Introduction
Formal Methods
Lecture 1

Farn Wang
Dept. of Electrical Engineering
National Taiwan University

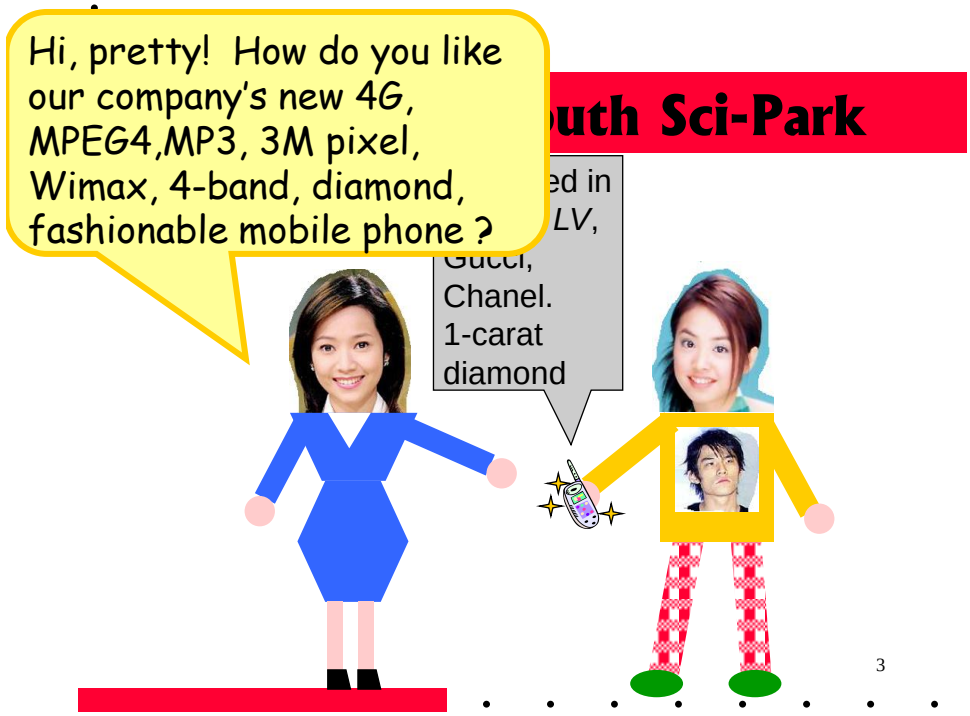
1

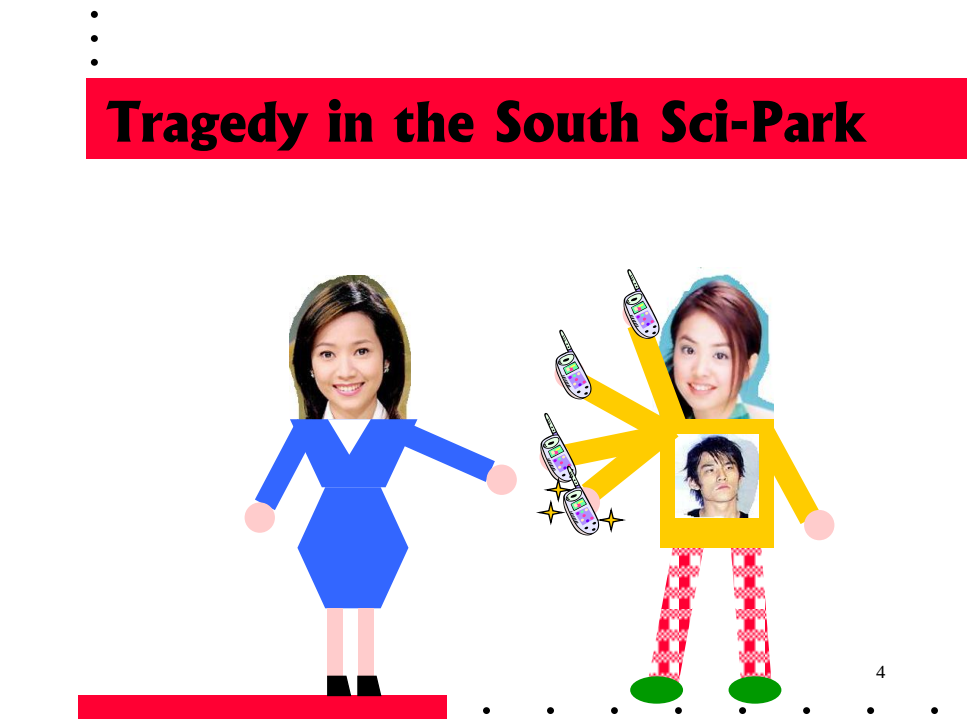
Specification & Verification ?

- Complete & sound specifications.
- Reducing bugs in a system.
- Making sure there are very few bugs.

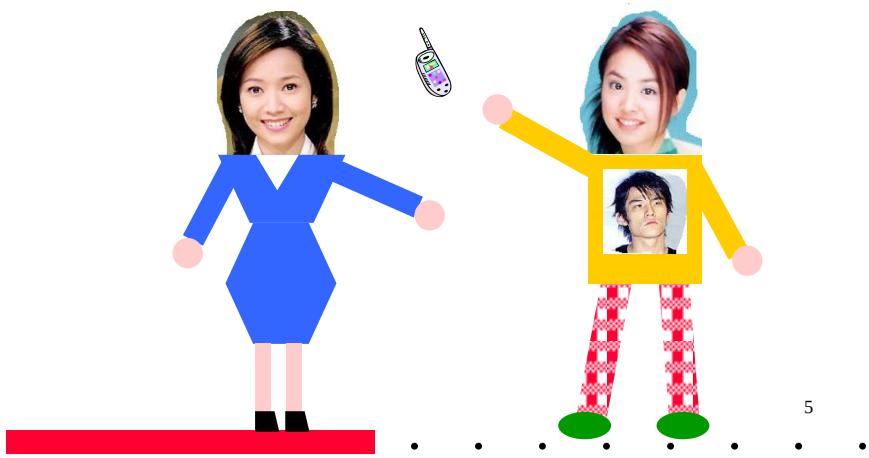
Very difficult!
Competitiveness of high-tech industry!
A way to survive for the students!
A way to survive for Taiwan!

2


Hi, pretty! How do you like our company's new 4G, MPEG4,MP3, 3M pixel, Wimax, 4-band, diamond, fashionable mobile phone ?
uth Sci-Park
ed in
LV,
Chanel.
1-carat
diamond
3

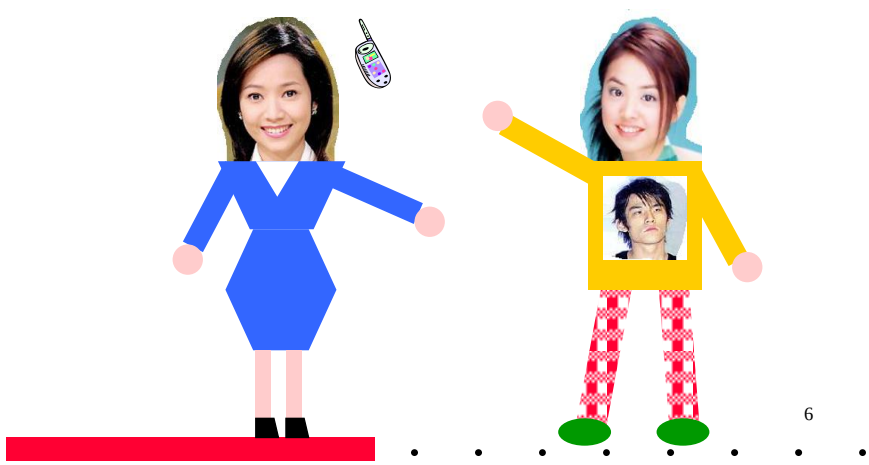

Tragedy in the South Sci-Park
4

Tragedy in the South Sci-Park



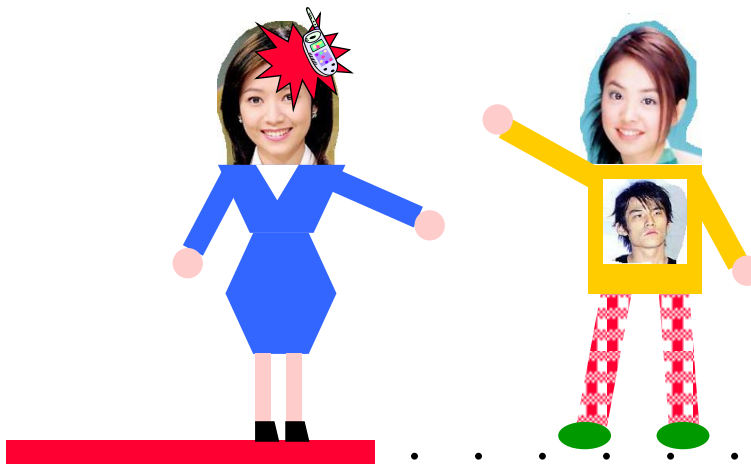
5

Tragedy in the South Sci-Park



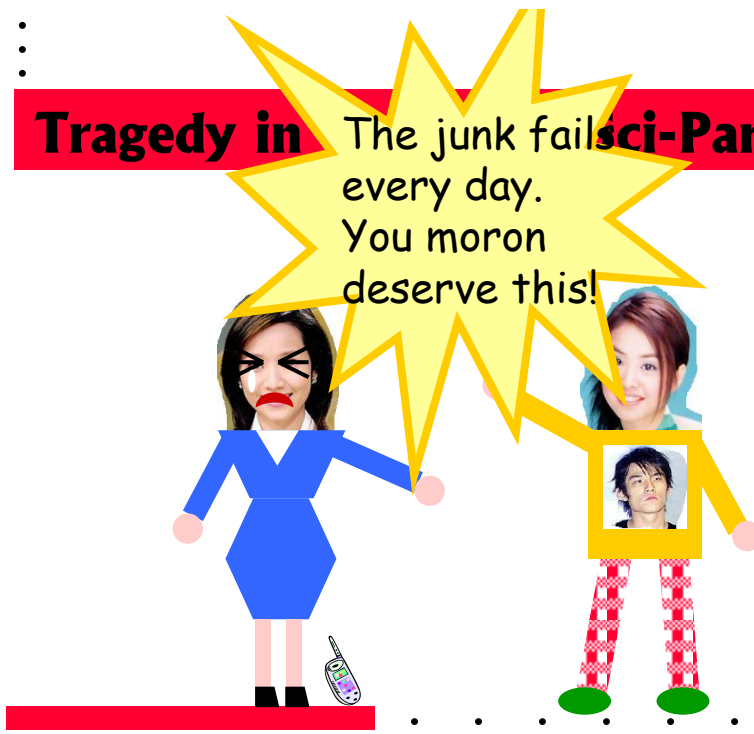
6

Tragedy in the South Sci-Park



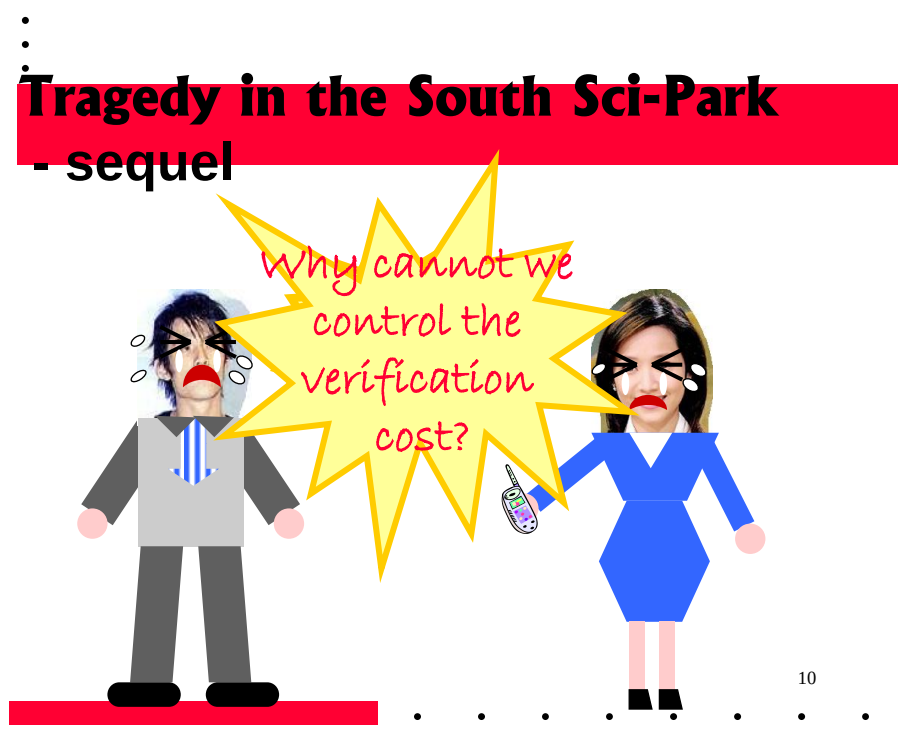
7

Tragedy in Sci-Park



8


-Park
Boss! I've had no slee for 3 mont testing the piece!
Good job! You just wasted half the budget
My wife is divorcing me!
Police just called me to bail out my kid.
9


Tragedy in the South Sci-Park
- sequel
Why cannot we control the verification cost?
10

Specifications, descriptions, & verification

- **specification:**
  - The user's requirement
- **description (implementation):**
  - The user's description of the systems
  - No strict line between description and specification.
- **verification:**
  - Does the description satisfy the specification ?

11

Formal specification & automated verification

- formal specificaton:
  - specification with rigorous mathematical notations
- automated verification:
  - verification with support from computer tools.

12

Why formal specifications ?

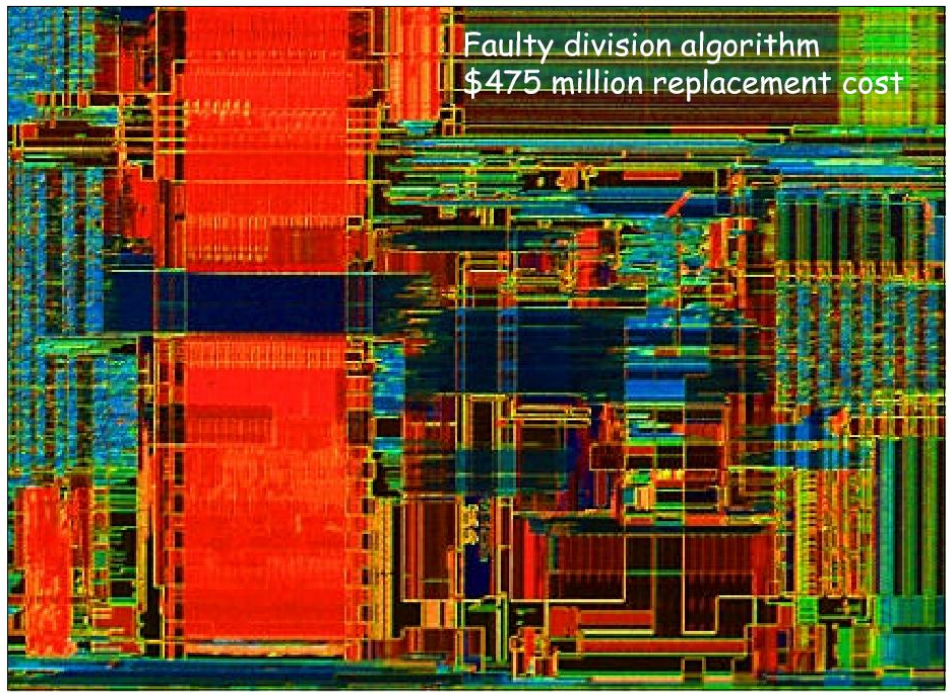
- to make the engineers/users understand the system to design through rigorous mathematical notations.
- to avoid ambiguity/confusion/misunderstanding in communication/discussion/reading.
- to specify the system precisely.
- to generate mathematical models for automated analysis.

13

Why automated verification ?

- to somehow be able to verify complexer & larger systems
- to liberate human from the labor-intensive verification tasks
  - to set free the creativity of human
- to avoid the huge cost of fixing early bugs in late cycles.
- to compete with the core verification technology of the future.

14



Pentium Bug (1/4)

Floating-point division

- expected precision up to 18 positions
- in practice, only 4 positions
- Pentium 60MHz、90MHz
- Example:

  5505001 / 294911

  wrong answer: 18.66600093

  expected answer: 18.6665197

16

Pentium Bug (2/4)

- Only for very few number pairs
- reproducible!
- affecting large scientific computations, statistics applications, engineering computations, spreadsheet, simulation, …
- may affect applications compiled with the CPU.

17

Pentium Bug (3/4)

- discovered by Dr. Thomas R. Nicely at Lynchburg College
  - nicely@acavax.lynchburg.edu
- announced in Compuserve on 10/30/1994
- printed in media on 11/7/1994
- fixed in mid 1994, but Intel insisted
  - new chips scheduled to major customers at the end of the year.
  - no replacement unless bug effects proved individually

18

Pentium Bug (4/4)

- triggered a wave of research in formal verification
- Intel maintained a large team of formal method. Until a few years ago, its size was a secret.
- Now we believe computation theory could be useful.
  - Grants and funds poured in.
  - Timely achievements in theory and tools.

19

THE "BUG" HEARD 'ROUND THE WORLD (1/4)

Discussion of the Software Problem Which Delayed the First Shuttle Orbital Flight

John R. Garman

Deputy Chief

Spacecraft Software Division

NASA, Johnson Space Center

Houston, Texas

Aug 24, 1981

ACM SIGSOFT Software Engineering Notes, Vol. 6, Nr. 5, Oct, 1981

20

THE "BUG" HEARD 'ROUND THE WORLD (2/4)

Discussion of the Software Problem Which Delayed the First Shuttle Orbital Flight

- 4/10/1981, 20 mins before the first launch of space shuttle, the 5th backup computer could not initialize.
- 4 General Purpose Computer (GPC) and 1 backup computer
- FO/FS fault-tolerant
  - one-fault-operate (still can vote)
  - two-fault-safe (still can return safely)

21

THE "BUG" HEARD 'ROUND THE WORLD (3/4)

Discussion of the Software Problem Which Delayed the First Shuttle Orbital Flight

- Software on the GPCs and the backup were developed by different teams.
- Cyclic processing
- Before the launch, the program on the GPCs had run for 30 hrs without problems.

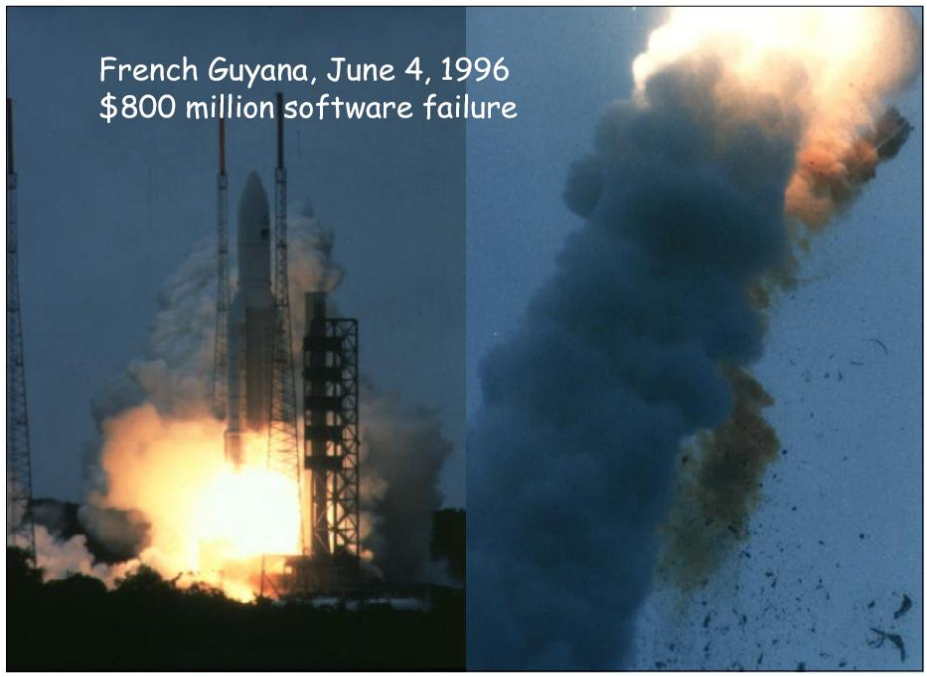
22

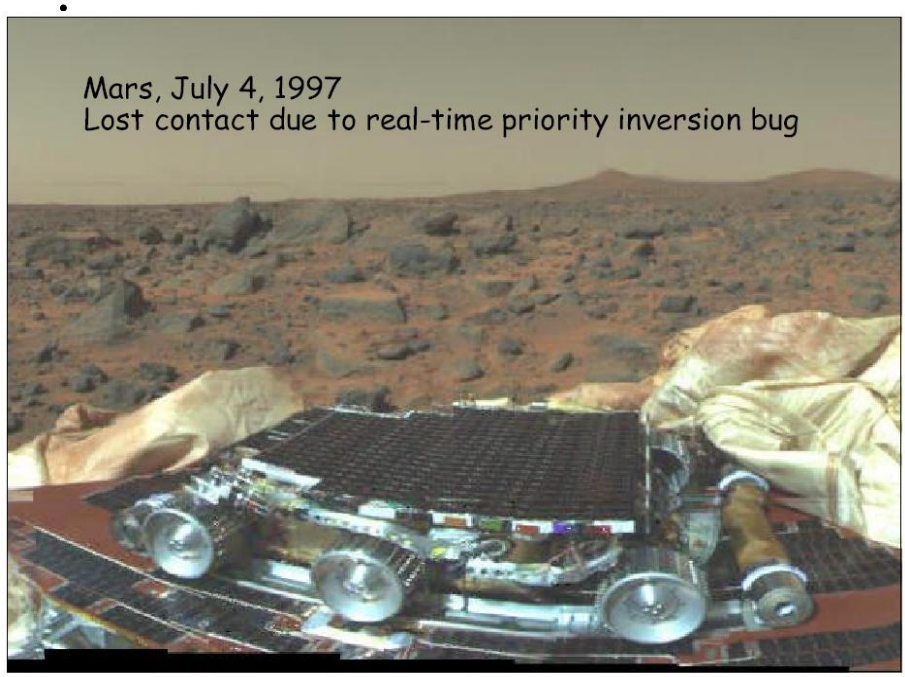
THE "BUG" HEARD 'ROUND THE WORLD (4/4)

Discussion of the Software Problem Which Delayed the First Shuttle Orbital Flight

- 1 hr later, IBM dumped the memory of the GPCs and found out a software bug in timing synchrony.
- Processes in the GPCs were out-of-phase.
- The backup could not get the out-of-phase signal and claimed the GPCs were faulty.

23





Therac-25 Incidents

- Medical linear accelerator by AECL
- Computer-controlled (DEC PDP-11)
- Dual modes of X-ray and electron beams
- Successor to Therac-20 and Therac-6 by AECL and CGR
- available in late 1982
- 11 Therac-25 were installed

6 accidents with death and serious injuries from 6/1985 to 1/1987

Therac-25 incidents (continued, 2)

- **Independently developed by AECL after breaking up with CGR**
- **A fault-tree safety analysis was performed with the assumption that software was correct.**
- **Controlled by legacy software from Therac-20 and Therac-6**
  - Therac-20 and -6 only used computer for convenience
  - Get rid of hardware interlock since software never went wrong with Therac-20 and Therac-6
    - ***In fact, most software errors of Therac-20 and Therac-6 had been masked by hardware interlocks.***

27

Therac-25 incidents (continued, 3)

- **Error message happened so often that technicians thought they were normal.**
- **Most of the errors did not hurt.**
- **The AECL said**
  - "Improper scanning was not possible!"
  - "This incident was never reproted to AECL prior to this date …" (after 10 months of a filed lawsuit)

28

Therac-25 incidents (continued, 4)

- **On May 2, 1986, FDA declared Therac-25 defective and demanded CAP.**
- **AECL remedied something and claimed that Therac-25 was 10,000 times safer.**
- **FDA believed them.**
- **Software errors have been identified in all these six admitted accidents.**
- **Finally, the hardware interlocks were put back in on Feb. 2, 1987.**

29

Therac-25 incidents (continued, 5)

- **Worst accidents series in 35-year history of medical accelerator**
- **References:**
  - N. Leveson, C.S. Turner, *An investigation of theTherac-25 accidents*, IEEE Computer, Vol. 26, Nr. 7, July 1993, pp.18-41

30

GOVERNMENT NEWS GCN July 13, 1998

Software glitches leave Navy Smart Ship dead in the water

Gregory Slabodkin, GCN Staff

"Using Windows NT, which is known to have some failure modes, on a warship is similar to hoping that luck will be in our favor," DiGiorgio said.

Some more bugs (1)

- *Mars climate orbiter smashed into the planet instead of reaching a safe orbit ($165M), 1999*
  - **Failure to convert English measures to metric values**
  - **Software shut the engine off 100ft above the surface.**
- *US Vicennes mistook airbus 320 for a F-14 and shot it down, 1st Gulf War, 1988.*
  - **290 people dead**
  - **Why: Software bug - cryptic and misleading output displayed by the tracking software**

Some more bugs (2)

Failure of the London Ambulance Service on 26 and 27 November 1992

- **Load increased**
- **Emergencies accumulated**
- **System made incorrect allocations**
  - **more than one ambulance being sent to the same incident**
  - **the closest vehicle was not chosen for the emergency**
- **At 23:00 on October 28 the LAS eventually instigated a backup procedure, after the death of at least 20 patients**

Some more bugs (3)

- British destroyer H.M.S. Sheffield; sunk in the Falkland Islands war
  - **ship's radar warning system software allowed missile to reach its target**
- An Air New Zealand airliner crashed into an Antarctic mountain
- North American Aerospace Defense Command reported that the U.S. was under missile attack;
  - **traced to faulty computer software - generated incorrect signals**
- Manned space capsule Gemini V missed its landing point by 100 miles;
  - **software ignored the motion of the earth around the sun**

["*The development of software for ballistic-missile defense*," by H. Lin, Scientific American, vol. 253, no. 6 (Dec. 1985), p. 48]

Some more bugs (4)

- An error in an aircraft design program contributed to several serious air crashes

  ["Software Engineering: Report on a Conference sponsored by the NATO Science Committee, Brussels NATO Scientific Affairs Division," 1968, p. 121]

- Dallas/Fort Worth air-traffic system began spitting out gibberish in the Fall of 1989 and controllers had to track planes on paper

  ["Ghost in the Machine," Time Magazine, Jan. 29, 1990. p. 58]

35

Some more bugs (5)

- **F-18 fighter plane crashed**
  - due to a missing exception condition

  [ACM SIGSOFT Software Engineering Notes, vol. 6, no. 2]
- **F-14 fighter plane was lost**
  - to uncontrollable spin, traced to tactical software

  [ACM SIGSOFT Software Engineering Notes, vol. 9, no. 5]
- **Chicago cat owners were billed $5 for unlicensed dachshunds.**
  - A database search on "DHC" (for dachshunds) found "domestic house cats" with shots but no license

  [ACM SIGSOFT Software Engineering Notes, vol. 12, no. 3]

36

Some more bugs (6)

- CyberSitter censors "menu */ #define"
  - **because of the string "nu...de“**

    [Internet Risks Forum NewsGroup (RISKS), vol. 19, issue 56]

- London‘s Docklands Light Railway – train stopped in the middle of nowhere due to future station location programmed in software

37

Some more bugs (7)

CNN.com

- Russia: Software bug made Soyuz stray.
  - **STAR CITY, Russia(AP) – A computer software error likely sent a Russian spacecraft into a rare ballistic descent that subjected the three men on board to check-crushing gravity loads that made it hard to breathe, space experts said Tuesday.**
- Korean Air crashed in Guam and killed 228 people.
  - **A poorly programmed ground-based altitude warning system**
- Faulty software in anti-lock brakes forced the recall of 39,000 trucks and tractors and 6,000 school buses in 2000.
- Mars Polar lander, $165M, 1999.
  - **Software shut the engines off 100 feet above the surface.**
- **US$59.5 billions loss in economy, 0.6%GDP, April 27, 2003**

38


$4 billion development effort
> 50% system integration & validation cost
2,500,000+1,500,000 lines of codes (most in Ada)

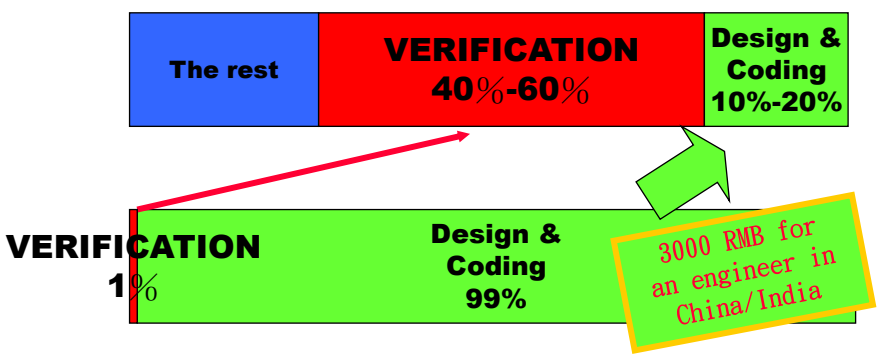

400 horses
100 microprocessors
M NH 2519

Bugs in complex software

- They take effects only with special event sequences.
  - the number of event sequences is factorial and super astronomical!
- It is impossible to check all traces with test/simulation.

41

Budget appropriation



Training in Taiwan College

42

Automated verification - FV+simulation+testing

Competitiveness in Europe/America

- Intel、Motorola、Ericsson…
- Cadence、Synopsys、…Graphics、…

Where is Taiwan's future ?

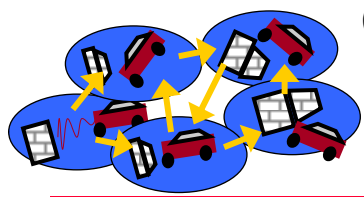
3000 RMB for an engineer in China/India

D…Taiwan industry

- …ast/mass production to the cus…
- very little automated verification tools for large-scale system development.

43

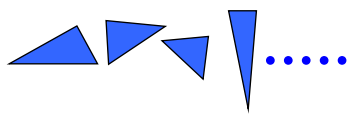
Three techniques in verification

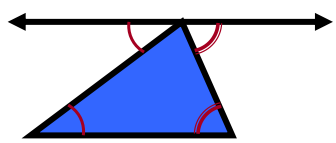


- **Testing (real wall for real cars)**
  - Expensive
  - Low coverage
  - Late in development cycles
- **Simulation(virtual wall for virtual cars)**
  - Economic
  - Low coverage
  - Don't know what you haven't seen.
- **Formal Verification (virtual car checked)**
  - Expensive
  - Functional completeness
    - ◆100% coverage
  - Automated!
    - ◆With algorithms and proofs.

44

Sum of the 3 angles = 180 ?





- **Testing (check all Δs you see)**
  - Expensive
  - Low coverage
  - Late in development cycles
- **Simulation (check all Δs you draw)**
  - Economic
  - Low coverage
  - Don't know what you haven't seen.
- **Formal Verification(we prove it.)**
  - Expensive
  - Functional completeness
    - ◆100% coverage
  - Automated!
    - ◆With algorithms and proofs.

45

Promise of Formal Verification (FV) ?

Use mathematics to prove the correctness of system designs!

Advantage:

- **Functional completeness**
  - Mechanical & exhaustive exploration of all cases
- **Automated verification**
  - Cut down verification cost
  - Relieve us of mechanical verification tasks

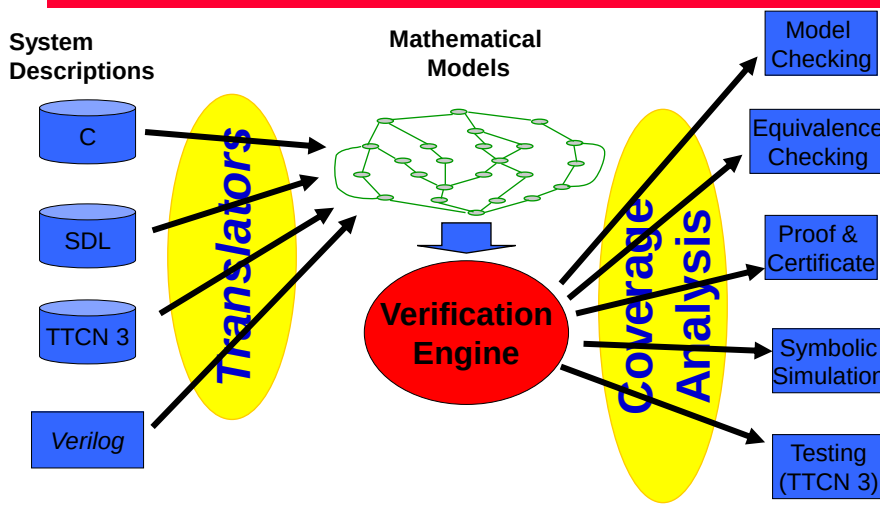
46

Integration of the verification techniques

- **Formal verification is still infeasible in practice.**
- **At the moment, we rely on the following.**
  - sound descipline in system design
    - software engineering
  - simulation/testing for easy-to-find bugs
  - formal verification (automated or manuel) for early bugs or safety-critical systems.

47

Verification Infrastructure



48

Major desciplines (1/2)

- mode-checking
  - AT&T、CMU、UC-Berkeley、Stanford、North Carolina、Cornell、Intel、Cadence-Berkeley
- theorem-proving
  - UT-Austin、SRI、MIT、MITRE、XEROX-PARC

49

Major desciplines (2/2)

- process algebra
  - + Oxford、Cambridge、Edinburgh、Uppsala
- formal methods
  - + Oxford、IFAD、IBM UK、CRI、Formal Systems Ltd、Praxis、CWI、VERILOG

50

Major conferences

- CAV (Computer-Aided Verification)
- FME (Formal Method Europe)
- AMAST (Algebraic Methodo. & Sw Tec.)
- TACAS (Tools & Algorithms for CAS)
- ATVA
- SAS (Static Analysis Symposium)
- FORTE(Formal Description Technique)

51

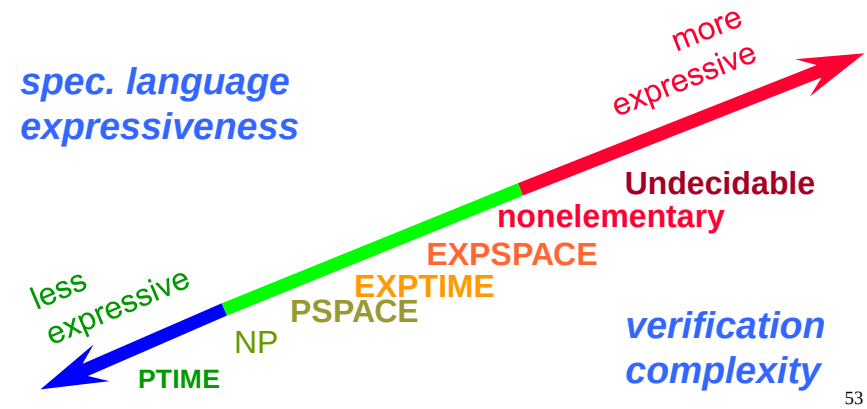
Major conferences - with related sessions

* FOCS
* LICS
* POPL
* STACS
* RTAS
* CONCUR
* SAS
* FTRTFT
* STOC
* PODC
* MFCS
* RTSS
* ICALP
* FORTE
* CADE
* RTCSA

52

Theory spectrum

expressiveness vs verification complexity

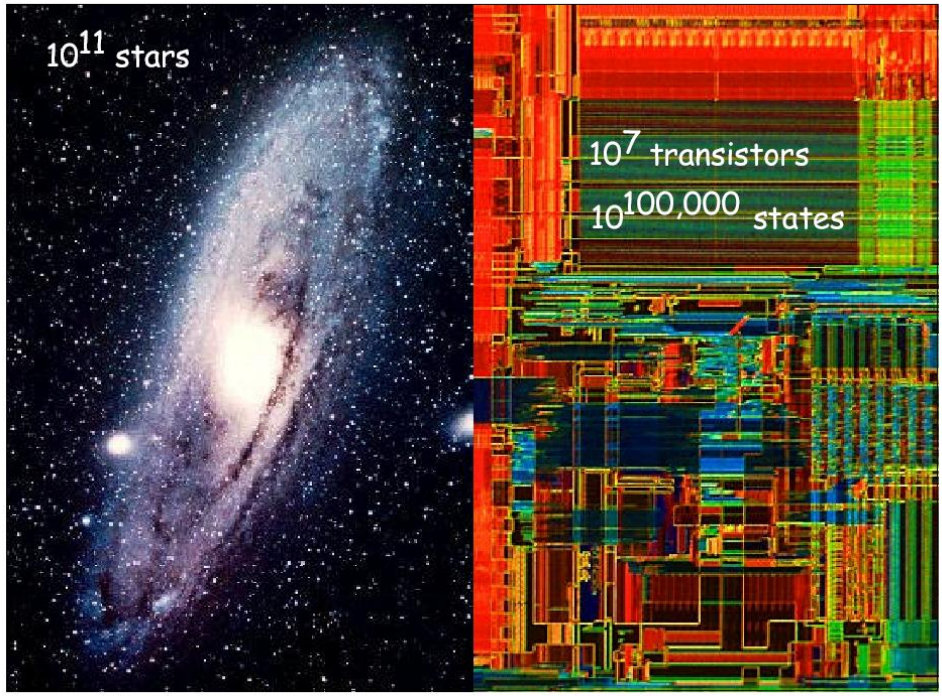


53

Sources of verification complexity

- Number of system states – astronomical
  - the values of all variables
  - the content of program counters
  - the messages in communication channels
- unbounded variable value ranges
- non-regular behaviors
  - inexpressible with finite-state automata.
  - stack, queue, polynomials, arithmetic, induction

54



The state of the art

- balance between
  - specification expressiveness and
  - verification complexity.
- non-regular systems → proof checking
  - sound proof plans by engineers
  - interaction between engineers and tools
- regular systems
  - huge time/space complexity
  - efficient algorithms

56

Deciplines

- temporal logics
- automaton theory
- process algebrae
- first-order logics
- Petri-Net
- formal methods
- graphical languages: statechart, modechart
- formal semantics

57

Deciplines - temporal logics

- a branch of modal logics
  - transitions between possible worlds
  - two modal operators: □, ◇
    - □: assert for all possible worlds
    - ◇: assert for one possible world
  - the model of modal logics is Kripke structure

directed graphs

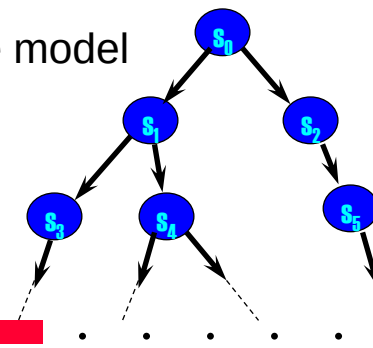
- In temporal logics,
  - □ : from now on, true for all states
  - ◇: from now on, true for a state

58

Deciplines - temporal logics

- linear-time model

- branching-time model



∃ : there is a path

∀: for all paths

59

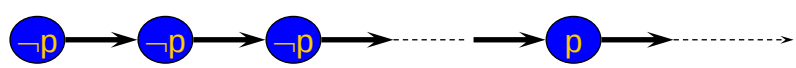
Deciplines - temporal logics

linear-time model &

PLTL (Propositional Linear Temporal Logic)

- eventually p will be true.

  ◇p : *liveness property*



★Amir Pnueli, 1996 Turing Award Winner

60

Deciplines - temporal logics

linear-time model & PLTL

- p and q will never be true the same time.

$\Box \neg (p \wedge q)$: *safety property*

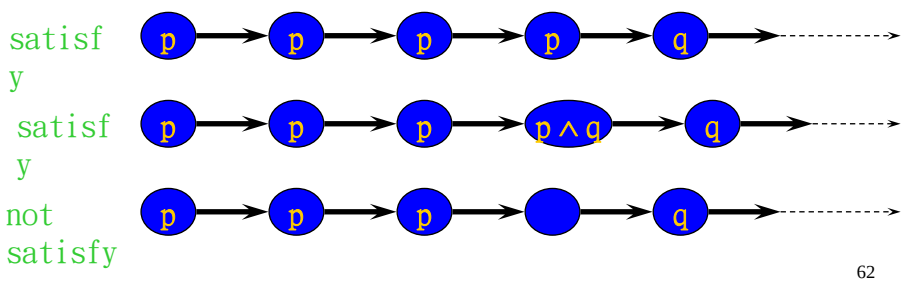
61

Deciplines - temporal logics

linear-time model and PLTL

- p is true until q is true (p until q)

pUq



62

Deciplines - temporal logics

- satisfaction relation
  - a linear-time model satisfies a PLTL formula.
- a formula defines a set of models.
- a formula is unsatisfiable if its set of models is empty.
- check the satisfiability of $P \wedge \neg S$
  - P is the implementation
  - S is the specification
  - The satisfiability problem is PSPACE-complete.

63

Deciplines - temporal logics

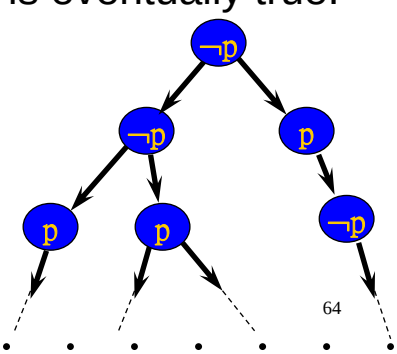
branching-time

& CTL (Computation-tree Logic)

- for all computations, p is eventually true.

$\forall\Diamond p$

Inevitibility



64

Deciplines - temporal logics

branching-time

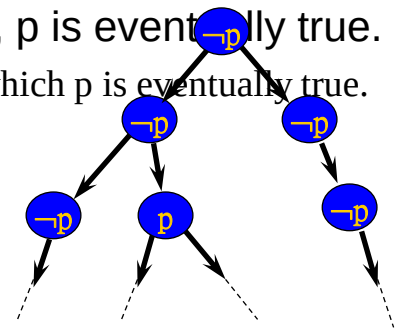
& CTL (Computation-tree Logic)

- for all computations, p is eventually true.
  - there is a path along which p is eventually true.

$\exists \Diamond p$

$\exists \neg p \boldsymbol{U} p$

Reachability



65

Deciplines - temporal logics

- satisfaction relation
  - a tree model satisfies a CTL formula.
- a CTL formula defines a set of models.
- a CTL formula is unsatisfiable if its set of model is empty.
- check the satisfiability of $P \wedge \neg S$
  - The satisfiability problem of CTL is deterministic EXPTIME-complete.

66

Deciplines - model-checking

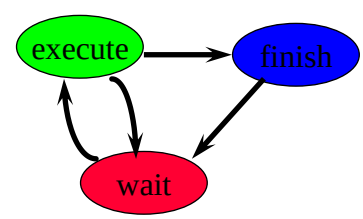
Given a model *M* and a temporal logic formula *P, does M satisfy P ?*

- Usually *M is a finite-state autoamta.*
- When P is a CTL formula, the model-checking problem is in PTIME.

67

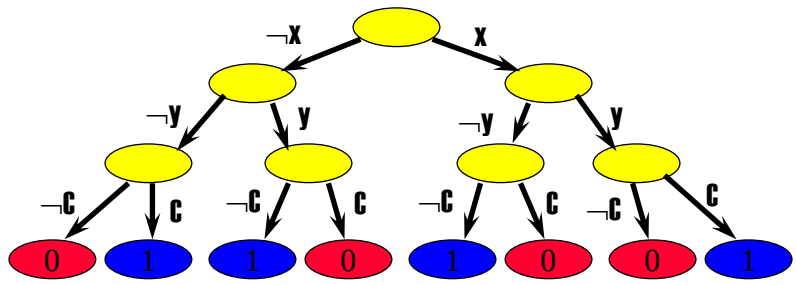
Deciplines - model-checking

- state-space analysis & exploration
  - state-space represented as a finite Kripke structure
  - nodes: system states, possible worlds,
  - arcs: state transitions
- regular behaviors
- huge state-spaces



68

Deciplines
- model-checking



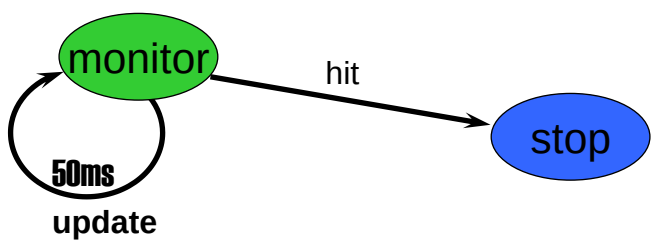
an automata
for a full adder $S = x \oplus y \oplus C$

69

Deciplines
- model-checking

Timed automata - regular behaviors

update the missile direction every 50ms
until the target is hit.

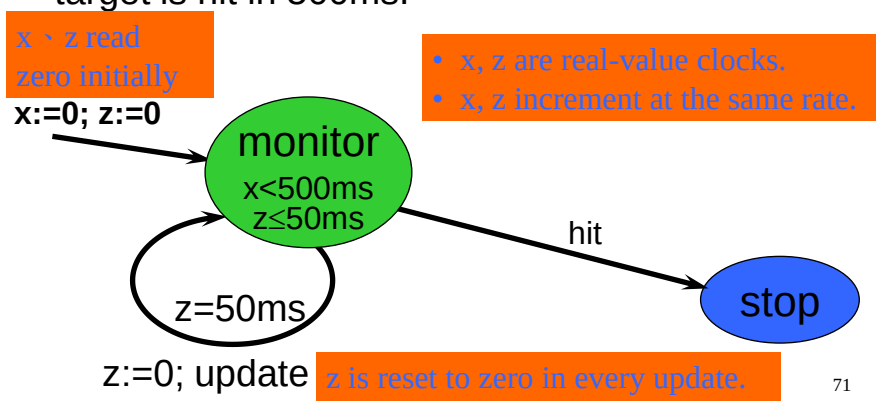


70

Deciplines
- model-checking

Timed automata - regular behaviors

update the missile direction every 50ms until the target is hit in 500ms.



71

Deciplines
- model-checking

TCTL (Timed CTL)

$\forall\Box$ **(monitor** $\rightarrow \forall\Diamond_{<500}$ **stop)**

$\forall\Box$ **(monitor** $\rightarrow$ **x.**$\forall\Diamond$ **(x** $<$ **500** $\wedge$ **stop))**

$\forall\Box$**x. (monitor** $\rightarrow \forall\Diamond$ **(x** $<$ **500** $\wedge$ **stop))**

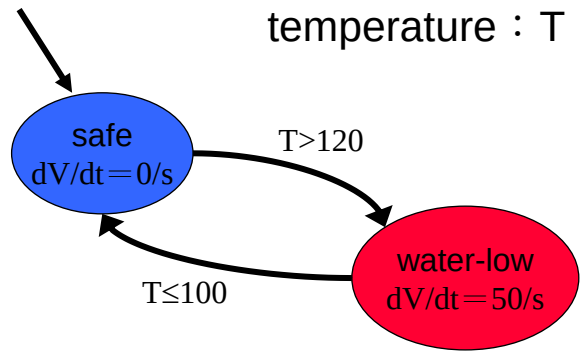
72

Deciplines
- model-checking

Hybrid automata

volume：V

temperature：T



specification:

∀□ **T＜130**

73

Deciplines
- process algebrae

- **CSP,** Communicating Sequential Processes
  - C.A.R. Hoare, Turing Award winner
  - *Communicating Sequential Processes*, Prentice-Hall, 1985
- **CCS,** Calculus of Communicating Systems
  - Robin Milner, Turing Award winner
  - *Communication and Cuncurrency*, Prentice-Hall, 1989
  - strong equivalence, observational equivalence, observational congruence

74

Deciplines - process algebrae

vending machine

$VM = (in5p \rightarrow choc \rightarrow VM \mid in2p \rightarrow toffee \rightarrow VM)$

– action set: αVM = {in5p, choc, in2p, toffee}

- models: traces
  - *in5p choc in5p*
  - *in5p choc in2p toffee in2p toffee*
- process: the behavior pattern of an object
  - in syntax, a set of rules
  - in semantics, a set of traces

75

Deciplines - process algebrae

operation on traces

- prefix : $x \rightarrow P$ (*x then P*)
  - a guarded command
- recursion : $P = (x \rightarrow P)$

  or $P = \mu X.\ (x \rightarrow X)$ （*least fixed-point*）
- choice : P = (P_1 | P_2)

76

Deciplines - process algebrae

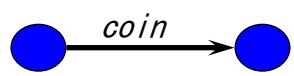
operations on traces

- **catenation:**
  〈coin,choc〉^〈coin,toffee〉=〈coin,choc,coin,toffee〉
- **restriction:**
  〈coin,choc,coin,toffee〉↾{coin}=〈coin,coin〉
- **head & tail**
  〈coin,choc,coin,toffee〉$_0$ = coin
  〈coin,choc,coin,toffee〉' =〈choc,coin,toffee〉
- **ordering**
  $s \leftrightarrows t = (\exists u.s\hat{}u = t)$
- **length:** # 〈coin,choc,coin,toffee〉=4

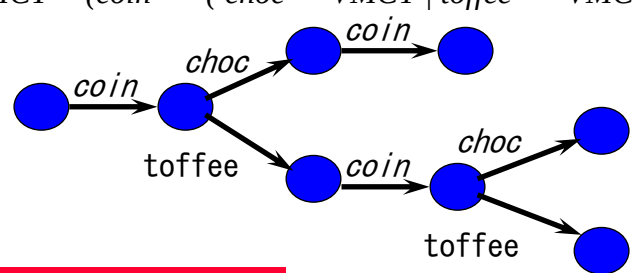
77

Deciplines - process algebra

- $(coin \rightarrow STOP_{\alpha VMS})$



- $VMCT = (coin \rightarrow (\ choc \rightarrow VMCT \mid toffee \rightarrow VMCT\ ))$

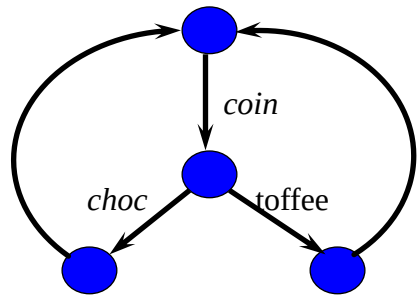


78

Deciplines - process algebra

Infinite behaviors

VMCT = (coin → (choc →VMCT | toffee → VMCT))



79

Deciplines - process algebrae

Operations on processes

concurrency: **P||Q**

- In**αP∩αQ**, all actions must be synchronized.
- ***(c→P)||(c→Q) = (c→(P||Q))***
- ***(c→P)||(d→Q) = STOP*** **; if c≠d**

  deadlock

80

Deciplines - process algebrae

Operations on processes

- nondeternimism: P ⊓ Q (P, Q same priority)

  P □ Q （P first）
- communications: (c!v→ P) || (c?v→ Q)
- concatenation: P;Q
- hiding: P/C
  - all actions in P from C are hidden.
  - C: a set of actions

Deciplines - process algebrae

specifications

- *tr* : a place-holder for any trace
- specification: a requirement for process

Example:

- At any time, the count of coins to the VM is greater than the count of chocolate pieces sold.

  NOLOSS =(#(tr ↑{choc}) ó #(tr ↑{coin}))
- Before a piece of chocolate is out, no coin will be input.

  FAIR =((tr↓coin) ó (tr↓choc) + 1)

  ↓:selection

Deciplines
- process algebra ***satisfaction***

- P **sat** S
  - P, a process； S, a trace specification
- Verification techniques
  - Proof-checking
    - laws of processes and traces
  - Model-checking
    - exploration in a finite space

83

Deciplines
- theorem-proving

- atoms, functions, predicates, ∨, ∧, ¬, ∃, ∀

  $\forall x\ (Man(x) \rightarrow Mortal(x))$

  Every man will die.
- *Satisfiability problem*: undecidable!
  - Resolution Principle - J.A. Robinson
  - proof-checking : mechanical theorem proving
  - computer support, human guidance

84

Deciplines - theorem-proving

Two decidable subclasses of 1st-order arithmetic

- Presburger Arithmetic, N, +, -, $\vee, \wedge, \neg, \exists, \forall$

  $\exists z \forall y\ (z<y \vee \exists x(x+z<y))$

  – elementary decision procedure
- 1st-order logic with p(x) and $\leq$

  $\forall y(p(y) \rightarrow \exists x\ (y \leq x \wedge\ q(x))$

  – nonelementary decision procedure

monadic predicate

85

Deciplines - theorem-proving

NqThm, Boyer & Moore

- *A Computational Logic Handbook* Academic Press, 1988
- quantifier-free, first-order, similar to pure-Lisp
- a very famous theorem-proving environment

example : (pp. 190-197 of the book)

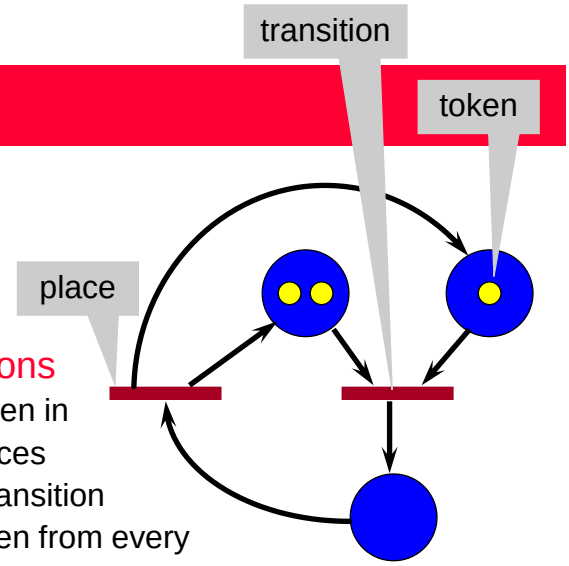
"a list = the reverse of the reverse of the list"

- wrong: (PROVE-LEMMA REVERSE-REVERSE (REWRITE)
  (EQUAL (REVERSE (REVERSE X)) X))
- correct: (PROVE-LEMMA REVERSE-REVERSE (REWRITE)
  (IMPLIES (PROPERP X)
  (EQUAL (REVERSE (REVERSE X)) X)))

86

Deciplines - Petri-Nets

- C.A. Petri (1972)
- finite # of places
- tokens
- finite # of transitions
  - enabled : if a token in every source places
  - firing: enabled transition consumes a token from every source places and put a token to all destination places.



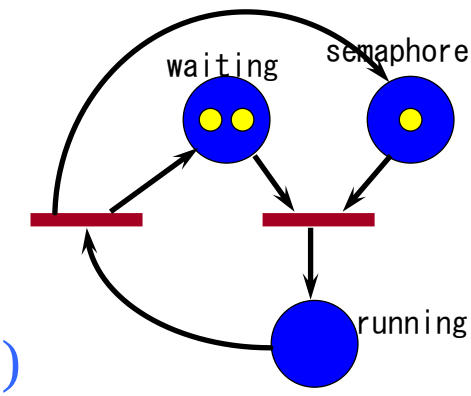
87

Deciplines - Petri-Nets

- equivalent to Karp & Miller's vector addition systems
- cannot detect the non-existence of events
- marking (state):

  places $\mapsto$ N

  (2,1,0)

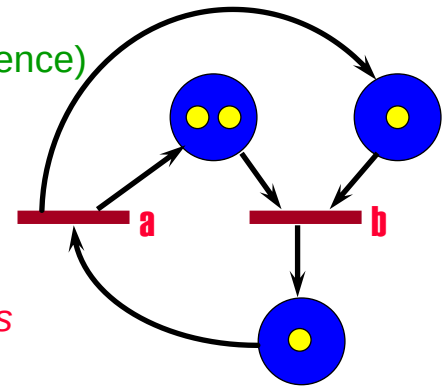


88

Deciplines - Petri-Nets

computation (state sequence)

- marking sequence
- connected with enabled transitions
  – *interleaving semantics*



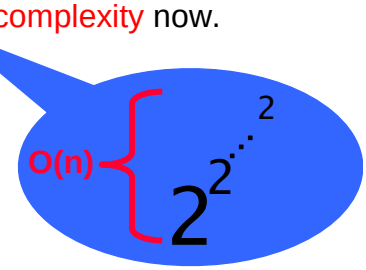
$$(2,1,0) \xrightarrow{b} (1,0,1) \xrightarrow{a} (2,1,0) \xrightarrow{b} (1,0,1) \xrightarrow{a} \cdots\cdots$$

89

Deciplines - Petri-Nets

Verification problems

- infinite markings from an initial marking
- Reachability problem:
  Is there a computation from marking M1 to marking M2 ?
  – decidable but non-elementary complexity now.
- Coverability problem
- Boundedness problem



Deciplines
- Formal Methods

- originated from the industry
- first, *VDM* (Vienna Development Method)
- Z notation
- RAISE (Rigorous Approach to Industrial Software Engineering)
- Estelle, from the ESPRIT SEDOS project
  - semantics defined with Petri-net
  - ISO OSI for computer network architecture standard.
- SDL: Specification & Description Language
  - CCITT Z1.00, for protocol specification

91

Deciplines
- Formal Methods *VDM*

- IBM research, Vienna, 1960s
- C.B.Jones & D.Bjærner made rigorous definitions in 1970-1982.
- widely accepted by the industry
  - with practical effect.
  - comparing with other academic work
- Many software tools
  - experimental or commercial
- stepwise refinement

92

Deciplines
- Formal Methods ***VDM proof***

stepwise refinement, 3 components

- direct definition
- implicit specification
  - model-oriented specification
    - set-theoretical notation
      - predicates, sets, relations, functions, sequences
- proof obligation

  direct definition→implicit specification

93

Deciplines
- Formal Methods ***VDM proof***

stepwise refinement, 3 components

- **direct definition**

  $sumn : N \rightarrow N$

  $sumn(n)$ $\quad$ *if* $n=0$ *then* 0 *else* $n + sumn(n-1)$
- **implicit specification**

  $sumn\ (n:N)\ \ r:N$

  post $r=n*(n+1)/2$
- **proof obligation**

  $n \in N \quad sumn(n) \in N \wedge post\text{-}sumn(n,sumn(n))$

94

Deciplines
- Formal Methods ***VDM proof***

stepwise refinement, 3 components

two rules

- **rule 1 :** $\dfrac{}{sumn(0)=0}$

- **rule 2 :** $\dfrac{n \in N;\ n \neq 0;\ sumn(n-1)=k}{sumn(n)=n+k}$

95

Deciplines
- Formal Methods ***VDM proof*** (1/2)

stepwise refinement, 3 components

From $n \in N$

1. *sumn(0) = 0*	Rule 1
2. *sumn(0)* $\in N$	1, N
3. *0=0*(0+1)/2*	N
4. *sumn(0) = 0*(0+1)/2*	*=-subs(3,1)*
5. *post-sumn(0,sumn(0))*	*post-sumn,4*
6. *sumn(0)* $\in N \wedge$ *post-sumn(0,sumn(0))*	$\wedge$ *-I(2,5)*

7. *from* $n \in N$, $sumn(n) \in N$, *post-sumn(n,sumn(n))*

96

Deciplines
- Formal Methods ***VDM proof*** (2/2)

stepwise refinement, 3 components

7.1	$n+1 \neq 0$	h7,***N***
7.2	$n+1 \in N$	h7,***N***
7.3	*sumn(n) = n*(n+1)/2*	*post-sumn,* ih7
7.4	*sumn(n+1) = n+1+n*(n+1)/2*	Rule 2(7.2,7.1,7.3)
7.5	$sumn(n+1) \in N$	7.4, ***N***
7.6	*sumn(n+1) = (n+1)*(n+2)/2*	7.4, ***N***
7.7	*post-sumn(n+1)=(n+1)*(n+2)/2*	*post-sumn,* 7.6
	infer $sumn(n+1) \in N \wedge$ *post-sumn(n+1,sumn(n+1))*	$\wedge$ *-I(7.5,7.7)*
infer $sumn(n) \in N \wedge$ *post-sumn(n,sumn(n))*		***N****-ind(6,7)*

97

Deciplines
- graphical languages

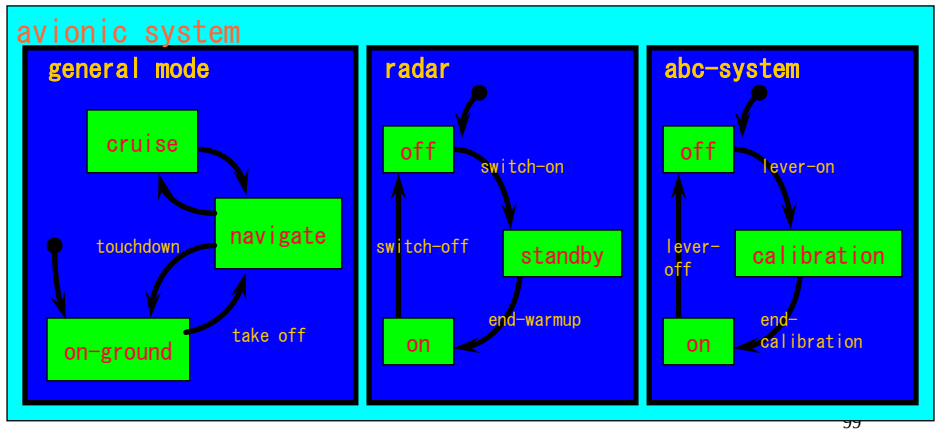
statechart

- David Harel, 1986
  - science of computer programming
- developed from automaton theory
- concurrent computations,
- discrete events
- nested modules

98

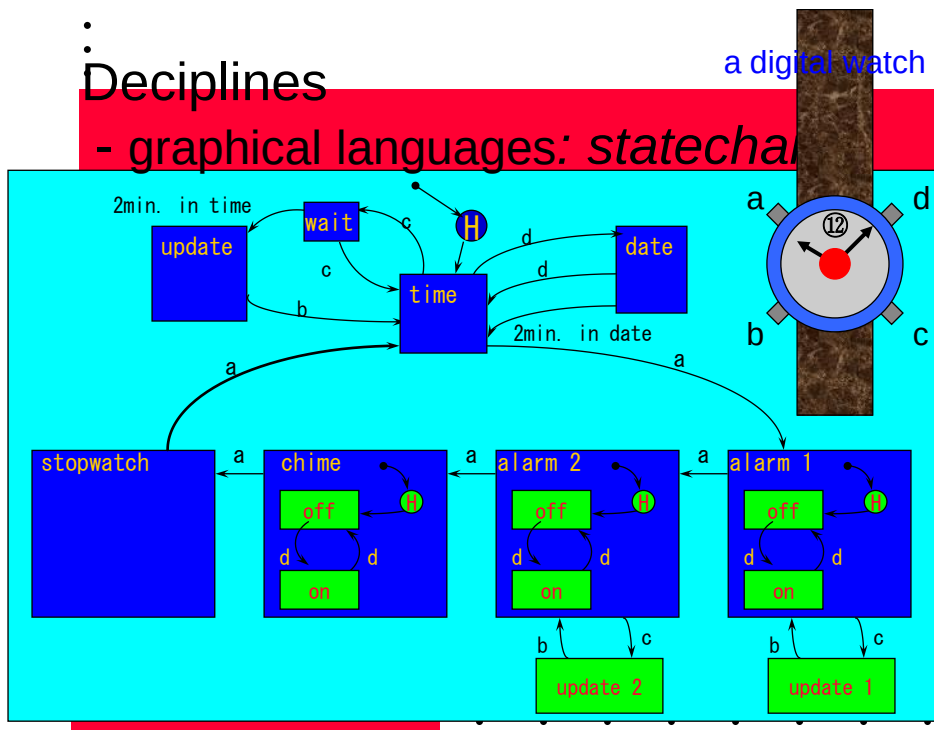
Deciplines
- graphical languages: *statechart*

Parallel modes for orthogonality



Deciplines
- graphical languages: *statecha*

a digital watch

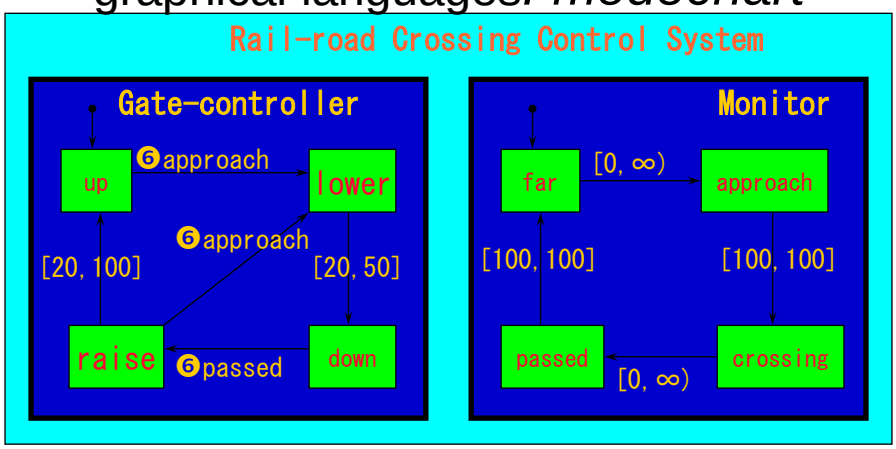


Deciplines
- graphical languages: *modechart*

- F. Jahanian, A.K. Mok, 1986
  - IEEE Transactions on SE
- extended from statechart with timing structures
- semantics defined with RTL
  - RTL（real-time logic）is also proposed by Jahanian & Mok (IEEE TC)

101

Deciplines
- graphical languages: *modechart*



102

Deciplines - language semantics

- operational semantics
  - with an abstract machine, like Java machine
- denotational semantics
  - treat program as functions, types, λ-calculus
- axiomatic semantics
  - relation between the pre-condition & post-conditions of programs
  - heavily used in verification

Deciplines - language semantics

Axiomatic Semantics

- **sequential programs**
  - E.W.Dijkstra, Turing Award
    A Discipline of Programming, Prentice-Hall, 1976
- **distributed programs**
  - Chandy、Misra的UNITY
    Parallel Program Design - A Foundation
    Addison-Wesley, 1988

Deciplines
- language semantics

Axiomatic Semantics

- **guarded-command language**

  $x,y := x+y,0 \ if \ y>0$

- **Fairness assumption，interleaving semantics**
- **program composition**
- **precondition**、**postcondition**

  $\{y>0\}\ x,y := x+y,0 \ if \ y>0\ \{y=0\}$

- **proof-checking**
  - 許多laws
  - safety properties、liveness properties

105

Deciplines
- language semantics

Axiomatic Semantics

Example : *Given N, M, compute x,y such that*

- $x \times N + y = M$
- $0 \Leftrightarrow \ y < N$

Program *division*
declare *x,y,z,k* : integer
initially *x,y,z,k = 0, M, N, 1*
assign *z,k :=* *2z, 2,k* *if y≥ 2xz ~*
N, 1 *if y < 2xz*
x,y := *x+k, y-z* *if y ≥ 2xz*
end *{division}*

106

Deciplines
- language semantics

Axiomatic Semantics

Properties: Given a program F

- *p unless q* iff for all *s* in **F**, $\{p \wedge \neg q\} s \{p \vee q\}$
- **stable** *p* iff *p unless false*
- **invariant** *p* iff **stable** $p \wedge$ (initial condition $\rightarrow p$)
- *p ensures q* iff (*p unless q*) $\wedge \exists s$ in **F**, $\{p \wedge \neg q\} s \{q\}$

Deciplines
- language semantics

Axiomatic Semantics

Properties: Given a program **F,** $p \rightarrow q$ iff

$$\frac{p \text{ ensures } q}{p \rightarrow q} \qquad \frac{p \rightarrow q, \quad q \rightarrow r}{p \rightarrow r}$$

$$\frac{\forall \ w \text{ in } W (p(w) \rightarrow q)}{(\exists \ w \text{ in } W(p(w))) \rightarrow q}$$

Deciplines
- language semantics

Axiomatic Semantics

Some theorems :

$$\frac{p \text{ unless } q,\ q \text{ unless } r}{p \vee q \text{ unless } r} \qquad \frac{p \text{ unless } q}{p \vee r \text{ unless } q \vee r}$$

$$\frac{p \rightarrow q,\ r \text{ unless } b}{p \wedge r \quad (q \wedge r) \vee b}$$

Practical achievements (1/6)

first-order logic & theorem-proving

- NqThm by Boyer & Moore
- 1985 , the verification of a 4-bit CPU, FM 8501, to the bit level
- verification of a high-level language compiler
- *A Computational Logic Handbook*, Academic Press, 1988

Practical achievements (2/6)

Formal Methods

- occam
- IMS T800 transputer
- estimated reduction in the development time by 12 months.
- 1990 Queen's award!!!
  - (49 recepients)

111

Practical achievements (3/6)

Formal Methods

- Z notations for the formal specifications
- IMS CICS
- estimated reduction of development cost by 5 millions
- 1992 Queen's award

112

Practical achievements (4/6)

model-checking

- Burch, Clarke, McMillan, Dill, Hwang
- BDD symbolic manipulation
- CMU SMV, fully automatic
- Intel 32-bit ALU / 8 registers / two-layer pipeline
- state counts up to 10^{120}
- 4 hr 20min in SUN 4

Practical achievements (5/6)

- Symbolic Trajectory Evaluation
- Hazelburst, Seger
- 64-bit multiplier
- simple or Wallace tree
- combinational circuits
- 800 sec. in Sparc 10/51

Practical achievements (6/6)

First-order logic and linear hybrid automata

- Bosscher, Polak, Vaandrager
- proof-checking
- Phillip audio control network, physical layer
- voltage variations, clock rate stability
- assuming no collison, signal delay
- proving that 1/17 frequency variation is acceptable.
  - The tolerance of Phillip is 5%.
- automatically checked with Henzinger's HyTech

115

Workout 1: proof of a simple program

Initially x = 0;

- consumer:

for (; 1;) if (x > 0) x--;

- Producer:

for (; 1;) if (x< 1) x++;

- Atomicity assumption
- Please prove that x is always no greater than 1.

116

Another workout: Petrification

1. please model the two programs in the last page as Petri-nets.
2. Please explain the following terms.
   1. Safety property
   2. Liveness property
   3. Fairness assumption
   4. Interleaving semantics