

Refinement in Practise

© Michael Butler

University of Southampton

March 29, 2016

Abstraction

- ▶ Abstraction can be viewed as a process of **simplifying our understanding** of a system.
- ▶ The simplification should
 - ▶ focus on the **intended purpose** of the system
 - ▶ **ignore details of how** that purpose is achieved.
- ▶ The modeller needs to make judgements about what they believe to be the **key features** of the system.

Abstraction

- ▶ Abstraction can be viewed as a process of **simplifying our understanding** of a system.
- ▶ The simplification should
 - ▶ focus on the **intended purpose** of the system
 - ▶ **ignore details of how** that purpose is achieved.
- ▶ The modeller needs to make judgements about what they believe to be the **key features** of the system.
- ▶ If the purpose is to **provide some service**, then
 - ▶ model what a system does from the perspective of the service users
 - ▶ 'users' might be computing agents as well as humans.

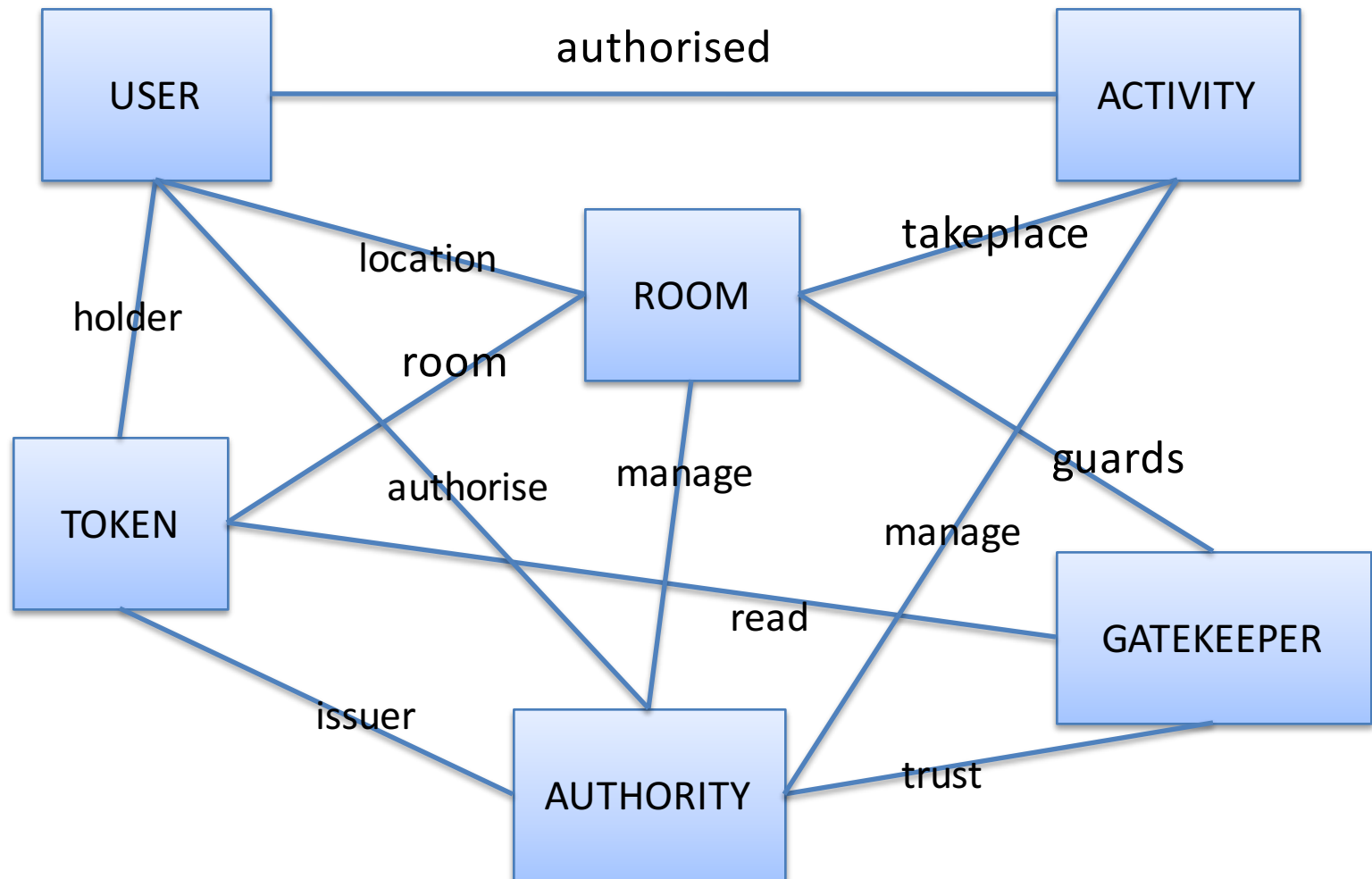
Abstraction

- ▶ Abstraction can be viewed as a process of **simplifying our understanding** of a system.
- ▶ The simplification should
 - ▶ focus on the **intended purpose** of the system
 - ▶ **ignore details of how** that purpose is achieved.
- ▶ The modeller needs to make judgements about what they believe to be the **key features** of the system.
- ▶ If the purpose is to **provide some service**, then
 - ▶ model what a system does from the perspective of the service users
 - ▶ 'users' might be computing agents as well as humans.
- ▶ If the purpose is to **control, monitor or protect** some phenomenon, then
 - ▶ the abstraction should focus on those phenomenon
 - ▶ in **what way** should they be controlled or protected?

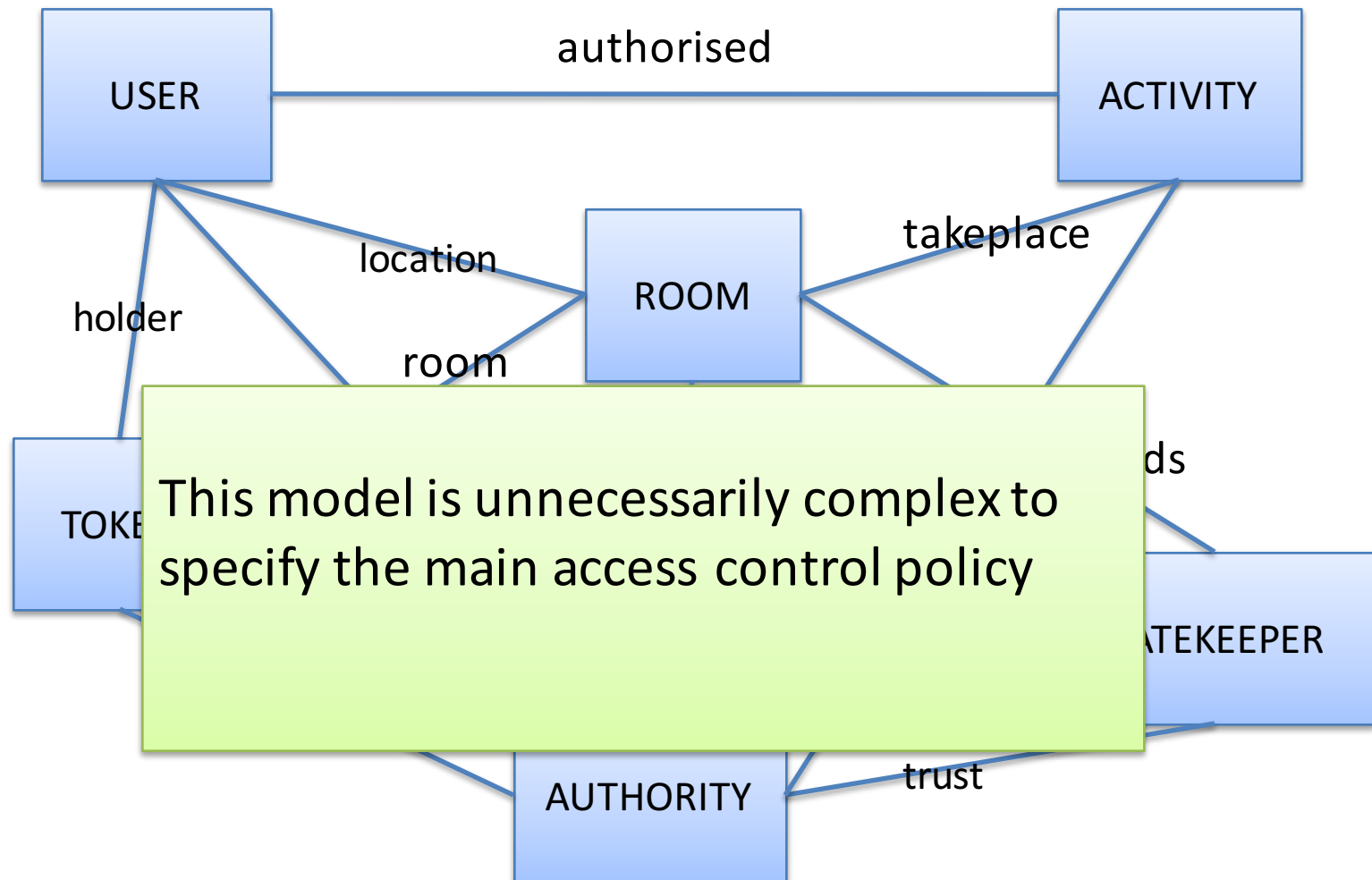
Access control system

- Users are authorised to engage in activities
- User authorisation may be added or revoked
- Activities take place in rooms
- Users gain access to a room using a one-time token provided they have authority to engage in the room activities
- Tokens are issued by a central authority
- Tokens are time stamped
- A room gateway allows access with a token provided the token is valid

Class diagram



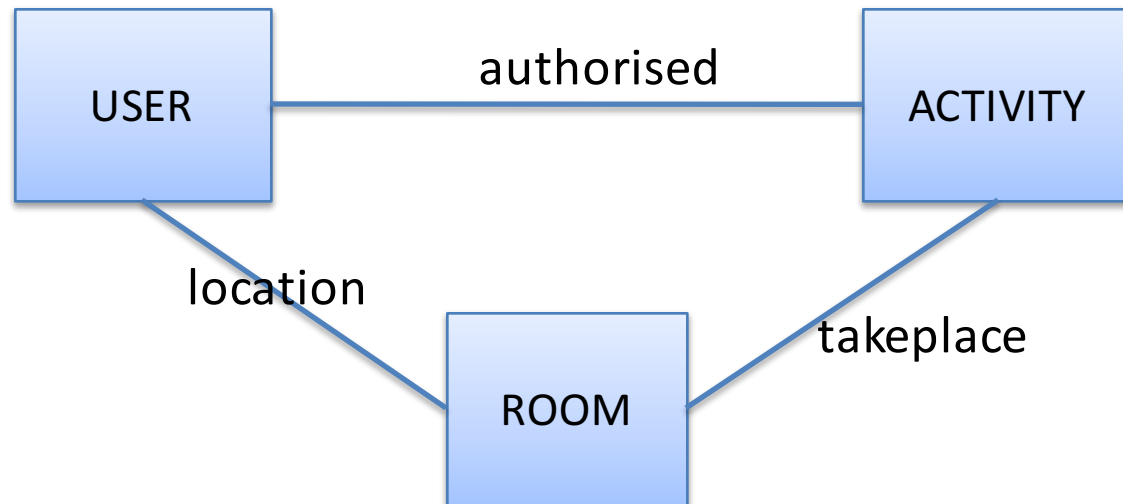
Class diagram



Extracting the essence

- **Access Control Policy:** *Users may be in a room only if they are authorised to engage in all activities that may take place in that room*
- To express this we only require **Users**, **Rooms**, **Activities** and **relationships** between them
- **Abstraction:** focus on key entities in the problem domain

Diagrammatic representation of an abstract model



Variables and invariants of Event-B model

Variables of Event-B model

@inv1	authorised \in User \leftrightarrow Activity	// relation
@inv2	takeplace \in Room \leftrightarrow Activity	// relation
@inv3	location \in User \mapsto Room	// partial function

Access control *invariant*:

if user u is in room r ,

then u must be authorised to engaged in all activities that can take place in \underline{r}

@inv4 $\forall u, r. u \in \text{dom}(\text{location}) \wedge \text{location}(u) = r \Rightarrow$
 $\text{takeplace}[r] \subseteq \text{authorised}[u]$

Refinement

- ▶ **Refinement** is a process of enriching or modifying a model in order to
 - ▶ **augment** the functionality being modelled, or
 - ▶ **explain how** some purpose is achieved

Refinement

- ▶ **Refinement** is a process of enriching or modifying a model in order to
 - ▶ **augment** the functionality being modelled, or
 - ▶ **explain how** some purpose is achieved
- ▶ In a **refinement step** we refine one model $M1$ to another model $M2$:
 - ▶ $M2$ is a refinement of $M1$
 - ▶ $M1$ is an abstraction of $M2$

Refinement

- ▶ **Refinement** is a process of enriching or modifying a model in order to
 - ▶ **augment** the functionality being modelled, or
 - ▶ **explain how** some purpose is achieved
- ▶ In a **refinement step** we refine one model $M1$ to another model $M2$:
 - ▶ $M2$ is a refinement of $M1$
 - ▶ $M1$ is an abstraction of $M2$
- ▶ We can perform a **series** of refinement steps to produce a series of models $M1, M2, M3, \dots$
- ▶ Facilitates abstraction: we can **postpone** treatment of some system features to later refinement steps

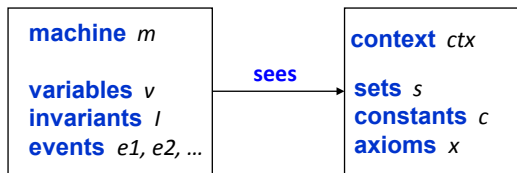
Refinement

- ▶ **Refinement** is a process of enriching or modifying a model in order to
 - ▶ **augment** the functionality being modelled, or
 - ▶ **explain how** some purpose is achieved
- ▶ In a **refinement step** we refine one model $M1$ to another model $M2$:
 - ▶ $M2$ is a refinement of $M1$
 - ▶ $M1$ is an abstraction of $M2$
- ▶ We can perform a **series** of refinement steps to produce a series of models $M1, M2, M3, \dots$
- ▶ Facilitates abstraction: we can **postpone** treatment of some system features to later refinement steps
- ▶ Event-B provides a notion of **consistency** of a refinement:
 - ▶ We use proof to **verify** the consistency of a refinement step
 - ▶ Failing proof can help us **identify inconsistencies** in a refinement step

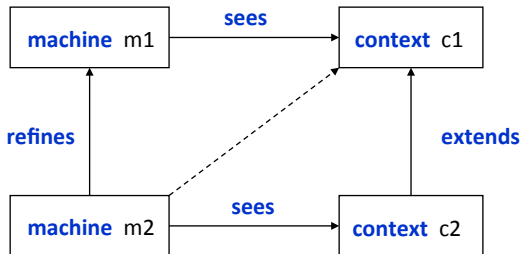
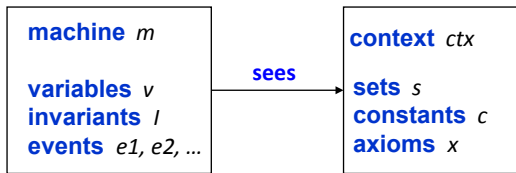
Refinement

- ▶ **Refinement** is a process of enriching or modifying a model in order to
 - ▶ **augment** the functionality being modelled, or
 - ▶ **explain how** some purpose is achieved
- ▶ In a **refinement step** we refine one model $M1$ to another model $M2$:
 - ▶ $M2$ is a refinement of $M1$
 - ▶ $M1$ is an abstraction of $M2$
- ▶ We can perform a **series** of refinement steps to produce a series of models $M1, M2, M3, \dots$
- ▶ Facilitates abstraction: we can **postpone** treatment of some system features to later refinement steps
- ▶ Event-B provides a notion of **consistency** of a refinement:
 - ▶ We use proof to **verify** the consistency of a refinement step
 - ▶ Failing proof can help us **identify inconsistencies** in a refinement step
- ▶ Abstraction and refinement together should allow us to **manage system complexity** in the design process

Modelling Components and Refinement



Modelling Components and Refinement



Extension Refinement in Event-B

A **refined** machine has the following form:

```
machine   M2  
refines  M1  
variables ...  
invariants ...  
events...
```

Extension refinement can be used to extend or add new features to a model.

- ▶ Add variables and invariants
- ▶ Extend existing events to act on additional variables
- ▶ Add new events to act on additional variables

All events must maintain the new invariants.

Event Extension

Event $E1$ of $M1$ may be **extended** by $E2$ in $M2$:

$E2$ **extends** $E1 \hat{=}$
 any $\langle \text{additional parameters} \rangle$ **where**
 $\langle \text{additional guards} \rangle$
 then
 $\langle \text{additional actions} \rangle$
 end

Event Extension

Event $E1$ of $M1$ may be **extended** by $E2$ in $M2$:

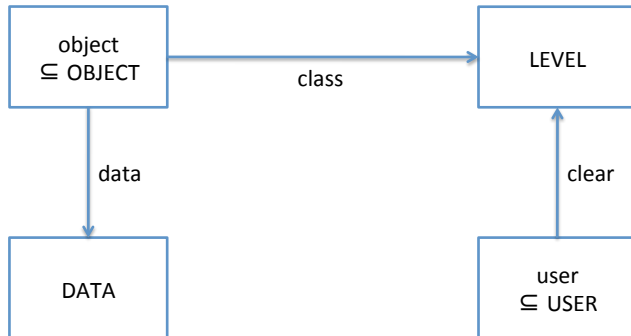
$E2$ **extends** $E1 \hat{=}$
 any $\langle \text{additional parameters} \rangle$ **where**
 $\langle \text{additional guards} \rangle$
 then
 $\langle \text{additional actions} \rangle$
 end

Extending an event means

- ▶ adding parameters
- ▶ adding guards
- ▶ adding actions

- ▶ Extension example: add ownership to secure database

Class diagram for secure database



Types and variables for Secure DB

context *c1*

sets *OBJECT DATA USER*

constants *LEVEL*

axioms *LEVEL = 1..10*

machine *SecureDB1*

sees *c1*

variables *object, user, data, class, clear*

invariants

object \subseteq *OBJECT*

user \subseteq *USER*

data \in *object* \rightarrow *DATA*

class \in *object* \rightarrow *LEVEL*

clear \in *user* \rightarrow *LEVEL*

Adding object ownership

Extend the database specification so that each object has an owner.

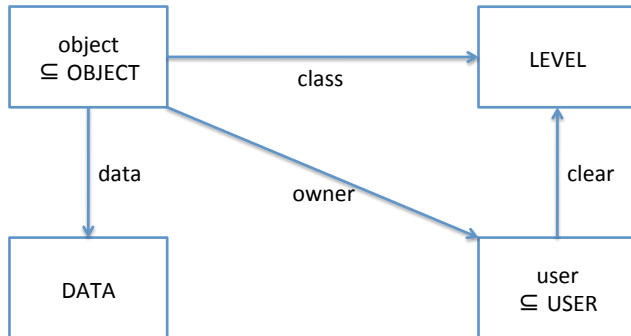
The clearance associated with that owner must be at least as high as the classification of the object.

Only the owner of an object is allowed to delete it.

What additional variables are required?

What events are affected?

Class diagram with ownership



Refinement

machine *SecureDB2*

refines *SecureDB1*

variables *object, user, data, class, clear, owner*

invariants

$$owner \in object \rightarrow user$$

Note we must list **all** the variables: those from M1 that we wish to retain as well as new ones

Here *owner* is a new variable.

We do not repeat invariants of M1 in M2

Adding users

```
AddUser  $\hat{=}$   
  any  $u, c$  where  
     $u \in USER$   
     $u \notin user$   
     $c \in LEVEL$   
  then  
     $user := user \cup \{u\}$   
     $clear(u) := c$   
  end
```

Do we need to modify this?

Adding objects

```
AddObject  $\hat{=}$   
  any  $o, d, c$  where  
     $o \in OBJECT$   
     $o \notin object$   
     $d \in DATA$   
     $c \in LEVEL$   
  then  
     $object := object \cup \{o\}$   
     $data(o) := d$   
     $class(o) := c$   
  end
```

Do we need to modify this?

Event Extension

AddObject **extends** *AddObject* $\hat{=}$
 any u **where**
 $u \in \text{user}$
 $\text{clear}(u) \geq \text{class}(o)$
 then
 $\text{owner}(o) := u$
 end

This is equivalent to

```
AddObject refines AddObject  $\hat{=}$   
  any  $o, d, c, u$  where  
     $o \in OBJECT$   
     $o \notin object$   
     $d \in DATA$   
     $c \in LEVEL$   
     $u \in user$   
     $clear(u) \geq class(o)$   
  then  
     $object := object \cup \{o\}$   
     $data(o) := d$   
     $class(o) := c$   
     $owner(o) := u$   
  end
```

Other events to consider

- ▶ *Read*
- ▶ *Write*
- ▶ *ChangeClass*
- ▶ *ChangeClear*
- ▶ *RemoveUser, RemoveObject*

Do we need new events?

Forms of Event-B Refinement

1. Extension:

- ▶ Add variables and invariants
- ▶ Extend existing events to act on additional variables
- ▶ Add new events to act on additional variables

Forms of Event-B Refinement

1. Extension:

- ▶ Add variables and invariants
- ▶ Extend existing events to act on additional variables
- ▶ Add new events to act on additional variables

2. Extension with Guard Modification:

- ▶ Similar to model extension, except that we modify guards of existing events

Forms of Event-B Refinement

1. Extension:

- ▶ Add variables and invariants
- ▶ Extend existing events to act on additional variables
- ▶ Add new events to act on additional variables

2. Extension with Guard Modification:

- ▶ Similar to model extension, except that we modify guards of existing events

3. Variable Replacement / Data Reification:

- ▶ Replace some variables with other variables, i.e., replace abstract variables with concrete variables
- ▶ Modify existing events, add new events

Forms of Event-B Refinement

1. Extension:
 - ▶ Add variables and invariants
 - ▶ Extend existing events to act on additional variables
 - ▶ Add new events to act on additional variables
2. Extension with Guard Modification:
 - ▶ Similar to model extension, except that we modify guards of existing events
3. Variable Replacement / Data Reification:
 - ▶ Replace some variables with other variables, i.e., replace abstract variables with concrete variables
 - ▶ Modify existing events, add new events
4. Variable Removal:
 - ▶ Remove variables that have become redundant through earlier introduction of other variables.

Forms of Event-B Refinement

1. Extension:

- ▶ Add variables and invariants
- ▶ Extend existing events to act on additional variables
- ▶ Add new events to act on additional variables

2. Extension with Guard Modification:

- ▶ Similar to model extension, except that we modify guards of existing events

3. Variable Replacement / Data Reification:

- ▶ Replace some variables with other variables, i.e., replace abstract variables with concrete variables
- ▶ Modify existing events, add new events

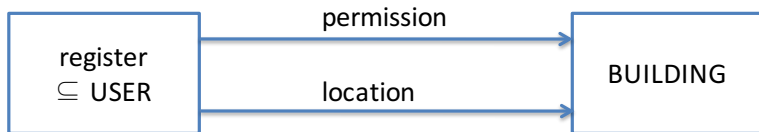
4. Variable Removal:

- ▶ Remove variables that have become redundant through earlier introduction of other variables.

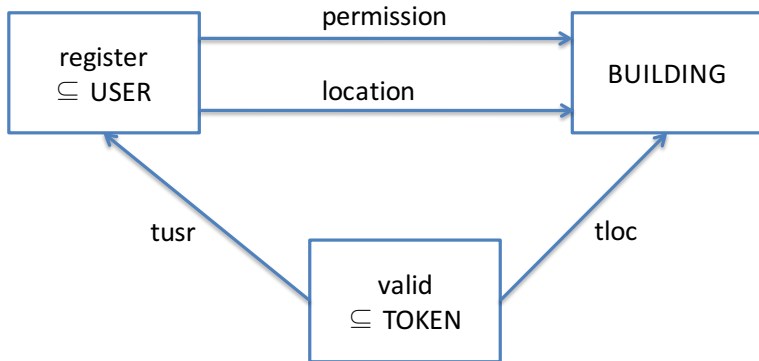
- ▶ Verification of 2, 3 and 4 requires gluing invariants that link abstract and concrete variables.

- ▶ Extension example: add ownership to secure database
- ▶ Extension with Guard Modification example: add tokens to buildings access system
- ▶ Variable replace example: simple data sampling system

Abstract model of building access



Refine this by introducing a token mechanism



Refinement of access control with tokens

$EnterBuilding \hat{=} \text{any } u, b \text{ where}$
 $u \notin dom(location)$
 $u \mapsto b \in permission$
then
 $location(u) := b$
end

$RefinedEnterBuilding \hat{=} \text{any } u, b, t \text{ where}$
 $u \notin dom(location)$
 $t \in valid$
 $tusr(t) = u$
 $tloc(t) = b$
then
 $location(u) := b$
end

GRD Proof Obligations

We need to prove that the guard of a refined event is not weaker than the guard of the abstract event.

E.g., the refined enter event should not weaken the conditions under which a user may enter a room.

GRD Proof Obligations

We need to prove that the guard of a refined event is not weaker than the guard of the abstract event.

E.g., the refined enter event should not weaken the conditions under which a user may enter a room.

GRD Proof obligation:

Assume: $\text{guard}(\text{RefinedEnter}) + \text{invariants}$

Prove: $\text{guard}(\text{AbstractEnter})$

GRD Proof Obligations

We need to prove that the guard of a refined event is not weaker than the guard of the abstract event.

E.g., the refined enter event should not weaken the conditions under which a user may enter a room.

GRD Proof obligation:

Assume: $\text{guard}(\text{RefinedEnter}) + \text{invariants}$

Prove: $\text{guard}(\text{AbstractEnter})$

For the access control refinement, we need this invariant:

$$\forall t \cdot t \in \text{valid} \implies \text{tsur}(t) \mapsto \text{tloc}(t) \in \text{permission}$$

Simple data sampling system

machine *MaxSet1*

variables *samples*

invariants *samples* $\subseteq \mathbb{N}$

initialisation *samples* := {0}

events

Add $\hat{=}$ **any** *x* **where**
 $x \in \mathbb{N}$
 then
 samples := *samples* \cup {*x*}
 end

GetMax $\hat{=}$ **any** *result* **where**
 $result = \max(samples)$
 end

Refine to a more optimal design

machine *MaxSet2*
refines *MaxSet1*
variables *m* we only need to store the maximum so far
invariants $m \in \mathbb{N}$
initialisation $m := 0$
events

Add $\hat{=}$ **any** *x* **where**
 $x \in \mathbb{N}$
 then
 $m := \max(\{m, x\})$
 end

GetMax $\hat{=}$ **any** *result* **where**
 $result = m$
 end

Gluing invariant

What is the relationship between m and *samples*?

Gluing invariant

What is the relationship between m and $samples$?

```
machine   MaxSet2  
refines  MaxSet1  
variables  $m$   
invariants  $m = \max(samples)$   
events...
```

This is called a **gluing invariant**: it specifies the relationship between the abstract and refined variables.

Proving that the gluing invariant is maintained

Abstract Add: $samples := samples \cup \{x\}$

Refined Add: $m := \max(\{m, x\})$

Proving that the gluing invariant is maintained

Abstract Add: $samples := samples \cup \{x\}$

Refined Add: $m := \max(\{m, x\})$

Assume: $m = \max(samples)$

Prove: $\max(\{m, x\}) = \max(samples \cup \{x\})$

Proving that the gluing invariant is maintained

Abstract Add: $samples := samples \cup \{x\}$

Refined Add: $m := \max(\{m, x\})$

Assume: $m = \max(samples)$

Prove: $\max(\{m, x\}) = \max(samples \cup \{x\})$

This is valid since:

$$\max(s \cup \{x\}) =$$

Proving that the gluing invariant is maintained

Abstract Add: $samples := samples \cup \{x\}$

Refined Add: $m := \max(\{m, x\})$

Assume: $m = \max(samples)$

Prove: $\max(\{m, x\}) = \max(samples \cup \{x\})$

This is valid since:

$$\max(s \cup \{x\}) = \max(\{\max(s), x\})$$

Closing Messages

- ▶ Role of **formal modelling**:
 - ▶ increase understanding
 - ▶ decrease errors
- ▶ Role of **refinement**:
 - ▶ manage complexity through multiple levels of abstraction
- ▶ Role of **verification**:
 - ▶ improve quality of models (consistency, invariants)
- ▶ Role of **tools**:
 - ▶ make verification as automatic as possible, pin-pointing errors