Embedded Systems Specification and Design
Model-based Design and Verification

David Kendall
Introduction

- Functions
- CHOOSE
- IF ... THEN ... ELSE
- Tuples and sequences
- Strings
- Illustrative example — a simple scheduler
A fancier building control system

Informal statement of requirements:

- The system should monitor persons as they enter and leave buildings
- The system should maintain a register of known persons. It should be possible to remove a person from the register
- The system should maintain information about access permissions, i.e. which buildings any registered person is allowed to enter. It should be possible both to grant and remove access permissions
- A person can only enter a building if they are registered and have permission for entry to the building
- The system should keep track of the location of persons on the register

We’ll use this as a running example to introduce some more of the TLA\(^+\) language
Representing the state of the system

The informal specification mentions

- persons who should be monitored
- buildings that persons may enter and leave
- a register of users
- information about access permissions of registered users
- information about location of registered users

**CONSTANT**
- PERSON, set of persons
- BUILDING, set of buildings

**VARIABLE**
- register, set of registered users
- permission, access permissions of registered users
- location, location of registered users
Representing the state of the system

The informal specification mentions

- *persons* who should be monitored
Representing the state of the system

The informal specification mentions

- persons who should be monitored

\[
\text{CONSTANT} \quad \text{PERSON},
\]
set of persons
Representing the state of the system

The informal specification mentions

- *persons* who should be monitored
- *buildings* that persons may enter and leave

\[
\text{CONSTANT} \\
\text{PERSON,}
\quad \text{set of persons}
\]
Representing the state of the system

The informal specification mentions

- *persons* who should be monitored
- *buildings* that persons may enter and leave

```
CONSTANT
PERSON, BUILDING
```

set of persons

set of buildings
Representing the state of the system

The informal specification mentions

- *persons* who should be monitored
- *buildings* that persons may enter and leave
- a *register* of users

**CONSTANT**

- **PERSON**, set of persons
- **BUILDING**, set of buildings
Representing the state of the system

The informal specification mentions

- *persons* who should be monitored
- *buildings* that persons may enter and leave
- a *register* of users

\[
\begin{align*}
\text{CONSTANT} \\
\text{PERSON, BUILDING} \\
\text{set of persons, set of buildings} \\
\text{VARIABLE} \\
\text{register} \\
\text{set of registered users}
\end{align*}
\]
Representing the state of the system

The informal specification mentions

- **persons** who should be monitored
- **buildings** that persons may enter and leave
- a **register** of users
- information about **access permissions** of registered users

**CONSTANT**

- PERSON, BUILDING
  - set of persons
  - set of buildings

**VARIABLE**

- register
  - set of registered users
Representing the state of the system

The informal specification mentions

- persons who should be monitored
- buildings that persons may enter and leave
- a register of users
- information about access permissions of registered users

**CONSTANT**
- PERSON, set of persons
- BUILDING, set of buildings

**VARIABLE**
- register, set of registered users
- permission, access permissions of registered users
Representing the state of the system

The informal specification mentions

- persons who should be monitored
- buildings that persons may enter and leave
- a register of users
- information about access permissions of registered users
- information about location of registered users

**CONSTANT**

- PERSON, set of persons
- BUILDING, set of buildings

**VARIABLE**

- register, set of registered users
- permission, access permissions of registered users
Representing the state of the system

The informal specification mentions

- **persons** who should be monitored
- **buildings** that persons may enter and leave
- a **register** of users
- information about **access permissions** of registered users
- information about **location** of registered users

**CONSTANT**

- **PERSON**, set of persons
- **BUILDING**, set of buildings

**VARIABLE**

- **register**, set of registered users
- **permission**, access permissions of registered users
- **location**, location of registered users
Representing the state of the system

The informal specification mentions

- *persons* who should be monitored
- *buildings* that persons may enter and leave
- a *register* of users
- information about *access permissions* of registered users
- information about *location* of registered users

**CONSTANT**

- **PERSON**, set of persons
- **BUILDING**, set of buildings

**VARIABLE**

- **register**, set of registered users
- **permission**, access permissions of registered users
- **location**, location of registered users
How to represent the access permissions?

We have declared a variable, *permission*, to represent the access permissions of registered users

What kind of values should *permission* be able to take?
How to represent the access permissions?

We have declared a variable, *permission*, to represent the access permissions of registered users.

What kind of values should *permission* be able to take?

Suppose

\[
\begin{align*}
\text{PERSON} &= \{"p1", "p2", "p3", "p4"\} \\
\text{BUILDING} &= \{"b1", "b2", "b3"\}
\end{align*}
\]

give some examples of what *permission* might look like
How to represent the access permissions?

We have declared a variable, *permission*, to represent the access permissions of registered users.

What kind of values should *permission* be able to take?

Suppose

\[
\begin{align*}
register &= \{"p1", "p4"\} \\
BUILDING &= \{"b1", "b2", "b3"\}
\end{align*}
\]

give some examples of what *permission* might look like.
How to represent the access permissions?

We have declared a variable, *permission*, to represent the access permissions of registered users.

What kind of values should *permission* be able to take?

Suppose

\[
\text{register} = \{"p1", "p4"\} \\
\text{BUILDING} = \{"b1", "b2", "b3"\}
\]

give some examples of what *permission* might look like.

These are written in TLA$^+$ as *functions*.
What is a function in TLA⁺?

Programmers can think of functions as being like arrays, i.e. a mapping from some index set to some set of values.

Arrays in a programming language are often restricted to a very limited index set, e.g. in C an array of size $N$ has an index set that is the set of natural numbers from 0 to $N - 1$.

In TLA⁺ functions are primitive objects whose index set can be any set, even an infinite set, e.g.

$$f \triangleq [n \in 1..5 \mapsto n + 1]$$

defines a function, $f$, that maps every number $n$ in the set $\{1, 2, 3, 4, 5\}$ to its successor $n + 1$.

You can read $\mapsto$ as “maps to”. It is written as $\rightarrow$ in ASCII.
We can define a function $sqr$

$$sqr \triangleq [i \in Nat \mapsto i^2]$$

where $Nat$ is the infinite set of natural numbers

TLA$^+$ provides an alternative syntax for this kind of function definition, e.g.

$$sqr2[i \in Nat] \triangleq i^2$$

Function application is written using square brackets, e.g. $sqr[3]$, which is equal to 9

A function is only defined for elements in its index set. In TLA$^+$ the index set of a function is called its DOMAIN, e.g. $\text{DOMAIN } f$ is \{1, 2, 3, 4, 5\} and $\text{DOMAIN } sqr$ is $Nat$
Two functions \( f \) and \( g \) are equal iff they have the same domain and \( f[x] = g[x] \) for all \( x \) in their domain, e.g. \( sqr = sqr2 \).

The *range* of a function \( f \) is just the set of values \( f[x] \) for all \( x \) in the domain of \( f \). It can be defined as follows in TLA\(^+\):

\[
\text{Range}(f) \equiv \{ f[x] : x \in \text{DOMAIN } f \} 
\]

\([S \rightarrow T]\)

This expression denotes the set of all functions \( f \) whose domain is \( S \) and whose range is any subset of \( T \).

**Exercise** Let \( A = 1..3 \) and \( B = \{ "a", "b" \} \). Calculate the set \([B \rightarrow A]\). Use TLC to check your answer.

The empty function, whose domain is the empty set, is written as \( \langle \rangle \) (\( << >\) in ASCII).  

TLA\(^+\) functions (ctd)
Revisiting the state of the system

CONSTANT
PERSON,
BUILDING

VARIABLE
register
permission
location

TypeOk
\hat{=}
\land register \subseteq PERSON \land permission \in [register \rightarrow SUBSET BUILDING] \land location \in [register \rightarrow (BUILDING \cup \{OUTSIDE\})]
Revisiting the state of the system

CONSTANT
PERSON, BUILDING

VARIABLE
register permission location

How should we define $\textit{TypeOk}$?

$\textit{TypeOk} \triangleq$
Revisiting the state of the system

CONSTANT
    PERSON,
    BUILDING

VARIABLE
    register
    permission
    location

How should we define TypeOk?

\[ TypeOk \equiv \land \text{register} \subseteq \text{PERSON} \]
Revisiting the state of the system

CONSTANT
    PERSON,
    BUILDING

VARIABLE
    register
    permission
    location

How should we define \( \text{TypeOk} \)?

\[
\text{TypeOk} \equiv \\
\quad \land \text{register} \subseteq \text{PERSON} \\
\quad \land \text{permission} \in [\text{register} \rightarrow \text{SUBSET BUILDING}]
\]
Revisiting the state of the system

CONSTANT
   PERSON,
   BUILDING

VARIABLE
   register
   permission
   location

How should we define \textit{TypeOk}? 

\textit{TypeOk} \iff
\begin{align*}
\land \ register \subseteq PERSON \\
\land \ permission \in [register \rightarrow \text{SUBSET BUILDING}] \\
\land \ location \in [register \rightarrow \text{BUILDING}]
\end{align*}
Revisiting the state of the system

CONSTANT
    PERSON,
    BUILDING

OUTSIDE ≜ CHOOSE x : x ∉ PERSON ∪ BUILDING

VARIABLE
    register
    permission
    location

How should we define $TypeOk$?

$TypeOk ≜$

\[ \land \text{register} \subseteq PERSON \]
\[ \land \text{permission} \in \text{[register} \rightarrow \text{SUBSET BUILDING]} \]
\[ \land \text{location} \in \text{[register} \rightarrow \text{BUILDING]} \]
Revisiting the state of the system

\[
\text{CONSTANT}
\begin{align*}
\text{PERSON}, \\
\text{BUILDING}
\end{align*}
\]

\[
\text{OUTSIDE} \triangleq \text{CHOOSE } x : x \notin \text{PERSON} \cup \text{BUILDING}
\]

\[
\text{VARIABLE}
\begin{align*}
\text{register} \\
\text{permission} \\
\text{location}
\end{align*}
\]

How should we define \textit{TypeOk}?

\[
\text{TypeOk} \triangleq \begin{align*}
\wedge \text{register} & \subseteq \text{PERSON} \\
\wedge \text{permission} & \in [\text{register} \to \text{SUBSET BUILDING}] \\
\wedge \text{location} & \in [\text{register} \to (\text{BUILDING} \cup \{\text{OUTSIDE}\})]
\end{align*}
\]
Revisiting the state of the system

CONSTANT
  PERSON, BUILDING

OUTSIDE ≡ CHOOSE x : x ∉ PERSON ∪ BUILDING

VARIABLE
  register
  permission
  location

How should we define TypeOk?

TypeOk ≡
  ∧ register ⊆ PERSON
  ∧ permission ∈ [register → SUBSET BUILDING]
  ∧ location ∈ [register → (BUILDING ∪ {OUTSIDE})]
The **CHOOSE** operator

The CHOOSE operator is related to the existential quantifier $\exists$

The formula $\exists x \in S : P(x)$ asserts that there is a value $x$ for which $P(x)$ is TRUE. If that is the case, then CHOOSE $x \in S : P(x)$ equals such a value.

We can use CHOOSE to select the unique value that satisfies the specified property.

**Example:** We can define the maximum of a set of integers like this

$$\text{Max}(S) \equiv \text{CHOOSE } x \in S : \forall y \in S : x \geq y$$

CHOOSE satisfies the following property

$$\exists x \in S : P(x) \Rightarrow \bigwedge (\text{CHOOSE } x \in S : P(x)) \in S$$

$$\bigwedge P(\text{CHOOSE } x \in S : P(x))$$

If there is no element $x$ in $S$ satisfying $P(x)$, then we don’t know anything about the value CHOOSE $x \in S : P(x)$.
The Enter operation

Assume $p \in \text{PERSON}$ and $b \in \text{BUILDING}$

How can we model an action which changes the state so that person $p$ enters building $b$?

$\text{Enter}(p, b) \hat{=} \\land p \in \text{register} \land b \in \text{permission}[p] \land \text{location}' = [\text{location} \setminus \text{EXCEPT } p] = b \land \text{UNCHANGED} \langle \text{register}, \text{permission} \rangle$
The Enter operation

Assume $p \in \text{PERSON}$ and $b \in \text{BUILDING}$
The Enter operation

Assume $p \in PERSON$ and $b \in BUILDING$

How can we model an action which changes the state so that person $p$ enters building $b$?
The \textbf{Enter} operation

Assume $p \in \text{PERSON}$ and $b \in \text{BUILDING}$

How can we model an action which changes the state so that person $p$ enters building $b$?

$\text{Enter}(p, b) \triangleq$
The Enter operation

Assume $p \in PERSON$ and $b \in BUILDING$

How can we model an action which changes the state so that person $p$ enters building $b$?

$$
\text{Enter}(p, b) \equiv \wedge p \in \text{register}
$$
The **Enter** operation

Assume $p \in PERSON$ and $b \in BUILDING$

How can we model an action which changes the state so that person $p$ enters building $b$?

$$\text{Enter}(p, b) \equiv \\
\land p \in \text{register} \\
\land b \in \text{permission}[p]$$
The **Enter** operation

Assume $p \in PERSON$ and $b \in BUILDING$

How can we model an action which changes the state so that person $p$ enters building $b$?

$$Enter(p, b) \equiv$$

\[
\land p \in register \\
\land b \in permission[p] \\
\land location'[p] = b
\]
The **Enter** operation

Assume $p \in PERSON$ and $b \in BUILDING$

How can we model an action which changes the state so that person $p$ enters building $b$?

\[
\text{Enter}(p, b) \equiv \\
\wedge p \in \text{register} \\
\wedge b \in \text{permission}[p] \\
\wedge \text{location}' = [\text{location} \text{ EXCEPT ![p] = b}]
\]
The **Enter** operation

Assume \( p \in PERSON \) and \( b \in BUILDING \)

How can we model an action which changes the state so that person \( p \) enters building \( b \)?

\[
\text{Enter}(p, b) \equiv \\
\land p \in \text{register} \\
\land b \in \text{permission}[p] \\
\land location' = [\text{location EXCEPT } ![p] = b] \\
\land \text{register'} = \text{register} \land \text{permission'} = \text{permission}
\]
The Enter operation

Assume $p \in PERSON$ and $b \in BUILDING$

How can we model an action which changes the state so that person $p$ enters building $b$?

$$
\text{Enter}(p, b) \triangleq \\
\land p \in \text{register} \\
\land b \in \text{permission}[p] \\
\land \text{location}' = [\text{location} \ \text{EXCEPT} \ ![p] = b] \\
\land \text{UNCHANGED} \langle \text{register}, \text{permission} \rangle
$$
The **Enter** operation

Assume \( p \in PERSON \) and \( b \in BUILDING \)

How can we model an action which changes the state so that person \( p \) enters building \( b \)?

\[
Enter(p, b) \equiv \\
\land p \in register \\
\land b \in permission[p] \\
\land location' = [location \ EXCEPT \ ![p] = b] \\
\land UNCHANGED \langle register, permission \rangle
\]
The **EXCEPT** construct

Programmers may think that it should be possible to update \( location \) in the \( Enter \) operation by writing

\[
\text{location}'[p] = b
\]

Why is this wrong?

Because it only specifies the new value of \( location[p] \). It says nothing about the value of any of the other elements in the domain of the \( location \) function.

It doesn’t even require that the new \( location \) function have the same domain.

We could specify the new value completely like this

\[
\text{location}' = \left[ x \in \text{DOMAIN} \text{location} \mapsto \begin{array}{ll}
\text{IF } x = p \text{ THEN } b \text{ ELSE } \text{location}[x]\end{array} \right]
\]

\( \text{TLA}^+ \) allows us to shorten this long-winded formulation using EXCEPT,

\[
\text{location}' = [\text{location EXCEPT } ![p] = b]
\]
The value of $\text{IF } p \text{ THEN } e_1 \text{ ELSE } e_2$ is $e_1$ if $p$ is $\text{TRUE}$ and $e_2$ if $p$ is $\text{FALSE}$

IF expressions can be nested, e.g.

$$\text{IF } i > 0 \text{ THEN } 1 \text{ ELSE } (\text{IF } i = 0 \text{ THEN } 0 \text{ ELSE } -1)$$

or

$$\text{IF } i \geq 0 \text{ THEN } (\text{IF } i = 0 \text{ THEN } 0 \text{ ELSE } 1) \text{ ELSE } -1$$
The **Leave** operation

**Exercise:** Take to 2 to 3 minutes to write down a specification of the *Leave* operation, by which it is recorded that person $p$ leaves building $b$:

\[
\text{Leave}(p, b) \triangleq p \in \text{register} \land \text{location}[p] = b \land \text{location}' = \text{EXCEPT}!\text{location} = \text{OUTSIDE} \land \text{UNCHANGED} \langle \text{register}, \text{permission} \rangle
\]
The **Leave** operation

**Exercise:** Take to 2 to 3 minutes to write down a specification of the *Leave* operation, by which it is recorded that person *p* leaves building *b*

\[
\text{Leave}(p, b) \equiv \\
\quad \land p \in \text{register} \\
\quad \land \text{location}[p] = b \\
\quad \land \text{location}' = [\text{location EXCEPT } ![p] = \text{OUTSIDE}] \\
\quad \land \text{UNCHANGED} \langle \text{register, permission} \rangle
\]
The **Leave** operation

**Exercise:** Take to 2 to 3 minutes to write down a specification of the *Leave* operation, by which it is recorded that person *p* leaves building *b*

\[
\text{Leave}(p, b) \equiv \]
\[
\forall p \in \text{register} \\
\forall \text{location}[p] = b \\
\forall \text{location}' = [\text{location EXCEPT ![p] = OUTSIDE}] \\
\forall \text{UNCHANGED} \langle \text{register, permission} \rangle
\]
The **Register** operation

\[
Register(p) \equiv
\]

In this case, the \textit{permission} and \textit{location} functions need to be extended to include \( p \) in their domains. This prevents us from using a formulation with \textit{EXCEPT} and we must use the more verbose form to ensure that the domains of the new functions are specified and mappings determined for every element.
The Register operation

\[
\text{Register}(p) \triangleq \quad \\ \\
\wedge p \quad \in \quad \text{PERSON} \setminus \text{register}
\]
The **Register** operation

\[
\text{Register}(p) \triangleq \\
\land p \in \text{PERSON} \setminus \text{register} \\
\land \text{register}' = \text{register} \cup \{p\}
\]
The **Register** operation

\[
\text{Register}(p) \triangleq \\
\land p \in PERSON \setminus \text{register} \\
\land \text{register}' = \text{register} \cup \{p\} \\
\land \text{permission}' = [x \in \text{DOMAIN} \text{ permission} \cup \{p\} \mapsto \\
\hspace{1cm} \text{IF } x \in \text{DOMAIN} \text{ permission} \\
\hspace{1cm} \text{THEN } \text{permission}[x] \\
\hspace{1cm} \text{ELSE } \{\}]
\]
The **Register** operation

\[ \text{Register}(p) \triangleq \]

\[ \land p \in PERSON \setminus \text{register} \]
\[ \land \text{register}' = \text{register} \cup \{p\} \]
\[ \land \text{permission}' = [x \in \text{DOMAIN permission} \cup \{p\} \mapsto \\
\quad \begin{cases} 
\text{IF } x \in \text{DOMAIN permission} \\
\text{THEN } \text{permission}[x] \\
\text{ELSE } \{\} 
\end{cases}] 
\]
\[ \land \text{location}' = [x \in \text{DOMAIN location} \cup \{p\} \mapsto \\
\quad \begin{cases} 
\text{IF } x \in \text{DOMAIN location} \\
\text{THEN } \text{location}[x] \\
\text{ELSE OUTSIDE} 
\end{cases}] 
\]

In this case, the *permission* and *location* functions need to be extended to include \( p \) in their domains. This prevents us from using a formulation with EXCEPT and we must use the more verbose form to ensure that the domains of the new functions are specified and mappings determined for every element.
The **Register** operation

\[
\text{Register}(p) \equiv \\
\begin{align*}
\land p & \in PERSON \setminus \text{register} \\
\land \text{register}' & = \text{register} \cup \{p\} \\
\land \text{permission}' & = [x \in \text{DOMAIN permission} \cup \{p\} \mapsto \begin{cases} 
\text{IF } x \in \text{DOMAIN permission} & \text{THEN permission}[x] \\
\text{ELSE } \{\} & \end{cases}] \\
\land \text{location}' & = [x \in \text{DOMAIN location} \cup \{p\} \mapsto \begin{cases} 
\text{IF } x \in \text{DOMAIN location} & \text{THEN location}[x] \\
\text{ELSE OUTSIDE} & \end{cases}]
\end{align*}
\]

In this case, the **permission** and **location** functions need to be extended to include \( p \) in their domains. This prevents us from using a formulation with **EXCEPT** and we must use the more verbose form to ensure that the domains of the new functions are specified and mappings determined for every element.
The **Register** operation

\[
\text{Register}(p) \triangleq
\begin{align*}
\land p &\in \text{PERSON} \setminus \text{register} \\
\land \text{register}' &= \text{register} \cup \{p\} \\
\land \text{permission}' &= [x \in \text{DOMAIN} \text{ permission} \cup \{p\} \rightarrow \\
&\quad \text{IF } x \in \text{DOMAIN} \text{ permission} \\
&\quad \text{THEN } \text{permission}[x] \\
&\quad \text{ELSE } \{\}\] \\
\land \text{location}' &= [x \in \text{DOMAIN} \text{ location} \cup \{p\} \rightarrow \\
&\quad \text{IF } x \in \text{DOMAIN} \text{ location} \\
&\quad \text{THEN } \text{location}[x] \\
&\quad \text{ELSE } \text{OUTSIDE}] 
\end{align*}
\]

In this case, the **permission** and **location** functions need to be extended to include \(p\) in their domains. This prevents us from using a formulation with EXCEPT and we must use the more verbose form to ensure that the domains of the new functions are specified and mappings determined for every element.
The **Next** operation

\[
\text{Next } \equiv \\
\exists \ p \in \text{PERSON}, \ b \in \text{BUILDING} : \\
\begin{align*}
& \lor \text{Register}(p) \\
& \lor \text{DeRegister}(p) \\
& \lor \text{AddPermission}(p, b) \\
& \lor \text{RevokePermission}(p, b) \\
& \lor \text{Enter}(p, b) \\
& \lor \text{Leave}(p, b)
\end{align*}
\]

The **AddPermission**, **RevokePermission**, and **DeRegister** operations are left as exercises.