UNIVERSITY OF Southampton

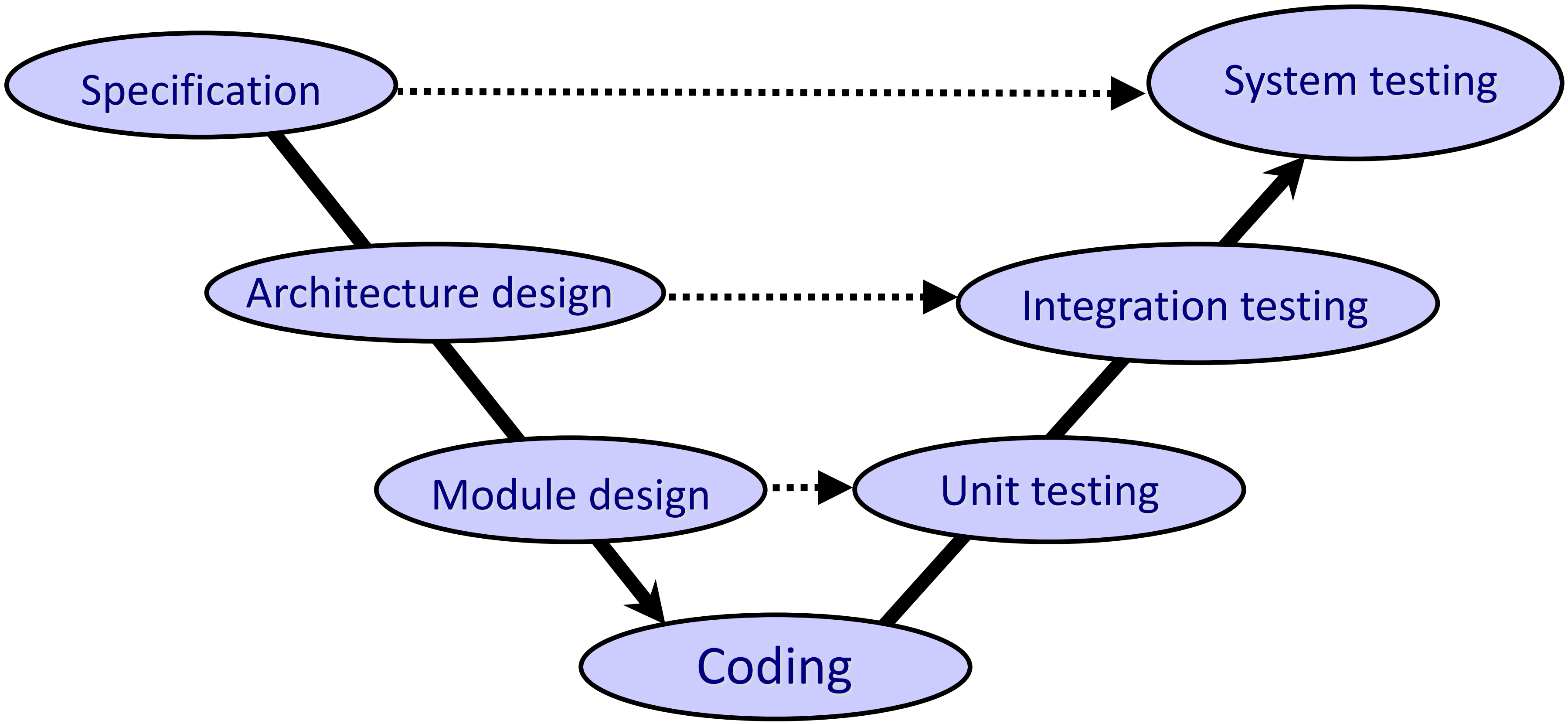
Introducing Event-B

Michael Butler

(edited by Thai Son Hoang)

25 February 2020

V model of software development



Defects discovered too late…

- *“Requirements and architecture defects make up approximately 70% of all system defects”*

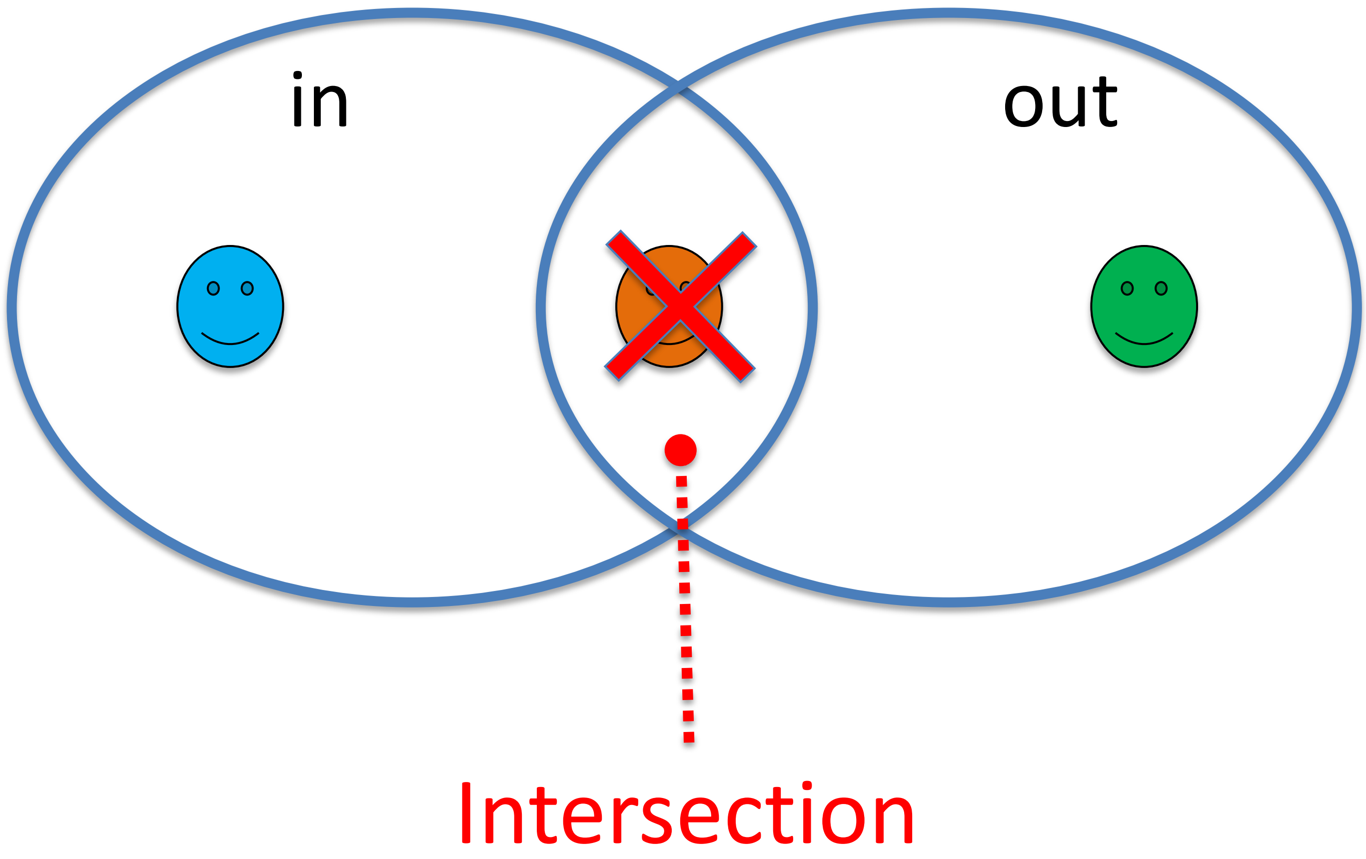
- *“80% of these defects are discovered late in the development life cycle”*

Four Pillars for Improving the Quality of Safety-Critical Software-Reliant Systems
Carnegie Mellon SEI, 2013
https://resources.sei.cmu.edu/asset_files/WhitePaper/2013_019_001_47803.pdf

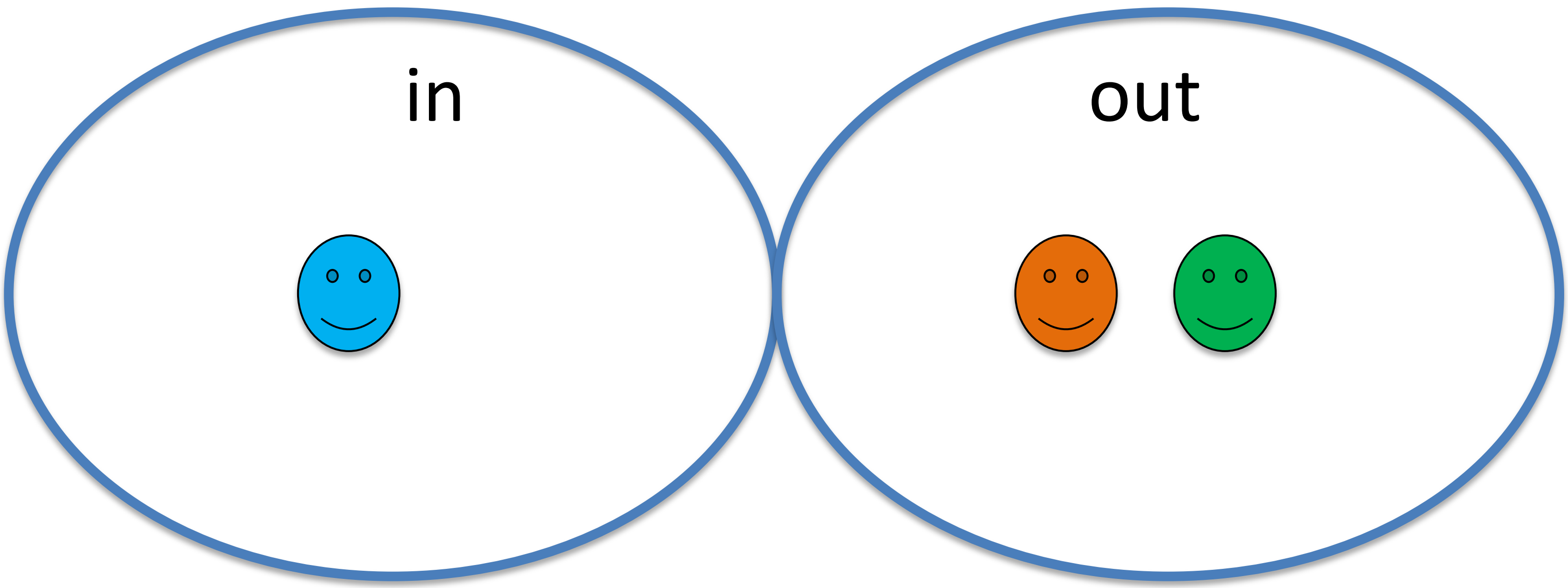
Example Requirements for a Building Control System

- 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

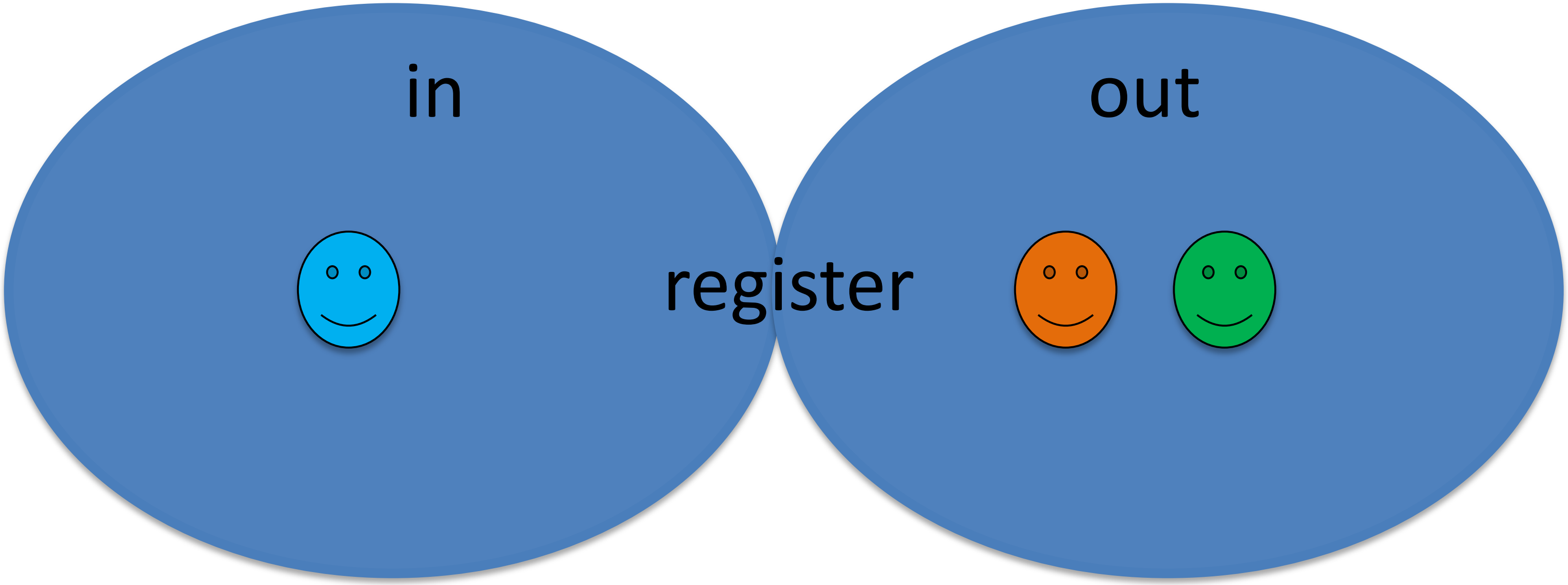


Disjoint sets



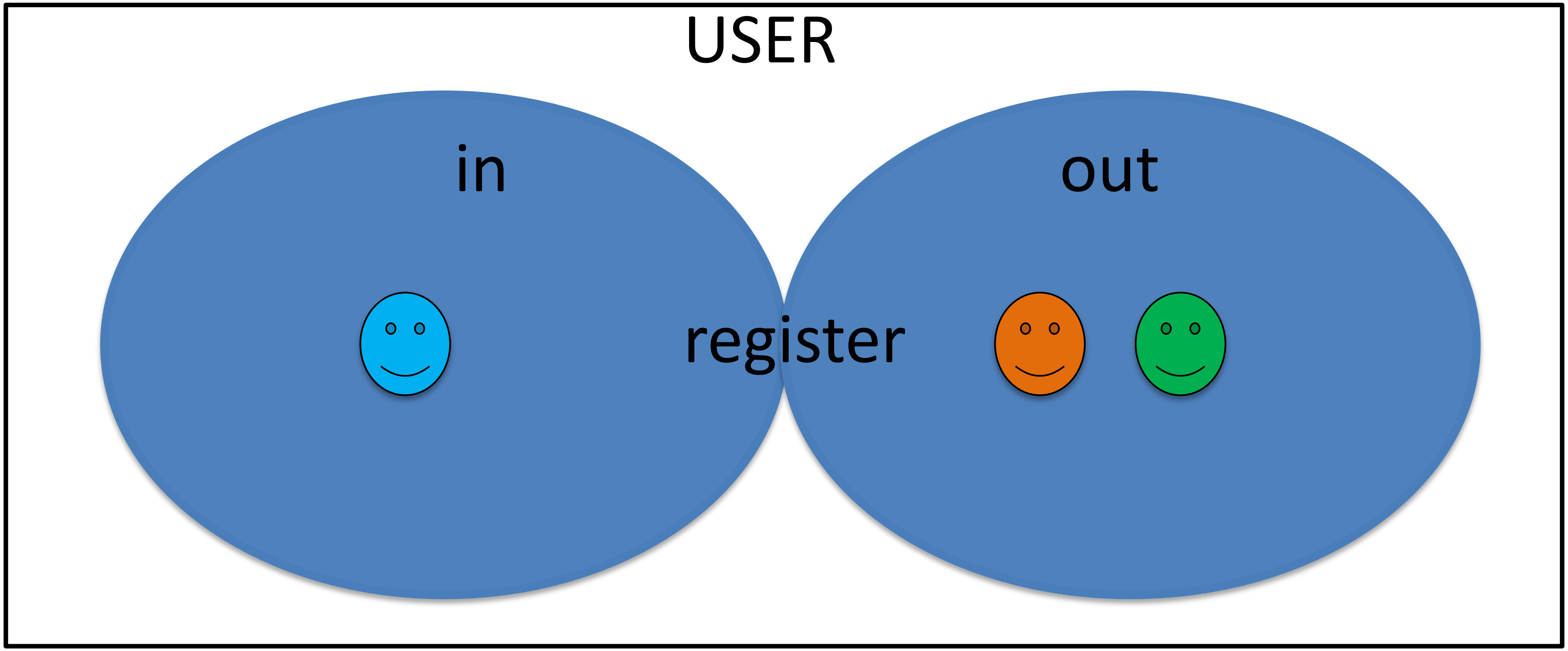
Invariant: $in \cap out = \{\}$

Registered users are either *in* or *out*



$$register \;=\; in \cup out$$

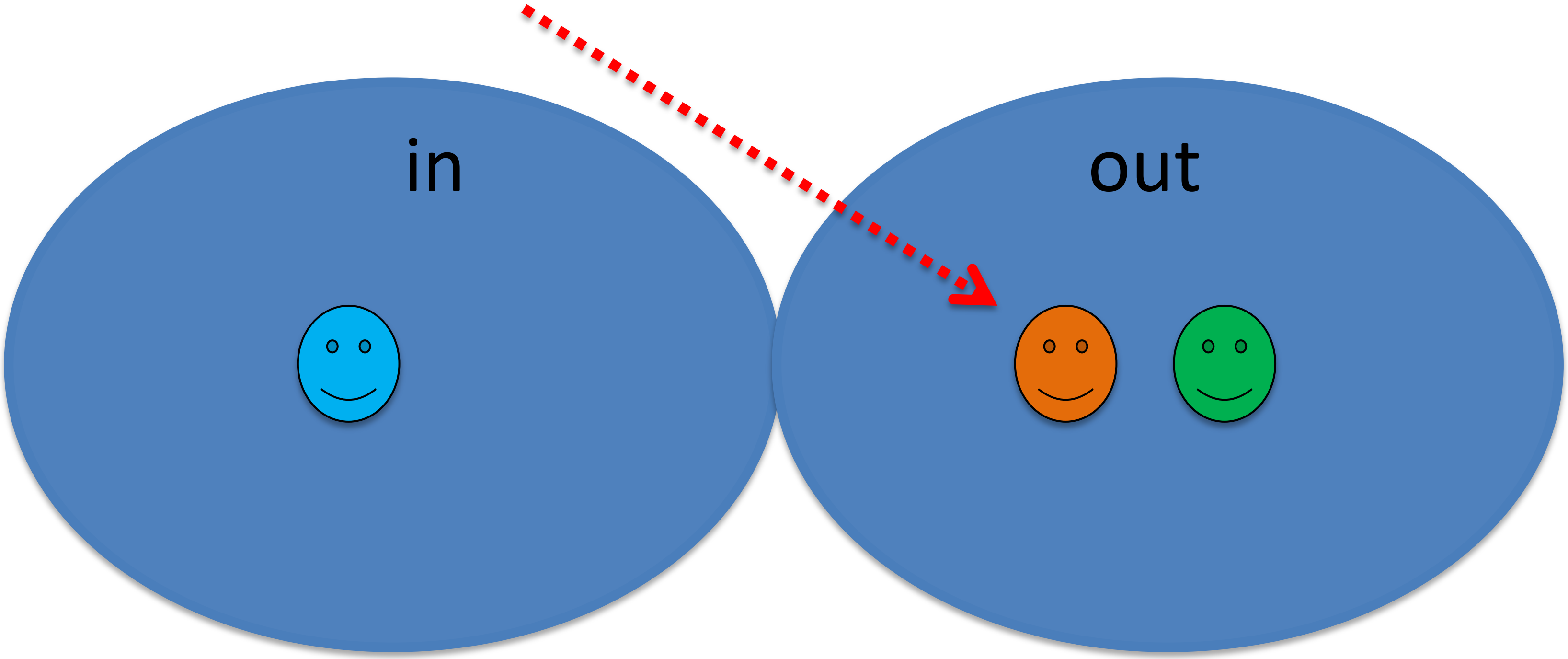
Carrier Set: type for users



$$register \subseteq USER$$

Event: user ***enters*** building

Guard: $s \in out$

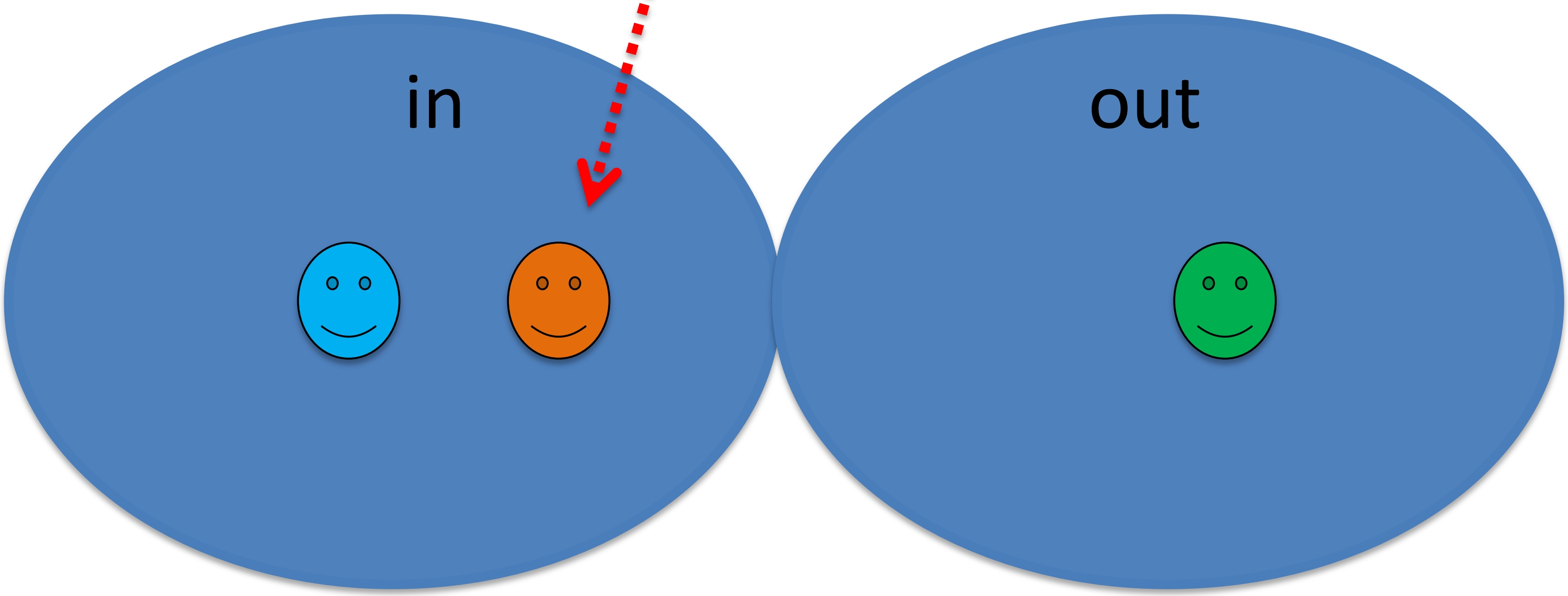


Action: $in := in \cup \{s\}$

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

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:

$$\boxed{S \cup T}$$

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

$$\boxed{S \cap T}$$

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

$$\boxed{S \setminus T}$$

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

$\texttt{inv1:}\quad register \subseteq USER$ // set of registered users

$\texttt{inv2:}\quad register = in \cup out$ // all registered users must be // either inside or outside

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

Entering and Leaving the Building

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

events

$Enter \ \hat{=}$
any s **where**
$s \in out$
then
$in := in \cup \{s\}$
$out := out \setminus \{s\}$
end

$Leave \ \hat{=}$
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.

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

What is the error in this specification?

Adding New Users

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

Vevox (**120-802-577**)!

1. The restriction on s is too much: s can be a registered user
2. We need to add s to the set of users inside the building
3. We need to add s to the set of users outside the building
4. We need ensure before hand that s is not inside the building

Adding New Users – Correct Version

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

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

Some formal methods

- VDM (Bjørner & Jones , 1970s)
  - IBM Vienna Labs: Vienna Development Method
  - Designed for defining programming languages
  - Extended to specify sequential programs
- Z Notation (Oxford group , 1980s)
  - Specification of software systems
  - Makes extensive use of set theory and logic
- B Method (Abrial , 1990s)
  - Evolved from Z, emphasis on tools (proof + code generation)
  - Mainly used in railway industry
- Alloy (Jackson, 1990s/2000s)
  - Focus on modelling and automated verification

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 reasoning about system behaviour
- 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,…)

Event-B in Software Development

- System specifications are derived from requirements
- System specification is an important precursor to programming and testing
- Event-B: formal language for writing high-level specifications of computer systems
- Event-B language includes logic and set theory
- Formal specification is more precise and consistent than an informal (natural language) specification.
- Event-B typically used in safety-critical or mission-critical applications.

Industrial uses of Event-B

- Event-B in Railway Interlocking
  - Alstom, Systerel
- Event-B in Smart Grids
  - Selex, Critical Software
- Other industrial users:
  - 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

Dictionary modelled with sets

Simple Example: Dictionary

context *DictionaryContext*
sets *WORD* // *WORD* is a basic type introduced for this model
end

machine *Dictionary*
variables *known*

invariants $known \subseteq WORD$ // set of known words

initialisation $known := \{\}$

Adding words to the Dictionary

events

$$
\begin{array}{l}
\mathit{AddWord} \;\; \widehat{=} \\
\quad \textbf{any} \; w \; \textbf{where} \\
\qquad w \in \mathit{WORD} \\
\quad \textbf{then} \\
\qquad \mathit{known} := \mathit{known} \cup \{w\} \\
\quad \textbf{end}
\end{array}
$$

This event has a parameter w representing the word that is added to the set of known words.

Checking if a word is in a dictionary: 2 cases

$CheckWordOK \triangleq$
any $w, result$ **where**
$w \in known$
$result = TRUE$
then
$skip$ // omit in Rodin
end

$CheckWordNotOK \triangleq$
any $w, result$ **where**
$w \notin known$
$result = FALSE$
then
$skip$ // omit
end

Cases are represented by separate events.

In both cases, *result* represents a result parameter.

Counting Dictionary

machine *CountingDictionary*
variables *known* *count*
invariants

$$known \subseteq WORD$$
$$count = card(known)$$

events

$$
\begin{array}{l}
AddWord \ \ \widehat{=} \\
\quad \textbf{any} \ w \ \textbf{where} \\
\quad \quad w \in WORD \\
\quad \textbf{then} \\
\quad \quad known := known \ \cup \ \{w\} \\
\quad \quad count := count + 1 \\
\quad \textbf{end}
\end{array}
$$

- Is this specification of *AddWord* correct?

Adding Words

$$
\begin{array}{l}
\mathit{AddWord} \;\; \widehat{=} \\
\quad \textbf{any} \;\; w \;\; \textbf{where} \\
\qquad w \in \mathit{WORD} \\
\quad \textbf{then} \\
\qquad \mathit{known} := \mathit{known} \;\cup\; \{w\} \\
\qquad \mathit{count} := \mathit{count} + 1 \\
\quad \textbf{end}
\end{array}
$$

Vevox (**120-802-577**)!

1. Yes, it is correct
2. No, w must be a known word
3. No, w must be an unknown word
4. No, we have to decrease count

Word deletion in Counting Dictionary

$$
\begin{array}{l}
\mathit{RemoveWord} \ \ \hat{=} \\
\quad \textbf{any}\ w\ \textbf{where} \\
\quad\quad w \in \mathit{WORD} \\
\quad \textbf{then} \\
\quad\quad \mathit{known} := \mathit{known} \setminus \{w\} \\
\quad\quad \mathit{count} := \mathit{count} - 1 \\
\quad \textbf{end}
\end{array}
$$

- Is this specification of *RemoveWord* correct?

Removing Words

$$
\begin{array}{l}
\mathit{RemoveWord} \;\hat{=} \\
\quad \textbf{any}\ w\ \textbf{where} \\
\qquad w \in \mathit{WORD} \\
\quad \textbf{then} \\
\qquad \mathit{known} := \mathit{known} \setminus \{w\} \\
\qquad \mathit{count} := \mathit{count} - 1 \\
\quad \textbf{end}
\end{array}
$$

Vevox (**120-802-577**)!

1. Yes, it is correct
2. No, w must be a known word
3. No, w must be an unknown word
4. No, we have to increase count

Correct versions of Add and Remove

$$
\begin{array}{l}
AddWord \;\; \triangleq \\
\quad \textbf{any } w \textbf{ where} \\
\quad\quad w \in WORD \setminus known \\
\quad \textbf{then} \\
\quad\quad known := known \cup \{w\} \\
\quad\quad count := count + 1 \\
\quad \textbf{end}
\end{array}
\qquad
\begin{array}{l}
RemoveWord \;\; \triangleq \\
\quad \textbf{any } w \textbf{ where} \\
\quad\quad w \in known \\
\quad \textbf{then} \\
\quad\quad known := known \setminus \{w\} \\
\quad\quad count := count - 1 \\
\quad \textbf{end}
\end{array}
$$

- Both of these events maintain the invariant $count = card(known)$ that links $count$ and $known$.

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

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

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

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 $\mathbb{P}(WORD)$

Classification of Types

Simple Types:

- $\mathbb{Z}$, $\mathbb{B}$
- Basic types (e.g., *WORD*, *NAME*)

Constructed Types:

- $\mathbb{P}(T)$

$\mathbb{P}(T)$ is a type that is constructed from T.

We will see more constructed types later.

Why Types?

- Types help to structure specifications by differentiating objects.
- Types help to prevent errors by not allowing us to write meaningless things.
- Types can be checked by computer.

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
$x \notin S$	$\neg\,(x \in S)$
$x \in S \cup T$	$x \in S \;\vee\; x \in T$
$x \in S \cap T$	$x \in S \;\wedge\; x \in T$
$x \in S \setminus T$	$x \in S \;\wedge\; x \notin T$
$S \subseteq T$	$\forall x \cdot x \in S \implies x \in T$

Event-B Lecture Notes

- For overview of modelling with sets in Event-B see Notes:
- http://eprints.soton.ac.uk/402239/
- (also linked from COMP1216 web page)

- Read Sections 1-5