Formal Method in Software Engineering (SE-313)

Course Teacher
Assistant Professor
Engr. Mustafa Latif

Z Formal Specification

Determining the validity of arguments

- **Semantic Tableaux**
- **The basic idea of semantic tableaux is to determine if it is possible for a conclusion to be false when all of the premises are true.**
- **If this is not possible, then the conclusion must be true when the premises are true, and so the conclusion is semantically entailed by the premises.**
- **Semantic tableaux is based on the idea of proof by contradiction.**

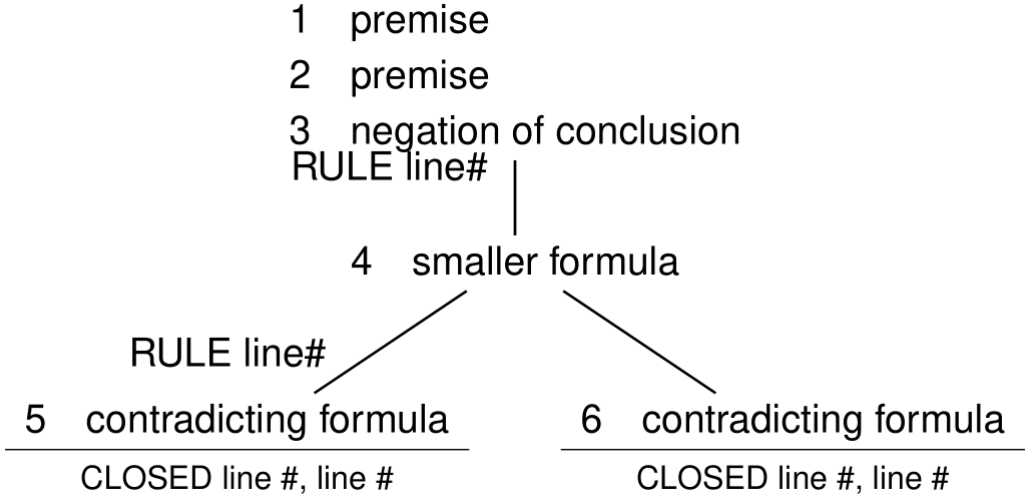
Z Formal Specification

Semantic Tableaux

- **Whenever a logical expression A and its negation ¬A appear in a branch of the tableau, then an inconsistency has been identified in that branch, and the branch is said to be closed.**
- **The method of proof is to negate the conclusion, and to show that all branches in the semantic tableau are closed and that therefore it is not possible for the premises of the argument to be true and for the conclusion to be false. Therefore, the argument is valid and the conclusion follows from the premises.**

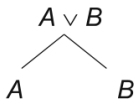
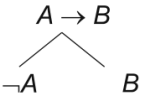
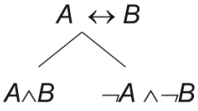
3

General Form of Tableaux

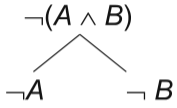


4

Table 6.12 Rules of semantic tableaux

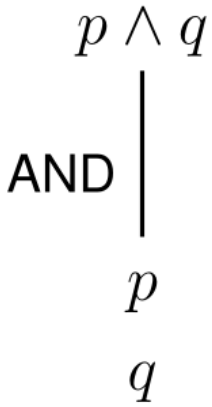
Rule No.	Definition	Description
1	$A \wedge B$ A B	If $A \wedge B$ is true, then both A and B are true and may be added to the branch containing $A \wedge B$
2.		If $A \vee B$ is true, then either A or B is true, and we add two new branches to the tableaux, one containing A and one containing B
3.		If $A \rightarrow B$ is true, then either $\neg A$ or B is true, and we add two new branches to the tableaux, one containing $\neg A$ and one containing B
4.		If $A \leftrightarrow B$ is true, then either $A \wedge B$ or $\neg A \wedge \neg B$ is true, and we add two new branches, one containing $A \wedge B$ and one containing $\neg A \wedge \neg B$

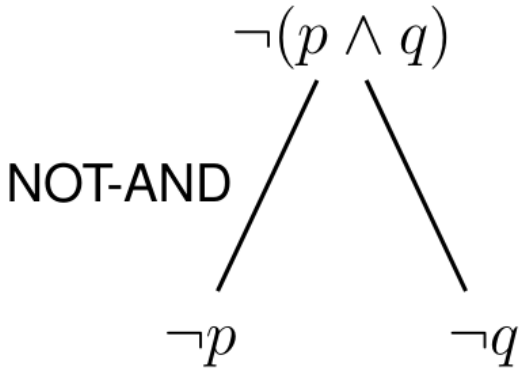
5

5.	$\neg\neg A$ A	If $\neg\neg A$ is true, then A may be added to the branch containing $\neg\neg A$
6.		If $\neg(A \wedge B)$ is true, then either $\neg A \vee \neg B$ is true, and we add two new branches to the tableaux, one containing $\neg A$ and one containing $\neg B$
7.	$\neg(A \vee B)$ $\neg A$ $\neg B$	If $\neg(A \vee B)$ is true, then both $\neg A \wedge \neg B$ are true and may be added to the branch containing $\neg(A \vee B)$
8.	$\neg(A \rightarrow B)$ A $\neg B$	If $\neg(A \rightarrow B)$ is true, then both $A \wedge \neg B$ are true and may be added to the branch containing $\neg(A \rightarrow B)$

6

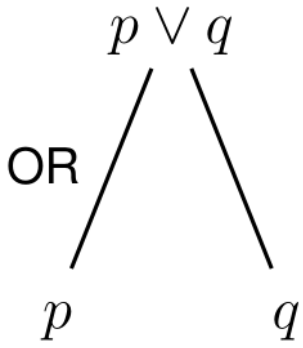
Rules for Conjunction

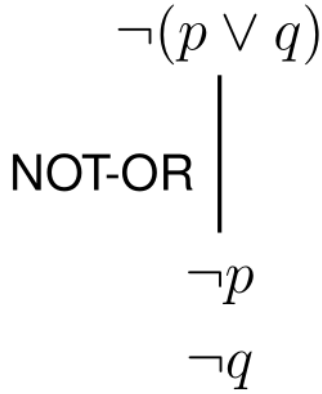




7

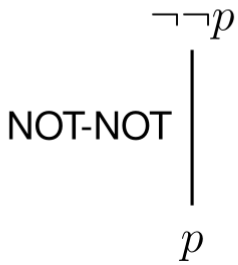
Rules for Disjunction





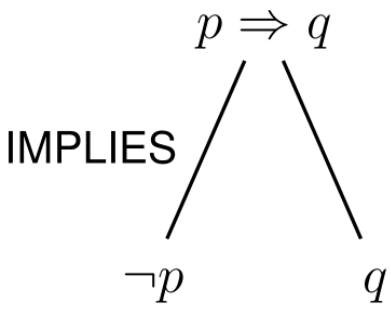
8

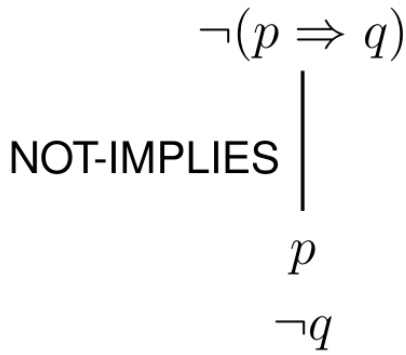
Rule for Negation



9

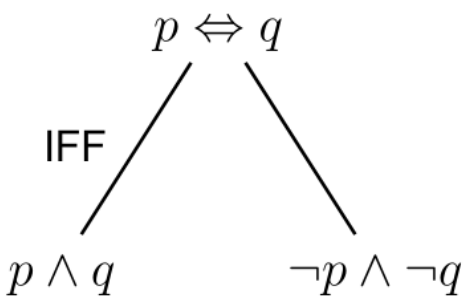
Rules for Implication

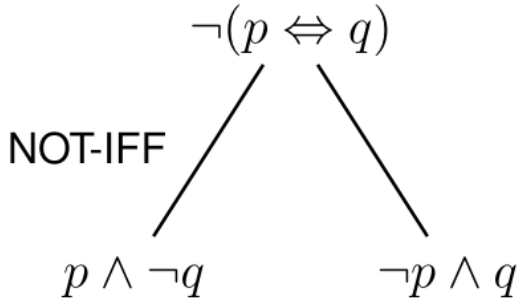




10

Rules for Equivalence





Z Formal Specification

Determining the validity of arguments

- **Example Consider the argument and determine if it is valid or not . Perform the proof using semantic tableaux.**
- **Argument:**
  - **If the pianist plays the concerto, then crowds will come if the prices are not too high.**
  - **If the pianist plays the concerto, then the prices will not be too high.**
  - **Therefore, if the pianist plays the concerto, then crowds will come.**

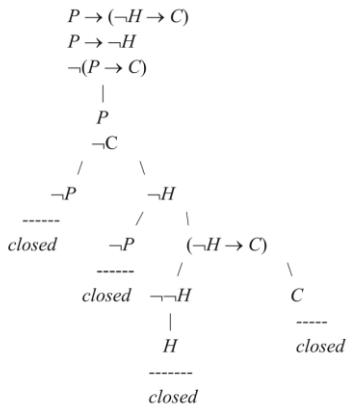
Z Formal Specification

Determining the validity of arguments

- **Solution:**
  - **Let P stand for "The pianist plays the concerto";**
  - **C stands for "Crowds will come";**
  - **and H stands for "Prices are too high".**
- **Then, the argument may be expressed in propositional logic as:**

$$
\begin{array}{ll}
P \rightarrow (\neg H \rightarrow C) & \text{(Premise 1)} \\
P \rightarrow \neg H & \text{(Premise 2)} \\
\ldots & \\
P \rightarrow C & \text{(Conclusion)}
\end{array}
$$

13



- We have showed that all branches in the semantic tableau are closed and that therefore it is not possible for the premises of the argument to be true and for the conclusion to be false. Therefore, the argument is valid as required.

14

Z Formal Specification

Extra rules of semantic tableaux (for predicate calculus)

From	Can Derive	Name / Abbreviation
(∀x)P(x)	P(t) where t is a variable or constant Symbol	Universal Instantiation- **ui**
(∃x)P(x)	P(t) where t is a variable or constant symbol not previously used in a proof sequence	Existential Instantiation- **ei**
P(x)	(∀x)P(x)	Universal Generalization- **ug**
P(x)	(∃x)P(x)	Existential Generalization- **eg**

15

Z Formal Specification

Determining the validity of arguments

- **Example Show that the Argument:**
  - **"All Greeks are mortal;**
  - **Socrates is a Greek;**
  - **therefore Socrates is mortal" is a valid argument in predicate calculus.**

16

Z Formal Specification

Determining the validity of arguments

- **Solution:**
  - **Let M(x) stand for " x is mortal";**
  - **G(x) stands for "x is Greeks";**
- **Then, the argument may be expressed in predicate logic as:**
  - **(∀ x)(G(x) → M(x)); G(s); M(s).**
  - **Therefore, we negate the conclusion (i.e. ¬M(s)) and try to construct a closed tableau.**
-

17

$(\forall x)(G(x) \rightarrow M(x))$

$G(s)$

$\neg M(s).$

$G(s) \rightarrow M(s)$

$\wedge$

$\neg G(s)$ $M(s)$

---

closed *closed*

18

Z Formal Specification

Determining the validity of arguments

- **Solution:**
  - **Let M(x) stand for " x is mortal";**
  - **G(x) stands for "x is Greeks";**
- **Then, the argument may be expressed in predicate logic as:**
  - **(∀ x)(G(x) → M(x)); G(s); M(s).**
  - **Therefore, we negate the conclusion (i.e. ¬M(s)) and try to construct a closed tableau.**
-

19

Z Formal Specification

Determining the validity of arguments

- **Determine whether the following argument is valid.**
  - **All lecturers are motivated.**
  - **Anyone who is motivated and clever will teach well.**
  - **Joanne is a clever lecturer.**
  - **Therefore, Joanne will teach well.**

20

Z Formal Specification

Modelling Concurrent Systems

- Concurrency is a form of computing in which multiple computations (processes) are executed during the same period of time.
- Concurrency employs interleaving where the execution steps of each process employ time-sharing slices so that only one process runs at a time, and if it does not complete within its time slice it is paused; another process begins or resumes; and then later the original process is resumed.

21

Z Formal Specification

Modelling Concurrent Systems

- Concurrency-specific errors such as deadlock and livelock.
- A deadlock is a situation in which the system has reached a state in which no further progress can be made, and at least one process needs to complete its tasks.
- Livelock refers to a situation where the processes in a system are stuck in a repetitive task and are making no progress towards their functional goals.

22

Z Formal Specification

Modelling Concurrent Systems

- It is essential that safety properties such as mutual exclusion (at most one process is in its critical section at any given time) are not violated.
- In other words, something bad (e.g. a deadlock situation) should never happen;
- liveness properties (a desired event or something good eventually happens) are satisfied;
- and invariants (properties that are true all the time) are never violated.
- These behavior errors may be mechanically detected if the systems are properly modelled and analyzed.

23

Z Formal Specification

Modelling Concurrent Systems

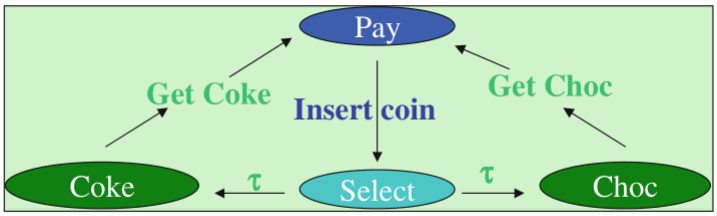
- Transition systems are often used as models to describe the behavior of systems, and these are directed graphs with nodes representing states and edges that represent state transitions. A state describes information about a system at a certain moment of time.
- For example, the state of a sequential computer consists of the values of all program variables and the current value of the program counter (pointer to next program instruction to be executed).

24

Z Formal Specification

Modelling Concurrent Systems

- A transition describes the conditions under which a system moves from one state to another.



Simple transition system

25

Z Formal Specification

Modelling Concurrent Systems

- The transitions are associated with action labels that indicate the actions that cause the transition.
- For example, in the Figure the Insert coin is a user action, whereas the Get coke and Get choc are actions that are performed by the machine.
- The activity τ represents an internal activity of the vending machine that is not of interest to the modeler.

26

Z Formal Specification

Modelling Concurrent Systems

- Formally, a transition system TS is a tuple (S, Act, $\rightarrow$, I, AP, L) such that:
- S — is the set of states
- Act — is the set of actions
- $\rightarrow \subseteq$ S x Act x S — is the transition relation (source state, action and target state)
- I $\subseteq$ S — is the set of initial states
- AP — is a set of atomic propositions
- L: S $\rightarrow \mathbb{P}$ AP — is a labelling function
  - The transition (s, a, s′) is written as $s \xrightarrow{a} s'$
  - L(s) are the atomic propositions in AP that are satisfied in state s.

27

Z Formal Specification

Modelling Concurrent Systems

- A concurrent system consists of multiple processes executing concurrently.
- If a concurrent system consists of n processes where each process $proc_i$ is modelled by a transition system TS_i, then the concurrent system may be modelled by a transition system (|| is the parallel composition operator): $TS = TS_1 || TS_2 || \ldots || TS_n$
- There are various operators used in modelling concurrency with transition systems, including operators for interleaving, communication via shared variables, handshaking and channel systems.

28

Z Formal Specification

Temporal Logic

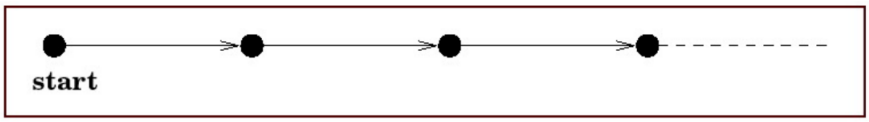
Temporal logic is concerned with the expression of properties that have time dependencies, and the various temporal logics express facts about the past, present and future.

Temporal logic has been applied to specify temporal properties of natural language, Artificial Intelligence and the specification and verification of program and system behavior.

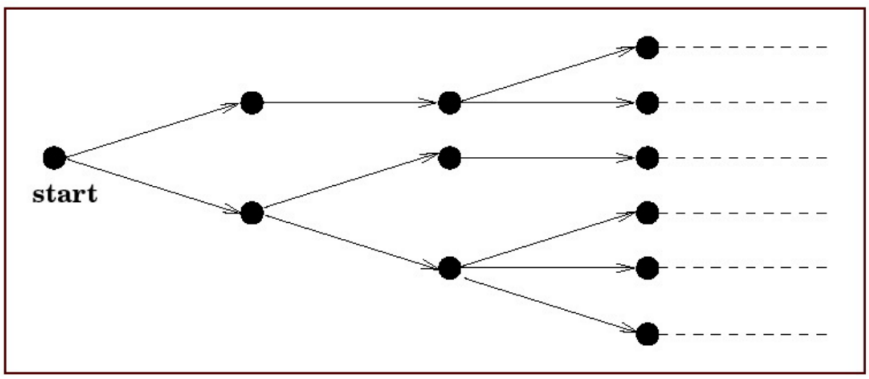
The statements made in temporal logic can have a truth-value that varies over time.

The two main types of temporal logics (linear time and branching time logics .

29



linear time logics



branching time logics

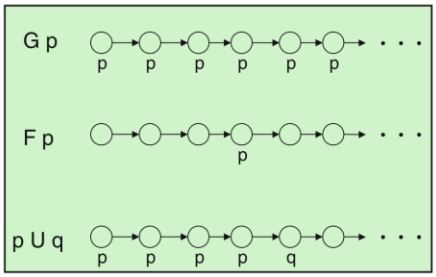
30

Z Formal Specification

Linear temporal logic (LTL) is a modal temporal logic that can encode formulae about the future of paths (e.g. a condition that will eventually be true).

Operator	Description
F_p	p holds sometime in the future
G_p	p holds globally in the future
X_p	p holds in next time instant
$p \cup q$	p holds until q is true

Basic temporal operators



LTL operators

31

Z Formal Specification

For example, consider how the sentence

"This microwave does not heat up until the door is closed"

is expressed in temporal logic.

This is naturally expressed with the until operator p U q as follows:

¬Heatup U DoorClosed

32

Z Formal Specification

Computational tree logic (CTL) is a branching time logic, which means that its model of time is a tree-like structure in which the future is not determined, and so there are many paths in the future such that any of them could be an actual path that is realized.

Computational tree logic can express many properties of finite state concurrent systems.

Each operator of the logic has two parts, namely the path quantifier(A—"every path", E—"there exists a path") and the state quantifier (F, P, X, U)

33

CTL temporal operators

Operator	Description
Aφ (*all*)	φ holds on *all* paths starting from the current state
Eφ (*exists*)	φ holds on at least one path starting from the current state
Xφ (*next*)	φ holds in the next state
Gφ (*global*)	φ has to hold on the entire subsequent path
Fφ (*finally*)	φ eventually has to hold (somewhere on the subsequent path)
$\varphi \cup \psi$ (*until*)	φ has to hold until at some position ψ holds
φWψ (weak *until*)	φ has to hold until ψ holds (no guarantee ψ will ever hold)

For example, the following is a valid CTL formula that states that

"it is always possible to get to the restart state from any state":

AG(EF restart)

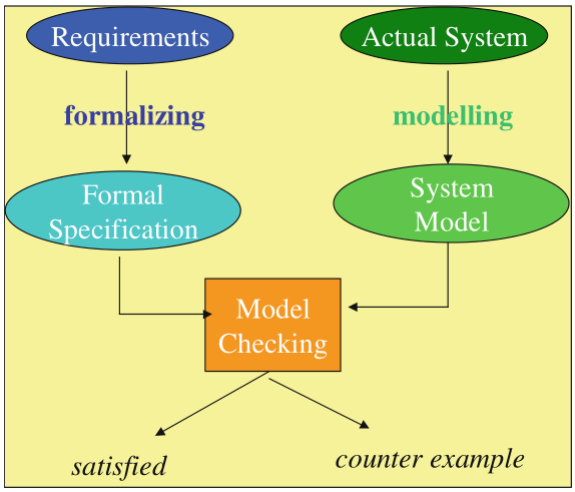
34

Z Formal Specification

Model Checking

- Model checking is an automated technique such that given a finite state model of a system and a formal property, then it systematically checks whether the property is true of false in a given state in the model.
- It is an effective technique to identify potential design errors, and it increases the confidence in the correctness of the system design.
- Model checking is a highly effective verification technology and is widely used in the hardware and software fields.
- Model checking tool: model checker (Spin, NuSMV, ...)

35



Model checking

36

Introduction to Z Formal Specification

37

Z Formal Specification

Refinement

- **Development often proceeds from an abstract specification to a detailed design. This process is called refinement.**
- **Both the specification and the design are models, but the specification is closer to the users' view, while the design is closer to an executable program.**

38

Z Formal Specification

What is refinement?

- A design is more concrete than a specification.
- A design is correct if it has all the properties of the specification; it usually has some additional properties as well.
- This relation is expressed precisely by logical implication: The predicate that describes the design must imply the predicate that describes the specification.

39

Z Formal Specification

What is refinement?

- Here is a trivial example. Our specification requires that we increase the value of x:
  - $x' > x$
- We propose to achieve this by adding one to x. Our design is this stronger predicate:
  - $x' = x + 1$
- The design should imply the specification:
  - $x' = x + 1 \Rightarrow x' > x$
- This implication is clearly true, so the refinement is correct. We can say that $x' = x+1$ refines $x' > x$.

40

Z Formal Specification

A refinement example

From sets to sequences

- Our abstract state is a set s of elements of type X.
- [X]

Abstract

$s : \mathbb{P}\, X$

- We define an abstract operation that stores an element in the set, using the set union operator.

AStore

$\Delta Abstract$

$x? : X$

$s' = s \cup \{x?\}$

41

Z Formal Specification

A refinement example

From sets to sequences

- Our concrete state is not a set but a sequence ss of elements of type X.

Concrete

$ss : \text{seq}\, X$

- Here is the concrete operation that stores an element in the sequence, using the concatenation operator.

CStore

$\Delta Concrete$

$x? : X$

$ss' = ss \frown \langle x? \rangle$

42

Z Formal Specification

A refinement example

Checking the refinement

A sequence is a function from natural numbers to elements, so the elements stored in the sequence are the range of this function. The range of the sequence must be the same as the set. $s = \operatorname{ran} ss \wedge s' = \operatorname{ran} ss'$

- This must be true before and after any operation, so equations appear for unprimed and primed variables.

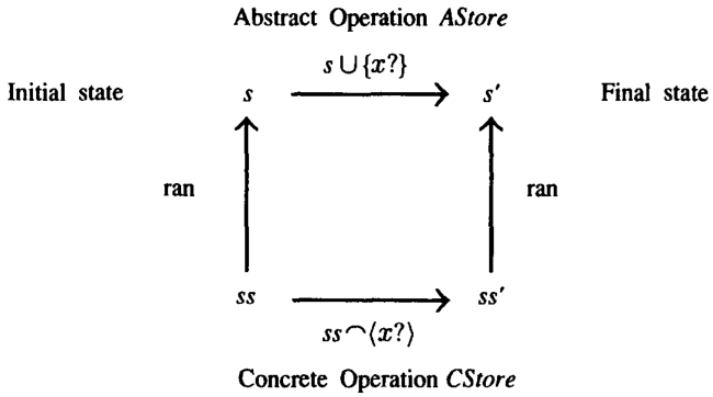
$$ss' = ss \frown \langle x? \rangle \wedge s = \operatorname{ran} ss \wedge s' = \operatorname{ran} ss' \Rightarrow s' = s \cup \{x?\}$$

- The refinement is correct if this predicate is true.

43

Z Formal Specification

A refinement example



Refinement of Astore by CStore

44

Checking the refinement

Here is the proof of the refinement:

Here is the proof of the refinement:

$$ss' = ss \frown \langle x? \rangle \land s = \operatorname{ran} ss \land s' = \operatorname{ran} ss' \Rightarrow s' = s \cup \{x?\} \qquad \text{[Given]}$$

$$\Leftrightarrow s' = s \cup \{x?\} \qquad \text{[Assume antecedent.]}$$

$$\Leftrightarrow \operatorname{ran} ss' = s \cup \{x?\} \qquad \text{[Antecedent } s' = \operatorname{ran} ss'.\text{]}$$

$$\Leftrightarrow \operatorname{ran}(ss \frown \langle x? \rangle) = s \cup \{x?\} \qquad \text{[Antecedent } ss' = ss \frown \langle x? \rangle.\text{]}$$

$$\Leftrightarrow \operatorname{ran} ss \cup \operatorname{ran}\langle x? \rangle = s \cup \{x?\} \qquad \text{[Law about ran}(s \frown t).\text{]}$$

$$\Leftrightarrow \operatorname{ran} ss \cup \{x?\} = s \cup \{x?\} \qquad \text{[Law about ran}\langle x? \rangle.\text{]}$$

$$\Leftrightarrow s \cup \{x?\} = s \cup \{x?\} \qquad \text{[Antecedent } s = \operatorname{ran} ss.\text{]}$$

$$\Leftrightarrow true \qquad [e = e \Leftrightarrow true.]$$

This concludes the proof of correctness for this refinement.

45

Z Formal Specification

Formal reasoning

- Formal reasoning plays somewhat the same role for mathematical models that testing does for code.
- Just as you can experiment with code by running it, you can investigate the behavior of a nonexecutable prototype by reasoning.
- You can check important system properties before you write a single line of code.

46

Z Formal Specification

Formal reasoning

- **We can use formal reasoning to validate a mathematical model against requirements.**
- **A model is valid if its properties satisfy the intent of the requirements.**
- **It can show that a detailed design is a correct refinement of a more abstract specification**
- **It can verify that code implements its specification.**
- **An exercise in formal reasoning is sometimes called a proof.**
- **The proof of various properties about the programs increases confidence in its correctness.**

47

Z Formal Specification

Calculation and proof

- **An exercise in formal reasoning is a kind of calculation. The arithmetic calculations you learned in school are examples of formal reasoning.**
- **E.g. A train moves at a constant velocity of sixty miles per hour. How far does the train travel in four hours?**
- **To solve this problem, you calculate the distance by multiplying velocity and time. We can express the problem in Z:**

48

Z Formal Specification

Calculation and proof

$$\begin{array}{|l} distance, velocity, time : \mathbb{N} \\ \hline distance = velocity * time \\ velocity = 60 \\ time = 4 \end{array}$$

- The problem is to find distance, which is implicit in the other predicates. We calculate the solution in steps. Here is the solution written out in full detail:

$$\begin{aligned} distance &= velocity * time && [Definition] \\ &= 60 * time && [velocity = 60] \\ &= 60 * 4 && [time = 4] \\ &= 240 && [\text{Arithmetic}] \end{aligned}$$

- This calculation is a formal proof of the predicate distance = 240 (miles).

49

Calculation and proof

$$\begin{array}{|l} philip : PERSON \\ adhesives, materials, research, manufacturing : \mathbb{P}\, PERSON \\ \hline adhesives \subseteq materials \\ materials \subseteq research \\ philip \in adhesives \end{array}$$

- A formal proof of the predicate philip $\in$ research is

$$\begin{aligned} philip \in adhesives && [\text{Definition}] \\ \subseteq materials && [\text{Definition}] \\ \subseteq research && [\text{Definition}] \end{aligned}$$

50

Z Formal Specification

Checking specifications

- We can use formal reasoning to check our work for certain kinds of errors and oversights.
- In the course of writing a complex specification it is not difficult to write formulas that conflict. This error is called inconsistency.
- There are systematic ways to apply formal reasoning to discover certain kinds of inconsistencies.

51

Z Formal Specification

Checking specifications

- If the specification is consistent, this initial state must satisfy the state invariant, so the predicate ∃ State • Init must be true. This predicate is called the Initialization Theorem.
- The precondition investigation of an operation describes all the initial states where the operation is defined. The precondition of operation schema Op on state schema S is described by the schema expression ∃ S' • Op: "There exists a final state that satisfies the predicate of the operation schema."

52

Z Formal Specification

Tools for Checking specifications

- **CADIZ is an automated checker and typesetter for Z specifications.**
- **CZT, tools for editing, type checking and animating formal specifications written in the Z specification language**

53

Z Formal Specification

Industrial Applications of Z

The Z specification language is one of the more popular formal methods, and it has been employed for the formal specification and verification of safety critical software.

IBM use Z on the CICS (Customer Information Control System) project at its plant in Hursley, England.

Logica employed Z for the formal verification of a smart card-based electronic cash system (the Mondex smart card) in the early 1990s.

Computer Management Group (CMG) employed Z for modelling data and operations as part of the formal specification of a movable barrier.

54