

# Linear Temporal Logic and Büchi Automata

(Based on [Manna and Pnueli 1992, 1995] and  
[Clarke et al. 1999])

Yih-Kuen Tsay

Department of Information Management  
National Taiwan University

# Outline

Introduction

Propositional Temporal Logic: The Future

Simple On-the-Fly Translation

Propositional Temporal Logic: The Past

PTL to Automata: A Tableau Construction

Quantified Propositional Temporal Logic (QPTL)

Equivalences and Congruences

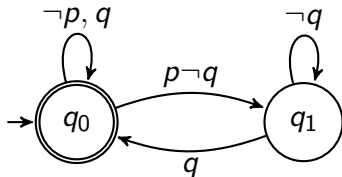
Concluding Remarks

References

# Introduction

- 🌐 We have seen how automata, in particular **Büchi automata**, may be used to describe the behaviors of a concurrent system.
- 🌐 Büchi automata “localize” temporal dependency between occurrences of events (represented by propositions) to relations between states and tend to be of lower level.
- 🌐 We will study an alternative formalism, namely **linear temporal logic**.
- 🌐 Temporal logic formulae describe temporal dependency without explicit references to time points and are in general more abstract.

# Introduction (cont.)



- The above Büchi automaton says that, whenever  $p$  holds at some point in time,  $q$  must hold at the same time or will hold at a later time.

Note: the alphabet is  $\{pq, p\bar{q}, \bar{p}q, \bar{p}\bar{q}\}$ ;  $q$  alone represents any input symbol from  $\{pq, \bar{p}q\}$ .

- It may not be easy to see that this indeed is the case.
- In linear temporal logic, this can easily be expressed as  $\Box(p \rightarrow \Diamond q)$ , which reads “always  $p$  implies eventually  $q$ ”.

# PTL: The Future

- 🌐 We first look at the future fragment of **Propositional Temporal Logic (PTL)**.
- 🌐 Future operators include  $\bigcirc$  (**next**),  $\diamond$  (**eventually**),  $\square$  (**always**),  $\mathcal{U}$  (**until**), and  $\mathcal{W}$  (**wait-for**).
- 🌐 With  $\mathcal{W}$  replaced by  $\mathcal{R}$  (**release**), this fragment is often referred to as **LTL** (linear temporal logic) in the model checking community.
- 🌐 Let  $V$  be a set of boolean variables.
- 🌐 The future PTL formulae are defined inductively as follows:
  - ☀ Every variable  $p \in V$  is a PTL formula.
  - ☀ If  $f$  and  $g$  are PTL formulae, then so are  $\neg f$ ,  $f \vee g$ ,  $f \wedge g$ ,  $\bigcirc f$ ,  $\diamond f$ ,  $\square f$ ,  $f \mathcal{U} g$ , and  $f \mathcal{W} g$ .  
( $\neg f \vee g$  is also written as  $f \rightarrow g$  and  $(f \rightarrow g) \wedge (g \rightarrow f)$  as  $f \leftrightarrow g$ .)
- 🌐 Examples:  $\square(\neg C_0 \vee \neg C_1)$ ,  $\square(T_1 \rightarrow \diamond C_1)$ .

# PTL: The Future (cont.)

- 🌐 A PTL formula is interpreted over an infinite sequence of states  $\sigma = s_0s_1s_2 \cdots$ , relative to a position in that sequence.
- 🌐 A **state** is a subset of  $V$ , containing exactly those variables that evaluate to true in that state.
- 🌐 If each possible subset of  $V$  is treated as a symbol, then a sequence of states can also be viewed as an infinite word over  $2^V$ .
- 🌐 The semantics of PTL in terms of  $(\sigma, i) \models f$  ( $f$  holds at the  $i$ -th position of  $\sigma$ ) is given below.
- 🌐 We say that a sequence  $\sigma$  satisfies a PTL formula  $f$  or  $\sigma$  is a model of  $f$ , denoted  $\sigma \models f$ , if  $(\sigma, 0) \models f$ .

# PTL: The Future (cont.)

🌐 For a boolean variable  $p$ ,

☀️  $(\sigma, i) \models p \iff p \in s_i$

🌐 For boolean operators,

☀️  $(\sigma, i) \models \neg f \iff (\sigma, i) \models f$  does not hold

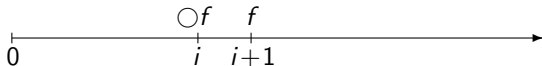
☀️  $(\sigma, i) \models f \vee g \iff (\sigma, i) \models f$  or  $(\sigma, i) \models g$

☀️  $(\sigma, i) \models f \wedge g \iff (\sigma, i) \models f$  and  $(\sigma, i) \models g$

# PTL: The Future (cont.)

For future temporal operators,

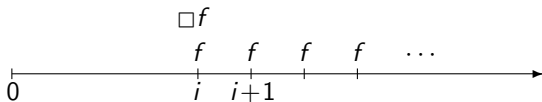
$$\odot (\sigma, i) \models \circ f \iff (\sigma, i+1) \models f$$



$$\odot (\sigma, i) \models \diamond f \iff \text{for some } j \geq i, (\sigma, j) \models f$$



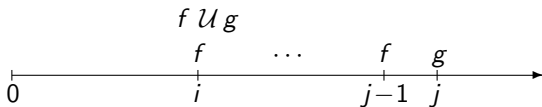
$$\odot (\sigma, i) \models \square f \iff \text{for all } j \geq i, (\sigma, j) \models f$$





# PTL: The Future (cont.)

- 🌐 For future temporal operators (cont.),
- ☀️  $(\sigma, i) \models f \mathcal{U} g \iff$  for some  $j \geq i$ ,  $(\sigma, j) \models g$  and for all  $k$ ,  $i \leq k < j$ ,  $(\sigma, k) \models f$



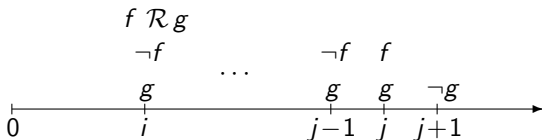
- ☀️  $(\sigma, i) \models f \mathcal{W} g \iff$  (for some  $j \geq i$ ,  $(\sigma, j) \models g$  and for all  $k$ ,  $i \leq k < j$ ,  $(\sigma, k) \models f$ ) or (for all  $k \geq i$ ,  $(\sigma, k) \models f$ )

$f \mathcal{W} g$  holds at position  $i$  if and only if  $f \mathcal{U} g$  or  $\Box f$  holds at position  $i$ .

# PTL: The Future (cont.)

🌐 For future temporal operators (cont.),

- ☀ When  $\mathcal{R}$  is preferred over  $\mathcal{W}$ ,  
 $(\sigma, i) \models f \mathcal{R} g \iff$  for all  $j \geq i$ , if  $(\sigma, k) \not\models f$  for all  $k, i \leq k < j$ ,  
 then  $(\sigma, j) \models g$ .



# Simple On-the-Fly Translation

- 🌐 We will study a tableau-based algorithm [GPVW] for obtaining a Büchi automaton from a PTL formula.
- 🌐 The algorithm is geared towards being used in model checking in an on-the-fly fashion:  
*It is possible to detect that a property does not hold by only constructing part of the model and of the automaton.*
- 🌐 The algorithm can also be used to check the validity of a temporal logic assertion.
- 🌐 To apply the translation algorithm, we first convert the formula  $\varphi$  into the *negation normal form*.

# Preprocessing of Formulae

Every LTL formula can be converted into the negation normal form:

$$\text{⊕} \neg(p \wedge q) = (\neg p) \vee (\neg q)$$

$$\text{⊕} \neg(p \vee q) = (\neg p) \wedge (\neg q)$$

$$\text{⊕} \diamond p \text{ (or } \mathbf{F}p) = \textit{True} \mathcal{U} p$$

$$\text{⊕} \square p \text{ (or } \mathbf{G}p) = \textit{False} \mathcal{R} p$$






$$\text{⊕} \neg(p \mathcal{U} q) = (\neg p) \mathcal{R} (\neg q)$$

$$\text{⊕} \neg(p \mathcal{R} q) = (\neg p) \mathcal{U} (\neg q)$$

$$\text{⊕} \neg \bigcirc p \text{ (or } \neg \mathbf{X}p) = \bigcirc \neg p$$

Note: “ $p \mathcal{W} q$ ” was not treated in the original on-the-fly translation algorithm;  $\neg(p \mathcal{W} q) \cong (\neg q) \mathcal{U} (\neg p \wedge \neg q)$ .

# Data Structure of an Automaton Node

-  *ID*: a string that identifies the node.
-  *Incoming*: the incoming edges, represented by the IDs of the nodes with an outgoing edge leading to this node.
-  *New*: a set of subformulae that must hold at this state and have not yet been processed.
-  *Old*: the subformulae that must hold at this state and have already been processed.
-  *Next*: the subformulae that must hold in all states that are immediate successors of states satisfying the formulae in *Old*.

# The Algorithm: Start and Overview

- Start with a single node having a single incoming edge labeled *init* (i.e., from an initial node).
- The starting node has initially one obligation in *New*, namely  $\varphi$ , and *Old* and *Next* are initially empty.
- Expand the starting node (which generates new nodes) in an *DFS* manner.
- Fully processed nodes are put in a list called *Nodes*.

```
function create_graph( $\varphi$ )  
  expand([ID  $\leftarrow$  new_ID(),  
        Incoming  $\leftarrow$  {init},  
        Old  $\leftarrow$   $\emptyset$ ,  
        New  $\leftarrow$  { $\varphi$ },  
        Next  $\leftarrow$   $\emptyset$ ],  
         $\emptyset$ );
```

**end function**

# The Algorithm: Node-Expansion

- 🌐 Check if there are unprocessed obligations in *New* of the current node *N*.
- 🌐 If *New* is empty, it means node *N* is fully processed and ready to be added to *Nodes*.
- 🌐 Otherwise, a formula in *New* is selected, processed, and moved to *Old*.

```
function expand(q, Nodes)  
  if New(q) =  $\emptyset$  then  
    if  $\exists r \in \text{Nodes} : \text{Old}(r) = \text{Old}(q) \wedge \text{Next}(r) = \text{Next}(q)$  then  
      ...  
    else ...  
  else let  $\eta \in \text{New}(q)$ ;  
     $\text{New}(q) := \text{New}(q) - \eta$ ;  
    ...  
end function
```

# The Algorithm: Node-Expansion (cont.)

```

/* in function expand */
if New(q) =  $\emptyset$  then
  if  $\exists r \in Nodes : Old(r) = Old(q) \wedge Next(r) = Next(q)$  then
    Incoming(r) := Incoming(r)  $\cup$  Incoming(q);
    return(Nodes);
  else expand([ID  $\leftarrow$  new_ID(),
             Incoming  $\leftarrow$  {ID(q)},
             Old  $\leftarrow$   $\emptyset$ ,
             New  $\leftarrow$  Next(q),
             Next  $\leftarrow$   $\emptyset$ ], Nodes  $\cup$  {q});
  end if
else let  $\eta \in New(q)$ ;
     New(q) := New(q) -  $\eta$ ;
     if  $\eta \in Old(q)$  then expand(q, Nodes);
     else ... /* cases according to the form of  $\eta$  */

```



A fully processed current node  $N$  is added to  $Nodes$  as follows:

- 🌐 If there already is a node in  $Nodes$  with the same obligations in both its *Old* and *Next* fields, the incoming edges of  $N$  are incorporated into those of the existing node.
- 🌐 Otherwise, the current node  $N$  is added to  $Nodes$ .
- 🌐 With the addition of node  $N$  in  $Nodes$ , a new current node is formed for its successor as follows:
  1. There is initially one edge from  $N$  to the new node.
  2. *New* is set initially to the *Next* field of  $N$ .
  3. *Old* and *Next* of the new node are initially empty.

# The Algorithm: Node-Expansion (cont.)

A formula  $\eta$  in *New* is processed as follows:

- 🌐 If  $\eta$  is just a literal (a proposition or the negation of a proposition), then
  - ☀ if  $\neg\eta$  is in *Old*, the current node is discarded;
  - ☀ otherwise,  $\eta$  is added to *Old*.
- 🌐 If  $\eta$  is not a literal, the current node can be split into two or not split, and new formulae can be added to the fields *New* and *Next*.
- 🌐 The exact actions depend on the form of  $\eta$ .

# The Algorithm: Node-Expansion (cont.)

**case  $\eta$  of**

$p \wedge q: q' := [ID \leftarrow new\_ID(),$   
 $Incoming \leftarrow Incoming(q),$   
 $Old \leftarrow Old(q) \cup \{\eta\},$   
 $New \leftarrow New(q) \cup \{p, q\},$   
 $Next \leftarrow Next(q)];$   
 $expand(q', Nodes);$

$p \vee q: \dots$

$p \mathcal{U} q: \dots$






$p \mathcal{R} q: \dots$

$\bigcirc p: \dots$

**end case**

# The Algorithm: Node-Expansion (cont.)

Actions on  $\eta$  (that is not a literal):

-   $\eta = p \wedge q$ , then both  $p$  and  $q$  are added to *New*.
-   $\eta = p \vee q$ , then the node is split, adding  $p$  to *New* of one copy, and  $q$  to the other.
-   $\eta = p \mathcal{U} q (\cong q \vee (p \wedge \circ(p \mathcal{U} q)))$ , then the node is split. For the first copy,  $p$  is added to *New* and  $p \mathcal{U} q$  to *Next*. For the other copy,  $q$  is added to *New*.
-   $\eta = p \mathcal{R} q (\cong (p \wedge q) \vee (q \wedge \circ(p \mathcal{R} q)))$ , similar to  $\mathcal{U}$ .
-   $\eta = \circ p$ , then  $p$  is added to *Next*.

Note:  $p \mathcal{W} q \cong q \vee (p \wedge \circ(p \mathcal{W} q))$

# The Algorithm: Handling $\mathcal{U}$

case  $\eta$  of

...

```
 $p \mathcal{U} q$ :  $q_1 := [ID \leftarrow new\_ID(),$   
           $Incoming \leftarrow Incoming(q),$   
           $Old \leftarrow Old(q) \cup \{\eta\},$   
           $New \leftarrow New(q) \cup \{p\},$   
           $Next \leftarrow Next(q) \cup \{p \mathcal{U} q\}];$   
 $q_2 := [ID \leftarrow new\_ID(),$   
           $Incoming \leftarrow Incoming(q),$   
           $Old \leftarrow Old(q) \cup \{\eta\},$   
           $New \leftarrow New(q) \cup \{q\},$   
           $Next \leftarrow Next(q)];$   
 $expand(q_2, expand(q_1, Nodes));$ 
```

...

end case

# The Algorithm: Handling $\mathcal{R}$

case  $\eta$  of

...

```
 $p \mathcal{R} q$ :  $q_1 := [ID \leftarrow new\_ID(),$   
           $Incoming \leftarrow Incoming(q),$   
           $Old \leftarrow Old(q) \cup \{\eta\},$   
           $New \leftarrow New(q) \cup \{q\},$   
           $Next \leftarrow Next(q) \cup \{p \mathcal{R} q\}];$   
 $q_2 := [ID \leftarrow new\_ID(),$   
           $Incoming \leftarrow Incoming(q),$   
           $Old \leftarrow Old(q) \cup \{\eta\},$   
           $New \leftarrow New(q) \cup \{p, q\},$   
           $Next \leftarrow Next(q)];$   
 $expand(q_2, expand(q_1, Nodes));$ 
```

...

end case

# Nodes to GBA

The list of nodes in  $Nodes$  can now be converted into a **generalized Büchi automaton**  $B = (\Sigma, Q, q_0, \Delta, F)$ :

1.  $\Sigma$  consists of sets of propositions from  $AP$ .
2. The set of states  $Q$  includes the nodes in  $Nodes$  and the additional initial state  $q_0$ .
3.  $(r, \alpha, r') \in \Delta$  iff  $r \in Incoming(r')$  and  $\alpha$  satisfies the conjunction of the negated and nonnegated propositions in  $Old(r')$
4.  $q_0$  is the initial state, playing the role of *init*.
5.  $F$  contains a separate set  $F_i$  of states for each subformula of the form  $p \mathcal{U} q$ ;  $F_i$  contains all the states  $r$  such that either  $q \in Old(r)$  or  $p \mathcal{U} q \notin Old(r)$ .

# PTL: The Past

- 🌐 We now add the past fragment.
- 🌐 Past operators include  $\ominus$  (**before**),  $\ominus$  (**previous**),  $\diamond$  (**once**),  $\square$  (**so-far**),  $\mathcal{S}$  (**since**), and  $\mathcal{B}$  (**back-to**).
- 🌐 The full PTL formulae are defined inductively as follows:
  - ☀ Every variable  $p \in V$  is a PTL formula.
  - ☀ If  $f$  and  $g$  are PTL formulae, then so are  $\neg f$ ,  $f \vee g$ ,  $f \wedge g$ ,  $\bigcirc f$ ,  $\diamond f$ ,  $\square f$ ,  $f \mathcal{U} g$ ,  $f \mathcal{W} g$ ,  $\ominus f$ ,  $\ominus f$ ,  $\diamond f$ ,  $\square f$ ,  $f \mathcal{S} g$ , and  $f \mathcal{B} g$ .  
 ( $\neg f \vee g$  is also written as  $f \rightarrow g$  and  $(f \rightarrow g) \wedge (g \rightarrow f)$  as  $f \leftrightarrow g$ .)
- 🌐 Examples:
  - ☀  $\square(p \rightarrow \diamond q)$  says “every  $p$  is preceded by a  $q$ .”
  - ☀  $\square(\diamond \neg p \rightarrow \diamond q)$  is another way of saying  $p \mathcal{W} q$ !



# PTL: The Past (cont.)

For past temporal operators,

$$\odot f \models (\sigma, i) \iff i = 0 \text{ or } (\sigma, i - 1) \models f$$

$$\ominus f \models (\sigma, i) \iff i > 0 \text{ and } (\sigma, i - 1) \models f$$

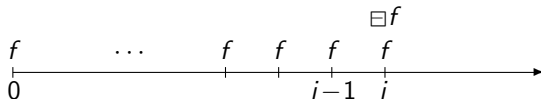


The difference between  $\odot f$  and  $\ominus f$  occurs at position 0.

$$\diamond f \models (\sigma, i) \iff \text{for some } j, 0 \leq j \leq i, (\sigma, j) \models f$$



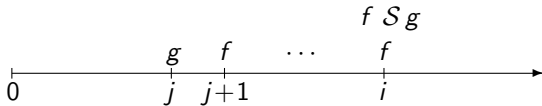
$$\boxplus f \models (\sigma, i) \iff \text{for all } j, 0 \leq j \leq i, (\sigma, j) \models f$$



# PTL: The Past (cont.)

🌐 For past temporal operators (cont.),

☀️  $(\sigma, i) \models f \mathcal{S} g \iff$  for some  $j$ ,  $0 \leq j \leq i$ ,  $(\sigma, j) \models g$  and for all  $k$ ,  $j < k \leq i$ ,  $(\sigma, k) \models f$









☀️  $(\sigma, i) \models f \mathcal{B} g \iff$  (for some  $j$ ,  $0 \leq j \leq i$ ,  $(\sigma, j) \models g$  and for all  $k$ ,  $j < k \leq i$ ,  $(\sigma, k) \models f$ ) or (for all  $k$ ,  $0 \leq k \leq i$ ,  $(\sigma, k) \models f$ )

$f \mathcal{B} g$  holds at position  $i$  if and only if  $f \mathcal{S} g$  or  $\exists f$  holds at position  $i$ .

# A Hierarchy of Temporal Properties

## Classes of temporal properties:

-  Safety properties:  $\Box p$
-  Guarantee properties:  $\Diamond p$
-  Obligation properties:  $\bigwedge_{i=1}^n (\Box p_i \vee \Diamond q_i)$
-  Response properties:  $\Box \Diamond p$
-  Persistence properties:  $\Diamond \Box p$
-  Reactivity properties:  $\bigwedge_{i=1}^n (\Box \Diamond p_i \vee \Diamond \Box q_i)$

Here  $p, q, p_i, q_i$  are arbitrary past temporal formulae.

## The hierarchy

$$\begin{array}{ccccccc} \text{Safety} & & & & \text{Response} & & \\ \text{Guarantee} & \subseteq & \text{Obligation} & \subseteq & \text{Persistence} & \subseteq & \text{Reactivity} \end{array}$$

## Every temporal formula is equivalent to some reactivity formula.

# More Common Temporal Properties

🌐 Safety properties:  $\Box p$

Example:  $p \mathcal{W} q$  is a safety property, as it is equivalent to  $\Box(\Diamond \neg p \rightarrow \Diamond q)$ .

🌐 Response properties

☀ Canonical form:  $\Box \Diamond p$

☀ Variant:  $\Box(p \rightarrow \Diamond q)$  ( $p$  leads-to  $q$ ), which is equivalent to  $\Box \Diamond(\neg p \mathcal{B} q)$ .

🌐 Reactivity properties:  $\bigwedge_{i=1}^n (\Box \Diamond p_i \vee \Diamond \Box q_i)$

🌐 (Simple) reactivity properties

☀ Canonical form:  $\Box \Diamond p \vee \Diamond \Box q$

☀ Variants:  $\Box \Diamond p \rightarrow \Box \Diamond q$  or  $\Box(\Box \Diamond p \rightarrow \Diamond q)$ , which is equivalent to  $\Box \Diamond q \vee \Diamond \Box \neg p$ .

☀ Extended form:  $\Box((p \wedge \Box \Diamond r) \rightarrow \Diamond q)$

- 🌐 We next study the Tableau Construction as described in [Manna and Pnueli 1995], which handles both future and past temporal operators.
- 🌐 More efficient constructions exist, but this construction is relatively easy to understand.
- 🌐 A **tableau** is a graphical representation of all models/sequences that satisfy the given temporal logic formula.
- 🌐 The construction results in essentially a GBA, but leaving propositions on the states (rather than moving them to the incoming edges of a state).
- 🌐 Our presentation will be slightly different, to make the resulting GBA more apparent.









# Expansion Formulae

- The requirement that a temporal formula holds at a position  $j$  of a model can often be decomposed into requirements that
  - a simpler formula holds at the same position and
  - some other formula holds either at  $j + 1$  or  $j - 1$ .
- For this decomposition, we have the following expansion formulae:

$$\begin{array}{ll}
 \Box p \cong p \wedge \bigcirc \Box p & \Box p \cong p \wedge \ominus \Box p \\
 \Diamond p \cong p \vee \bigcirc \Diamond p & \Diamond p \cong p \vee \ominus \Diamond p \\
 p \mathcal{U} q \cong q \vee (p \wedge \bigcirc (p \mathcal{U} q)) & p \mathcal{S} q \cong q \vee (p \wedge \ominus (p \mathcal{S} q)) \\
 p \mathcal{W} q \cong q \vee (p \wedge \bigcirc (p \mathcal{W} q)) & p \mathcal{B} q \cong q \vee (p \wedge \ominus (p \mathcal{B} q))
 \end{array}$$

Note: this construction does not deal with  $\mathcal{R}$ .

# Closure

-  We define the **closure** of a formula  $\varphi$ , denoted by  $\Phi_\varphi$ , as the smallest set of formulae satisfying the following requirements:
-   $\varphi \in \Phi_\varphi$ .
  -  For every  $p \in \Phi_\varphi$ , if  $q$  a subformula of  $p$  then  $q \in \Phi_\varphi$ .
  -  For every  $p \in \Phi_\varphi$ ,  $\neg p \in \Phi_\varphi$ .
  -  For every  $\psi \in \{\Box p, \Diamond p, p \mathcal{U} q, p \mathcal{W} q\}$ , if  $\psi \in \Phi_\varphi$  then  $\bigcirc \psi \in \Phi_\varphi$ .
  -  For every  $\psi \in \{\Diamond p, p \mathcal{S} q\}$ , if  $\psi \in \Phi_\varphi$  then  $\ominus \psi \in \Phi_\varphi$ .
  -  For every  $\psi \in \{\Box p, p \mathcal{B} q\}$ , if  $\psi \in \Phi_\varphi$  then  $\ominus \psi \in \Phi_\varphi$ .
-  So, the closure  $\Phi_\varphi$  of a formula  $\varphi$  includes all formulae that are relevant to the truth of  $\varphi$ .

# Classification of Formulae

$\alpha$	$K(\alpha)$
$p \wedge q$	$p, q$
$\Box p$	$p, \Box p$
$\Box p$	$p, \neg \Box p$

$\beta$	$K_1(\beta)$	$K_2(\beta)$
$p \vee q$	$p$	$q$
$\Diamond p$	$p$	$\Box \Diamond p$
$\Diamond p$	$p$	$\neg \Box \Diamond p$
$p \mathcal{U} q$	$q$	$p, \Box(p \mathcal{U} q)$
$p \mathcal{W} q$	$q$	$p, \Box(p \mathcal{W} q)$
$p \mathcal{S} q$	$q$	$p, \neg \Box(p \mathcal{S} q)$
$p \mathcal{B} q$	$q$	$p, \neg \Box(p \mathcal{B} q)$

- An  $\alpha$ -formula  $\varphi$  holds at position  $j$  iff all the  $K(\varphi)$ -formulae hold at  $j$ .
- A  $\beta$ -formula  $\psi$  holds at position  $j$  iff either  $K_1(\psi)$  or all the  $K_2(\psi)$ -formulae (or both) hold at  $j$ .



- 🌐 We define an **atom** over  $\varphi$  to be a subset  $A \subseteq \Phi_\varphi$  satisfying the following requirements:
- ☀️  $R_{sat}$  : the conjunction of all **state formulae** in  $A$  is satisfiable.
  - ☀️  $R_{\neg}$  : for every  $p \in \Phi_\varphi$ ,  $p \in A$  iff  $\neg p \notin A$ .
  - ☀️  $R_\alpha$  : for every  $\alpha$ -formula  $p \in \Phi_\varphi$ ,  $p \in A$  iff  $K(p) \subseteq A$ .
  - ☀️  $R_\beta$  : for every  $\beta$ -formula  $p \in \Phi_\varphi$ ,  $p \in A$  iff either  $K_1(p) \in A$  or  $K_2(p) \subseteq A$  (or both).
- 🌐 For example, if atom  $A$  contains the formula  $\neg \diamond p$ , it must also contain the formulae  $\neg p$  and  $\neg \bigcirc \diamond p$ .

# Mutually Satisfiable Formulae

- A set of formulae  $S \subseteq \Phi_\varphi$  is called **mutually satisfiable** if there exists a model  $\sigma$  and a position  $j \geq 0$ , such that every formula  $p \in S$  holds at position  $j$  of  $\sigma$ .
- The intended meaning of an **atom** is that it represents a **maximal** mutually satisfiable set of formulae.

## Claim (atoms represent necessary conditions)

*Let  $S \subseteq \Phi_\varphi$  be a mutually satisfiable set of formulae. Then there exists a  $\varphi$ -atom  $A$  such that  $S \subseteq A$ .*

- It is important to realize that inclusion in an atom is only a **necessary condition** for mutual satisfiability (e.g.,  $\{\circ p \vee \circ \neg p, \circ p, \circ \neg p, p\}$  is an atom for the formula  $\circ p \vee \circ \neg p$ ).

# Basic Formulae

- 🌐 A formula is called **basic** if it is either a proposition or has the form  $\bigcirc p$ ,  $\ominus p$ , or  $\odot p$ .
- 🌐 Basic formulae are important because their presence or absence in an atom uniquely determines all other closure formulae in the same atom.
- 🌐 Let  $\Phi_\varphi^+$  denote the set of formulae in  $\Phi_\varphi$  that are not of the form  $\neg\psi$ .

## Algorithm (atom construction)

1. Find all basic formulae  $p_1, \dots, p_b \in \Phi_\varphi^+$ .
2. Construct all  $2^b$  combinations.
3. Complete each combination into a full atom.

# Example

Consider the formula  $\varphi_1 : \Box p \wedge \Diamond \neg p$  whose basic formulae are

$$p, \bigcirc \Box p, \bigcirc \Diamond \neg p.$$

Following is the list of all atoms of  $\varphi_1$ :

$$\begin{array}{l}
 A_0 : \{ \neg p, \neg \bigcirc \Box p, \neg \bigcirc \Diamond \neg p, \neg \Box p, \Diamond \neg p, \neg \varphi_1 \} \\
 A_1 : \{ p, \neg \bigcirc \Box p, \neg \bigcirc \Diamond \neg p, \neg \Box p, \neg \Diamond \neg p, \neg \varphi_1 \} \\
 A_2 : \{ \neg p, \neg \bigcirc \Box p, \bigcirc \Diamond \neg p, \neg \Box p, \Diamond \neg p, \neg \varphi_1 \} \\
 A_3 : \{ p, \neg \bigcirc \Box p, \bigcirc \Diamond \neg p, \neg \Box p, \Diamond \neg p, \neg \varphi_1 \} \\
 A_4 : \{ \neg p, \bigcirc \Box p, \neg \bigcirc \Diamond \neg p, \neg \Box p, \Diamond \neg p, \neg \varphi_1 \} \\
 A_5 : \{ p, \bigcirc \Box p, \neg \bigcirc \Diamond \neg p, \Box p, \neg \Diamond \neg p, \neg \varphi_1 \} \\
 A_6 : \{ \neg p, \bigcirc \Box p, \bigcirc \Diamond \neg p, \neg \Box p, \Diamond \neg p, \neg \varphi_1 \} \\
 A_7 : \{ p, \bigcirc \Box p, \bigcirc \Diamond \neg p, \Box p, \Diamond \neg p, \varphi_1 \}
 \end{array}$$

# The Tableau

- 🌐 Given a formula  $\varphi$ , we construct a directed graph  $T_\varphi$ , called the **tableau** of  $\varphi$ , by the following algorithm.

## Algorithm (tableau construction)

1. The nodes of  $T_\varphi$  are the atoms of  $\varphi$ .
2. Atom  $A$  is connected to atom  $B$  by a directed edge if all of the following are satisfied:
  - 👉  $R_\circ$  : For every  $\circ p \in \Phi_\varphi$ ,  $\circ p \in A$  iff  $p \in B$ .
  - 👉  $R_\ominus$  : For every  $\ominus p \in \Phi_\varphi$ ,  $p \in A$  iff  $\ominus p \in B$ .
  - 👉  $R_\odot$  : For every  $\odot p \in \Phi_\varphi$ ,  $p \in A$  iff  $\odot p \in B$ .

- 🌐 An atom is called **initial** if it does not contain a formula of the form  $\ominus p$  or  $\neg \odot p$  ( $\cong \ominus \neg p$ ).

# Example

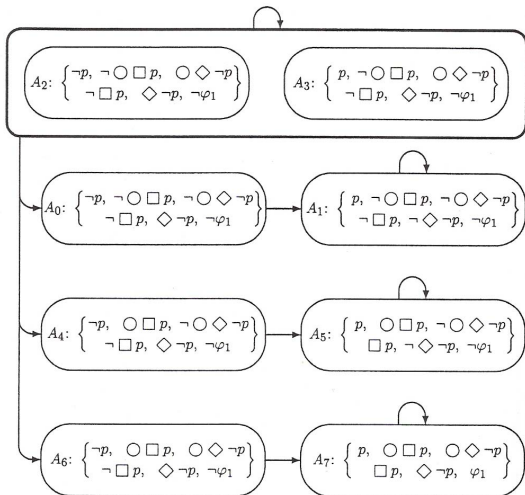











Figure: Tableau  $T_{\varphi_1}$  for  $\varphi_1 = \Box p \wedge \Diamond \neg p$ . Source: [Manna and Pnueli 1995].

# From the Tableau to a GBA

-  Create an initial node and link it to every initial atom that contains  $\varphi$ .
-  Label each directed edge with the atomic propositions that are contained in the ending atom.
-  Add a set of atoms to the accepting set for each subformula of the following form:
  -   $\diamond q$ : atoms with  $q$  or  $\neg \diamond q$ .
  -   $p \mathcal{U} q$ : atoms with  $q$  or  $\neg(p \mathcal{U} q)$ .
  -   $\neg \square \neg q$  ( $\cong \diamond q$ ): atoms with  $q$  or  $\square \neg q$ .
  -   $\neg(\neg q \mathcal{W} p)$  ( $\cong \neg p \mathcal{U}(q \wedge \neg p)$ ): atoms with  $q$  or  $\neg q \mathcal{W} p$ .
  -   $\neg \square q$  ( $\cong \diamond \neg q$ ): atoms with  $\neg q$  or  $\square q$ .
  -   $\neg(q \mathcal{W} p)$  ( $\cong \neg p \mathcal{U}(\neg q \wedge \neg p)$ ): atoms with  $\neg q$  or  $q \mathcal{W} p$ .

# Correctness: Models vs. Paths

- 🌐 For a model  $\sigma$ , the infinite atom path  $\pi_\sigma : A_0, A_1, \dots$  in  $T_\varphi$  is said to be **induced** by  $\sigma$  if, for every position  $j \geq 0$  and every closure formula  $p \in \Phi_\varphi$ ,

$$(\sigma, j) \models p \text{ iff } p \in A_j.$$

## Claim (models induce paths)

*Consider a formula  $\varphi$  and its tableau  $T_\varphi$ . For every model  $\sigma : s_0, s_1, \dots$ , there exists an infinite atom path  $\pi_\sigma : A_0, A_1, \dots$  in  $T_\varphi$  **induced** by  $\sigma$ .*

*Furthermore,  $A_0$  is an initial atom, and if  $\sigma \models \varphi$  then  $\varphi \in A_0$ .*



# Correctness: Promising Formulae

-  A formula  $\psi \in \Phi_\varphi$  is said to **promise** the formula  $r$  if  $\psi$  has one of the following forms:

$$\diamond r, p \mathcal{U} r, \neg \square \neg r, \neg(\neg r \mathcal{W} p).$$

or if  $r$  is the negation  $\neg q$  and  $\psi$  has one of the forms:

$$\neg \square q, \neg(q \mathcal{W} p).$$

## Claim (promise fulfillment by models)

*Let  $\sigma$  be a model and  $\psi$ , a formula promising  $r$ . Then,  $\sigma$  contains infinitely many positions  $j \geq 0$  such that*

$$(\sigma, j) \models \neg \psi \text{ or } (\sigma, j) \models r.$$

# Correctness: Fulfilling Paths

- Atom  $A$  **fulfills** a formula  $\psi$  that promises  $r$  if  $\neg\psi \in A$  or  $r \in A$ .
- A path  $\pi : A_0, A_1, \dots$  in the tableau  $T_\varphi$  is called **fulfilling**:
  - $A_0$  is an initial atom.
  - For every promising formula  $\psi \in \Phi_\varphi$ ,  $\pi$  contains infinitely many atoms  $A_j$  that fulfill  $\psi$ .

## Claim (models induce fulfilling paths)

*If  $\pi_\sigma : A_0, A_1, \dots$  is a path induced by a model  $\sigma$ , then  $\pi_\sigma$  is fulfilling.*

## Correctness: Fulfilling Paths (cont.)








### Claim (fulfilling paths induce models)

If  $\pi : A_0, A_1, \dots$  is a fulfilling path in  $T_\varphi$ , there exists a model  $\sigma$  inducing  $\pi$ , i.e.,  $\pi = \pi_\sigma$  and, for every  $\psi \in \Phi_\varphi$  and every  $j \geq 0$ ,

$$(\sigma, j) \models \psi \text{ iff } \psi \in A_j.$$

### Proposition (satisfiability and fulfilling paths)

Formula  $\varphi$  is satisfiable iff the tableau  $T_\varphi$  contains a fulfilling path  $\pi = A_0, A_1, \dots$  such that  $A_0$  is an initial  $\varphi$ -atom.

-  Quantified Propositional Temporal Logic (QPTL) is PTL extended with quantification over boolean variables (so, every PTL formula is also a QPTL formula):
  -  If  $f$  is a QPTL formula and  $x \in V$ , then  $\forall x: f$  and  $\exists x: f$  are QPTL formulae.
-  Let  $\sigma = s_0 s_1 \dots$  and  $\sigma' = s'_0 s'_1 \dots$  be two sequences of states.
-  We say that  $\sigma'$  is a **x-variant** of  $\sigma$  if, for every  $i \geq 0$ ,  $s'_i$  differs from  $s_i$  at most in the valuation of  $x$ , i.e., the symmetric set difference of  $s'_i$  and  $s_i$  is either  $\{x\}$  or empty.
-  The semantics of QPTL is defined by extending that of PTL with additional semantic definitions for the quantifiers:
  -   $(\sigma, i) \models \exists x: f \iff (\sigma', i) \models f$  for some  $x$ -variant  $\sigma'$  of  $\sigma$
  -   $(\sigma, i) \models \forall x: f \iff (\sigma', i) \models f$  for all  $x$ -variant  $\sigma'$  of  $\sigma$

# Expressiveness

## Theorem

*PTL is strictly less expressive than Büchi automata.*

## Proof.

1. Every PTL formula can be translated into an equivalent Büchi automaton.
2. “ $p$  holds at every even position” is recognizable by a Büchi automaton, but cannot be expressed in PTL.




## Theorem

*QPTL is expressively equivalent to Büchi automata.*

# Equivalences and Congruences

- 🌐 A formula  $p$  is **valid**, denoted  $\models p$ , if  $\sigma \models p$  for every  $\sigma$ .
- 🌐 Two formulae  $p$  and  $q$  are **equivalent** if  $\models p \leftrightarrow q$ , i.e.,  $\sigma \models p$  if and only if  $\sigma \models q$  for every  $\sigma$ .
- 🌐 Two formulae  $p$  and  $q$  are **congruent**, denoted  $p \cong q$ , if  $\models \Box(p \leftrightarrow q)$ .
- 🌐 Congruence is a stronger relation than equivalence:
  - ☀️  $p \vee \neg p$  and  $\neg \ominus(p \vee \neg p)$  are equivalent, as they are both true at position 0 of every model.
  - ☀️ However, they are not congruent;  $p \vee \neg p$  holds at all positions of every model, while  $\neg \ominus(p \vee \neg p)$  holds only at position 0.

# Congruences

 A minimal set of operators:

$$\neg, \vee, \circ, \mathcal{W}, \ominus, \mathcal{B}$$

Other operators could be encoded:

$$\begin{array}{ll} \square p \cong p \mathcal{W} \text{ False} & \ominus p \cong \neg \ominus \neg p \\ \diamond p \cong \neg \square \neg p & \boxplus p \cong p \mathcal{B} \text{ False} \\ p \mathcal{U} q \cong (p \mathcal{W} q \wedge \diamond q) & \diamond p \cong \neg \boxplus \neg p \\ p \mathcal{S} q \cong (p \mathcal{B} q \wedge \diamond q) & \end{array}$$

 Weak vs. strong operators:

$$\begin{array}{ll} \ominus p \cong (\ominus p \wedge \ominus \text{True}) & \ominus p \cong (\ominus p \wedge \ominus \text{False}) \\ p \mathcal{U} q \cong (p \mathcal{W} q \wedge \diamond q) & p \mathcal{W} q \cong (p \mathcal{U} q \vee \square p) \\ p \mathcal{S} q \cong (p \mathcal{B} q \wedge \diamond q) & p \mathcal{B} q \cong (p \mathcal{S} q \vee \boxplus p) \end{array}$$

# Congruences (cont.)

## Duality:

$$\neg \bigcirc p \cong \bigcirc \neg p$$

$$\neg \ominus p \cong \ominus \neg p$$

$$\neg \diamond p \cong \square \neg p$$

$$\neg \odot p \cong \ominus \neg p$$

$$\neg \square p \cong \diamond \neg p$$

$$\neg \diamond p \cong \square \neg p$$

$$\neg (p \mathcal{U} q) \cong (\neg q) \mathcal{W} (\neg p \wedge \neg q)$$

$$\neg \boxplus p \cong \diamond \neg p$$

$$\neg (p \mathcal{S} q) \cong (\neg q) \mathcal{B} (\neg p \wedge \neg q)$$

$$\neg (p \mathcal{U} q) \cong (\neg p) \mathcal{R} (\neg q)$$

$$\neg (p \mathcal{W} q) \cong (\neg q) \mathcal{U} (\neg p \wedge \neg q)$$

$$\neg (p \mathcal{B} q) \cong (\neg q) \mathcal{S} (\neg p \wedge \neg q)$$

$$\neg (p \mathcal{R} q) \cong (\neg p) \mathcal{U} (\neg q)$$

$$\neg \exists x: p \cong \forall x: \neg p$$

$$\neg \forall x: p \cong \exists x: \neg p$$

A formula is in the *negation normal form* if negation only occurs in front of an atomic proposition.

Every PTL/QPTL formula can be converted into an equivalent formula in the negation normal form.



# Congruences (cont.)

🌐 Expansion formulae:

$$\Box p \cong p \wedge \bigcirc \Box p$$

$$\Diamond p \cong p \vee \bigcirc \Diamond p$$

$$p \mathcal{U} q \cong q \vee (p \wedge \bigcirc (p \mathcal{U} q))$$

$$p \mathcal{W} q \cong q \vee (p \wedge \bigcirc (p \mathcal{W} q))$$

$$p \mathcal{R} q \cong (q \wedge p) \vee (q \wedge \bigcirc (p \mathcal{R} q))$$


$$\Box p \cong p \wedge \ominus \Box p$$

$$\Diamond p \cong p \vee \ominus \Diamond p$$

$$p \mathcal{S} q \cong q \vee (p \wedge \ominus (p \mathcal{S} q))$$

$$p \mathcal{B} q \cong q \vee (p \wedge \ominus (p \mathcal{B} q))$$

Note: we have seen that these expansion formulae are essential in translation of a temporal formula into an equivalent Büchi automaton.

 Idempotence:

$$\diamond\diamond p \cong \diamond p$$

$$\diamond\diamond p \cong \diamond p$$

$$\square\square p \cong \square p$$

$$\square\square p \cong \square p$$

$$p \mathcal{U} (p \mathcal{U} q) \cong p \mathcal{U} q$$

$$p \mathcal{S} (p \mathcal{S} q) \cong p \mathcal{S} q$$

$$p \mathcal{W} (p \mathcal{W} q) \cong p \mathcal{W} q$$

$$p \mathcal{B} (p \mathcal{B} q) \cong p \mathcal{B} q$$

$$(p \mathcal{U} q) \mathcal{U} q \cong p \mathcal{U} q$$

$$(p \mathcal{S} q) \mathcal{S} q \cong p \mathcal{S} q$$






$$(p \mathcal{W} q) \mathcal{W} q \cong p \mathcal{W} q$$

$$(p \mathcal{B} q) \mathcal{B} q \cong p \mathcal{B} q$$

# Concluding Remarks

- 🌐 PTL can be extended in other ways to be as expressive as Büchi automata, i.e., to express all  $\omega$ -regular properties.
- 🌐 For example, the industry standard IEEE 1850 Property Specification Language (PSL) is based on an extension that adds classic regular expressions.
- 🌐 Regarding translation of a temporal formula into an equivalent Büchi automaton, there have been quite a few algorithms proposed in the past.
- 🌐 How to obtain an automaton as small as possible remains interesting, for both theoretical and practical reasons.

# References

-  E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking*, The MIT Press, 1999.
-  E.A. Emerson. Temporal and modal logic, *Handbook of Theoretical Computer Science* (Vol. B), MIT Press, 1990.
-  G.J. Holzmann. *The SPIN Model Checker: Primer and Reference Manual*, Addison-Wesley, 2003.
-  Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*, Springer, 1992.
-  Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety*, Springer, 1995.