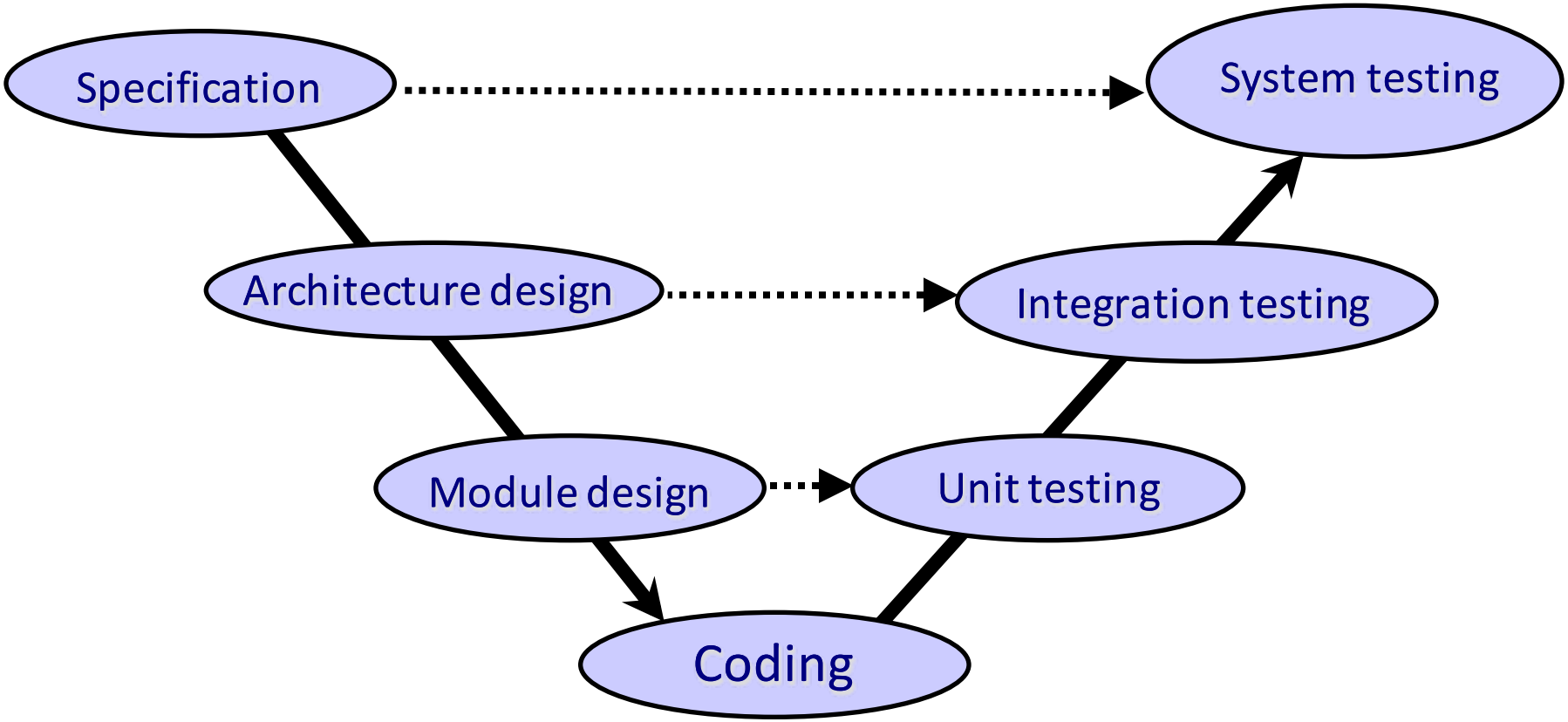
UNIVERSITY OF Southampton

Modelling and verification with Event-B

Michael Butler

SETSS 2016, Chongqing

V model of software development



B Method (Abrial, from 1990s)

- *Model* using set theory and logic (following Z notation)
- *Analyse models* using proof, model checking, animation
- Refinement-based development
  - verify conformance between *higher-level* and *lower-level* models
  - chain of refinements
- Commercial tools, :
  - *Atelier-B* (ClearSy, FR) - used mainly in railway industry
  - *B-Toolkit* (B-Core, UK)

B evolves to Event-B (from 2004)

- B Method was designed for *software* development
- Realisation that it is important to reason about *system* behaviour, not just software
- Event-B is intended for modelling and refining system behaviour
- Refinement notion is more flexible than B
  - Same set theory and logic
- Rodin tool for Event-B (www.event-b.org)
  - Open source, Eclipse based, open architecture
  - Range of plug-in tools (provers, ProB model checker, UML-B,…)

Industrial uses of Event-B

- Event-B in Railway Interlocking
  - Alstom, Systerel
- Event-B in Smart Grids
  - Selex, Critical Software
- External Adopters:
  - AWE: Experience of Applying Rodin in an Industrial Environment
  - Thales: Formal Modelling of Railway Interlocking Using Event-B and the Rodin Tool-chain

www.advance-ict.eu/industry_days

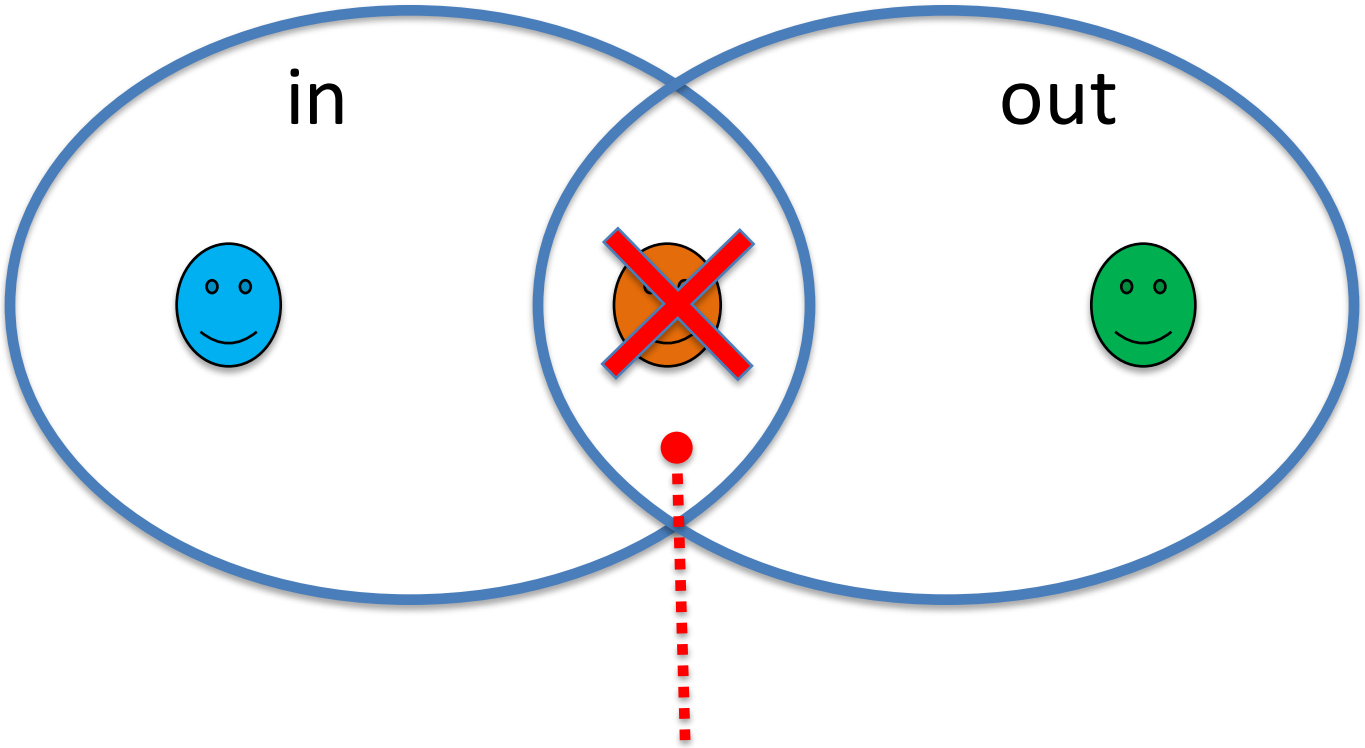
SETSS lectures on Event-B

- Modelling with sets and invariants
- Model verification with Rodin prover
- Modelling with relations, class diagrams
- Refinement
  - model extension
  - data refinement

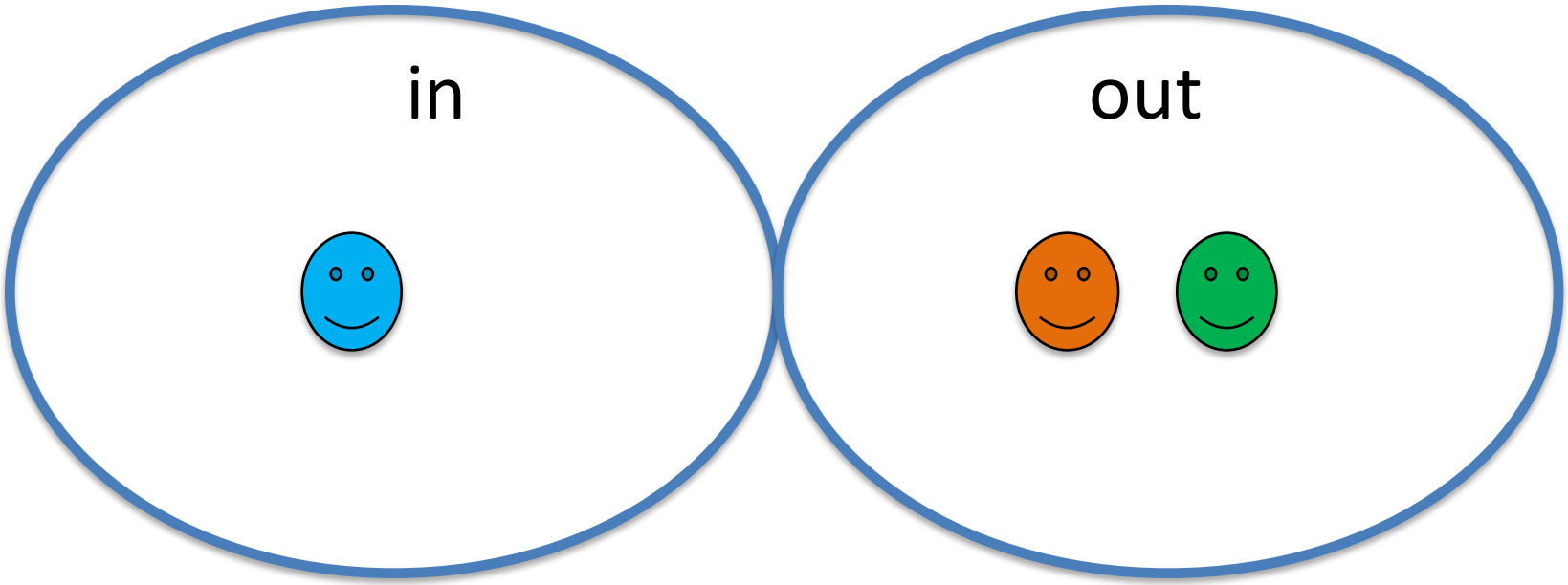
Example Requirements for a Building Control System

- Specify a system that monitors users entering and leaving a building.
- A person can only enter the building if they are a registered user.
- The system should be aware of whether a registered user is currently inside or outside the building.

Venn Diagram

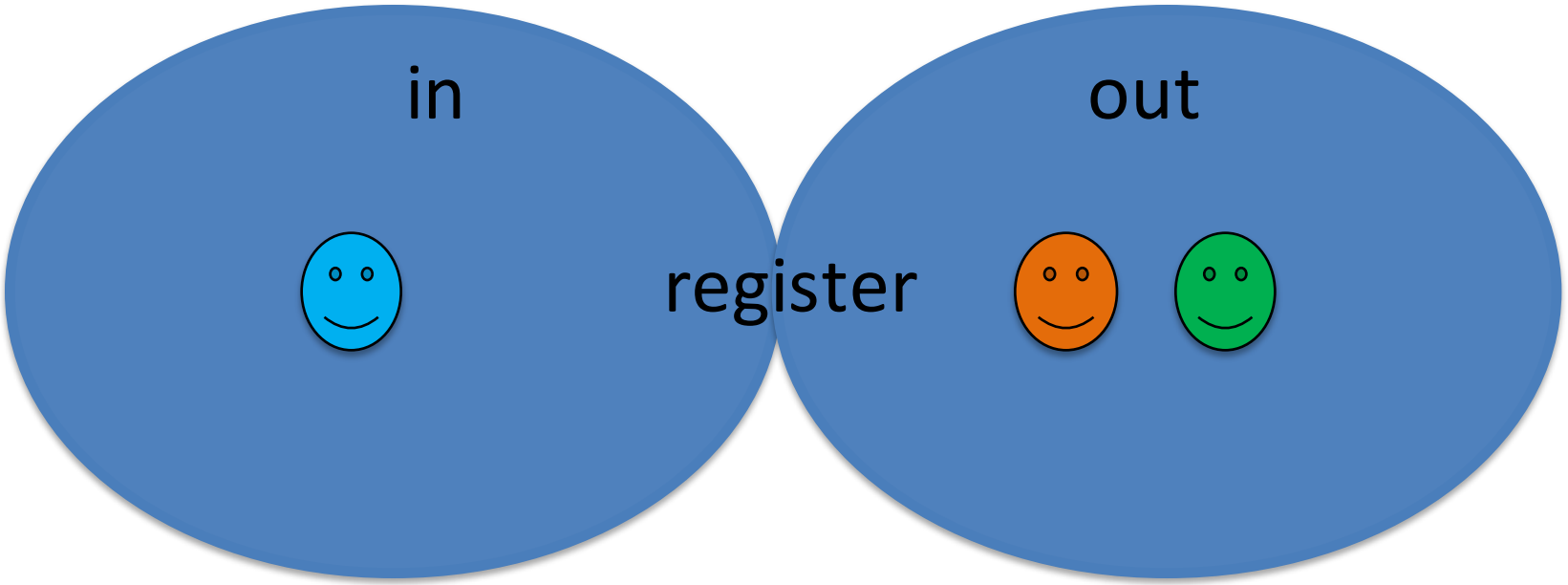


Disjoint sets



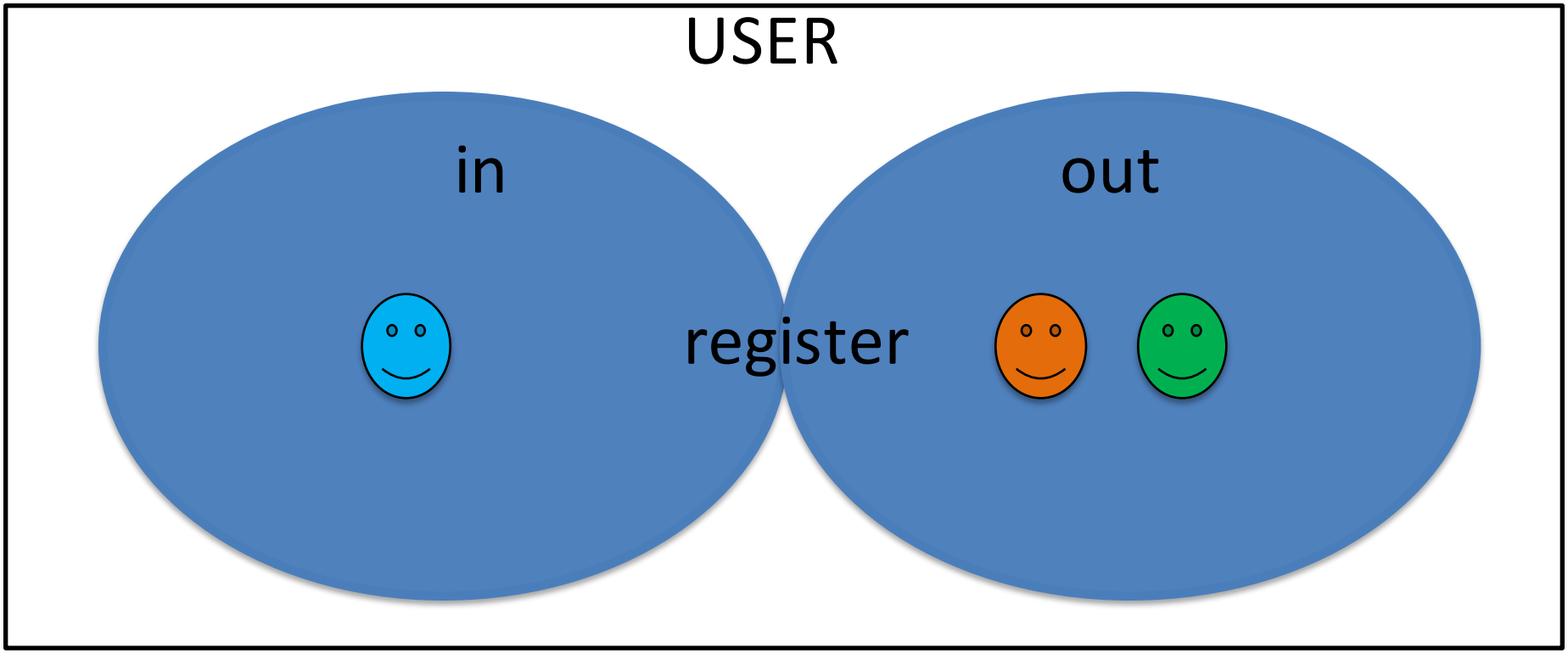
Invariant: $in \cap out = \{\}$

Registered users are either *in* or *out*



$$register = in \cup out$$

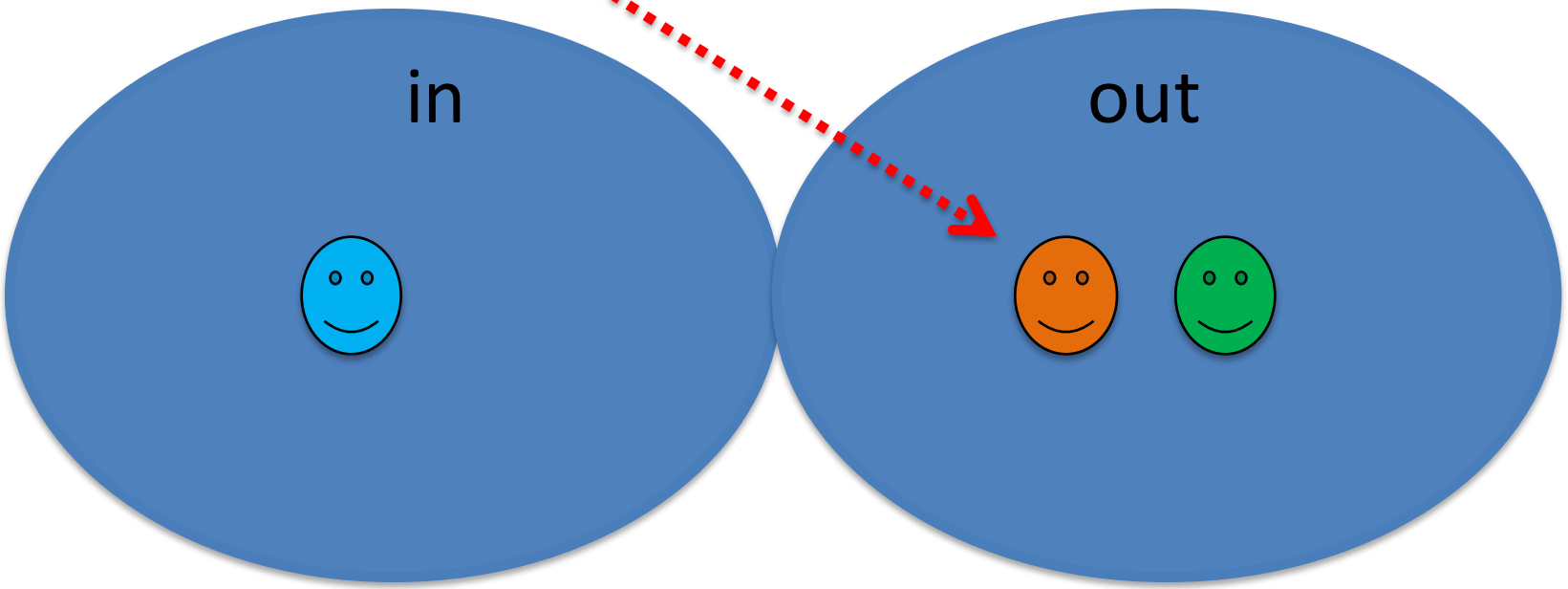
Carrier Set: type for users



$$register \subseteq USER$$

Event: user ***enters*** building

Guard: $s \in out$

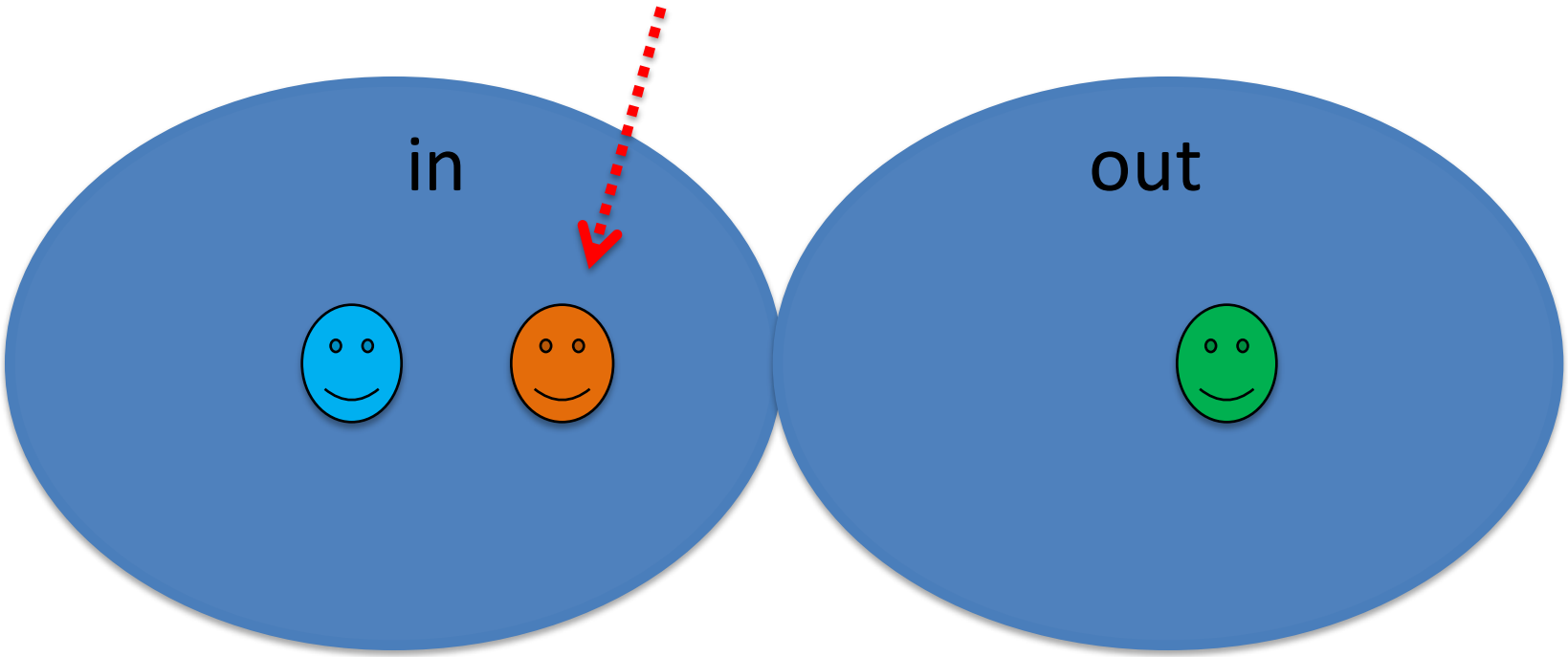


Action:

$$in := in \cup \{s\}$$
$$out := out \setminus \{s\}$$

Event: user ***leaves*** building

Guard: $s \in in$



Action:

$$in := in \setminus \{s\}$$
$$out := out \cup \{s\}$$

Basic Set Theory

- A set is a collection of elements.
- Elements of a set are not ordered.
- Elements of a set may be numbers, names, identifiers, etc.
- Sets may be finite or infinite.
- Relationship between an element and a set: is the element a member of the set.

For element x and set S, we express the membership relation as follows:

$$\boxed{x \in S}$$

Subset and Equality Relations for Sets

- A set S is said to be subset of set T when every element of S is also an element of T. This is written as follows:

$$\boxed{S \subseteq T}$$

- For example: $\{ 5, 8 \} \subseteq \{ 4, 5, 6, 7, 8 \}$

- A set S is said to be equal to set T when $S \subseteq T$ and $T \subseteq S$.

$$\boxed{S = T}$$

- For example: $\{ 5, 8, 3 \} = \{ 3, 5, 5, 8 \}$

Operations on sets

- Union of S and T: set of elements in either S or T:

$$S \cup T$$

- Intersection of S and T: set of elements in both S and T:

$$S \cap T$$

- Difference of S and T: set of elements in S but not in T:

$$S \setminus T$$

Example Set Expressions

$$
\begin{aligned}
\{a, b, c\} \cup \{b, d\} &= \{a, b, c, d\} \\
\{a, b, c\} \cap \{b, d\} &= \{b\} \\
\{a, b, c\} \setminus \{b, d\} &= \{a, c\}
\end{aligned}
$$

$$
\begin{aligned}
\{a, b, c\} \cap \{d, e, f\} &= \{\} \\
\{a, b, c\} \setminus \{d, e, f\} &= \{a, b, c\}
\end{aligned}
$$

context *BuildingContext*
sets *USER*
end

machine *Building*
variables *register* *in* *out*
invariants

`inv1:` $register \subseteq USER$ // set of registered users

`inv2:` $register = in \cup out$ // all registered users must be either inside or outside

`inv3:` $in \cap out = \{\}$ // no user can be inside and outside

Entering and Leaving the Building

initialisation $in,\ out,\ register\ :=\ \{\},\ \{\},\ \{\}$

events

$Enter\ \widehat{=}$

any s **where**

$s \in out$

then

$in := in \cup \{s\}$

$out := out \setminus \{s\}$

end

$Leave\ \widehat{=}$

any s **where**

$s \in in$

then

$in := in \setminus \{s\}$

$out := out \cup \{s\}$

end

Event-B *context*

- **Carrier Sets**: abstract types used in specification
- **Constants**: logical variables whose value remain constant
- **Axioms**: constraints on the constants. An axiom is a logical predicate.

Event-B *machine*

- **Sees:** one or more contexts
- **Variables**: state variables whose values can change
- **Invariants**: constraints on the variables that should always hold true. An invariant is a logical predicate.
- **Initialisation**: initial values for the abstract variables
- **Events**: guarded actions specifying ways in which the variables can change. Events may have parameters.

Adding New Users

New users cannot be registered already.

$$
\begin{array}{l}
\mathit{NewUser} \;\widehat{=}\; \\
\quad \textbf{any } s \textbf{ where} \\
\quad\quad s \in (\mathit{USER} \setminus \mathit{register}) \\
\quad \textbf{then} \\
\quad\quad \mathit{register} := \mathit{register} \cup \{s\} \\
\quad \textbf{end}
\end{array}
$$

What is the error in this specification?

Adding New Users – Correct Version

$$
\begin{array}{l}
\mathit{NewUser} \;\hat{=}\; \\
\quad \textbf{any } s \textbf{ where} \\
\quad\quad s \in (\mathit{USER} \setminus \mathit{register}) \\
\quad \textbf{then} \\
\quad\quad \mathit{register} := \mathit{register} \cup \{s\} \\
\quad\quad \mathit{out} := \mathit{out} \cup \{s\} \\
\quad \textbf{end}
\end{array}
$$

Newly registered users must be added either to *in* or *out* to preserve to *inv*2.

Rodin demo

- Animation with ProB
- Checking for invariant violations with ProB

Types in Event-B

- Predefined Types:
  $\mathbb{Z}$ Integers
  $\mathbb{B}$ Booleans $\{$ TRUE, FALSE $\}$

- Basic Types (or Carrier Sets):
  sets *WORD* *NAME*

Basic types are introduced to represent the entities of the problem being modelled.

Note: $\mathbb{N}$ is a subet of $\mathbb{Z}$ representing all non-negative integers (including 0).

Type for sets?

- $w \in WORD$ means that the type of w is $WORD$.
- $known \subseteq WORD$ - what is the type of $known$?

Powersets

The powerset of a set S is the set whose elements are all subsets of S:

$$\boxed{\mathbb{P}(S)}$$

Example

$$\mathbb{P}(\ \{a, b, c\}\ ) \quad = \quad \{\ \{\},\ \{a\},\ \{b\},\ \{c\},\ \{a, b\},\ \{a, c\},\ \{b, c\},\ \{a, b, c\}\ \}$$

Note $\quad S \in \mathbb{P}(T) \quad$ is the same as $\quad S \subseteq T$

Sets are themselves elements – so we can have sets of sets.
$\mathbb{P}(\ \{a, b, c\}\ )$ is an example of a set of sets.

Types of Sets

All the elements of a set must have the same type.

For example, $\{3, 4, 5\}$ is a set of integers.
More Precisely: $\{3, 4, 5\} \in \mathbb{P}(\mathbb{Z})$.
So the type of $\{3, 4, 5\}$ is $\mathbb{P}(\mathbb{Z})$

To declare x to be a set of elements of type T we write *either*

$$x \in \mathbb{P}(T) \qquad \text{or} \qquad x \subseteq T$$

- $known \subseteq WORD$ - so type of $known$ is $\mathbb{P}(WORD)$

Predicate Logic

Basic predicates: $\boxed{x \in S}$ $\boxed{S \subseteq T}$ $\boxed{x \leq y}$

Predicate operators:

- Negation: $\boxed{\neg P}$ P does not hold
- Conjunction: $\boxed{P \ \wedge \ Q}$ both P and Q hold
- Disjunction: $\boxed{P \ \vee \ Q}$ either P or Q holds
- Implication: $\boxed{P \implies Q}$ if P holds, then Q holds
- Universal Quantification: $\boxed{\forall x \cdot P}$ P holds for all x.
- Existential Quantification: $\boxed{\exists x \cdot P}$ P holds for some x.

Defining Set Operators with Logic

Predicate	Definition
$x \notin S$	$\neg\,(x \in S)$
$x \in S \cup T$	$x \in S \lor x \in T$
$x \in S \cap T$	$x \in S \land x \in T$
$x \in S \setminus T$	$x \in S \land x \notin T$
$S \subseteq T$	$\forall x \cdot x \in S \implies x \in T$