Temporal Logic

We want to write specifications of systems in a precise logical language. A crucial feature of our language is that it has to describe systems that change in time: hence the logic is called temporal logic. There are many variations, we will talk about Linear Temporal Logic (LTL).

Example specifications in a concurrent programming setup:

(a) MUTUAL EXCLUSION: you want to say only one process can have access to a critical resource. For example at a traffic light you want cars to go through in one direction only.

(b) LIVENESS: If you make a request it will eventually be granted.

(c) If you request access to a resource you cannot make another request until the first has been granted.

You notice some key words: always (implicit in (a)) eventually and until. Our logic will have the usual boolean operations $\wedge$, $\vee$, $\neg$, $\Rightarrow$, $\Leftrightarrow$ but ~~three~~ four new ones: $\bigcirc$, $\Diamond$, $\Box$, $\mathcal{U}$

We write p, q, etc. for basic statements.

EXAMPLES (a) Write p_1 to mean process 1 has access to a critical resource, p_2 for process 2 has access.

MUTEX says Always $(p_1 \Rightarrow \neg p_2 \wedge p_2 \Rightarrow \neg p_1)$

We use $\Box$ to denote always so we write

$$\Box(p_1 \Leftrightarrow \neg p_2)$$

(b) We write $\Diamond$ for eventually

$$\Box(\text{request} \Rightarrow \Diamond \text{granted})$$

(c) $$\Box(\text{request} \Rightarrow (\text{no new request}) \,\mathcal{U}\, (\text{request granted}))$$

The symbol $\bigcirc$ is used to mean "next" step.

We think of a system as having <u>states</u>, <u>actions</u> and <u>transitions</u>

VENDING MACHINE:

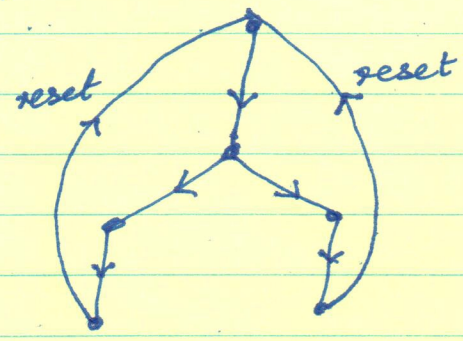
States: init, coffee_requested, tea_requested, coffee_dispensed, tea_dispensed

Actions: insert money, request coffee, request tea, reset

Transitions: init $\xrightarrow{\text{request coffee}}$ coffee_requested.

- - - - (several more) - - -

Often we use a picture



The other arrows are labelled as we have seen before

EXAMPLE FORMULA

coffee_requested $\Rightarrow \bigcirc\bigcirc$ coffee_disp.

Temporal logic on words:

Linear Temporal logic allows you to talk about sequences of actions. To talk about branching you need another type of temporal logic CTL.

LTL on words: Fix an alphabet $\{a, b\} = \Sigma$. We assume 2 predicates Q_a, Q_b. A formula like Q_a, or Q_b is a true at a position in a word. We can use temporal logic to explore the rest of the word.

word $w = abbababb$

Q_a is true at the initial position of w.

We write $w \models Q_a$ w satisfies Q_a.

More true formulas for w:

$w \models \bigcirc Q_b$ $\quad w \models \bigcirc\bigcirc Q_b$

$w \models \Box(a \Rightarrow \Diamond b)$ $\quad \Box(Q_a \Rightarrow \Diamond Q_b)$

What about $w = ababababab$?

$$w \models \Box[(Q_a \Rightarrow \bigcirc Q_b) \wedge (Q_b \Rightarrow Q_a)] \times$$

No! Because the last b does not have a following a.

$w \models \Box(Q_a \Rightarrow \bigcirc Q_b)$ ✓

We have no way of talking about previous positions. Surprisingly, this does not matter.

Can we describe regular languages using temporal logic?

How do we say we are at the end?

$\bot$: false, no position can satisfy this.

so if we write $\bigcirc\bot$ we are saying "next position does not exist".

$\Diamond\bigcirc\bot$: eventually we will reach a position with no next position; i.e. the word is finite.

$L = (ab)(ab)^* = \{ab, abab, ababab, \dots\}$

$\forall w \in L \quad w \vdash \Box[(Q_a \Rightarrow \bigcirc Q_b) \wedge (Q_b \Rightarrow (\bigcirc Q_a \vee \bigcirc\bot))] \vee$

Is this a proper characterization of L?

No!

bababab also satisfies this!

$$\{\Box[(Q_a \Rightarrow \bigcirc Q_b) \wedge (Q_b \Rightarrow (\bigcirc Q_a \vee \bigcirc\bot))]\} \wedge \underline{Q_a}$$

starts with a.

There are regular languages that cannot be defined with temporal logic: Every odd position is an "a"

. Easy with reg. exp. $(a(a+b))^*$