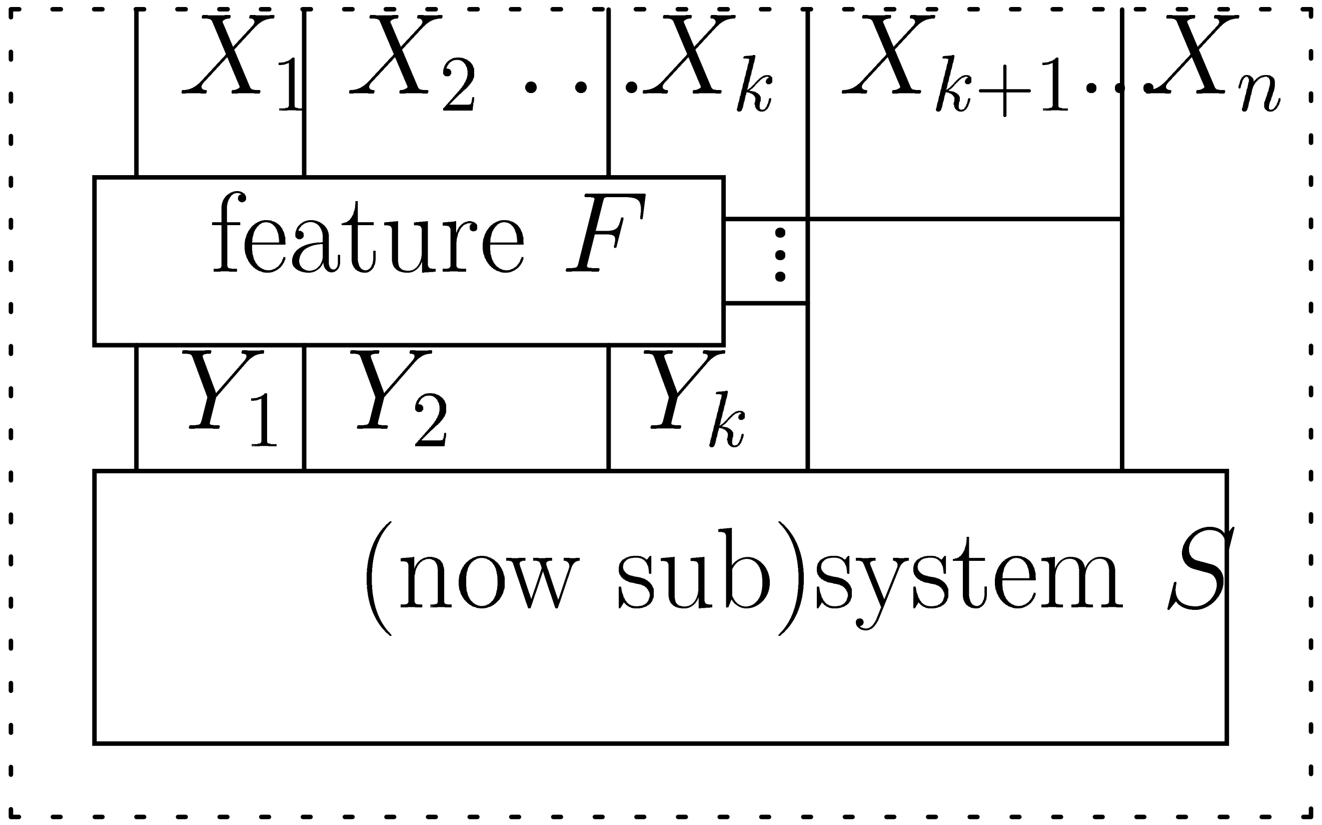
1. Features as substitutions
2. Features of access control systems
3. The social factor in FI, or: who Muffy and me aren’t dinosaurs after all!

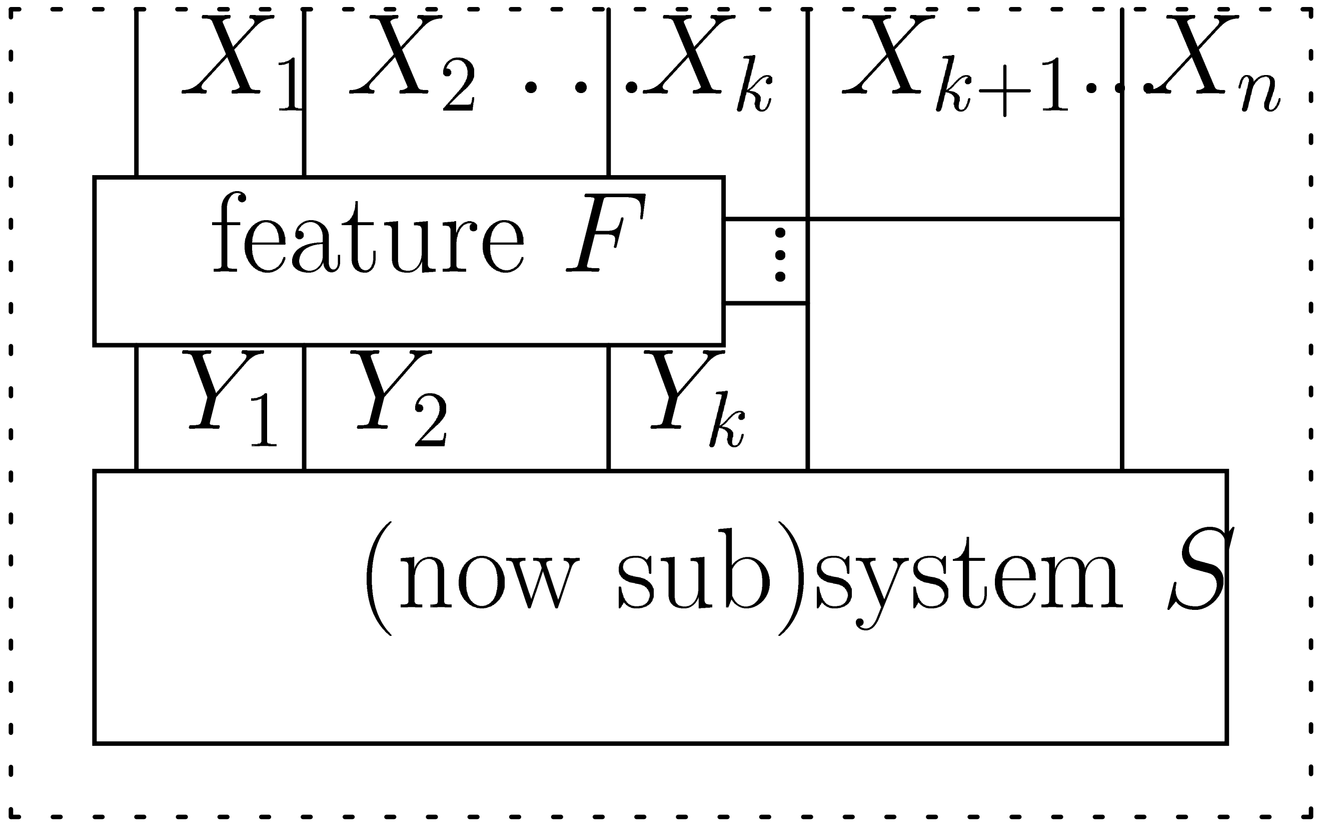
Representing feature integration by substitution

Dimitar Guelev / Mark Dermot Ryan / Pierre Yves Schobbens

Example: Hardware Features



Example: Hardware Features



Feature-ready Specifications

We assume that systems consist of an *immutable* base and *mutable parts*. Mutable parts of S are assumed to be described by *subformulas* of $[S]$. That is why we can write

$$[S] \doteq \underbrace{[\lambda\overline{x_1}.\varphi_1/P_1, \dots, \lambda\overline{x_n}.\varphi_n/P_n]}_{S_1}\, B$$

where: $P_1, \dots, P_n$ are the *parts' names*
$\lambda x_{i,1} \dots x_{i,n_i}.\varphi_i$ describes part i
B describes the immutable base.

Mutability of parts can be further stratified:

$$[S] \doteq S_n \dots S_{k+1} S_k \dots S_1 B$$

- Instantiation of this framework using a simple while-based programming language, and the logic DC (duration calculus);
- Case study: Samborski stack model;
- Compositional reasoning: establish properties of a feature, which will hold of any system which integrates it in a way satisfying certain conditions.

Features of Access Control Systems

Mark Dermot Ryan / Nan Zhang /
Dimitar Guelev / Pierre Yves Schobbens

Access Control Systems

- File access control in OSs, firewalls, spam filters;
- On-line communities, e.g. Yahoo groups, OSS management, auction sites, p2p;
- Databases, e.g. student information system, health care records

often need to be reconfigured, because the requirements of the organisation changes when new people arrive or are deployed in different ways, or when new services are added, or when new regulations are introduced.

Superuser makes the changes:

- leads to a bottleneck
- security vulnerability

Permissions about permissions

Permissions are themselves data which are subject to access permissions

- model delegated or devolved authority (*subsidiarity*)
- avoid superuser bottleneck

Indirect paths, or implicit permissions: an agent might not have a certain permission, but may have sufficient permissions about permissions to execute a sequence of actions which gives him the desired permission.

Example: student information system

- Students can *read* only their own marks, and *write* none.
- Professors can *read* all marks, and *write* those about their lecture courses.

Features brought about by new regulations

F1 A student may delegate readability of his/her marks to another student.

F2 A professor may appoint student assistants, and delegate writability of marks to them.

Features and feature interferences

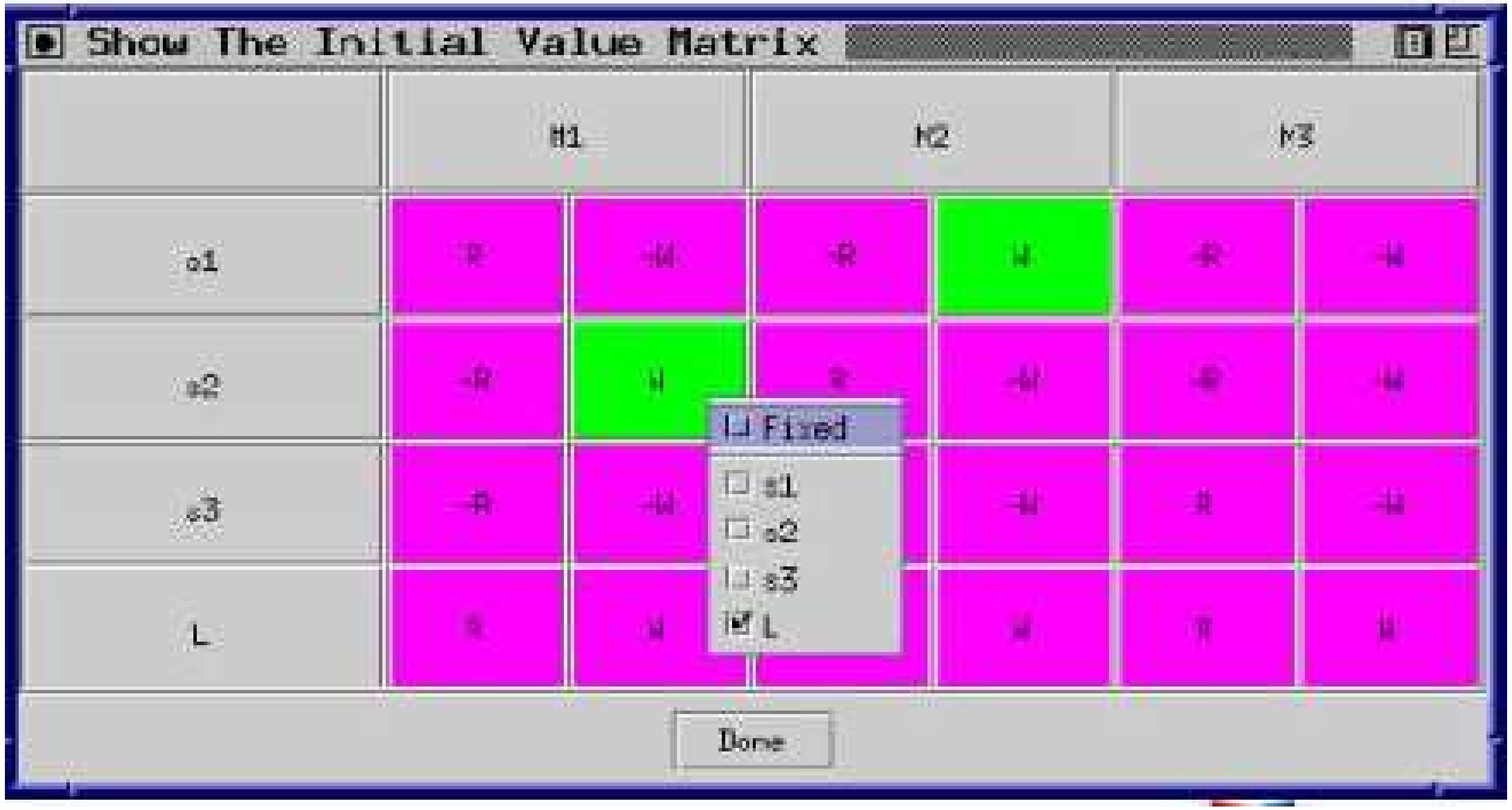
F1 A student may delegate his permissions to another student.

F2 A professor may delegate writability of marks to a student.

Feature interferences

- F1 + F1
- F2 + F2
- F1 + F2

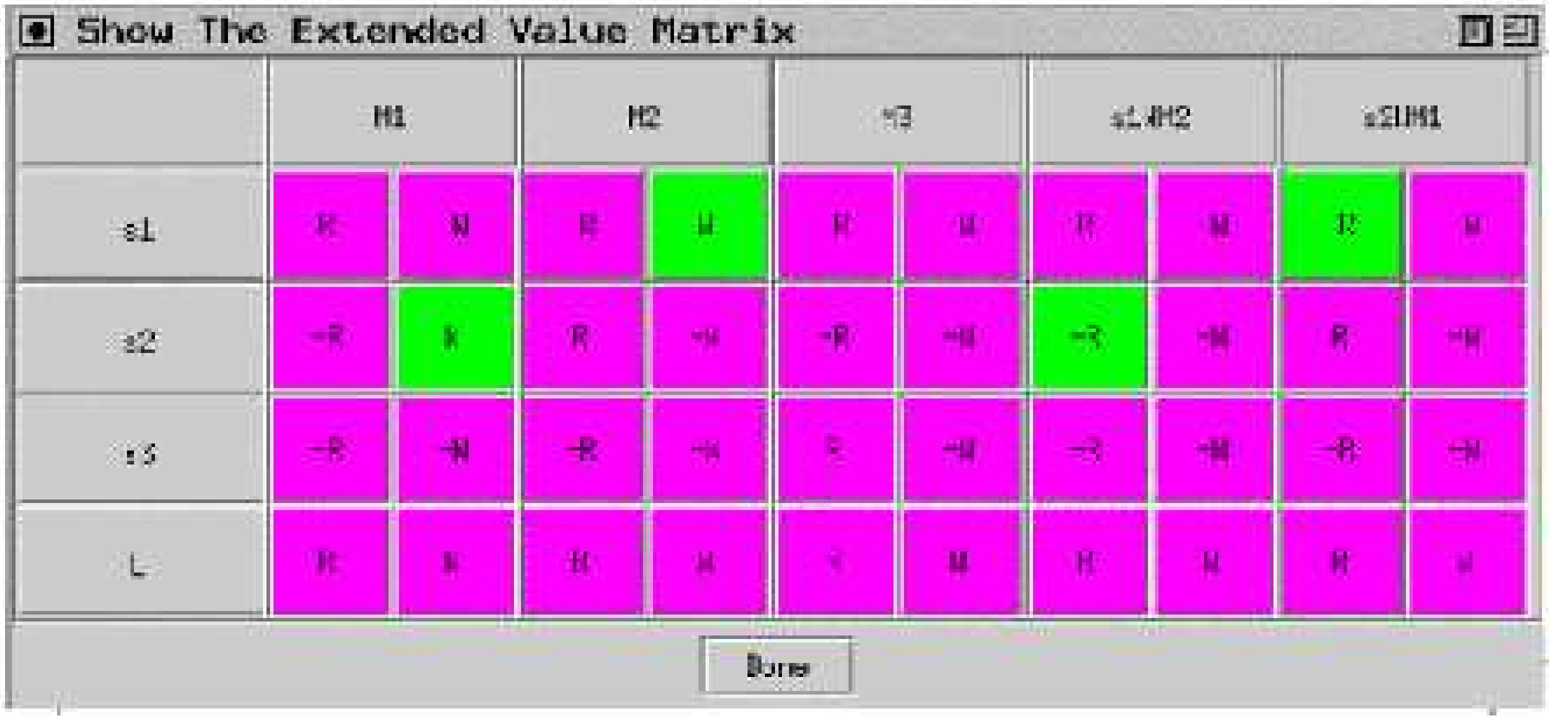
Prototype in Java



	M1		M2		M3	
s1	-R	-W	-R	W	-R	-W
s2	-R	W	R	-W	-R	W
s3	-R	-W	[illegible]	-W	R	-W
L	R	W	[illegible]	W	R	W

- [ ] s1
- [ ] s2
- [ ] s3
- [✓] L

Extended matrix


Show The Extended Value Matrix

	M1		M2		s3		s1∧M2		s2∪M1	
s1	R	W	R	W	R	W	R	W	R	W
s2	-R	W	R	-W	-R	-W	-R	-W	R	-W
s3	-R	-W	-R	-W	R	-W	-R	-W	-R	-W
L	R	W	R	W	R	W	R	W	R	W

Model Checking

- From such an access control matrix, can obtain a transition system: states are values of the matrix, and each state thus determines which other states it can transition to.
- The prototype program outputs SMV code, and specifications such as
  - $\mathrm{EF}(\mathrm{s1Wm2} \wedge \mathrm{s2Wm1})$
  - $\mathrm{E}[p \ \mathrm{U}\ q]$, $\mathrm{AG}(p \rightarrow \mathrm{EF}q)$, etc

  can be verified.

The logic of access control

- A logic which axiomatises read- and write- permissions;
- provides two kinds of models (abstract and concrete)
- we aim to show correspondences between the models and completeness of the axioms (not all yet done)

Syntax

Logic. $\phi ::= v \mid \phi \Rightarrow \phi \mid \bot \mid \mathtt{R}_A\phi \mid \mathtt{W}_A\phi$

where $A \in \Sigma$.

Concrete semantics.

States Q $\quad s : V \cup P \to \mathrm{Bool}$

Programs $\quad S \ ::= \ vp := e \mid \text{if } e \text{ then } S_1 \text{ else } S_2 \mid S_1; S_2 \mid \ \text{skip} \mid \text{fail}$

Logic semantics

- $\mathcal{C}, s \Vdash \mathtt{R}_A\phi$ iff $\exists S(\forall t([\![S]\!]_A t{\downarrow} \to ([\![S]\!]_A t)(p) = t(\phi)) \wedge [\![S]\!]_A s{\downarrow})$
- $\mathcal{C}, s \Vdash \mathtt{W}_A\phi$ iff $\forall b \in \mathrm{Bool}\exists S$ for A $[\![S]\!]s(\phi) = b$

Conclusions

- Permissions about permissions, root bottleneck avoidance
- Model checking
- Logic with concrete and abstract semantics, and correspondences.

Future work

- More on modelling existing systems:
  - implemented ACSs
  - models of ACS such as OASIS, RBAC
- Continue proofs of logic system!

FIW'03: "social factor"

- Zave: features are purposes, not mechanisms.
- Boxton: we should design cameras/cars/browsers/. . . not computers.
- Turner/Gray: busy is a person state, not a device state – and the answer depends on who's asking.
- Bredereke: Many FIs are abstractability failures.
- . . . etc . . .

No dinosaurs

$$\underbrace{S}_{\text{base system}} \quad + \quad \underbrace{f}_{\text{mechanism}} \quad + \quad \models \quad \underbrace{\phi}_{\text{purpose}}$$

Feature interference occurs if the mechanism(s) don't fulfil the purpose(s).