U O W

Software Requirements, Specifications and Formal Methods

A/Prof. Lei Niu

UNIVERSITY
OF WOLLONGONG
AUSTRALIA

Sequence and Logic

Sequences

- Sets are unordered collections
- When the order is significant, we can *use sequence to model the ordered collections*
- In programming languages, sequences can model arrays, lists and queues
- In Z, we declare a sequence of items from set S as: seq S

$$DAYS ::= friday \mid monday \mid saturday \mid sunday \mid thursday \mid tuesday \mid wednesday$$

$$\begin{array}{|l} weekday : \mathrm{seq}\, DAYS \\ \hline weekday = \langle monday, tuesday, wednesday, thursday, friday \rangle \end{array}$$

Sequences

- Actually, sequences are just functions whose domains are consecutive numbers, starting with one.
- Another way to write weekday is

$$weekday = \{1 \mapsto monday, 2 \mapsto tuesday, 3 \mapsto wednesday, 4 \mapsto thursday, \\ 5 \mapsto friday\}$$

- Therefore sequences are also functions, so operators defined for functions also apply to sequences

$$weekday\ 3 = wednesday$$

- Since sequences are also set, all the set operators also apply on sequences

  #weekday = 5

Sequences

E.g., lists, arrays, files, sequences, trace histories, ... Elements indexed and contiguously numbered.

1	2	3	4	5	6	7	8	9	$\cdots$

A sequence has 1^{st} element, 2^{nd} element, 3^{rd} element, etc... (numbered from 1 rather than 0 in Z)

$$\operatorname{seq} T == \{s : \mathbb{N} \text{ ⇻ } T \mid \operatorname{dom} s = 1 \mathrel{..} \# s\}$$

N.B.: sequences have a finite (but arbitrary) length. Sequences are functions, which are relations, which are sets.

Length of sequence s is its cardinality, $\# s$.

Empty sequence, $s = \emptyset$ has $\# s = 0$.
Normally written as:

$\langle\rangle$ – the empty sequence

Like the empty set $\emptyset$, the empty sequence is typed.

Sequences

Non-empty sequences:

$$\mathrm{seq}_1\, T == \mathrm{seq}\, T \setminus \{\langle\rangle\}$$

Injective sequences (i.e., no repeated elements):

$$\mathrm{iseq}\, T == \mathrm{seq}\, T \cap (\mathbb{N} \rightarrowtail\!\!\!\!\!\!\!+\;\; T)$$

The sequence containing just one element, $s = \{1 \mapsto x\}$, has $\#s = 1$ and is written as

$\langle x \rangle$ – a singleton sequence

The sequence $\{1 \mapsto x_1, 2 \mapsto x_2, \ldots, n \mapsto x_n\}$ is normally written as

$\langle x_1, x_2, \ldots, x_n \rangle$ – a multi-element sequence

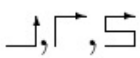
Sequences

Examples:

$\langle 11, 29, 3, 7 \rangle \in \mathrm{seq}\, primes$

$\langle \mathrm{J}, \mathrm{O}, \mathrm{N}, \mathrm{A}, \mathrm{T}, \mathrm{H}, \mathrm{A}, \mathrm{N} \rangle \in \mathrm{seq}\, CHAR$

Unlike for standard sets, two 'N' elements are distinct maplets $3 \mapsto$ N and $8 \mapsto$ N.

$\langle$  $\rangle \in \mathrm{seq}\, Path$

Length (= cardinality) of above sequences is 4, 8 and 3 respectively.

$\wr\wr\wr$ N.B., unlike sets, sequences *can* have repeated elements. E.g.:

$$\langle Emma \rangle \neq \langle Emma, Emma, Emma \rangle$$

$\wr\wr\wr$ Also, the order is significant:

$$\langle Alice, Emma \rangle \neq \langle Emma, Alice \rangle$$

Sequences

Concatenation operation $\frown$

The concatenation operation combines two sequences into one.

week = = <weekday> $\frown$ <weekend> or

week = = <weekday> $\frown$ <Saturday> $\frown$ <Sunday>

Note: we must use the brackets to make <Saturday> and <Sunday> into sequences of one element because both operands of the concatenation operator must be sequences.

Sequences

Concatenation

For $s, t \in \operatorname{seq} T$

$s \frown t$ – the function $1 \mathbin{..} (\#s + \#t) \to T$
with elements

$$j \mapsto \begin{cases} s(j) & \text{if } 1 \leq j \leq \#s \\ t(j - \#s) & \text{if } \#s < j \leq (\#s + \#t) \end{cases}$$

More formally:

$$s \frown t = s \cup (\_ - \#s) \mathbin{⨾} t$$

Concatenation of two sequences of length 5 and 3:

$$\overbrace{\underbrace{\square\square\square\square\square}_{s} \;\; \underbrace{\square\square\square}_{t}}^{s \frown t}$$

$$\#s = 5 \quad \#t = 3$$

$$\#(s \frown t) = \#s + \#t = 5 + 3 = 8.$$

Sequences

Concatenation example:

$$\langle \mathsf{A} \rangle \frown \langle \mathsf{L}, \mathsf{I}, \mathsf{C}, \mathsf{E} \rangle =$$
$$\langle \mathsf{A}, \mathsf{L} \rangle \frown \langle \mathsf{I}, \mathsf{C}, \mathsf{E} \rangle =$$
$$\langle \mathsf{A}, \mathsf{L}, \mathsf{I}, \mathsf{C}, \mathsf{E} \rangle$$

$\langle a, b, c, \ldots \rangle$ is shorthand for $\langle a \rangle \frown \langle b \rangle \frown \langle c \rangle \frown \ldots$

Laws

$$\langle\rangle \frown s = s \frown \langle\rangle = s$$
$$r \frown (s \frown t) = (r \frown s) \frown t$$
$$(r \frown s = r \frown t) \Rightarrow s = t$$

Sequences

Other sequence operations

If $s \in \mathrm{seq}\, T$ and $s \neq \langle\rangle$ (i.e., $s \in \mathrm{seq}_1\, T$):

First element:

$$head\ s \;=\; s(1)$$ ●○○○○○

Last element:

$$last\ s \;=\; s(\#s)$$ ○○○○○●

No first element:

$$tail\ s \;=\; succ \,⨾\, (\{1\} ⩤ s)$$ ○●●●●●

No last element:

$$front\ s \;=\; \{\#s\} ⩤ s$$ ●●●●●○

Avoid applying these functions to an empty sequence;
e.g., $head\langle\rangle$ is undefined. $\wr\wr\wr$

Sequences

Example

C_1	O_2	D_3	E_4

For $s = \langle C, O, D, E \rangle$
(which is $\{1 \mapsto C, 2 \mapsto O, 3 \mapsto D, 4 \mapsto E\}$)

$$head\ s = C$$

$$last\ s = E$$

$$\begin{aligned} tail\ s &= \{0 \mapsto 1, 1 \mapsto 2, 2 \mapsto 3, 3 \mapsto 4, \ldots\} \mathbin{;} \\ &\quad \{2 \mapsto O, 3 \mapsto D, 4 \mapsto E\} \\ &= \{1 \mapsto O, 2 \mapsto D, 3 \mapsto E\} \\ &= \langle O, D, E \rangle \end{aligned}$$

$$tail\ s \;=\; succ \mathbin{;} (\{1\} \mathbin{⩤} s)$$

$$\begin{aligned} front\ s &= \{4\} \mathbin{⩤} \{1 \mapsto C, 2 \mapsto O, 3 \mapsto D, 4 \mapsto E\} \\ &= \{1 \mapsto C, 2 \mapsto O, 3 \mapsto D\} \\ &= \langle C, O, D \rangle \end{aligned}$$

$$front\ s \;=\; \{\#s\} \mathbin{⩤} s$$

UNIVERSITY OF WOLLONGONG AUSTRALIA

Sequences

More general versions of *front* and *tail*, using *generic construction*:

$$
\begin{array}{l}
[T] \\
\hline
\_\ \underline{for}\ \_, \\
\_\ \underline{after}\ \_ : (\operatorname{seq} T) \times \mathbb{N} \rightarrow (\operatorname{seq} T) \\
\hline
\forall s : \operatorname{seq} T;\ n : \mathbb{N} \bullet \\
\quad s\ \underline{for}\ n = (1 \mathinner{..} n) \lhd s\ \wedge \\
\quad s\ \underline{after}\ n = (\{0\} \mathbin{⩤} succ^n) \mathbin{;} s
\end{array}
$$

E.g., for extracting portions of files as a sequence of bytes.

For $s \in \operatorname{seq}_1 T$,

$$
\begin{aligned}
front\ s &= s\ \underline{for}\ (\#s - 1) \\
tail\ s &= s\ \underline{after}\ 1
\end{aligned}
$$

Laws:

$$
\begin{aligned}
s\ \underline{for}\ 0 &= \langle\rangle \\
s\ \underline{for}\ \#s &= s \\
s\ \underline{after}\ 0 &= s \\
s\ \underline{after}\ \#s &= \langle\rangle
\end{aligned}
$$

Sequences

Reversal

Reverse of a sequence: $\mathit{rev}\ s$

a_1	b_2	c_3	d_4	e_5	f_6	g_7

g_1	f_2	e_3	d_4	c_5	b_6	a_7

Sequence $\langle \mathrm{D}, \mathrm{O}, \mathrm{G} \rangle$ is converted to $\langle \mathrm{G}, \mathrm{O}, \mathrm{D} \rangle$ using the *rev* function!

Laws

$$
\begin{aligned}
\mathit{rev}\langle\rangle &= \langle\rangle \\
\mathit{rev}\langle x \rangle &= \langle x \rangle \\
\mathit{rev}(\mathit{rev}\ s) &= s \\
\mathit{rev}(s \frown t) &= (\mathit{rev}\ t) \frown (\mathit{rev}\ s)
\end{aligned}
$$

For example

$$\mathit{rev}(\langle a, b \rangle \frown \langle c, d \rangle) = \langle d, c \rangle \frown \langle b, a \rangle$$

Sequences

Distributed operations

Concatenation of a sequence of sequences:

$$\frown/\langle\rangle = \langle\rangle$$

$$\frown/\langle a, b, \ldots, n\rangle = a \frown b \frown \ldots \frown n$$

More formally, $\frown/ : \mathrm{seq}(\mathrm{seq}\, T) \to \mathrm{seq}\, T$ satisfies

$$\frown/(\langle a\rangle \frown s) = \langle a\rangle \frown (\frown/\, s)$$

and also

$$\frown/(s \frown \langle a\rangle) = (\frown/\, s) \frown \langle a\rangle$$

Question: What is $rev(\frown/\langle\langle \mathsf{N}, \mathsf{A}, \mathsf{H}\rangle, \langle \mathsf{T}, \mathsf{A}\rangle, \langle \mathsf{N}, \mathsf{O}\rangle, \langle \mathsf{J}\rangle\rangle)$?

Basic predicates

- Predicates are the textual unit of logic

- A few kinds of basic predicates
  - Two values for predicates: True and false
    - Equals: e1 = e2
    - Set membership: $x \in S$

Using predicates in Z

- We create models by a process of specialization or restriction
- First, we declare data types, variables
- Then, we add predicates to *<u>specify the particular objects that we want</u>*

For example, we throw two dices with two integer variables, this declaration restricts their values to the range from one to six:

$$\mid\ d_1, d_2 : 1 \mathbin{..} 6$$

Using predicates in Z

- A situation is a particular assignment of values to variables
- The two dices example has 36 distinct situations

d_1	1	1	1	1	1	1	2	2	2	2	...	5	5	5	6	6	6	6	6	6
d_2	1	2	3	4	5	6	1	2	3	4	...	4	5	6	1	2	3	4	5	6

- If we restrict the situations, then we can add a predicate to admit only situations where two numbers add up to seven:

$$
\begin{array}{|l}
d_1, d_2 : 1 \ldots 6 \\
\hline
d_1 + d_2 = 7
\end{array}
$$

d_1	1	2	3	4	5	6
d_2	6	5	4	3	2	1

Relations as predicates

- Predicates in Z definitions don't have to be equations

$$\begin{array}{|l} d_1, d_2 : 1 \mathinner{..} 6 \\ \hline d_1 < d_2 \end{array}$$

- The predicate in this definition is satisfied in these situations

d_1	1	1	1	1	1	2	2	2	2	3	3	3	4	4	5
d_2	2	3	4	5	6	3	4	5	6	4	5	6	5	6	6

Relations as predicates

- Less than (<) is a relation
- In Z, we can use relationship to form a predicate
- Unary relations
  - One argument
  - Such as odd(*x*) and leap_year(*year*)
- Binary relations
  - Two arguments
  - Such as <, divides, ⊆, and mother(*son, mum*)

Relations as predicates

- In Z, we can define our own relations with a prefix, i.e., an expression without its arguments

$$\begin{array}{|l} odd\_ : \mathbb{P}\,\mathbb{Z} \\ \hline \ldots \text{definition omitted} \ldots \end{array}$$

- Then we can express that k is odd via *odd(k)*
- Another example: mother

$$\begin{array}{|l} mother\_ : PERSON \leftrightarrow PERSON \\ \hline (mother\_) = \{(ishmael, hagar), (isaac, sarah), (esau, rebekah), \\ \qquad (jacob, rebekah)\} \end{array}$$

Relations as predicates

- A binary relations divides, the set of pairs of numbers where the first evenly divides the second,
  - 4 divides 12 is true
  - 5 divides 12 is false

- We can define divides

$$\begin{array}{|l}
\textit{divides} : \mathbb{Z} \leftrightarrow \mathbb{Z} \\
\hline
\ldots \text{definition omitted} \ldots
\end{array}$$

- And we can express that 4 divides 12 as
  - $(4, 12) \in$ divides, or
  - 4 divides 12

Logical connectives

- We use logical connectives to build complex predicates from simple ones

- The truth value of a predicate that contains logical connective is determined by the truth values of its constituent simple predicates

- In the following discussion, we will use p and q to stand for any predicate

Conjunction

- The predicate $p \wedge q$ (p and q) is called a conjunction
- A conjunction is used to strengthen predicates by combining requirements
- A conjunction is only satisfied by situations that satisfy both of its conjects: It is true only when both of its conjects are true
- Truth table for the conjunction

p	q	$p \wedge q$
false	*false*	*false*
false	*true*	*false*
true	*false*	*false*
true	*true*	*true*

Conjunction

- This predicate says that the numbers on the two dice add up to seven, and the first number is less than the second:

$$\begin{array}{|l} d_1, d_2 : 1 \mathinner{.\,.} 6 \\ \hline (d_1 + d_2 = 7) \land (d_1 < d_2) \end{array}$$

- It is satisfied in three situations only:

d_1	1	2	3
d_2	6	5	4

Disjunction

- The predicate $p \vee q$ (p or q) is called a disjunction

- Disjunction is used to offer alternatives

- A disjunction is satisfied by any situation that satisfied any of its disjuncts

- It is true when either or both of its disjuncts is true

p	q	$p \vee q$
false	*false*	*false*
false	*true*	*true*
true	*false*	*true*
true	*true*	*true*

Disjunction

- A disjunction is said to be weaker than its disjuncts because it is usually satisfied by a larger number of situations

$$(d_1 + d_2 = 7) \vee (d_1 < d_2)$$

d_1	1	2	3	4	5	6	1	1	1	1	1	2	2	2	2	3	3	3	4	4	5
d_2	6	5	4	3	2	1	2	3	4	5	6	3	4	5	6	4	5	6	5	6	6

- Disjunctions can be used to express case analyses where situations can be classified into cases and all the situations in a case are handled the same way

Disjunction

- The following predicates define the status of water on different degrees

$$TEMP == \mathbb{Z}$$

$$PHASE ::= solid \mid liquid \mid gas$$

$$
\begin{array}{|l}
temp : TEMP \\
phase : PHASE \\
\hline
(temp < 0 \land phase = solid) \lor \\
(0 \leq temp \leq 100 \land phase = liquid) \lor \\
(temp > 100 \land phase = gas)
\end{array}
$$

Negation

- The predicate $\neg p$ (not p) is called a negation.

- Negation inverts the truth value of a predicate

- Negation $\neg p$ is satisfied in all situation that are not satisfied by p

- When p is true/false, its negation is false/true

p	$\neg p$
true	*false*
false	*true*

Equivalence

- The predicate $p \Leftrightarrow q$ (p equals q) is called equivalence
- The equivalence is true when p and q have the same truth value, no mater it is true or false
- An equivalence is satisfied in situations that make both its constituent predicates true or false

p	q	$p \Leftrightarrow q$
false	*false*	*true*
false	*true*	*false*
true	*false*	*false*
true	*true*	*true*

Equivalence

Quiz:

- Check the following predicates and answer what happens at temp = 0? and temp = 100?

$$
\begin{array}{|l}
temp : TEMP \\
phase : PHASE \\
\hline
temp \leq 0 \Leftrightarrow phase = solid \\
0 \leq temp < 100 \Leftrightarrow phase = liquid \\
temp > 100 \Leftrightarrow phase = gas
\end{array}
$$

- When temp = 0, the water is a mixture of solid and liquid
- When temp =100, the water is not in any status

Implication

- The predicate $p \Rightarrow q$ (p implicate q) is called implication
- The implication $p \Rightarrow q$ is true in every case except when p is true and q is false

p	q	$p \Rightarrow q$
false	*false*	*true*
false	*true*	*true*
true	*false*	*false*
true	*true*	*true*

- p is the antecedent (pre-condition), and q is the consequent (post-condition).
- If a true antecedent generates a false consequent, then the implication is false. Otherwise, the implication is always true

Universal quantifier

- Quantifiers introduce local variables into predicates
- The simple ∀ (for all), is the universal quantifier
- A general form of a universally quantified predicate is

$$\forall\, \textit{declaration} \bullet \textit{predicate}$$

- It indicates the predicate after the delimiter (dot) is true for all values of bound variables that are admitted by the declaration before the dot
- The scope of the bound variables is limited to the predicate; outside this scope, the bound variables are undefined

Universal quantifier

$$
\begin{array}{|l}
nmax : \mathbb{Z} \\
ns : \mathbb{P}\,\mathbb{Z} \\
\hline
\forall\, i : ns \bullet i \leq nmax
\end{array}
$$

- This predicate is pronounced, “For all i in ns, i is less than or equal to nmax”
- The bound variable i is just a place-holder that stands for any element of ns

Universal quantifier

- Consider the relation divides,

$$divides == \{\ldots, (5, 10), (10, 10), (1, 11), (11, 11), (1, 12), (2, 12), (3, 12), \ldots\}$$

- Can be rewritten in a more formal way

$$\begin{array}{|l}
divides : \mathbb{Z} \leftrightarrow \mathbb{Z} \\
\hline
\forall d, n : \mathbb{Z} \bullet \\
\qquad d \; \underline{divides} \; n \Leftrightarrow n \bmod d = 0
\end{array}$$

Universal quantifier

Let assume ns = {n1, n2, n3,…}

Then the quantified predicate

$$\forall i : ns \bullet i \leq nmax$$

means the same think as

$$n_1 \leq nmax \land n_2 \leq nmax \land n_3 \leq nmax \land \ldots$$

So in another word, the universal quantifier is a generalization of logic "and".

Existential quantifier

- There is another quantifier which is a generalization of logic “or”
- The existential quantifier ∃ (exists)
- A general form of a existential quantified predicate is

$$\exists \text{ declaration} \cdot predicate$$

- It indicates the predicate after the delimiter (dot) is true for at least one values of bound variables that are admitted by the declaration before the dot

Existential quantifier

$$\exists i : ns \bullet i \leq nmax$$

- This predicate is pronounced, “there exists an i in ns, such that i is less than or equal to nmax.”

- It is an abbreviation for this disjunction:

$$n_1 \leq nmax \lor n_2 \leq nmax \lor n_3 \leq nmax \lor \ldots$$

Boolean types

- Z has no built-in Boolean type
- The following example is NOT Z (wrong example)

$$
\begin{array}{|l}
beam, door : BOOLEAN \\
\hline
beam \Rightarrow door
\end{array}
$$

✗

- In Z, we have to define the binary enumerations instead

$$BEAM ::= off \mid on$$

$$DOOR ::= closed \mid open$$

Set Comprehensions

- We had introduced all basic components of Z, and now we can do some real work with Z notations.

$$ODD == \{\ldots, -5, -3, -1, 1, 3, 5, \ldots\}$$

- This is not a formal definition of set ODD in Z

- A formal definition of set ODD shall be

$$ODD == \{i : \mathbb{Z} \bullet 2 * i + 1\}$$

Set Comprehensions

- A set comprehension has the form

  { *declaration* | *predicate* • *expression* }

  { *source* | *filter* • *pattern* }

- Declaration (source): introduce all variables used in the predicate and expression

- Predicate (filter): specify the constrictions on the values of the variables

- Expression (pattern): specify the features of the set members

- The predicate and expression are optional

Set Comprehension

How to define a set of odd numbers beginning with 11 :

1. Define the set of natural number (source) with the declaration

$$\{\, i : \mathbb{N} \,\} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, \ldots\}$$

2. Add the predicate (filter), i.e., only elements larger than four can pass through

$$\{\, i : \mathbb{N} \mid i > 4 \,\} = \{5, 6, 7, 8, \ldots\}$$

3. Add the expression (pattern) and transform the elements

$$\{\, i : \mathbb{N} \mid i > 4 \bullet 2 * i + 1 \,\} = \{\, 11, 13, 15, 17, \ldots \}$$

Lambda expression

- Functions can also be defined using the set comprehension
- We can also use the *lambda expression* to define a function
- A lambda expression is an abbreviation for a set comprehension and retains the same declaration, predicate and expression structure

$$(\lambda\, \textit{declaration} \mid \textit{predicate} \bullet \textit{expression})$$

- The Greek letter lambda indicates it is a function, but not just an ordinary set

Lambda expression

Three ways to define a function

- Using the set comprehension

$$isqr == \{ i : \mathbb{Z} \bullet i \mapsto i * i \}$$

- Using a lambda expression

$$isqr == (\lambda\, i : \mathbb{Z} \bullet i * i)$$

- Using an axiomatic definition with a quantifier

$$\begin{array}{|l} isqr : \mathbb{Z} \rightarrow \mathbb{N} \\ \hline \forall i : \mathbb{Z} \bullet isqr\ i = i * i \end{array}$$

They all means the same:

$$isqr = \{ \ldots, -2 \mapsto 4, -1 \mapsto 1, 0 \mapsto 0, 1 \mapsto 1, 2 \mapsto 4, \ldots \}$$

Formal specification of prime number

With the set comprehension, we can define the formal specification of sets

- The formal specification of prime number:
  - An integer larger than 1 that is only divisible by itself and 1
  - 2, 3, 5, 7, 11, 13, …

$$PRIME == \{ n : \mathbb{N} \mid n > 1 \land \neg (\exists m : 2 \mathinner{..} n - 1 \bullet n \bmod m = 0) \}$$

- Alternatively, we can use the set difference operator (\ ) to remove all exclusive elements

$$\mathbb{N}_2 == \mathbb{N} \setminus \{0, 1\}$$

$$PRIME == \mathbb{N}_2 \setminus \{ \forall n, m : \mathbb{N}_2 \bullet n * m \}$$

Local definitions

- Sometimes we want to introduce a local variable that has on particular value, then we can use "*let*"

- We use the *let* construct to avoid writing the same expression again and again

For example, we introduced the integer square root predicate:

$$\forall a : \mathbb{N} \bullet iroot(a) * iroot(a) \leq a < (iroot(a) + 1) * (iroot(a) + 1)$$

The predicate spells out iroot(a) four time, then we can use let to abbreviate it with the single letter r

$$\forall a : \mathbb{N} \bullet (\mathbf{let}\ r == iroot(a) \bullet r * r \leq a < (r + 1) * (r + 1))$$

Conditional expressions

- Sometimes we wish to assign one or another value to a variable, depending on the truth of some predicate

- We can use the *conditional expression* construct " **if … then … else …** " to express the two-way cases analysis

- For example, the absolute value function can be defined as

$$
\begin{array}{|l}
|\_| : \mathbb{Z} \rightarrow \mathbb{N} \\
\hline
\forall x : \mathbb{Z} \bullet |x| = \textbf{if } x \geq 0 \textbf{ then } x \textbf{ else } -x
\end{array}
$$

- Actually, the conditional expression is an abbreviation for disjunction

$$\forall x : \mathbb{Z} \bullet (x \geq 0 \wedge |x| = x) \vee (x < 0 \wedge |x| = -x)$$