# Temporal Modality

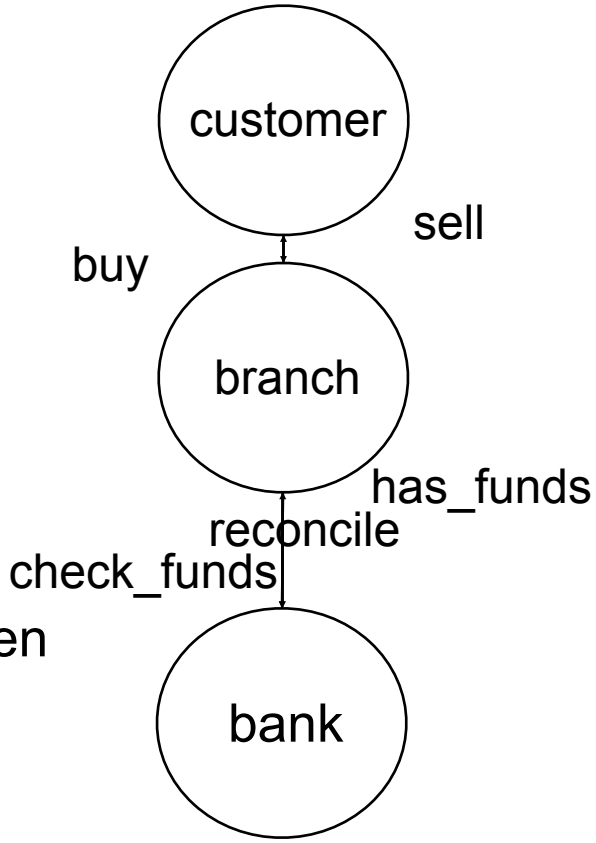

In this lecture you will be introduced to logics for systems in which the things that are true are relative to states of the world.

This allows logic to be used for problems in which the things that are true may change over time.

We apply this to the problem of analysing a protocol.

# Problem: Banking Protocol

a(customer, C) :: buy ⇒ a(branch, B) then
sell ⇐ a(branch, B) then
a(customer, C)
a(branch, B) :: buy ⇐ a(customer, C) then
check_funds(C) ⇒ a(bank, X) then
has_funds(C) ⇐ a(bank, X) then
sell ⇒ a(customer, C) then
reconcile(C) ⇒ a(bank, X) then
a(branch, B)
a(bank, X) :: ( (check_funds(C) ⇐ a(branch, B) then
has_funds(C) ⇒ a(branch, B) ) or
reconcile(C) ⇐ a(branch, B) then
a(bank, X)




Are there flaws in this protocol?

# A State Sequence (S)

S
- wakes_up(dave)
- reads_mail(dave)
- gives_lecture(dave)
- reads_mail(dave)
- goes_home(dave)
- sleeps(dave)

access(J, *S*, *F*) means *F* is the Jth state in *S*

e.g. access(2, *S*, *F*) is true if

*F* = reads_mail(dave)

# Proof Rules (Intra-State)

| Proof | Sub-proofs |
|---|---|
| $(S,J) \vdash A$ | access(J, $S$, $F$) \| $F \vdash A$ |
| $(S,J) \vdash A$ and B | $(S,J) \vdash A$ \| $(S,J) \vdash B$ |
| $(S,J) \vdash$ A or B | $(S,J) \vdash A$ |
| $(S,J) \vdash$ A or B | $(S,J) \vdash B$ |
| $(S,J) \vdash$ not(A) | $(S,J) \nvdash A$ |

access(J, $S$, $F$) means $F$ is the Jth state in $S$

# Proof Rules (Inter-State)

| Proof | Sub-proofs |
|---|---|
| $(S,J) \vdash \text{next}(A)$ | $(S,J+1) \vdash A$ |
| $(S,J) \vdash \text{prev}(A)$ | $(S,J-1) \vdash A$ |
| $(S,J) \vdash \text{e\_future}(A)$ | $(S,K) \vdash A$ for some $K > J$ |
| $(S,J) \vdash \text{e\_past}(A)$ | $(S,K) \vdash B$ for some $K < J$ |
| $(S,J) \vdash \text{a\_future}(A)$ | $(S,K) \vdash A$ for all $K > J$ |
| $(S,J) \vdash \text{a\_past}(A)$ | $(S,K) \vdash A$ for all $K < J$ |

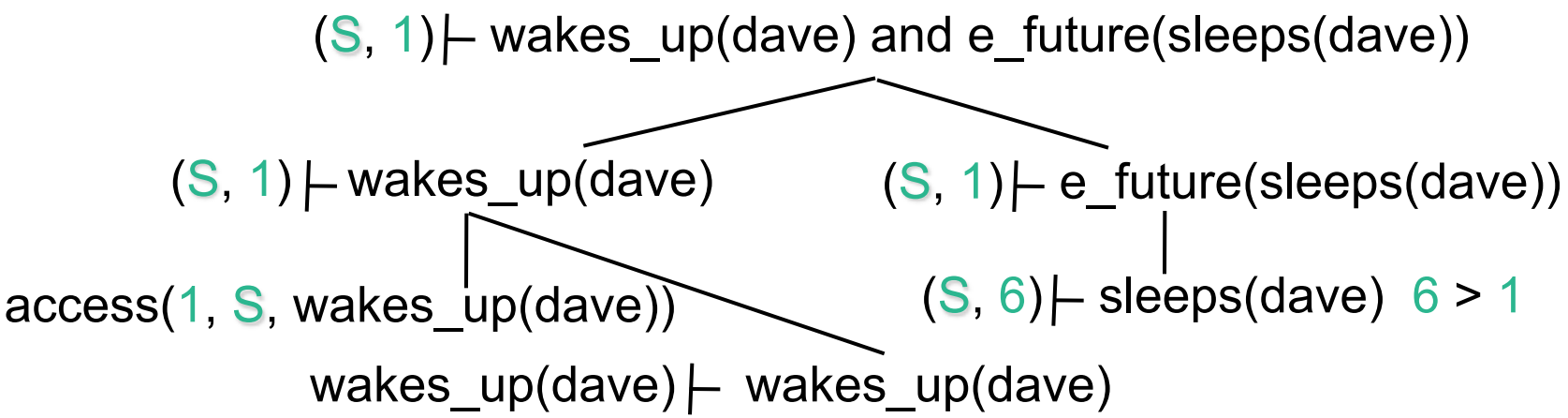
Notice this can be inefficient.

# A Proof (1)

Given S {
wakes_up(dave
reads_mail(dave)
gives_lecture(dave)
reads_mail(dave)
goes_home(dave)
sleeps(dave)

Show: wakes_up(dave) and e_future(sleeps(dave))

# A Proof (2)

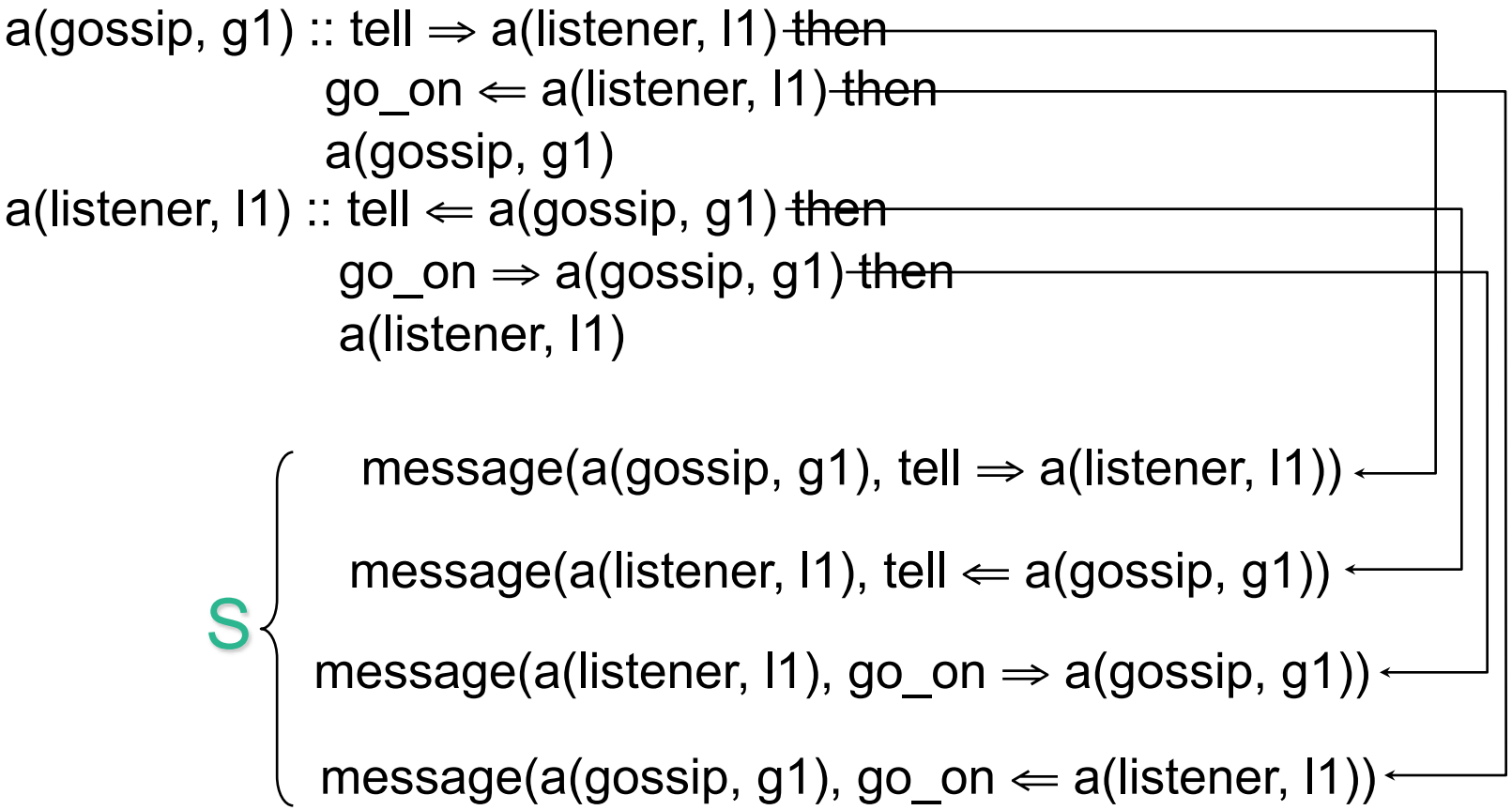



S {
wakes_up(dave
reads_mail(dave)
gives_lecture(dave)
reads_mail(dave)
goes_home(dave)
sleeps(dave)
}

# Solution: Banking Protocol


- Express the problem in a way that can be represented by temporal sequences.
- Then think of situations that could be problematic and express these as temporal properties.
- Then attempt to prove that the problematic properties are true.

# A Message Sequence (S)



# Accessing States in a Sequence

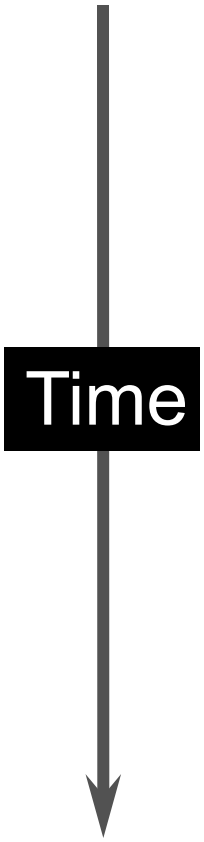
access(J, *S*, *F*) means *F* is the Jth state in *S*

e.g. access(2, *S*, *F*) is true if

*F* = message(a(listener, l1), tell ⇐ a(gossip, g1))

*S* {
- message(a(gossip, g1), tell ⇒ a(listener, l1))
- message(a(listener, l1), tell ⇐ a(gossip, g1))
- message(a(listener, l1), go_on ⇒ a(gossip, g1))
- message(a(gossip, g1), go_on ⇐ a(listener, l1))

# Banking Protocol Sequence (S)




message(a(customer,c1), buy ⇒ a(branch,b1))

message(a(branch,b1), buy ⇐ a(customer,c1))

message(a(branch,b1), check_funds(c1) ⇒ a(bank,x1))

message(a(bank,x1), check_funds(c1) ⇐ a(branch,b1))

message(a(bank,x1), has_funds(c1) ⇒ a(branch,b1))

message(a(branch,b1), has_funds(c1) ⇐ a(bank,x1))

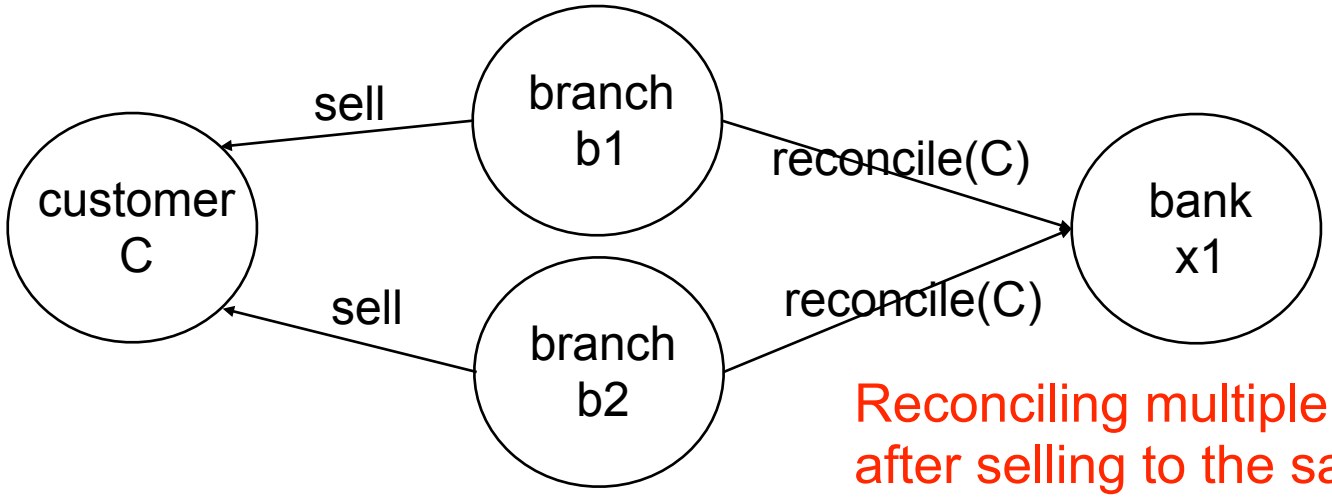
message(a(branch,b1), sell ⇒ a(customer,c1))

message(a(customer,c1), sell ⇐ a(branch,b1))

message(a(branch,b1), reconcile(c1) ⇒ a(bank,x1))

# Undesirable Temporal Property

message(a(customer,C),sell<=a(branch,b1)) and
e_future(message(a(customer,C),sell<=a(branch,b2)) and
e_future(message(a(branch,b1),reconcile(C)=>a(bank,x1)) and
next(message(a(branch,b2),reconcile(C)=>a(bank,x1)))))




Reconciling multiple transactions after selling to the same customer.

# Sequence Satisfying Property


message(a(customer,c1), buy ⇒ a(branch,b1))
message(a(branch,b1), buy ⇐ a(customer,c1))
message(a(branch,b1), check_funds(c1) ⇒ a(bank,x1))
message(a(bank,x1), check_funds(c1) ⇐ a(branch,b1))
message(a(bank,x1), has_funds(c1) ⇒ a(branch,b1))
message(a(branch,b1), has_funds(c1) ⇐ a(bank,x1))
message(a(branch,b1), sell ⇒ a(customer,c1))
message(a(customer,c1), sell ⇐ a(branch,b1))
message(a(customer,c1), buy ⇒ a(branch,b2))
message(a(branch,b2), buy ⇐ a(customer,c1))
message(a(branch,b2), check_funds(c1) ⇒ a(bank,x1))
message(a(bank,x1), check_funds(c1) ⇐ a(branch,b2))
message(a(bank,x1), has_funds(c1) ⇒ a(branch,b2))
message(a(branch,b2), has_funds(c1) ⇐ a(bank,x1))
message(a(branch,b2), sell ⇒ a(customer,c1))
message(a(customer,c1), sell ⇐ a(branch,b2))
message(a(branch,b1), reconcile(c1) ⇒ a(bank,x1))
message(a(branch,b2), reconcile(c1) ⇒ a(bank,x1))

# Technical Things to Revise


- Expressing sentences in English as quantified logical expressions.
- Proving tautology and inconsistency using truth tables
- Given a set of proof rules, be able to apply them to produce proofs at the level of difficulty of those in the lectures (you do not need to remember the proof rules, but you do need to be able to apply them – so practice that).