A Calculational Approach to Reverse Specification

UNU/IIST — Seminar n.º 4
5–16 May 1997
Macau

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

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

Abstract

- The Student Records Office of Minho University has run for many years on top of a “home made” information system which was, until a couple of years ago, by and large undocumented.
- The situation became more and more risky until three final year students were encharged of the process of reversing the underlying ORACLE code and of recording as much of the missing information as possible.
- Two of these sudents used ‘ad hoc’ techniques, while the other based his analysis on formal methods. His strategy focussed on reversing the database structure by <u>inverse calculation</u> in the SETS calculus.
- The main outcome of this systematic exercise was a concise formal specification of the whole system structure. This made it possible to formally specify the University’s rules for student assessment and course follow-up as a datatype invariant written on top of the formal specification derived by inverse calculation.
- On the whole, informal and formal methods blended well in rescuing this information system from eventually collapsing down.

Project Info

- **Place of work**: Student Records Office of Minho University, Gualtar Campus, 4700 Braga, Portugal.
- **Main target**: Analysis of the underlying information system in order to identify its high-level structure and behaviour, so that reliable documentation for user and maintenance purposes can be produced.
- **Human resources**: 3 final year students (1st semester of academic year 1994/95)
- **Environment**: ORACLE (Pro-C embedded SQL, SQLPlus).
- **Background**:
  - an earlier system had existed since the mid 1970's, running on a WANG machine and written in WANG BASIC;
  - it became obsolete by the mid 1980's, when it was discontinued and replaced from current ORACLE-based system, built from scratch.
  - by the mid 1990's, the maintenance team had changed twice;
  - current ORACLE system is functioning well but Student Records officers fear that it may collapse due to lack of documentation, increased complexity of University internal regulations, on top of a large amount and diversity of information.

About "Geriatric" Computing

Reverse engineering of program code is the process of constructing a higher level of abstraction of an implementation in order to facilitate the understanding of a system that may be in a "legacy" or "geriatric" state. [GC96]

Terminology

(Following [CG93]):

1. **Forward Engineering**: develop system by moving from abstract specifications to detailed implementations;
2. **Reverse Engineering**: analysis of a system in order to identify components and intended behaviour in order to create higher level abstractions of the system.
3. **Restructuring**: process of creating a logically equivalent system at the same level of abstraction (*e.g.* transforming "spaghetti" code into structured code).
4. **Re-Engineering**: examination and alteration of a system to rebuild it in a new form and subsequent implementation of the new form.

ℹ This project, which falls into category 2, is expected to evolve into category 4.

Our Strategy — a Two-Tier Approach

- **Informal work**:
  - Analysis of data-base files supported by interviews with the system's operators;
  - Generation of conventional "ad hoc" documentation — ER-diagrams, Data Dictionaries, DF-diagrams, Portuguese text.
- **Formal work**:
  - Derivation of a formal model of the system;
  - Rapid prototyping of formal model plus added features;
  - Formal software component (modular) analysis;
  - Preparing for re-engineering (*e.g.* object-orientation).

This talk will concentrate on the formal method work.

Some Details about Informal Work

- Analysis of data-base files — identification of primary and foreign keys, informal properties *etc.*.
- Trial-and-error inference of ER-diagrams.
- A LEX/YACC tool was built for automatic "who needs what" cross-referencing between transactions ("programs") and entities or relationships ("tables"), *e.g.*:

Prog./Tab.	**ALUNOS**	**PERMISSOES**	**VER_ALUNOS**	**ALUNOS_INSCRITOS**	···
act_alunos.pc	A	L	LR		
act_bolseiro.pc	L			LA	
act_dirigente.pc	L			LA	
act_militar.pc	L			LA	
act_regime_estag.pc				A	L
···					

where L = read, I = insert, R = remove, A = modify, W = any (W is typical of SQL forms).

By Feb. 1995 this (sparse) matrix contained 55 columns and 231 lines (192 lines referring to `*.pc` code and 39 to `*.inp` forms).

A couple of tables were found which had no access whatsoever.

- Derivation of DFD description of system behaviour (not made available in technical report [ELS95]).

The Formal Analysis

Informal analysis paved the way to the formal exercise (see report [Vel95]):

- ER-diagrams and Data Dictionaries — they provided a starting point for SETS inverse calculation;
- "Who needs what" matrix — this proved very useful in the identification of the involved formal software components (*cf.* object-encapsulation).

Reversing ER-diagrams

We have seen this in Seminar n.º 3.

Reversing Data Dictionaries

The process of reversing this kind of information, usually associated with data-flow diagrams (DFDs), is a direct one — see the table which follows:

DATA DICTIONARY	SETS
$A = \{B + C\}$	$A \cong 2^{B \times C}$
$A = \{\underline{B} + C\}$	$A \cong B\mathsf{C}$
$A = \{B + \{C\}\}$	$A \cong 2^{B \times D}$ $D \cong 2^{C}$
$A = \{\underline{B} + C\}$	$A \cong B \hookrightarrow 2^{C}$
$A = \{B + (C \mid D)\}$	$A \cong 2^{B \times (C + D)}$
$A = \{\underline{B} + (C \mid D)\}$	$A \cong B \hookrightarrow (C + D)$

However, from a data dictionary it is more difficult to spot local invariants than from an ER-diagram.

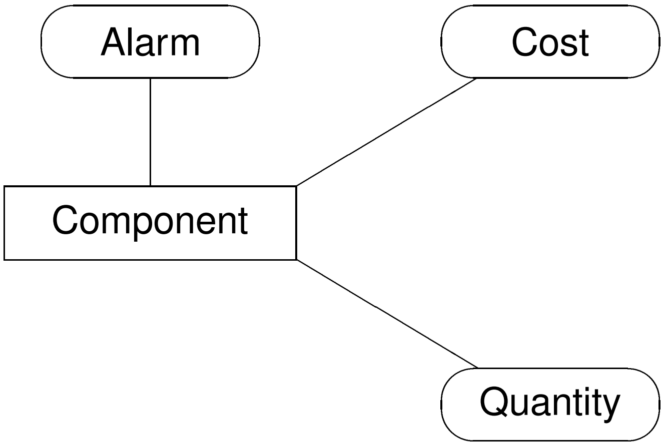
The Abstraction Process

- After obtaining from a given ER-diagram or DD a first structure in SETS, it is necessary to abstract from it (generalize it);
- The idea is factor out accidental detail and emphasize structural relations among the entities.

- We should recall that ERA-entities always originate, when reversed to SETS, “outermost” finite mappings, for instance

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

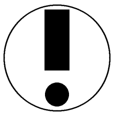
for



etc.

Abstraction process then proceeds as follows:

1st step — Collapse all sub-expressions involving only primitive data sorts (typically, attribute domains) into independent sort variables.



Never collapse entity keys.

For example, in the following abstract structure in SETS

$$\begin{aligned}\Sigma \;\cong\; & (\#Number \hookrightarrow Name \times Address \times Telephone) \\ & \times(\#Car \hookrightarrow 2^{\#Number \times Hours \times Kms})\end{aligned}$$

sub-expression $Name \times Address \times Telephone$ can be collapsed into a single variable A and sub-expression $Hours \times Kms$ can be collapsed into another variable B:

$$\Sigma' \;\cong\; (\#Number \hookrightarrow A) \times (\#Car \hookrightarrow 2^{\#Number \times B}) \qquad (1)$$

2nd step — Replace key identifiers by variables, in order to obtain a structure without direct references to the real system. In the example above, from (1) one might obtain, for instance:

$$\Sigma'' \;\cong\; (\#N \hookrightarrow A) \times (\#C \hookrightarrow 2^{\#N \times B})$$

3rd step — Apply to the structure obtained in the previous step as many SETS laws as possible, **from right to left**, in particular laws which match structural invariants, in order to lift them.

That is, starting from an abstract structure E_ϕ, by applying this step one might obtain a structure E' such that

$$E' \trianglelefteq E_\phi$$

provided that

1. E is an instance of B in some law

$$A \trianglelefteq_f^\phi B$$

in the calculus and E' is the corresponding instance of A.

2. Same as above, but $\phi = \psi \wedge \alpha \circ f$ and law

$$A \trianglelefteq_f^\psi B$$

is valid — in this case one obtains E'_α.

Invariant lifting is determinant in this process.

Sometimes one needs to “guess” invariants which are not captured by the ER-diagram, in order to chose the right law.

For instance, only the concrete invariant can help us in deciding between

$$A \hookrightarrow B \times C \trianglelefteq (A \hookrightarrow B) \times (A \hookrightarrow C) \qquad (2)$$

or

$$A \hookrightarrow (B + C) \trianglelefteq (A \hookrightarrow B) \times (A \hookrightarrow C) \qquad (3)$$

see, respectively, concrete invariants eqd and djd in SETS ongoing course.

The Case Study

1. Derivation of a formal model of the system

- ER-diagram is A3 in size — see extra slides 8.A and 8.B
- Starting Set expression is a finitary product of 47 factors, either powersets or finite mappings — see extra slides 8.12–13.
- Starting (concrete) invariant is a finitary conjunction of 45 predicates — see extra slides 8.13–15.
- Final Set expression is the following finitary product of 14 factors = 13 finite mappings + 1 powerset:

$$
\begin{array}{ll}
SA' \cong & \\
\quad (\#A \hookrightarrow & ((\#C \times Inf01) + ((\#C \times \#P \times Inf02) \times \\
 & ((AL \hookrightarrow ((\#C \times \#P \times Inf03) + (\#C \times \#P \times Inf38))) \times \\
 & ((\#D \times AL) \hookrightarrow Inf04) \times ((\#D \times AL) \hookrightarrow Inf05) \times (Inf06 + 1) \\
 & (AL \hookrightarrow (\#C \times Inf10)) \times (AL \hookrightarrow Inf20) \times ((AL \times \#D) \hookrightarrow Inf24) \times \\
 & (Inf27 + 1) \times ((AL \times \#D) \hookrightarrow Inf32) \times ((AL \times \#D) \hookrightarrow Inf33) \times \\
 & ((AL \times \#D \times EP) \hookrightarrow Inf34) \times ((AL \times \#D) \hookrightarrow Inf35) \times \\
 & (AL \hookrightarrow (\#C \times Inf42)) \times ((\#C \times \#P \times Inf48) + 1) \times (AL \hookrightarrow Inf50) \times \\
 & (AL \hookrightarrow (\#C \times Inf53)) \times (AL \hookrightarrow (\#P \times Inf55))))) \times \\
\quad (\#C \hookrightarrow & ((\#P \times Inf12) \times ((AL \times \#D) \hookrightarrow Inf11) \times ((\#P \times Inf13) + 1) \times \\
 & 2^{Inf21} \times 2^{\#D \times Inf22} \times 2^{\#D \times Inf23} \times \\
 & ((AL \times EP \times CH \times \#D) \hookrightarrow Inf28) \times (AL \hookrightarrow Inf40))) \times \\
\quad (\#D \hookrightarrow & ((\#DP \times Inf16) \times (AL \hookrightarrow (\#C \times DC \times Inf52)) \times \\
 & (AL \hookrightarrow (\#C \times Inf54)) \times 2^{AL} \times (AL \hookrightarrow (\#C\#T \times Inf43)))) \times \\
\quad (\#P \hookrightarrow & (Inf47 \times 2^{\#D \times Inf46} \times (Inf49 + 1))) \times \\
\quad (\#DP \hookrightarrow & Inf14) \times \\
\quad (\#DC \hookrightarrow & (\#DP \times \#E \times Inf18)) \times \\
\quad (\#E \hookrightarrow & Inf19) \times \\
\quad (\#ES \hookrightarrow & (\#E \times Inf25)) \times \\
\quad (\#ET \hookrightarrow & Inf26) \times \\
\quad ((AL \times EP \times CH) \hookrightarrow & Inf29) \times \\
\quad (\#H \hookrightarrow & Inf31) \times \\
\quad 2^{DAC \times DAN} & \times \\
\quad (\#O \hookrightarrow & (Inf39 \times 2^{PR \times Inf45})) \times \\
\quad (\#T \hookrightarrow & Inf44))_{\phi}
\end{array}
$$

Legend:

A — student records
C — degree courses (*e.g.* “civil engineering”, “medicine”)
D — courses (*i.e.* disciplines, *e.g.* “formal methods”)
P — degree couse plans
D — departments
DC — teachers
E — addresses
ES — schools (faculties)
ET — other universities or higher-education schools
$AL \times EP \times CH$ — exams information
H — student access conditions (“habilitações”)
$DAC \times DAN$ — course equivalence table
O — operator control subsystem
T — text processing data relative to certificate automatic generation.

See slides 10 and 11 for a pictorial description of the involved Set-models.
Inspection of these expression trees proved useful in the system analysis.

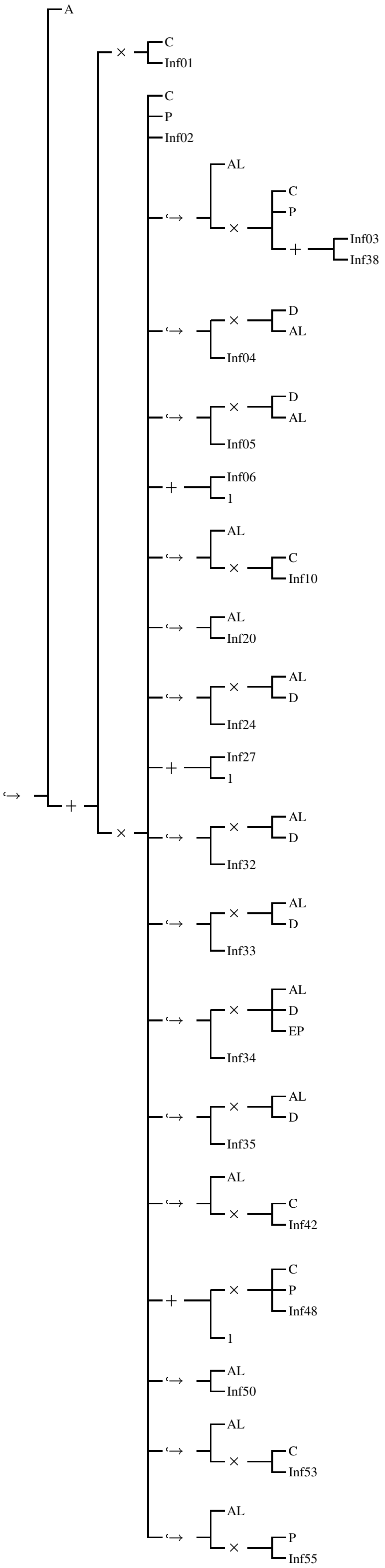
- Fragment of (remaining) local invariant:

Auxiliary function(s):

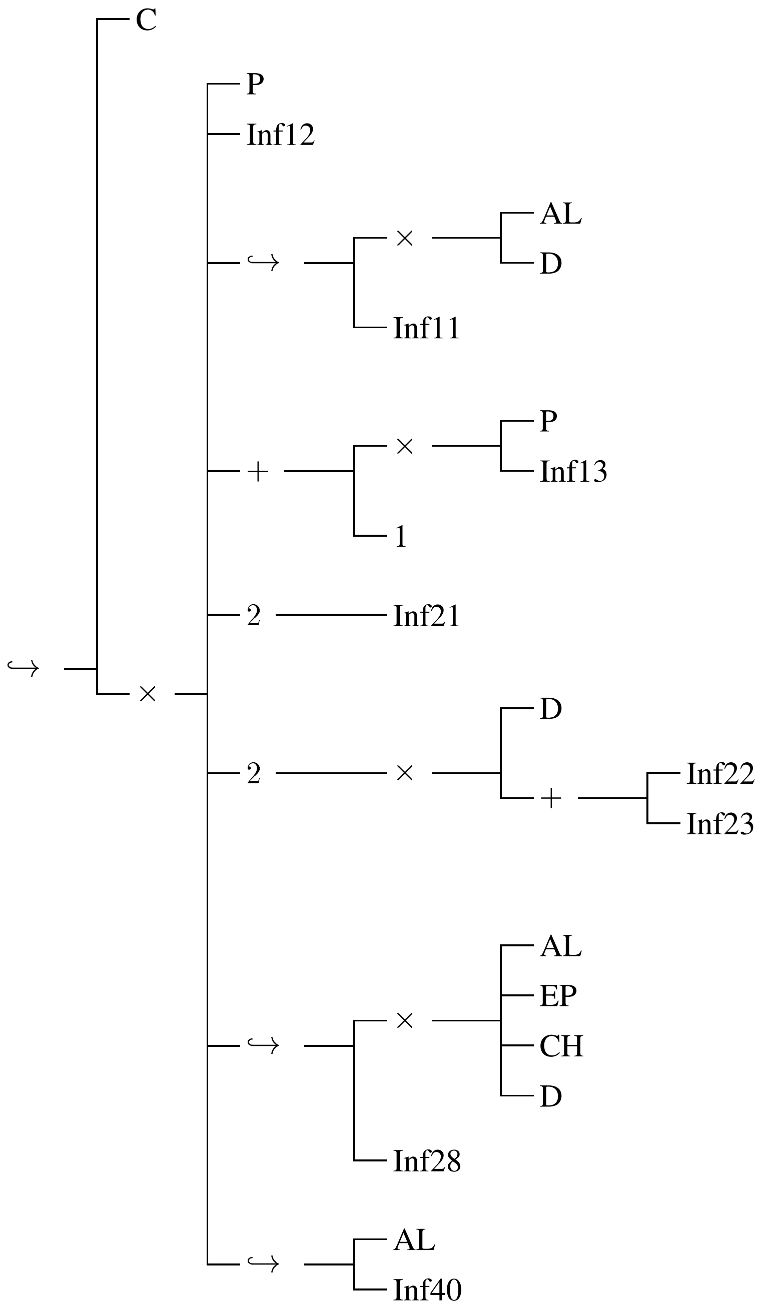
$$
\begin{array}{rcl}
sel_{j=1,2} & : & A \hookrightarrow (B_1 + B_2) \longrightarrow A \hookrightarrow B_j \\
sel_j(\sigma) & \stackrel{\mathrm{def}}{=} & \left(\begin{array}{c} a \\ b \end{array} \right)_{a \in dom(\sigma) \wedge \sigma(a) = i_j(b)}
\end{array}
$$

REMARK 1: **Expression tress for Students and Courses**

Students (σ_1)

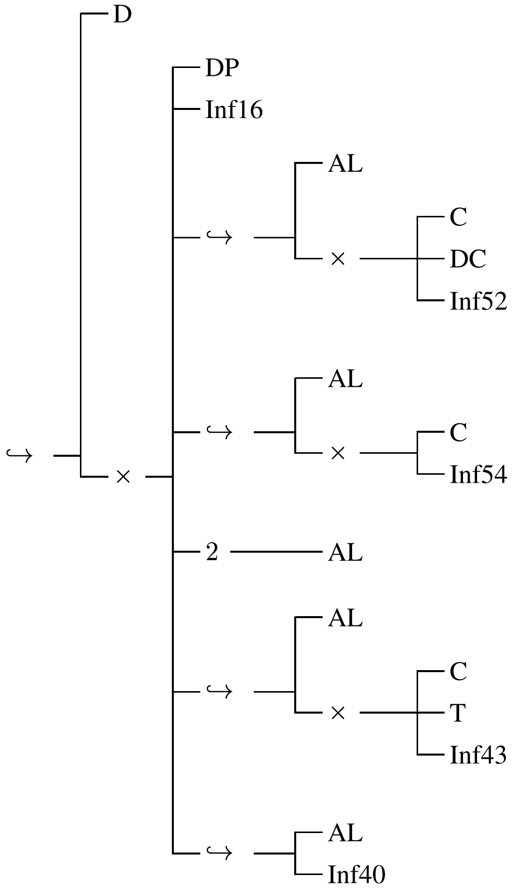


Degree courses (σ_2)

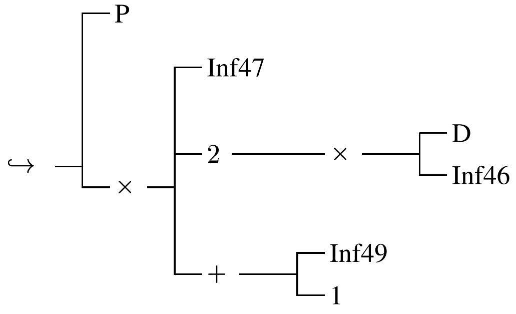


REMARK 2: **Other expression trees**

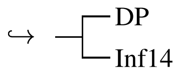
Courses (σ_3)



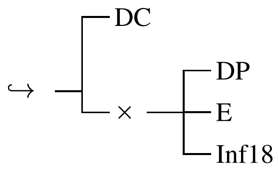
Plans (σ_4)



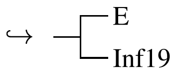
Departments (σ_5)



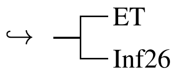
Teachers (σ_6)



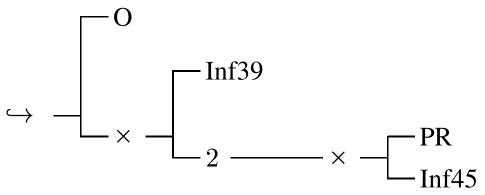
Addresses (σ_7)



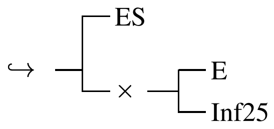
Other universities (σ_9)



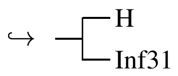
Operators (σ_{13})



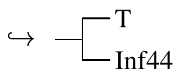
Schools (σ_8)



Access conditions (σ_{11})



Titles (σ_{14})



Invariant fragment:

$$
\begin{array}{l}
\phi(\langle \sigma_1, \ldots, \sigma_{14} \rangle) \stackrel{\rm def}{=} \\
\quad \begin{array}{ll}
let & x_1 = sel_1(\sigma_1) \\
 & x_2 = sel_2(\sigma_1) \\
in &
\end{array}
\end{array}
$$

(a) */*student's current degree course must exist */*

$$2^{\pi_1}(rng(x_1) \cup rng(x_2)) \subseteq dom(\sigma_2) \wedge$$

(b) */*student's current plan must belong to degree course */*

$$
\forall i \in rng(x_2) : \begin{array}{ll}
let & c = \pi_1(i) \qquad \wedge \\
 & p = \pi_2(i) \\
in & \pi_1(\sigma_2(c)) = p
\end{array}
$$

(c) */*student' past registrations obey to constraint (b) */*

$$
\forall i \in rng(x_2) : \begin{array}{ll}
let & c = \pi_1(i) \qquad \wedge \\
 & p = \pi_2(i) \\
 & a = \pi_4(i) \\
in & \forall t \in rng(a) : \pi_1(t) = c \wedge \pi_2(t) = p
\end{array}
$$

(d) */*students cannot do exams of courses they are not registered in */*

$$
\forall i \in rng(x_2) : \begin{array}{ll}
let & a = \pi_5(i) \qquad \wedge \\
 & b = \pi_6(i) \\
 & c = \pi_{12}(i) \\
in & \forall t \in (dom(a) \cup dom(b)) : \\
 & \pi_1(t) \in \{\pi_2(v) \mid v \in dom(c) \wedge \pi_1(v) = \pi_2(t)\}
\end{array}
$$

(e) */*student is registered in one degree course only in the back up structure */*

$$
\forall i \in rng(x_2) : \begin{array}{ll}
let & \sigma = \pi_8(i) \qquad \wedge \\
 & c = \pi_1(i) \\
in & \forall al \in dom(\sigma) : \pi_1(\sigma(al)) = c
\end{array}
$$

(f) */*courses in all academic years must belong to degree plan */*

$$
\forall c \in dom(\sigma_2) : \begin{array}{ll}
let & a = \sigma_2(c) \qquad \wedge \\
 & b = \pi_3(a) \\
 & p = \pi_1(a) \\
in & \forall t \in dom(b) : \pi_2(t) \in 2^{\pi_1}(\pi_2(\sigma_4(p)))
\end{array}
$$

(g) */*same as (f) concerning every student */*

$$
\forall i \in rng(x_2) : \begin{array}{ll}
let & a = \pi_{10}(i) \\
 & p = \pi_2(i) \\
in & \forall t \in dom(a) : \pi_2(t) \in 2^{\pi_1}(\pi_2(\sigma_4(p)))
\end{array}
$$

$(\ldots)$ */*............ etc .... etc .... */*

Adding new information to the formal model

The availability of an abstract model of the SA (="Academic Service") system made it possible to specify an important piece of local legislation, the $RIAPA$.

This is internal Norm RT-30/92 which contains a set of rules restricting student course registration, assessment and degree course follow-up.
Because it was issued in 1992, maintenance group was not confident enough to implement it in the system. So, SA officers have had to check its validity partly by hand.

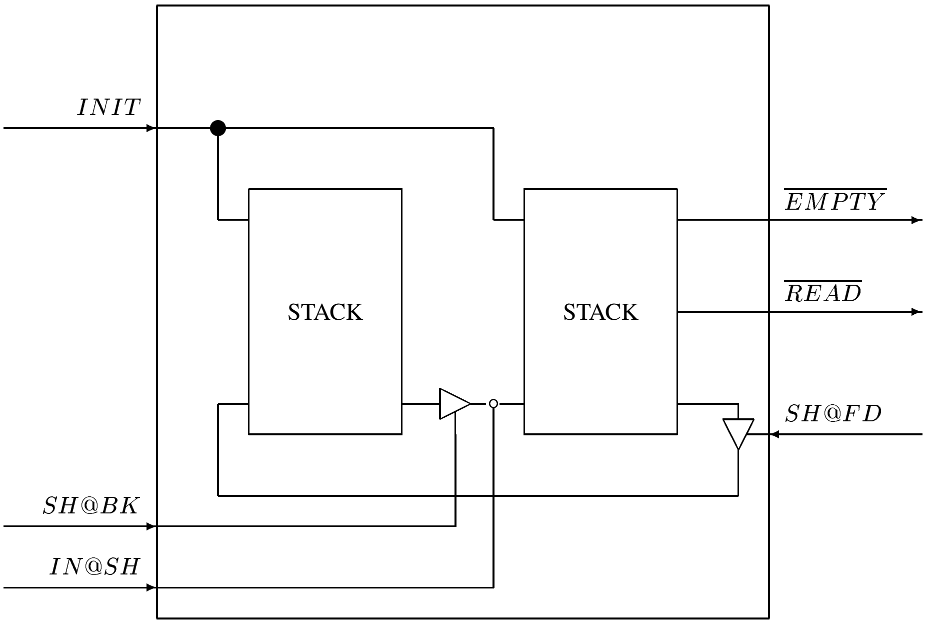
ϕ_{RIAPA} becomes a newly recorded local invariant:

$$
\begin{array}{l}
\phi_{RIAPA}(\sigma) \stackrel{\text{def}}{=} \\
\quad let \quad x_2 = \begin{pmatrix} a \\ b \end{pmatrix}_{a \in dom(\pi_1(\sigma)) \wedge \pi_1(\sigma)(a) = i_2(b)} \\
\quad in \quad \forall a \in dom(x_2) : \\
\qquad \left\{ \begin{array}{rcl}
2ndYear(a,\sigma) & \Rightarrow & \begin{array}{l}
(a) \hfill \text{/*2nd year students */} \\
(yearCo(da(a),a,\sigma) \leq 3 \wedge yearCo(de(a,al-1,\sigma),\sigma) \geq 1.5) \\
\hfill \vee \hfill \\
(credits(de(a,al-1,\sigma)) \geq \frac{3}{4} Pe(a) \wedge da(a) \leq 4)
\end{array} \\
lastYear(a,\sigma) & \Rightarrow & \begin{array}{l}
(b) \hfill \text{/*last year students */} \\
yearCo(da(a),a,\sigma) \leq 4
\end{array} \\
otherwise(a,\sigma) & \Rightarrow & \begin{array}{l}
(c) \hfill \text{/*all other students */} \\
(yearCo(da(a),a,\sigma) \leq 3) \\
\hfill \vee \hfill \\
(credits(de(a,al-1),\sigma) \geq \frac{3}{4} Pe(a) \wedge da(a) \leq 4)
\end{array}
\end{array} \right.
\end{array}
$$

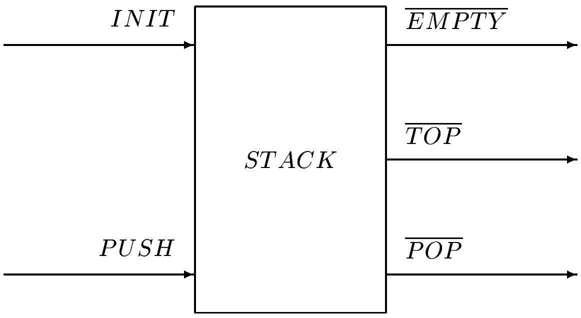
(The definitions of auxiliary functions above are complex and have to be omitted for brevity.)

Formal software component (modular) analysis

We would like to be able to organize the newly reversed system as a network of formal software components, that is, in the same way as “toy” example



which aggregates two copies of software component



Roughly speaking, we have 14 candidate software components in SA. How do we go about synthesizing them?

- We need to assign concrete programs to abstract structures.

- One off-product of inverse calculation is a (non-injective!) **aggregation function** mapping each of the original 55 tables to some of the Set 14 expressions, *cf.* the following fragment of it:

Final	**Set of aggregated table numbers**
A	$\{1, 2, 3, 38, 4, 5, 6, 10, 20, 24, 27, 32, 33, 34, 35, 42, 48, 50, 53, 55\}$
C	$\{12, 11, 13, 21, 22, 23, 28, 40\}$
D	$\{16, 52, 54, 43\}$
P	$\{47, 46, 49\}$
O	$\{39, 45\}$
DP	$\{14\}$
DC	$\{18\}$
E	$\{19\}$
ES	$\{25\}$
ET	$\{26\}$
AL-EP-CH	$\{29\}$
H	$\{31\}$
T	$\{44\}$

- By applying this aggregation function to the column headers of the "who needs what" matrix, we get a much "coarser" matrix,

Prog./Tab.	**A**	**O**	**A**	**A**	$\cdots$
act_alunos.pc	A	L	LR		
act_bolseiro.pc	L			LA	
act_dirigente.pc	L			LA	
act_militar.pc	L			LA	
act_regime_estag.pc				A	L
$\cdots$					

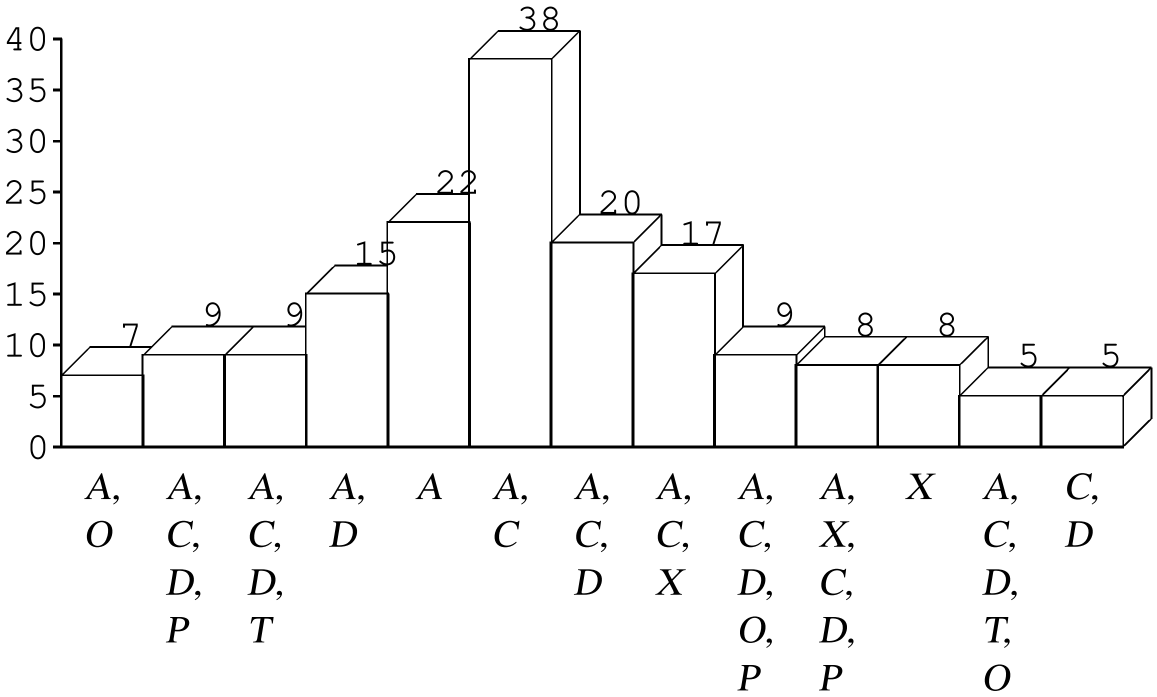
in which two kinds of operator can be found:

- Relative to one structure only, and thus calling some encapsulable "method" at that abstraction level.
- Common to two or more structures, and thus external to such structures (synchronizations).

For instance, while `act_bolseiro.pc`, `act_dirigente.pc` and `act_militar.pc` happen to be "private" to the A component (students) — *cf.* $IN@SH$ in $FOLDER$ realized solely in terms of $PUSH$ of $STACK$ — `act_alunos.pc` and `act_regime_estag.pc` are like $SH@BK$ in $FOLDER$ — they provide a kind of “rendez vous” between two components, A and O (synchronization).

Clearly, the largest components will be A (students), C (degree courses) and D (courses) — as one would expect.

The following distribution of transactions per abstract structure is but a confirmation of that:



(Histogram shows only component synchronizations which are made by at least 5 different transactions; X represents the aggregation of a few less relevant and rather small tables.)

Getting Ready for Re-engineering

We have seen how SETS-supported reasoning enabled us to have a clear picture of the system and to obtain guidelines for transaction “encapsulation”.

Concerning future re-engineering of the system, further analysis of the obtained formal model points to several directions.

Improving the data model

Consider the following fragment of the A (students) formal model:

$((AL \times D) \hookrightarrow Inf24) \times ((AL \times D) \hookrightarrow Inf32) \times ((AL \times D) \hookrightarrow Inf33)$

This is but an example among many others in which the Set-pattern

$$(A \hookrightarrow B \times C) \;\;\times\;\; (A \hookrightarrow B \times D)$$

is spot ($B = 1$ in the above example). As we know, this pattern may "resolve" to

$$A \hookrightarrow B \times (C + D)$$

or to

$$A \hookrightarrow B^2 \times C \times D$$

should the relevant concrete invariants hold.

SETS-pattern inspection helps in re-evaluating former informal analysis.

Temporalization

This is another example of model assessment via SETS-pattern inspection: we see too many terms of pattern

$$(AL \times A) \hookrightarrow B$$

where AL (= "ano lectivo" = academic year) is a temporal dimension of the system.

Formal work on a SETS-based approach to temporal information systems ([Gro96, Bar96]) suggests that the adoption of an explicit temporal model might improve the overall system in a significant way.

Object Orientation

An interesting re-engineering direction will be to move into object-oriented design. How can we "re-use", in this way, what we achieved so far?

SETS will help us once again.

A very abstract model of an OO-database would be

$$K \hookrightarrow Structure$$

In which $Structure$ must be able to accommodate a given class system.

Let $\bullet_{a_A}$ denote the fact that object class name a has attribute A (which may be an aggregation of several attributes, *e.g.* $A = A_1 \times \ldots \times A_n$). If a is the only class in the system, then we will have

$$K \hookrightarrow A$$

in Set. Now add subclass $\bullet_{b_B}$ of a to the class system:

$$\begin{tikzcd}
\bullet\, a_A \arrow[d, no head] \\
\bullet\, b_B
\end{tikzcd}$$

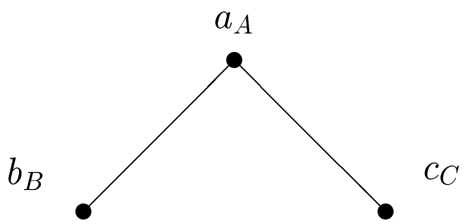
For this situation, we want to have

$$K \hookrightarrow (A + A \times B)$$

in Set, that is,

$$Structure \;=\; A \times (1 + B)$$

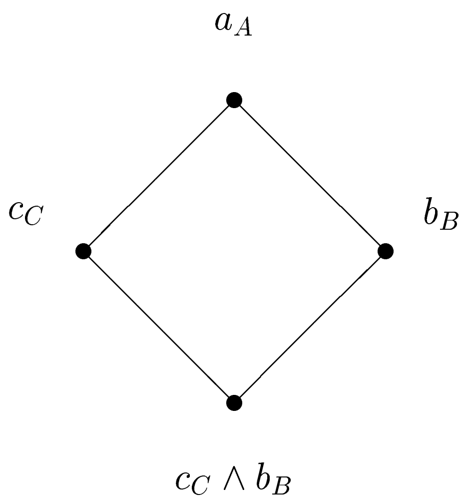
By adding another subclass $\bullet_{c_C}$ of a to the class hierarchy,



we will want

$$\begin{aligned} Structure &= A + A \times B + A \times C \\ &= A \times (1 + B + C) \end{aligned}$$

Furthermore, we don't want to rule out multiple inheritance,



Thus we want the above class system to be translated into Set as follows:

$$\begin{aligned} Structure &= A \times (1 + B + C + B \times C) \\ &= A \times (1 + B) \times (1 + C) \end{aligned}$$

In SA' we find many occurrences of the $A \times \Pi_i^n (1 + B_i)$ pattern, *e.g.*

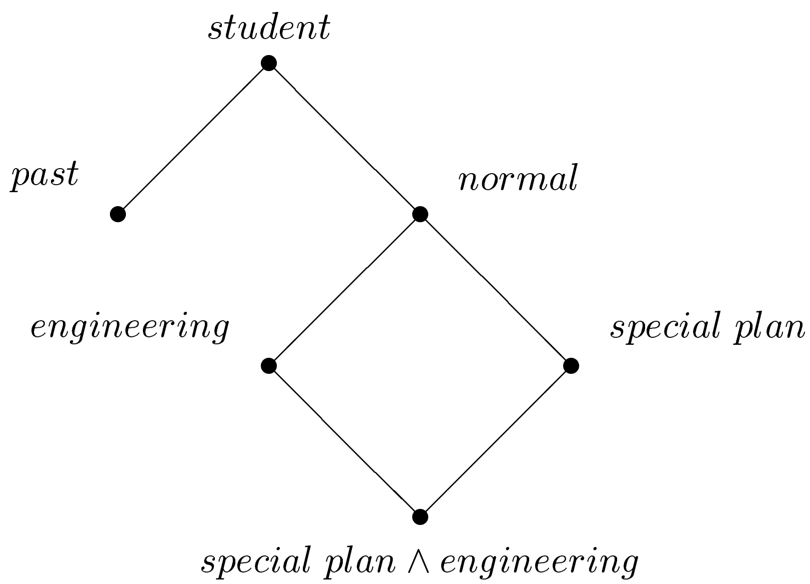
$\ldots \times (1 + Inf27) \ \times$ /**engineering student* */

$$(1 + C \times P \times Inf48) \quad \times \qquad \text{/*}\textit{special plan}\text{ */}$$

$$\ldots$$

in A (students).

Put into its hierarchical contex, this expression will lead to the following starting point for the $student$ class and its descendant classes:



etc., *etc.*

Final Remarks

- Most work in the literature refers to formal reverse engineering of algorithmic code, in particular resorting to wp-calculi, see *e.g.* [CG93, GC96].
- Other applications of formal methods in the OO-analysis of existing software (*e.g.* [CA93]) include the use of PVS (Prototype Verification Systems) formal tools but do not have a calculation flavour.
- The SETS calculus seems to provide an overall "do-it-by-calculation" discipline for large system reverse-engineering which can be eas-

ily combined with wp-calculi (*e.g.* the Oxford Refinement Calculus) towards their OO re-engineering.

- On the whole, the presented case study provides evidence of how informal and formal methods blend well in rescuing legacy information system from (often a too early…) “death”.

References

[Bar96] L. S. Barbosa. The monadic structure of time. Technical report, Departamento de Informática, Universidade do Minho, 1996. (Submitted for publication).

[CA93] B.H.C. Cheng and Brent Auernheimer. Applying formal methods and object-oriented analysis to existing flight software. Technical report, DCS, Michigan State University and DCS, California State University, 1993.

[CG93] B.H.C. Cheng and G.C. Gannod. A two-phase approach to reverse engineering using formal methods. In *Formal Methods in Programming and Their Applications*. Springer-Verlag, 1993. Lecture Notes in Computer Science.

[ELS95] A. Esteves, J. Leal, and J.M. Silva. Documentação dos serviços académicos da U.M. Technical Report LESI-Op-IIIa, Universidade do Minho, Fev. 1995. (In Portuguese).

[GC96] G.C. Gannod and B.H.C. Cheng. Using informal and formal techniques for the reverse engineering of c programs. Technical report, DCS, Michigan State University, 1996.

[Gro96] INESC 2361 Group. Temporalização da NBDC. Technical report, © SONAE Company, Maia, May 1996. Consultancy report (in Portuguese).

[Vel95] M.J.P. Veloso. Engenharia reversa em larga escala de sistema de informação — caso concreto: O sistema de informação dos S.A. Technical Report LESI-Op-IIIb, Universidade do Minho, Fev. 1995. (In Portuguese).