Formal Methods and Beyond

***Important** part of the course*

Prof. T.H. Tse
Department of Computer Science
Email: thtse@cs.hku.hk
Web: hku.hk/thtse

Problems with Industrial Practices

- ***Craft and not engineering***
  - Learn from experience
  - Trial and error
  - No quality assurance

We sell it as "an iterative approach" ☹

- The profession is now where civil engineering was in Greek and Roman times before they had calculus and Newtonian mechanics

Formal Methods

- *Mathematical* techniques that support rigorous reasoning in software development and quality assurance
- ***Examples***
  - Abstract models such as VDM, Z
  - Algebraic models such as OBJ3, FOOPS
  - Concurrency models such as CSP, CCS, Petri nets
  - Category theory
  - First order predicate logic such as Prolog

We will introduce CSP

Advantages of Formal Methods

- Consistent and unambiguous
- Reduces coding time despite slightly longer time for specification
- Proof of correctness
- Eases future enhancements

Motivating Example

Teleconferencing

1 phone* communicates with *2 lines

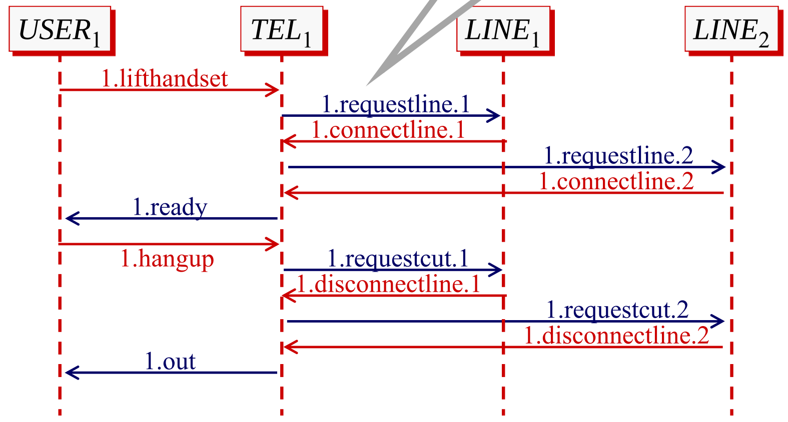
Requirements for teleconferencing system

- System consists of *N* phones
- To save line charges, only *N* external lines
- A user picks up the hand set of phone *i*
- They may use this phone to connect to 2 clients:
  - External line *i* for 1 client
  - Then external line *i* + 1 for 2nd client
- In case external line *i* + 1 is used by a user at phone *i* + 1, they must wait

Teleconferencing Example

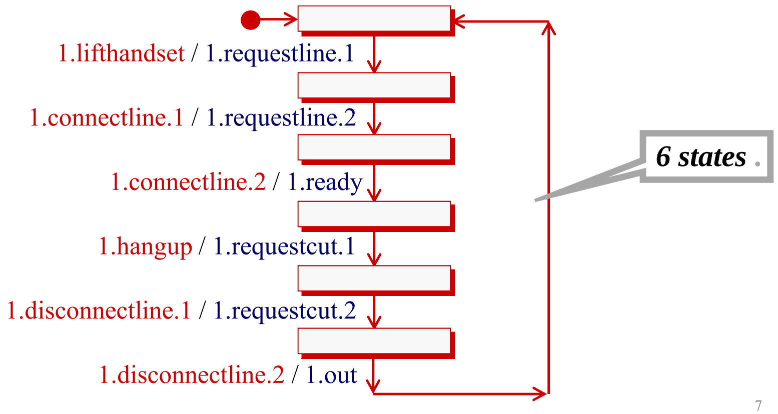
Sequence Diagram

TEL_1 requests $LINE_1$



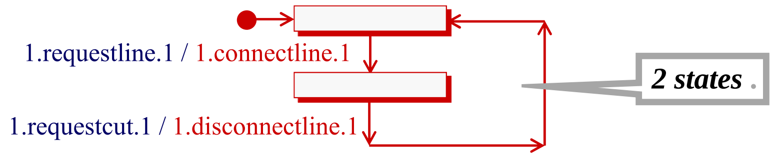
Teleconferencing Example

State Machine for TEL_1



Teleconferencing Example

State Machine for $LINE_1$



Teleconferencing Example

Limitations of UML

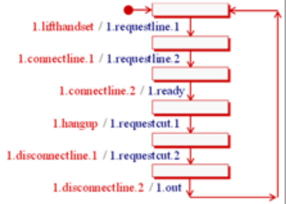
- ***Deadlock Problem in the Example***
  - *N* users picked up the handsets
- ***Starvation Problem in the Example***
  - If telephone $i+1$ is used very often, a user cannot find a line for telephone i
- Sequence diagram for one telephone is not useful for solving these problems
- State machine of one object is not useful either .

Everybody waits forever for second line

Teleconferencing Example

More Limitations

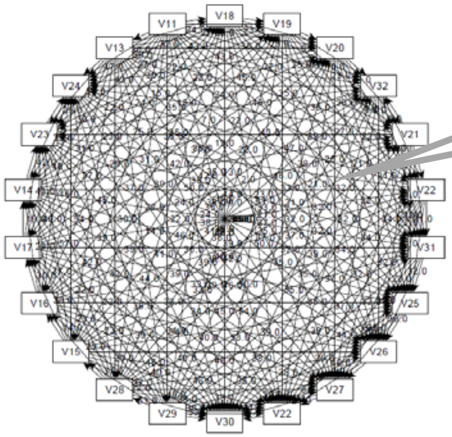
- State machine for all objects?
  - 6^N x 2^N states
  - 20 736 states for $N = 4$
  - 2 985 984 states for $N = 6$
- Too complex .



- 6^N states for N telephones
- 2^N states for N lines

Source: L. Mariani et al.

Example of Complex State Machine



Impossible to solve problems visually .

Teleconferencing Example

Solutions to Limitations of UML

- The need for
  - Concise but precise languages
  - Validation and verification methods

- Concise = brief
- Precise = accurate

- Validation = checking with user requirements
- Verification = checking with the specification .

Communicating Sequential Processes (CSP)

Classic Reference

- C.A.R. Hoare, *Communicating Sequential Processes*, Prentice-Hall (1985)
  - Also available at http://www.usingcsp.com/cspbook.pdf .

- ***Do not download unless you are an experienced, advanced student***
- ***Ask me questions if you don't understand .***

Processes

Intuitively similar to state machine in UML

- A process is the behaviour pattern of an ***object***
- Denoted by upper case characters
  - ***Examples:*** *P*
    TEA_MACHINE .

Events

Intuitively similar to events in UML

- Internal events
  - ***Examples:*** *v*
    maketea
- Communication events
  - Via some interface channel
  - ***Format:*** *channel.v*
  - ***Examples:*** *coinslot.$*
    hatch.tea
- Denoted by lower case letters .

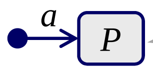
Alphabet α

- The set of events relevant to a given process
  - ***Example***

    α *TEA_MACHINE*
    = {*maketea*, *coinslot.$5*, *hatch.tea*} .

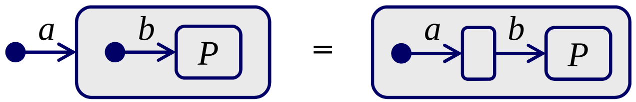
Prefix ($a \to P$)
Intuitive Introduction

- **($a \to P$)**



Intuitively similar to change of state (due to event) in UML

- **($a \to (b \to P)$)** = **($a \to b \to P$)**



Prefix
($a \to P$)

The brackets are part of the syntax

- Given a process P and an event a,
  $(a \to P)$
  is a process, read as "a then P"
- For conceptual simplicity, extend αP to include "a"
- Note the syntax:
  - $a \to P$ **??** ⊗
  - $(a \to b)$ **??** ⊗
  - $(P \to Q)$ **??** ⊗
  - $(a \to b \to P)$

- ***Yes, this is correct***
- ***It means***
  $(a \to (b \to P))$.

Prefix Examples

- *TEA_MACHINE*
  = (*coinslot.$5*
  → *maketea*
  → *hatch.tea*
  → ***END_OF_SALES***)
- *BAD_TEA_MACHINE*
  = (*coinslot.$5*
  → ***STUCK***)

Successful Termination
SKIP

- ***Previous Example: END_OF_SALES***
- In general, simply write ***SKIP*** .

Hint
- "**1**" in ***SK1P*** for success .

Failure
STOP

- ***Previous Example: STUCK***
- In general, simply write ***STOP*** .

Hint
- "1" in ***SK1P*** for success
- "0" in ***ST0P*** for failure .

Termination Examples

- *TEA_MACHINE*
  = (*coinslot.$5*
  → *maketea*
  → *hatch.tea*
  → ***SKIP***)
- *BAD_TEA_MACHINE*
  = (*coinslot.$5*
  → ***STOP***)

Recursion

- To model the behaviour

  CLOCK = (*tick* → *tick* → *tick* → …)

  we write

  CLOCK = (*tick* → ***CLOCK***)

Exercise
- Use recursion to model

  CLOCK = (*tick* → *tock* → *tick* → *tock* → …) .

Recursion Example 1

- ***TEA_MACHINE***
  = (*coinslot.$5*
  → *maketea*
  → *hatch.tea*
  → ***TEA_MACHINE***)

Recursion Example 2

- *BAD_TEA_MACHINE*

  = (*coinslot*.$5

  $\rightarrow$ *BAD_TEA_MACHINE*)

Choice by Input Value ($a \rightarrow P \mid b \rightarrow Q$)

The *two* brackets are part of the syntax

- The process behaves like P or Q, depending on whether the event value is a or b
- "|" is read as "choice"
- ***Example***

  MACHINE

  = (*coinslot*.**$5** $\rightarrow$ *hatch.tea* $\rightarrow$ *MACHINE*

  | *coinslot*.**$10** $\rightarrow$ *hatch.cappucino* $\rightarrow$ *MACHINE*)

Choice by Channel ($x.a \rightarrow P$ ▯ $y.b \rightarrow Q$)

The *two* brackets are part of the syntax

- The process behaves like P or Q, depending on whether the communication channel is x or y
- "▯" is read as "fetbar"
- ***Example***

  MACHINE

  = (*coinslot*.$5

  $\rightarrow$ (*tea_button*.*press* $\rightarrow$ *hatch.tea* $\rightarrow$ *MACHINE*

  ▯ *icetea_button*.*press* $\rightarrow$ *hatch.icetea* $\rightarrow$ *MACHINE*))

The story goes that this was a typo for "fatbar"

More Choice Example

NEW_MACHINE

= (*coinslot*.**$5** $\rightarrow$

(*coinslot*.$5 $\rightarrow$ *hatch.cappucino* $\rightarrow$ *NEW_MACHINE*

▯ *drink_button*.*press* $\rightarrow$ *hatch.tea* $\rightarrow$ *NEW_MACHINE*)

| *coinslot*.**$10** $\rightarrow$

(*change_button*.*press* $\rightarrow$ *change*.$5 $\rightarrow$ *hatch.tea*

$\rightarrow$ *NEW_MACHINE*

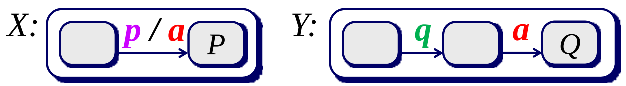
▯ *drink_button*.*press* $\rightarrow$ *hatch.cappucino*

$\rightarrow$ *NEW_MACHINE*))

Interaction
$X \parallel Y$

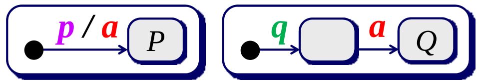
No brackets added before or after X and Y

- Processes X and Y can start or finish at any time
- Any potentially common event a (relevant to both X and Y) must be synchronized
- Intuitively similar to state machine interaction



- ***But*** CSP does not distinguish between incoming or outgoing event …

Interaction Example 1

- Processes $(p \to a \to P)$ and $(q \to a \to Q)$ interact
- Intuitively  interact

- We write

$$(p \to a \to P) \parallel (q \to a \to Q)$$

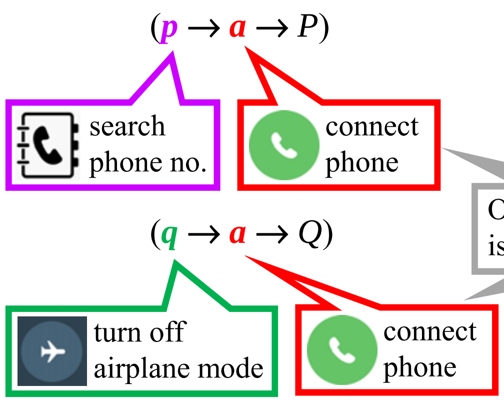
The 4 brackets are due to the original processes, not added by interaction .

Interaction Example 1

$$(p \to a \to P) \parallel (q \to a \to Q)$$

- Need to specify the characteristic of every event, e.g.,
  - a is a potentially ***common*** event relevant to both P and Q
    - and hence in both αP and αQ
  - p is an ***internal*** event relevant only to P but not Q
    - and hence in αP but not αQ
  - q is an ***internal*** event relevant only to Q but not P
    - and hence in αQ but not αP
- Only the ***common*** event a is synchronized .

Interaction Example 1
Real Life Application



Only "connect phone" is synchronized .

Interaction Example 2

Consider two processes

- ***GREEDY_CUST***
- ***MACHINE***

with 4 potentially common events

- *coinslot*.$5
- *coinslot*.$10
- *hatch.tea*
- *hatch.cappucino*

relevant to both processes .

Interaction Example 2

- ***GREEDY_CUST***

  = (*coinslot*.$5 →
  (*hatch.tea* → ***GREEDY_CUST***
  | *hatch.cappucino* → ***GREEDY_CUST***))

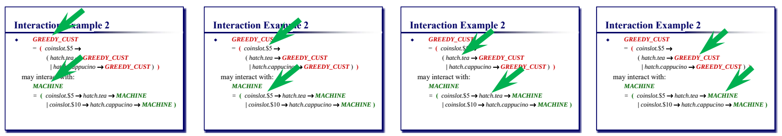
  interacts with:

  MACHINE

  = (*coinslot*.$5 → *hatch.tea* → ***MACHINE***
  | *coinslot*.$10 → *hatch.cappucino* → ***MACHINE***)
- We need only write ***MACHINE*** || ***GREEDY_CUST***
- What is the effect?

Interaction Example 2

The Effect



- ***MACHINE*** || ***GREEDY_CUST***

  = (*coinslot*.$5 → *hatch.tea*
  → ***MACHINE*** || ***GREEDY_CUST***)

Laws

- Minimum set of axioms that are taken to be true, to serve as starting points for formal reasoning
- All other system behaviour can then be proved

More reliable than human intuition .

Laws

Examples

- $P \parallel Q = Q \parallel P$
- $P \parallel (Q \parallel R) = (P \parallel Q) \parallel R$
- $P \parallel STOP = STOP$

Interactive processes may be presented in any order

Interactive processes may be grouped in any way

Interacting with a failing process will fail.

Laws

Examples (Continued)

- Given potentially common alphabets ***a*** and ***b*** relevant to both *P* and *Q*,
  - $(a \to P) \parallel (a \to Q) = (a \to (P \parallel Q))$

    If both immediate common events agree, successfully synchronized and fired
  - $(a \to P) \parallel (b \to Q) = STOP$ if $a \neq b$

    If immediate common events do not agree, interaction fails.

Laws

Examples (Continued)

- Given potentially common event ***a*** relevant to both *P* and *Q*, internal event ***p*** relevant to *P* but not *Q*, and internal event ***q*** relevant to *Q* but not *P*,
  - $(p \to P) \parallel (a \to Q) = (p \to (P \parallel (a \to Q)))$

    Given one immediate internal event, it will be fired
  - $(p \to P) \parallel (q \to Q) = (p \to (P \parallel (q \to Q)) \mid q \to ((p \to P) \parallel Q))$

    Given two immediate internal events, either one may be fired.

Formal Reasoning Based on the Laws

- For instance, we can then ***prove*** that

$$(p \to a \to P) \parallel (q \to a \to Q)$$
$$= (p \to q \to a \to (P \parallel Q) \mid q \to p \to a \to (P \parallel Q))$$

More reliable than human intuition.

Proof

$$(p \to a \to P) \,||\, (q \to a \to Q)$$
$$= \big(p \to ((a \to P) \,||\, (q \to a \to Q))$$
$$\;\; | \; q \to ((p \to a \to P) \,||\, (a \to Q)) \big)$$

Based on the law $(p \to \blacksquare) \,||\, (q \to \bullet)$
$= \big(p \to (\blacksquare \,||\, (q \to \bullet)) \,|\, q \to ((p \to \blacksquare) \,||\, \bullet) \big)$ ...

Proof

$$(p \to a \to P) \,||\, (q \to a \to Q)$$
$$= \big(p \to ((a \to P) \,||\, (q \to a \to Q))$$
$$\;\; | \; q \to ((p \to a \to P) \,||\, (a \to Q)) \big)$$
$$= \big(p \to q \to ((a \to P) \,||\, (a \to Q))$$
$$\;\; | \; q \to p \to ((a \to P) \,||\, (a \to Q)) \big)$$

Based on the law $(p \to \blacksquare) \,||\, (a \to \bullet)$
$= (p \to (\blacksquare \,||\, (a \to \bullet)))$ ...

Proof

$$(p \to a \to P) \,||\, (q \to a \to Q)$$
$$= \big(p \to ((a \to P) \,||\, (q \to a \to Q))$$
$$\;\; | \; q \to ((p \to a \to P) \,||\, (a \to Q))$$
$$= \big(p \to q \to ((a \to P) \,||\, (a \to Q))$$
$$\;\; | \; q \to p \to ((a \to P) \,||\, (a \to Q)) \big)$$
$$= \big(p \to q \to a \to (P \,||\, Q)$$
$$\;\; | \; q \to p \to a \to (P \,||\, Q) \big)$$

Based on the law
$(a \to \blacksquare) \,||\, (a \to \bullet)$
$= (a \to (\blacksquare \,||\, \bullet))$.

Teleconferencing Example

Telephones

For conciseness, not part of CSP

- Let $i \oplus 1 = i+1$ if $i < N$, and let $i \oplus 1 = 1$ if $i = N$
- TEL_i
  $= (\, i.lifthandset \to i.connectline.i \to i.connectline.i \oplus 1$
  $\to i.hangup \to i.disconnectline.i \to i.disconnectline.i \oplus 1$
  $\to TEL_i \,)$
- $TELS = TEL_1 \,||\, TEL_2 \,||\, \ldots \,||\, TEL_N$

Teleconferencing Example:
Lines

For conciseness, not part of CSP

- Let $i \ominus 1 = i - 1$ if $i > 1$, and let $i \ominus 1 = N$ if $i = 1$
- $LINE_i$
  $= (\ i.connectline.i \rightarrow i.disconnectline.i \rightarrow LINE_i$
  $\square\ i\ominus 1.connectline.i \rightarrow i\ominus 1.disconnectline.i \rightarrow LINE_i\ )$
- $LINES = LINE_1 \parallel LINE_2 \parallel \ldots \parallel LINE_N$
- $NETWORK = TELS \parallel LINES$

But have we solved the deadlock problem?

Teleconferencing Example
Solution of Deadlock Problem

Main Idea

- Install a guard that disallows *all telephones* to be used at any one time
- In other words, install a guard that allows no more than N–1 telephones to be used at any one time

Teleconferencing Example
Solution of Deadlock Problem

- Let $L = \{1.lifthandset, 2.lifthandset, ..., N.lifthandset\}$
- $x : L$ means that one of the telephones is picked up

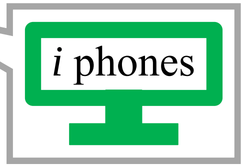
"$x : L$" means "$x \in L$"

- Let $H = \{1.hangup, 2.hangup, ..., N.hangup\}$
- $x : H$ means that one of the telephones is hung up

"$x : H$" means "$x \in H$".

Teleconferencing Example
Solution of Deadlock Problem

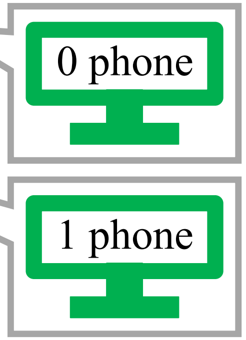
- Let $GUARD_i$ be a process that keeps a count i of the number of phones currently used



Teleconferencing Example
Solution of Deadlock Problem

- Suppose no phone is currently used
  - If we pick up one phone, then total no. being used = 1
- **We write**

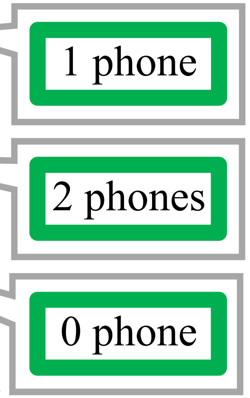
$GUARD_0 = (\, x : L \rightarrow GUARD_1 \,)$.



Teleconferencing Example
Solution of Deadlock Problem

- Suppose 1 phone is currently used
  - If we pick up one more phone, then total no. being used = 2
  - If we hang up one phone, then total no. being used = 0
- **We write**

$$GUARD_1 = (\, x : L \rightarrow GUARD_2 \mid x : H \rightarrow GUARD_0 \,) \ .$$

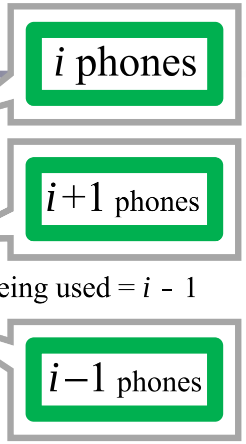


Teleconferencing Example
Solution of Deadlock Problem

- Suppose i phones are currently used (where i = 1, … N–2)
  - If we pick up one more phone, then total no. being used = i + 1
  - If hang up one phone, then total no. being used = i - 1
- **We write**

$$GUARD_i = (\, x : L \rightarrow GUARD_{i+1} \mid x : H \rightarrow GUARD_{i-1} \,)$$

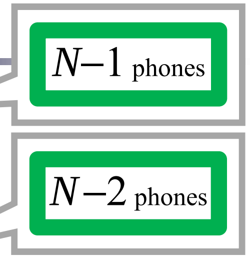
for i = 1, … N–2 .



Teleconferencing Example
Solution of Deadlock Problem

- Suppose $N-1$ telephones are currently used
  - The guard will not allow any further telephone to be picked up
  - If we hang up one telephone, then total no. of telephones used = N - 2
- **We write**

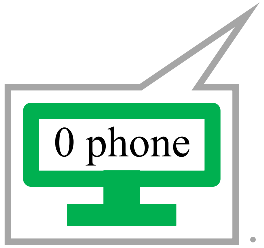
$GUARD_{N-1} = (\, x : H \rightarrow GUARD_{N-2} \,)$.



Teleconferencing Example

Solution of Deadlock Problem

- *NEW_NETWORK* = *TELS* || *LINES* || ***GUARD***$_0$



Teleconferencing Example

Verification

- Conventionally:
  - Verify the implementation using test data, *or*
  - Verify the specification using simulation
- ***But*** with $6^N \times 2^N$ states
  - The process is too complex
  - The chance of revealing a failure is small

Teleconferencing Example

Proof of Correctness

- Suppose *N* lines are being connected
  - Since no more than $N-1$ telephones have been picked up, some of them must be working with teleconferencing
- Suppose less than *N* lines are being connected
  - Some telephone can be connected to the spare line
- In either case, there is no deadlock

Teleconferencing Example

Solution of Starvation Problem?

- Starvation problem is beyond the scope of this course

☹

Disadvantages of Formal Methods

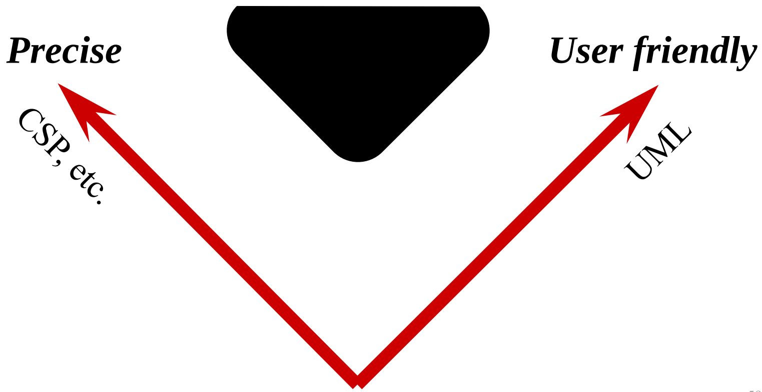
- Formal methods may not be accepted by software engineers
  - Not intuitive, full of jargon
  - Bottom up
  - Lack of training
  - Lack of track record
  - Not supported by CASE tools
  - Disregard of current practices
- The method may be very difficult for complex systems .

Disadvantages of Formal Methods

From: username@cs.hku.hk
To: thtse@cs.hku.hk

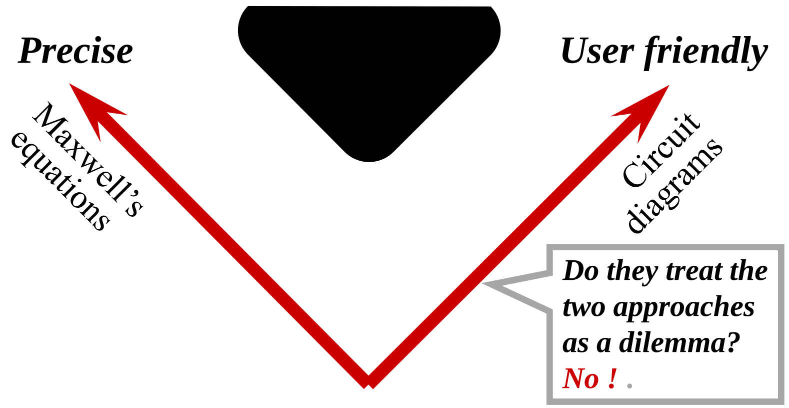
The CSP assignment is really so difficult. I can tell you that CSP is the most difficult language in the world. I really feel depressed and helpless in the past two weeks when dealing with the assignment. ... I really don't know what to do instead of just ignoring the assignment .

We are at the Crossroads



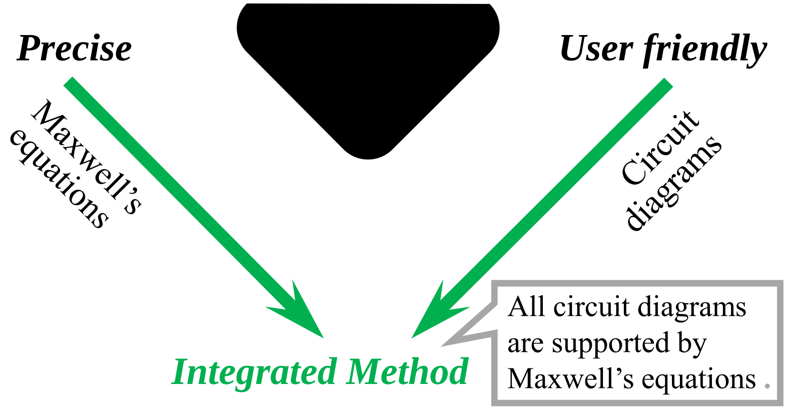
What about Other Engineering Disciplines?

Electrical Engineering

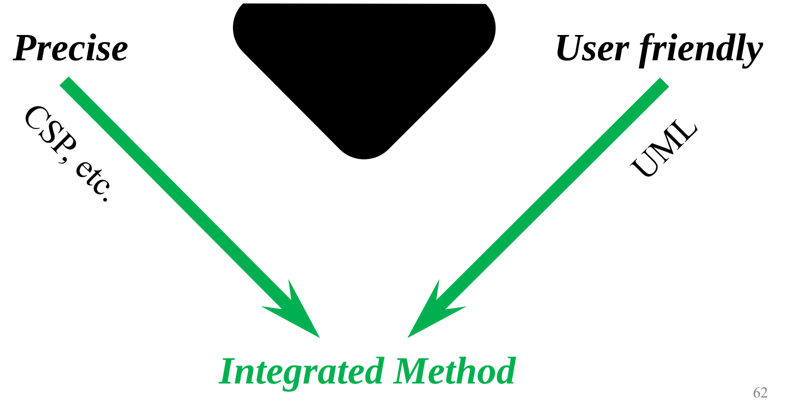


What about Other Engineering Disciplines?

Electrical Engineering



Future of Software Engineering



Future of Software Engineering

- ***Integrated method*** is beyond the scope of this course