Modelling systems and specifying temporal properties

a gentle introduction

Formal Methods in Software Development
Master Degree, 2017/2018
Prof. Anna Labella

Dr. Federico Mari
mari.di.uniroma1.it – mari@di.uniroma1.it

SAPIENZA
UNIVERSITÀ DI ROMA

Model Checking Laboratory Group
http://mclab.di.uniroma1.it/

Computer Science Department
Sapienza University of Rome

April 11, 2018

Agenda

- Introduction on modelling and specification
- Mutual exclusion
- The ferryman
- The alternating bit protocol

Modelling systems

Modelling Problem

Input

- requirements of real system in natural language

Output

- computational model of a system (as a finite set of automata interacting between them)

Usual problems when modelling

- Identify all single processes in the input real system
- Formalise dynamics of a single process
- Formalise communication among processes
- Choose the right level of abstraction

SAPIENZA
UNIVERSITÀ DI ROMA

Specification of properties

Specification Problem

Input

- computational model of a system
- properties to be verified on the given system

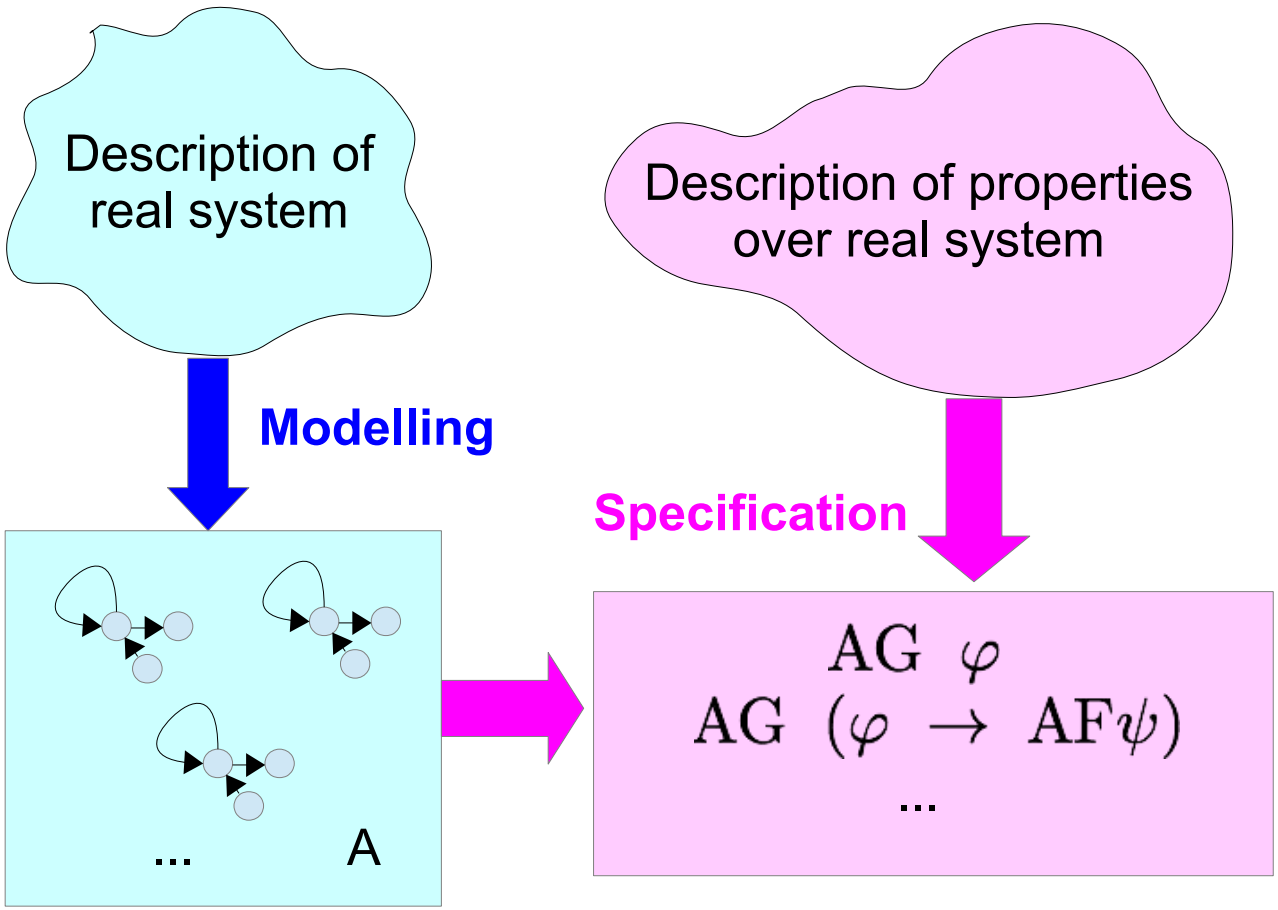
Output

- formal properties in temporal logic (LTL, CTL, CTL*, ...) modelling input properties

Usual problems in specification

- Link properties to elements of the input computational model
- Understand if a property is safety (it must hold for all states of the system), liveness (depending on evolution of time, e.g. *eventually in the future*), fairness (guaranteeing that bad behaviour does not repeat forever), ...
- Verification or planning?

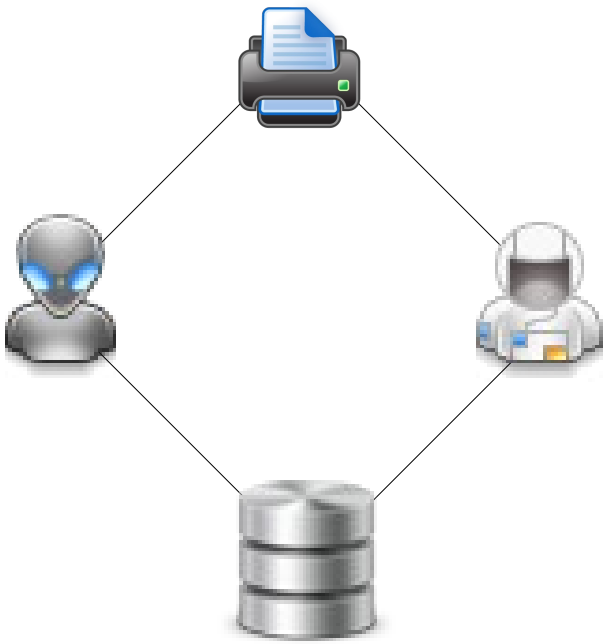
Modelling and Specification

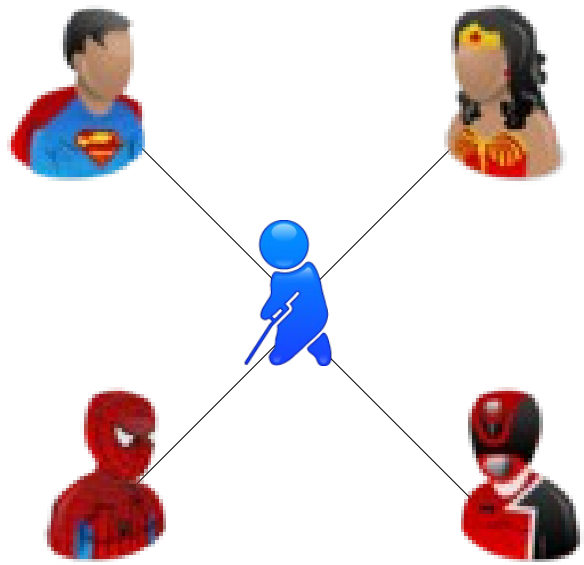


*Formal model with **some** behaviours of real system*

*Temporal logic properties specifying input properties on **abstract** model A*

Shared resources





Description of real system

- $N > 1$ processes
- 1 shared resource
- Each process wants to access the shared resource: critical session
- Each critical session must be as small as possible
- Only one process can be in its critical session at a time

Problem

Find a protocol for determining which process is allowed to enter its critical section at which time

Description of properties

Safety

Only one process is in its critical section at any time

Liveness

Whenever any process requests to enter its critical section, it will eventually be permitted to do so

Non-blocking

A process can always request to enter its critical section

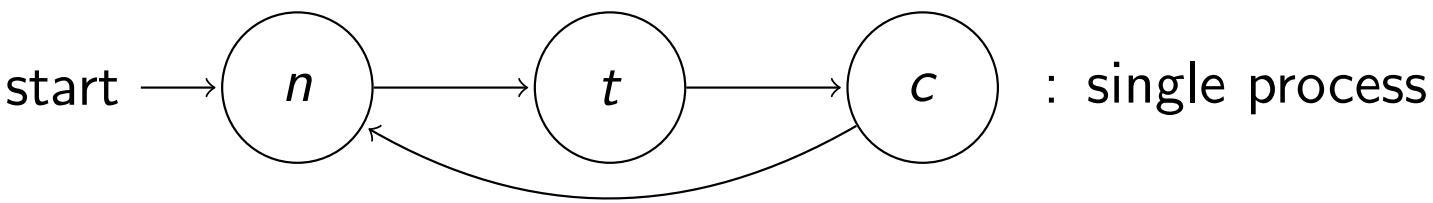
No strict sequencing

Processes need not enter their critical section in strict sequence

Modelling of real system

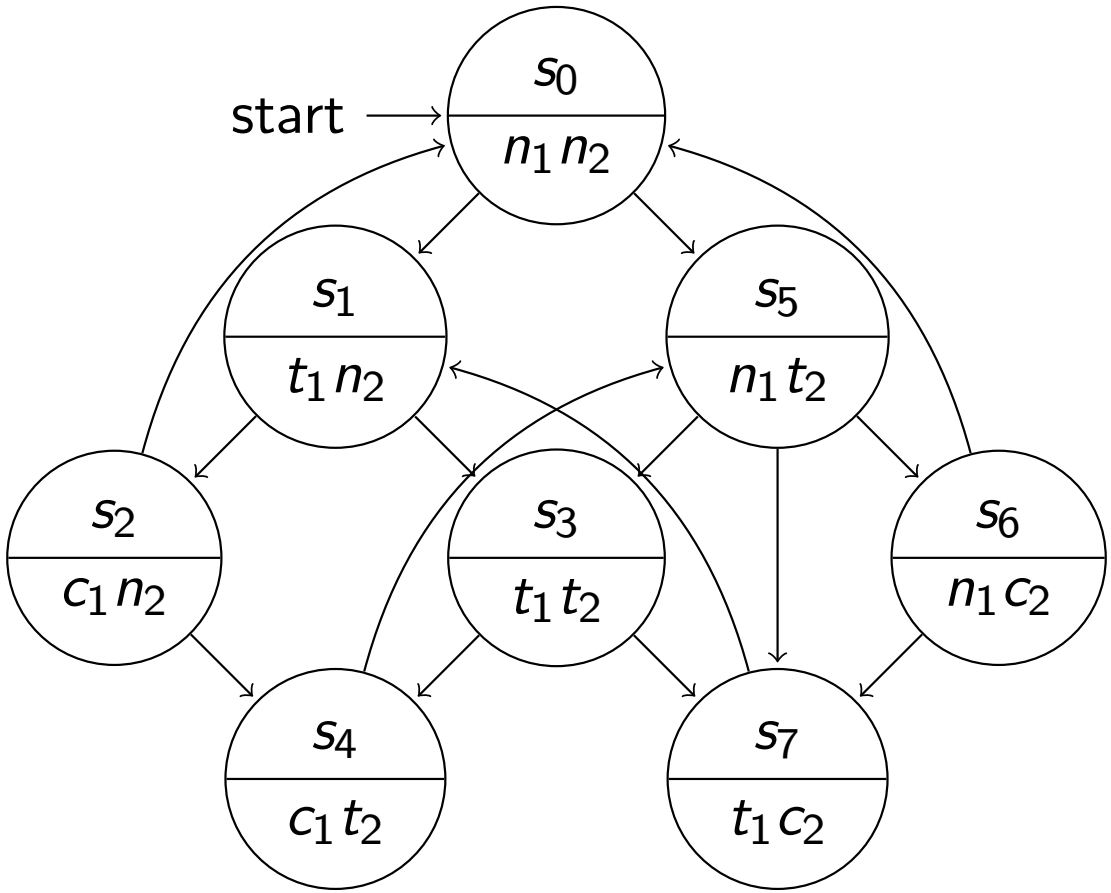
A first attempt

- 2 processes
- Each process has three states:
  - non-critical state (n)
  - trying to enter its critical state (t)
  - critical state (c)
- Each process undergoes transitions in the cycle $n \to t \to c \to n \to \ldots$



Modelling of real system

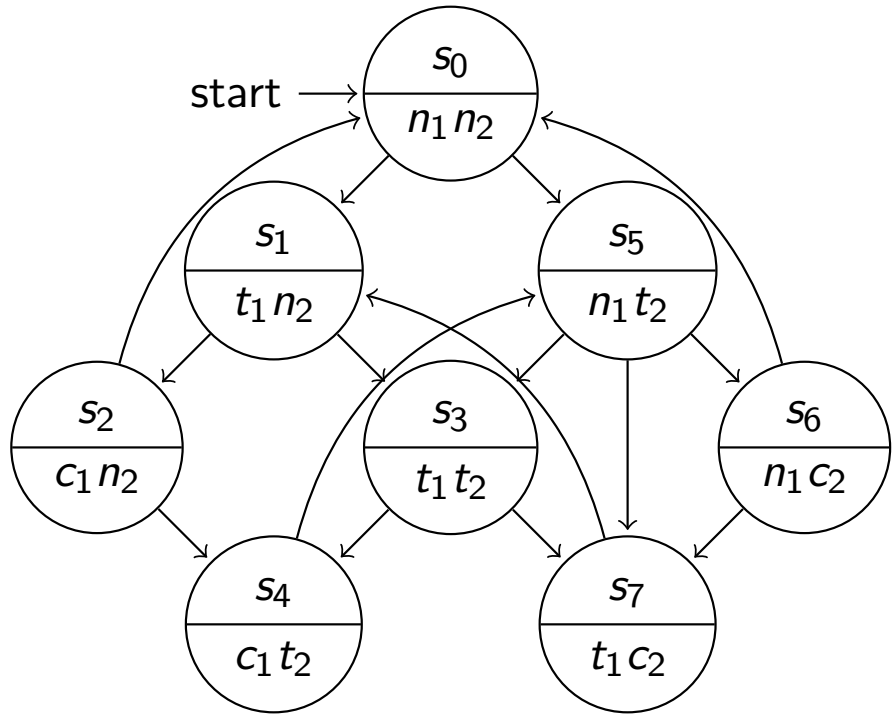
A first attempt



Specification of properties

Safety

Only one process is in its critical section at any time



Specification of properties

Safety

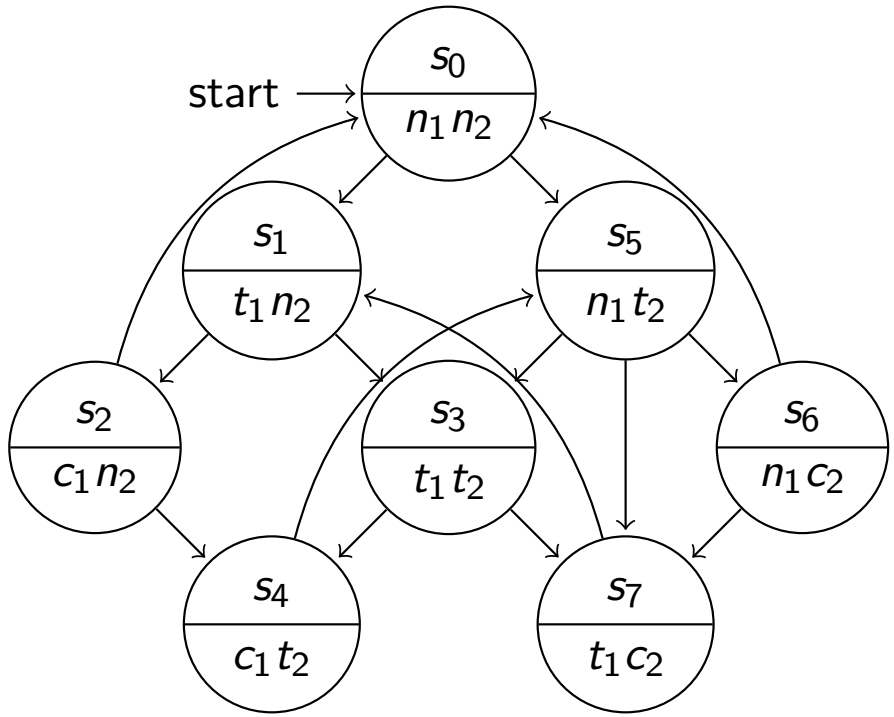
Only one process is in its critical section at any time

Specification (LTL)

$\mathbf{G}\ \neg(c_1 \wedge c_2)$

Note

Specification is True



Specification of properties

Liveness

Whenever any process requests to enter its critical section, it will eventually be permitted to do so

Specification of properties

Liveness

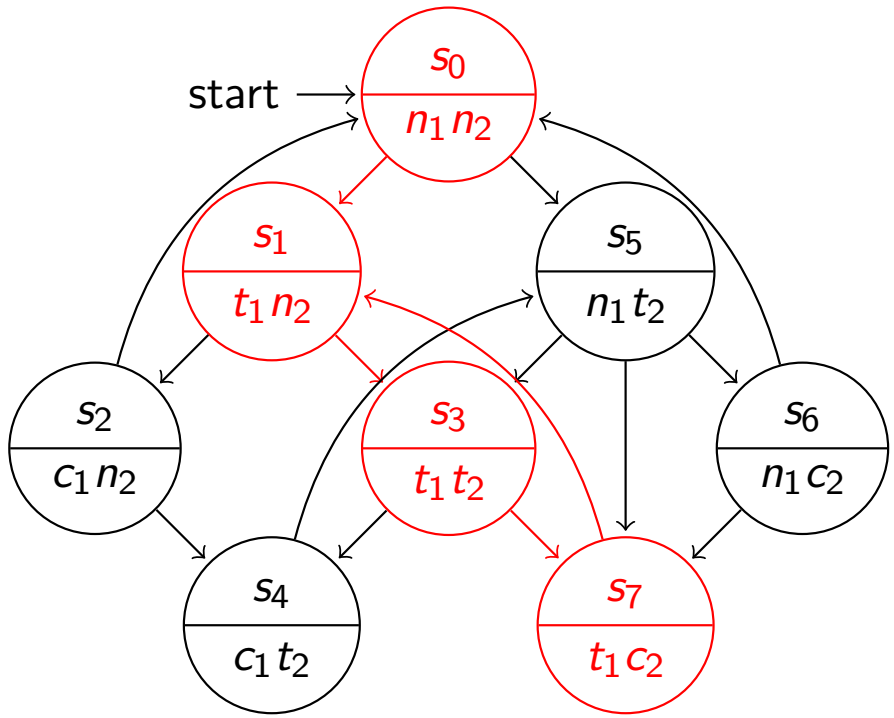
Whenever any process requests to enter its critical section, it will eventually be permitted to do so

Specification (LTL)

$\mathbf{G}\ (t_1 \rightarrow \mathbf{F}\ c_1)$

Note

Specification is False



Specification of properties

Non-blocking

A process can always request to enter its critical section

$\Rightarrow$ for each process i and for each state n_i there is a successor t_i

Specification

This existence quantifier on paths ("there is a successor satisfying...") cannot be expressed in LTL (it can be expressed in the logic CTL)

Specification of properties

No strict sequencing

Processes need not enter their critical section in strict sequence

$\Rightarrow$ There is a path with two distinct states satisfying c_1 such that no state in between them has that property

$\Rightarrow$ We cannot say in LTL "there exists a path"

But we can complement the property and check for complementation

No strict sequencing negation

All paths having a c_1 period which ends cannot have a further c_1 state until a c_2 state occurs

SAPIENZA UNIVERSITÀ DI ROMA

Specification of properties

No strict sequencing (negation)

All paths having a c_1 period which ends cannot have a further c_1 state until a c_2 state occurs

Specification of properties

No strict sequencing (negation)

All paths having a c_1 period which ends cannot have a further c_1 state until a c_2 state occurs

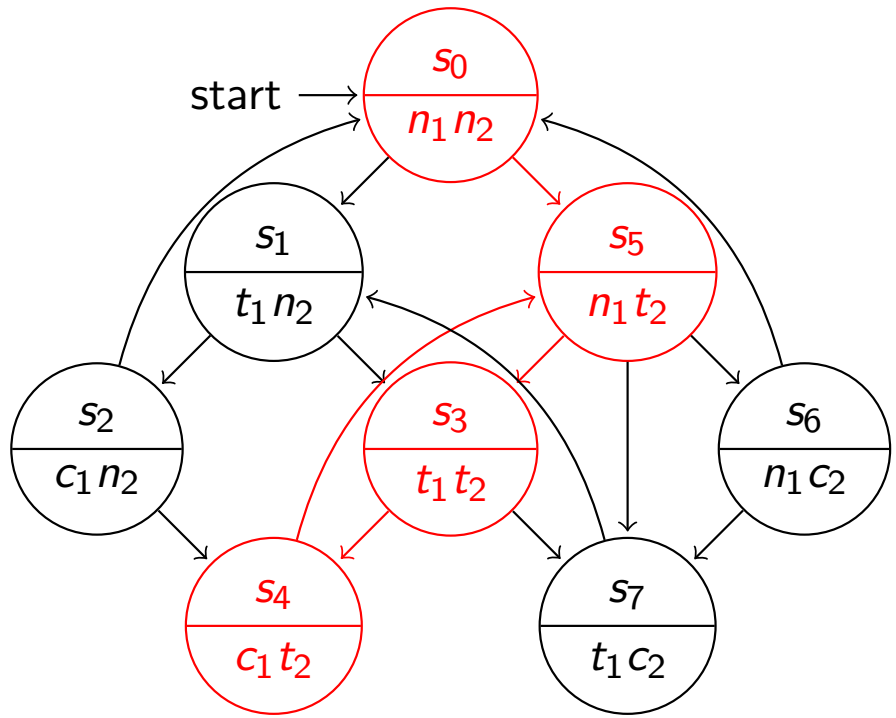
Specification (LTL)

$$\mathbf{G}\ (c_1 \rightarrow c_1 \mathbf{W}\ (\neg c_1 \wedge \neg c_1 \mathbf{W}\ c_2))$$

Note

Specification (negation) is False

$\Rightarrow$ No strict sequencing is True



Model refinement

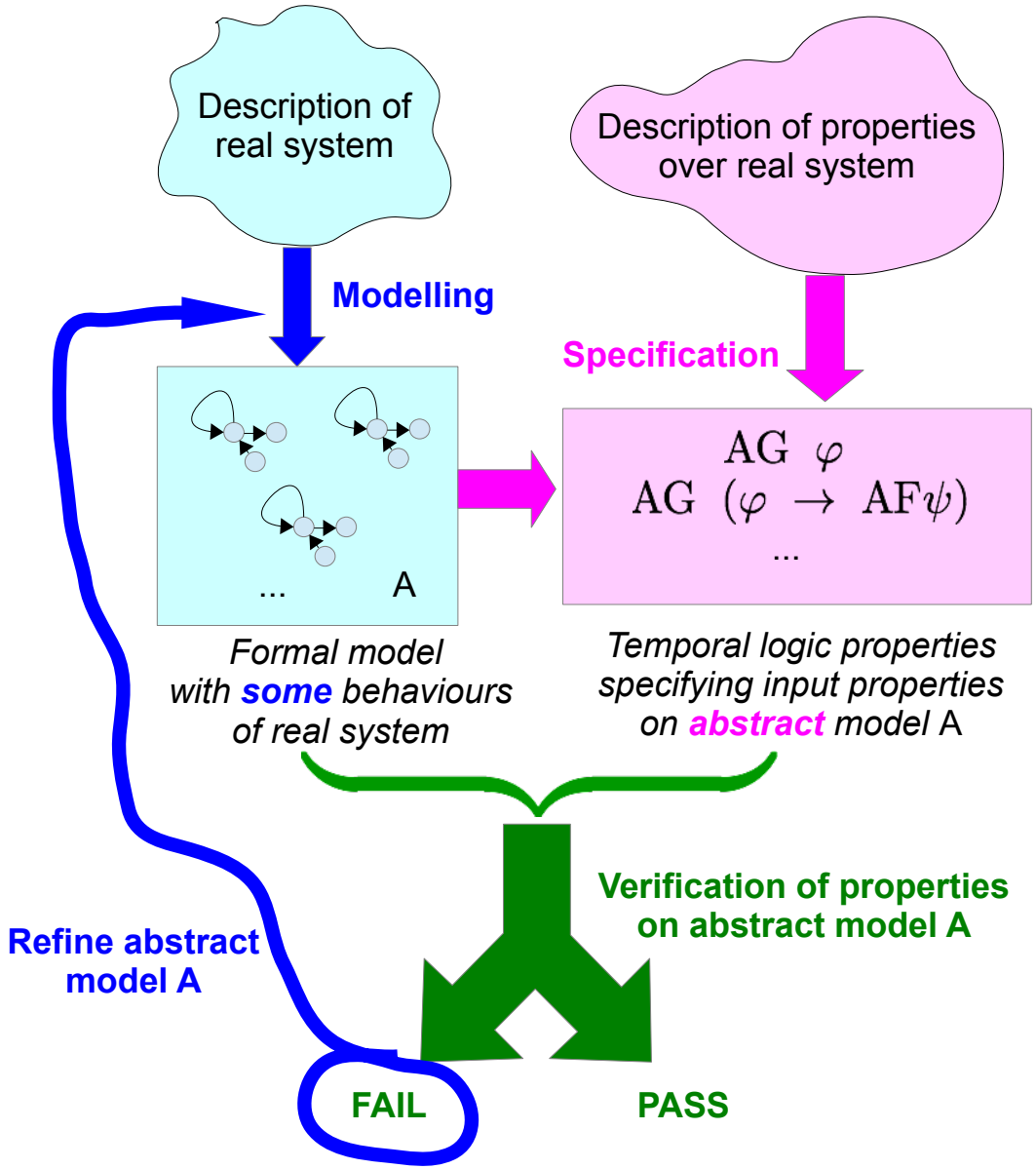
- Verification of liveness property failed on the first modelling attempt
- $\Rightarrow$ The model needs to be refined

MODEL REFINEMENT THROUGH PROPERTY VERIFICATION

Refinement of mutual exclusion first attempt model

Split s_3 into two states, distinguishing which process asked for its critical section first and giving precedence to it (exercise)

Model refinement through property verification



The ferryman

Description of real system

- Actors: ferryman, and the goods (goat, cabbage, and wolf)
- Actors must cross a river
- The ferryman can carry one good at a time on his boat
- Unsafe situations:
  - the goat and the cabbage cannot stay without ferryman on the same river bank
  - the wolf and the goat cannot stay without ferryman on the same river bank

Problem

Can the ferryman transport all the goods to the other side, without any conflicts occurring?

Description of real system

- Actors: ferryman, and the goods (goat, cabbage, and wolf)
- Actors must cross a river
- The ferryman can carry one good at a time on his boat
- Unsafe situations:
  - the goat and the cabbage cannot stay without ferryman on the same river bank
  - the wolf and the goat cannot stay without ferryman on the same river bank

Problem

Can the ferryman transport all the goods to the other side, without any conflicts occurring?

$\Rightarrow$ This is a *planning problem*

Plan

A plan is a *sequence of (state, action) pairs* such that from any *initial state* the system is driven to the *goal state*

SAPIENZA
UNIVERSITÀ DI ROMA

Description of properties

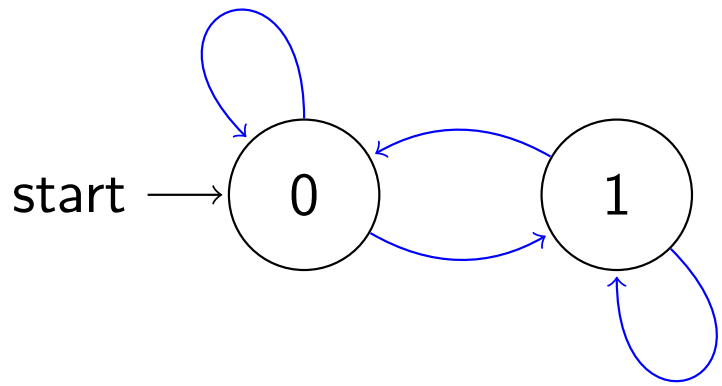
- Is there a path from the initial state such that it has a state along it at which all the goods are on the other side, and during the transitions to that state the goods are never left in an unsafe, conflicting situation?

Modelling of real system

- Each actor (agent) is modelled with an automaton with actions on edges
- Agents
  - f: ferryman
  - g: goat
  - c: cabbage
  - w: wolf
- Agents can be in the initial bank (value 0) or in the destination bank (value 1)
- We introduce variable `carry` taking a value indicating whether the goat, cabbage, wolf or nothing is carried by the ferryman

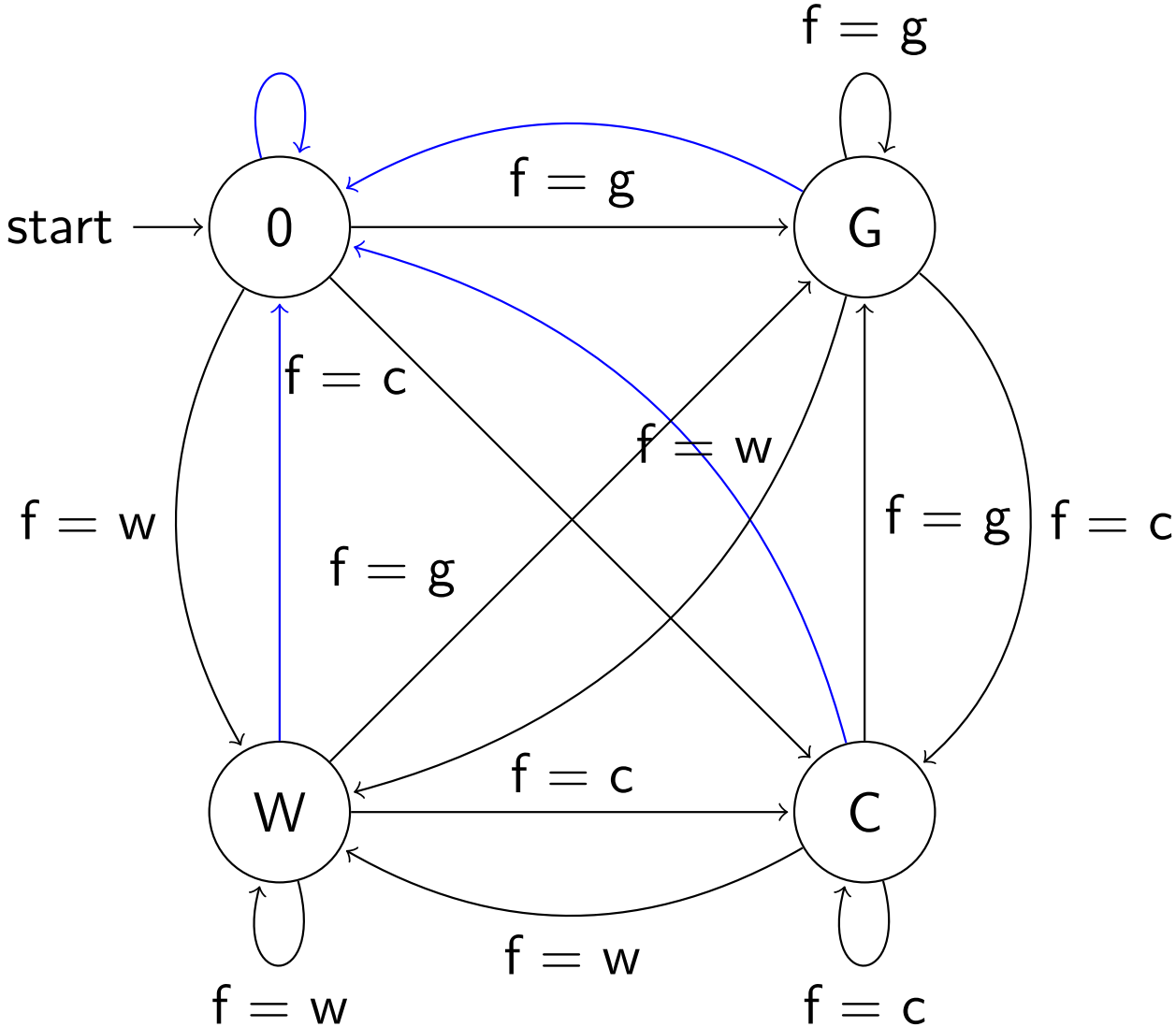
Modelling of real system

The ferryman f (blue edges are without any action)



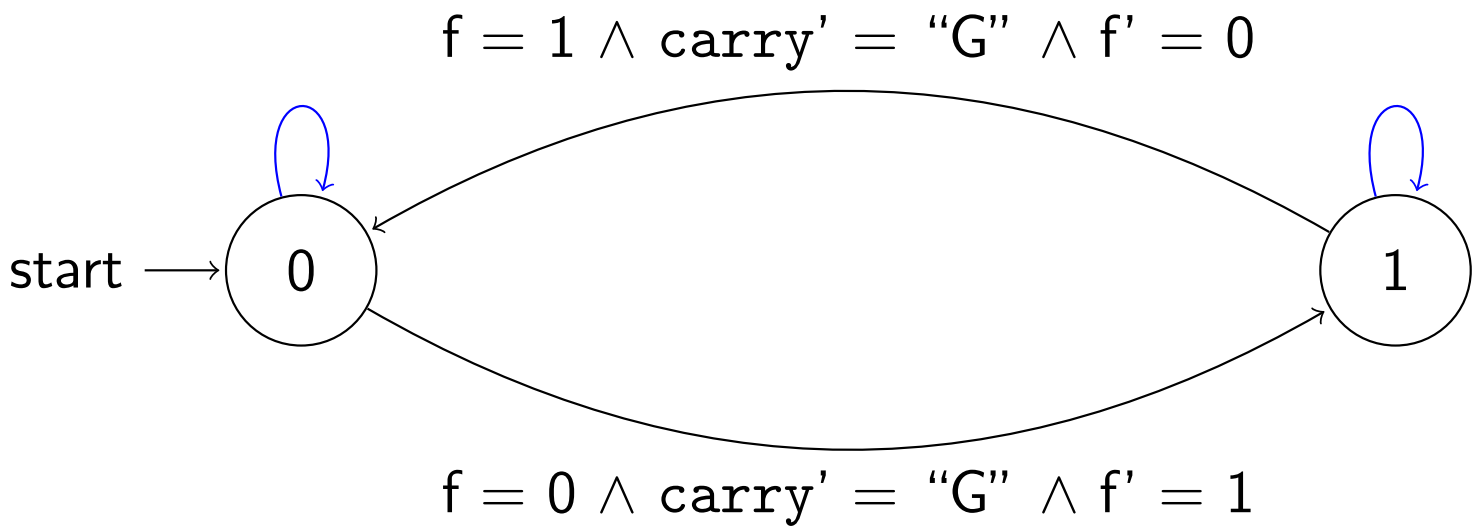
Modelling of real system

Variable `carry` (blue edges are without any action)



Modelling of real system

The goat g (blue edges are without any action)



Next state variables

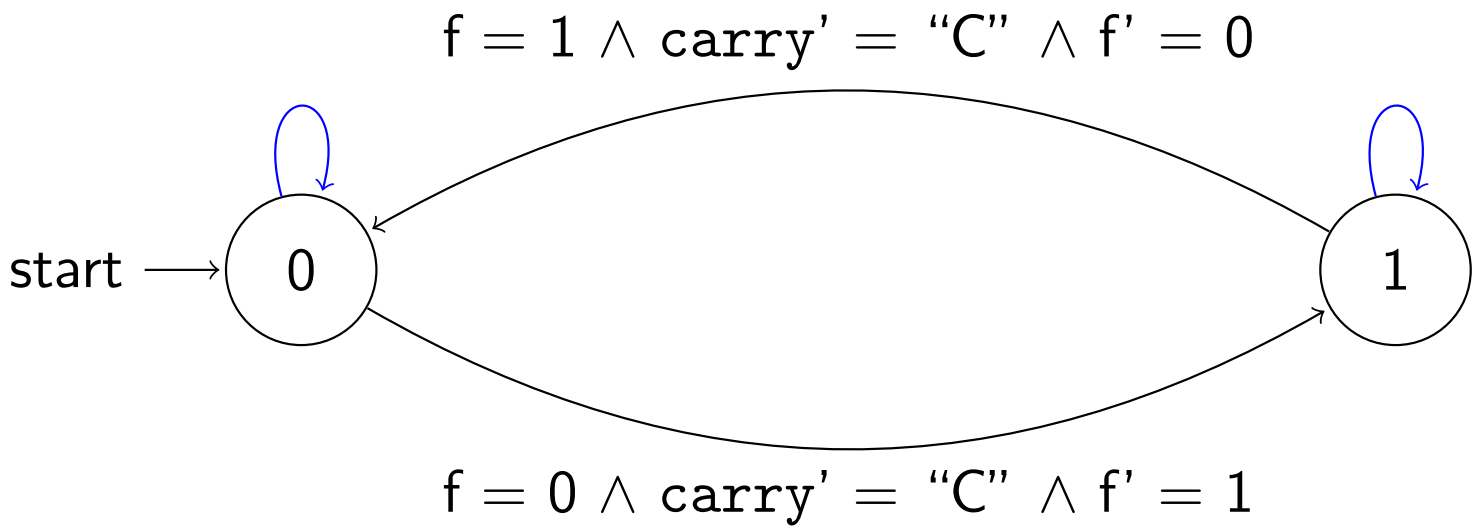
Next state variables are primed: carry', f', ...

They represent the value of corresponding variable at next time step

Non primed variables represent the value at this time step

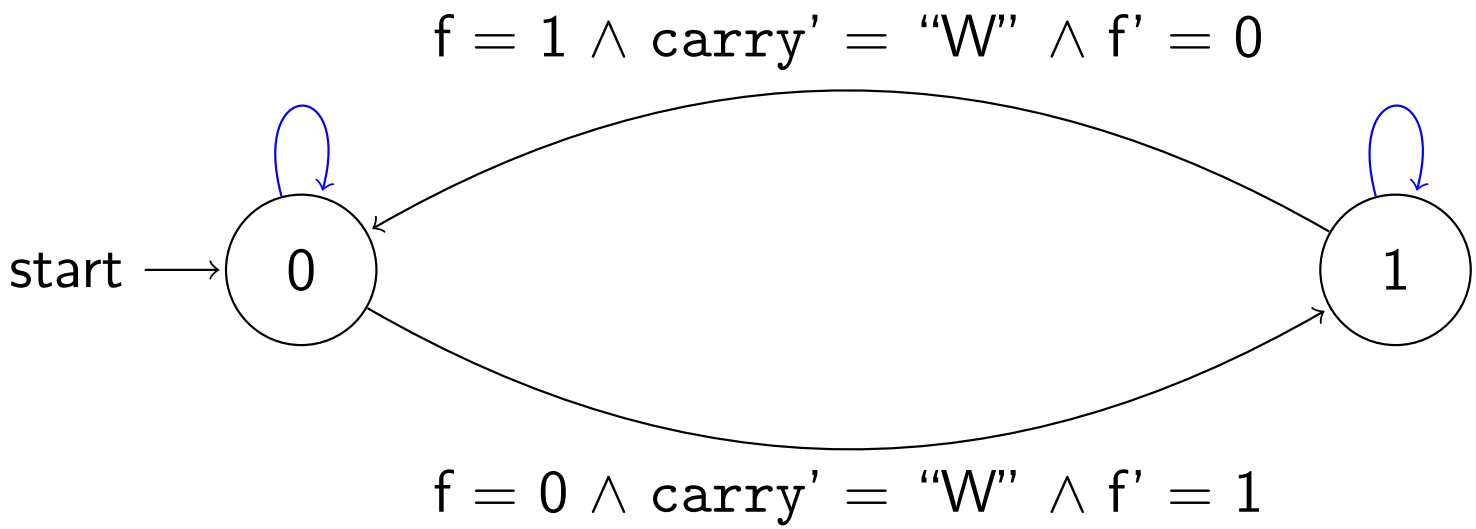
Modelling of real system

The cabbage c (blue edges are without any action)



Modelling of real system

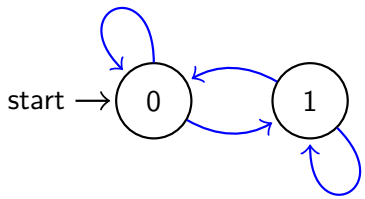
The wolf w (blue edges are without any action)



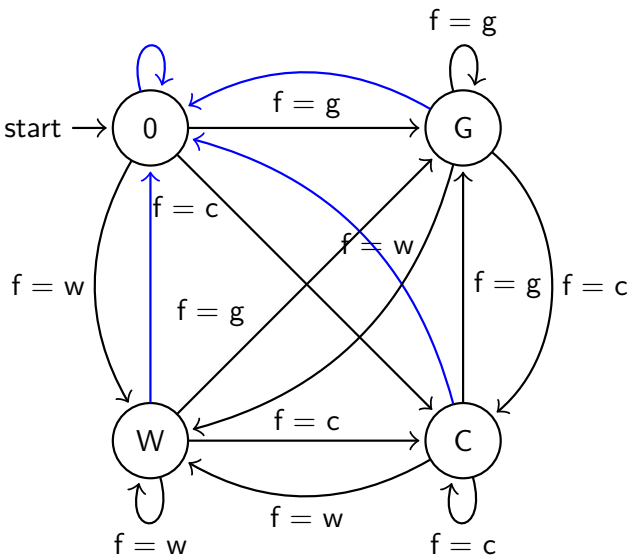
Modelling of real system

Automata composition

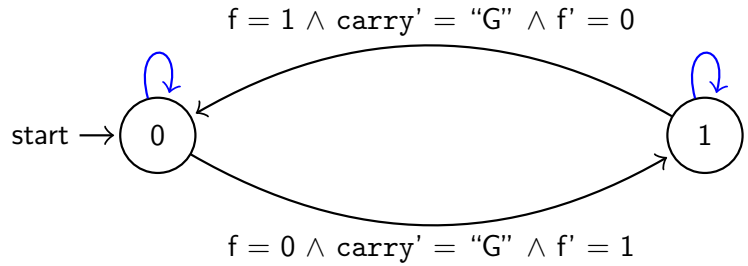
Variable f



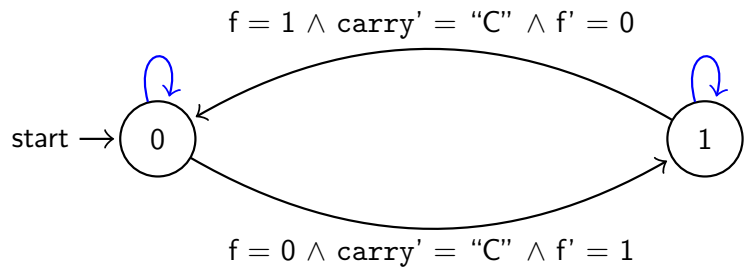
Variable carry



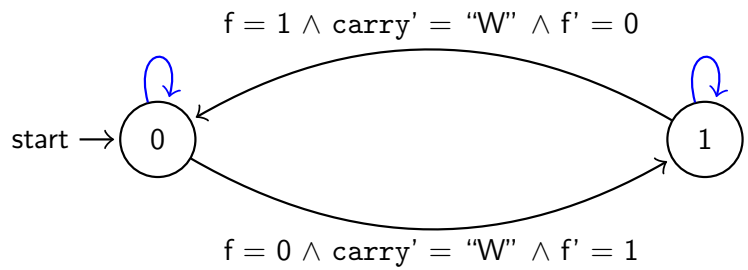
Variable g



Variable c



Variable w



SAPIENZA UNIVERSITÀ DI ROMA

Specification of properties

Strategy to avoid unsafe situation

If the goat and the cabbage, or the wolf and the goat, are on the same river bank then the goat is with the ferryman

Planning problem goal

The goat, the cabbage, and the wolf are on the other river bank (value 1)

Specification of properties

Strategy to avoid unsafe situation

If the goat and the cabbage, or the wolf and the goat, are on the same river bank then the goat is with the ferryman

Planning problem goal

The goat, the cabbage, and the wolf are on the other river bank (value 1)

Specification (LTL)

$((g = c \vee w = g) \rightarrow g = f)\ \mathbf{U}\ (f \wedge g \wedge c \wedge w)$

Specification of properties

Strategy to avoid unsafe situation

If the goat and the cabbage, or the wolf and the goat, are on the same river bank then the goat is with the ferryman

Planning problem goal

The goat, the cabbage, and the wolf are on the other river bank (value 1)

Specification (LTL)

$((g = c \vee w = g) \rightarrow g = f)\ \mathbf{U}\ (f \wedge g \wedge c \wedge w)$

Specification for planning problem (LTL)

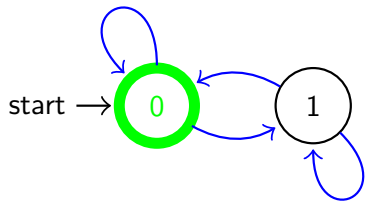
$\neg(((g = c \vee w = g) \rightarrow g = f)\ \mathbf{U}\ (f \wedge g \wedge c \wedge w))$

$\Rightarrow$ Finding a counterexample yields to a plan

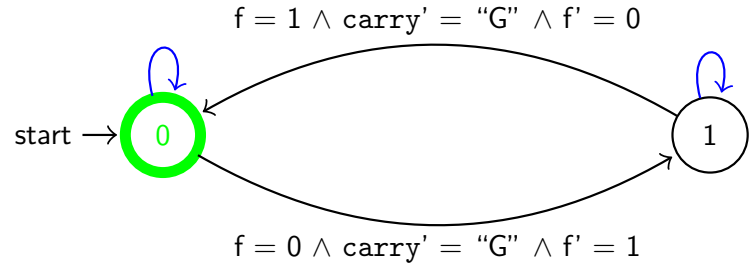
Plan

Step 1

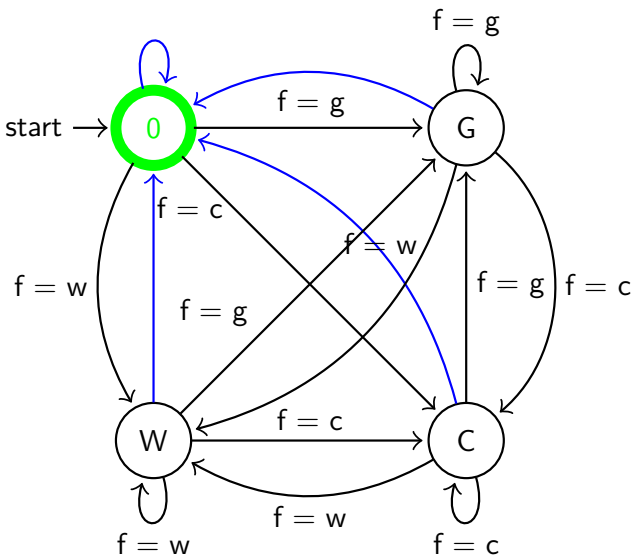
Variable f



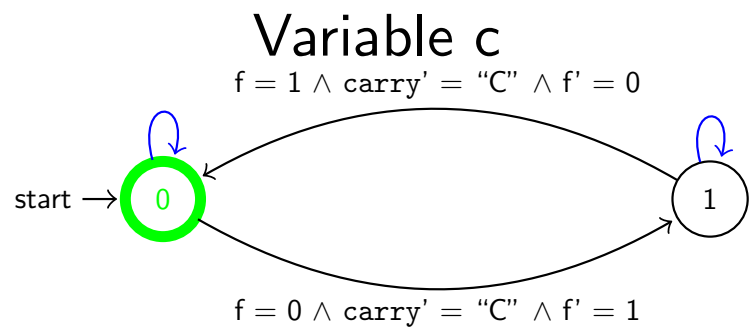
Variable g



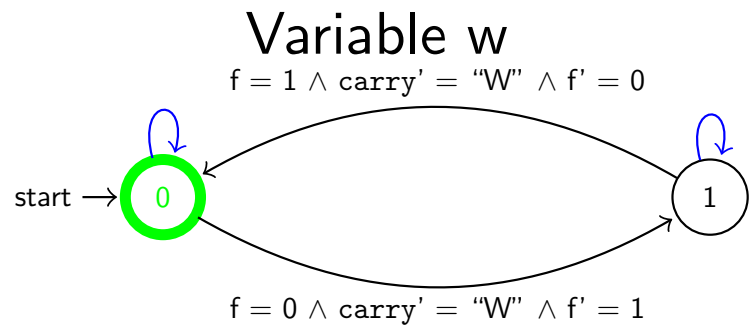
Variable carry



Variable c



Variable w

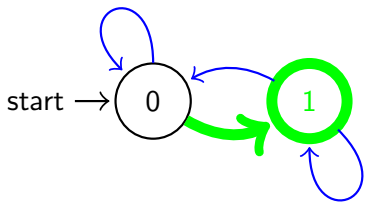


SAPIENZA UNIVERSITÀ DI ROMA

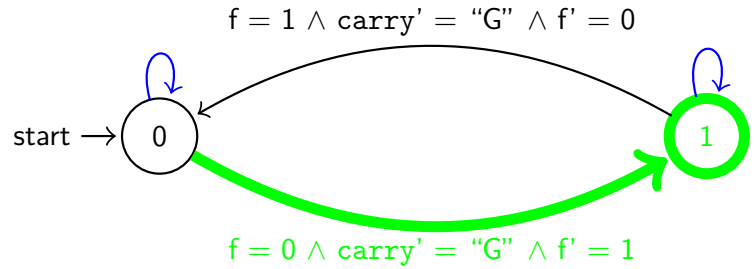
Plan

Step 2

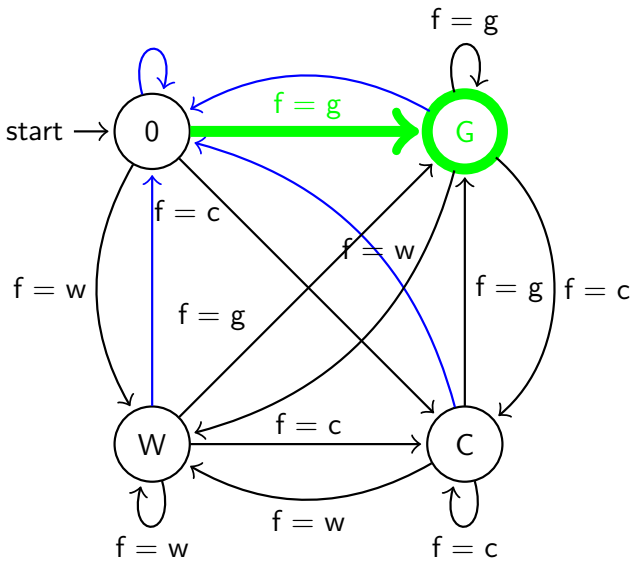
Variable f



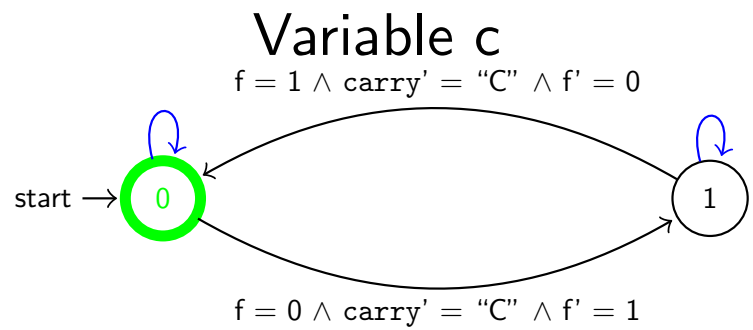
Variable g



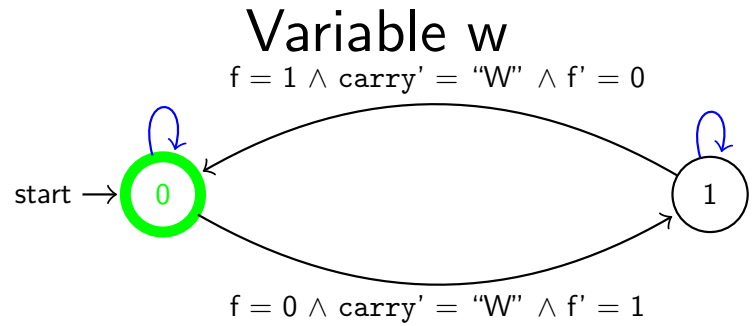
Variable carry



Variable c



Variable w

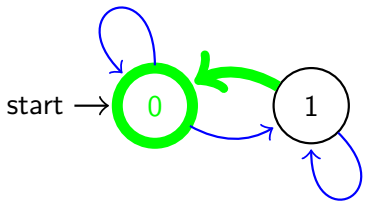


SAPIENZA UNIVERSITÀ DI ROMA

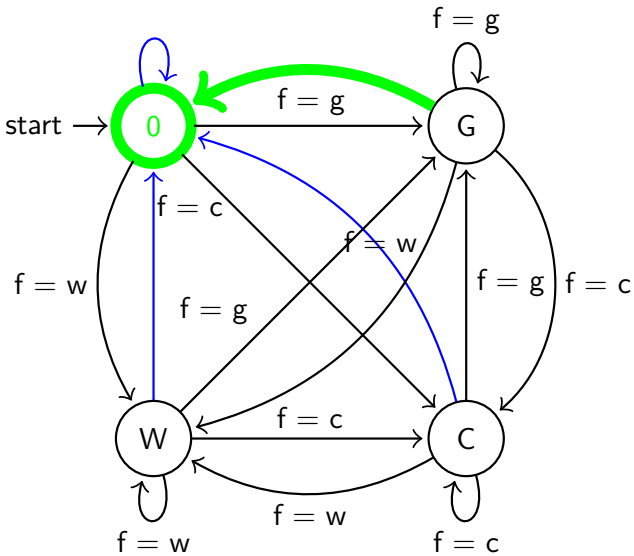
Plan

Step 3

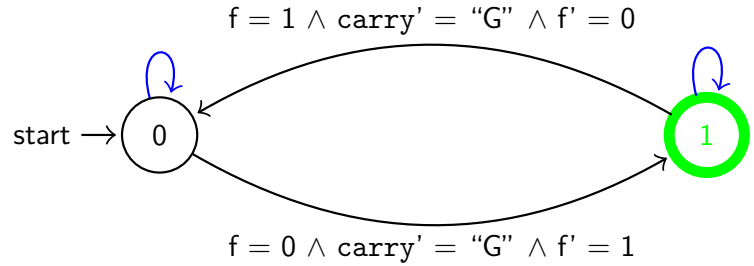
Variable f



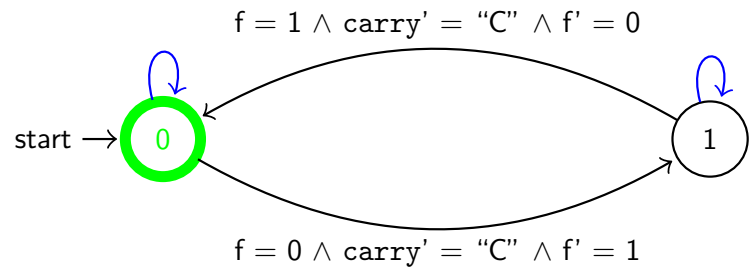
Variable carry



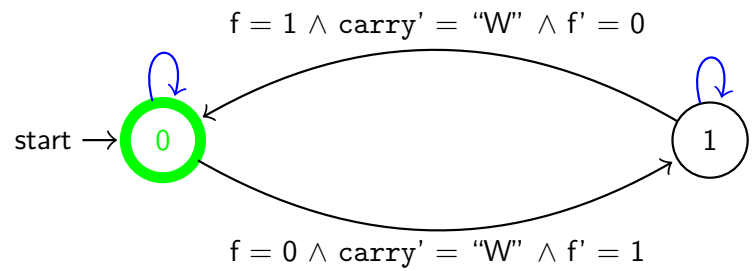
Variable g



Variable c



Variable w

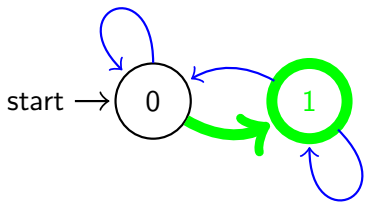


SAPIENZA UNIVERSITÀ DI ROMA

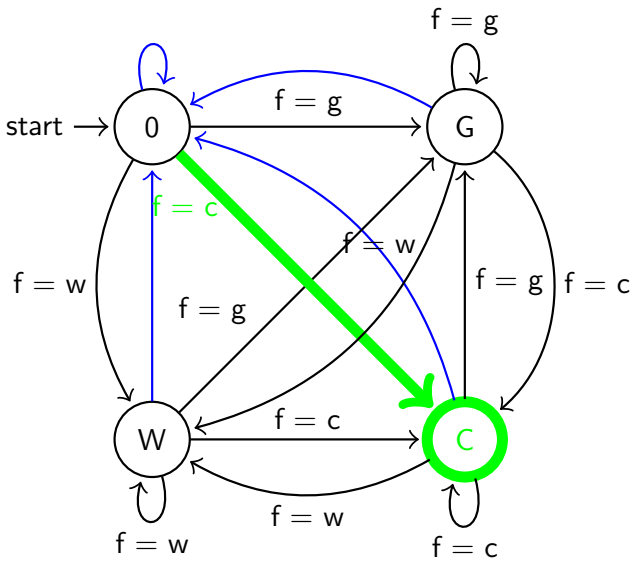
Plan

Step 4

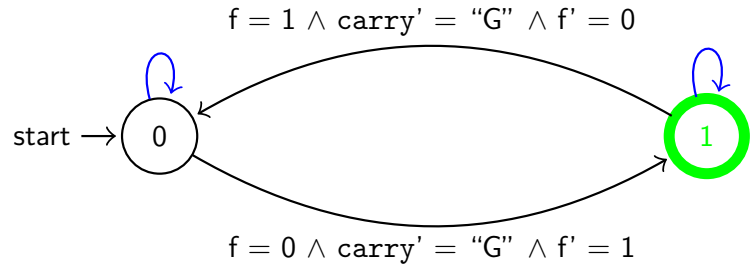
Variable f



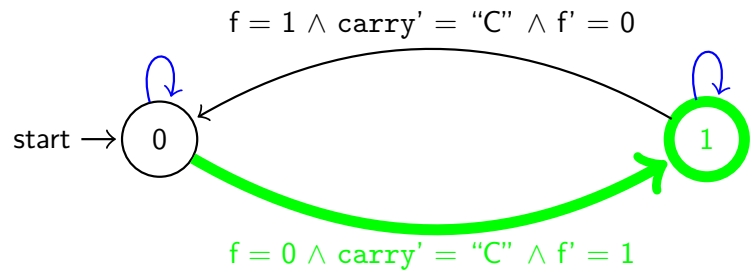
Variable carry



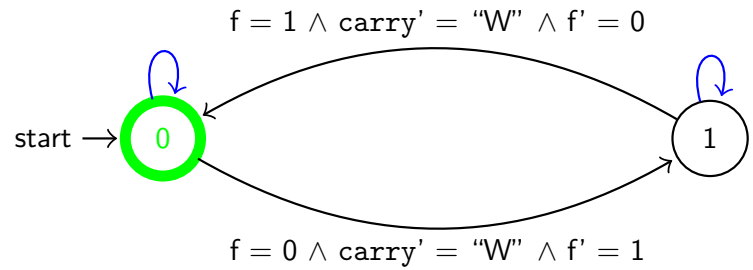
Variable g



Variable c



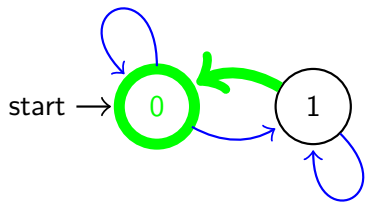
Variable w



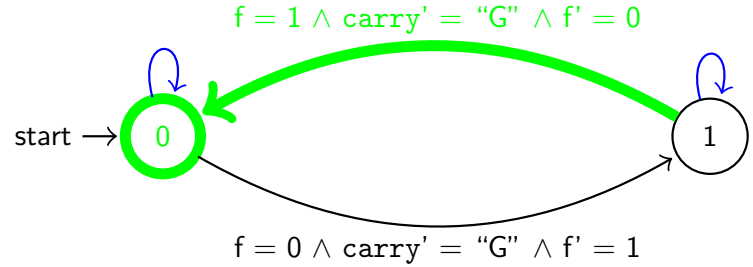
Plan

Step 5

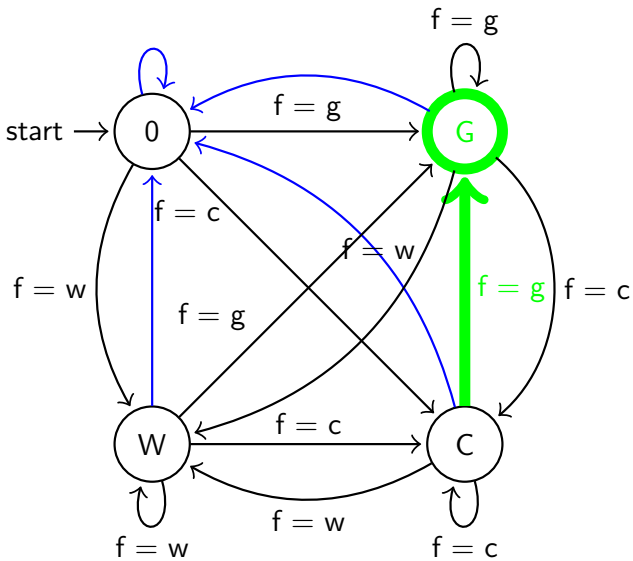
Variable f



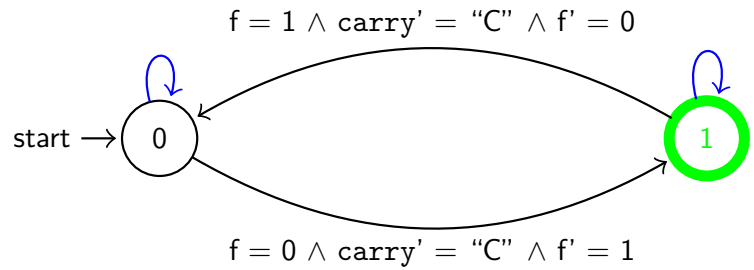
Variable g



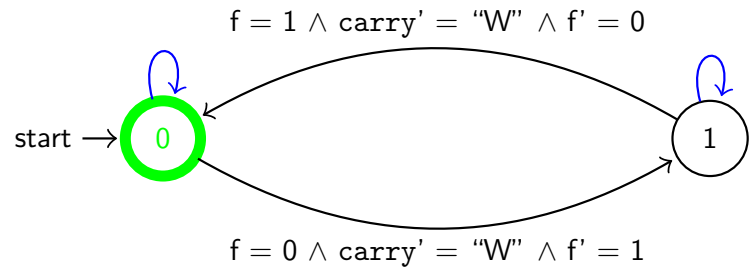
Variable carry



Variable c



Variable w

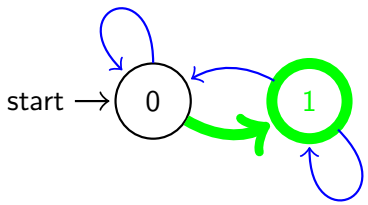


SAPIENZA UNIVERSITÀ DI ROMA

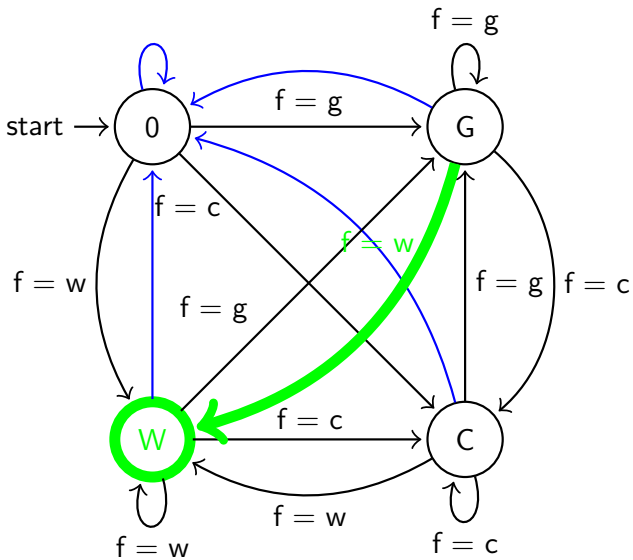
Plan

Step 6

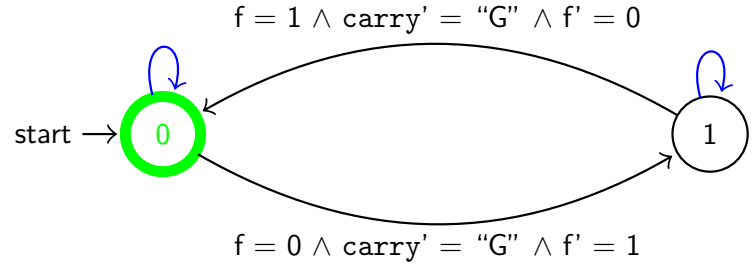
Variable f



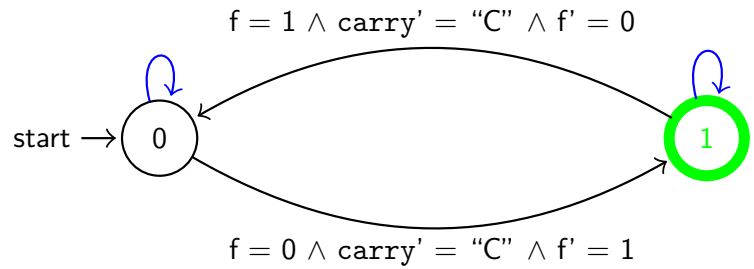
Variable carry



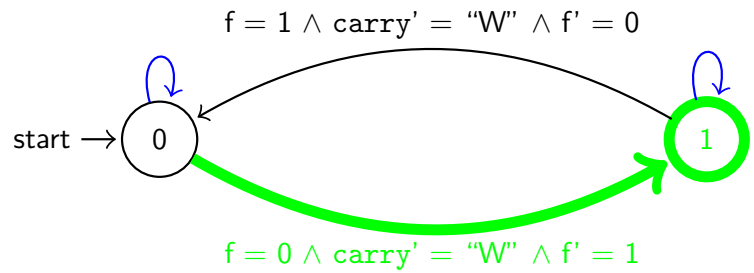
Variable g



Variable c



Variable w

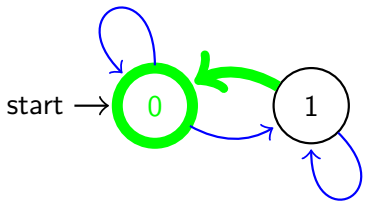


SAPIENZA UNIVERSITÀ DI ROMA

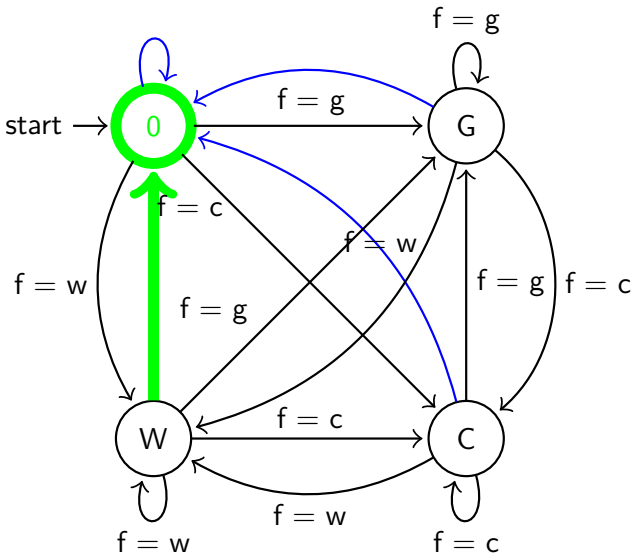
Plan

Step 7

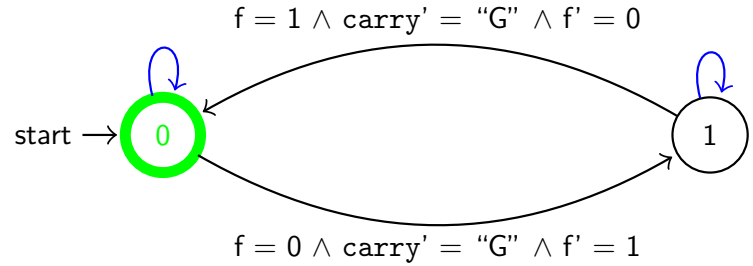
Variable f



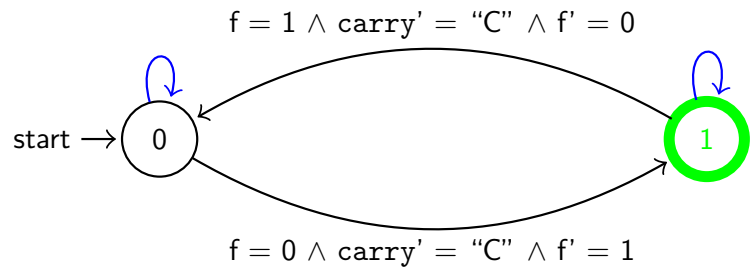
Variable carry



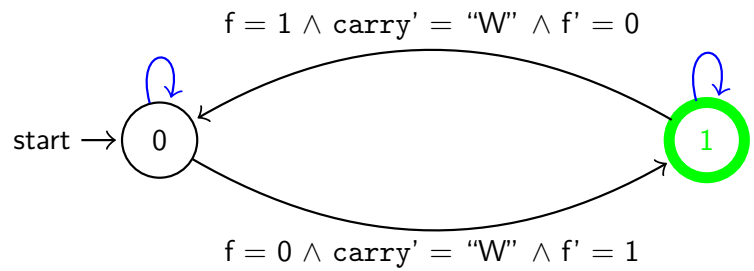
Variable g



Variable c



Variable w

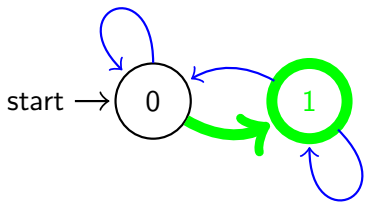


SAPIENZA UNIVERSITÀ DI ROMA

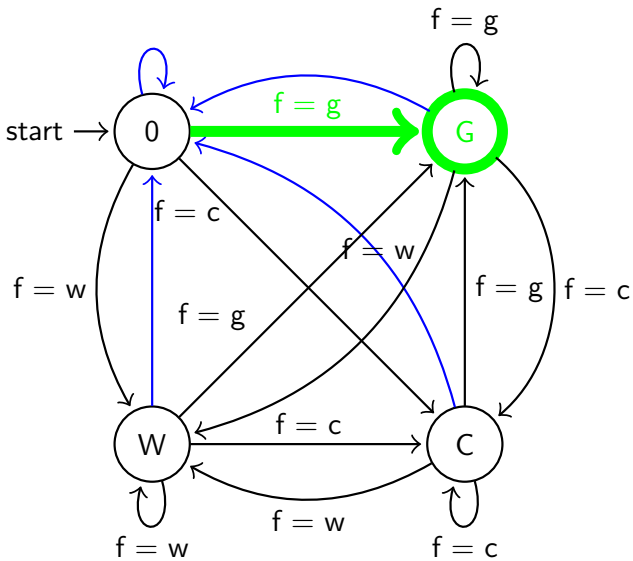
Plan

Step 8

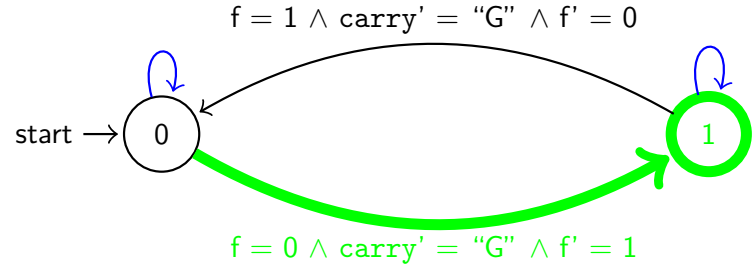
Variable f



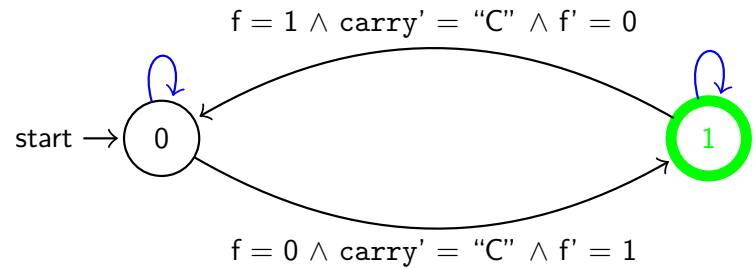
Variable carry



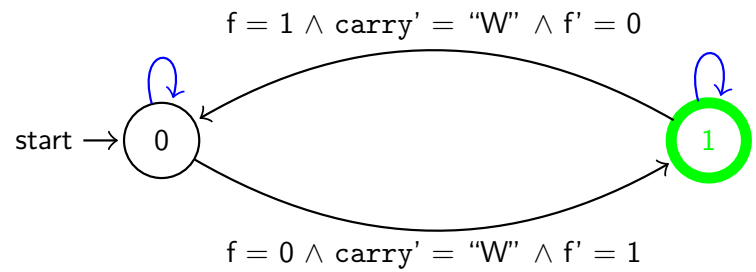
Variable g



Variable c

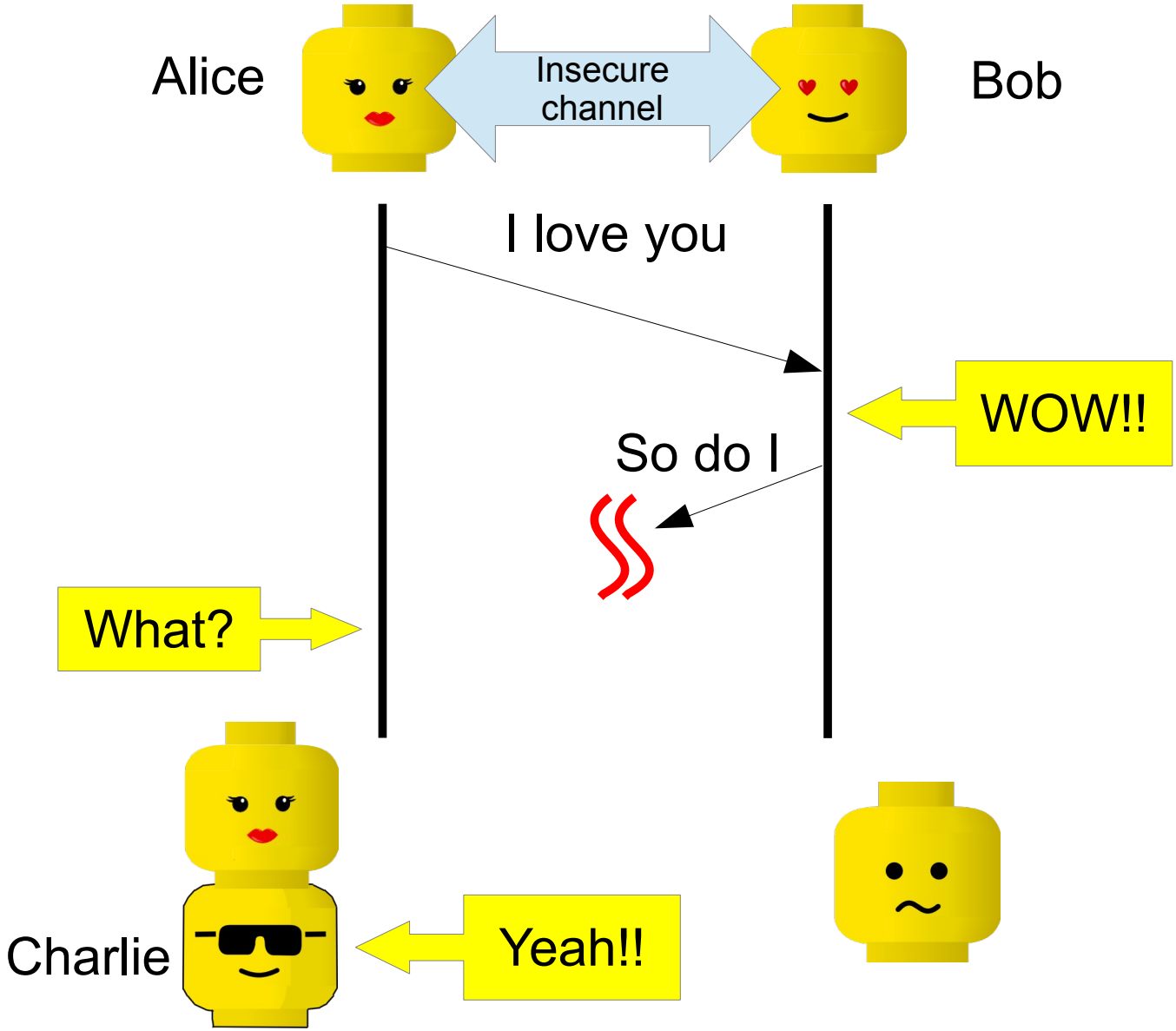


Variable w



SAPIENZA UNIVERSITÀ DI ROMA

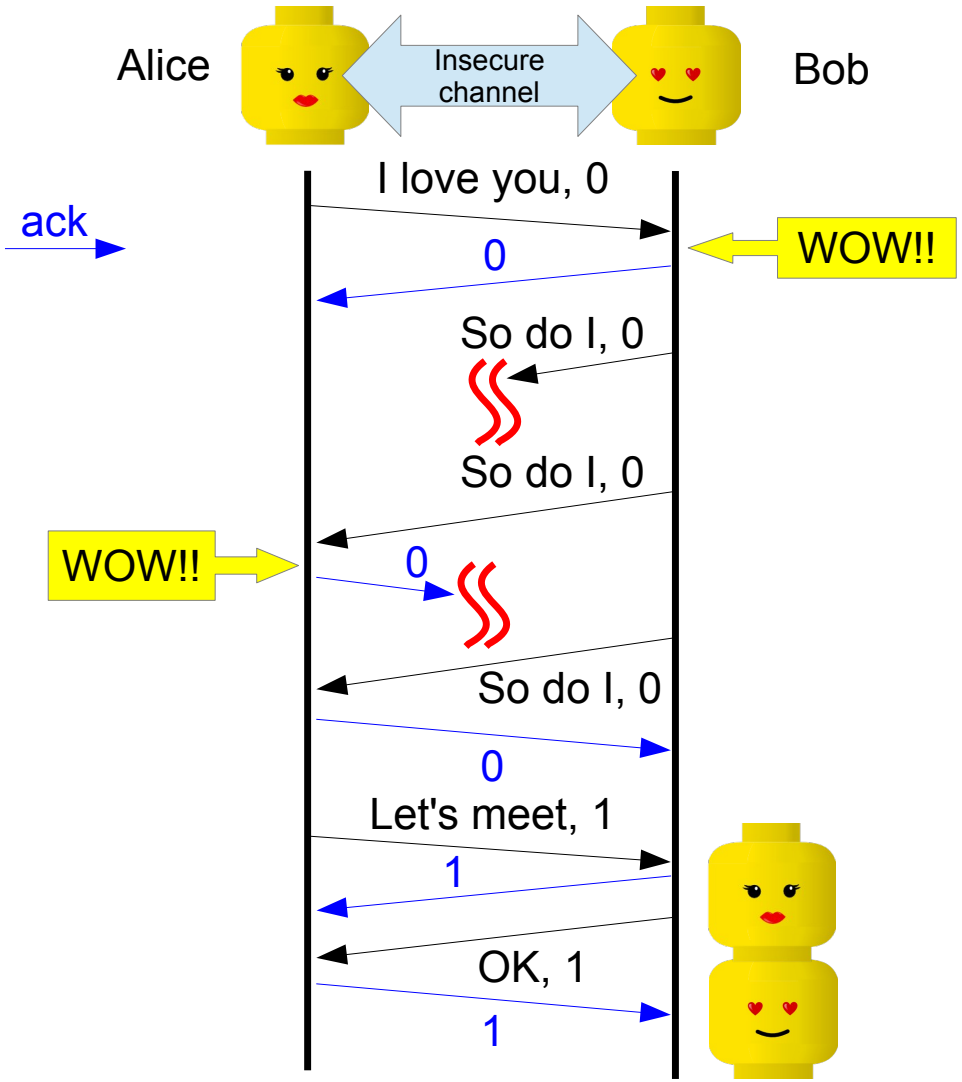
The alternating bit protocol: motivations



Description of real system

- Alice and Bob want to communicate
- On a lossy line, i.e. a line which may lose or duplicate messages
- The line does not lose infinitely many messages (fairness)
- The line does not corrupt messages
- Protocol uses acknowledgements

Description of real system



- Agents
  - Sender
  - Receiver
  - Message channel
  - Ack channel
- Figure shows two instances of Alternating Bit Protocol (ABP)
  1. Sender: Alice, Receiver: Bob
  2. Sender: Bob, Receiver: Alice
- Sender sends again the same message until he receives the corresponding ack
- Receiver sends again the same ack until he receives a message with expected bit

Description of properties

Safety

If the message bit 1 has been sent and the correct acknowledgement has been returned, then a 1 was indeed received by the receiver (the same holds for 0)

Liveness

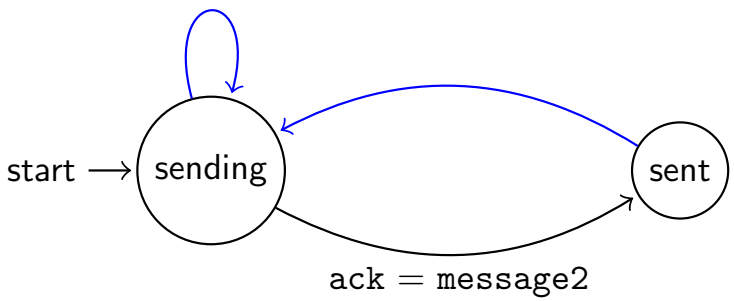
Messages get through eventually
$\Rightarrow$ For any state there is inevitably a future state in which the current message has got through
Similarly, *acknowledgements* get through eventually

Modelling of real system

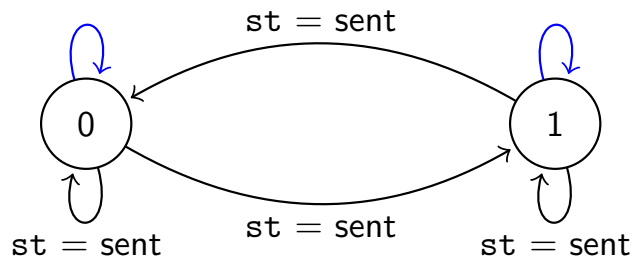
The sender

Input variable `ack` boolean

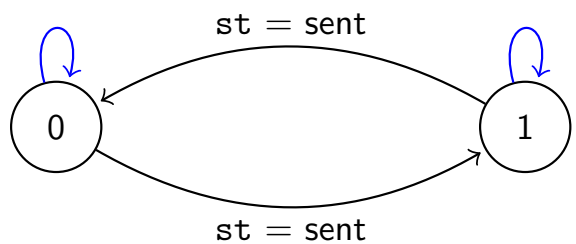
Sending status (`st`)



Variable `message1`



Variable `message2`



- The sender sends the bit in variable `message1`
- The ABP control bit is in variable `message2`
- Fairness: The sender must be run infinitely often

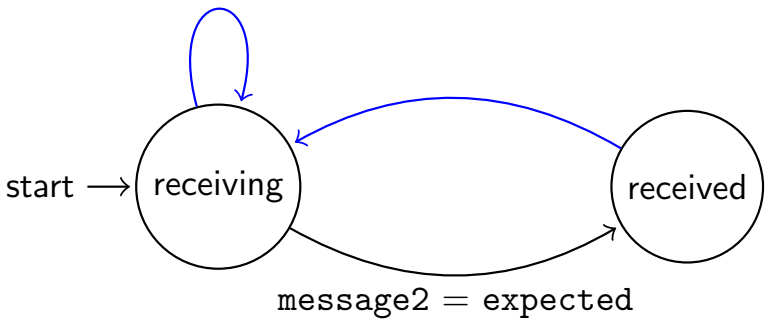
Modelling of real system

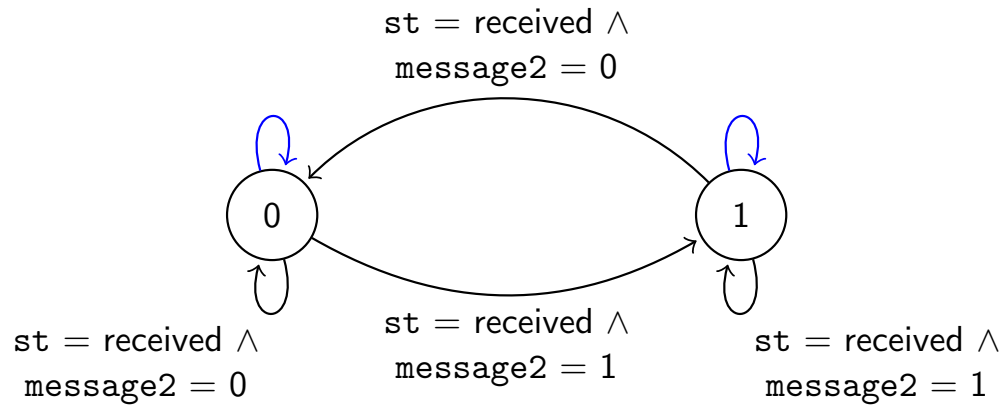
The receiver

Input variables `message1` (actual message), `message2` (ABP control bit) boolean

Variable `ack`

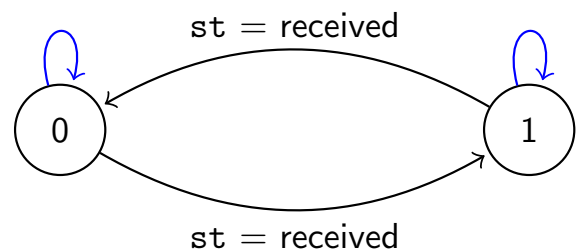
Receiving status (`st`)





Variable `expected`

Fairness: The receiver must be run infinitely often

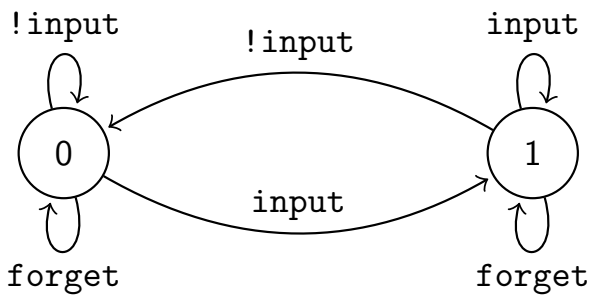


Modelling of real system

The acknowledgement channel (1 bit)

Input variable `input` boolean

Variable `output`



Variable `forget`

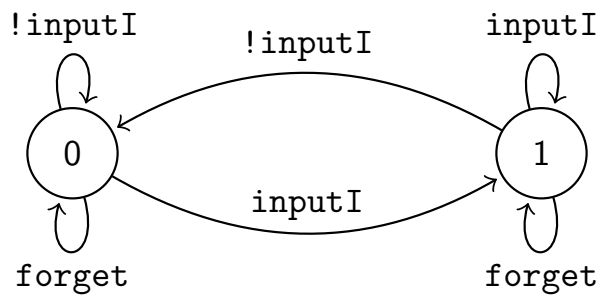
- Fairness: The one bit channel must be run infinitely often
- Fairness: Variable `forget` cannot be always 1, for both input 1 and 0

Modelling of real system

The message channel (2 bits)

Input variables `input1`, `input2` boolean

Variables `outputI`, `I` in $\{1, 2\}$

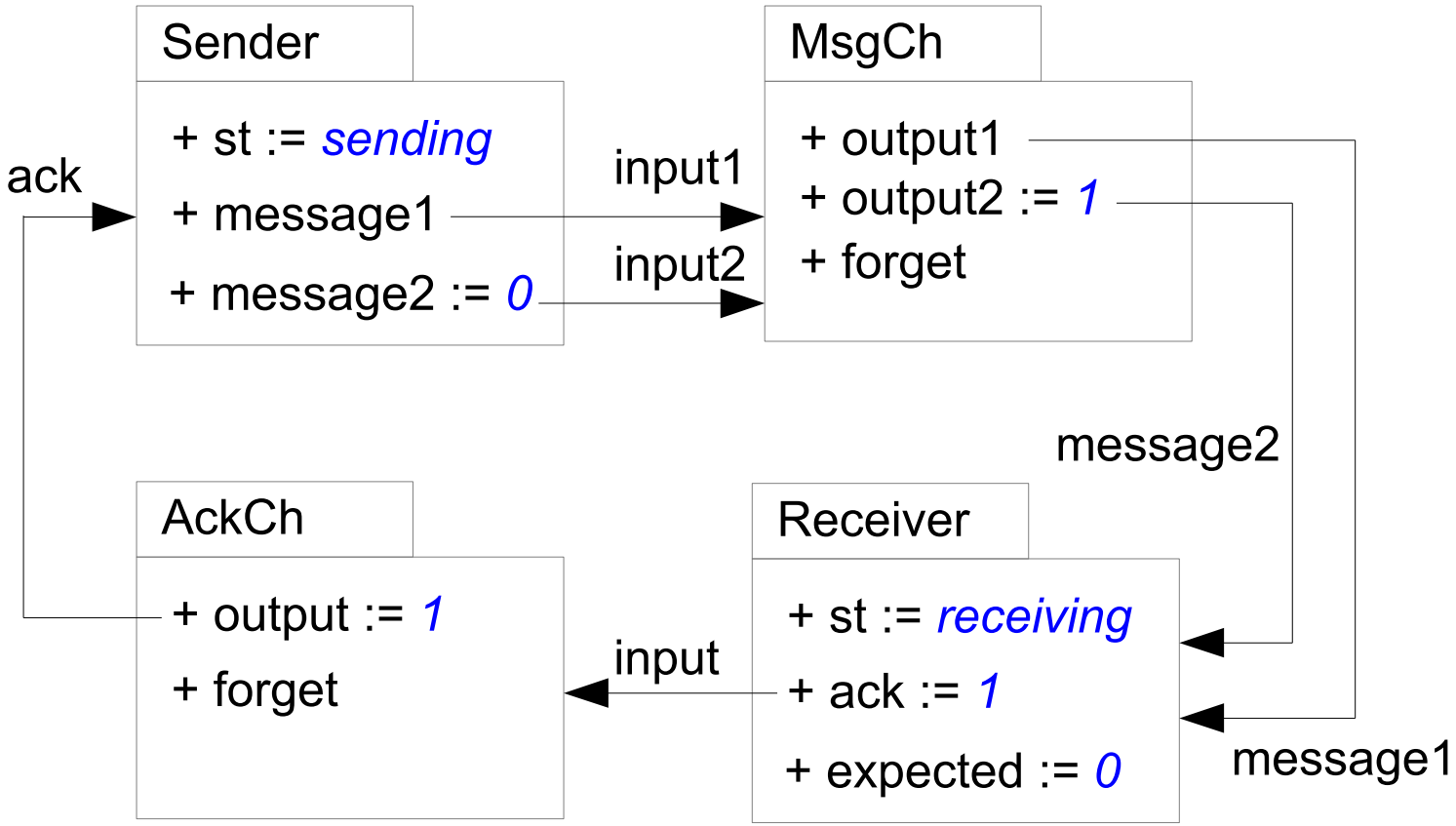


Variable `forget`

- Fairness: The two bit channel must be run infinitely often
- Fairness: Variable `forget` cannot be always 1, for all inputs (4 cases)

Modelling of real system

The whole protocol



Note

Variable `forget` models insecure channel (non-deterministic)

Specification of properties

Safety

If the message bit 1 has been sent and the correct acknowledgement has been returned, then a 1 was indeed received by the receiver (the same holds for 0)

Specification of properties

Safety

If the message bit 1 has been sent and the correct acknowledgement has been returned, then a 1 was indeed received by the receiver (the same holds for 0)

Specification (LTL)

$$\mathbf{G}\ (\text{Sender.}\mathtt{st} = \text{sent} \wedge \text{Sender.}\mathtt{message1} = 1 \rightarrow \text{MsgCh.}\mathtt{output1} = 1)$$

Note

Safety specification is True

Specification of properties

Liveness

Messages get through eventually

$\Rightarrow$ For any state there is inevitably a future state in which the current message has got through

Similarly, *acknowledgements* get through eventually

Specification of properties

Liveness

Messages get through eventually

$\Rightarrow$ For any state there is inevitably a future state in which the current message has got through

Similarly, *acknowledgements* get through eventually

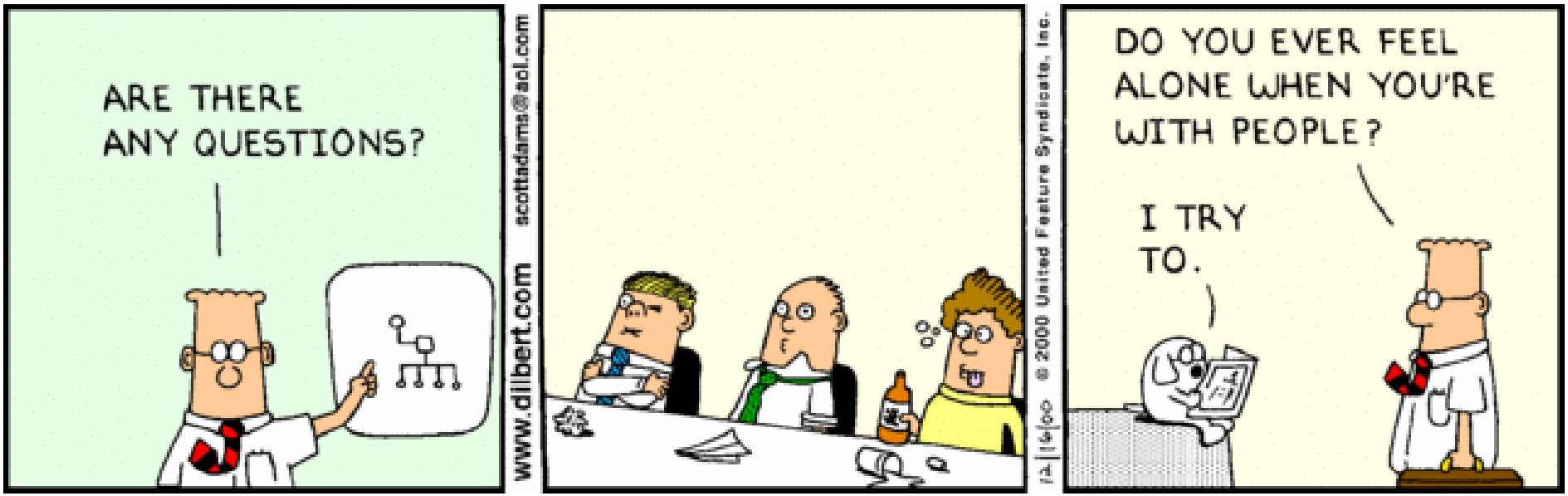
Specification (LTL)

G F (Sender.st = sent)

G F (Receiver.st = received)

Note

Liveness specification is True


ARE THERE ANY QUESTIONS?
www.dilbert.com scottadams@aol.com
DO YOU EVER FEEL ALONE WHEN YOU'RE WITH PEOPLE?
I TRY TO.
© 2000 United Feature Syndicate, Inc.