UTA

Today's Agenda

- Quiz 4
- Temporal Logic

UTA

Automata and Logic

- Introduction
- Buchi Automata
- Linear Time Logic
- Summary

UTA

Buchi Automata

The SPIN model checker is based on the theory of Buchi automata (or ω-automata).

Buchi automata does not only accept finite executions but also infinite executions.

SPIN does not only formalize correctness properties as Buchi automata, but also uses it to describe the system behavior.

UTA

Temporal Logic

Temporal logic allows time-related properties to be formally specified without introducing the explicit notion of time.

SPIN uses Linear Temporal Logic (LTL), which allows to specify properties that must be satisfied by all program executions.

Question: Why don't we use Buchi automata to specify correctness properties?

The Magic

The verification of a PROMELA model in SPIN consists of the following steps:

- Build an automaton to represent the system behavior
- For each correctness property, build an automaton to represent its negation
- Compute the intersection of the system automaton and each property automaton

Automata and Logic

- Introduction
- Buchi Automata
- Linear Time Logic
- Summary

UTA

FSA

A finite state automaton is a tuple (S, s_0, L, T, F), where

- S is a finite set of states
- s_0 is a distinguished initial state, $s_0 \in S$
- L is a finite set of labels
- T is a set of transitions, $T \in (S \times L \times S)$
- F is a set of final states, T

UTA

Determinism

An FSA is deterministic, if the successor state of each transition is uniquely defined by the source state and the transition label.

Many automata we will encounter are non-deterministic, which however can be easily determinized.

UTA

Run

A run of an FSA (S, s_0, L, T, F) is an ordered, possibly infinite, set of transitions

$$\{ (s_0, l_0, s_1), (s_1, l_1, s_2), (s_2, l_2, s_3), \ldots \}$$

such that

$$\forall i,\ i \geq 0 \rightarrow (s_i, l_i, s_{i+1}) \in T$$

Note that frequently, we will only refer to the sequence of states or transitions of a run.

UTA

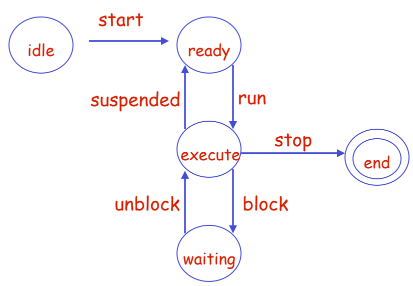
Accepting Run

A run is accepted by an FSA if and only if it terminates at a final state.

Formally, an accepting run of an FSA (S, s_0, L, T, F) is a finite run in which the final transition has the property that $s_n \in F$.

UTA

Example



{ start, run, block, unblock, stop }

UTA

Infinite Runs

Many systems have infinite runs, i.e., they do not necessarily terminate, such as a thread scheduler, a web server, or a telephone switch.

An infinite run is often called an ω-run. A classic FSA only accepts finite runs, not ω-runs.

UTA

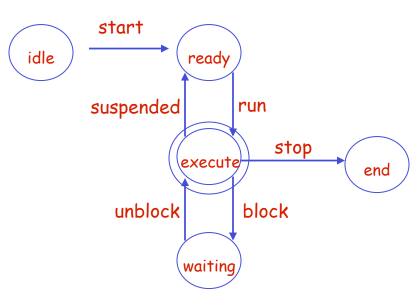
Buchi Acceptance

Intuitively, an infinite run is accepted if and only if the run visits some final state infinitely often.

Formally, an ω-run σ of FSA (S, s_0, L, T, F) is accepting if $\exists s_f, s_f \in F \wedge s_f \in \sigma^\omega$, where σ^ω is the set of states that appear infinitely often in σ.

UTA

Example



{ start, run, { suspended, run }* }

UTA

Stutter Extension

The stutter extension of finite run σ with final state s_n is the ω-run $\sigma, (s_n, \varepsilon, s_n)^\omega$, i.e., the final state persists forever by repeating the null action ε.

This extension allows Buchi acceptance to be applied to finite runs, i.e., a finite run is accepted by a Buchi automaton if and only if its final state is in the set of accepting states.

UTA

Decidability Issues

- Two properties of Büchi automata in particular are of interest and are both decidable:
  - language emptiness: are there any accepting runs?
  - language intersection: are there any runs that are accepted by 2 or more automata?
- Spin's model checking algorithm is based on these two checks
  - Spin determines if the intersection of the languages of a property automaton and a system automaton is empty

UTA

Automata and Logic

- Introduction
- Buchi Automata
- Linear Time Logic
- Summary

UTA

Temporal Logic

Temporal logic allows one to reason about temporal properties of system executions, without introducing the notion of time explicitly.

The dominant logic used in software verification is LTL, whose formulas are evaluated over a single execution.

LTL

A well-formed LTL formula is built from state formula and temporal operators:

- All state formulas are well-formed LTL formulas.
- If p and q are well-formed LTL formulas, then p U q, p *U* q, $\Box$ p, ◊ p, and X p are also well-formed LTL formulas.

Notations

- $\sigma \models f$: LTL formula f holds for ω-run σ
- σ_i : the i-th element of σ
- $\sigma[i]$: the suffix of σ that starts at the i-th element

LTL Operators (1)

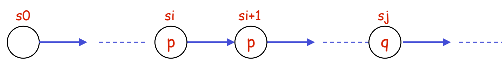
- Weak Until - U

  $\sigma[i] \models (p \; U \; q) \Leftrightarrow \sigma_i \models q \vee (\sigma_i \models p \wedge \sigma[i+1] \models (p \; U \; q))$

- Strong Until - *U*

  $\sigma[i] \models (p \; \mathcal{U} \; q) \Leftrightarrow \sigma[i] \models (p \; U \; q) \wedge \exists j, j \geq i, \sigma_j \models q$

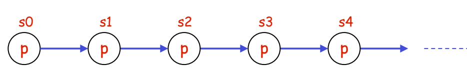
$\sigma[i] \vdash p \; \mathbf{U} \; q$



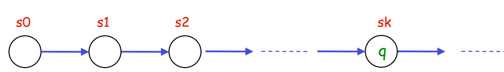
LTL Operators (2)

- always ($\Box$) : $\sigma \models \Box \; p \Leftrightarrow \sigma \models (p \; U \; \text{false})$
- eventuality ($\Diamond$) : $\sigma \models \Diamond \; q \Leftrightarrow \sigma \models (\text{true} \; \mathcal{U} \; q)$
- next (X) : $\sigma \models X \; p \Leftrightarrow \sigma_{i+1} \models p$

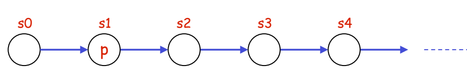
$\sigma \vdash \Box \; p$



$\sigma \vdash \Diamond \; q$



$\sigma \vdash \mathbf{X} \; p$



UTA

LTL Example (1)

Consider how to express the informal requirement that p implies q. In other words, p causes q.

[] ((p -> X (<> q)) ∧ <> p)

UTA

LTL Example (2)

Consider a traffic light. The lights keep changing in the following order: green -> yellow -> red -> green

Use a LTL formula to specify that from a state where the light is green the green color continues until it changes to yellow?

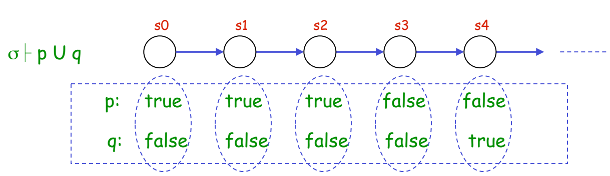
Frequently Used Formulas

- invariance : $\Box$ p
- guarantee : $\Diamond$ p
- response : p $\rightarrow \Diamond$ q
- precedence : p $\rightarrow$ q U r
- recurrence (progress) : $\Box \Diamond$ p
- stability (non-progress) : $\Diamond \Box$ p
- correlation : $\Diamond$ p $\rightarrow \Diamond$ q

Valuation Sequence

Let P be the set of all state formulas in a given LTL formula. Let V be the set of valuations, i.e., all possible truth assignments, of these formulas.

Then, we can associate each run σ with a sequence of valuations V(σ), denoting the truth assignments of all the state formulas at each state.



LTL and ω-automata (1)

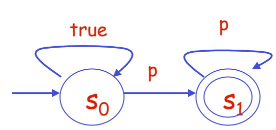
For every LTL formula, there exists an equivalent Buchi automaton, i.e., one that accepts precisely those runs that satisfy the formula.

SPIN provides a separate parser that translates an LTL formula to a never claim.

LTL and ω-automata (2)

$ spin –f '<> [] p'

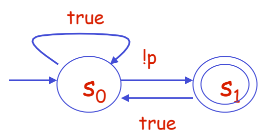
```
never {    /* <>[]p */
T0_init:
       if
       :: ((p)) -> goto accept_S4
       :: (1) -> goto T0_init
       fi;
accept_S4:
       if
       :: ((p)) -> goto accept_S4
       fi;
}
```



LTL and ω-automata (3)

$ spin -f '! <> [] p'

```
never {    /* !<>[]p */
T0_init:
        if
        :: (! ((p))) -> goto
accept_S9
        :: (1) -> goto T0_init
        fi;
accept_S9:
        if
        :: (1) -> goto T0_init
        fi;
}
```



Example (1)

```
int x = 100;
active proctype A ()
{
  do
  :: x % 2 -> x = 3 *x + 1
  od
}
active proctype B ()
{
  do
  :: ! (x % 2) -> x = x / 2
  od
}
```

UTA

Example (2)

- Prove that x can never become negative, and also never exceed its initial value.
- Prove that the value of x always eventually returns to 1.

UTA

Automata and Logic

- Introduction
- Buchi Automata
- Linear Time Logic
- Summary

UTA

Summary

- Unlike classic FSA, which only accepts finite runs, ω-automata accepts both finite and infinite runs.
- LTL can be used to specify properties that must be satisfied by all the system executions.
- An LTL formula can be translated to an equivalent ω-automata.