:

$$\begin{array}{lcll} FS & \cong & Id \hookrightarrow (File + Dir) & \text{/*FS = file system */} \\ & & & \text{/*Id = identifier */} \\ & & & \text{/*File = file contents */} \\ Dir & \cong & FS & \text{/*Dir = directory, i.e. sub-file-system */} \end{array}$$

in short:

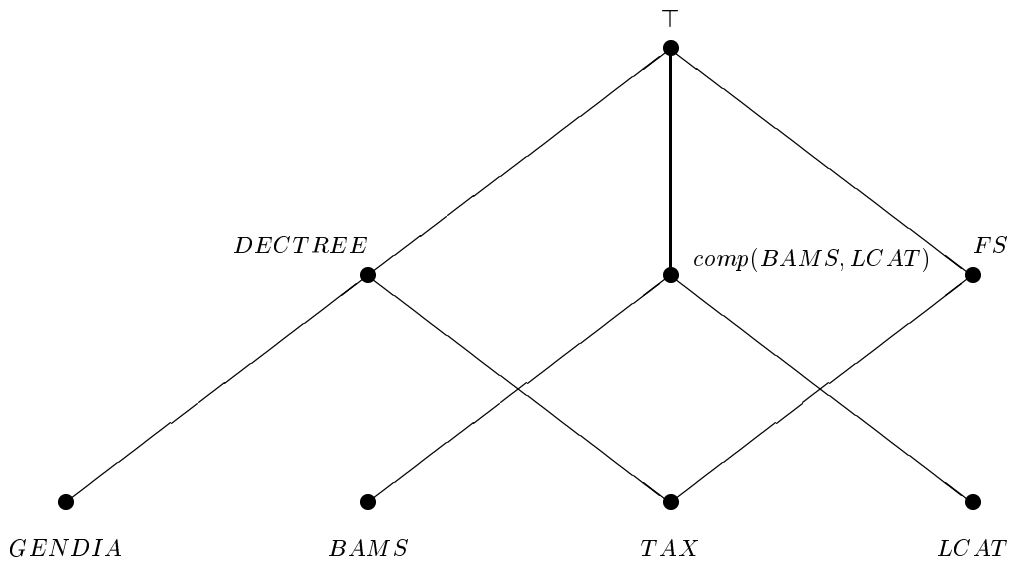
$$FS \cong Id \hookrightarrow (File + FS)$$

Let $File = 0$ and recall that 0 is the identity of $+$:

$$\begin{array}{lcl} FS & \cong & Id \hookrightarrow (0 + FS) \\ & & Id \hookrightarrow FS \end{array}$$

Up to variable renaming, we obtain the same data model as Tax (29), confirming our intuition: if we build a file system with no files at all, such a nested structure of directories is but a taxonomy of (file-system) identifiers.

Putting everything in a repository (diagram):



NB:

- Other intermediate components may exist which are not explicitly recorded in the diagram, which is a typical “is-a” hierarchy with multiple inheritance.
- For instance, any operation specified for FS or $DECTREE$ can — suitably “transformed” via the relevant $\cong$-ismorphism — be inherited by TAX.

State Invariants

They order formal components by logical implication.

Inheritance of state transformers becomes non-monotonic in general.

Final Remarks

- A tentative tool was built in TOOLBOOK ™for assisting ER to SETS translation.
- ENVISION ™is (another) example of a monolithic piece of software which we couldn’t convince to talk with our tool.
- Confidentiality constraints didn’t let us to access to the whole OIS Ricerca information system, but. . .
- . . . the approach proved to be a promising one in the comparison of about 30 company-models extracted from U.Minho Library (workstudy-in-industry report series) — see [Cru93].
- More detailed information is available as a technical report [OC93].

References

[Cru93] A.M. Cruz. Engenharia reversa em larga escala de sistemas de informação. Technical report, University of Minho, Dec. 1993. Workstudy Report at INESC Group 2361 (in Portuguese).

[OC93] J. N. Oliveira and A. M. Cruz. Formal calculi applied to software component knowledge elicitation. Technical Report C19-WP2D, DI/INESC, December 1993. IMI Contract *C.1.9. Sviluppo di Metodologie, Sistemi e Servizi Innovativi in Rete*.

[Oli95] J. N. Oliveira. Fuzzy object comparison and its application to a self-adaptable query mechanism. In *IFSA'95*, volume I, pages 245–248, 22–28 July 1995. Proc. of the 6th International Fuzzy Systems Association World Congress, S. Paulo, Brazil.

[PD91] R. Prieto-Diáz. Implementing faceted classification for software reuse. *CACM*, 34(5):89–97, May 1991.

[PDF87] R. Prieto-Diáz and P. Freeman. Classifying software for reusability. *IEEE Software*, 4(1):6–16, January 1987.