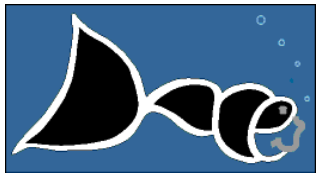
# Generalized Universality

***R. Guerraoui, EPFL***







# Act 1
# Classical Universality

# Act 2
# Modern Universality

# Act 3
# Generalized Universality

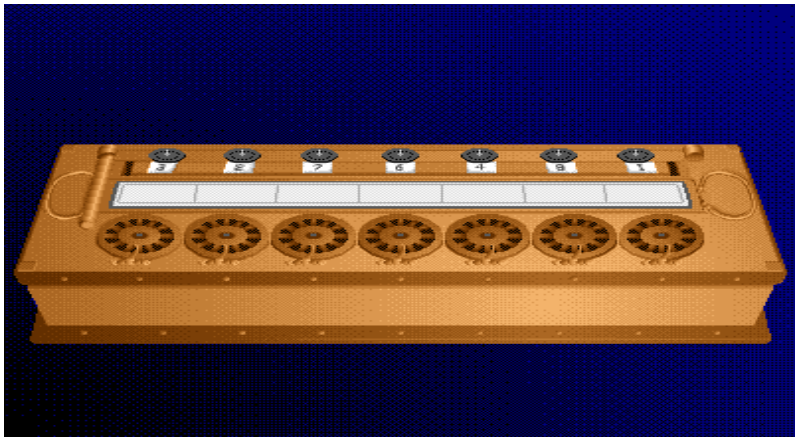
# Algorithm

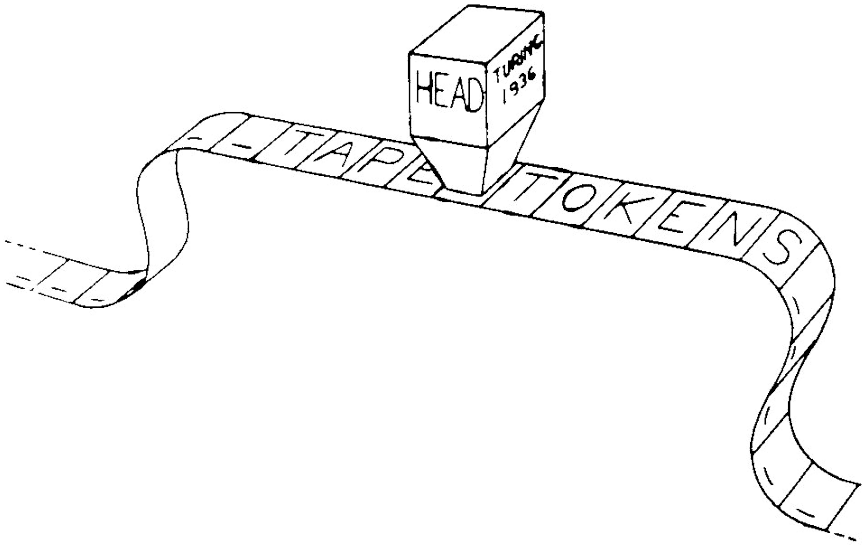
A finite set of precise instructions

The only intelligence required is to understand and compute the instructions

Must always produce a result

# Which machine enables to compute everything?





# Act 1
# Classical Universality

# Act 2
# Modern Universality

# Act 3
# Generalized Universality

# Algorithm

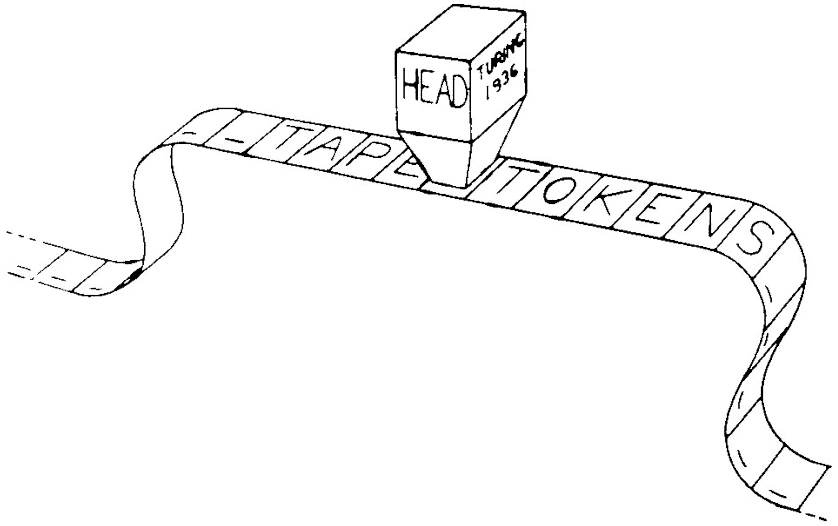
A finite set of precise instructions

The only intelligence required is to understand and compute the instructions

Must **always** produce a result

NB. Despite concurrency and failures

# Universality of consensus [Lamport-Schneider-Herlihy-CT]




Linearizable (atomic)

Highly-available (wait-free)

# Consensus

Processes propose each a value and ***agree*** on one of those values

output = ***propose(***input***)***

# Consensus

***Validity***: every value decided has been proposed

***Agreement***: no two different values are decided

***Termination***: every correct process that proposes a value eventually decides

# Universal construction

A state machine of which each process holds a copy

A list of commands local to each process

A list of consensus objects shared by the processes

# Universal construction

- while(true)

- c = commands.next()
- cons = Consensus.next()

- c’ = cons.***propose***(c)
- sM.***perform***(c’)

# Act 1
# Classical Universality

# Act 2
# Modern Universality

# Act 3
# Generalized Universality

# Generalized Universality

What if consensus is not available?

Consensus is the particular case of k-consensus

# K-consensus [Chauduri, Afek et al.]

- Every process proposes a vector of k values and returns a value at some position

- Every process invokes kVectCons with propose(kVect) and returns a pair (value,position)

- NB. Equivalent of invoking with a value and obtaining a value such that at most k are different

# K-consensus

- ***Validity***: the value returned at any position has been proposed at that position
- ***Agreement***: no two values returned at the same position are different
- ***Termination***: every correct process that proposes eventually returns

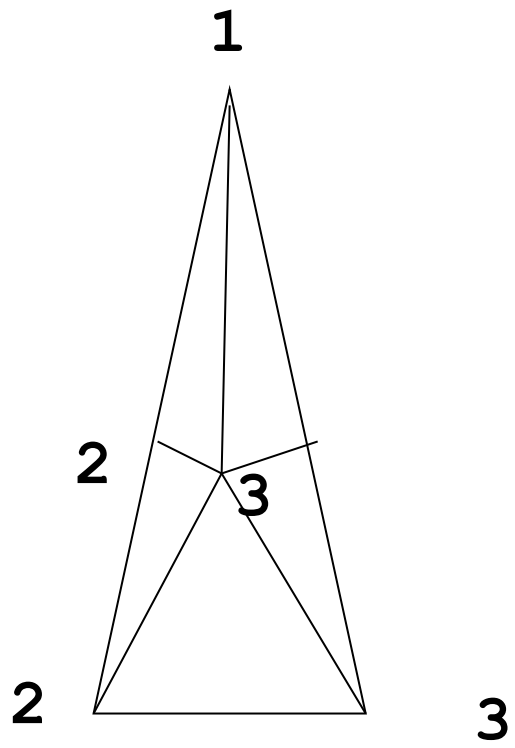
# K-consensus

Wait-free impossible in an asynchronous shared memory system (registers) with k+1 processes

HS,BG,SZ 93 (Godel prize 2004)

k-consensus is strictly weaker than consensus in any system of more than k+1 processes

# K-consensus (Sperner)




Sperner’s Lemma: at least one triangle has three colors

# K-consensus

Leader(): returns a process such that eventually the same correct process is returned to all

Leader-k(): returns a subset of processes of size k such that eventually the set is the same and contains at least one correct process

# What form of universality with K-consensus?

With consensus

Processes implement a highly-available state machine

With k-consensus

Processes implement k state machines of which ***at least one*** is highly-available

## Generalized Universality

# Act 1
# Classical Universality

# Act 2
# Modern Universality

# Act 3
# Generalized Universality

# k state machines

k state machines: each process holding a copy of each (sM(i))

k lists of commands local to each process

A list of k-vector consensus objects (kVectCons)

Reads and writes in shared memory

# Universal construction

- while(true)
- c = commands.next()
- cons = consensus.next()

- c’ = cons.propose(c)
- sM.perform(c’)

# Generalized universality?

- while(true)
- for j = 1 to k: com(j) = commands(j).next()
- kVectC = kVectCons.next()

- (c,i) = kVectC.propose(com)
- sM(i).perform(c)

# Generalized universality?

- while(true)
- for j = 1 to k: com(j) = commands(j).next()
- kVectC = kVectCons.next()

- (c,i) = kVectC.propose(com)
- read shared memory and update any missing c’
- sM(i).perform(c)
- write (c,i) in shared memory

# Commitment (adopt-commit)

write (c) at level 1
let V1 be the set of values at level 1
if V1 has only c, write (commit, c) at level 2

let V2 be the set of values at level 2
if V2 has only (commit, c) then return(commit, c)
if V2 has some (commit, c') then return(adopt, c')
else return (adopt, c)

# Commitment

- ***Invariant (1)***: if a value v is committed then no other value is returned

- ***Invariant (2)***: if all processes propose the same command then the command is committed

# Generalized universality (step 0)

- newCom = commands.next()
- while(true)
- kVectC = kVectCons.next()

# Generalized universality (step 1)

- …
- (c,i) = kVectC.propose(newCom)
- …

# Generalized universality (step1-2)

- …
- (c,i) = kVectC.propose(newCom)
- vect(i) = commitment(i,c)
- …

# Generalized universality (step1-2-2’)

- …
- (c,i) = kVectC.propose(newCom)
- vect(i) = commitment(i,c)
- for j = 1 to k except i:
  - vect(j) = commitment(newCom(j))

  …

# Generalized universality (step 3)

…

for i = 1 to k

- if ok(vect(i)) then
  - sM(i).perform(vect(i))
  - newCom(i) = commands(i).next()
- else
  - newCom(i) = vect(i)

# Generalized universality (step 3’)

…

for i = 1 to k

- If older(newCom(i),vect(i)) then
  sM(i).perform(newCom(i))
- If no(vect(i)) then newCom(i) = vect(i)
- else
- sM(i).perform(vect(i))
- If  vect(i) = newCom(i) then
  - newCom(i) = commands(i).next()
- add(newCom(i),vect(i))

# Commitment

- ***Safety***: a process does not perform a command unless all others know the command

- ***Liveness***: at least one process executes a command in every round

NB. Every correct process executes at least one command every two rounds

# Act 1
# Classical Universality

# Act 2
# Modern Universality

# Act 3
# Generalized Universality