innovate achieve lead

BITS Pilani
Pilani Campus

CS/IS F214
Logic in Computer Science

MODULE: TEMPORAL LOGICS

LTL – Semantics and Model Checking

LTL – Model Checking: Input Simplification

- Before we look at the model checking algorithm, we simplify the syntax of the input formulas by using an adequate set of operators:
  - [Propositional:] $\wedge$ and $\neg$
  - [Temporal:] **X** , **G**, and **U**.

- Propositional operators such as $\vee$ and --> can be eliminated:
  - $\wedge$ and $\neg$ form an adequate set.

- What about the other temporal operators?
  - e.g. They can be rewritten using **U**.

LTL - Automating Evaluation of Formulas

- Should the evaluation algorithm *generate one path at a time* and *evaluate the formula in that path*?
- i.e.
  - repeat
    - generate a path π and evaluate ϕ in π
  - until (*there are no more paths*)

- *This won't work*
  - because each path is infinite and evaluating in one path may not terminate!

LTL - Automating Evaluation of Formulas

- But as it turns out:
  - *we can traverse the state machine (i.e. the graph) while evaluating the formula*!

- This is achieved by a ***marking*** or a ***labeling*** algorithm:
  - *traverse the graph* that is the state machine:
    - mark (or label) each state **s** with the (sub-)formulas that are satisfied in **s**.

- Will this terminate?
  - *Note that there is only a finite number of states*!

LTL – Model Checking Algorithm

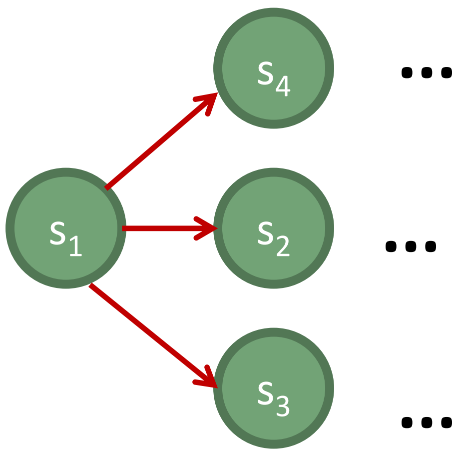
- Thus we end up with a model-checking algorithm which
  - takes as inputs
    i. a model (i.e. a state machine) $\mathbf{M = (S, \rightarrow, L)}$ and
    ii. an LTL formula ϕ
  - and outputs:
    - the set of states in **S** that satisfy ϕ.
- i.e. the signature of this procedure is:
  - **SetOfStates MC(Model M, Formula ϕ)**
- From this output, we can
  - decide $\mathbf{M,s \models \phi}$ by
  - verifying $\mathbf{s \in MC(M, \phi)}$

LTL – Model Checking – Marking Algorithm

- Given **S**, $\rightarrow$, **L**, and ϕ the marking algorithm
  1. recursively decomposes ϕ into sub-formulas
  2. marks each **s** in **S** with sub-formulas that are satisfied in **s** and
  3. propagates this information to predecessor(s) of a state as necessary.
- Note that we are working with a *finite state machine* and therefore the graph has a *finite number of vertices*.
  - But our *paths are infinite* because there are *cycles in the graph*.
    - How does information propagate (**step 3**) along cycles?

LTL – Marking Algorithm: Information Propagation

- How does information propagate along paths?
  - For instance, say, we need to verify whether **G ϕ** is satisfied in a state, say, $\mathbf{s_1}$ in this model **M**:



- Then we can evaluate $\mathbf{M,s_1 |= G\ \phi}$
- by evaluating

$\mathbf{(M,s_1 |= \phi)\ AND}$

$\mathbf{(M,s_2 |= G\ \phi)\quad AND \quad (M,s_3 |= G\ \phi) \quad AND \quad (M,s_4 |= G\ \phi)}$

LTL – Marking Algorithm: Recursive Evaluation

- Evaluation is *recursive* on the *given formula* and the *given model* :
  - i.e. the result is an *aggregate of the results* of
    - evaluating the *formula or its subformulas* on the *given state and its successors*.
- Computing whether **M,s|= ϕ** is done recursively on ϕ and **M**
  - where the latter is an aggregation of recursive evaluation on paths starting from **s**

LTL – Marking Algorithm: Recursive on ϕ and M

- Computing whether **M,s|=** ϕ is done recursively on ϕ and **M**
  - The given formula is finite:
    - i.e. the recursion on ϕ would terminate
  - But recursion on paths starting from **s** may not terminate:
    - because, a path may be cyclic.

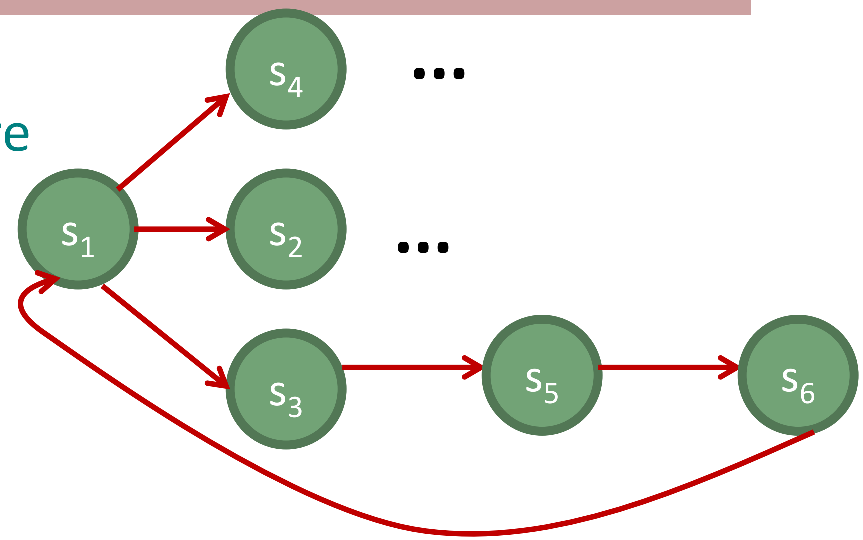
Marking : Information Propagation Along a Cycle

- For instance, given this model M,
- computing **$M,s_1 \models G\ \phi$** could require
  - computing **$M,s_3 \models G\ \phi$**

  which in turn could require
  - computing **$M,s_5 \models G\ \phi$**

  which in turn could require
  - computing **$M,s_6 \models G\ \phi$**

  which in turn could require
  - computing **$M,s_1 \models G\ \phi$**



- In general, we have the following question:
  - Given an equation of the form

    $$\mathbf{MC(M, s, \phi) = \{ \ldots\ MC(M, s, \phi)\ \ldots \}}$$

    how do you compute a solution?

Marking: Solving Recursive Equations

- Fix-point Theorem (*w/o proof*):
  - The solution to a recursive equation of the form:

    f(X) = { … f(X) … }

    is the ***(least) fixed point*** of **f** for all monotonic **f**

- A fixed point of a function **f**, is a value **X**, such that **f(X) = X**
  - i.e. computationally, we can iterate over values of **X**,
  - until we find the least value **X** such that **f(X) = X**
    - and that value would be a solution to the recursive equation.

- What is the **f** in our algorithm?

LTL – Marking Algorithm: Termination

- Thus our ***marking algorithm*** would evaluate a formula ϕ at a state **s** in model **M** by
  - (basis): evaluating sub-formulas of ϕ at **s**
  - (step): evaluating ϕ or its sub-formulas at successors of **s** and aggregating the results
- The evaluation will proceed by marking
  - i.e. updating in each state, the set of satisfied sub-formulas
    - until evaluation does not change:
      - i.e. sets of satisfied sub-formulas *remain as in previous iteration.*

LTL – Model Checking: Marking Algorithm

- **MC(St, $\rightarrow$, L, ϕ, s) // marking algorithm**
  - if ϕ is:
    - FALSE: done /* no state satisfies FALSE */
    - TRUE: done /* all states implicitly marked TRUE */
    - **p** for some atomic proposition **p**:
      - mark state **s** with **p**, if **p** is in **L(s)**

MC(St, $\rightarrow$, L, ϕ, s) // marking algorithm

- if ϕ is:
  - $\underline{\psi_1 \wedge \psi_2}$:
    - if **s** <u>is marked with ψ_1 and ψ_2</u> then mark it with $\psi_1 \wedge \psi_2$
  - $\underline{\neg\psi}$:
    - if **s** <u>is not marked with ψ</u> then mark it with $\neg\psi$

Recursive calls: **MC**

MC(St, $\rightarrow$, L, ϕ, s) // marking algorithm [contd.]

- if ϕ is:
  - **G ψ**:
    - repeat {

      mark state **s** with **G ψ** if

      s is marked with ψ **and**

      all its successors are marked with **G ψ**

      } until *no change*

Recursive calls: **MC**

  - **X ψ**: ?? /* complete this! */

MC(St, $\rightarrow$, L, ϕ, s) // marking algorithm [contd.]

- if ϕ is:
  - ψ_1 **U** ψ_2:
    - if **s** is marked with ψ_2 then mark it with ψ_1 **U** ψ_2
    - else repeat {
      - mark a state **s** with ψ_1 **U** ψ_2
        - if it is marked with ψ_1 and
        - all of its successors are marked with ψ_1 **U** ψ_2
    - } until (*no change*)

LTL Model Checking – Recursion and Fix-Point Computation

- Notes on the *marking* algorithm:
  - The *no change* condition in the two repeat loops (for the **G** case and the **U** case)
    - is essentially a test for the fix-point:
      - **MC(M,ϕ,s)** is computed iteratively on the states in **M**
      - until its value (i.e. the set of satisfied sub-formulas in **s**) *remains unchanged*!

LTL Model Checking – Time Taken

- Notes on the marking algorithm:
  - The time taken by **mark(S, $\to$, L, ϕ)** is
    - **C * |ϕ| * |S| * |$\to$|** steps
      - where **|ϕ|** denotes the number of operators in **ϕ**
      - and **|S|** denotes the number of states
      - and **|$\to$|** denotes the number of edges, which is (upper-)bounded by **|S| * |S|**
      - and **C** is a constant