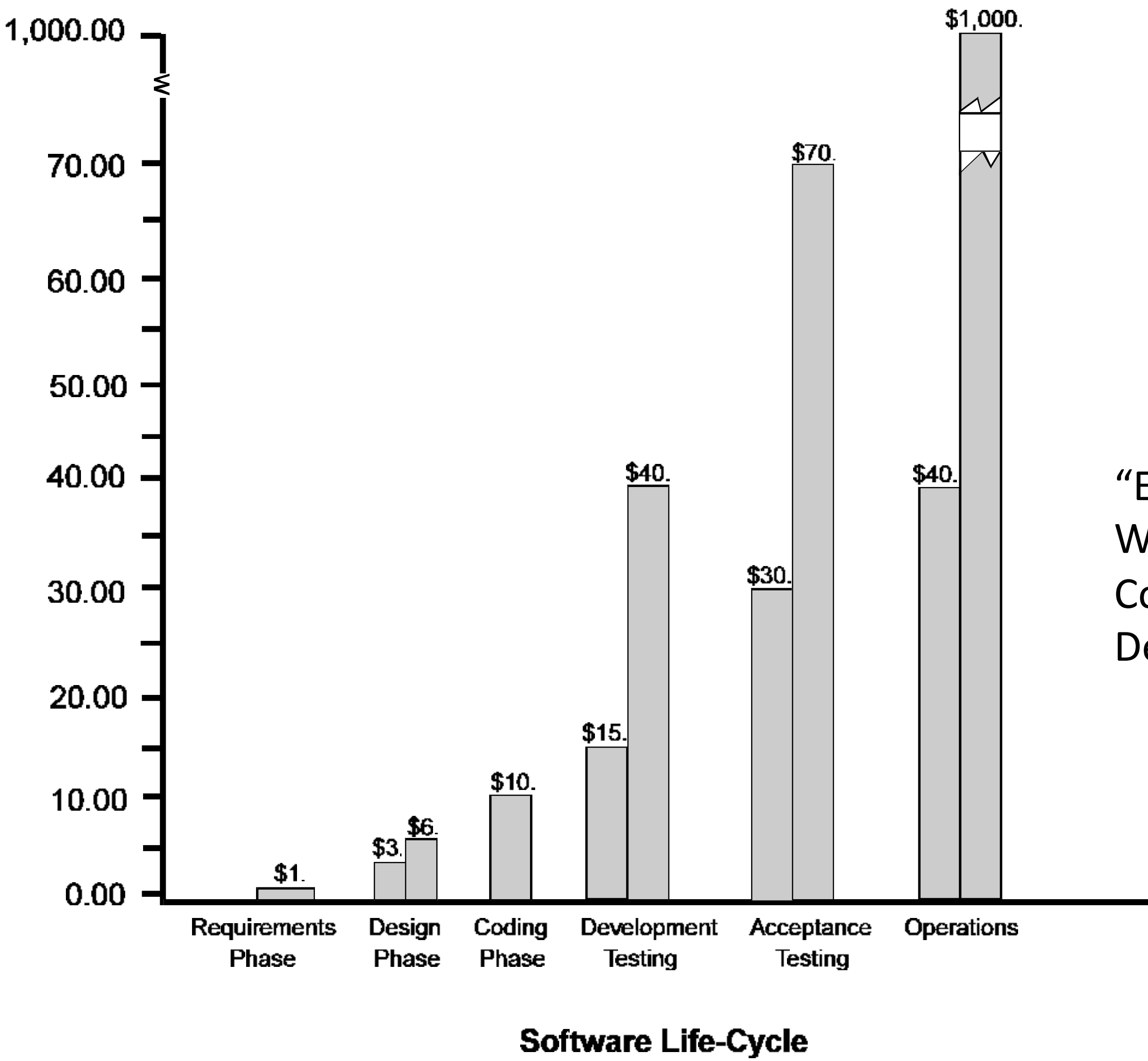
Introduction to Modelling
(with Event-B)

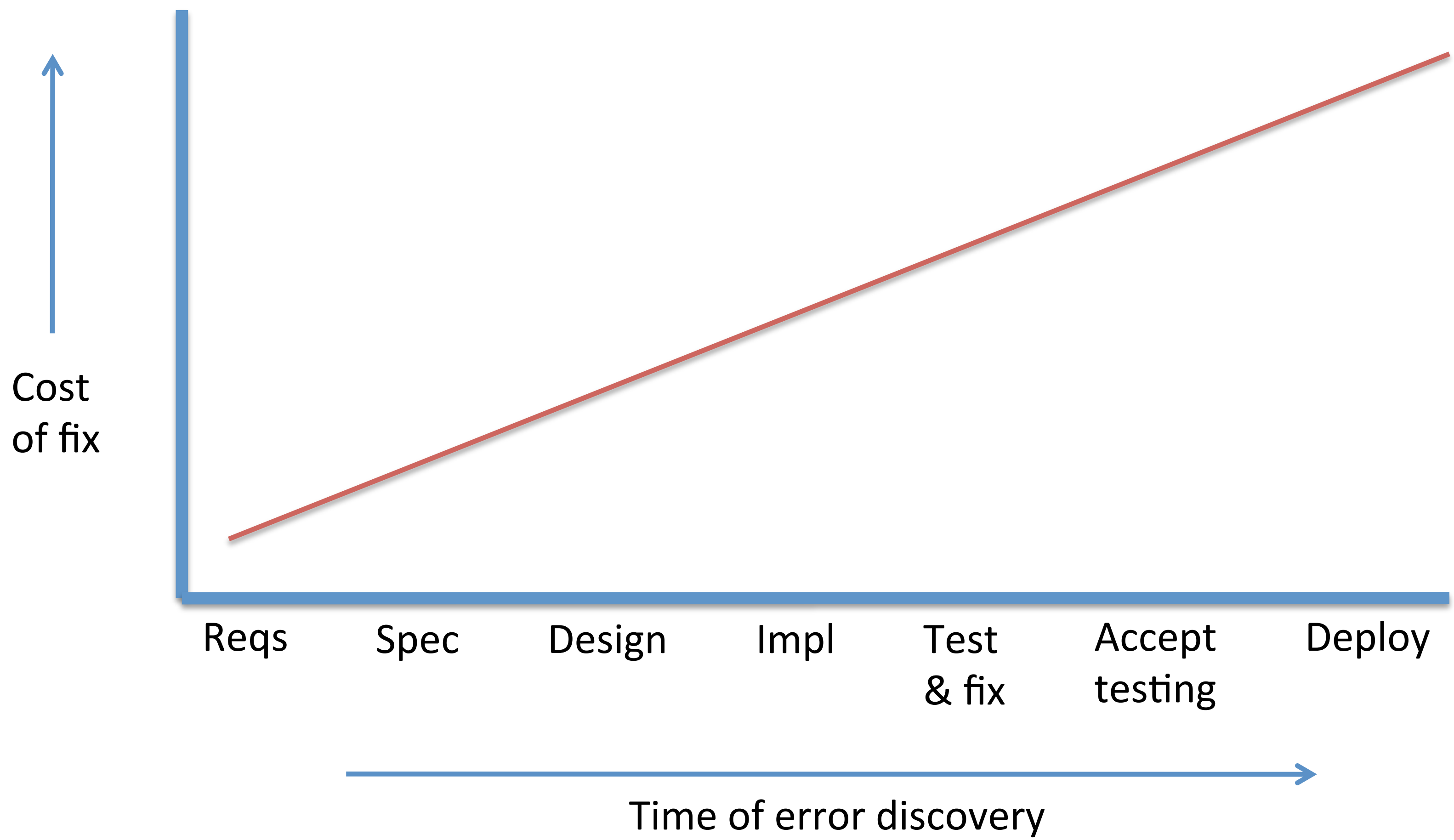
www.event-b.org

Cost of fixing requirements errors

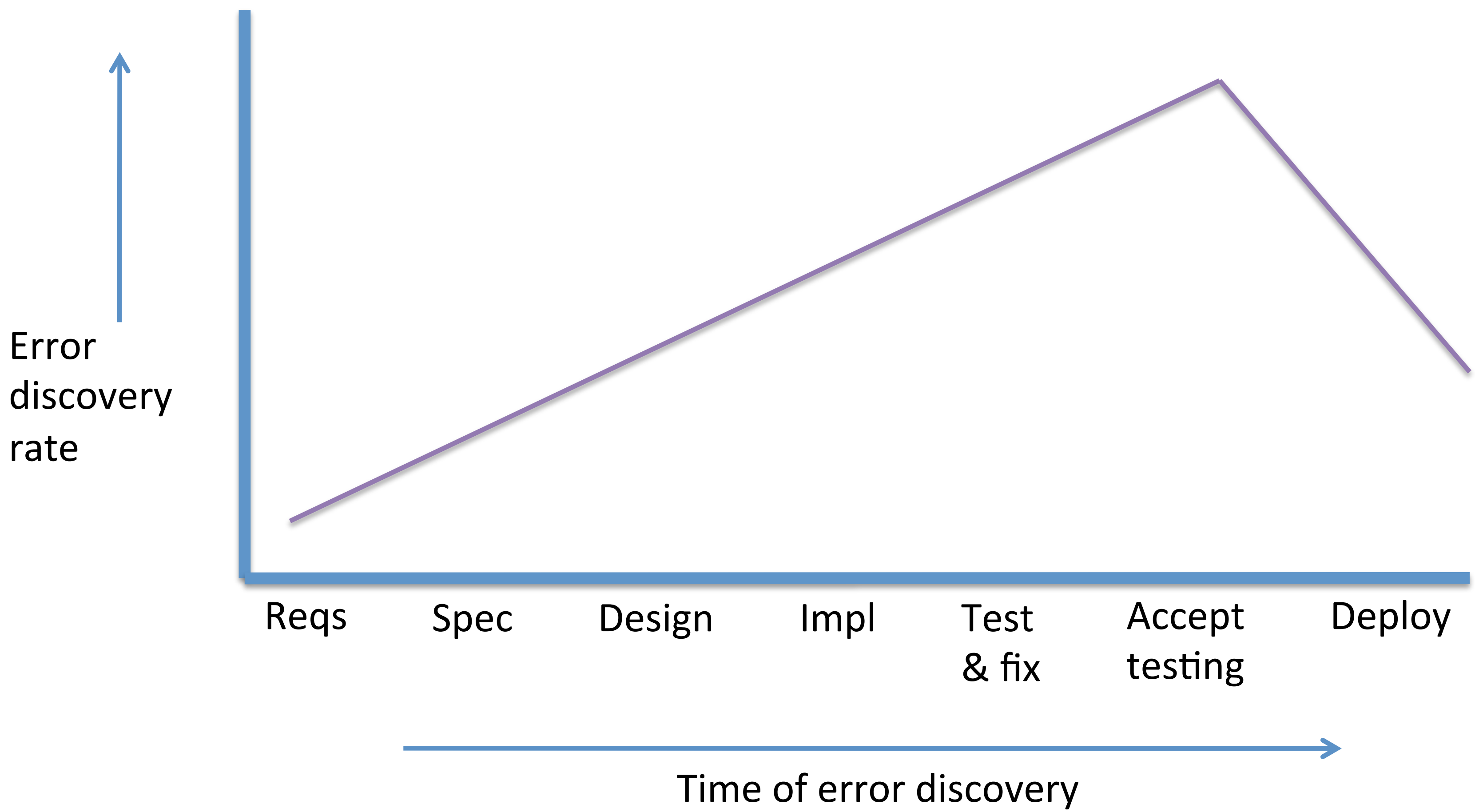


"Extra Time Saves Money"
Warren Kuffel
Computer Language
December 1990

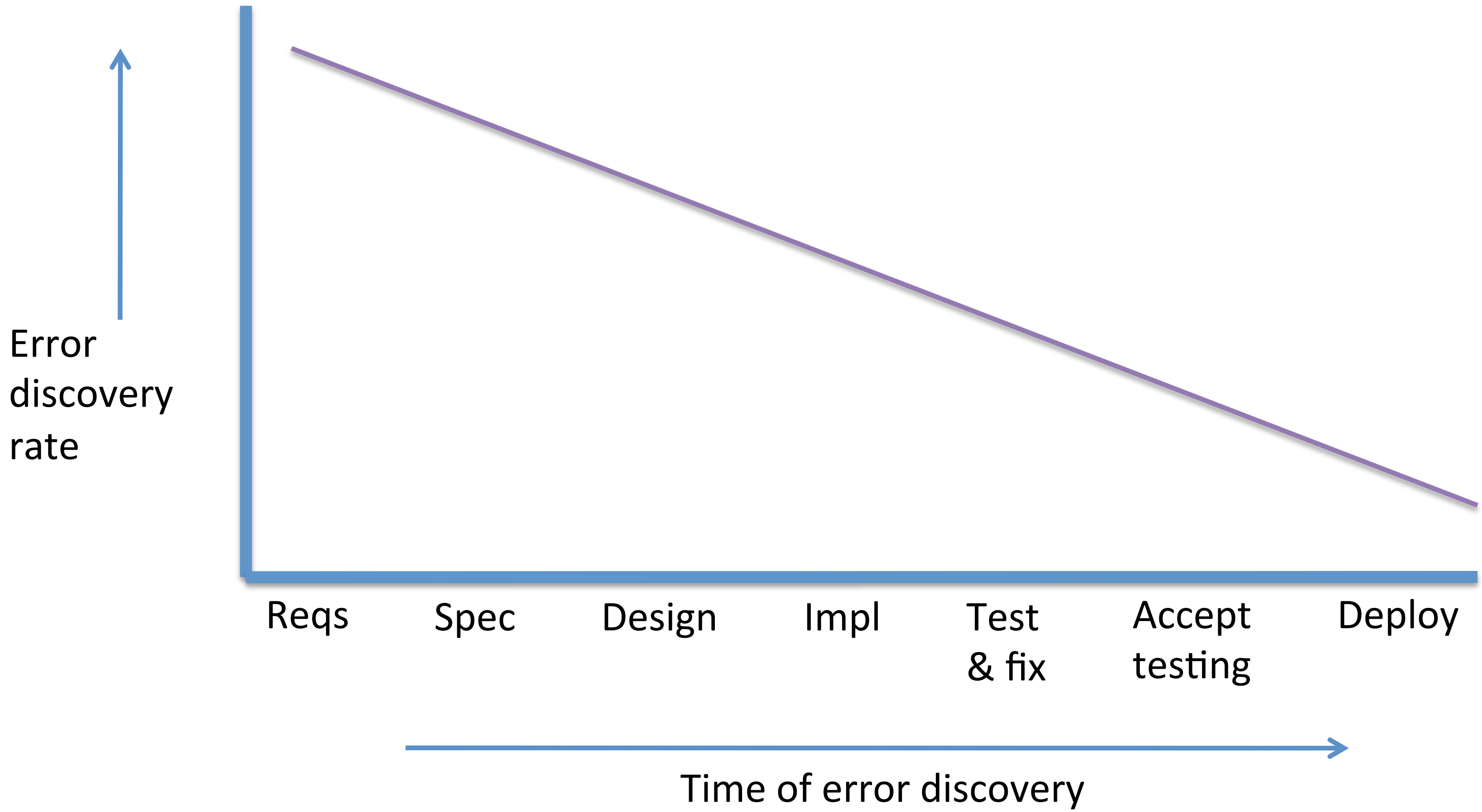
Cost of error fixes grows - difficult to change this



Rate of error discovery



Invert error identification rate?



Why is it difficult to identify errors?

- Lack of precision
  - ambiguities
  - inconsistencies

- Too much complexity
  - complexity of requirements
  - complexity of operating environment
  - complexity of designs

Need for precision and abstraction at early stages (front-loading)

- Precision through early stage models
  - Amenable to analysis by tools
  - Identify and fix ambiguities and inconsistencies as early as possible

- Mastering complexity through abstraction
  - Focus on *what* a system does (its purpose)
  - Incremental analysis and design

Rational design, by example

- Example: access control system

- Example intended to give a feeling for:
  - problem abstraction
  - modelling language
  - model refinement
  - role of verification and Rodin tool

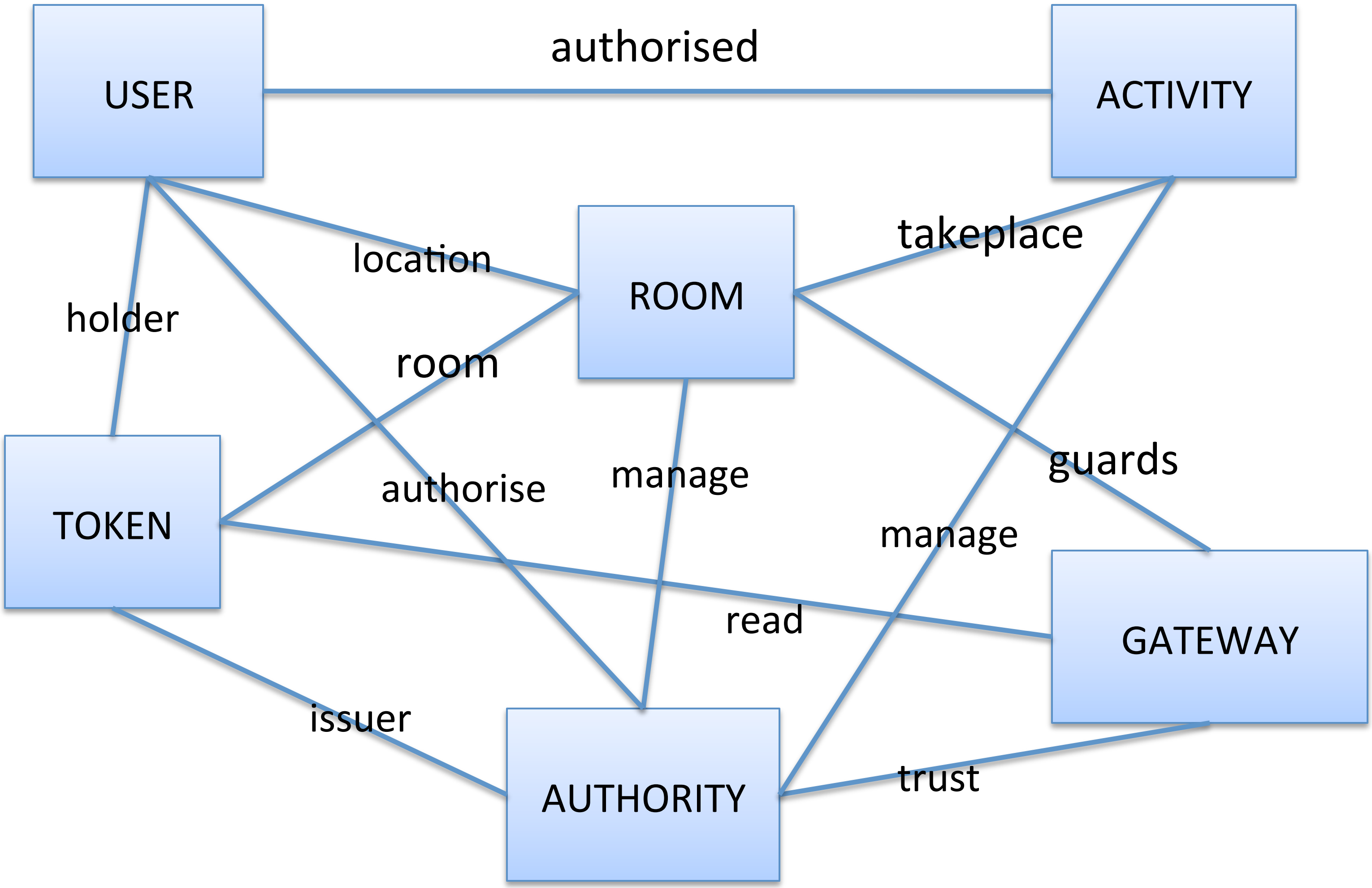
Access control requirements

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

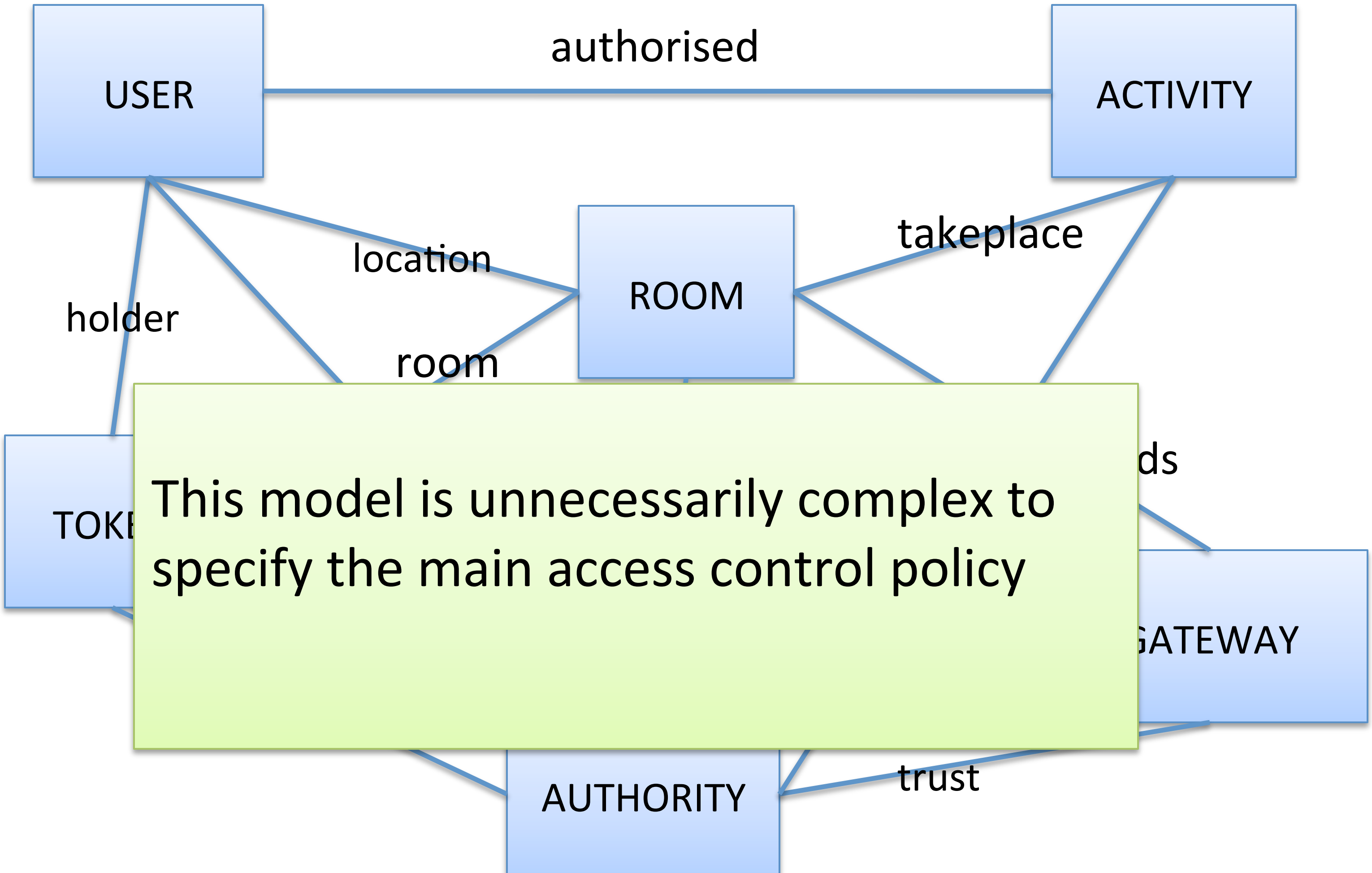
Access control requirements

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

Entities and relationships



Entities and relationships

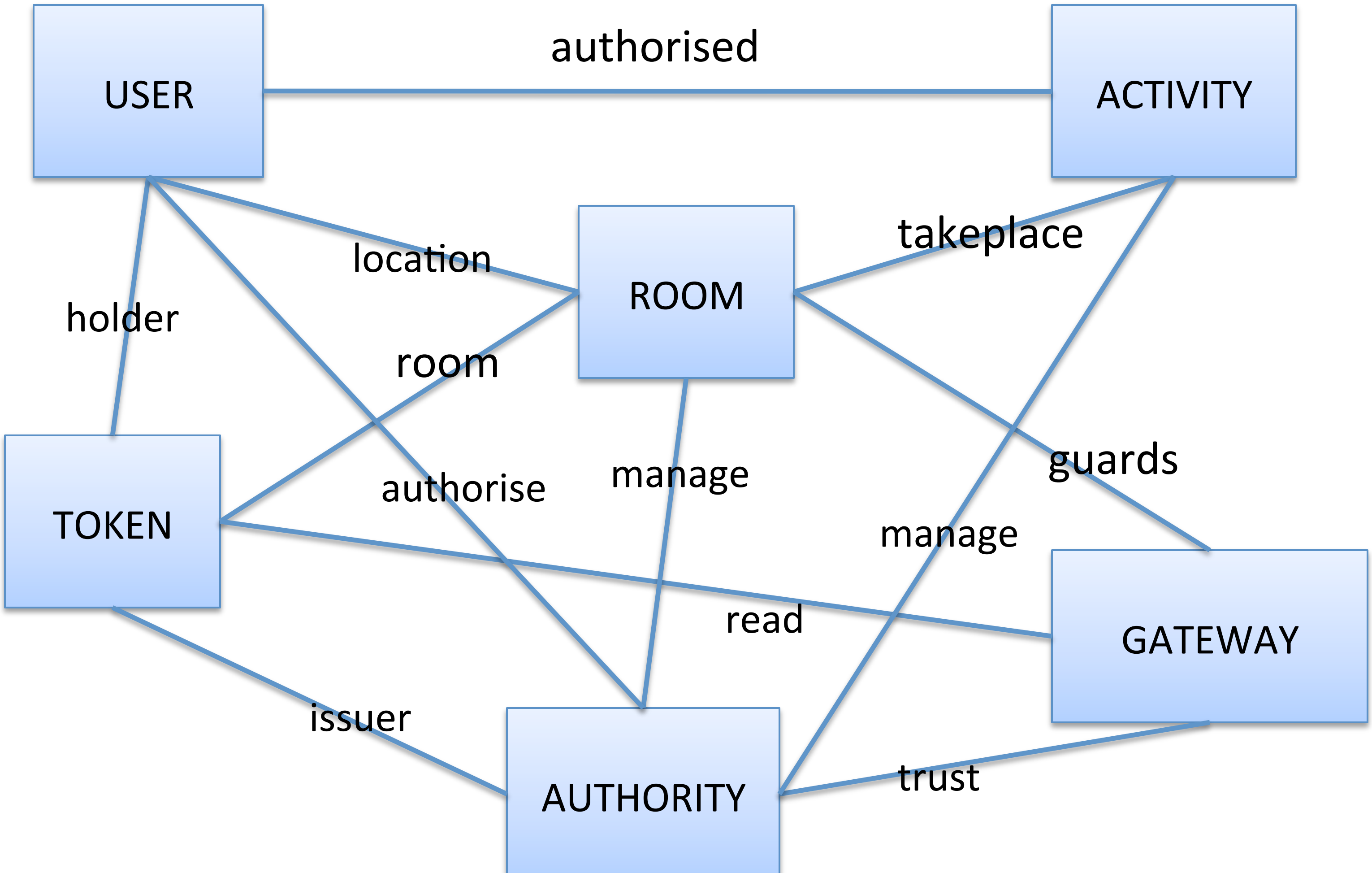


This model is unnecessarily complex to specify the main access control policy

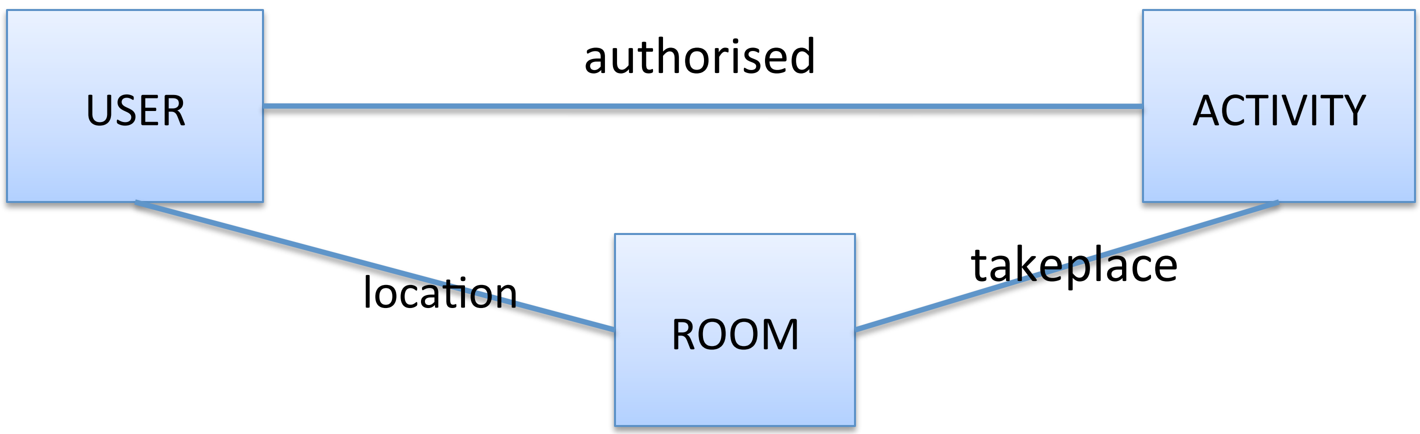
Extracting the essence

- Purpose of our system is to enforce an access control policy
- Access Control Policy: *Users may only be in a room 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 related to the purpose of the system

Entities and relationships



Abstract by removing entities



Relationships represented in Event-B

authorised $\in$ USER $\leftrightarrow$ ACTIVITY // relation

takeplace $\in$ ROOM $\leftrightarrow$ ACTIVITY // relation

location $\in$ USER $\mapsto$ ROOM // partial function

Access control invariant

$$\forall u,r . \quad u \in \mathrm{dom}(location) \;\wedge$$
$$location(u) = r$$
$$\Rightarrow$$
$$takeplace[r] \subseteq authorised[u]$$

if user *u* is in room *r*,
then *u* must be authorised to engaged in
all activities that can take place in *r*

State snapshot as tables

USER	ACTIVITY
u1	a1
u1	a2
u2	a1

authorised

ROOM	ACTIVITY
r1	a1
r1	a2
r2	a1

takeplace

USER	ROOM
u1	r1
u2	r2
u3	

location

Event for entering a room

Enter(u,r) $\triangleq$

when

grd1 : $u \in USER$

grd2 : $r \in ROOM$

grd3 : $takeplace[\, r \,] \subseteq authorised[\, u \,]$

then

act1 : $location(u) := r$

end

Does this event maintain the access control invariant?

Role of invariants and guards

- Invariants: specify properties of model variables that should *always* remain true
  - violation of invariant is undesirable (safety)
  - use (automated) proof to verify invariant preservation

- Guards: specify *enabling conditions* under which events may occur
  - should be strong enough to ensure invariants are maintained by event actions
  - but not so strong that they prevent desirable behaviour (liveness)

Remove authorisation

RemoveAuth(u,a) $\widehat{=}$

when

grd1 : $u \in USER$

grd2 : $a \in ACTIVITY$

grd3 : $u \mapsto a \in authorised$

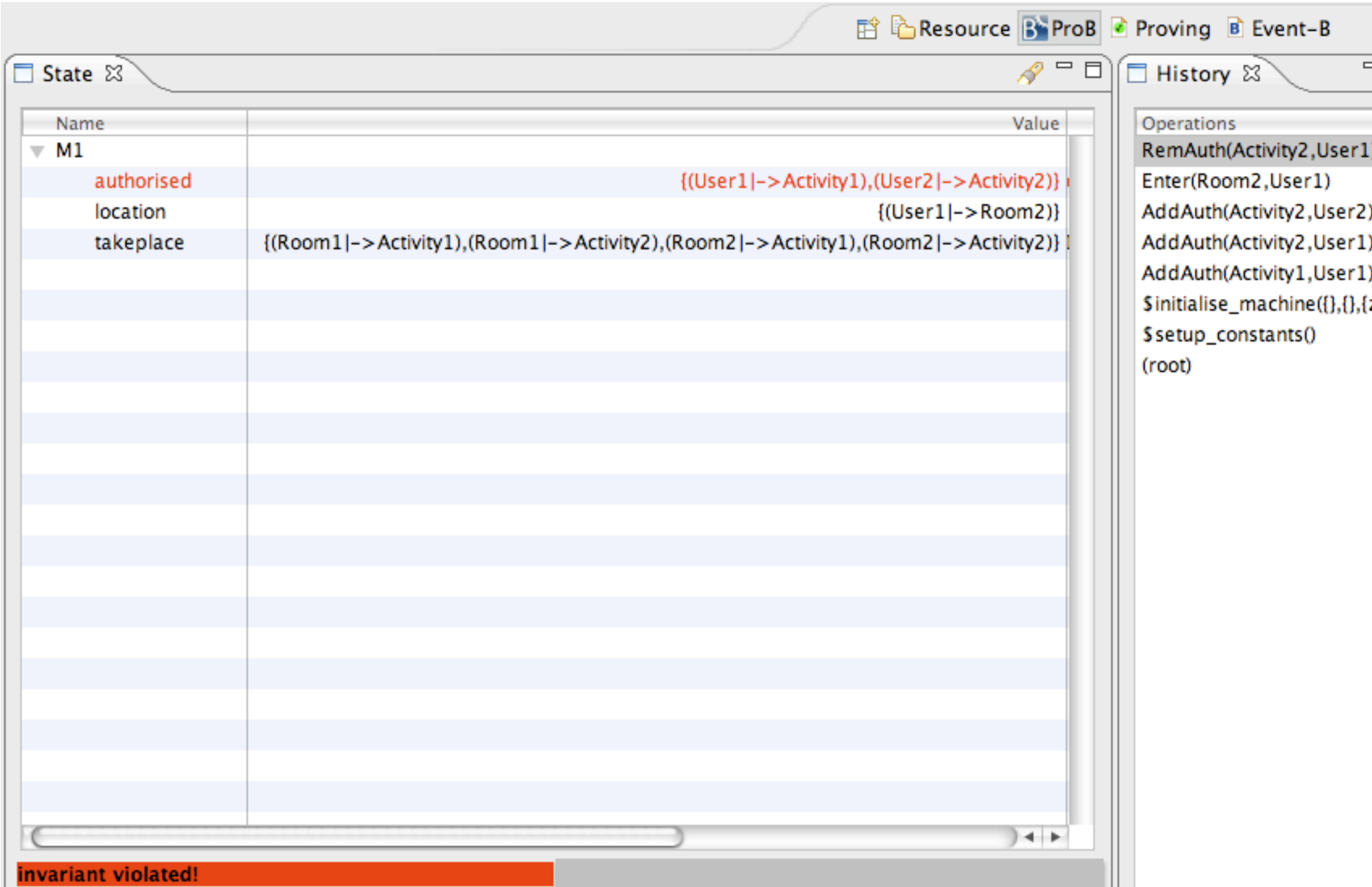
then

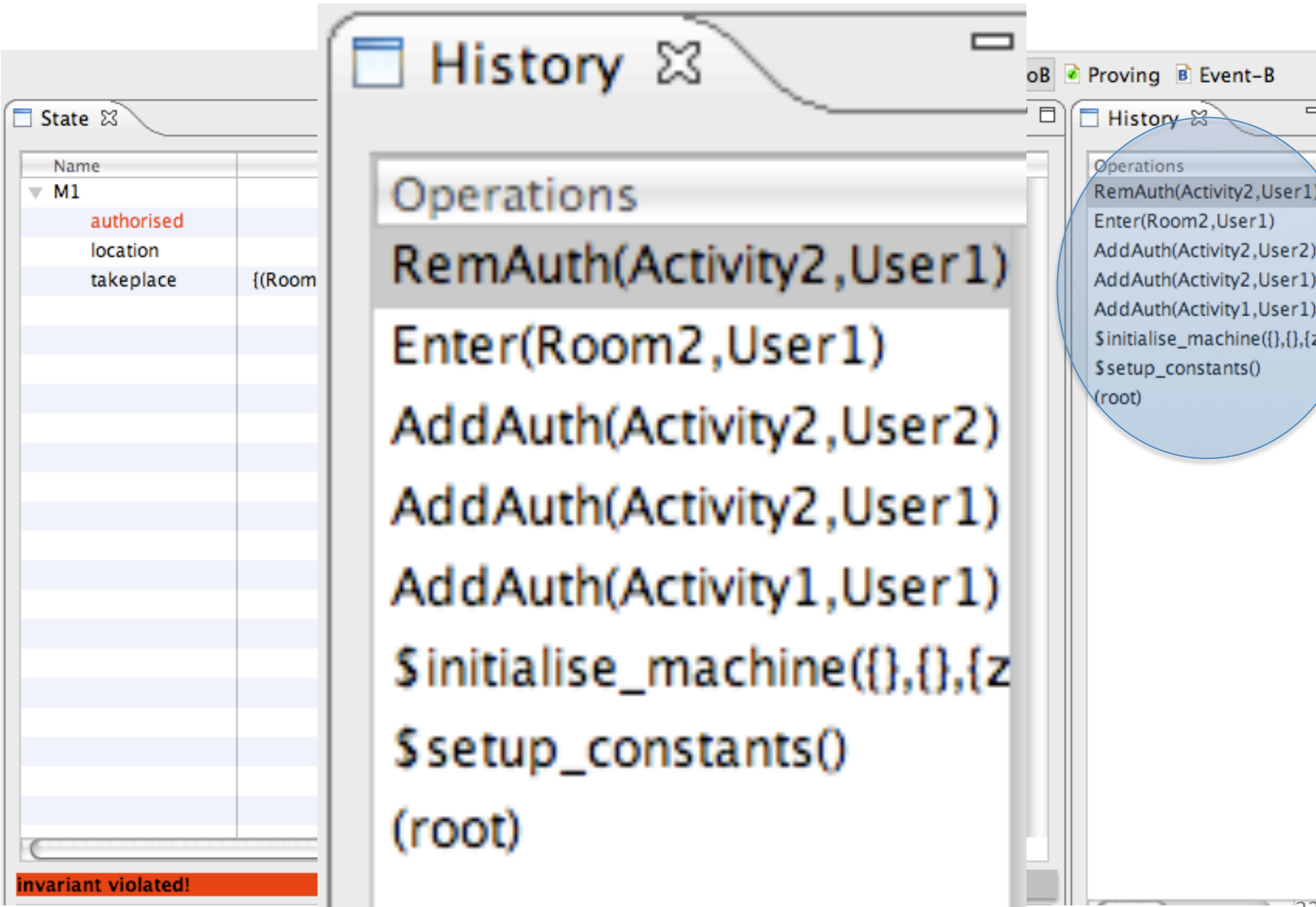
act1 : $authorised := authorised \setminus \{ u \mapsto a \}$

end

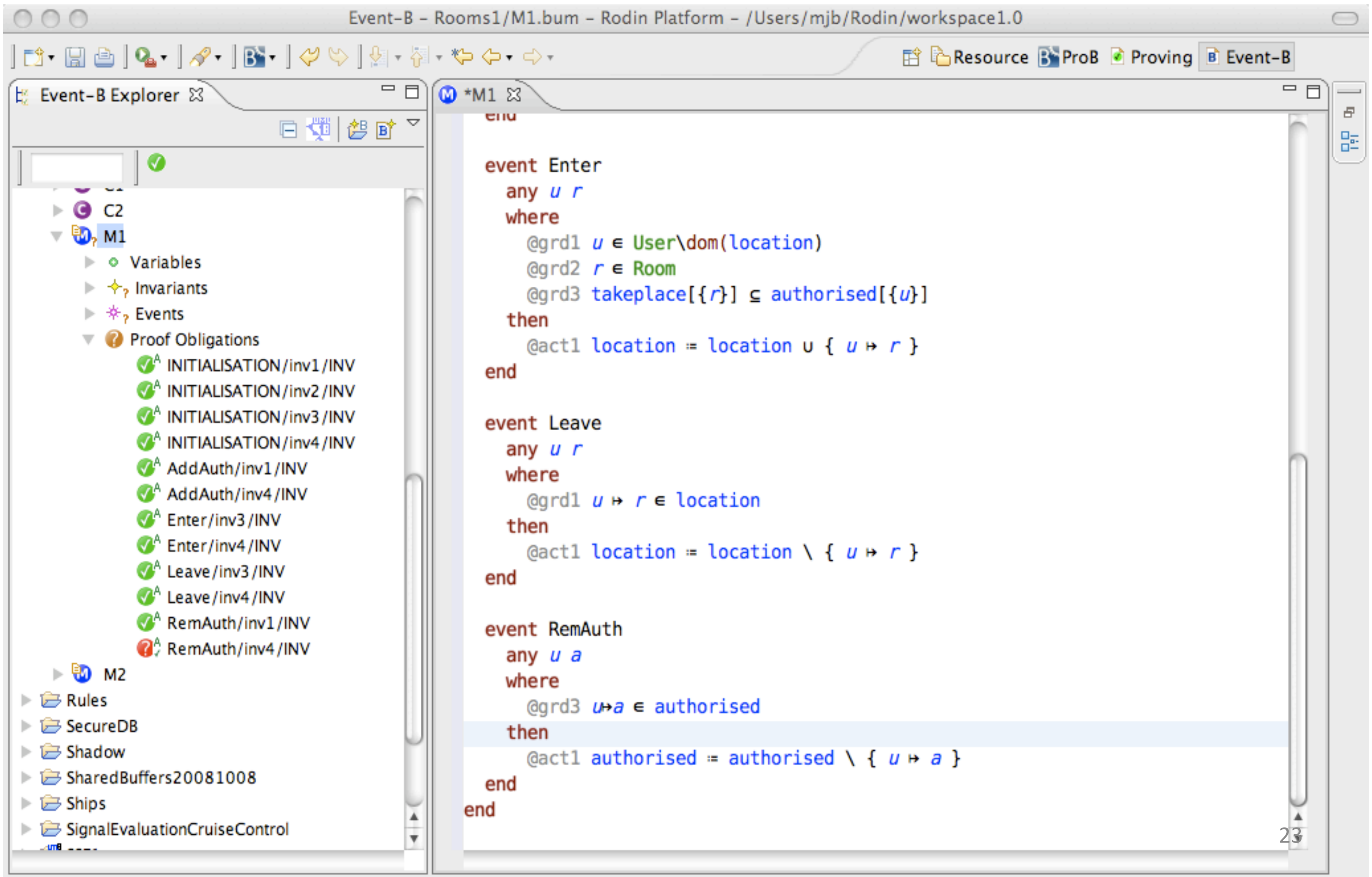
Does this event maintain the access control invariant?

Counter-example from model checking

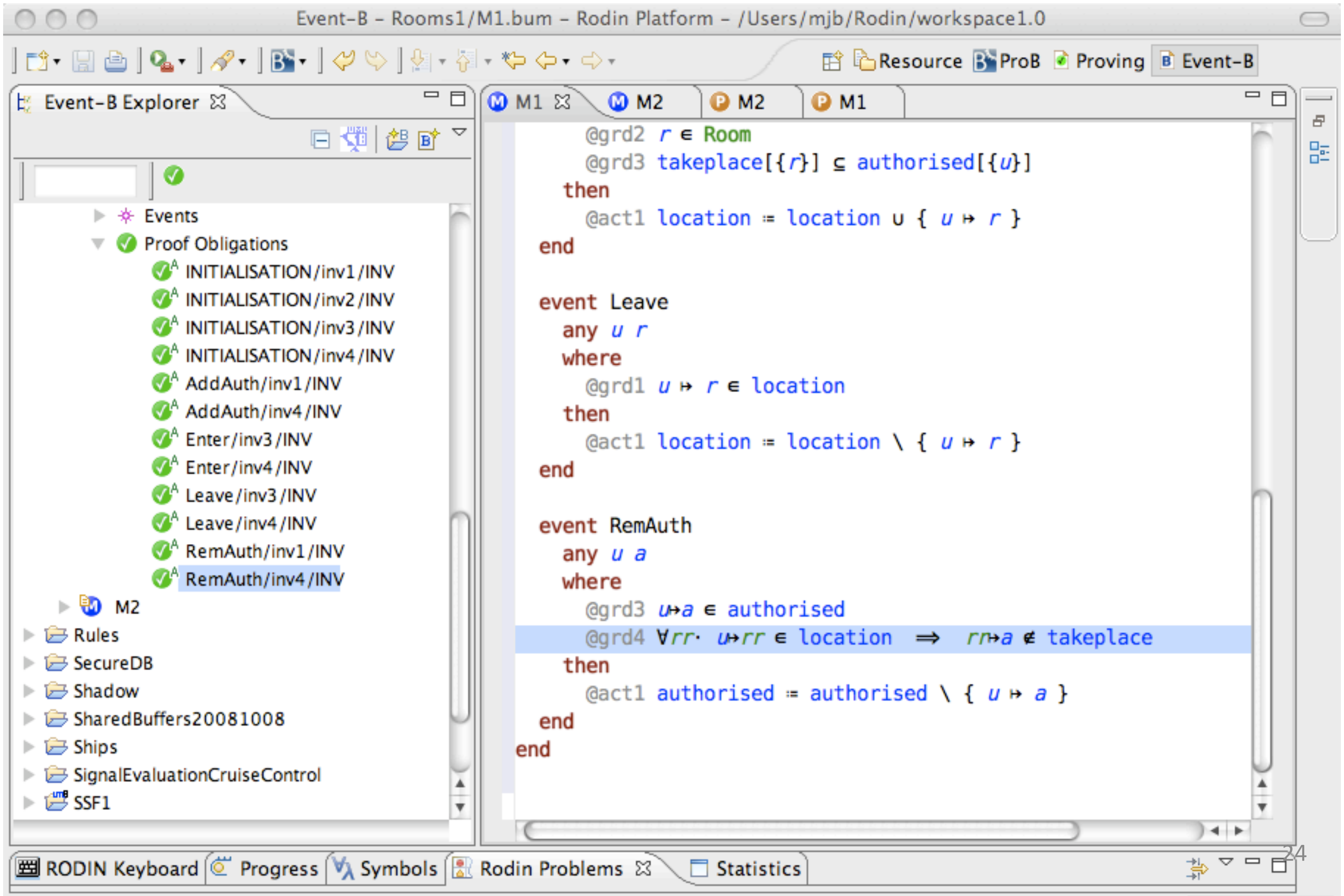



State
Name
M1
authorised
location
takeplace
{(Room
invariant violated!
History
Operations
RemAuth(Activity2,User1)
Enter(Room2,User1)
AddAuth(Activity2,User2)
AddAuth(Activity2,User1)
AddAuth(Activity1,User1)
$initialise_machine({},{},{z
$setup_constants()
(root)
oB
Proving
Event-B
History
Operations
RemAuth(Activity2,User1)
Enter(Room2,User1)
AddAuth(Activity2,User2)
AddAuth(Activity2,User1)
AddAuth(Activity1,User1)
$initialise_machine({},{},{z
$setup_constants()
(root)

Failing proof



Strengthen guard of *RemAuth*



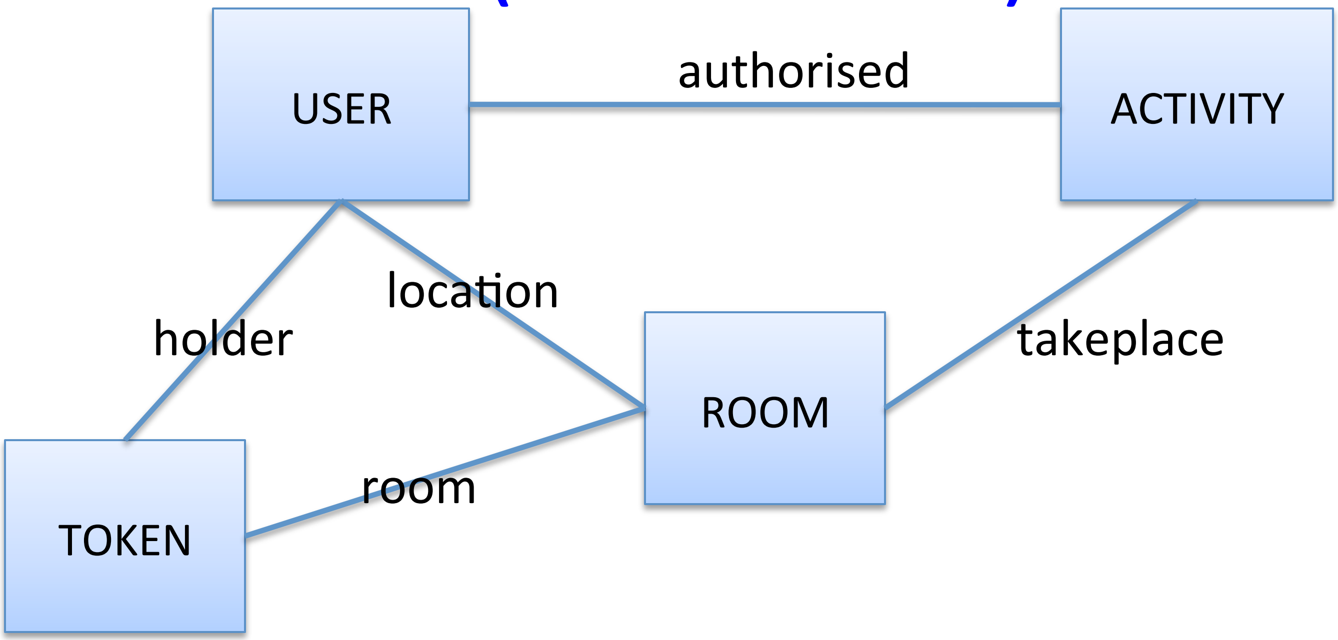
Early stage analysis

- We constructed a simple abstract model

- Already using verification technology we were able to identify errors in our conceptual model of the desired behaviour
  - we found a solution to these early on
  - verified the "correctness" of the solution

- Now, lets proceed to another stage of analysis…

We construct a new model (refinement)



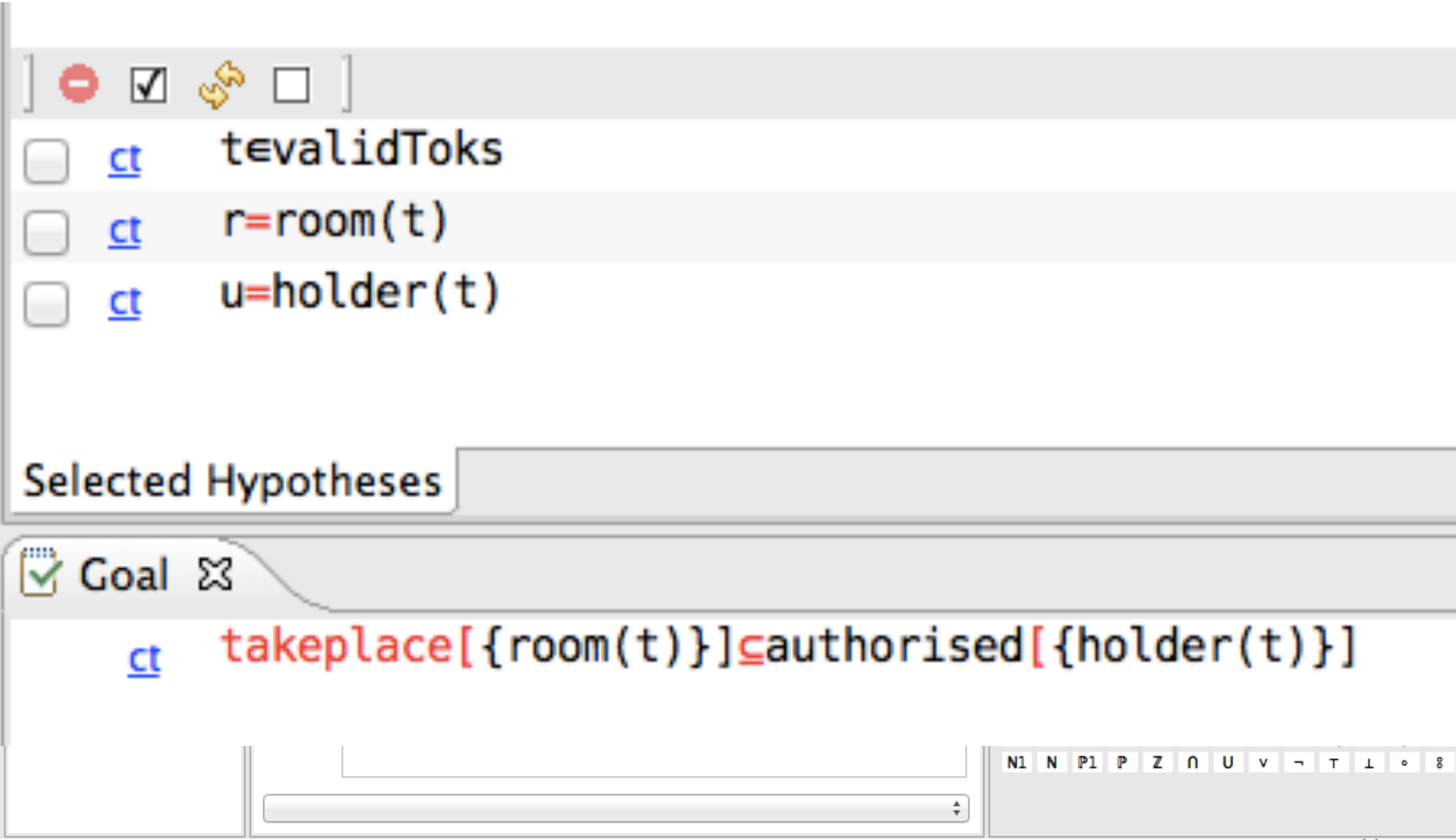
Guard of abstract Enter event:

grd3: $takeplace[\ r\ ] \subseteq authorised[\ u\ ]$

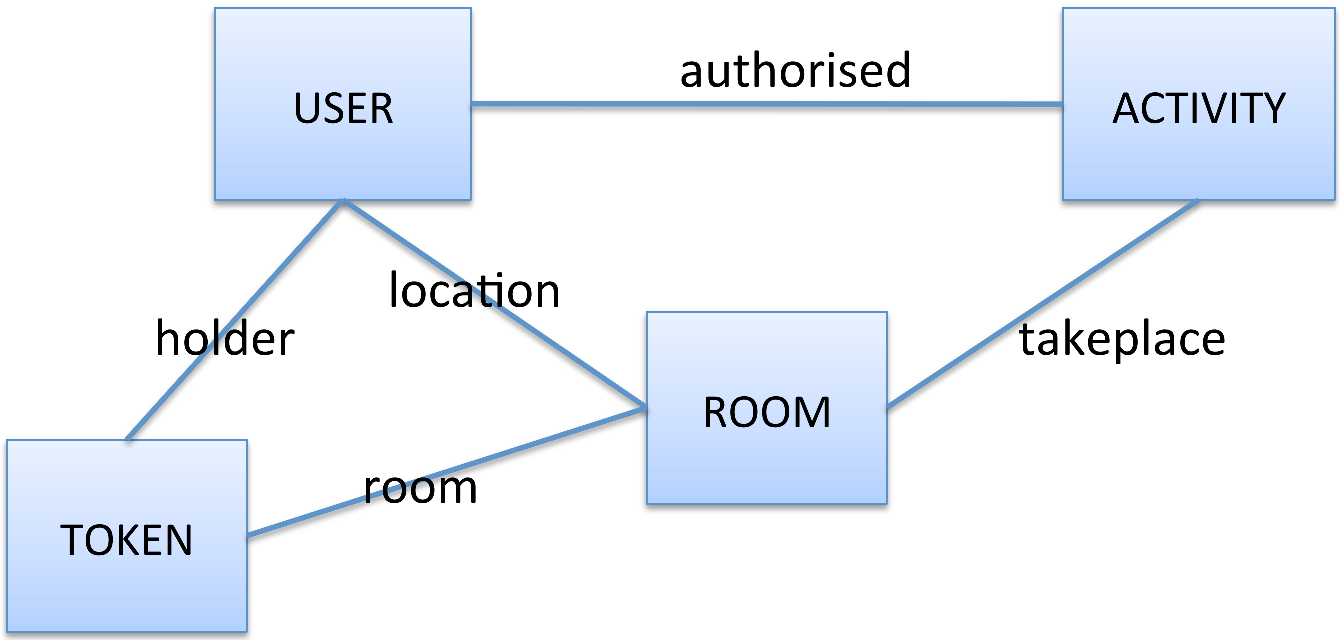
is replaced by a guard on a token:

grd3b: $t \in valid \ \wedge\ room(t) = r \ \wedge\ holder(t) = u$

Failing refinement proof



Gluing invariant



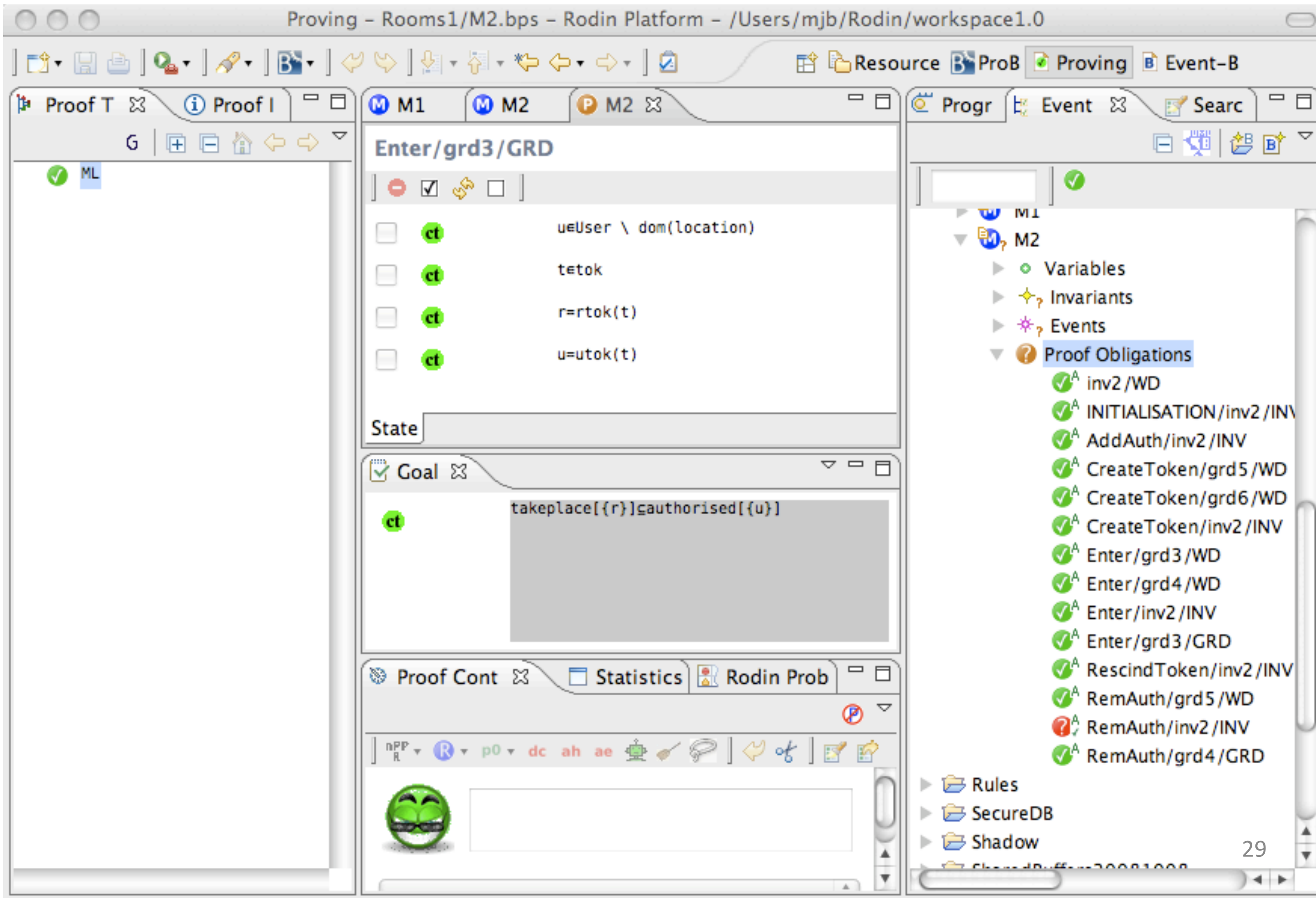
To ensure consistency of the refinement we need invariant:

inv 6: $t \in valid$

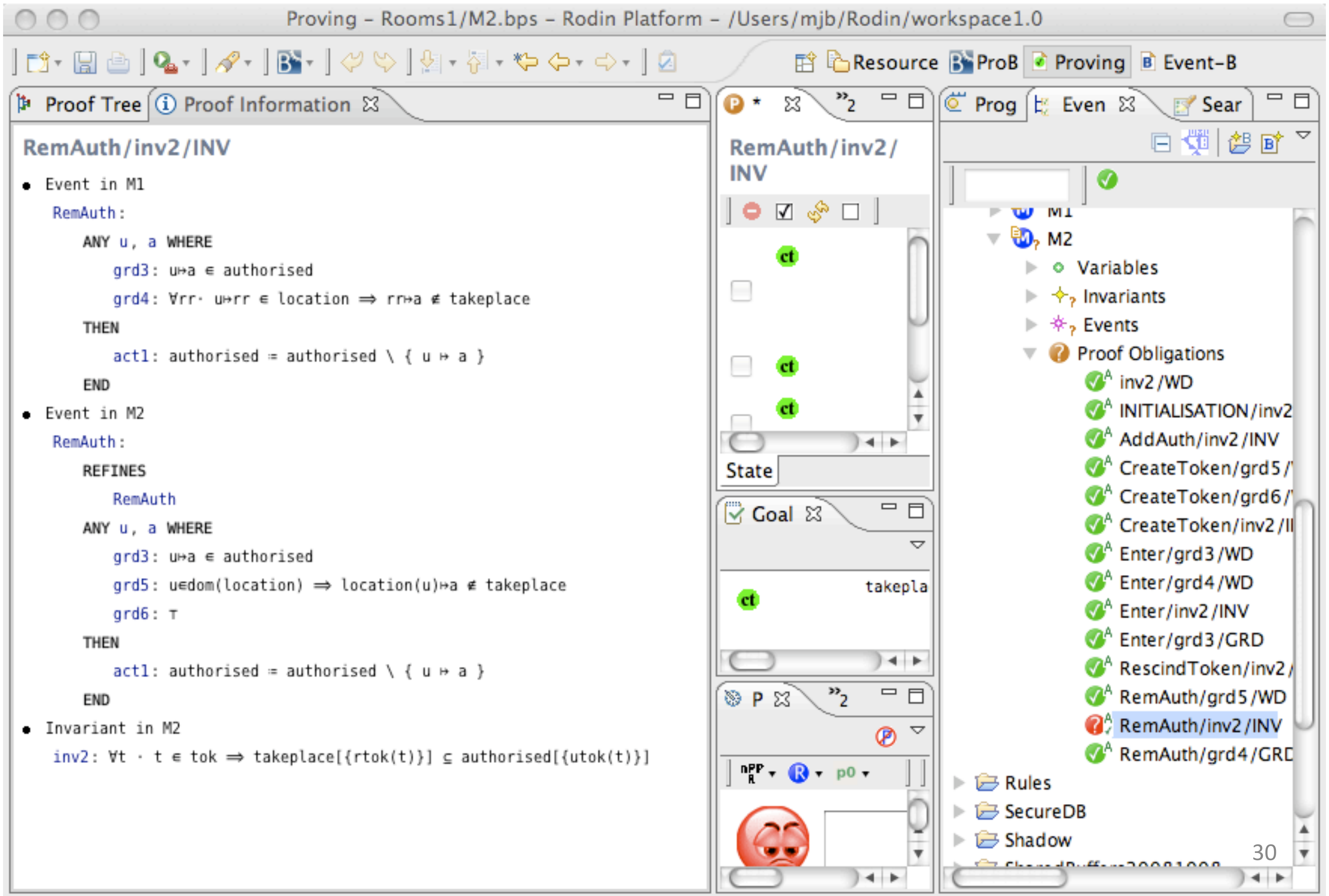
$\Rightarrow$

$takeplace\,[\,room(t)\,] \subseteq authorised[\,holder(t)\,]$

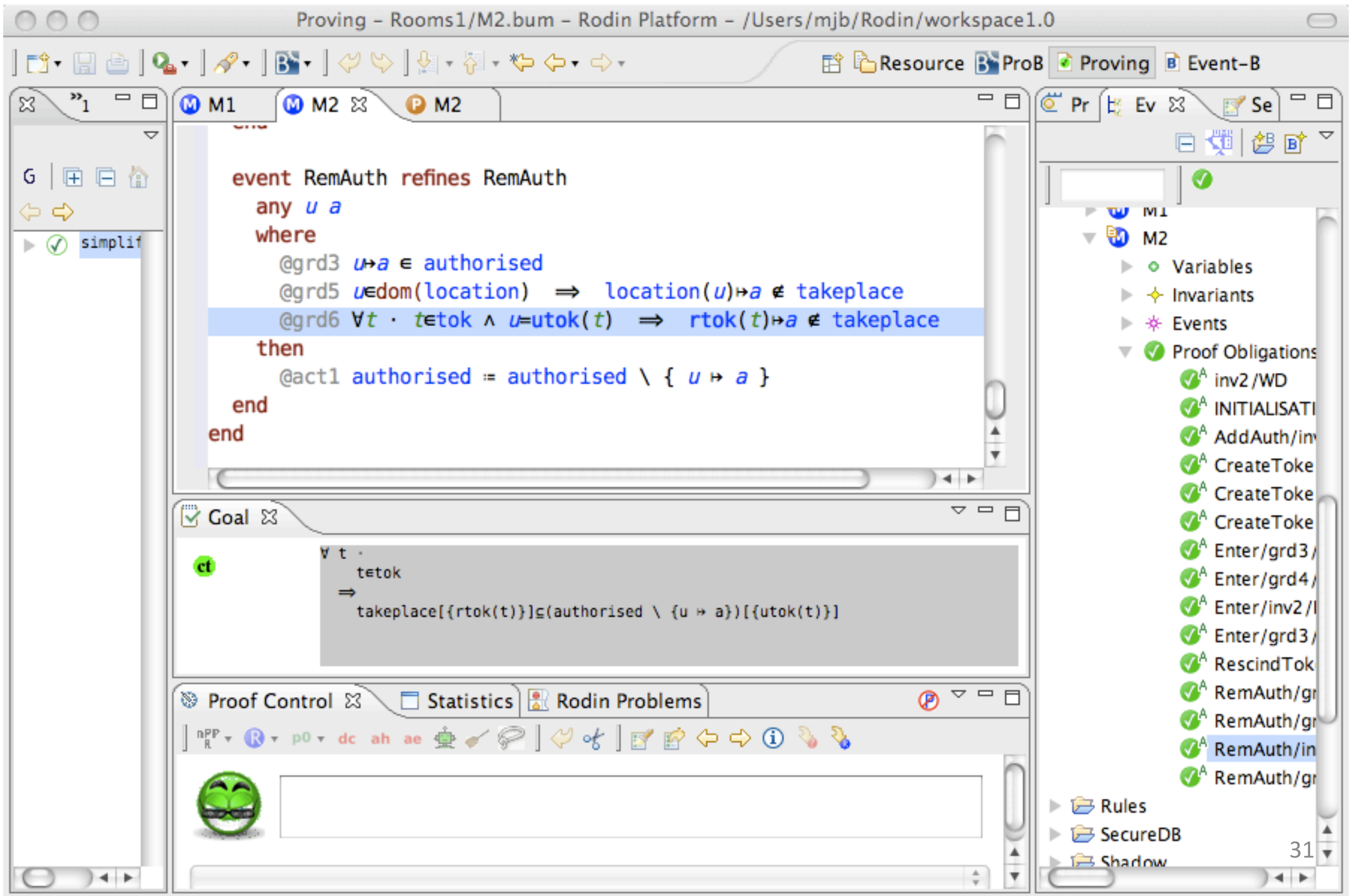
Invariant enables PO discharge



But get new failing PO



Strengthen guard of refined *RemAuth*



Requirements revisited

1. Users are authorised to engage in activities
2. User authorisation may be added or revoked
3. Activities take place in rooms
4. …

Question: was it obvious initially that revocation of authorisation was going to be problematic?

Rational design – what, how, why

- *What* does it achieve?

  if user *u* is in room *r*,

  then *u* must be authorised to engaged in
  all activities that can take place in *r*

- *How* does it work?

  Check that a user has a valid token

- *Why* does it work?

  For any valid token *t*, the holder of *t* must be authorised to engage in all activities that can take place in the room associated with *t*

What, how, why written in B

- *What* does it achieve?

  inv1: $u \in dom(location) \wedge location(u) = r$
  $\Rightarrow$
  $takeplace[r] \subseteq authorised[u]$

- *How* does it work?

  grd3b: $t \in valid \wedge r = room(t) \wedge u = holder(t)$

- *Why* does it work?

  inv2: $t \in valid$
  $\Rightarrow$
  $takeplace[room(t)] \subseteq authorised[holder(t)]$

System level reasoning

- Examples of systems modelled in Event-B:
  - Train signalling system
  - Mechanical press system
  - Access control system
  - Air traffic information system
  - Electronic purse system
  - Distributed database system
  - Cruise control system
  - Processor Instruction Set Architecture
  - …
- System level reasoning:
  - Involves abstractions of *overall* system not just software components

Problem 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/analyst should make judgements about what they believe to be the key features of the system.

Abstraction (continued)

- 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, monitored or protected?

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
- Facilitates abstraction: we can postpone treatment of some system features to later refinement steps
- Event-B provides a notion of consistency of a refinement:
  - Use proof to verify the consistency of a refinement step
  - Failing proof can help us identify inconsistencies