PB - 191

B.H. Mayoh: Galleries and Institutions

TRYK: DAIMI/RECAU

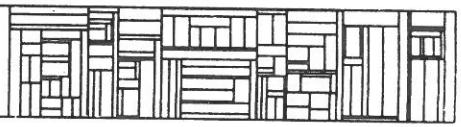
ISSN 0105-8517

Galleries and Institutions

Brian H. Mayoh

DAIMI PB - 191
March 1985

Computer Science Department
AARHUS UNIVERSITY
Ny Munkegade – DK 8000 Aarhus C – DENMARK
Telephone: 06 – 12 83 55



Galleries and Institutions

Abstract:

This preprint has four parts:

(1) Unified theory of knowledge representation (presented at AIMSA 84 conference in Varna).

(2) Unified theory for modal, dynamic, temporal and process logics (presented at PUC conference in Rio de Janeiro).

(3) Unified theory for logical programming and semantic representation (presented at Copenhagen workshop).

(4) Unified theory of languages, models and logics.

The "unified theory" in the titles of each part refers to the theory of galleries, a development of the theory of institutions. These theories seem to be relevant in many areas of computer science: (1) - (3) are applications to particular areas and papers on:

- processes, event systems, Petri nets and other forms of parallelism
- specification, program development and design languages

are under preparation.
The current status of the general theory is given in part 4, but it is not yet clear how this theory should be developed further.

UNIFIED THEORY OF KNOWLEDGE REPRESENTATION

Brian H. Mayoh
Aarhus University

In any artificially intelligent program knowledge of the external world must be represented. Many formal languages for representing knowledge have been invented and used in recent years, but there is a need for a unified theory in which one can

- prove general results that hold for many particular knowledge representation languages;
- transfer results for one knowledge representation language to other languages;
- combine knowledge represented in one language with knowledge represented in other languages.

In the first, second and last section of this paper we present a unified theory; in section 3 we indicate how the theory includes typical knowledge representation languages; in section 4 we indicate how the theory include Montague's formalisation of natural language. The motivating example in section 1 is an algebraic language for relational data bases.

The key concept of our unified theory, the concept of a gallery has a long history. Many years ago logicians generalised the study of models of first order theories to "soft model theory", the study of models of families of logics. Recently this concept of a logical family has been generalised to that of an institution [G].

Galleries are a further generalisation of institutions, in which one can have general values of expressions (terms) not just sentences with truth values.

#1 Signature, Frames, Structures and Galleries

In this section we gradually develop the concept of a gallery by introducing the concepts of signature, frame and structure. The development is motivated by the data base example in figure 1.

A particular relational data base is given by a "conceptual schema" with domains of object types and names of relations on these domains. Such a conceptual schema is an example of a sorted signature Σ where one has a set of sorts and a family $\Sigma(S)$ of relation names for each set S of sorts. In figure 1 Σ(NAME,ADDRESS) and Σ(ADDRESS,PRICE,SIZE) have one element and the other families $\Sigma(S)$ are empty.

Signature — domains: NAME, ADDRESS, PRICE, SIZE

relations: PERSONS ⊂ NAME, × ADDRESS
HOUSES ⊂ ADDRESS × PRICE × SIZE

Two frames

e1: Join (PERSONS, HOUSES)

e2: Project (PERSONS, NAME)

Two structures

m1:

PERSONS	
NAME	ADDRESS
Brian	Ry 66
Pia	Ry 66

HOUSES		
ADDRESS	PRICE	SIZE
Ry 66	750000	Stor
Alken 12	200000	Lille

$m1_{NAME} = m1_{ADDRESS} = m1_{SIZE} = WORD \quad m1_{PRICE} = INTEGER$

m2:

PERSONS	
NAME	ADDRESS
Brian	Ry 66
Pia	Bra 93

HOUSES		
ADDRESS	PRICE	SIZE
Ry 66	750000	Stor
Alken 12	200000	Lille
Bra 93	400000	Middel

$m2_{NAME} = m2_{ADDRESS} = m2_{SIZE} = WORD \quad m2_{PRICE} = INTEGER$

Evaluation

Val(e1,m1) =

NAME	ADDRESS	PRICE	SIZE
Brian	Ry 66	750000	Stor
Pia	Ry 66	750000	Stor

Val(e1,m2) =

NAME	ADDRESS	PRICE	SIZE
Brian	Ry 66	750000	Stor
Pia	Bra 93	400000	Middel

Val(e2,m1) =

NAME
Brian
Pia

= Val(e2,m2)

Figure 1: Relational Data Base Example

In the algebraic approach to relational data bases one can define new relations from old using algebraic oeprations. We can suppose that the expressions for defining new relations are given by

```
<exp> ::=  Persons | Houses |
           Union        (<exp>,<exp>)|
           Intersection (<exp>,<exp>)|
           Join         (<exp>,<exp>)|
           Project      (<exp>,<list>)|
           Select       (<exp>,<list>)|

<list> ::= NAME|ADDRESS|PRICE|SIZE|<list>,<list>
```

This grammar is an example of the definition of the Σ-frames for a signature Σ.

The contents of our data base will change when new houses are built and people buy and sell houses. At any time m we have a set m_{NAME} of names, a set $m_{ADDRESS}$ of addresses, a set m_{PRICE} of prices, a set m_{SIZE} of sizes, a relation $m_{PERSONS} \subset m_{NAME} \times m_{ADDRESS}$, and a relation $m_{HOUSES} \subset m_{ADDRESS} \times m_{PRICE} \times m_{SIZE}$. We may also have data base operations of insertion and deletion, which change the contents m of the data base to some m'. This is an example of the definition of the Σ-structures and Σ-structure-morphisms for a signature Σ.

The value of a data base expression e is a relation that depends on the contents m of the data base; it is given by the usual function for evaluating algebraic expressions in a data base. This is an example of an evaluation function which gives a value to each Σ-frame in each Σ-structure.

Definition A discrete gallery consists of a set SIGN of signatures, a set FRM(Σ) of Σ-frames, a category STR(Σ) of Σ-structures and an evaluation function Val_Σ:FRM(Σ) × STR(Σ) → SET for each Σ ∈ SIGN.

The data base D in figure 1 is closely related to the data base D' in figure 2, because D' is given by renaming relations and domains, then adding a new relation. As figure 2 shows, this operation takes each D-frame e into a D'-frame e' and each D'-structure m' into a D-structure m, in such a way that: Val(e,m) = Val'(e',m'). This is an example of a signature morphism φ from Σ to Σ' generating a function FRM(φ):FRM(Σ) ⟶ FRM(Σ') and a functor STR(φ) from STR(Σ') to STR(Σ), such that $Val_\Sigma(e,STR(\phi)m') = Val_{\Sigma'}(FRM(\phi)e,m')$.

Definition A gallery consists of a category SIGN of signatures, a functor FRM from SIGN to some category of sets, a functor STR from SIGN to some category of categories, and an evaluation function Val_Σ:FRM(Σ) × STR(Σ) ⟶ SET for each Σ ∈ SIGN such that

$$Val_\Sigma(e,STR(\phi)m') = VAL_{\Sigma'}(FRM(\phi)e,m')$$

for each signature morphism φ from Σ to Σ', each e ∈ FRM(Σ) and each m' ∈ STR(Σ').

Comment An institution (G) is a gallery such that $Val_\Sigma(e,m)$ ∈ (true,false) for each signature Σ, each e ∈ FRM(Σ) and each m ∈ STR(Σ).

Signature Σ' domains: N,A,P,S
relations: PP⊂N × A,HH⊂A × P × S,VV⊂N × S

Two frames e1': join (PP,HH)
e2': project(PP,N)

Two structures

m1':

PP	
N	A
Brian	Ry 66
Pia	Ry 66

HH		
A	P	S
Ry 66	750000	Stor
Alken 12	200000	Lille

VV	
N	S
Brian	Stor
Pia	Stor

$m1'_N$ = INTEGER $m1'_N = m1'_A = m1'_S$ = WORD

m2:

PP	
N	A
Brian	Ry 66
Pia	Bra 93

HH		
A	P	S
Ry 66	750000	Stor
Alken 12	200000	Lille
Bra 93	400000	Middel

VV	
N	S

$m2'_P$ = INTEGER $m2'_N = m2'_A = m2'_S$ = WORD

Morphisms For Σ in figure 1, renaming gives a signature morphism ϕ from Σ to Σ' such that:

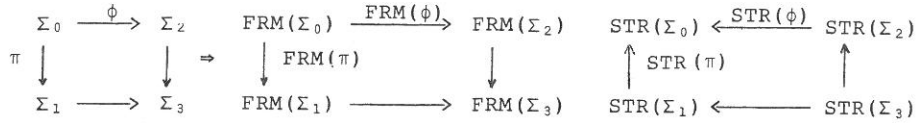
FRM(ϕ) e1 = e1',FRM(ϕ) e2 = e2'
STR(ϕ) m1' = m1,STR(ϕ) m2' = m2

Evaluation function

Val'(e1',m1') = Val(e1,m1), Val'(e1',m2') = Val(e1,m2)
Val'(e2',m1') = Val'(e2',m2') = Val(e2,m1) = Val(e2,m2)

Figure 2: Morhpisms of Relational Data Bases

One reason why signature morphisms are important is that they capture parametrization; one often has a formal parameter signature Σ_0, an actual parameter signature Σ_1, a parametrized signature Σ_2 and parametrization diagrams



In most useful galleries the category SIGN has pushouts so Σ_3 is determined by the signature morphisms ϕ and π.

#2 Specifications and Theories

The contents of the data base D in figure 1 may vary in time but "constraints" can restrict them to a particular class M of D-structures. Two data base expressions, e_1 and e_2, may or may not satisfy the condition

$$\forall m \in M.\ Val(e_1,m) = Val(e_2,m)$$

This condition gives an equivalence relation that is the theory determined by M.

Definition Let $M \subset STR(\Sigma)$ for some signature Σ in a gallery <SIGN,FRM,STR,Val>. The theory determined by non-empty M is the equivalence relation on $FRM(\Sigma)$ with the equivalence classes

$$[e]_M = \{e' \in FRM(\Sigma) \mid Val(e',m) = Val(e,m) \text{ for all } m \in M\}$$

A true sentence of this theory is a frame $e \in FRM(\Sigma)$ such that $\forall m \in M.\ Val(e,m) = true$.

Comment If a theory has a true sentence, then one of its equivalence classes consists of all the true sentences.

Example For the data base in figure 1 "Project(join(persons, houses),name)" is equivalent to "Project(persons),name" in the theory determined by M = {m1,m2}.

Definition For any signature Σ in a gallery <SIGN,FRM,STR,Val> the theory determined by the empty subset of $STR(\Sigma)$ is the "Universal" equivalence relation with $[e]_\phi = FRM(\Sigma)$ as its only equivalence class.

The reason for the last definition is that it gives a connection between subsets of $STR(\Sigma)$ and theories with such useful properties as

$$[e]_M = \{[e]_m \mid m \in M\}$$

$$\phi \subseteq m1 \subseteq m2 \subseteq STR(\Sigma) \rightarrow [e]_{STR(\Sigma)} \subseteq [e]_{m2} \subseteq [e]_{m1} \subseteq [e]_\phi$$

This connection is a natural generalisation of the more usual Galois connection in which theories are identified with their set of true sentences.

Definition The specification determined by non-empty $E \subset FRM(\Sigma)$ in a gallery <SIGN,FRM,STR,Val> is the equivalence on $STR(\Sigma)$ with the equivalence classes

$$[m]_E = \{m' \in STR(\Sigma) \mid Val(e',m) = Val(e,m') \text{ for all } m \in M\}$$

Comment For each result about theories there is a dual result about specifications.

#3 Knowledge Representation Language

In this section we indicate how our unified theory captures A.I. languages for knowledge representation by looking at two examples: Sowa's conceptual structures [S] and Wolfengang's frames [W]. The representation of "a monkey eating a walnut with a spoon made out of the walnut shell" in the two languages is shown in figure 3.

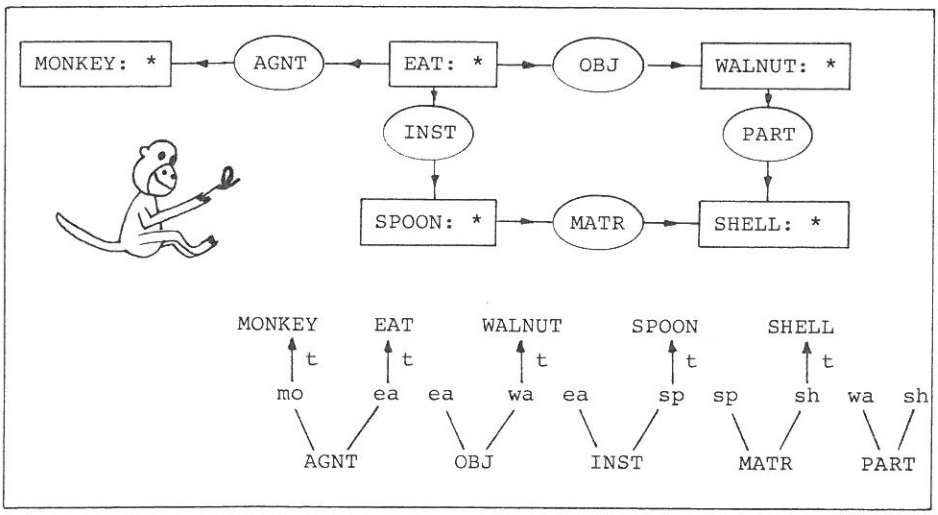


Figure 3: Two Frames with the Same Meaning

The signature Σ of a Sowa representation consists of (1) a set C of "concept types" partially ordered by "is a subconcept of" (2) a family Σ_c of individual names for each $c \in C$ (3) disjoint families $\Sigma(c_1,c_2,\ldots)$ of relation names for each sequence $(c_1,c_2,\ldots) \in C^*$.

The Σ-frames are given by

- instances of relations in the signature
- restrictions of Σ-frames in which concept types are replaced by subconcept types
- joins of frames in which matching concept types are identified.

Figure 4 shows instances of signature relations and restrictions of Σ-frames; the join of its five restrictions is the frame in figure 3. Sowa [p. 92] also allows copying and simplification of frames, but this is not necessary when frame joining is defined appropriately.

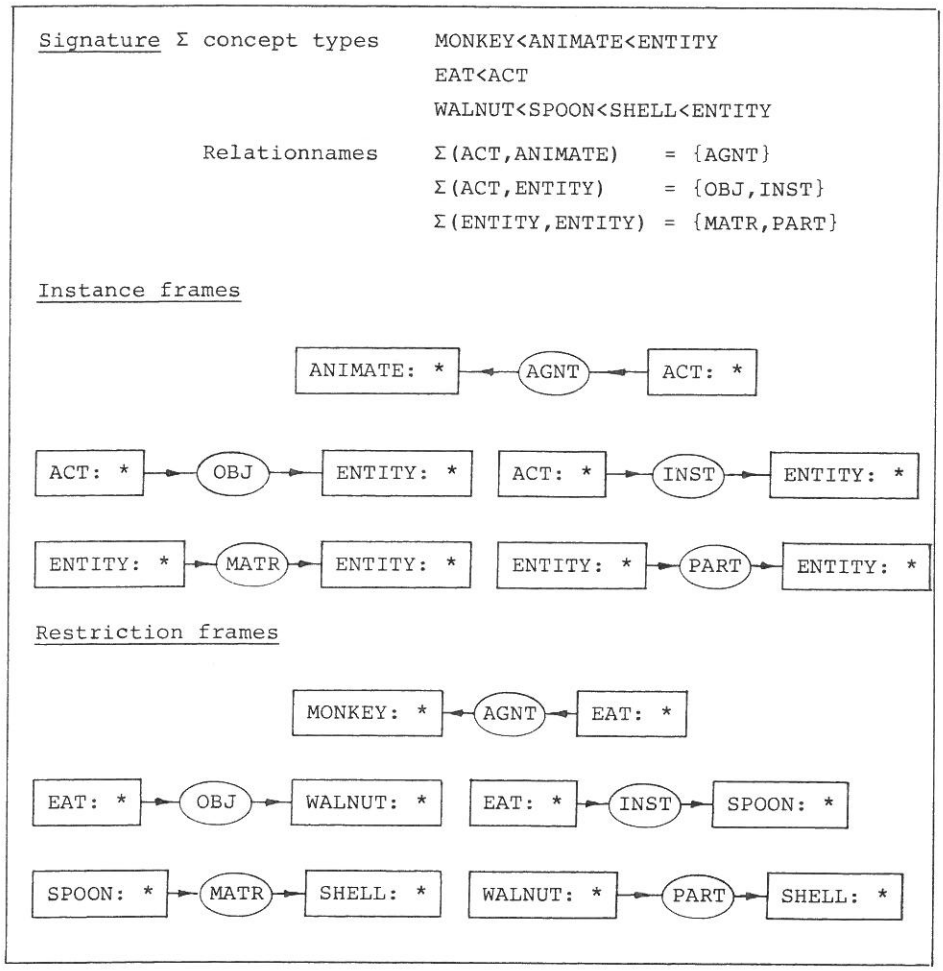


Figure 4: Sowa Representation Language

The Σ-structures are given by (1) a set m_c for each concept type c such that c1<c2 implies $m_{c_1} \subseteq m_{c_2}$ (2) an element $m_i \in m_c$ for each individual name $i \in \Sigma_c$ (3) a relation $m_p \subseteq m_{c_1} \times m_{c_2} \ldots$ for each $p \in \Sigma(c_1, c_2 \ldots)$.

An assignment to a Σ-frame e in a Σ-structure m gives an element of m_c to each [c: *] in e. The natural inductive definition gives either "e is true in m for the assignment a" or "e is false in m for the assignment a". We can define Val(e,m) as the set of assignments that make e true in m. When there are no *-concepts in e, there is a unique assignment a_o to e in m; if e is true in m, then Val(e,m) = $\{a_o\}$; if e is false in m then Val(e,m) is the empty set; Σ-frames without *-concepts are Σ-sentences.

Our Σ-specifications correspond to Sowa's canons, sets of canonical graphs; our Σ-structures correspond to his closed worlds; for simplicity we have ignored his open worlds and replaced his game semantics for modal and higher order logic by traditional first order semantics. We will make similar simplifications in our description of the gallery for the knowledge representation language in [W].

A signature Σ of a Wolfengangen representation consists of (1) a set C of sorts such that $c_1 \to c_2, c_1 \times c_2 \in C$ for all sorts $c_1, c_2 \in C$ (2) disjoint families Σ(c) of function names for each sort $c \in C$. The Σ-frames are given by

- instances of function names in the signature
- variables for each sort
- $\langle t_1, \ldots, t_n \rangle$ where $t_1 \ldots t_n$ are Σ-frames
- (s,t) where s is a Σ-frame of sort $c_1 \to c_2$ and t is a frame of sort c_1
- $(\lambda y.t)$ where t is a Σ-frame and y is a variable.

Figure 5 illustrates some of these ways of building new frames from old; the product of its ten frames is given in figure 3.

Signature	Σ sorts:	ENTITY,ACT,ACT → ENTITY ...	
	Functionnames:	Σ(ACT × ENTITY → 2)	= {AGNT,OBJ,INST}
		Σ(ENTITY × ENTITY → 2)	= {MATR,PART}
		Σ(ACT → 2)	= {EAT}
		Σ(ENTITY → 2)	= {MONKEY,WALNUT, SPOON,SHELL}
Frames	(MONKEY,mo),(EAT,ea),(WALNUT,wa),(SPOON,sp),(SHELL,sh)		
	(AGNT,<ea,mo>),(OBJ,<ea,wa>),(INST,<ea,sp>)		
	(MATR,<sp,sh>),(PART,<wa,sh>)		

Figure 5: Wolfengangen Representation Language

The Σ-structures are given by (1) a set m_c for each sort c such that $m_{c_1} \times m_{c_2} = m_{c_1} \times m_{c_2}$ and $m_{c_1} \to m_{c_2}$ is the set of functions from m_{c_1} to

to m_{c_2} (2) an element m_i of m_c for each function name $i \in \Sigma_c$. An assignment a to a Σ-frame e in a Σ-structure m gives an element of m_c to each free variable of sort c in e. The natural inductive definition gives a function f_a for each <e,m,a>, so we can define Val(e,m) as the map taking assignment a into function f_a. When there are no free variables in e, there is unique assignment ao to e in m and Val(e,m) is essentially f_{ao}; Σ-sentences are Σ-frames without free variables such that f_{ao} is a truth value.

#4 Natural Languages

In this section we indicate how our unified theory captures Montague grammars [P], one of the most elaborate formal theories of natural language. In this formal theory a natural language expression e has zero, one or more representations as formula in an intensional logic. The meanings of e are given by the values of its representative formula. For simplicity we only consider equational logic here, not full intensional logic.

The signature Σ of a Montague grammar consists of (1) a set C of sorts (2) a family Σ_c of variables for each sort c (3) a family $\Sigma_{c_1,c_2} \to c_0$ of function names for each non-empty sequence of sorts c_0,c_1,c_2 ... The Σ-frames are given by

- variables from the signature
- $\sigma(t_1,t_2 \ldots)$ where $\sigma \in \Sigma c_1,c_2 \ldots \to c_0$, t_1 is frame of sort c_1, t_2 is frame of sort c_2
- $t_1 = t_2$ where t_1 and t_2 are frames of the same sort.

This set of frames is unusual because a Σ-frame always has one sort, but it can have more than one sort. Figure 6 is the Montague version of figures 4 and 5.

Signature Σ	sorts:	ENTITY,ACT,ANIMATE	
	Functionsnames:	Σ(ACT,ENTITY → 2)	= {OBJ,INST}
		Σ(ACT,ANIMATE → 2)	= {AGNT,OBJ,INST}
		Σ(ENTITY,ENTITY → 2)	= {MATR,PART}
	Variables:	Σ, ANIMATE	= {mo,wa,sp,sh}
		Σ, ENTITY	= {wa,sp,sh}
		Σ, ACT	= {ea}
Frames	AGNT(ea,mo),OBJ(ea,wa),INST(ea,sp),MATR(sp,sh),PART(wa,sh)		

Figure 6: Montague Representation

There are two kinds of ambiguity in the Montague approach to natural language. A natural language expression L can be syntactically ambiguous because L can be represented by more than one Σ-frame. Even

if the expression L is syntactically unambiguous, it can be semantically ambiguous because the unique representative e of L can be assigned sorts in several ways. Semantic ambiguity affects our definition of Val(e,m) - assignments in the domain of Val(e,m) may be of different sorts.

The Σ-structures are given by (1) a set m_c for each sort c (2) a function m_f: $m_{c_1} \times m_{c_2} \ldots \to m_{c_0}$ for each f in $\Sigma_{c_1,c_2 \ldots \to c_0}$. An assignment a to a Σ-frame e in a Σ-structure m gives an element of m_c to each variable of sort c in e. The natural inductive definition gives "value of e for the assignment a", so we can define Val(e,m) as the map taking assignment a into value va. When the frame e has the form $t_1=t_2$, then va is a truth value for all a - the value <u>true</u> when assignment a gives the same m-value to t_1 and t_2, the value <u>false</u> when a gives different values to t_1 and t_2.

#5 Gallery Morphisms

In this section we introduce gallery morphisms; the key to our achieving two of the three aims of the unified theory - transferring results from one representation language to another and combining knowledge represented in several languages.

Consider two galleries, G = <SIGN,FRM,STR,Val> and G' = <SIGN',FRM',STR',Val'>. One can argue for the following requirements on a morphism Ψ from G to G':

(1) a function Ψ:SIGN $\to$ SIGN'

(2) functions α_Σ: FRM'($\Psi(\Sigma)$) $\to$ FRM(Σ) for each $\Sigma \in$ SIGN

(3) functors β_Σ: STR(Σ) $\to$ STR'($\Psi(\Sigma)$) for each $\Sigma \in$ SIGN

(4) $Val_\Sigma(\alpha_\Sigma(e'),m) = Val'_{\Psi(\Sigma)}(e',\beta_\Sigma(m))$ for each $\Sigma \in$ SIGN, $m \in STR(\Sigma)$, $e' \in FRM'(\Psi(\Sigma))$

Figure 7 illustrates these requirements by indicating a morphism ϕ from the data base gallery in section 1 to the first knowledge representation gallery in section 3. The reader can check that ϕ satisfies the

<u>Definition</u> A gallery morphism from G to G' consists of (1) a functor Ψ from SIGN to SIGN' (2) a natural transformation α from FRM';Ψ to FRM (3) a natural transformation β from STR to STR';Ψ such that $Val_\Sigma(\alpha_\Sigma(e'),m) = Val'_{\Psi(\Sigma)}(e',\beta_\Sigma(m))$ for $m \in STR(\Sigma)$, $e' \in FRM'(\Psi(\Sigma))$.

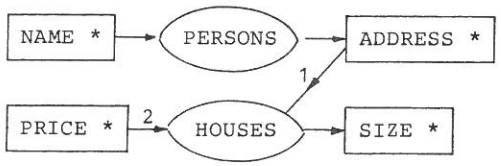
<u>Motivation</u> Gallery morphisms transfer G-signatures, G-frames, G-structures, G-specifications and G-theories into G'-signatures, G'-frames, G'-structures, G'-specifications and G'-theories.

Signatures Σ

domains	NAME,ADDRESS,PRICE,SIZE	NAME,ADDRESS,PRICE,SIZE
relations	PERSONS⊂NAME×ADDRESS	$\Sigma_{NAME,ADDRESS=\{PERSONS\}}$
	HOUSES⊂ADDRESS×PRICE×SIZE	$\Sigma_{ADDRESS,PRICE,SIZE=\{HOUSES\}}$

Frames

join(PERSONS,HOUSES) $\overset{\alpha_\Sigma}{\Leftarrow}$



Structures

m_{PRICE} = Integer

$m_{NAME} = m_{ADDRESS} = m_{SIZE}$ = WORD

PERSONS	
NAME	ADDRESS
Brian	Ry 66
Pia	Ry 66

$\overset{\beta_\Sigma}{\Rightarrow}$

m'_{PRICE} = Integer

$m'_{NAME} = m'_{ADDRESS} = m'_{SIZE}$ = WORD

$m'_{PERSONS}$ = {<Brian, Ry 66>, <Pia, Ry 66>}

m'_{HOUSES} = {<Ry 66,750000,Stor>, <Alken 12,200000,Lille>}

HOUSES		
ADDRESS	PRICE	SIZE
Ry 66	750000	Stor
Alken 12	200000	Lille

Figure 7: A Gallery Morphism

References

[G] J.A. Goguen, R.M. Burstall: Introducing Institutions, Springer LNCS 164, pp. 221-256.

[P] B.H. Partree, ed.: Montague Grammars, Academic Press, 1976.

[S] J.F. Sowa: Conceptual Structures, Addison-Wesley, 1984.

[W] V.E. Wolfengangen: Frame theory and computations, Computers and Artificial Intelligence 3 (1984), pp. 1-30.

UNIFIED THEORY FOR MODAL, DYNAMIC, TEMPORAL AND PROCESS LOGICS

by
Brian H. Mayoh

Many years ago logicians generalised the study of models of first order theories to "soft model theory", the study of models of families of logics. Recently this concept of a logical family has been generalised to that of an institution [GB]. Galleries are a further generalisation of institutions, in which one can have general values of expressions and terms, not just sentences with truth values. In this paper we show that there are natural galleries for modal, dynamic, algorithmic, temporal and process logics. The general theory of galleries is given in [MA]; that paper uses category theory but this does not. We will present galleries by giving

- a set SIGN of signatures,
- a set FRM(Σ) of frames for each $\Sigma \in$ SIGN
- a category STR(Σ) of structures for each $\Sigma \in$ SIGN
- a functor Val_Σ from STR(Σ) to FRM(Σ) $\rightarrow$ Set for each $\Sigma \in$ SIGN

and the proof that our galleries satisfy the "categoric" requirements is omitted. It seems more appropriate here to describe the logical innovations of computer science: programs as new modalities (dynamic logic), sequences and trees of variable assignments (temporal and process logic).

For all the logics we shall consider, a signature consists of a ranked set of function and predicate symbols. "Individual constants" are function symbols of rank 0 and "propositions" are predicate symbols of rank 0. We could have allowed "sorted" signatures, but the presentation of our galleries is simpler if the signatures in SIGN are only ranked, not sorted.

First Order Logic

The frames of the first order logic for any signature Σ are given by the grammar

(1)
$$\begin{aligned}
&\langle\text{variable}\rangle ::= x_1 \mid x_2 \ldots \\
&\langle\text{term}\rangle ::= \langle\text{variable}\rangle \mid \langle\text{function symbol}\rangle(\langle\text{term}\rangle,\ldots,\langle\text{term}\rangle) \\
&\langle\text{formula}\rangle ::= \langle\text{predicate symbol}\rangle(\langle\text{term}\rangle,\ldots,\langle\text{term}\rangle) \mid \underline{\text{false}} \mid \langle\text{term}\rangle,\ldots,\langle\text{term}\rangle \\
&\quad \mid \neg\langle\text{formula}\rangle \mid (\langle\text{formula}\rangle \wedge \langle\text{formula}\rangle) \mid (\langle\text{formula}\rangle \vee \langle\text{formula}\rangle) \\
&\quad \mid \forall\langle\text{variable}\rangle,\langle\text{formula}\rangle \mid \exists\langle\text{variable}\rangle.\langle\text{formula}\rangle
\end{aligned}$$

where Σ gives the function and predicate symbols and the number of terms after such a symbol is given by its rank. These frames are the elements of the set FRM(Σ).

The structures of the first order logic for signature Σ are Σ-algebras - a set M, a function $f_M: M^k \to M$ for each function symbol f of rank k, and a relation $r_M \subset M^\ell$ for each predicate symbol r of rank ℓ. Such structures are the objects in a category STR(Σ), if the morphisms are defined to be algebra homomorphisms [BG]. Suppose we have a Σ-algebra and an assignment or <u>state</u> σ: $\langle$variable$\rangle \to M$. Then each Σ-frame can be given a value by the rules

(2)
$$\begin{aligned}
&[\![x_i]\!]\sigma = \sigma(x_i) \qquad [\![f(t_1,\ldots,t_k)]\!]\sigma = f_M\,([\![t_1]\!]\sigma,..,[\![t_k]\!]\sigma) \\
&[\![r(t_1,\ldots,t_\ell)]\!]\sigma \;.\equiv.\; r_M([\![t_1]\!]\sigma,..,[\![t_\ell]\!]\,\sigma) \qquad [\![\underline{\text{false}}]\!]\,\sigma = 0 \\
&[\![t_1{=}t_2]\!]\sigma \;.\equiv.\; [\![t_1]\!]\sigma = [\![t_2]\!]\sigma \qquad [\![\neg F]\!]\sigma = 1-[\![F]\!]\sigma \\
&[\![(F_1 \wedge F_2)]\!]\sigma = \min([\![F_1]\!]\sigma\,,[\![F_2]\!]\sigma\,) \qquad [\![(F_1 \vee F_2)]\!]\sigma = \max([\![F_1]\!]\sigma\,,[\![F_2]\!]\sigma\,) \\
&[\![\forall x_i\ F]\!]\sigma = \min([\![F\ ]\!]\sigma' \mid \sigma' = \sigma[m/x_i] \text{ for } m \in M) \\
&[\![\exists x_i\ F]\!]\sigma = \max([\![F\ ]\!]\sigma' \mid \sigma' = \sigma[m/x_i] \text{ for } m \in M)
\end{aligned}$$

Notice that each formula gets either 1 or 0 as its value, 1 and 0 are the two truth values of our logics.

To complete our definition of the gallery for first order logic we should define Val_Σ from STR(Σ) to FRM(Σ) $\to$ Set and prove for each signature morphism $\langle \phi^6, \phi^\# \rangle$ that $Val_{\Sigma 1}(\phi^\#(m))e = Val_{\Sigma 2}(m)\phi^6(e)$ for each $e \in$ FRM(Σ1) and $m \in$ STR(Σ2). Here we shall only define Val_Σ by

(3) $\mathrm{Val}\ (m)e = \lambda y_1 \ldots y_n .\ \underline{\mathrm{let}}\ \sigma(x_{i_1}) = y_1\ \underline{\mathrm{and}}\ \ldots\ \underline{\mathrm{and}}\ \sigma(x_{i_n}) = y_n\ \underline{\mathrm{in}}\ [\![e]\!]\sigma$

where $x_{i_1}, \ldots, x_{i_n}$ are the free variables in the frame e and $i(1) < \ldots < i(n)$.

Modal Logic

The frames of the modal logic for signature Σ are given by the grammar (1) with the extra rule

(4) <formula> ::= ◇ <formula> | □ <formula>

The structures of the modal logic for signature Σ are "Kripke universes", Σ-algebras with a binary relation R on states. If we extend (2) by

$$[\![\Diamond\ F]\!]\sigma = \max([\![F]\!]\sigma' \mid \sigma\ R\ \sigma')$$

$$[\![\Box\ F]\!]\sigma = \min([\![F]\!]\sigma' \mid \sigma\ R\ \sigma')$$

then we can still use (3) to define the Val function in the gallery for modal logic.

Different varieties of modal logic are given by placing restrictions on the relation R in Kripke universes. Computer scientists have been particularly interested in the logic of linear time (R must be a linear order) and the logic of branching time (R must be a partial order with a first element), where R is interpreted as "before". They have been still more interested in

Dynamic and Algorithmic Logic

Dynamic logic [FL] and algorithmic logic [SA] are essentially the same - one adds a number of program symbols. The frames of the dynamic logic for signature Σ are given by the grammar (1) with the extra rule

(6) <formula> ::= "<"<program symbol>">"<formula>|[<program symbol>]<formula>

and the structures are Σ-algebras with a binary relation R_p on states for each program symbol p. If we extend (2) by:

$$(7)\qquad \begin{aligned} [\![\langle p\rangle F]\!]\sigma &= \max([\![F]\!]\sigma' \mid \sigma\ R_p\ \sigma') \\ [\![[p] F]\!]\sigma &= \min([\![F]\!]\sigma' \mid \sigma\ R_p\ \sigma') \end{aligned}$$

then we can still use (3) to define the Val function in the gallery for dynamic logic. This logic is important for computer science because the constructions:

$R_{p;q}$ is: $\sigma\ R_p\ \sigma''$ and $\sigma''\ R_q\ \sigma'$ for some σ''

R_{p+q} is: $\sigma\ R_p\ \sigma'$ or $\sigma\ R_q\ \sigma'$

R_{p*} is: $\sigma = \sigma'$ or $\sigma\ R_p\ \sigma''$ and $\sigma''\ R_{p*}\ \sigma'$ for some σ''

capture the basic sequencing, choice and iteration concepts of programming.

Temporal Logic

When a sequential program executes on a computer, it starts in an initial state, passes through many intermediate states, and usually ends in a final state. This is the motivation for temporal logic [PN] with its formulae about state sequences. The frames of the temporal logic for signature Σ are given by the grammar (1) with the extra rule

(8) <formula> ::= ◇ <formula> | □ <formula> | ∘ <formula>

and the structures are just Σ-algebras. For such an algebra and a finite sequence $\sigma_0,\sigma_1,\sigma_2,\ldots,\sigma_m$ of states, which is not empty, we can define a value for each frame by

$$[\![\Diamond F]\!]\sigma_0\sigma_1 \ldots \sigma_m = \max([\![F]\!]\sigma_i \ldots \sigma_m \quad 0\leq i\leq m)$$

$$[\![\Box F]\!]\sigma_0\sigma_1 \ldots \sigma_m = \min([\![F]\!]\sigma_i \ldots \sigma_m \quad 0\leq i\leq m)$$

$$[\![\circ F]\!]\sigma_0\sigma_1 \ldots \sigma_m = [\![F]\!]\sigma_1 \ldots \sigma_m$$

and $[\![e]\!]\sigma_0\sigma_1 \ldots \sigma_n$ for the frames given by (1) is the value $[\![e]\!]\sigma_0$ given by (2). For the definition of the Val function in the gallery for temporal logic we can use,

$$(10) \qquad \begin{aligned} Val((m)e) = \lambda\langle y_{01},\ldots,y_{0n}\rangle \ldots \langle y_{m1} \ldots y_{mn}\rangle \\ \underline{let}\ \sigma_j(x_{i1})=y_{j1}\ \underline{and} \ldots \underline{and}\ \sigma_j(x_{in})\ \underline{in} \\ [\![e]\!]\sigma_0\sigma_1 \ldots \sigma_m \end{aligned}$$

where $x_{i1},\ldots,x_{in}$ are the free variables in e and $i1< \ldots <in$. In the usual formulations of temporal logic there are more connectives than those in (8) and state sequences can be infinite, but the semantics is similar to that we have given.

Process Logic

Process logic [PR] is the natural combination of dynamic and temporal logic. The frames of the process logic for signature Σ are given by the grammar (4) and the rules (6) and (8). The structures of the process logic for signature Σ are Σ-algebras with a set S_p of state sequences for each program symbol p. If we extend (9) by

$$(11) \qquad \begin{aligned} [\![\langle p\rangle F]\!]\sigma_0\sigma_1 \ldots \sigma_m &= \max([\![F]\!]\sigma_i \ldots \sigma_m \mid \sigma_0 \ldots \sigma_i \in S_p) \\ [\![[p]F]\!]\sigma_0\sigma_1 \ldots \sigma_m &= \min([\![F]\!]\sigma_i \ldots \sigma_m \mid \sigma_0 \ldots \sigma_i \in S_p) \end{aligned}$$

then we can use (10) to define the Val function in the gallery for process logic.

Behaviour Logic

For many years it has been apparent that state sequences are not the right concept of computation for distributed, parallel and concurrent programs. In the literature more appropriate concepts have been suggested: synchronisation trees [MI], net processes [P2]. Each of these suggestions can be represented by a family of sets of state sequences, so the behaviour of a program is given by a collection of families of sets of state

sequences. The logic of such behaviours has not yet been formulated in a totally adequate way, but adequate formulation will probably be a form of modal logic [HE] or type theory.

Schema Logic

Until now we have interpreted predicate and function symbols in the structures of the logic, but it is sometimes convenient to treat these symbols as schema variables and interpret them in the state in the same way that individual variables get a value in a state. In the usual formulation of induction

$$\phi(0) \wedge \forall n(\phi(n) \rightarrow \phi(n')) \rightarrow \forall n\ \phi(n)$$

in Peano arithmetic, the symbol ϕ is a schema variable which is given a first order formula with one free variable in a state. Logics with schema variables have been used by computer scientists for specifying data types [BT], so it is not inappropriate to define the corresponding gallery here.

The signatures and the frames in the gallery for schema logic are the same as those in the gallery for first order logic. The structures for schema logic are the structures for first order logic except that there is no interpretation for schema variables. In the state σ the schema variable ψ is assigned a first order frame e_ψ with the appropriate number of free variables - if ψ is a predicate schema variable, it is assigned a formula; if ψ is a function schema variable, it is assigned a term. The valuation function in the gallery for schema logic is given by adding $\psi_M = Val_\Sigma(m)e_\psi$ given by (3) for each schema variable ψ to the first order rules (2).

Clearly schema logic variables can be added to modal, dynamic, temporal, process and behaviour logic. Temporal logic with predicate schema variables for relations in a data base gives a very natural way of expressing integrity constraints in relational data bases ([CF], [VCF]).

Inductive Logic

Logicians have often studied definition by induction [Mo] because such definitions are very natural. Recently the computational insight behind inductive definition was revealed in [HK] and important applications to the theories of dynamic logic and data base queries were made. Here we will explain inductive logic as a generalization of schema logic in which schema variables are assigned inductive sets not just definable sets. Inductive sets are given by programs in the language IND, by sequences of labelled commands of the forms

l_1 : accept l_2 : reject

l_3 : $x_i \leftarrow \exists$ l_4 : $x_i \leftarrow \forall$

l_5 : if $R(x_1 \ldots x_n)$ then goto l_6

where $R(x_1 \ldots x_n)$ is an atomic formula.

In [HK] several examples of IND programs are given and it is shown that (loop-free) IND programs accept precisely the first order (definable) inductive relations. Although the intuitive idea of "the set accepted by an IND program" is clear, the precise definition is rather complicated. Suppose we have a first order structure $(m, f_m, r_m, \ldots)$, a state σ and an IND program π. For any function f from ProgramLabels $\rightarrow$ States to $\{0, \frac{1}{2}, 1\}$ we can define a new function $\tau(f)(l,\sigma)$ by:

(1) if l labels accept, then $\tau(f)(l,\sigma) = 1$

(2) if l labels reject, then $\tau(f)(l,\sigma) = 0$

(3) if l labels $x_i \leftarrow \exists$, then

$$\tau(f)(l,\sigma) = \max(f(l',\sigma') \mid \sigma' = \sigma[m/x_i] \text{ for } m \in M)$$

(4) if l labels $x_i \leftarrow \forall$, then

$$\tau(f)(l,\sigma) = \min(f(l',\sigma') \mid \sigma' = \sigma[m/x_i] \text{ for } m \in M)$$

(5) if l labels if $R(x_1 \ldots x_n)$ then goto l_6, then

$$\tau(f)(l,\sigma) = \begin{cases} f(l_6,\sigma) & \text{if } [\![R(x_1 \ldots x_n)]\!]\sigma = 1 \\ f(l',\sigma) & \text{if } [\![R(x_1 \ldots x_n)]\!]\sigma = 0 \end{cases}$$

where 1' is the label after 1 in the program π. The predicate ψ_M for the schema variable ψ is given by

$$(*)\quad \psi_M(v_1 \ldots v_n) \equiv \underline{\text{let}}\ \sigma(x_{i1}) = v_1\ \underline{\text{and}}\ \ldots\ \underline{\text{and}}\ \sigma(x_{in}) = v_n\ \underline{\text{in}}\ \hat{f}(l_0,\sigma) = 1$$

where l_0 is the initial label of the program π for ϕ, $x_{i1} \ldots x_{in}$ is a list of the variables in the program, and $\hat{f}$ is the least fixed point of the function τ starting with the function $f_0(l,\sigma) = \frac{1}{2}$.

The gallery for inductive logic is the same as that for schema logic except that the meaning of schema variables is given by (*) not: $\psi_M = \mathrm{Val}_\Sigma(m)e_\psi$. We have followed [HK] is allowing only atomic R in IND commands, but we could have allowed any first order test. One can also allow IND functions and develop very powerful "schematic" versions of modal, dynamic, temporal, process and behaviour logic.

States and Time

It can be argued that the various extensions of first order logic, we have described, are dishonest ways of suppressing state and time variables. First order logic with explicit state and time variables is often used in artificial intelligence and it occasionally appears in the data base literature [VCF]. Formulae of our various logics are translated by

$$\begin{aligned}
\Diamond F(s) &\leftrightarrow \exists s'(s \leq s' \wedge F(s')) \\
\Box F(s) &\leftrightarrow \forall s'(s \leq s' \rightarrow F(s')) \\
\circ F(s) &\leftrightarrow F(\mathrm{next}(s)) \\
\langle p \rangle F(s) &\leftrightarrow \exists s'(R_p(s,s') \wedge F(s')) \\
[p] F(s) &\leftrightarrow \forall s'(R_p(s,s') \rightarrow F(s')) \\
R_{p;q}(s,s') &\leftrightarrow \exists s''(R_p(s,s'') \wedge R_q(s'',s')) \\
R_{p+q}(s,s') &\leftrightarrow R_p(s,s') \vee R_q(s,s')
\end{aligned}$$

where F(s) is a first order formula with free variable s and F(s') is the result of substituting s' for s. One captures temporal and process logic if one lets the "index" variable s range over sequences or trees of states, not just single states.

The reader may well be worried by the fact that our translation only applies to formulae F(s) with at most one free variable. Most of the formulae that arise in practice have this property because (1) every function symbol f and predicate symbol has one argument of "index" sort, (2) forgetting the index sort in the indexed structure I gives a structure M such that

$$[\![f(t_1,\ldots,t_k,s)]\!]_I = [\![f(t_1,\ldots,t_k)]\!]_M(s)$$
$$[\![r(t_1,\ldots,t_k,s)]\!]_I \equiv [\![r(t_1,\ldots,t_k)]\!]_M(s)$$

for all frames $t_1,\ldots,t_k$ of the appropriate sort. However, it is not clear that our translation into first order logic is intuitively correct: what are the appropriate axioms for $\leq$ and next( ); the translation of <p;q> assumes "angelic" non-determinism; how should <p*> be translated? Not only are modal, dynamic, temporal and the other logics very natural extensions of first order logic, but they also allow computer scientists to express clearly and concisely the most important properties of programs and processes.

References

[BT] J.A. Bergstra & J. Terlouw: Standard model semantics for DSL - a data type specification language, Acta Inf. 19 (1983) 97-113.

[CF] M.A. Casanova & A.L. Furtado: On the description of data base transition constraints using temporal languages - Advances in Data Base Theory, vol. II: H. Gallaire, J. Minker and J.M. Nicolas (eds.), Plenum Press 1983.

[FL] M.J. Fisher, R.E. Ladner: Propositional dynamic logic of regular programs. J. Comp. Sci. 8 (1979) 194-211.

[GB] J.A. Goguen, R.M. Burstall: Introducing Institutions, Springer LNCS 164 (1984) 221-256.

[HE] M. Hennessy: Axiomatising finite delay operators. Acta Informatica (1984).

[HK] D. Harel & D. Kozen: A programming language for the inductive sets, and applications. Proc. ICALP82, Springer LNCS 140 (1982) 313-329.

[MA] B.H. Mayoh: Unified theory of languages, models and logics, to appear.

[MI] R. Milner: A calculus of communicating behaviour, Springer LNCS 92 (1980).

[Mo] Y.N. Moschovakis: Elementary induction on abstract structures, North-Holland 1974.

[NPW] M. Nielsen, G. Plotkin, G. Winskel: Petri nets, event structures and domains, Springer LNCS 70 (1979).

[P1] V.R. Pratt: Process Logic, 6 POPL Symposium (1979) 93-100.

[P2] V.R. Pratt: Mathematics of parallel processes, Proc. IFIP 83 (1983).

[PN] A. Pnueli: The temporal logic of programs, 18 FOCS Symposium (1977) 46-67.

[SA] A. Salwicki: Formalized algorithmic languages. Bull. Acad. Pol. Sci. Ser. Sci. Math. Astr. Phys. 18 (1970).

[VCF] P.A.S. Veloso, M.A. Casanova, A.L. Furtado: Formal Data Base Specification - and eclectic perspective. PUC Monograph 1 (84).

Unified Theory for Logical Programming and Semantic Representation

Brian H. Mayoh
Computer Science Department
Aarhus University
Aarhus, Denmark

Logical programming languages can be considered as semantic representation languages, and logic programs can be considered as theories, knowledge about some model of the real world. One can also consider scientific papers, lectures and textbooks as theories, expressed in informal natural language. In this paper we compare the expressive power of various semantic representation languages. The results are shown in Figure 1, where an arrow from L_1 to L_2 means: L_1 is not more expressive than L_2.

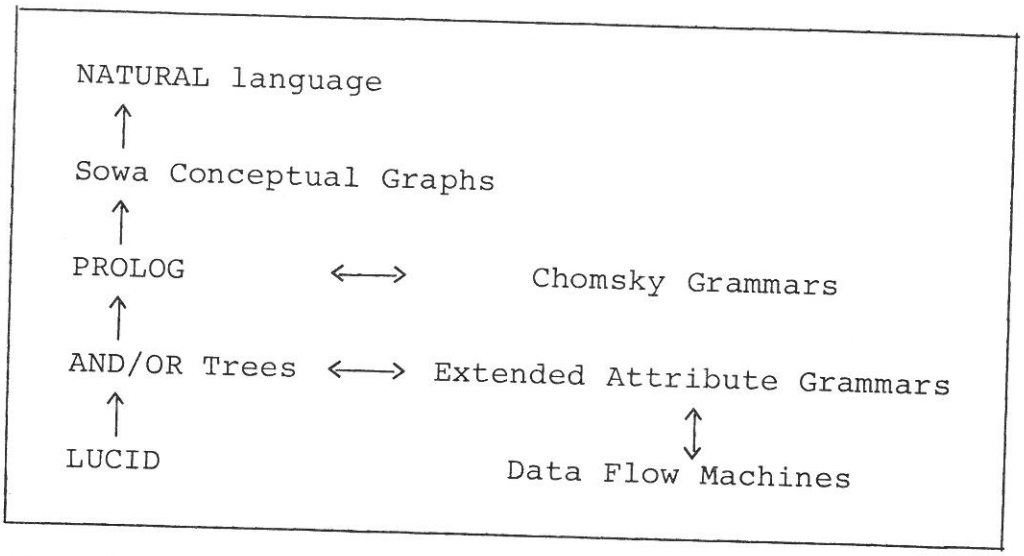


Fig. 1. Expressive Power of some Semantic Representation Languages

If we can give a precise meaning to "equivalence of theories", we can also give a precise meaning to an arrow from L_1 to L_2 in Figure 1:

- every L_1-theory can be translated into an equivalent L_2-theory.

One can distinguish between (1) two theories are weakly equivalent because they have the same conclusions (models, results) (2) two theories are strongly equivalent because they have the same proofs (deductions, computations). The distinction between weak and strong equivalence is the same as that

- of the computer scientist between denotational and operational semantics
- of the logician between model theory and proof theory
- of the linguist between semantics and syntax.

Note however that both kinds of equivalence presuppose a theory of meaning such that "equivalent theories have the same meaning". In this paper we choose strong equivalence and we give a unified theory of various semantic representation languages by devising a "gallery" for each language and "gallery morphisms" for translations between languages.

#1 Galleries

In a gallery one has signatures, frames, structures, and valuation functions. The galleries, we shall present for PROLOG and our other semantic representation languages, will have the same signatures and structures, but their frames and valuation functions will be different. For each of our galleries, the structures will be non-linguistic "possible worlds", but the signatures are always "vocabularies" - linguistic objects like those in Figure 2. Each structure gives a meaning to each linguistic object in its signature; these meanings can be objects, functions, relations or sets.

Σ_1 =

i,j,n,r : INTEGER
+,* : $\text{INTEGER}^2 \rightarrow \text{INTEGER}$
[j>n],[j≦n] $\subseteq \text{INTEGER}^2$

Σ_2 =

mo: MONKEY $\subset$ ENTITY
wa: WALNUT $\subset$ ENTITY
sp: SPOON $\subset$ ENTITY
sh: SHELL $\subset$ ENTITY
ea: EAT $\subset$ ACTIVITY
AGNT,OBJ,INST $\subseteq$ ACTIVITY × ENTITY
PART,MATR $\subseteq$ ENTITY × ENTITY

Fig. 2. Examples of Signatures

The valuation functions in a gallery give a "meaning" to every frame in every structure with the same signature. In each of our galleries the valuation functions will give a set of proofs to every frame in every possible world m with the same signature, and this set will be empty iff m is not a possible world for the frame e.

So far our description of galleries has been imprecise so we should give the

Definition 1 A gallery consists of a set of signatures, SIGN, and for each signature $\Sigma \in$ SIGN

- a set of frames, FRM(Σ)
- a set of structures, STR(Σ)
- a valuation function, Val_Σ: FRM(Σ) × STR(Σ) → Set.

A morphism from a gallery G_1 to a gallery G_2 with the same signatures and the same sets of structures is a set of maps ψ_Σ: $FRM_2(\Sigma) \rightarrow FRM_1(\Sigma)$ such that

$$Val_1(\psi_\Sigma(e),m) = Val_2(e,m)$$

for all $e \in FRM_2(\Sigma)$ and $m \in STR(\Sigma)$.

For this paper the importance of this definition is that: there is a morphism from the gallery for a language L_2 to the gallery for a language L_1 if and only if L_1 is not more expressive than L_2. The informal notion of the last section has been formalized - theories correspond to frames, translations correspond to the functions ψ_Σ, and strong equivalence corresponds to defining Val(e,m) as a set of proofs.

The galleries and morphisms, given by Definition 1, are rather special. In the general theory of galleries [Mal], one has structure morphisms, signature morphisms and more general gallery morphisms. Finding a rich set of signature morphisms for the gallery of a semantic representation language like PROLOG corresponds to finding powerful ways of modularising theories in the language. This is an important area of current research; in time it will lead to the appropriate definition of signature morphism in our galleries.

#2 The PROLOG Gallery

The frames in the PROLOG gallery are finite sets of assertions and definitions like the set in Figure 3. The primitive predicate symbols in a frame are those which do not occur in an assertion or on the left side of a definition. If the variable, function and primitive predicate (relation) symbols of a frame e are included in a signature Σ, then e is a Σ-frame, and it has a meaning for each Σ-structure. A Σ-structure m is a "possible world" where each symbol in Σ has a meaning; a possible world m_1 for the frame in Figure 3a is given by

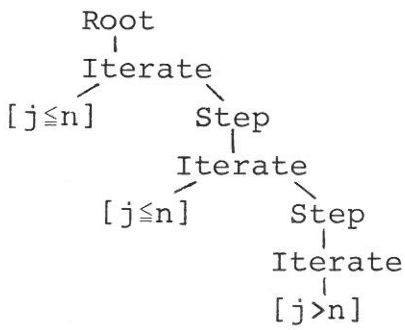
(a) frame

```
Root(n,r)       :- Iterate(0,1,n,r).
Iterate(i,j,n,i) :- [j>n].
Iterate(i,j,n,r) :- [j≤n], Step(i,j,n,r).
Step(i,j,n,r)    :- Iterate(i+1,j+2*i+3,n,r).
```

(b) computation

```
Root(7,2)
Iterate(0,1,7,2)
1≤7   Step(0,1,7,2)
      Iterate(1,4,7,2)
      4≤7   Step(1,4,7,2)
            Iterate(2,9,7,2)
            9>7
```

(c) proof



(d) proof rules

$$\frac{m(0/i,1/j) \vdash \text{Iterate}}{m \vdash \text{Root}} \qquad \frac{m(i+1/i,j+2*i+3/j) \vdash \text{Iterate}}{m \vdash \text{Step}}$$

$$\frac{m \vdash [j>n] \quad m \vdash r = i}{m \vdash \text{Iterate}} \qquad \frac{m \vdash [j\leq n] \quad m \vdash \text{Step}}{m \vdash \text{Iterate}}$$

(e) derivation

```
<6,17,7,2> ⊢ Root
──────────────────
<0,1,7,2> ⊢ Iterate
──────────────────────────────────────
<0,1,7,2> ⊢ [j≤n]   <0,1,7,2> ⊢ Step
                    ──────────────────
                    <1,4,7,2> ⊢ Iterate
                    ──────────────────────────────────────
                    <1,4,7,2> ⊢ [j≤n]   <1,4,7,2> ⊢ Step
                                        ──────────────────
                                        <2,9,7,2> ⊢ Iterate
                          ──────────────────────────────────────
                          <2,9,7,2> ⊢ [j>n]   <2,9,7,2> ⊢ r = i
```

Fig. 3. A PROLOG frame, computation and proof

- i denotes 6, j denotes 17, n denotes 7, r denotes 2
- + denotes "addition"
- * denotes "multiplication"
- [j>n] denotes "true"
- [j≦n] denotes "false"

This possible world has the linguistic representation <6,17,7,2>. In a possible world a Σ-goal is either true or false; in the structure m_1 the goal Root(n,r) is true and the goal Root(n,i) is false.

A PROLOG program consists of a Σ-frame and a Σ-goal for some signature Σ, and the execution of the program will either succeed or fail. If the execution of the program succeeds, it will produce a computation like that in Figure 3b. In the procedural interpretation of PROLOG every computation gives a "trace", a tree of procedure calls; in the logical interpretation of PROLOG every computation gives a "proof" like that in Figure 3c. The axioms and definitions of a PROLOG frame give "proof rules" like those in Figure 3d, and these proof rules give "derivations" like that in Figure 3e. Now suppose we have a computation or proof for a Σ-frame e and m is a Σ-structure. We may or may not be able to expand the proof to a derivation of $m \vdash R$ where R is the top node of the proof. We can define the meaning of the frame e in the possible world m as the set

$$\mathrm{Val}(e,m) = \begin{pmatrix}\text{e-proofs of R that can be expanded to}\\ \text{derivations of } m \vdash R\end{pmatrix}$$

This completes the definition of the PROLOG gallery.

What is the connection between the semantics given by this valuation function and the "official" semantics of PROLOG? Implementations may not be able to find computations that do exist; quite apart from the problems of "cut", the convention - an implementation takes the first matching assertion or

definition and works through a definition from left to right - ensures that some proofs may not be found by an implementation. At each step in its search for a proof, an implementation must choose

- the next assertion or definition
- the substitution instance of this assertion or definition.

This intrinsic double existence quantifier is the reason for the PROLOG problems with negation; it is also the reason why the logical semantics of PROLOG is cleaner than the procedural semantics. In the logical semantics a PROLOG frame is a first order theory, goals are atomic formulae, and goals may or may not be valid (true in all models of the theory). Every model of a theory has an interpretation of the non-primitive predicates in the theory, whereas our Σ-structures take the interpretation of these predicates that the theory gives (the least fixed point interpretation explained in the next section). The logical semantics of PROLOG makes precise our notion of "a goal is either true or false in a possible world", and it agrees with the valuation function in the PROLOG gallery if (1) every derivation gives a valid goal (2) every valid goal has a derivation. The inductive argument for (2) will be given in the next section, and the soundness of first order logic gives (1).

3 The GRAMMAR gallery

As Figure 4 shows, context free grammars become logic programs when each production $X_0 ::= X_1,\ldots,X_n$ is given a proof rule

$$\frac{M_1 \vdash X_1,\ldots,M_n \vdash X_n}{M_0 \vdash X_0}$$

where M_1, M_n, M_0 are substitution instances of the same structure m. A substitution instance of a Σ-structure m is the structure $m(t_1/v_1,\ldots,t_n/v_n)$, given by using Σ-terms $t_1,\ldots,t_n$

to give new values to Σ-variables $v_1,\ldots,v_n$. Now we can define the frames in the GRAMMAR gallery as: context free grammars with a proof rule for each production. The signatures and structures in the GRAMMAR gallery are the same as the signatures and structures in the PROLOG gallery. This is also true of the valuation function because proofs and derivations can be defined as we outlined in the last section. There is an obvious bijection between PROLOG frames and GRAMMAR frames so the galleries are isomorphic - PROLOG has the same expressive power as Chomsky gramamrs with proof rules.

Root ::= Iterate	$\dfrac{m(0/i,1/j) \vdash \text{Iterate}}{m \vdash \text{Root}}$
Iterate ::= [j>n]	$\dfrac{m \vdash [j>n] \quad m \vdash r = i}{m \vdash \text{Iterate}}$
|[j≦n] Step	$\dfrac{m \vdash [j\leq n] \quad m \vdash \text{Step}}{m \vdash \text{Iterate}}$
Step ::= Iterate	$\dfrac{m(i+1/i,j+2*i+3/j) \vdash \text{Iterate}}{m \vdash \text{Step}}$

Fig. 4. A GRAMMAR frame

The natural semantics of a GRAMMAR frame is given by rewriting its proof rules as first order formulae, then taking the least predicates that satisfy the formulae as the meaning of the frame. From the example in Figure 5 it is clear that these least predicates make the goal of every derivation valid. The converse is also true by the inductive argument

(1) a goal is valid iff it is true in the "least predicate" interpretation;

(2) the "least predicate" interpretation is given by starting with the interpretation where predicates are always false and gradually making $P(...)$ true as demanded by the formulae for the frame;

(3) the "least predicate" interpretation is given by the derivations.

This argument shows that the valuation functions in the GRAMMAR and PROLOG galleries give the natural semantics for these galleries.

Theory

Iterate(0,1,n,r)	.→.	Root(n,r)
j>n & r = i	.→.	Iterate(i,j,n,r)
j≦n & Step(i,j,n,r)	.→.	Iterate(i,j,n,r)
Iterate(i+1,j+2*3+i,n,r)	.→.	Step(i,j,n,r)

Least Predicate Model

Root(n,r) is "r = $\sqrt{n}$" by the iteration

Iterate	Step	Root
false	false	false
n<j & r=i	false	false
...	n<j+2i+3 & r=i+1	n<1 & r=0
j≦n<j+2i+3 & r=i+1	...	...
...	j+2i+3≦n<j+4i+8 & i=i+2	1≦n<4 & r=1
j+2i+3≦n<j+4i & r=i+2	...	...
...	j+4i+8≦n<j+6i+15 & r=i+3	4≦n<9 & r=2

Fig. 5. Theory and Interpretation for a GRAMMAR Frame

Comment This inductive argument is possible because GRAMMAR and PROLOG frames give sets of "Horn clauses", so the "least predicate" interpretation is an initial model in the (consistent) theory for the frame. In more expressive galleries one cannot reduce validity to truth in a particular model - every goal is valid in an inconsistent theory, consistent theories may not have an initial model [MM].

#4 The AND/OR Gallery

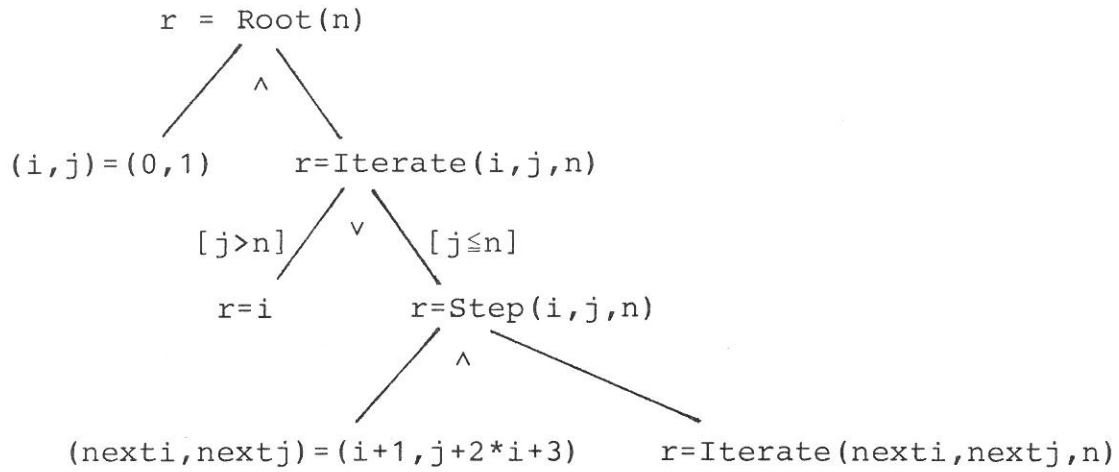
In [Ha] Harel introduced a programming language, based on the AND/OR trees so common in work in artificial intelligence. Sequencing and concurrency of programs are expressed by AND nodes in a tree; conditionals, non-determinism, repetition and recursion are expressed by OR nodes in a tree. Since the programming language is designed for decomposing complicated problems into simpler problems, it is unfair of us to choose so simple an example as that in Figure 6a. An execution of an AND/OR program gives a computation, and computations can be converted into proofs and derivations - the AND/OR frame in Figure 6a has the computation in Figure 6b, the proof in Figure 3c and the derivaiton in Figure 3e. The valuation function of the AND/OR gallery can now be defined in the same way that the valuation function of the PROLOG gallery was defined.

Val(e,m) = {e-proofs that give m-derivations}.

We get a morphism from the PROLOG gallery to the AND/OR gallery by the translation ψ_{Σ}

- AND/OR nodes give PROLOG definitions
- each branch of an OR node gives a PROLOG definition
- each AND/OR function becomes a PROLOG predicate with extra arguments for the function results.

(a) frame



(b) computation

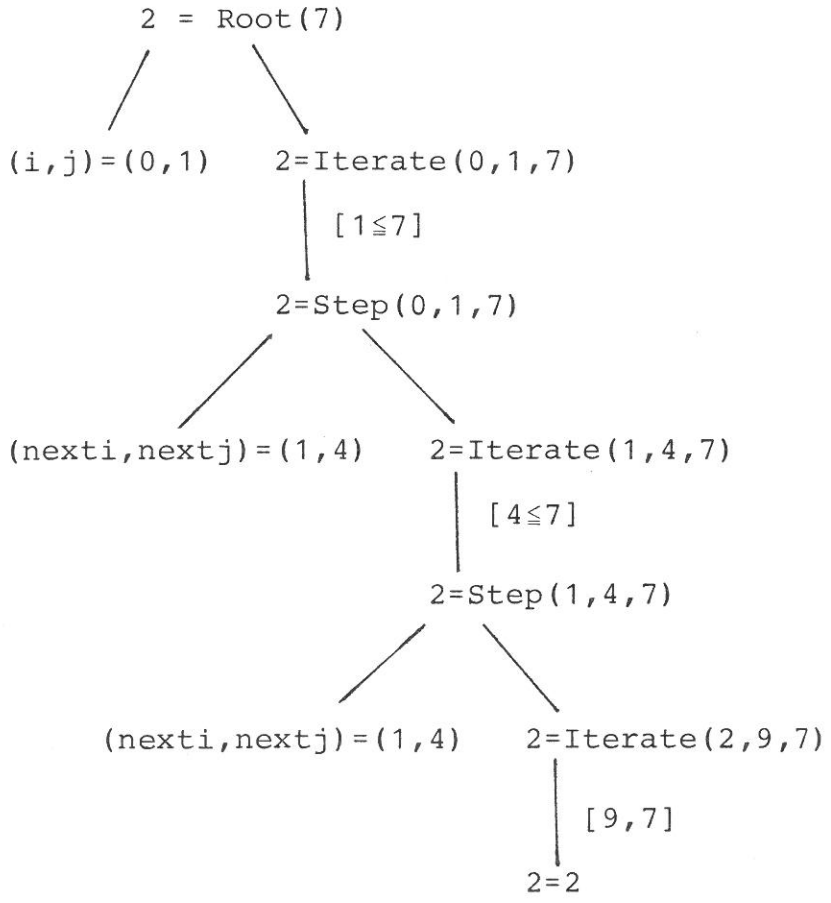
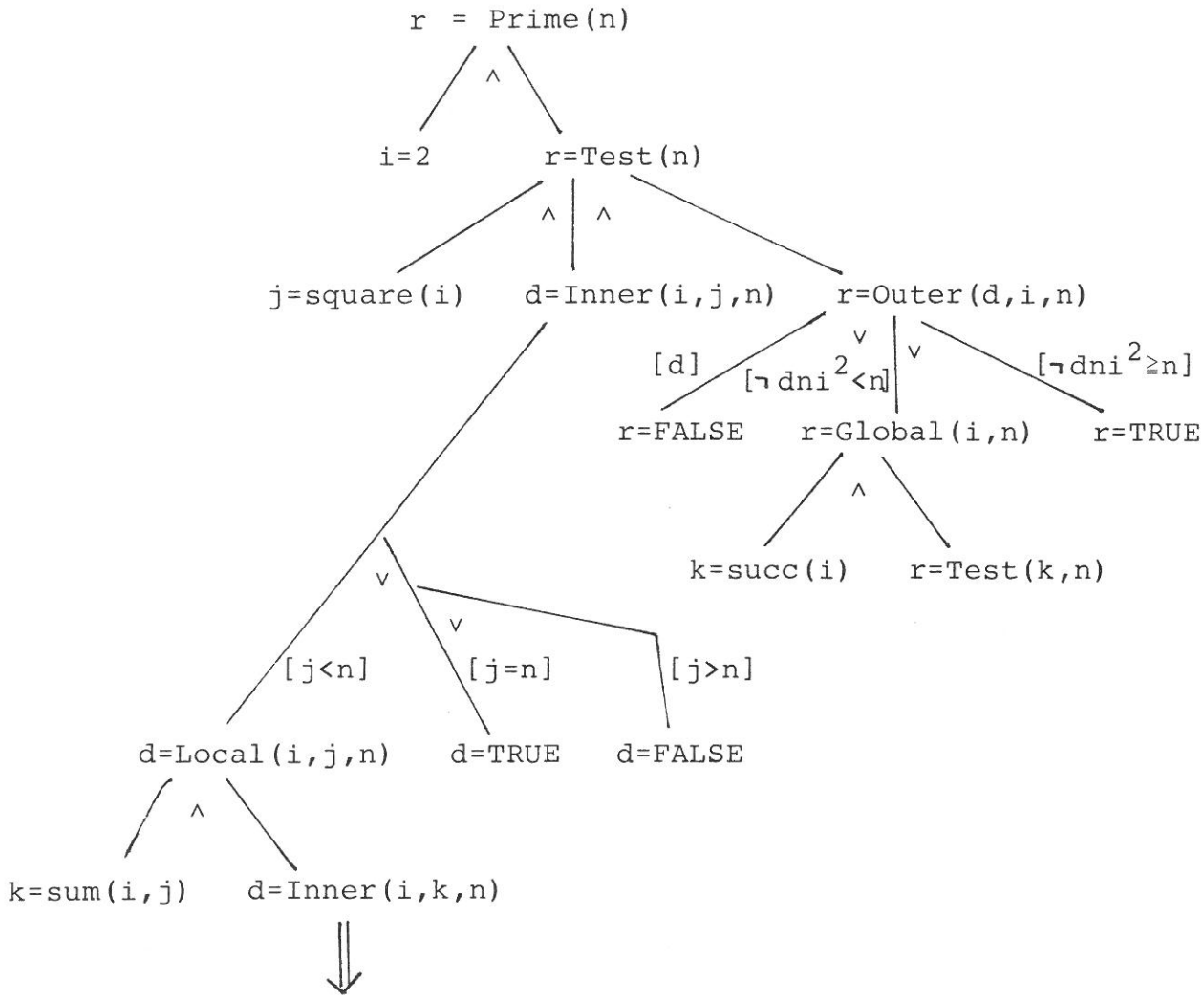


Fig. 6. An AND/OR frame and computation

Figure 7 shows the translation of a fairly complicated AND/OR program into a PROLOG frame.




```
Prime(n,r) :- Test(2,n,r).
Test(i,n,r):- Inner(i,i²,n,d),Outer(d,i,n,r).
Outer(d,i,n,FALSE) :- [d].
Outer(d,i,n,r)     :- [¬d∧i²<n],Global(i,n,r).
Outer(d,i,n,TRUE)  :- [¬d∧i²≧n].
Global(i,n,r)      :- Test(i+1,n,r).
Inner(i,j,n,d)     :- [j<n],Local(i,j,n,d).
Inner(i,j,n,TRUE)  :- [j=n].
Inner(i,j,n,FALSE) :- [j>n].
Local(i,j,n,d)     :- Inner(i,i+j,n,d).
```

Fig. 7. Translation for AND/OR to PROLOG

Do the PROLOG and AND/OR galleries have the same expressive power? There is no obvious way of translating a PROLOG frame into an equivalent AND/OR frame. The difference between PROLOG frames and AND/OR frames is the same as the difference between relations and non-deterministic functions - non-deterministic functions have designated input and output arguments, relations do not. Even if there is an equivalent AND/OR frame $\psi(e)$ for every PROLOG frame e, there may not be a gallery morphism because $\psi(e)$ may not have the same signature as e.

One would expect that the official semantics of AND/OR programs would use non-deterministic functions, but in fact they use the "least predicate" interpretation. In the last section the "least predicate" interpretation was described, and it was shown that the resulting semantics agrees with the semantics given by the valuation functions in the galleries defined so far.

#5 The EAG Gallery

Attribute grammars are not usually thought of as logical programs but this prejudice is misleading for extended attribute grammars [Ma]. Those with attribute grammar based compiler systems should be able to execute the grammar in Figure 8a as a logical program. An execution of this grammar is a computation and computations can be converted into proofs and derivations - the EAG frame in Figure 8a has the computation in Figure 8b, the proof in Figure 3c and the derivation in Figure 3e. The valuation function of the EAG gallery can now be defined in the same way as the valuation functions for the PROLOG and AND/OR galleries. The official semantics of extended attribute grammars is given by "least solution of attribute equations" but this agrees with the semantics given by the EAG valuation function.

(a) frame

```
Root(↓n↑r) ::- Iterate(↓0↓1↓n↑r).
Iterate(↓i↓j↓n↑i) ::- less(↓j↓n↑TRUE).
Iterate(↓i↓j↓n↑r) ::- less(↓j↓n↑FALSE),Step(↓i↓j↓n↑r).
Step(↓i↓j↓n↑r)    ::- Iterate(↓i+1↓j+2*i+3↓n↑r).
```

(b) computation

```
                    Root(↓7↑2)
                    Iterate(↓0↓1↓7↑2)
  ------------------------------------------------
  less(↓1↓7↑TRUE)   Step(↓0↓1↓7↑2)
                    ------------------------------
                    Iterate(↓1↓4↓7↑2)
                    ------------------------------
                    less(↓4↓7↑TRUE)   Step(↓1↓4↓7↑2)
                                      ------------------------------
                                      Iterate(↓2↓9↓7↑2)
                                      ------------------------------
                                      less(↓9↓7 FALSE)
```

Fig. 8. An EAG frame and computation

The EAG and AND/OR galleries have the same expressive power, because there are morphisms between them in both directions given by the translations illustrated in Figures 6a, 7a, 8a and 9a. The expressive power of the EAG gallery can be increased by allowing "relational attribute grammars" [CD] as frames and dropping the distinction between inherited (input) attributes and synthesized (output) attributes. One can define relational attribute grammars as context-free grammars with proof rules like those in Figure 9b for each production. As every GRAMMAR frame is a relational attribute grammar, every PROLOG frame can be translated to a relational attribute grammar. PROLOG fanatics may borrow good ideas from attribute grammar enthusiasts and vice versa.

(a)
```
Prime(↓n↑r) ::= Test(↓2↓n↑r)
Test(↓i↓n↑r) ::= Inner(↓i↓i²↓n↑d),Outer(↓d↓i↓n↑r)
Outer(↓TRUE↓i↓n↑FALSE) ::=
Outer(↓FALSE↓i↓n↑r)    ::= less(↓i²↓n↑TRUE),Global(↓i↓n↑r)
Outer(↓FALSE↓i↓n↑r)    ::= less(↓i²↓n↑FALSE)
Global(↓i↓n↑r)         ::= Test(↓i+1↓n↑r)
Inner(↓i↓j↓n↑d)        ::= less(↓j↓n↑TRUE),Local(↓i↓j↓n↑d)
Inner(↓i↓j↓n↑j=n)      ::= less(↓j↓n↑FALSE)
Local(↓i↓j↓n↑d)        ::= Inner(↓i↓i+j↑d)
```

(b)
$$\frac{m_1 \vdash \text{Test}}{m_0 \vdash \text{Prime}} \qquad \frac{m_1 \vdash \text{Inner} \quad m_2 \vdash \text{Outer}}{m_0 \vdash \text{Test}}$$

Fig. 9. Another EAG frame and general proof rules

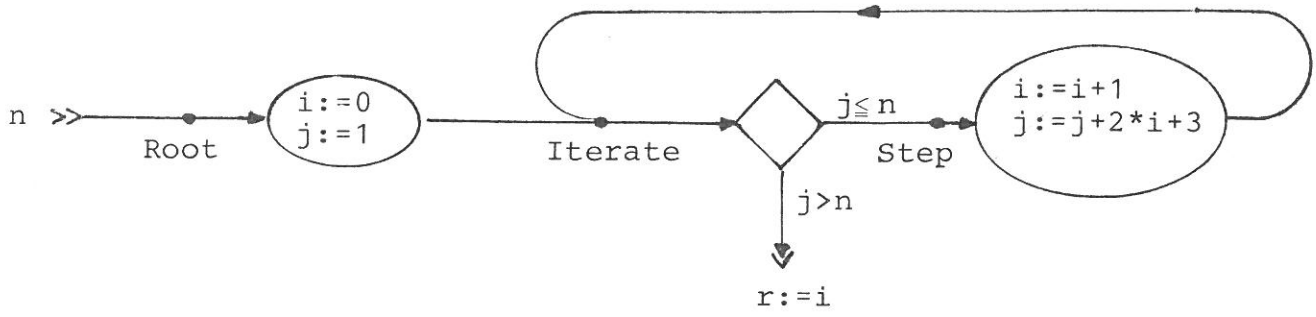
#6 The Data Flow Gallery

Data flow machines are not usually thought of as logical programs, but this impression is misleading because "machines that compute a function non-deterministically" are one way of defining a logical relation. Unlike von Neumann machines, data flow machines have no store, data flows between processes and a process transmits its results as soon as it has received its arguments [Ka]. The data flow machine in Figure 10a has the computation in Figure 10b, the proof in Figure 3c and the derivation in Figure 3e. The valuation function of the data flow gallery can be defined in the same way as the valuation function of our other galleries

Val(e,m) = {e-proofs that give m-derivations}

where m is a structure and e is a data flow machine/frame.

(a) frame



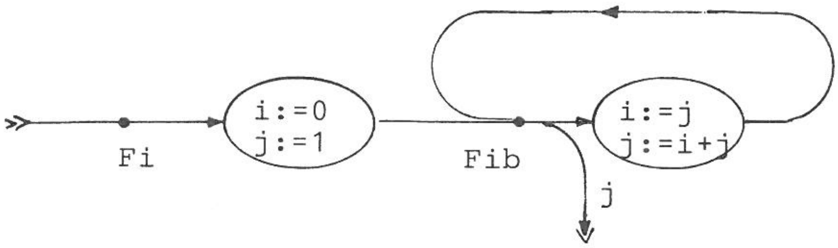
(b) computation

```
Root(7) = Iterate(0,1,7)
        = Step(0,1,7)
        = Iterate(1,4,7)
        = Step(1,4,7)
        = Iterate(2,9,7)   =  2
```

Fig. 10. A data flow frame and computation

It is clear that data flow machines can be translated into extended attribute grammars and vice versa, so data flow machines, extended attribute grammars and AND/OR programs have the same expressive power; the corresponding galleries are isomorphic. In Figure 11 equivalent logical programs for determining an interesting infinite object illustrate these gallery isomorphisms, and show that our data flow machines are proper generalisations of flow charts (see [BvE]).

frames




```
Fi(↑<1,2>) ::- Fib(↓<0,1,2>  ↓ <1,2> ↑ z)

Fib(↓<i,I> ↓ <j,J> ↑ <z,Z>) ::- +(↓x↓y↑z) Fib(↓I↓J↑Z)
```

```
<1,Z> = Fi( )                          <z,Z> = Fib(<i,I>,<j,J>)
  |                                          /    ∧    \
Z = Fib(<0,1,Z>,<1,Z>)              z=i+j              Z=Fib<I,J>
```

computation

1 2 3 5 ...	=	Fib( )
1 2 3 5 ...	=	Fib(0 1 1 2 3 5 ..., 1 1 2 3 5 ...)
1=0+1 2 3 5 ...	=	Fib(1 1 2 3 5 ..., 1 2 3 5 ...)
2=1+1 3 5 ...	=	Fib(1 2 3 5 ..., 2 3 5 ...)
3=1+2 5 ...	=	Fib(2 3 5 ..., 3 5 ...)
⋮	⋮	⋮

Fig. 11. Three frames with an infinite computation

#7 The LUCID Gallery

The logical programming language LUCID was introduced by Ashcroft and Wadge [AW] and a recent paper advocates LUCID style programming in PROLOG [BvE]. A LUCID program is a set of equations specifying a set of variables; a variable may be specified directly by an equation like V = E or indirectly by two equations: first V = E, next V = E'; every variable except "input" must be specified, and no variable may be specified more than once. Figure 12 gives two LUCID programs and shows how the "official" semantics gives computations as infinite sequences of values for the variables in the programs.

frames

```
        n = first input
first   i = 0
next    i = i+1
first   j = 1
next    j = j+2*i+3
output  = i as soon as j>n
```

```
first i = 0
first j = 1
next  i = j
next  j = i+j
output  = j
```

computations

```
input  = (7 ...)
  n    = (7 7 7 7 7 ...)
  i    = (0 1 2 3 4 ...)
  j    = (1 4 9 16 25 ...)
output = (2 2 2 2 2 ...)
```

```
i      =  (0 1 1 2 3 ...)
j      =  (1 1 2 3 5 ...)
output =  (1 1 2 3 5 ...)
```

Fig. 12. Two LUCID frames and computations

The structures and the signatures of the LUCID gallery are the same as those of all our other galleries and the frames are LUCID programs but what is the valuation function? Let us define the value of a LUCID program e in a structure m as the value of the data flow frame $\delta(e)$ in the structure m where δ is the translation given by

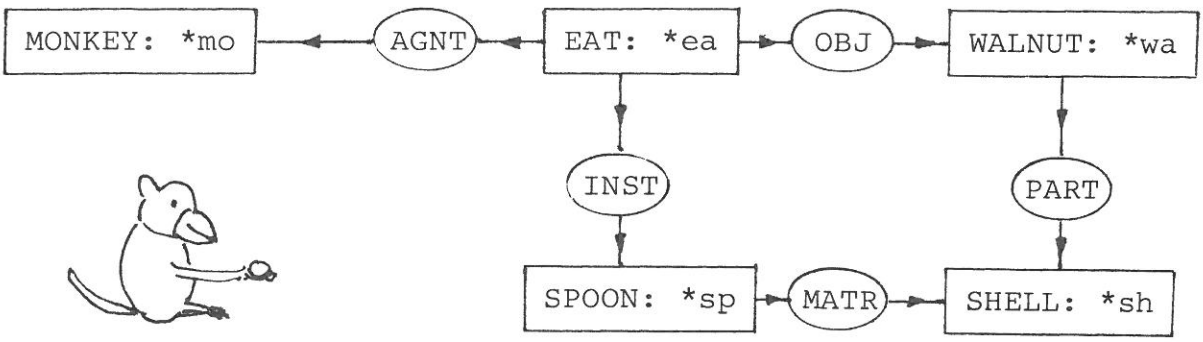
- there are two processes, FIRST and NEXT
- the process FIRST assigns the initial values of the variables which are not specified using as soon as
- the process NEXT computes the non-initial values of variables and also checks when the initial value of an as soon as variable is defined.

This translation gives a gellery morphism from the data flow gallery to the LUCID gallery, but it is not an isomorphism because there are data flow machines that are not translations of LUCID programs.

8 The Sowa Gallery

Let us now look at a typical knowledge representation language - Sowa's conceptual graphs [SO]. Figure 13a gives the conceptual graph for the natural language sentence "a monkey is eating a walnut with a spoon made out of the walnut shell". The vocabulary of this conceptual graph is given by the signature Σ_2 in Figure 2, and the graph is either true or false in each Σ_2-structure. The graph e is true in a structure m if and only if the PROLOG goal in Figure 13b is true in m.

(a)

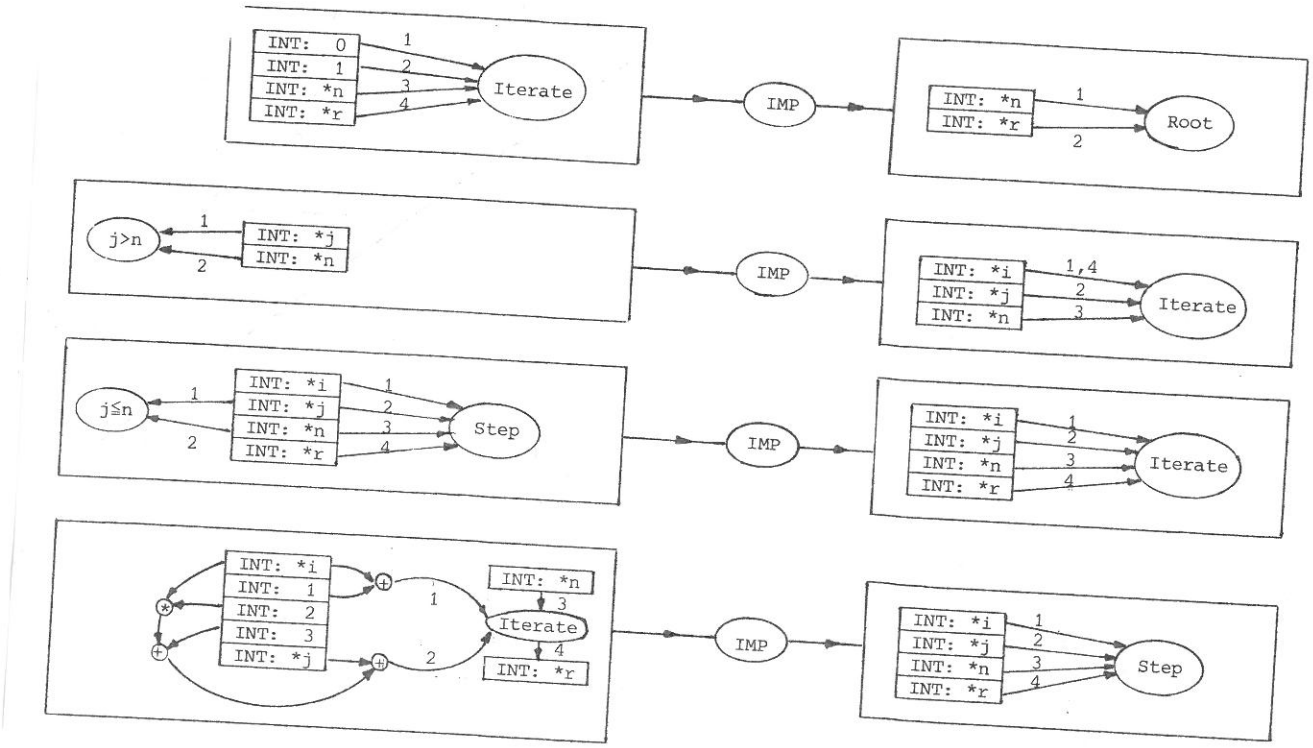


(b) AGNT(ea,mo) & OBJ(ea,wa) & INST(ea,sp) & MATR(sp,sh) & PART(wa,sh)

Fig. 13. A Sowa frame and a PROLOG goal

The structures and the signatures of the Sowa gallery are the same as those of all our other galleries and the frames are sets of conceptual graphs, but what is the valuation function? The valuation function in [Ma2] is not quite appropriate because here we are focussing on the expressive power of languages and free variables in conceptual graphs disappear (all variables occur in the signature). Before defining the modified valuation function, let us consider the conceptual graphs in Figure 14a. This set of conceptual graphs is more verbose than our other representations of the square root program because signature information is made explicit in conceptual graphs. The only logical operator in Figure 14 is implication, but quantifiers and modal operators are also allowed in conceptual graphs. Traditional logic gives proof rules like modus ponens, and these proof rules give proofs of a conceptual schema from a set of conceptual schemas. Figure 13b shows a typical proof with conceptual graphs translated into logical formulae. The implicit applications of modus ponens in Figure 14b have become explicit in Figure 14b. The obvious translation from PROLOG frames to conceptual graphs is a gallery morphism from the Sowa gallery to the PROLOG gallery if we define the valuation function in the Sowa gallery by: Val(e,m) = if e is the translation of a PROLOG frame e', then the PROLOG value of e' in m, else proofs from e and assumptions valid in the structure m. Because the Sowa gallery captures first order logic, there are Sowa frames that are not translations of PROLOG frames; the PROLOG and Sowa galleries are not isomorphic.

(a)



(b)

Root(7,2)

Iterate(0,1,7,2) — Iterate(0,1,n,r) → Root(n,r)

1≦7 Step(0,1,7,2) — 1≦7 & Step(0,1,7,2) → Iterate(0,1,7,2)

Iterate(1,4,7,2) — Iterate(1,4,7,2) → Step(0,1,7,2)

4≦7 Step(1,4,7,2) — 4≦7 & Step(1,4,7,2) → Iterate(1,4,7,2)

Iterate(2,9,7,2) — Iterate(2,9,7,2) → Step(1,4,7,2)

9>7 — 9>7 → Iterate(2,9,7,2)

Fig. 14. Another Sowa frame and a proof

#9 Natural Language

For every natural language we have a gallery L; the signatures and structures are the same as all our other galleries; the frames of L are the sentences of the natural language. The obvious translation of conceptual graphs into sentences gives a gallery morphism from L to the Sowa gallery if we define the valuation function in L by: Val(e,m) = if i is the translation of a conceptual graph e', then the Sowa value of e' in m, else arguments from sentence e that are valid in the "possible world" m. We do not know if conceptual graphs have the same expressive power as natural language. Is every sentence in every natural language the translation of some conceptual graph? Do all natural languages have the same expressive power? Perhaps these are not the right questions to put to linguists, because the privileged role of sentences in our language galleries and the identification

meaning of sentence e = valid arguments from e

may be unjustified. In the first section we considered the identification

meaning of sentence e = possible worlds where e is true

and there may be other reasonable formalisations of the "expressive power" of languages.

References

[AW] E.A. Ashcroft, W.W. Wadge: Lucid, a non-procedural language with iteration. CACM 20 (1977) 519-526.

[BvE] D.R. Brough, M.H. van Emden: Dataflow, Flowcharts and LUCID style programming in logic, Proc. IEEE Logic Prog. 84, 252-258.

[CD] B. Courcelle, P. Deransart: Proofs of partial correctness for attribute grammars and recursive procedures, INRIA no. 322, 1984.

[Ha] D. Harel: And/Or programs, a new approach to structured programming, ACM Tr. Prog. Lang. 2 (1980) 1-17.

[Ka] G. Kahn: The semantics of a simple language for parallel programming. Proc. IFIP 74, N. Holland.

[Ma] O.L. Madsen: On defining semantics by means of extended attribute grammars. Springer LNCS 94 (1980).

[Ma1] B.H. Mayoh: Unified theory of languages, models and logics, to appear.

[Ma2] B.H. Mayoh: Unified theory of knowledge representation. PROC. AIMS 84, Varna.

[MM] B. Mahr, J.A. Makowski: Characterising specification languages which admit initial semantics, TCS 31 (1984) 49-60.

[So] J.F. Sowa: Conceptual structures, Addison Wesley, 1984.

Unified Theory of Languages, Models and Logics

Most of theoretical computer science consists of studies of specific problems, illustrated by a particular choice of programming or specification language, program model or logic.
The literature is full of particular choices, but there is a need for a unified theory in which one can

- prove general results for many languages, models and logics.
- transfer results from one choice of language, model or logic to another.
- combine particular choices of languages, models and logics.

In this paper we present a unified theory, based on the concept of a gallery. Many years ago logicians generalised the study of models of first order theories to "soft model theory", the study of models of families of logics.
Recently this concept of a logical family has been generalised to that of an institution.[GB2]. Galleries are a further generalisation of institutions, in which one can have general values of expressions (terms) not just sentences with truth values. As figure 1 indicates, there are galleries "behind" most theories in computer science. For each gallery the figure gives its signatures, its frames and its structures, but the theories and the specifications of the gallery are not given.

GALLERY	FRAMES	STRUCTURES	SIGNATURES
Languages:			
:functional	expressions	domains	function names
:ADA	statements	bodies	interfaces
:programming	commands	domains	
:den.sem.	programs	domains	syntax
:op.sem.	programs	configurations	syntax
Models:			
:graph	graphs	domains	node & edge labels
:flowchort	flowchorts	domains	multi-exit boxes
:Turing	move tables	tapes	alphabet
:Petri	nets	behaviours	transition labels
:Milner	agents	synch. trees	actions
:fairness	parallel-programs	domains	primitives
:Pratt	process-nets	pomsets	process labels
:diagram	movesets	domains	configuration & move labels
Logics:			
:Equation	equations	Σ-algebras	sorts & symbols
:modal		Kripke structures	function/relation names
:1-order	1-order-form	Σ-algebras	sorts & symbols
:dynamic	[p] <p> ...	Kripke structures	program/relation names
:schema	schema form	Σ-algebras	sorts & symbols
:n-order	higher form	function algebras	sorts & symbols
:∞-order	lambda exp.	cartesian cat.	types
Inf. System	data-objects	object-sets	
Rec. prog. sch.	equations	domains	function names
Att:grammars	words	domains	grammars
Conc. graphs	graphs	closed worlds	names & concepts
Knowledge frames	lambda exp.	domains	sorts with
Montague gram.	formulae	domains	sorts & symbols

Fig. 1. A galaxy of galleries

In section 1 the contrast between the directness of galleries and the indirectness of institutions is illustrated by data type and programming language examples. In section 2 we introduce the concept of a room with a frame set, a structure category, and an evaluation function. As section 3 shows, every room has a multitude of theories and specifications. The concept of a signature is explored in section 4, and galleries are defined as a functor to rooms from signature categories. Institutions are no more than functors from signature categories to particular kinds of rooms, in which each frame is either "true" or "false" in each structure. The main reason for using category theory and indexing rooms by signatures is: constructions in the signature category may be reflected in room constructions. In section 5 this is illustrated by such examples as: pushouts induce parametrised structures.
Many galleries have useful substitution conventions and grammars are often convenient, so these are explored in section 6. The next section is devoted to morphisms between galleries, because they allow results to be transferred from one gallery to another and enhance the generality of our unified theory. The final section describes three "intrinsic" logics for reasoning in particular galleries - a traditional logic, a lambda calculus, and a type theory.

#1 Direct and indirect definitions

When one defines a data type in a programming language, one writes a declaration like: type RATIONAL is record ..... Usually various operations are associated with new data types, so modern programming languages allow modules or clusters like: package STACK is.... The interface to such a package is a signature Σ and the body of the package gives a Σ-structure, i.e. routines for the associated operations and value domains for the types in the package. The user of the package can compose Σ-frames, i.e. Σ-commands and Σ-expressions. The direct way of presenting a datatype is to give a function

$$Val_\Sigma : \Sigma\text{-frames} \times \Sigma\text{-structures} \longrightarrow \text{Set}$$

The indirect way of presenting a data type is to give an algebra or logical theory which determines a particular "initial" Σ-structure.

Programming languages can be considered as data types, whose signature Σ is given by a grammar: sorts for each non-terminal and

operations for each production. The Σ-frames are the programs and program fragments in the language. The direct way of presenting the semantics of a programming language - giving value domains to each sort and functions for each operation - is taken by operational and denotational semantics. The indirect way of presenting the semantics of a programming language - giving equations and formulae - is taken by algebraic and axiomatic semantics.

Direct definitions seem to be both more basic and closer to actual practice, but indirect definitions also have their virtues. In our unified theory indirect objects are special cases of direct objects, logical rooms are particular kinds of rooms, institutions are particular kinds of galleries.

#2 Rooms

The building blocks of our unified theory are called "rooms".

We shall lead up to the precise definition of a "room" by giving examples of special kinds of rooms.

Example 1

One can argue that data types are no more than functions from a set of expressions to values [M2]. One can consider the function

$$M: \text{Program Fragments} \times \text{Environment} \rightarrow (\text{State} \longrightarrow \text{State})$$

given by the semantics of a programming language as a data type but it seems more natural to consider M as an example of

Def.1. A discrete room consists of a set FRM of expressions; a set STR of contexts and a function $Val: FRM \times STR \rightarrow Set$. A data type of this room is a function $f: FRM \rightarrow Set$; it is a realisable data type iff $f(e) = Val(e,m)$ for some $m \in STR$.

Comment

Discrete rooms with finite sets FRM of objects and STR of attributes have been investigated in (P) under the name of information systems.

Example 2

Suppose we are interested in program transformations and we know the semantic function M in the last example. It is not unnatural to consider assertions like "fragment P_1 is equivalent to fragment P_2" as sentences that are true in an environment ρ if $M(P_1,\rho)=M(P_2,\rho)$ and false if $M(P_1,\rho)\neq M(P_2,\rho)$.
If we define $Val(\rho)=\{$sentences true in environment $\rho\}$, this is an example of

Def.2. A logical room consists of a set FRM of sentences, a category STR of structures and a functor Val from STR to P(FRM), the family of subsets of FRM considered as a category. A datatype of this room is a subset $E\subset FRM$; it is realisable if $E=Val(m)$ for some $m\in|STR|$.

Example 3

Information systems [S,2] are an interesting reformulation of domain theory. Each information system gives a logical room with

FRM = data objects $|STR|$=consistent subsets of FRM

$Val(m) = m$

when we agree that inclusions are the only STR morphisms.

Comment

In a logical room one often writes

$m \models e$ for $e \in Val(m)$

$m \not\models e$ for $e \notin Val(m)$

So in our example $\rho \models P_1 = P_2$ expresses $M(P_1,\rho)=M(P_2,\rho)$.
Logical rooms are the building blocks for the theory of institutions (GB2), In some institutions the logical rooms are discrete, because the only structure morphisms are the identities, so the room functor Val gives the function - $Val(e,m)=$ if $m \models e$ then 1 else 0 - and a subset $E\subset FRM$ corresponds to the datatype $f(e)$ = if $e\in E$ then 1 else 0. The natural generalisation of discrete and logical rooms is

Def.3. A room consists of a set FRM of frames, a category STR of structures, and a functor Val from STR to Bn(FRM), the category of functions from FRM to Set. The data types of the room are the objects in the category $B_n(FRM)$, families of sets indexed by FRM, $< v_e | e \in FRM >$. A data type d is realisable iff d=Val(m) for some $m \in STR$; it is a truth value iff $v_e=0$ or $v_e=1$ for all $e \in FRM$.

Comment

In Bn(FRM) a morphism from a datatype $< v_e | e \in FRM >$ to a datatype $< w_e | e \in FRM >$ is a family $< f_e | e \in FRM >$ of set functions indexed by FRM with f_e a function from v_e to w_e for all $e \in FRM$.
The reason why "truth value" is an appropriate term will become apparent when we describe the implicit logic of a room, but a few remarks might be welcome:

- 0 is a synonym for "false" or "the empty set ϕ"
- 1 is a synonym for "true" or "the set $\{\phi\}$"
- there is no function from 1 to 0
- there is one function from 0 to 0, 0 to 1 and 1 to 1
- the truth value $V = <v_e | e \in FRM >$ corresponds to $E_v=\{e | v_e=1\} \subseteq FRM$
- there is a morphism from truth value v to truth value w if and only if $E_v \subseteq E_w$

The last two remarks show that a room is logical iff each of its realisable data types is a truth value.
Since we have the bijection $Val(m)=\lambda e.\ Val(e,m)$, a room is discrete iff the only structure morphisms are identities.

Example 1 ctd.

The truth values in the room for M: Program Fragments x Environments $\longrightarrow$ (State -> State) are just "sets of program fragments". The evaluation function M has state functions as the meaning of programs but many other semantics are possible - continuation functions, predicate transformers, computation sequences, traces and much else appear in the literature. Each of these meaning functions gives a room with FRM=Program Fragments, STR=Environments but different data types in Bn(FRM) are realisable in the different rooms.

We will be very interested in maps between rooms. Our definition may seem strange, but it is adapted from [GB2] and ensures that institutions are galleries.

Def.4. A discrete room morphism ϕ from discrete room $D=< FRM,STR,Val>$ to discrete room $D'=< FRM',STR',Val' >$ consists of functions $\phi^6: FRM \to FRM'$, $\phi^\#: STR' \to STR$ such that

$$Val(e,\phi^\#(m')) = Val'(\phi^6(e),m')$$

for all $e \in FRM$, $m' \in STR'$.

Comment

The function ϕ^+ takes each data type $f':FRM' \to Set$ to data type $f' \circ \phi^6: FRM \to Set$. If the data type f' is realisable or a truth value, then $f' \circ \phi^6$ is also; if $\phi^\#$ is a surjection, then every realisable datatype $f:FRM \to SET$ has the form $f' \circ \phi^6$.

Def.5. A logical room morphism ϕ from logical room $L=< FRM,STR,Val >$ to logical room $L'=< FRM',STR',Val' >$ consists of a function $\phi^6:FRM \to FRM'$ and a functor $\phi^\#$ from STR' to STR such that

$$e \in Val(\phi^\#(m')) \ .\equiv. \ \phi^6(e) \in Val'(m')$$

for all sentences $e \in FRM$ and structures $m' \in STR'$.

Comment

The function ϕ^+ takes each truth value $E' \subseteq FRM'$ into the truth value $\{e \mid \phi^6(e) \in E'\} \subseteq FRM$. If the truth value E' is realised, so is $\phi^+(E')$; if $\phi^\#$ is a surjection, then every realised truth value has the form $\phi^+(E')$.

Def.6. A morphism ϕ from a room $R=<FRM,STR,Val>$ to a room $R'=<FRM',STR',Val'>$ consists of a function $\phi^6:FRM \to FRM'$ and a functor $\phi^\#$ from STR' to STR such that

$$Val(\phi^\#(m')) = Val'(m') \circ \phi^6$$

for all $m' \in STR'$.

Any function ϕ^6:FRM ⟶ FRM' gives a functor ϕ^+ from Bn(FRM') to Bn(FRM) by

$$\phi^+(< v_e \mid e \in FRM' >) = < v_{\phi^6(e)} \mid e \in FRM >$$

and this function takes a datatype d' to the datatype $d' \circ \phi^6 \in B_n(FRM)$. The requirement in definition... for room morphisms can be expressed in the communitive diagram

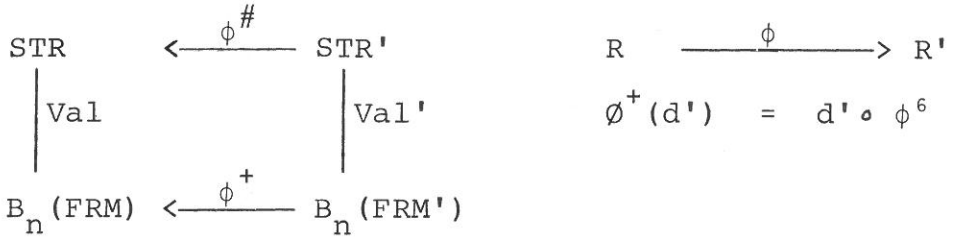


Fig. 2. Room morphism requirement

Example 2 ctd.

Consider the distinction between concrete and abstract syntax in some programming languages. In formal semantics it is usual to define meanings of abstract derivation trees, to define a room < Tree, STR, Val >. If we have a parser ϕ^6: Program Fragments ⟶ Tree, and we take $\emptyset^\#$ as the identity on STR, then $\emptyset$ is a room morphism from

$$< \text{Program Fragments}, STR, \lambda m.\ Val(m) \circ \phi^6 >$$

to < Tree, STR, Val >. If the programming language is unambiguous, there is no choice in the definition of the parser ϕ^6.

#3 Specifications and Theories

In software engineering specifications are used to describe programs and data types; in logic theories are used to define classes of models; in this section we show how every room has a multitude of theories and specifications; every class of structures gives a theory and every class of frames gives a specification.

Each frame e in a logical room L=<FRM,STR,Val> partitions STR into two equivalence classes

$$TRUE_e = \{m \in STR \mid e\ Val(m)\},\ FALSE_e = \{m \in STR \mid e \not\in Val(m)\}$$

Usually one says "e is a specification of $TRUE_e$" but we prefer "e is a specification of the partition $\{TRUE_e, FALSE_e\}$" because of the generalisation: each frame e in a room R=<FRM,STR,Val> partitions STR into equivalence calsses

$$[m]_e = \{m_1 \in STR \mid Val(m)e = Val(m)e)\}$$

This is further generalised in:

Def.7. Every non-empty set E of frames in a room R=<FRM,STR,Val> defines a specification, the partition of STR with equivalence classes

$$[m]_E = \{m_1 \in STR \mid \forall e \in E.\ Val(m_1)e = Val(m)e\}$$

A set M of structures is describable iff $M=[m]_E$ for some $m \in |STR|$ $E \subseteq FRM$.

In a logical room L=<FRM,STR,Val> the equivalence classes are given by $m_1 \in [m]_E .\equiv . \forall e \in E(e \in Val(m_1) \equiv e \in Val(m)$ and there is a distinguished equivalence class $E^* = \{m \in STR \mid E \subseteq Val(m)\}$. A structure m implements E iff $E \subseteq Val(m)$.

□

By convention the empty set of frames defines the trivial partition 0 of STR with one equivalence class, and 1 is the trivial partition of STR with one structure in each equivalence class. This gives

- $[m]_E = \bigcap \{[m]_e \mid e \in E\}$
- $\phi \subseteq E \subseteq E^1 \subset FRM$ implies $|\subseteq [m]_{FRM} \subseteq [m]_E \subseteq [m]_E \subseteq 0$
- $| = [m]_{FRM}$ iff Val is an injection.

- $\{A_2\}$ is describable iff $Val(m)=Val(m_1)$ implies $m=m_1$
- the intersection of describable sets is describable or empty

Specifications in non-logical rooms have been used to define closures and approximations to structure classes in figure 3.

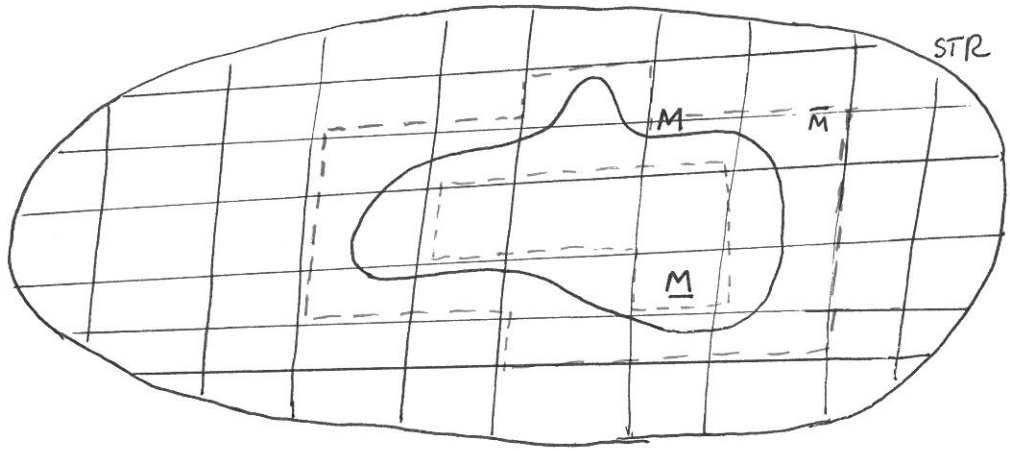


Fig.3. Upper and lower approximations

The small squares show a partition of structures, the inner curve shows a structure class M, the outer dashed line shows the upper approximation of M, and the inner dashed line shows the lower approcimation of M. Because the two approximations can be different, the "logic of the specification" is not classical; if the lower approximation $\underline{M}$ is chosen, this logic is intuitionistic, if the upper approximation $\overline{M}$ is chosen, it is paraconsistent.

In a logical room $TRUE_e = \{e\}^*$

$\phi \subseteq E \subseteq E \subset FRM$ implies $\phi \subseteq FRM^* \subseteq E'^* \subseteq E^* \subseteq STR$

and $0=FRM^*$ is equivalent to "every structure makes some sentence false".

For us theories are duals of specifications. Each structure m in a logical room L= <FRM,STR,Val> partitions FRM into two equivalence classes

$$DIAGRAM_m = \{e \in FRM \mid e \in Val(m)\}, \{e \in FRM \mid e \notin Val(m)\}$$

Usually one says "$DIAGRAM_m$ is the theory of m" but we prefer "the partition $<DIAGRAM_m, SENT - DIAGRAM_m>$ is the theory of m" because of the generalisation: each structure m in a room R=<FRM,STR,Val> partitions FRM into equivalence classes

$$[e]_m = \{e_1 \in FRM \mid Val(e,m) = Val(e,m)\}$$

This is further generalised in

Def 8. Every non-empty set M of structures in a room R = <FRM,STR,Val> defines a theory, the partition of FRM with equivalence classes

$$[e]_M = \{e_1 \in FRM \mid \forall m \in M.\, Val(m), e' = Val(m)e\}$$

A set E of frames is definable iff $E = [e]_M$ for some $e \in FRM, M \subseteq |STR|$.

In a logical room L = <FRM,STR,Val> the equivalence classes are given by $e_1 \in [e]_{m.} \equiv . \forall m \in M\, (Val(m)e_1 \equiv Val(m)e$ and there is a distinguished equivalence class

$$M^* = \bigcap \{Val(m) \mid m \in M\}$$

□

By convention the empty set or structures defines the trivial partition 0 of FRM with one equivalence class and 1 is the trivial partition of FRM with one frame in each equivalence class. This gives

- $[e]_M = \bigcap \{[e]_m \mid m \in M\}$
- $\phi \subseteq M \subseteq M \subseteq STR$ implies $1 \subseteq [e]_{STR} \subseteq [e]_M \subseteq [e]_M \subseteq 0$

- $\{e\}$ is definable iff $\forall m(Val(m)e = Val(m)e')$ implies $e = e'$
- $1 = [e]_{STR}$ iff all singletons $\{e\}$ are definable
- any intersection of definable framesets is definable or empty.

Our concept of a theory as a partition is not as strange as it might appear, it is very closely related to the algebraic theories introduced by Lawvere and used by many computer scientists. See the construction of algebraic theory from a congruence and the references in (GBI). The specification system CLEAR (BG) shows how complicated theories can be built from simple theories.

In a logical room $DIAGRAM_m = \{m\}^* = Val(m)$,

$\phi \subseteq M \subseteq M \subseteq STR$ implies $\phi \subseteq STR^*_{STR} \subseteq M'^* \subseteq M^* \subset FRM$

$\phi = STR^*$ is equivalent to "no sentence is true in all structures."

Notice that we have the usual Galois connection:

$M_1 \subseteq M_2 \longrightarrow M_2^* \subseteq M_1^*$ $\quad M \subseteq M^{**}$ $\quad M^* = M^{***}$

$E_1 \subseteq E_2 \longrightarrow E_2^* \subseteq E_1^*$ $\quad E \subseteq E^{**}$ $\quad E^* = E^{***}$

** is a closure operation, our M^* are the closed frame sets, and our E^* are the closed structure sets.

Notice also that Val can be replaced by any other function from |STR| to P(FRM) in our definitions of E^*, M^* and the Galois closure operations *, and there is such a function for every relation $r \subset FRM/|STR$. Because the subsets of FRM are "truth values", our definitions give

- a truth value for <u>any</u> subset of STR
- a subset of STR for <u>any</u> truth value

This is such a big improvement on

- a truth value Val(m) for any structure m
- the subset {m|Val(m)=E} for any truth value

that we should try to generalise the construction to rooms that are not logical.

Def.8. Every non-empty set M of structures in a room $R=\langle FRM,STR,Val\rangle$ defines a datatype $M^*=\lambda e.\bigcap\{Val(m)(e)\mid m\in M\}$.

Every datatype $d\in Bn(FRM)$ defines $d^*\in|STR|$ by

$$m\in d^* \;.\equiv.\; d(e)=d(e') \longrightarrow Val(m)(e)=Val(m)(e')$$

A structure m implements datatype d iff $m\in d^*$.

Comment

Every data type d gives a partition of FRM with equivalence classes $[e]_d = \{e'\mid d(e) = d(e')\}$ so the condition for $m\in d^*$ is $[e]_d \subseteq [e]_m$, the d-partition refines the theory of m.

Notice that ** is still a closure operation, our d* are the closed structure sets, and our M* are the closed data types. Notice also that Val can be replaced by any function from |STR| to Bn(FRM) in our definition of d* and M* - and there is such a function for any $f:FRM\times|STR| \longrightarrow Set$.

In most rooms there is a connection between theories and structure morphisms $m \longrightarrow m'$. Some of the possibilities are:

(MONOTONE) $m \longrightarrow m'$ implies $[e]_{m'} \subseteq [e]_m$

(ISOTONE) $m \longrightarrow m' \longrightarrow m$ iff $[e]_{m'} = [e]_m$

(INITIALITY) every describable $M \subset STR$ has an initial model m such that

$$[e]_M = [e]_m$$

The great succuss of initial models in the theory of abstract data types suggests that conjunctive rooms, where every definable $E \subset FRM$ has a frame e such that $[m]_E=[m]_e$, may warrant special study. In (T) there is a careful study of the connection between structure morphisms and theories in logical rooms.

What happens to the theories and specifications of a room R when it is mapped into a room R' by a morphism ϕ? A specification in R = <FRM,STR,Val> determines a room R_E=<E,STR,Val>; if $\phi^6(E) \subset E'$ then ϕ gives a morphism from R_E to $R_{E'}$=<E',STR'Val'>, the room determined by E'. A theory in R determines a room R_M=<FRM,M,Val>; if $\phi\#(M') \subset M$, then ϕ gives a morphism from R_M to $R_{M'}$. This motivates:

Def.9. Let ϕ be a morphism from R=<FRM,STR,Val> to R'=<FRM',STR'Val'>. The morphism is a specification morphism from $E \subset FRM$ to $E' \subset FRM'$ iff $\phi^6(E) \subseteq E'$; the morphism is a theory morphism from $M \subset STR$ to $M' \subset STR'$ iff $\phi\#(M') \subseteq M$.

Example

Consider the identity morphism on a room R. This gives a specification morphism from E_1 to E_2 when E_1 is a subset of E_2; it gives a theory morphism from M_1 to M_2 when M_2 is a subcategory of M_1.

Theorem

If ϕ is a morphism from room <FRM,STR,Val> to room <FRM',STR',Val'>, then:

(1) $m_1' \in [m']_{\phi^6(E)} .\equiv. \phi^{\#}(m_1') \in [\phi^{\#}(m')]_E$ for $E \subset FRM$

(2) $e_1 \in [e]_{\phi\#(M')} .\equiv. \phi^6(e_1) \in [\phi^6(e)]_{M'}$, for $M' \subset STR'$

(3) $\phi^+(M') = [\phi\#(M')]^*$ for any $M' \subset STR'$

(4) $\phi^{\#}(d'^*) \subseteq [\phi^+(d')]^*$ for any $d' \in Bn(FRM')$

Proof

$$(1)\quad m_1' \in [M']_{\phi^6(E)} .\equiv. \forall e \in E[Val'(m_1')\phi^6(e) = Val'(m')\phi^6(e)]$$
$$.\equiv. \forall e \in E[Val(\phi^\#(m_1'))e = Val(\phi^\#(m')e]$$
$$.\equiv. \phi^\#(m_1') \in [\phi^\#(m')]$$

$$(2)\quad e, \in [e]_{\phi^\#(M')} .\equiv. \forall m' \in M'[Val(\phi^\#(m'))e_1 = Val(\phi^\#(m'))e]$$
$$.\equiv. \forall m' \in M'[Val'(m')\phi^6(e_1) = Val'(m')\phi^6(e)]$$
$$\equiv \phi^6(e_1) \in [\phi^6(e)]_{M'}$$

$$(3)\quad \phi^+(M'^*) = M'^* \circ \phi^6 = [Val'(m') \mid m' \in M'] \circ \phi^6$$
$$= (Val'(m') \circ \phi^6 \mid m' \in M')$$
$$= (Val(\phi^\#(m'1) \mid m' \in M')$$
$$= (Val(m) \mid m \in \phi^\#(M')) = [\phi^\#(m')]^*$$

$$(4)\quad m' \in d'^* \longrightarrow d'(\phi^6(e_1)) = d'(\phi^6(e)) \longrightarrow$$
$$Val'(m')\phi^6(e_1) = Val'(m')\phi^6(e)$$
$$\equiv. \phi^+(d')(e_1) = \phi^+(d')(e) \longrightarrow$$
$$Val(\phi^\#(m'))e_1 = Val(\phi^\#(m'))e$$
$$\equiv. \phi^\#(m) \in [\phi^+(d')]^*$$
$$m \in \phi^\#(d'^*) .\equiv. \exists m' [m' \in d'^* \wedge \phi^\#(m') = m]$$
$$\longrightarrow \exists m' (\phi^\#(m') \in [\phi^+(d')]^* \wedge \phi^\#(m') = m]$$
$$\equiv. m \in [\phi^+(d')]^* \wedge \exists m' \; \phi^\#(m') = m$$

Corollary

If ϕ is a specification morphism from E⊂FRM to E'⊂FRM', then $[m']_{E'} \subseteq [m']_{\phi^6(E)}$ and

$$m_1' \in [M]_{E'} \longrightarrow \phi^\#(m'_1) \in [\phi^\#(m)]_E$$

If ϕ is a theory morphism from M⊂STR to M'⊂STR' then $[e]_M \subset [e]_{\phi^\#(M')}$ and $e_1 \in [e]_M \longrightarrow \phi^6(e_1) \in [\phi^6(e)]_{M'}$

If m' implements d', then $\phi^6(m')$ implements $\phi^+(d')$.

Comment

The reasons why one does not always have $\phi^{\#}(d'^*)=[\phi^{+}(d)]^*$ are (1) $m' \in d'^*$ is stronger than $\phi^{\#}(m) \in [\phi^{+}(d)]^*$ (2) there can be $m \in [\phi^{+}(d')]^*$ which not of the form $\phi^{\#}(m')$. Figure 4 summaries our results on preservation by room morphism from R to R'.

R'-datatype $\xrightarrow{\phi^{+}}$ R-datatype

R'-truth value $\xrightarrow{\phi^{+}}$ R-truth value $\xrightarrow{\phi^{6}}$ R'-truth value

R'-realised datatype $\xrightarrow{\phi^{+}}$ R-realised datatype

R'-closed Datatype $\xrightarrow{\phi^{+}}$ R-closed datatype

R'-structure $\xrightarrow{\phi^{\#}}$ R-structure R-frame $\xrightarrow{\phi^{6}}$ R'-frame

R'-theory $\xrightarrow{\phi^{\#}}$ R-theory R-specificaiton $\xrightarrow{\phi^{6}}$ R'-specification

R'-structures $\xrightarrow{\phi^{\#}}$ R-structure class

Figure 4. Properties preserved by a room morphism from R to R'

#4 Galleries, institutions and signatures

Now we can define the fundamental concept of our unified theory:

Def. 10. A gallery G is a functor to the category of rooms from a category SIGN of signatures.
The gallery G is an institution iff

- each object $G(\Sigma)$ is a logical room
- each morphism $G(\phi)$ is a logical room morphism

The gallery G is discrete iff

- each object $G(\Sigma)$ is a discrete room
- each morphism $G(\phi)$ is a discrete room morphism

Comment

For each signature morphism we have the commutative diagram and our previous results show that ϕ^+ not only carries realised Σ_2-data types into realised Σ_1-data types but also closed Σ_2-data types into closed Σ_1-data types.

$$\begin{array}{ccc} STR(\Sigma_1) & \xleftarrow{\phi^\#} & STR(\Sigma_2) \\ \Big\vert Val(\Sigma_1) & & \Big\vert Val(\Sigma_2) \\ Bn(FRM(\Sigma_1)) & \xleftarrow{\phi^+} & Bn(FRM(\Sigma_2)) \end{array}$$

One can think of signatures as vocabularies, that provide useful concept names ( ), but what kinds of signature categories are useful. The simplest vocabularies are given by discrete signarures, where the only signature morphisms are the identities.
Galleries with discrete signatures are no more than "indexed families of rooms. For each room R, (R) is an indexed family so we have:

every room gives a gallery with one signature.
every logical room gives an institution with one signature.
More complicated vocabularies are given by ranked signatures with a family $\Sigma(k,l)$ of names for each pair $\langle k,l \rangle$ of natural numbers.

Usually one insists on unambiguous ranked signatures where the families $\Sigma(k,l)$ are disjoint. Frequently $\Sigma(k,l)$ is empty for $l>1$ and we read:

- $c \in \Sigma(0,1)$ as "c names a constant"
- $f \in \Sigma(k,1)$ as "f names a k-ary function"
- $r \in \Sigma(k,0)$ as "r names a k-ary relation"

For ranked signatures the natural morphisms are

- <u>embeddings</u>, functions from each $\Sigma(k,l)$ to $\Sigma'(k,l)$
- <u>inclusions</u>, $\Sigma(k,l) \subseteq \Sigma'(k,l)$ for each $\langle k,l \rangle$
- <u>relabellings</u>, bijections between each $\Sigma(k,l)$ and $\Sigma'(k,l)$

To establish a gallery with ranked signatures one has to check that each morphism ϕ gives maps $\langle \phi^6, \phi^\# \rangle$ such that

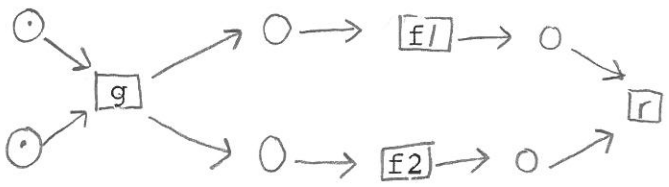
$$Val_\Sigma(\phi^\#(m'))e = Val_{\Sigma,\phi^6}(e)$$

for each $e \in FRM(\Sigma)$, $m' \in STR(\Sigma')$. This is usually easy; figure 5 gives an example:

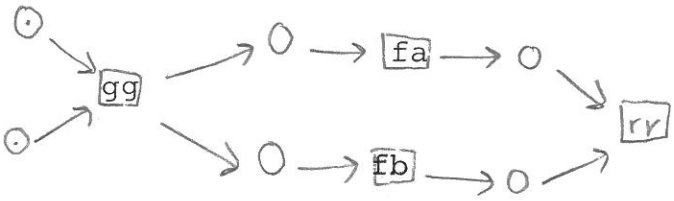
Signatures $\Sigma(k,l)$ is a family of transition names. Morphisms ϕ give injections of $\Sigma(k,l)$ into $\Sigma'(k,l)$.

Frames Marked nets where each k-entry l-exit transition has a name in $\Sigma(k,l)$. Function $\phi^6(e)$ relabels transitions

e_o =



$\phi^6(e_o)$ =



Structures are sequences m of transition names $m \models e$. $\equiv$. m is a firing sequence of e. Functor $\phi^{\#}(m')$ omits names outside $\phi(\Sigma)$

$$\phi^{\#}(m') \models e \;.\equiv:\; m' \models \phi^6(e)$$

m'_o = gg fa fb rr is a firing sequence of $\phi^6(e_o)$

$\phi^{\#}(m'_o)$ = g f1 f2 r is a firing sequence of e_o

Comment The most appropriate gallery for Petri nets is somewhat different from this institution.

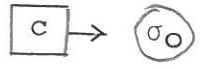
Fig. 5. A Petri net institution

Most vocabularies in computer science are given by sorted signatures with a set of Σ-sorts and a family $\Sigma(\sigma,\tau)$ of names for each pair $\langle\sigma,\tau\rangle$ of sequences (words) of Σ-sorts. Some common notation for sorted signatures is shown in figure 6. Usually galleries with sorted signatures are unambiguous and finitary: the families $\Sigma(\sigma,\tau)$ are disjoint, and empty when σ or τ is infinite, For sorted signatures the natural morphisms are

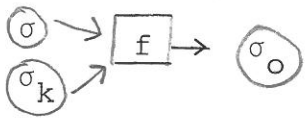
- embeddings, functions from each $\Sigma(\sigma,\tau)$ to $\Sigma'(\phi'(\sigma),\phi'(\tau))$
- inclusions, $\Sigma(\sigma,\tau) \subset \Sigma'(\sigma,\tau)$ for each $\langle\sigma,\tau\rangle$
- relabellings, bijections between each $\Sigma(\sigma,\tau)$ and $\Sigma'(\phi'(\sigma),\phi'(\tau))$

where ϕ' is the extension to sequences of a function from Σ-sorts to Σ'-sorts.

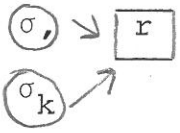
$c \in \Sigma(\ ,\sigma_o)$ constant names $c:\to\sigma_o$



$f \in \Sigma(\sigma_1..\sigma_k,\sigma_o)$ function name $f:\sigma_1 x..x\sigma_k\to\sigma_o$



$r \in \Sigma(\sigma_1..\sigma_{k1}$ relation name $r:\sigma_1 x\sigma_2..\sigma_k.$



$g \in \Sigma(\sigma_1..\sigma_k,\tau_1..\tau_1)$ $g:\sigma_1 x\sigma_2..x\sigma_k\to\tau_1 x...x\tau_1$

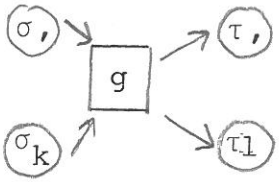


Fig. 6. Signature notation in sorted galleries

Sorted signatures are usually flat, ordered, polynomial or typed. In flat sorted signatures sorts are not related to one another; in ordered signatures there is a partial order on the sorts; in polynomial signatures there is a set of basic Σ-sorts and every Σ-sort is given by the production.

<sort>: := <basic-sort> | <sort>+<sort> | <sort> x <sort>

in typed signatures we also have the sorts given by the production: <sort>: := <sort> ——> <sort>.

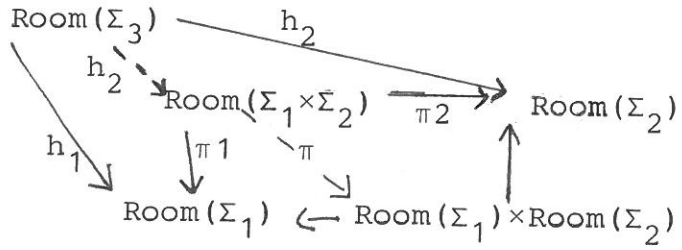
#5 Records, Arrays, Variants and Parameters

When the category of signatures in a gallery has various properties, then these properties are reflected in the frames, structures, theories and specifications of the galleries. In this section we look at SIGN properties that make it possible to construct complex rooms in the gallery from simpler rooms: products, powers, sums, copowers and pushouts. The literature on institutions has focused on sums and pushouts (colimits) because institutions originate from the desire to build complicated specifications from simple specifications.

Def.11. A gallery admits variants iff SIGN has products - for any $\Sigma_1, \Sigma_2 \in |SIGN|$ there is a signature $\Sigma_1 \times \Sigma_2$ and signature morphisms

$\pi 1: \Sigma_1 \times \Sigma_2 \longrightarrow \Sigma_1$, $\pi 2: \Sigma_1 \times \Sigma_2 \longrightarrow \Sigma_2$ such that for any $\zeta 1: \Sigma_3 \to \Sigma_1, \zeta 2: \Sigma_3 \to \Sigma_2$ there is $\zeta: \Sigma_3 \to \Sigma_1 \times \Sigma_2$ satisfying: $\zeta 1 = \pi 1 \circ \zeta$, $\zeta 2 = \pi 2 \circ \zeta$.

If a gallery admits variants, we have the diagram



where $Room(\Sigma_1) \times Room(\Sigma_2)$ has $FRM(\Sigma_1) \times FRM(\Sigma_2)$ as frameset, $STR(\Sigma_1) + STR(\Sigma_2)$ as structure category, and

$$Val(m)(e_1, e_2) = \underline{\text{if}}\ m \in STR(\Sigma_1)\ \underline{\text{then}}\ Val_{\Sigma}(m)e_1\ \underline{\text{else}}\ Val_{\Sigma}(m)e_2$$

as valuation fundtion. The room morphisms in this diagram connnect the frames, structures, datatypes theories and specifications in Room $(\Sigma_1 \times \Sigma_2)$ with those in Room (Σ_1) and Room (Σ_2).

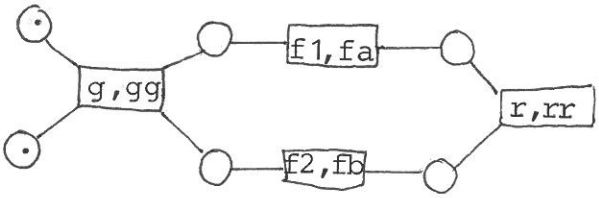
Example

If Σ_1 and Σ_2 are ranked signatures, one can define $\Sigma_1 \times \Sigma_2$ by $\Sigma_1 \times \Sigma_2(k,l) = \Sigma_1(k,l) \times \Sigma_2(k,l)$. Suppose Σ_1 and Σ_2 are the signatures in figure 2 so

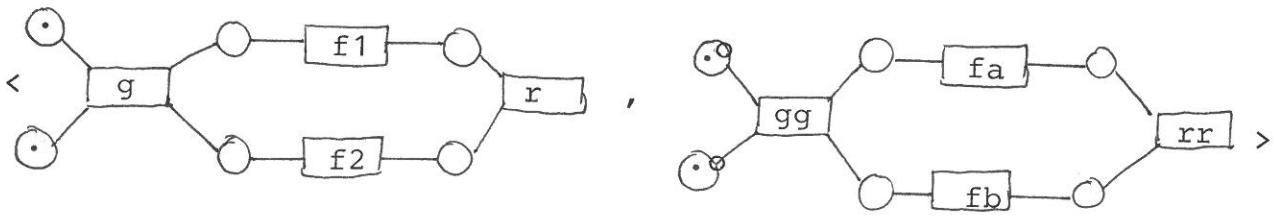
$$\Sigma_1 \times \Sigma_2(2,2) = (\langle g,gg) \qquad \Sigma_1\ \Sigma_2(2,0) = (\langle r,rr\rangle)$$

$$\Sigma_1 \times \Sigma_2(1,1) = (\langle f1,fa\rangle,\langle f2,fa\rangle,\langle f1,fb\rangle,\langle f2,fb\rangle)$$

and a typical memeber of $FRM(\Sigma_1 \times \Sigma_2)$ is



which is taken by π^6 to

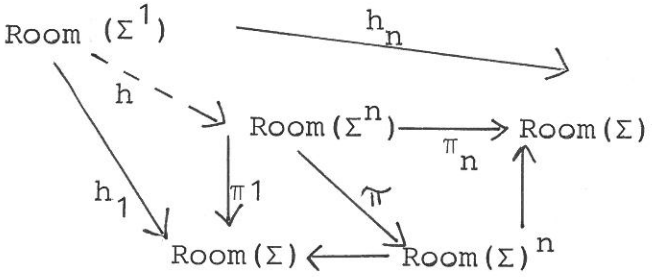


Clearly $FRM(\Sigma_1)\ FRM(\Sigma_2)$ has frame pairs which are not given by π^6.

If $\Sigma_1 \times \Sigma_2$ is to be the prouct of Σ_1 and Σ_2 in the category of signatures in our gallery, we must allow firing sequences like $\langle\langle g,*\rangle\langle f1,*\rangle\langle f2,*\rangle\langle r,*\rangle\rangle$ so $\pi 1^{\#}$ can be defined on $\langle g,f1,f2,r\rangle \in STR(\Sigma_1)$. Normal firing sequences like $\langle g,gg\rangle\langle f1,fa\rangle\langle f2,fb\rangle\langle r,rr\rangle$ are not in $STR(\Sigma_1)+STR(\Sigma_2)$, but they are so tightly coupled members of $STR(\Sigma_1) \times STR(\Sigma_2)$ that "variant" is a more appropriate term than "record".

Def.12. A gallery admits copies iff SIGN has powers - for any signature Σ there is a signature Σ^n and morphisms $\pi 1 \ldots \pi n: \Sigma^n \to \Sigma$ such that for any $\zeta 1 \ldots \zeta n: \Sigma \to \Sigma$ there is a $\zeta: \Sigma' \longrightarrow \Sigma^n$ satisfying: $\zeta 1 = \pi 1 \circ \zeta \ldots\ldots \zeta n = \pi n \circ \zeta$.

If a gallery admits copies, we have the diagram



Where $Room(\Sigma)^n$ has $FRM(\Sigma)^n$ as frames set $n \times STR(\Sigma)$ as structure category and

$$Val(n)(e_1,e_2 \ldots e_n) = \underline{case}\ m \in STR(\epsilon_i)\ \underline{of}\ Val_{\epsilon_i}(n)(e_i)\ \underline{end}$$

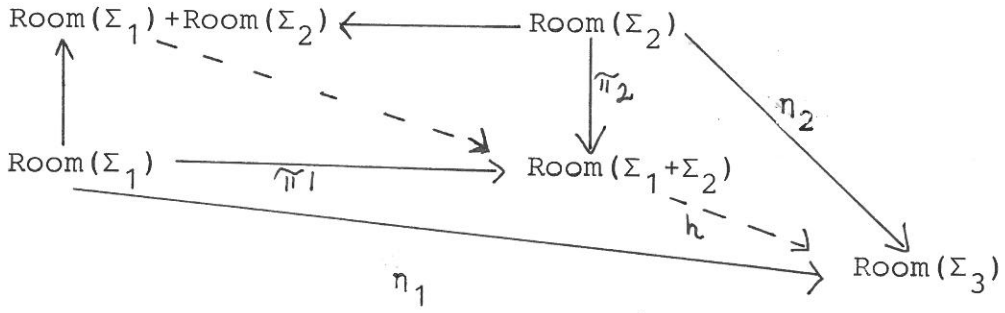
as valuation function. The room morphisms in this diagram connect the frames, structures, datatypes, theories and specifications in $Room(\Sigma^n)$ with those in $Room(\Sigma)$.

Example

If Σ is a ranked signature, one can define Σ^n by $\Sigma^n(K,1) = \epsilon(K,1)^n$. If Σ is the signature Σ_1 in figure 2, then $FRM(\Sigma^n)$ is much smaller than $FRM(\Sigma)^n$ and firing sequences like <g,*...*> <f_1,*...*> <r,*...*> must be allowed. Normal firing sequences like <g...g> <f_1,f_1...f1,f_2><f_2,...f_2,f_1><r...r> are not in $n \times STR(\epsilon)$, but they are so tightly coupled members of $STR(\epsilon)^n$ that "copies" is a more appropriate term than arrays.

Def.13. A gallery admits records iff SIGN has sums - for any Σ_1, $\Sigma_2 \in |SIGN|$ there is a signature $\Sigma_1 + \Sigma_2$ and morphisms $\pi 1:\Sigma_1 \longrightarrow \Sigma_1+\Sigma_2$, $\pi 2:\Sigma_2 \longrightarrow \Sigma_1+\Sigma_2$ such that for any $\zeta 1:\Sigma_1 \longrightarrow \Sigma_3, \zeta 2:\Sigma_2 \longrightarrow \Sigma_3$ there is $\zeta:\Sigma_1+\Sigma_2 \longrightarrow \Sigma_3$ satisfying: $\zeta 1=\zeta \circ \pi 1, \zeta 2=\zeta \circ \pi 2$.

If a gallery admits records, we have the diagram



where $Room(\Sigma_1)+Room(\Sigma_2)$ has $FRM(\Sigma_1)+FRM(\Sigma_2)$ as framesets $STR(\Sigma_1)\times STR(\Sigma_2)$ as structure category, and

$$Val(m_1,m_2)e = \underline{if}\ e\in FRM(\Sigma_1)\ \underline{Then}\ val_{\Sigma 1}(m_1)e\ \underline{else}\ Val_{\Sigma 2}(m_2)e$$

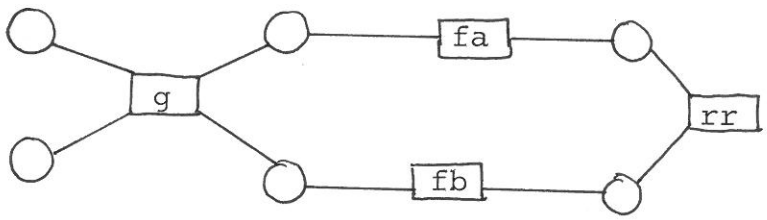
as valuation function. The room morphisms in this diagram give the connection between the frames, structures, datatypes, theories and specifications in $Room(\Sigma_1+\Sigma_2)$ and those in $Room(\Sigma_1)$ and $Room(\Sigma_2)$.

Example

If Σ_1 and Σ_2 are ranked signatures, one can define $\Sigma_1+\Sigma_2$ by $\Sigma_1+\Sigma_2(k,l) = \Sigma_1(k,l)+\Sigma_2(k,l)$. Suppose Σ_1 and Σ_2 are the signatures in fig. 2, so

$$\Sigma_1+\Sigma_2(2,2) = (g,gg)\quad \Sigma_1+\Sigma_2(2,0) = (r,rr)\quad \Sigma_1+\Sigma_2(1,1) = (f1,f2,fa,fb)$$

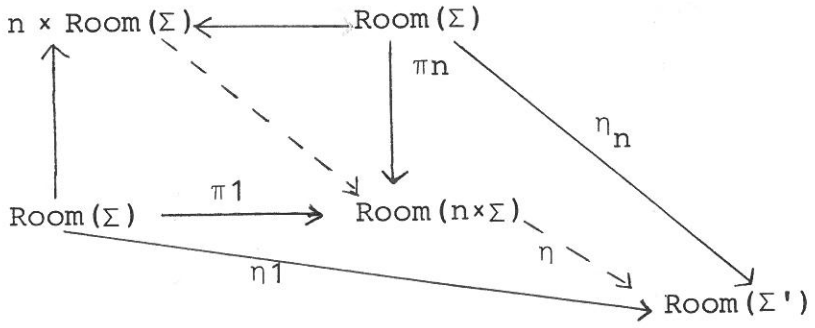
and a typical member of $FRM(\Sigma_1+\Sigma_2)$ is



This frame is not the image of a frame in $FRM(\Sigma_1)+FRM(\Sigma_2)$.
If m is a $\Sigma_1+\Sigma_2$-structure forgetting the Σ_2-symbols (Σ_1-symbols) gives a Σ_1-structure (Σ_2-structure).
This gives the map from $STR(\Sigma_1+\Sigma_2)$ to $STR(\Sigma_1)\times STR(\Sigma_2)$ but this map is not an injection because a Σ1-firing-sequence can be merged with a Σ2-firing-sequence in different ways.

Def.14. A gallery admits arrays iff SIGN has copowers for any signature Σ there is a signature $n\times\Sigma$ and morphisms $\pi_1 \ldots \pi_n$: $n\times\Sigma \longrightarrow \Sigma$ such that for any $\eta_1 .. \eta_n$: $\Sigma' \longrightarrow \Sigma$ there is $\eta: n\times\Sigma \longrightarrow \Sigma'$ satisfying $\eta_1 = \eta \circ \pi_1 .. \eta_n = \eta \circ \pi_n$.

If a gallery admits arrays, we have the diagram



where $n \times \text{Room}(\Sigma)$ has $n \times \text{FRM}(\Sigma)$ as its frame set, $\text{STR}(\Sigma)^n$ as its structure category and

$$\text{Val}(m_1 \ldots m_n)(e) = \underline{\text{case}}\ e \in \text{FRM}(\Sigma_i)\ \underline{\text{of}}\ \text{Val}_{\Sigma}(m_i)e\ \underline{\text{end}}$$

as its valuation function. The room morphisms in this diagram connect the frames, structures, datatypes, theories and specifications in $\text{Room}(\eta\times\Sigma)$ with those in $\text{Room}(\Sigma)$.

Example

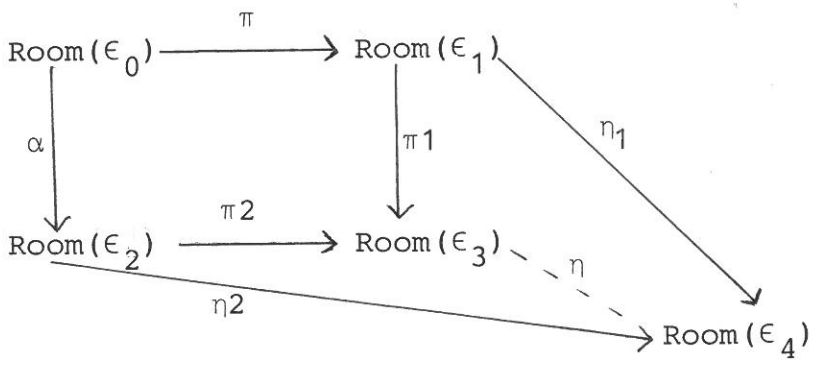
If Σ is a ranked signature, one can define $n \times \Sigma$ by $[n\times\Sigma](k,l) = n \times [\Sigma(k,l)]$. If Σ is the signature in figure 2, then $\text{FRM}(n\times\Sigma)$ is much larger than $n \times \text{FRM}(\Sigma)$. Forgetting symbols gives a map from $\text{STR}(n\times\Sigma)$ to $\text{STR}(\Sigma)^n$ but this map is not an injection because firing sequences can be merged in many different ways. —

Def.15. A gallery admits parameters iff SIGN has pushouts - for signature morphisms $\pi:\Sigma_o \to \Sigma_1$, $\alpha:\Sigma_o \to \Sigma_2$ we have signature Σ_3 and morphisms $\pi 1:\Sigma_1 \to \Sigma_3$, $\pi 2:\Sigma_2 \to \Sigma_3$ such that

(Comm) $\pi 1 \circ \pi = \pi 2 \circ \alpha$

(Uni) for any $\zeta 1:\Sigma_1 \to \Sigma_4, \zeta 2:\Sigma_2 \to \Sigma_4$ there is $\zeta:\Sigma_3 \to \Sigma_4$ satisfying $\zeta = \zeta \circ \pi 1, \zeta 2 = \zeta \circ \pi 2$.

If a gallery admits parameters, one has the diagram



where Σ_0 is the signature of the formal parameter room Σ_1 is the signature of the formally parametrised room, Σ_2 is the signature of the actual parameter room, and Σ_3 is the signature of the actually parametrised room. The room morphisms in this diagram give the connection between the frames, structures, datatypes, theories and specifications in Room(Σ_3) and those in Room(Σ_2), Room(Σ_1) and Room(Σ_0).

There are other properties of the signature category SIGN that help in building complex rooms from simple rooms:
We could have defined "gallery G admits routines" for "SIGN has exponents" and "Gallery G admits files" for "SIGN has limits of sequences".

Comment When galleries admit records, arrays and parameters, the room morphisms go to the constructed object from its components so component specifications are carried into specifications in the constructed room. This seems to be why the literature on institutions has focussed on these colimit constructions. Is it also the reason why x and → seem more natural than +? Why do records, arrays and parameters appear in most programming languages, while variants and copies have a more twilight existence?

#6 Presentations of rooms and galleries

In this section we describe systematic methods for determining possible Σ-frames and Σ-structures from a signature Σ. Consider a gallery with ranked signatures. A general method for defining FRM(Σ) is to identify Σ-frames with bipartite Σ-graphs by "k-entry 1-exit nodes" labelled by names in Σ(k,1) (see figure 7). When there are only constant, function and relation names in Σ one can identify Σ-frames with Σ-trees or Σ-terms. Note that components of Σ-terms are ordered implicitly, and one may or may not insist that the edges of Σ-graphs and branches of Σ-trees are ordered. In many ranked galleries one allows ranked variables in Σ-frames:

- k-entry 1-exit nodes in Σ-graphs can be labelled by variables of rank(k,1);
- vertices in Σ-trees may be labelled by variables of the appropriate rank;
- variables of the appropriate rank can replace constant, function and relation names in Σ-terms.

Every Σ-frame has a rank(k,1)

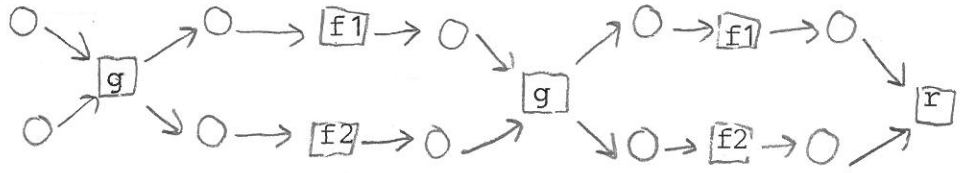
- in a Σ-graph k is the number of source nodes and 1 is the number of sink nodes;
- in a Σ-tree or Σ-term k is the number of variable leaves and 1 is if relation root then 1 else 0;

Now we can define substitution: if e is a Σ-frame of rank(k,1) and for some variables x_i in e we have a frame e_i of the same rank, then e $[e_1/x_1, e_2/x_2 ..]$ is the frame given by substitutions e_1 for x_1, e_2 for x_2...

Example

In figure 7 the graph N and the term T have rank (2,o).
Suppose x is a variable of rank (2,o). If x replaces r in N, we get the graph N_x and $N = N_x [\circ\!\!\rightarrow r / x]$

$N_x[N_x / x]$ =



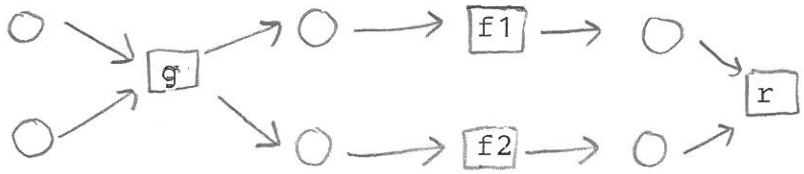
If x replaces r in T we get the term:

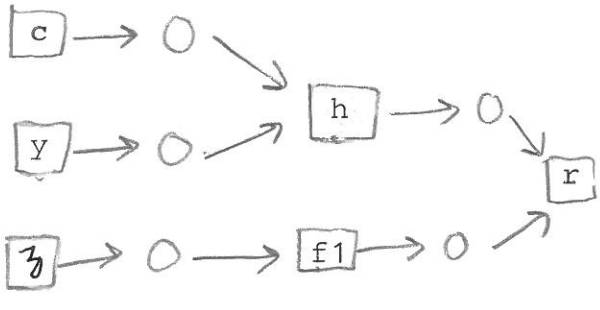
T_x and $T = T_x [r(y, z)/x] = r(h(c,y),f'(z))$

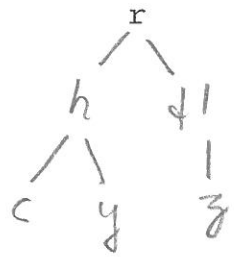
$T_x [T_x / x] = r(h(c,h(c,y)),f1(f1(z)))$

$T [c1/y,c2/z] = r(h(c,c1),f1(c2))$

N =







T=r(h(c,y),f1(3))

$C \in \Sigma(0,1)\ f1, f2 \in \Sigma(1,1)\ h \in \Sigma(2,1)\ g \in \Sigma(2,2)\ r \in \Sigma(2,0)$

Fig.7. Frame presentation

A general method for defining structures in a ranked gallery is: $m \in STR(\Sigma)$ has a set m_o of objects, a function $m_f : m_o^k \to m_o^{l+1}$ for each $f \in \Sigma(k,l+1)$, and a relation $m_r \subset m_o^k$ for each $r \in \Sigma(k,0)$. Occasionally we want to restrict $STR(\Sigma)$ to those m that satisfy constraints like "f_1 is e_1, f_2 is e_2, ... f_n is e_n". In any gallery with substitution this can be done by

$$m \in STR'(\Sigma) \ .\equiv. \ \forall e \in FRM(\Sigma) \, Val_\Sigma(m)e = Val_\Sigma(m)e[f1/e1, .. f_n/e_n]$$

Recursive constraints where the names f_j occur in the frames e_j, may be so restrictive that no structures satisfy the constraints; in some galleries fixed point properties ensure that recursive constraints are satisfied.

Our systematic methods for determining frames and structures in a ranked gallery also work in sorted galleries. When a signature Σ is sorted, it is convenient to label the "circle" vertices in the Σ-graphs by the sort, and insist that m_f and m_r respect the sorts in a Σ-structure m.

Another systematic method for determining the frames in a sorted gallery should be mentioned: Chomsky grammars.

The sorts are given by the non-terminals of the grammar, the elements of $\Sigma(\sigma,\tau)$ are given by the productions, and the Σ-frames are the words in the language generated by the grammar. If the grammar is context free, τ has length 1 in non-empty $\Sigma(\sigma,\tau)$ and Σ-trees or Σ-terms may be used instead of Σ-graphs.

The grammatical way of determining frames in galleries is so useful that one often gives a common grammar Γ_o to a gallery, and defines the Σ-frames using both Σ and Γ_o. Once the values of the Σ-frames, not involving Γ_o, have been defined in Σ-structures, the values of all Σ-frames can be defined using semantic functions for the Γ_o-productions. This method can also be used for presenting a room <FRM,STR,Val> - some grammar Γ determines FRM and Val.

Example 4.

Suppose we want to extend some gallery Base (g) by adding conditional equations as new Σ-frames for each signature Σ. We can introduce the common grammar Γ_o:

<sentence>::= <term>=<term>|<sentence>—><sentence>

and define the new Σ-frames as the sentences given by G_o when both sides of every equation are old Σ-frames of the same sort. The old evaluation function can be extended by:

$$m \models t = t' \quad .\equiv. \quad Val_\Sigma(m)t = Val_\Sigma(m)t'$$

$$m \models e \rightarrow e' \quad .\equiv. \quad m \not\models e \text{ v } m \models e'$$

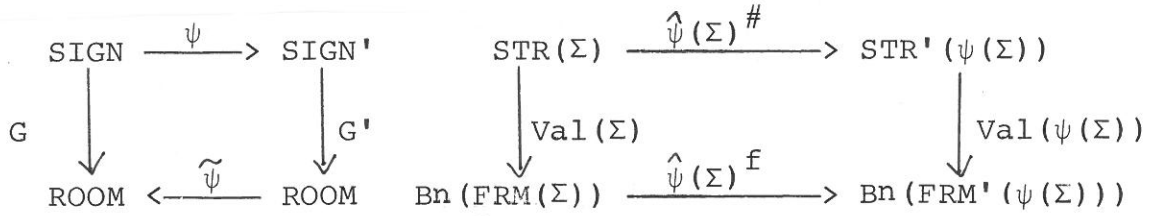
where t,t' are old Σ-frames, m is a Σ-structure, and e,e' are new Σ-frames.

#7 Gallery morphisms

In this section we introduce gallery morphisms; the key to achieving two of the three aims of our unified theory: galleries are particular theories, one should be able to combine galleries and transfer results from one gallery to another. Intuitively there is a morphism Ψ from a gallery G to a gallery G' if every room in G corresponds to a room in G'. This is made precise in:

Def.16. A gallery morphism from G to G' consists of a functor Ψ from SIGN to SIGN' and for each signature Σ a room morphism $\hat{\psi}(\Sigma)$ from $G'(\psi(\Sigma))$ to $G(\Sigma)$

The picture is



Comment In such a gallery morphism we have

$$Val_{\Sigma}(\psi^{6}(e'),m)=Val'_{\psi(\Sigma)}(e',\psi^{\#}(m))$$

for $\Sigma \in SIGN$, $m \in |STR(\Sigma)|$, $e' \in FRM'(\psi(\Sigma))$.
In [G] there is an extra requirement on institution morphisms: $\Psi^{6}:\Psi;FRM' \longrightarrow FRM$, $\Psi^{\#}:STR \longrightarrow \Psi;STR'$ are natural transformations.

Example

There is a morphism from the one room gallery {R} to the one room gallery {R'} if and only if there is a room morphism from R' to R.

Example 5.

If gallery G is included in gallery G', we have $SIGN \subset SIGN'$ and $G(\Sigma) = G'(\Sigma)$ for each $\Sigma \in SIGN$
so there is a gallery morphism from G to G' given by:

$$\psi(\Sigma)=\Sigma, \hat{\psi}(\Sigma)^{6}e'=e', \hat{\psi}(\Sigma)^{\#}m=m'$$

One still has this morphism if $FRM'(\Sigma) \subseteq FRM(\Sigma)$ and $STR(\Sigma) \subseteq STR'(\Sigma)$; In example 4 we have a morphism from the new gallery to the old because the new gallery had not only the frames of the old gallery but also sentences.

Every gallery G determines an institution Inst(G) by: drop all frames that are not sentences, keep all signatures and structures. Example 5 shows that there is a gallery morphism from G to Inst(G); with example 4 we see how grammars can give a morphism from G to some "basic" gallery Base(G).

Figure 8 shows what happens to frames, structures, data types, truth values, when we have a gallery morphism from G to G'.

$$
\begin{array}{ll}
\text{G-data type} & \xrightarrow{\hat{\psi}^{+}} \text{G'-data type} \\
\text{G-truth value} & \xrightarrow{\hat{\psi}^{+}} \text{G'-truth value} \\
\text{G-realised data type} & \xrightarrow{\hat{\psi}^{+}} \text{G'-realised data type} \\
\text{G-closed data type} & \xrightarrow{\hat{\psi}^{+}} \text{G'-closed data type} \\
\text{G-structure} & \xrightarrow{\hat{\psi}^{\#}} \text{G'-structure} \\
\text{G-theory} & \xrightarrow{\hat{\psi}^{\#}} \text{G'-theory} \\
\text{G'-frame in FRM'}(\psi(\Sigma)) & \xrightarrow{\tilde{\psi}^{6}} \text{G-frame in FRM}(\Sigma) \\
\text{G'-truth value in P(FRM'}(\psi(\Sigma)) & \xrightarrow{\hat{\psi}^{6}} \text{G-truth value in P(FRM}(\Sigma)) \\
\text{G'-specificaiton in FRM'}(\psi(\Sigma)) & \xrightarrow{\hat{\psi}^{6}} \text{G-specification in FRM}(\Sigma) \\
\text{G-signature} & \xrightarrow{\psi} \text{G'-signature} \\
\text{G-signature morphism} & \xrightarrow{\psi} \text{G'-signature morphism}
\end{array}
$$

Fig. 8. Properties preserved by a gallery morphism from G to G'

There are three natural logics for reasoning about rooms and galleries. The simplest is first order logic with sorts for frames and structures. The syntax of this intrinsic logic for a room(FRM,STR,Val) is given by

<term> ::= <Variable> | <Structure> | <Frame> | <term>(<term>)

<formula> ::= <term> = <term> | ¬<formula> | (<formula> → <formula>)
| ⊥ | (<formula> ∧ <formula> | (<formula> ∨ <formula>)
| ∃<Variable>.<Formula> | ∀<Variable>.<Formula>

where <Structure> = STR, <Frame> = FRM and <Variable> = $(m,m_1,m_2.....e,e_1,e_2....M,M_1M_2....E,E_1,E_1,E_2....)$. The semantics of the logic will give a meaning to each well formed term and formula. Suppose we have an assignment S which gives a structure to each free m-variable, a frame to each free e-variable, a structure class to each free M-variable, and a frame class to each free E-variable in a term t or formula F. Such an assignment gives a value to the term t or formula F by:

$[[v]] [s] = s(V)$

$[[Str]][s] = Str$

$[[Frm]][s] = Frm$

$[[t_1(t_2)]][s] = Val([[t_1]][s])([[t_2]][s])...[[t_1]][s]\in|STR|,[[t_2]][s]\in FRM$

$\models [[t_1(t_2)]][s]$.≡.

$--[[t_1]][s] \subset STR, [[t_2]][s] \in STR\,[t_2][s]\in[t_1][s]$

or $[[t_1]][s] \subset FRM, [t_2][s] \in FRM$

$[[\bot]][s] = 0$

$\models [[t_1=t_2]][\ ]$.≡. $([[t_1]][s] = [[t_2]][s])$

$[[F]][s]$.≡. not $\models [[F]][s]$

$\models [[F_1\rightarrow F_2]][S]$.≡. $\models[[F_1]][S]$ implies $\models [[F_2]][S]$

$\models [[F_1\wedge F_2]][s]$.≡. $\models[[F_1]][s]$ and $\models [[F_2]][s]$

$\models [[F_1\vee F_2]][S]$.≡. $\models[[F_1]][S]$ or $\models [[F_2]][S]$

$\models [[\forall v.F]][S]$.≡. $\models[[F]][S']$ for all $s' = s[c/v]$

$\models [[\exists v.F]][S]$.≡. $\models[[F]][s']$ for some $s' = s[c/v]$

where c ranges over structures (frames, structure classes, frame classes) when v is an m-variable (e-variable, M-variable, F-variable). We also use = and ≡. when values are either 0 or 1.

Example

The formula "E(e) → (m(e) = m_1(e))" expresses the proposition "$m_1 \in [\![m]\!]_E$" where $[\![\ ]\!]_E$ is the specification given by the frame set E. The formula "M(m) → (M(e) = M(e_1))" expresses the proposition "$e_1 \in [\![e]\!]_M$" where $[\![\ ]\!]_M$ is the theory given by the structure set. Simple logical arguments give our earlier assertions

$$E_1 \subseteq E_2 \quad \text{implies} \quad [\![\ ]\!]_{E_2} \subseteq [\![\ ]\!]_{E_1}$$

$$M_1 \subseteq M_2 \quad \text{implies} \quad [\![\ ]\!]_{M_2} \subseteq [\![\ ]\!]_{M_1}$$

the logic captures our informal arguments about rooms. ∎

In the first order logic for a gallery one has a structure, frame, structure class, and frame class sorts for each signature Σ. There are infinitely many variables for each sort, and an assignment s has to respect the sorts. There is a constant term φ for each gallery logic are given by the semantics of the room logic and the clauses.

$$[\![\phi(t)]\!]\ [\ ] = \phi^{\#}([\![t]\!][s]) \qquad -- [\![t]\!][s] \in |STR(\Sigma_2)|$$

$$[\![\phi(t)]\!]\ [s] = \phi^{6}([\![t]\!][s]) \qquad -- [\![t]\!][s] \in FRM(\Sigma_1)$$

when φ is a signature morphism from Σ_1 to Σ_2.

Example

The clause "$Val_1(\phi^{\#}(m_2))(e_1) = Val_2(m_2)\phi^{6}(e_1)$" in the definition of a gallery gives the law

$$\phi(m_2)(e_1) = m_2(\phi(e_1))$$

in our first order logic - for any assignments we have $s(m_2) \in STR(\Sigma_2)$, $s(m_1) \in FRM(\Sigma_1)$, $\models [\![\phi(m_2)(e_1) = m_1(\phi(e))]\!][s]$.

As most of the informal arguments about rooms and galleries are captured by our first order logics, one may well ask why look for other "intrinsic logics". One answer is there seems to be no natural way of handling structure morphisms; another answer is that one should be able to formalise all informal arguments about rooms and galleries. The second answer is reasonable in view of the fact that there are many suggestions for program logics, in which one can formalise informal arguments about programs, but no one suggestion is generally accepted.

We can devise two more natural logics of rooms and galleries from the idea of data types. For the datatypes of a room <FRM,STR,Val> can define

$$d \times d' = (d(e) \times d'(e) \mid e \in FRM)$$

$$d \to d' = (f_e \mid f_e : d(e) \to d'(e) \text{ for all } e \in FRM)$$

$$1 = (\mid \mid e \in FRM)$$

so $\mathcal{B}n(FRM)$ is a Cartesian closed category. It is known that Cortesian closed categories correspond to typed lambda calculi, so our second intrinsic logic for a room can be the lambda calculus with the syntax

<type> ::= <Structure>|1|<type> × <type>|<type> → <type>

<term> ::= <Variable>|*|<Structure Morphism>
|(<term>,<term>)|π1(<term>)|π2(<term>)
|<term>(<term>)|λ<Variable>.<term>

<formula> ::= <term> = <term>

where <Structure> = STR, <Variable> has an infinite supply of variables of each type and <Structure Morphism> has a symbol

σ of type $m_1 \to m_2$ for each morphism from structure m_1 to structure m_2. An assignment s from variables to values of the appropriate type also gives datatypes to terms and formulae by

$$[\![v]\!][s_e] = s_e(v)(e)$$

$$[\![*]\!][s_e] = 1$$

$$[\![\sigma]\!][s_e] = Val(\sigma)(e)$$

$$[\![(t_1,t_2)]\!][s_e] = [\![t_1]\!][s_e] \times [\![t_2]\!][s_e]$$

$$[\![\pi 1(t_1,t_2)]\!][s_e] = [\![t_2]\!][s_e]$$

$$[\![\pi 2(t_1,t_2)]\!][s_e] = [\![t_2]\!][s_e]$$

$$[\![t_1(t_2)]\!][s_e] = [\![t_1]\!][s_e]([\![t_2]\!][s_e]) \quad \text{-- explained later}$$

$$[\![\lambda v.t]\!][s_e] \quad \text{-- defined later}$$

$$\models [\![t_1=t_2]\!][s_e] \quad .\equiv. [\![t_1]\!][s_e] = [\![t_2]\!][s_e]$$

where $[s_e]$ abbreviates $[s](e)$.

Notice that an assignment s to the variables in a formula $t_1 = t_2$ gives a truth value in Ω to the formula for each frame e we have either $t_1 = t_2[s_e] = 0$ or $t_1 = t_2[s_e] = 1$. In general assignments give data types to terms; if the term t_1 receives a data type f_e in $d \to d'$ and the term t_2 receives a datatype V_e in d, then the term $t_1(t_2)$ receives the data type $(f_e(V_e) \mid e \in FRM)$ in d', if t_1 is a lambda term $\lambda v.t$, then $t_1(t_2)$ should receive the same data type as the term given by substituting t_2 for the variable v in t_1. This motivates

$$[\![\lambda v.t]\!][s_e] = f_e \text{ where } f_e(v_e) = [\![t]\!][s'_e] \text{ and s' is s except } s'(v)(e) = v_e$$

Example

The formula $\sigma_1(\sigma_2)(v)) = \sigma_3(\sigma_4(v))$ expresses the structure morphism equation $\sigma_1 \circ \sigma_2 = \sigma_3 \circ \sigma_4$. This equation is meaningful iff there are structure types A,B,C,D such that σ_1: B → D, σ_2: A → B, σ_3: C → D, σ_4: A → C. The formula is meaningful iff a is a variable of type A. For an assignment s we have

$$[\![a]\!][s_e] \in Val(A)e$$

$$[\![\sigma_2(a)]\!][s_e] = Val(\sigma_2)([\![a]\!][s_e]$$

$$[\![\sigma_1(\sigma_2(a))]\!][s_e] = Val(\sigma_1) \circ Val(\sigma_2)([\![a]\!][s_e])$$

$$[\![\sigma_3(\sigma_4(a))]\!][s_e] = Val(\sigma_3) \circ Val(\sigma_4)([\![a]\!][s_e])$$

$$\models [\![\sigma_1(\sigma_2(a)) = \sigma_3(\sigma_4(a))]\!][s_e]$$

$$.\equiv. Val(\sigma_1) \circ Val(\sigma_2)([\![a]\!][s_e]) = Val(\sigma_3) \circ Val(\sigma_4)([\![a]\!][s_e])$$

so the equation is true for all assignments if and only if $Val(\sigma_1) \circ Val(\sigma_2) = Val(\sigma_3) \circ Val(\sigma_4)$. ∎

In order that the semantics of the lambda calculus for a room be as simple as possible, we have followed (S1) not other authors who would define a $[\![t]\!]$ as a morphism from the type of the free variables of t. However, we do have to say something about variables of type d where d(e) in the empty set for some frame e. If term t has a free variable v of type d and d(e) = 0, then $[\![t]\!][s_e]$ is undefined and

$$\models [\![t_2 = t']\!][s_e] .\equiv. [\![t']\!][s_e] \text{ is undefined.}$$

Notice what happens when v is bound in $\lambda v.t$, the function $[\![\lambda v.t]\!][s_e]$ is the unique function defined on the empty set.

Because the set of data types Bn(FRM) is a particular kind of Cartesian Category, we have other intrinsic logics of rooms and gallery. In a later paper we will look at the dependent type theory,

advocated by Martin Lof and used in the languages PEBBLE ML and PL/CV3. Here we focus on the fact that Bn(FRM) becomes a topos with the definition

$$P(d) = \{\text{set families } s \text{ such that } s_e \subseteq d(e) \text{ for all } e \in FRM\}$$

$$\Omega = \{(0,1) \mid e \in FRM\}$$

Since topoi correspond to higher order type theory, our last intrinsic logic for a room can be the type theory with the syntax

<type> ::= <Structure>|Π|Ω|<type> × <type>|<type> → <type>
|P<type>

<type> ::= <Variable>|×|<Structure Morphism>
|(<term>,<term>|π,(<term>)|π_2(<term>)
|<term>(<term>)|λ<Variable>.<term>

<formula> ::= <term> = <term>|<term> $\in$ <term>
|(<formula> $\wedge$ <formula>|(<formula> $\vee$ <formula>
|(<formula> → <formula>)|$\perp$| <formula>
|$\exists$<Variable>.<formula>|$\forall$<Variable>.<formula>

where <Variable> has an infinite supply of variables of all types and <Structure> <Structure Morphism> are as before. The semantics of our logic is given by extending the lambda calculus definition of a data type for each term and formula under an assignment s, the following clauses must be added:

$$\models [\![t_1 \in t_2]\!][s_e] \quad .\equiv. \quad [\![t_1]\!][s_e] \in [\![t_2]\!][s_e]$$

$$\models [\![(F_1 \wedge F_2)]\!][s_e] \quad .\equiv. \quad \models [\![F_1]\!][s_e] \text{ and } \models [\![F_2]\!][s_e]$$

$$\models [\![(F_1 \vee F_2)]\!][s_e] \quad .\equiv. \quad \models [\![F_1]\!][s_e] \text{ or } \models [\![F_2]\!][s_e]$$

$$\models [\![(F_1 \rightarrow F_2)]\!][s_e] \quad .\equiv. \quad \models [\![F_1]\!][s_e] \text{ implies } \models [\![F_2]\!][s_e]$$

$$[\![\bot]\!][s_e] \quad = \quad 0$$

$$\models [\![\neg F_1]\!][s_e] \quad .\equiv. \text{ not } \models [\![F_1]\!][s_e]$$

$$\models [\forall v.F][s_e] \quad .\equiv. \quad \models [\![F]\!][s'_e] \text{ for all } s' = s[c/v]$$

$$\models [\![\forall v.F]\!][s_e] \quad .\equiv. \quad \models [\![F]\!][s'_e] \text{ for some } s' = s[c/v]$$

where e ranges over the type of v.

Can we talk about specifications in our type theory? For each frame set Ec FRM, the type $P(m)$ has the set family

$$m(F)_k = \underline{\text{if}}\ e \in E\ \underline{\text{then}}\ \text{Val}(m)e\ \underline{\text{else}}\ 0$$

so each $P(n)$ "contains" the truth values. If $[\![\ ]\!]_E$ is the specification given by the frame set E, then the proposition $m_1 \in [\![m]\!]_E$ can be expressed by "$m_1(E) = m(E)$", but this is not a formula of our type theory. This suggests an extension of our type theory allow variables over structures and frame sets. Answering the question "can theories in a room be expressed in our type theory" suggests the extension - allow variables over frames and structure sets. If we made these two extensions to our type theory by introducing a "universal type of structures", the resulting theory would include all three of our intrinsic logics of rooms. Instead of making this extension, let us define the intrinsic type theory of a gallery. There are types for each signature as well as types for each structure. There are infinitely many variables of each

type, and an assignment s has to respect the types. There is a constant term for each signature morphism ϕ. The semantics of the type theory for the gallery are given by the semantics of the type theory for rooms and

$$[\![\phi(t)]\!][s_e] = \phi^+([\![t]\!][s_e])$$

where ϕ is a signature morphism to the room containing the data type $[\![t]\!][s]$. A formula F is valid iff $= [\![f]\!][s_e]$ for all frames e and all assignments s to the free variables in F. If the formula F has a free variable of type d and some d(e) is empty, then F says nothing about e such that d(e) is empty, so the validity of F is unaffected by such e. Notice also that $\models [\![F]\!][s_e]$.≡. $\models [\![\neg\neg F]\!][s_e]$ so

$$F \text{ is valid } .\equiv. \neg\neg F \text{ is valid}$$

because the set of truth values Ω is a Boolean algebra. We would have had an intuitionistic type theory if rooms had been defined with valuation functions from FRM to $\mathcal{B}n(STR)$ instead of from STR to $B_n(FRM)$. A study of this intuitionistic logic would be interesting because $\mathcal{B}n(STR)$ as the topos $SET^{STR^{op}}$ contains a full and faithful copy of the category STR.

Comment

Consider a room morphism ϕ from $\langle FRM_1, STR_1, Val_1\rangle$ to $\langle FRM_2, STR_2, Val_2\rangle$. The associated functor ϕ^+ from $\mathcal{B}n(FRM_2)$ to $\mathcal{B}n(FRM_1)$ is what the category theorists call a logical morphism; it gives a translation from the logic of $\langle FRM_2, STR_2, Val_2\rangle$ to the logic of $\langle FRM_1, STR_1, Val_1\rangle$. Another way of capturing the room morphism is to combine the logics of the two rooms, to consider the logic for either $\mathcal{B}n(FRM_1+FRM_2)$ = $Bn(FRM_1| \times \mathcal{B}n(FRM_2)$ or $Bn(FRM_1) + Bn(FRM_2)$. In our type theory we have adopted the last of the three alternatives - data types are given by the sum of $Bn(FRM(\Sigma))$ over all signatures Σ.

Our three natural logics do not exhaust the richness of the set of data types Bn(FRM). For any two data types d_1 and d_2 we have data types

$$d_1 \cup d_2 = (d_1(e) \cup d_2(e) \mid e \in FRM)$$

$$d_1 \cap d_2 = (d_1(e) \cap d_2(e) \mid e \in FRM)$$

$$d_1 \subset d_2 = (d_1(e) \subset d_2(e) \mid e \in FRM) \quad \text{-- a truth value.}$$

For any room <FRM,STR,Val> the data type 1 represents FRM and we have data types

$(\lambda n.Val(m)e \mid e \in FRM)$ --- the valuation function

$(STR \mid e \in FRM)$ --- the set of structures

$(\phi^6(e) \mid e \in FRM)$ --- morphisms from the room

$(\sigma_e : Val(m_1)e \to Val(m_2)e \mid e \in FRM)$

where σ_e is given by the functor Val on a structure morphism σ from m_1 to m_2. There must be intrinsic logics for rooms and galleries, that capture more of this algebraic richness of $\mathcal{B}n(FRM)$ than the three logics we have described. However, the richness of $\mathcal{B}n(FRM)$ is not always relevant for particular rooms and galleries; the aim of this section has been to argue for and show how a particular room or gallery can determine an "intrinsic" logic for reasoning in that particular room or gallery.

References

(BG?) R. M. Burstall & J. A. Goguen: An informal introduction to CLEAR, a specification language in R. Boyer & J. Moore, eds. The correctness problem in computer science. Academic Press 1981.

(GB1) J. A. Goguen & R. M. Burstall: Some fundamental algebraic tools for the semantics of computation, Th. Comp. Sci. 31 (1984) pp. 175 - 209, 263 - 296.

(GB2) J. A. Goguen & R. M. Burstall: Introducing Institution in: E. Clarke ed. Proc. Logics of Programs, Springer LNCS 164(1984) pp. 221 - 256.

(M1) J. A. Makowsky: Model theoretic issues in theoretical computer science, Proc. Logic Coll. 82. Florence 1982.

(M2) B. H. Mayoh: Data types as functions, Proc. MFCS 78. Springer LNCS 64(1978) pp. 56 - 70.

(P) Z. Pawlak: Information systems - theoretical foundations, Information Systems 6(1981) pp. 205 - 218.

(S1) D. S. Scott: Relating theories of the λ-calculus in: J. P. Seldin & J. R. Hindley, To H. B. Curry, Essays in combinatory logic, Lambda calculus and foundations. Academic Press 1980.

(S2) D. S. Scott: Domains for denotational semantics, Proc. ICALP 9. Springer LNCS 140(1982) pp. 577 - 613.

(T) A. Tarlecki: Free constructions in algebraic institutions, Proc. MFCS 84. Springer LNCS 176(1984) pp. 526 - 534.