Software Requirements, Specifications and Formal Methods

Dr. Shixun Huang

UNIVERSITY
OF WOLLONGONG
AUSTRALIA

Objectives

- To introduce an approach to formal specification based on mathematical system models
- To present some features of the Z specification language
- The illustrate the use of Z using small examples
- To show how Z schemas may be used to develop incremental specifications
- To show the process from informal specification to formal specification

Model-based specification

- Defines a model of a system using well-understood mathematical entities such as sets and functions
- The state of the system is not hidden (unlike algebraic specification)
- State changes are straightforward to define
- Z is one of the most widely used model-based specification languages

Z as a specification language

- Based on typed set theory
- Probably now the most widely-used specification language
- Includes schemas, an effective low-level structuring facility
- Schemas are specification building blocks
- Graphical presentation of schemas make Z specifications easier to understand

A first example in Z

```
int iroot(int a)
/* Integer square root */
{
  int i,term,sum;

  term=1; sum=1;
  for (i=0; sum <= a; i++)
      term=term+2;
      sum=sum+term;
  }
  return i;
}
```

Computer the integer square root

A first example in Z

$$
\begin{array}{|l}
iroot : \mathbb{N} \to \mathbb{N} \\
\hline
\forall a : \mathbb{N} \bullet \\
\qquad iroot(a) * iroot(a) \leq a < (iroot(a) + 1) * (iroot(a) + 1)
\end{array}
$$

- N stands for natural number, 0, 1, 2,…, so both the input and output must be a natural number
- It explains what happens when the argument does not have the perfect integer square root, i.e., return the largest integer square root
- It does not explain how to do it
- Z is not a programming language

Z schemas

- Introduce specification entities and defines invariant predicates over these entities

- A schema includes
  - A name identifying the schema
  - A signature introducing entities and their types
  - A predicate part defining invariants over these entities

- Schemas can be included in other schemas and may act as type definitions

- Names are local to schemas

Introducing schemas: text editor example

Model a simple text editor

- The text editor can only deal with texts composed of characters
- The text editor has the size limit, i.e., *65535* characters
- A document can be modelled as two texts and the cursor, i.e., *left* indicates the text before the cursor, and *right* indicate the text following the cursor
- Users can insert new character in the document to the left of the cursor
- Users can move the cursor forward or backward without changing the existing characters in the document
- The cursor locates the end of file (EOF) when there is no character after the cursor

Basic types and abbreviation definition

- We declare a *basic type*: the set of all characters first.
- Then we define an *abbreviation definition* for a text: a sequence of characters.

$$[CHAR]$$

$$TEXT == \mathrm{seq}\, CHAR$$

- CHAR is the character set, no need to specify the details, might be ASCII
- CHAR is a full-fledged Z data type.
- In Z, a new data type can be introduced by writing its name inside brackets
- In Z, *seq* is used to defined a sequence of any type.
- *==* indicates *TEXT* is an abbreviation for *seq CHAR*

UNIVERSITY
OF WOLLONGONG
AUSTRALIA

Axiomatic descriptions

In Z, we use axiomatic descriptions to define constants

$$
\begin{array}{|l}
maxsize : \mathbb{N} \\
\hline
maxsize \leq 65535
\end{array}
$$

- *maxsize* is a nonnegative integer
- *maxsize* can be any number up to 65,535
- *maxsize* still a constant, but we don’t have to specify its value
- Constants declared in are global

State schema

In Z, state schemas indicate the states of the system, i.e., collections of state variables and their values. State variables are also called components.

left and *right* are the two state variables for our editor.

Name → *Editor*

$$\begin{array}{|l}
\textit{left}, \textit{right} : \textit{TEXT} \\
\hline
\#(\textit{left} \frown \textit{right}) \leq \textit{maxsize}
\end{array}$$

Signature: *left*, *right* : *TEXT*

Predicate: $\#(\textit{left} \frown \textit{right}) \leq \textit{maxsize}$

- The variables declared inside the schema box are *local*.
- *left*, *right* are TEXT
- The document can hold no more than *maxsize* characters.
- $\textit{left} \frown \textit{right}$ construct the whole text from *left* and *right*
- \# is the size operator

Initialization schema

Every system must have a start up state. In Z, this state is described by a schema conventionally named *Init*.

$$
\begin{array}{|l}
\textbf{\textit{Init}} \\
\hline
\textit{Editor} \\
\hline
\textit{left} = \textit{right} = \langle\rangle \\
\hline
\end{array}
$$

- *Init* schema includes *Editor* schema
- All the declarations and predicates in *Editor* schema apply to *Init* schema as well, so *Init* schema can use the local state variables *left* and *right* from *Editor* schema
- < > is the empty sequence

Operation schema

In Z, we define operation schemas to indicate how the system's states are changed. For example, the user can insert characters into our editor.

The *Insert* schema below defines how a single character is inserted in the document to the left of the cursor.

$$| \; printing : \mathbb{P}\, CHAR$$

Insert

$$\Delta Editor$$
$$ch? : CHAR$$

$$ch? \in printing$$
$$left' = left \frown \langle ch? \rangle$$
$$right' = right$$

Operation schema

- A schema name prefixed by the Greek letter Delta (Δ) means that the operation changes some or all of the state variables.
- ? Indicates the input of the operation schema
- The predicate tells use how the editor's state changes.
- Unprimed variables *left* and *right* denote the texts before the insert
- Primed variables *left'* and *right'* denote the texts after the insert.
- $ch? \in printing$ is the precondition, and it must be true before the insert operation can occur.
- The rest of the predicate is a postconditions: it describes the state of the editor after the operation.
- $left' = left \frown \langle ch? \rangle$ tells the new character is appended to the end of text preceding the cursor
- $right' = right$ says the text following the cursor does not change. In Z, we also must specify unchanged things.

Forward operation

This schema defines the operation by pressing the right arrow key on the keyboard.

$$
\begin{array}{|l}
right\_arrow : CHAR \\
\hline
right\_arrow \notin printing
\end{array}
$$

Forward

$$
\begin{array}{|l}
\Delta Editor \\
ch? : CHAR \\
\hline
ch? = right\_arrow \\
left' = left \frown \langle head(right) \rangle \\
right' = tail(right)
\end{array}
$$

Forward

$$
\begin{array}{|l}
\Delta Editor \\
ch? : CHAR \\
\hline
ch? = right\_arrow \\
right \neq \langle\rangle \\
left' = left \frown \langle head(right) \rangle \\
right' = tail(right)
\end{array}
$$

- *head()* returns the first element of a sequence
- *tail()* returns remaining sequence without the first element

Operation specification

- Operations may be specified incrementally as separate schema then the schema combined to produce the complete specification
- Define the ‘normal’ operation as a schema
- Define schemas for exceptional situations
- Combine all schemas using the disjunction (or) operator

Schema calculus

The editor can't crash, and it must be robust. We should define a total version of forward operation. We can define it in pieces, where each piece is a schema. Then we will use the schema calculus to put them together. This is a common way in Z to define complex operations.

EOF

$Editor$

$right = \langle \rangle$

RightArrow

$ch? : CHAR$

$ch? = right\_arrow$

$$T\_Forward \mathrel{\widehat{=}} Forward \vee (EOF \wedge RightArrow \wedge \Xi Editor)$$

- A schema name prefixed by the Greek letter Xi (Ξ) means that the state of the schema will not be changed

$[CHAR]$

$TEXT == \mathrm{seq}\, CHAR$

$$
\begin{array}{|l}
maxsize : \mathbb{N} \\
\hline
maxsize \leq 65535
\end{array}
$$

$$
\begin{array}{|l}
\textit{Editor} \\
\hline
left, right : TEXT \\
\hline
\#(left \frown right) \leq maxsize \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\textit{Init} \\
\hline
Editor \\
\hline
left = right = \langle\rangle \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
printing : \mathbb{P}\, CHAR
\end{array}
$$

$$
\begin{array}{|l}
\textit{Insert} \\
\hline
\Delta Editor \\
ch? : CHAR \\
\hline
ch? \in printing \\
left' = left \frown \langle ch? \rangle \\
right' = right \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
right\_arrow : CHAR \\
\hline
right\_arrow \notin printing
\end{array}
$$

$$
\begin{array}{|l}
\textit{Forward} \\
\hline
\Delta Editor \\
ch? : CHAR \\
\hline
ch? = right\_arrow \\
right \neq \langle\rangle \\
left' = left \frown \langle head(right) \rangle \\
right' = tail(right) \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\textit{EOF} \\
\hline
Editor \\
\hline
right = \langle\rangle \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\textit{RightArrow} \\
\hline
ch? : CHAR \\
\hline
ch? = right\_arrow \\
\hline
\end{array}
$$

$$T\_Forward \mathrel{\widehat{=}} Forward \vee (EOF \wedge RightArrow \wedge \Xi Editor)$$

Another Example: A Water Tank

- A water tank contains a container and indicator;
- The container contains water, and the indicator reads and indicates the water level;
- The container has limits in both maximal and minimal water level;
- If the water level drops below the danger level, the indicator's light will turn on;
- When the indicator's light turning on, we shall add water to the container. However, the total water in the container shall no more than the capacity of the container;
- When the water level is above the danger level, the indicator's light turns off;

Container schema: State Schema

Schema name

Schema signature

Schema predicate

Container

contents: $\mathbb{N}$

capacity: $\mathbb{N}$

---

contents $\leq$ capacity

Indicator schema: State Schema

Indicator

light:{off, on}
reading: $\mathbb{N}$
danger_level: $\mathbb{N}$

---

light = on $\Leftrightarrow$ reading $\leq$ danger_ level

Storage tank schema: Init Schema

```
┌─ Storage_tank ──────────────────
│
│ Container
│ Indicator
├─────────────────────────────────
│ reading = contents
│ capacity = 5000
│ danger_level = 50
└─────────────────────────────────
```

Full specification of storage tank schema

Storage_tank

contents: $\mathbb{N}$
capacity: $\mathbb{N}$
reading: $\mathbb{N}$
danger_level: $\mathbb{N}$
light: {off, on}

---

contents $\leq$ capacity
light = on $\Leftrightarrow$ reading $\leq$ danger_level
reading = contents
capacity = 5000
danger_level =50

A partial spec. of a fill Operation

Fill-OK

ΔStorage_tank
amount?: $\mathbb{N}$

---

contents + amount? $\leq$ capacity
contents' = contents + amount?

Storage tank fill operation

OverFill

Ξ Stora ge-tank
amount?: $\mathbb{N}$
r!: seq CHAR

capacity < contents + amount?
r! = “Insufficient tank capacity – Fill cancelled”

Fill

Fill-OK $\vee$ OverFill

Data dictionary modelling

- A data dictionary may be thought of as a mapping from a name (the key) to a value (the description in the dictionary)
- Operations are
  - Add: makes a new entry in the dictionary or replaces an existing entry
  - Lookup: given a name, returns the description.
  - Delete: deletes an entry from the dictionary
  - Replace: replaces the information associated with an entry

Data dictionary entry: state schema

DataDictionaryEntry

entry: NAME
desc: seq char
type: Sem_model_types
creation_date: DATE

$\#desc \leq 2000$

Data dictionary: state schema

DataDictionary

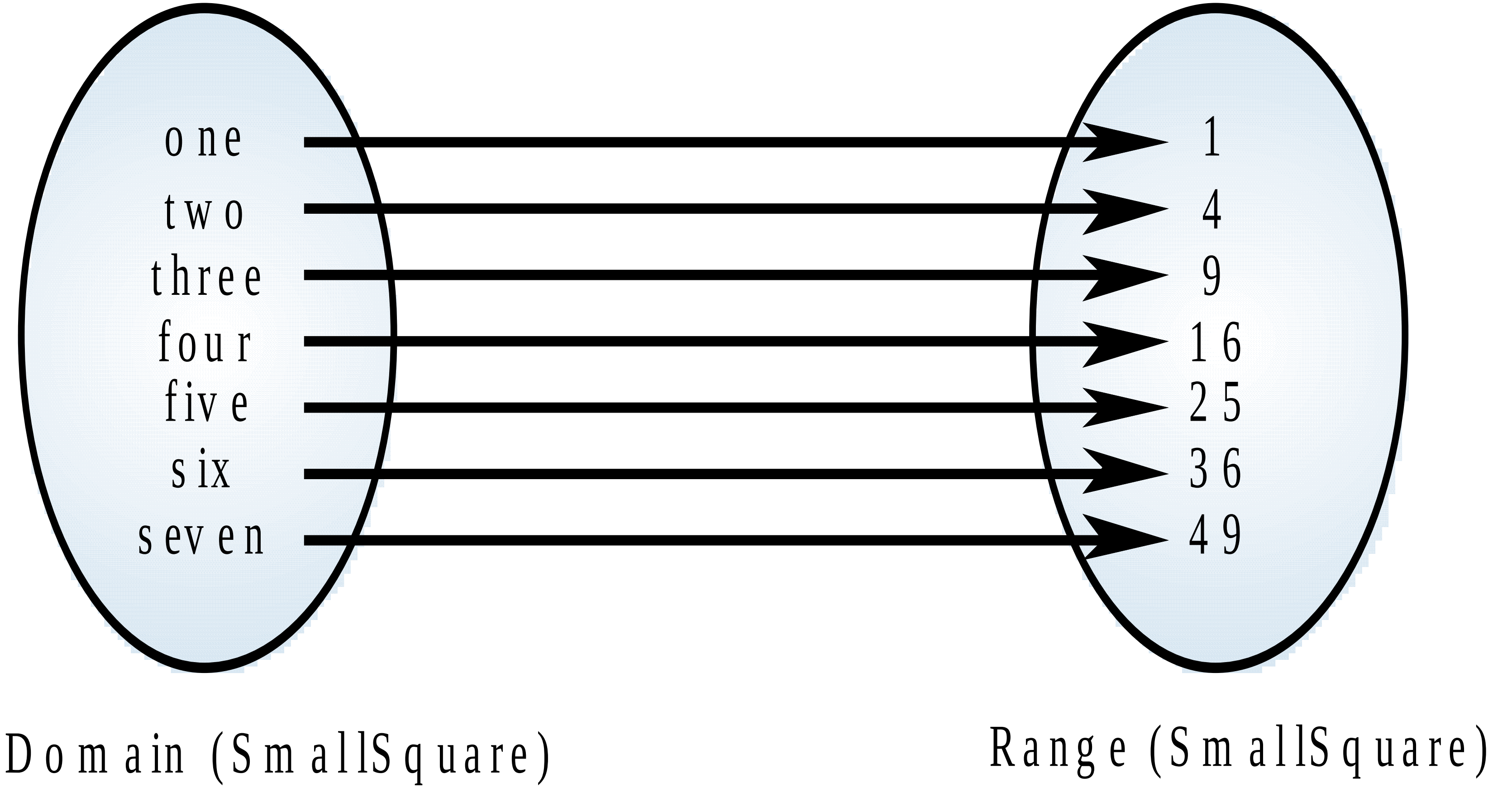
$DataDictionaryEntry$

$ddict : NAME \pfun \{DataDictionaryEntry\}$

Specification using functions

- A function is a mapping from an input value to an output value
  - SmallSquare = {1 $\rightarrow$ 1, 2 $\rightarrow$ 4, 3 $\rightarrow$ 9, 4 $\rightarrow$ 16, 5 $\rightarrow$ 25, 6 $\rightarrow$ 36, 7 $\rightarrow$ 49 }
- The domain of a function is the set of inputs over which the function has a defined result
  - dom SmallSquare = {1, 2, 3, 4, 5, 6, 7 }
- The range of a function is the set of results which the function can produce
  - ran SmallSquare = {1, 4, 9, 16, 25, 36, 49 }

The function SmallSquare



Data dictionary - initial state

Init-DataDictionary

DataDictionary’

ddict’ = Ø

Add and lookup operations

Add_OK

Δ DataDictionary
name?: NAME
entry?: DataDictionaryEntry

---

name? $\notin$ dom ddict
ddict' = ddict $\cup$ {name? $\mapsto$ entry?}

Lookup_OK

Ξ DataDictionary
name?: NAME
entry!: DataDictionaryEntry

---

name? $\in$ dom ddict
entry! = ddict (name?)

Add and lookup operations

Add_Error

Ξ DataDictionary
name?: NAME
error!: seq char

---

name? $\in$ dom ddict
error! = "Name already in dictionary"

Lookup_Error

Ξ DataDictionary
name?: NAME
error!: seq char

---

name? $\notin$ dom ddict
error! = "Name not in dictionary"

Function over-riding operator

- Replace Entry uses the function overriding operator (written $\oplus$). This adds a new entry or replaces an existing entry.
  - phone = $\{$ Ian $\rightarrow$ 3390, Ray $\rightarrow$ 3392, Steve $\rightarrow$ 3427$\}$
  - The domain of phone is {Ian, Ray, Steve} and the range is {3390, 3392, 3427}.
  - newphone = $\{$Steve $\rightarrow$ 3386, Ron $\rightarrow$ 3427$\}$
  - phone $\oplus$ newphone = $\{$ Ian $\rightarrow$ 3390, Ray $\rightarrow$ 3392, Steve $\rightarrow$ 3386, Ron $\rightarrow$ 3427$\}$

update operation

update

Δ DataDictionary
name?: NAME
entry?: DataDictionaryEntry

$ddict' = ddict \oplus \{name? \mapsto entry?\}$

Deleting an entry

- Uses the domain subtraction operator (written ⩤) which, given a name, removes that name from the domain of the function
  - phone = { Ian $\rightarrow$ 3390, Ray $\rightarrow$ 3392, Steve $\rightarrow$ 3427}
  - {Ian} ⩤ phone
  - Phone = {Ray $\rightarrow$ 3392, Steve $\rightarrow$ 3427}

Delete entry

Delete_OK

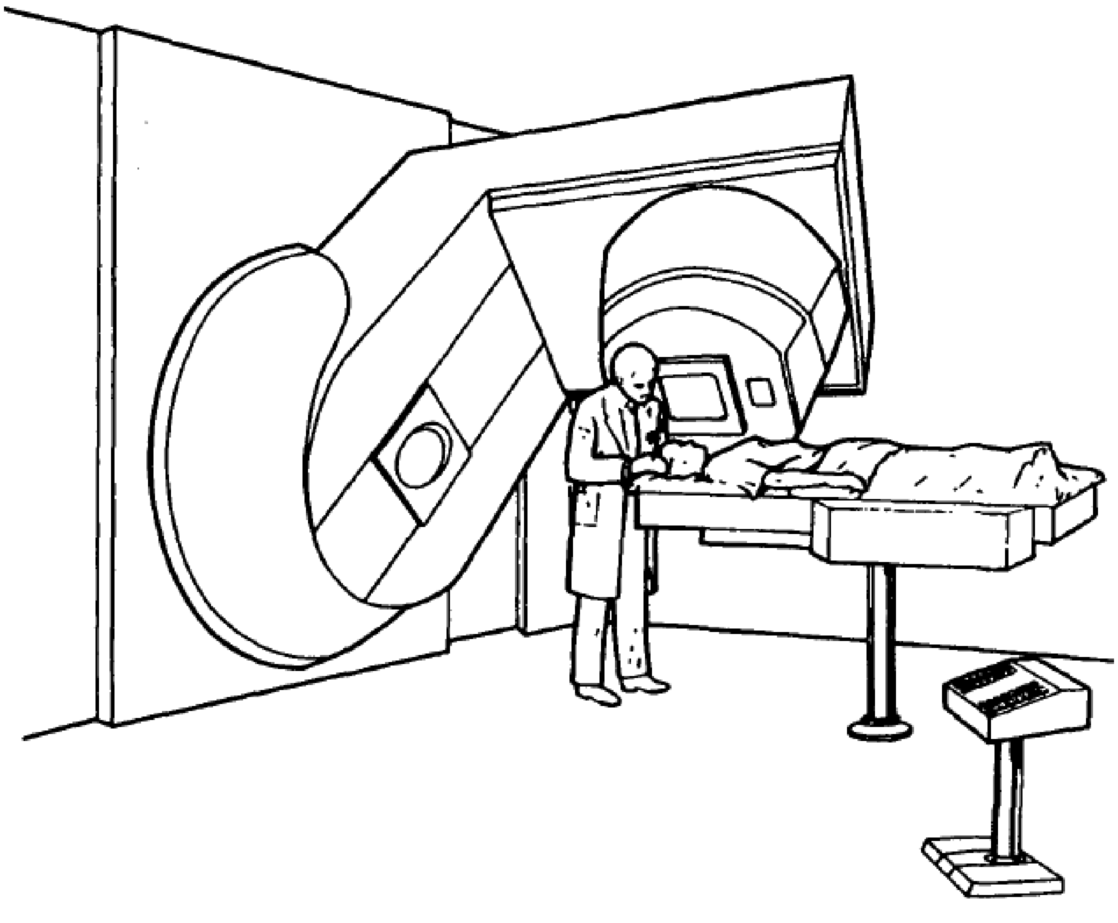
Δ DataDictionary
name?: NAME

---

name? $\in$ dom ddict
ddict' = {name?} $⩤$ ddict

From informal to formal description

- A control program for the therapist's console on a radiation therapy machine

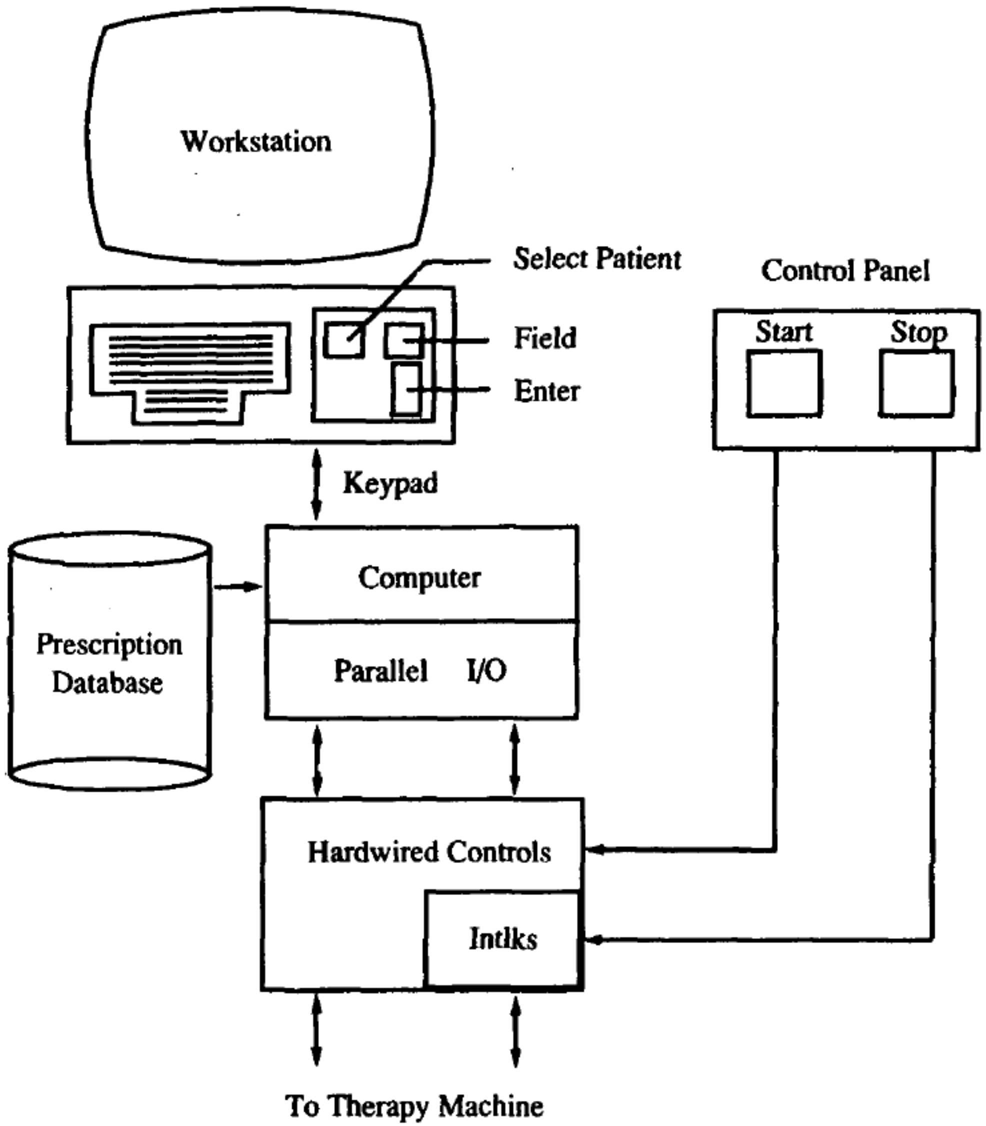


A control program for a therapy machine

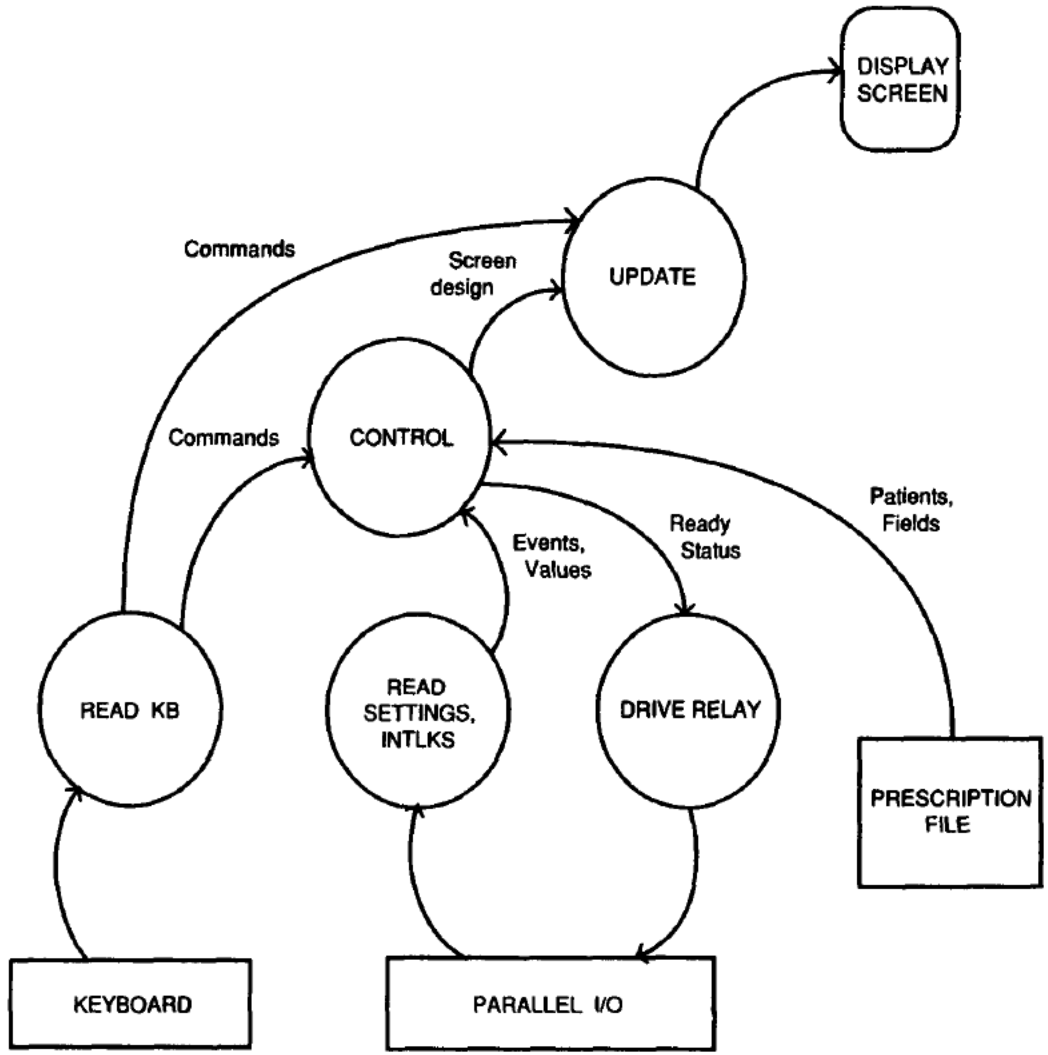
Informal requirements

- The system has a database of prescriptions for many patients
- Each prescription contains fields to define machine setting.
- The therapist operates the control program by pressing labelled function keys
- The control program is only responsible for checking the prescribed settings and actual settings agree
- Select Patient key is used to choose a patient's prescription from the system. Enter key is used to confirm the selection
- When all the settings match the prescription, the control program closes its relay and the workstation indicates the machine is ready
- Start button is used to turns the beam on
- Stop button is used to turns the beam off

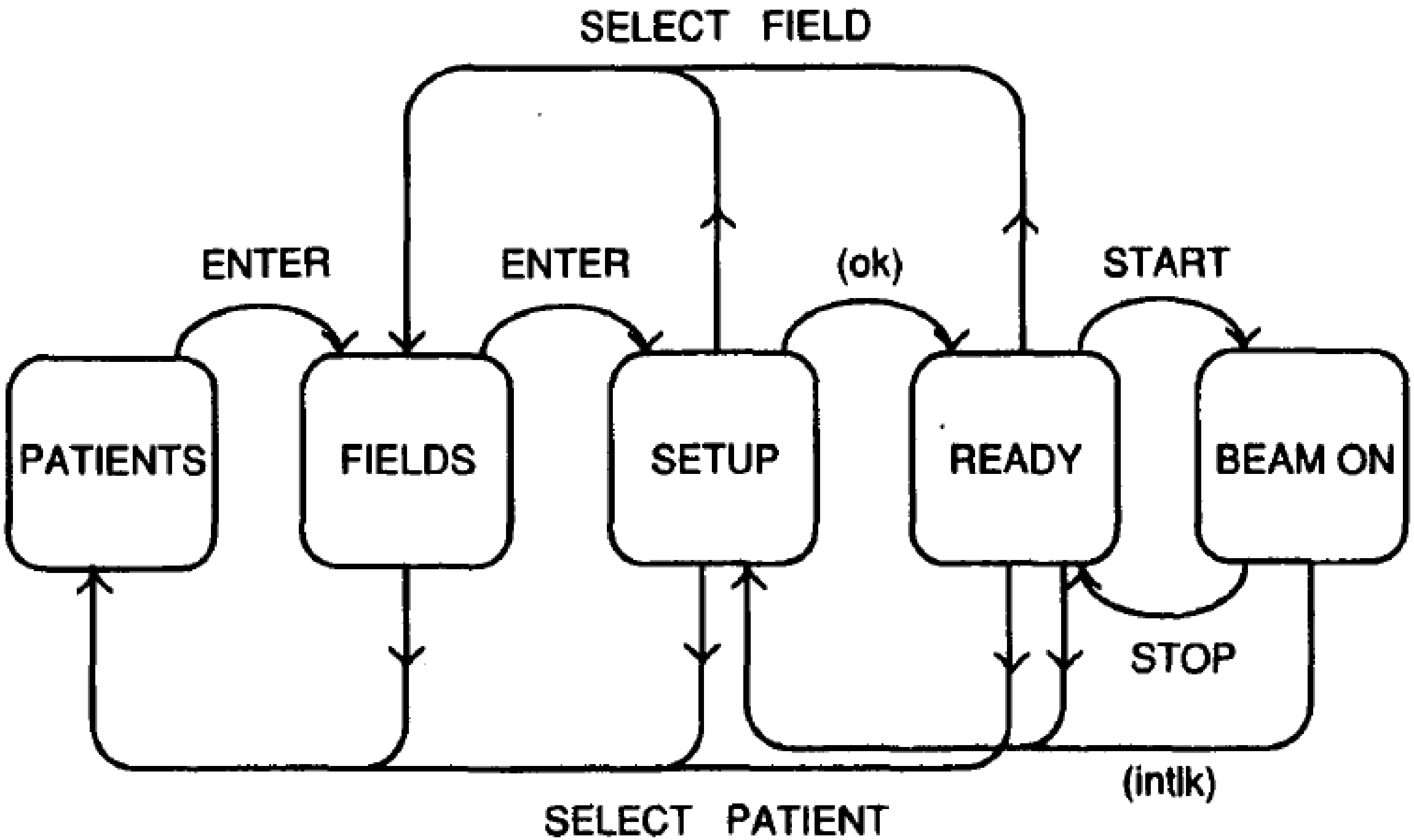
Therapy control console block diagram



Informal model: Data flow diagram



Informal model: state diagram



Informal model: state transition table

	SELECT PATIENT	SELECT FIELD	ENTER	ok	START	STOP	intlk
PATIENTS	—	—	FIELDS	—	—	—	—
FIELDS	PATIENTS	—	SETUP	—	—	—	—
SETUP	PATIENTS	FIELDS	—	READY	—	—	—
READY	PATIENTS	FIELDS	—	—	BEAM ON	—	SETUP
BEAM ON	—	—	—	—	—	READY	SETUP

Formal model: Z schema

Therapy control console

$STATE ::= patients \mid fields \mid setup \mid ready \mid beam\_on$

$EVENT ::= select\_patient \mid select\_field \mid enter \mid start \mid stop \mid ok \mid intlk$

$FSM == (STATE \times EVENT) \rightarrow\!\!\!\!\!\!\!+\;\; STATE$

$$
\begin{array}{|l}
no\_change, transitions, control : FSM \\
\hline
control = no\_change \oplus transitions \\
no\_change = \{\, s : STATE;\; e : EVENT \bullet (s, e) \mapsto s \,\} \\
transitions = \{\, (patients, enter) \mapsto fields, \\
\qquad (fields, select\_patient) \mapsto patients, (fields, enter) \mapsto setup, \\
\qquad (setup, select\_patient) \mapsto patients, (setup, select\_field) \mapsto fields, \\
\qquad\qquad (setup, ok) \mapsto ready, \\
\qquad (ready, select\_patient) \mapsto patients, (ready, select\_field) \mapsto fields, \\
\qquad\qquad (ready, start) \mapsto beam\_on, (ready, intlk) \mapsto setup, \\
\qquad (beam\_on, stop) \mapsto ready, (beam\_on, intlk) \mapsto setup \,\}
\end{array}
$$

Summary

- Model-based specification relies on building a system model using well-understood mathematical entities
- Z specifications are made up of mathematical model of the system state and a definition of operations on that state
- A Z specification is presented as a number of schemas
- Schemas may be combined to make new schemas
- Operations are specified by defining their effect on the system state. Operations may be specified incrementally then different schemas combined to complete the specification
- Z functions are a set of pairs where the domain of the function is the set of valid inputs. The range is the set of associated outputs. A sequence is a special type of function whose domain is the consecutive integers

Key points

- Operations are specified by defining their effect on the system state. Operations may be specified incrementally then different schemas combined to complete the specification
- Z functions are a set of pairs where the domain of the function is the set of valid inputs. The range is the set of associated outputs. A sequence is a special type of function whose domain is the consecutive integers