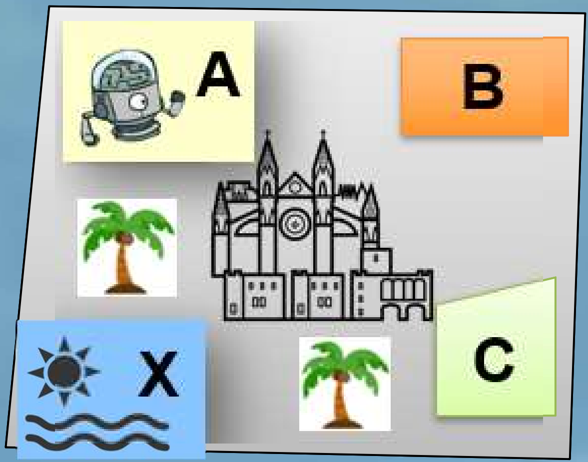
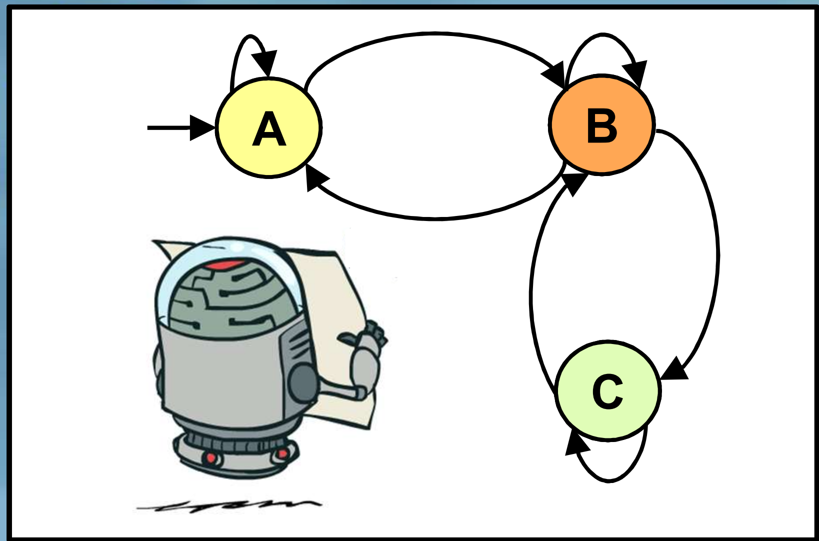
Graz University of Technology
Institute for Applied Information Processing and Communications

Automata and LTL Model Checking
Part-3

Bettina Könighofer





Model Checking SS21

June 10th 2021

Outline

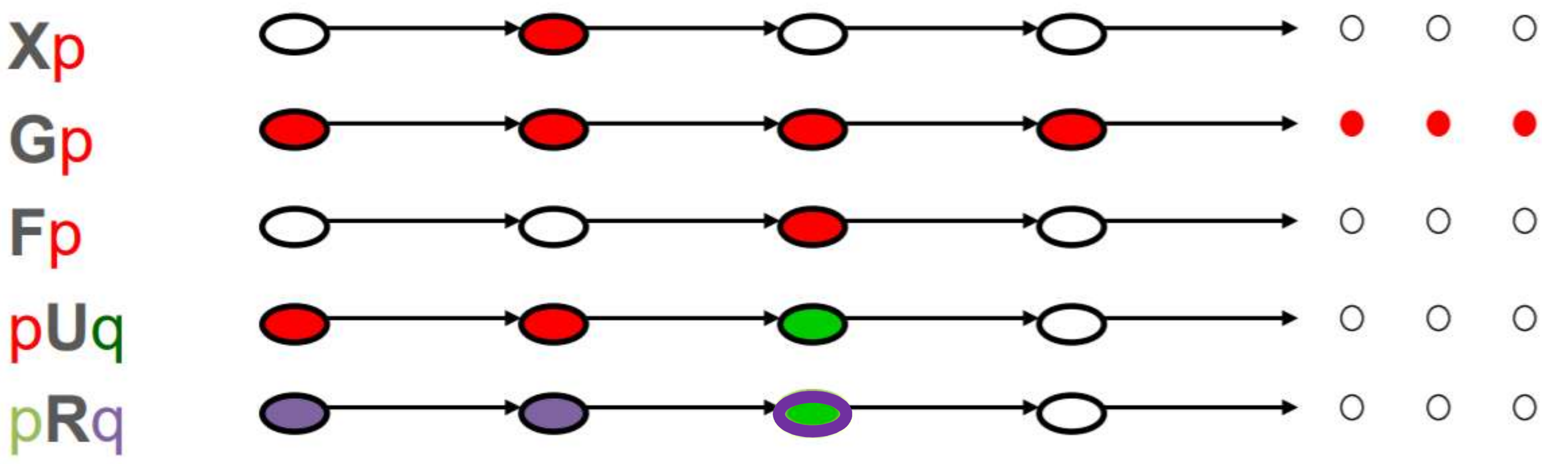
- Finite automata on finite words
- Automata on infinite words (Büchi automata)
- Deterministic vs non-deterministic Büchi automata
- Intersection of Büchi automata
- Checking emptiness of Büchi automata
- Generalized Büchi automata
- Automata and Kripke Structures
- Model checking using automata
- **Translation of LTL to Büchi automata**

Translation of LTL to Büchi automata

Given an LTL formula φ, construct a generalized Büchi automaton $\mathcal{A}_\varphi$

- $\mathcal{A}_\varphi$ accepts exactly all the traces that satisfy φ

Recall LTL Semantics



Translation of LTL to Büchi automata

Given an LTL formula φ, construct a generalized Büchi automaton $\mathcal{A}_\varphi$

1. Translate φ into generalized Büchi Automaton
2. Translate generalized Büchi to Büchi automaton

Rewriting

- Algorithm only handles
  - $\neg, \wedge, \vee, X, U, (R)$
- Use rewriting Rules $\neg G\varphi = F\neg\varphi$
  - $F\varphi = true\ U\varphi$
  - $G\varphi = \neg F\neg\varphi = false\ R\ \varphi$
  - $\neg(\varphi R\psi) = \neg\varphi U\neg\psi$

From LTL formula φ to GBA $\mathcal{A}_\varphi$

- Each state of the automata is **labelled** with **a set of properties/sub-formulas** that should be satisfied **on paths starting at that state**

Closure of an LTL formula φ – cl(φ)

- cl(φ)
  - … subformulas of φ and their negation
  - … subsets of cl(φ) define state space of $\boldsymbol{\mathcal{A}}_{\varphi}$

Closure of an LTL formula φ – cl(φ)

- cl(φ)
  - … subformulas of φ and their negation
- Formally:
  - $\varphi \in cl(\varphi)$.
  - **If** $\boldsymbol{\varphi_1 \in cl(\varphi), then\ \neg\varphi_1 \in cl(\varphi)}$.
  - **If** $\boldsymbol{\neg\varphi_1 \in cl(\varphi), then\ \varphi_1 \in cl(\varphi)}$.
  - If $\varphi_1 \vee \varphi_2 \in cl(\varphi), then\ \varphi_1 \in cl(\varphi)\ and\ \varphi_2 \in cl(\varphi)$.
  - If $X\ \varphi_1 \in cl(\varphi), then\ \varphi_1 \in cl(\varphi)$.
  - If $\varphi_1\ U\ \varphi_2 \in cl(\varphi), then\ \varphi_1 \in cl(\varphi)\ and\ \varphi_2 \in cl(\varphi)$.

Closure of an LTL formula φ – cl(φ)

- cl(φ)
  - … subformulas of φ and their negation

- $\varphi \coloneqq (\neg p \; \mathrm{U} \; ((\mathrm{X}q) \vee r))$
- Compute cl(φ)

Closure of an LTL formula φ

- cl(φ)
  - … subformulas of φ and their negation

- $\varphi \coloneqq (\neg \mathrm{p}\ \mathrm{U}\ ((\mathrm{Xq}) \vee \mathrm{r}))$
- $cl((\neg pU((Xq) \vee r))) =$

$$\{ \left(\neg pU\left((Xq) \vee r\right)\right), \neg(\neg pU((Xq) \vee r)),$$
$$\neg p, p,$$
$$((Xq) \vee r), \neg((Xq) \vee r),$$
$$(Xq), \neg(Xq),$$
$$q, \neg q, r, \neg r \}$$

Good sets in cl(φ)

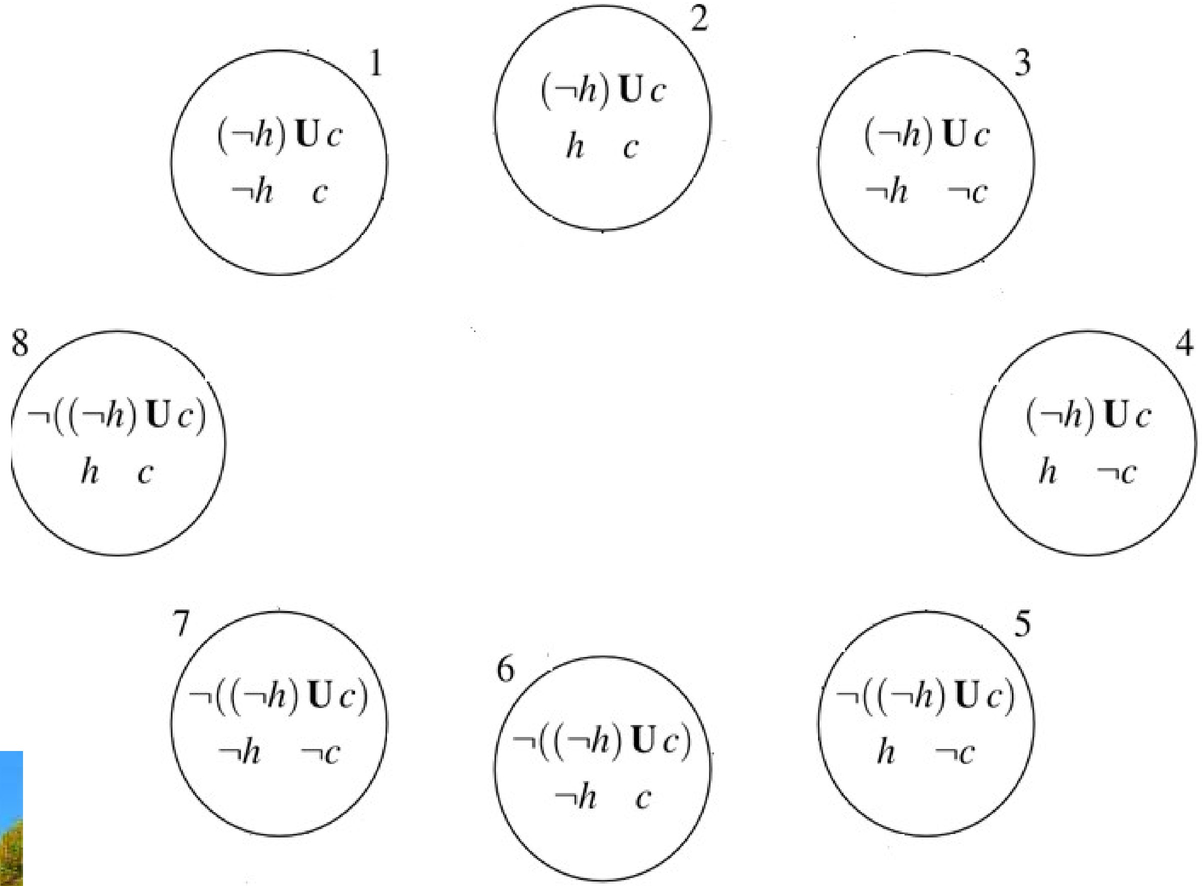
- $S \subseteq cl(\varphi)$ is ***good*** *in* $cl(\varphi)$ if S is a maximal set of formulas in $cl(\varphi)$ that is **consistent**:

1. For all $\varphi_1 \in cl(\varphi)$: $\varphi_1 \in S \Leftrightarrow \neg \varphi_1 \notin S$,
2. For all $\varphi_1 \vee \varphi_2 \in cl(\varphi)$: $\varphi_1 \vee \varphi_2 \in S \Leftrightarrow$ at least one of φ_1, φ_2 is in S.

The set of all **good sets** of cl(φ) defines the state space of $\mathcal{A}_\varphi$

Give the state space Q of $\mathcal{A}_\varphi$ representing

$\varphi = (\neg h \; U \; c)$


1
$(\neg h)\,\mathbf{U}\,c$
$\neg h \quad c$
2
$(\neg h)\,\mathbf{U}\,c$
$h \quad c$
3
$(\neg h)\,\mathbf{U}\,c$
$\neg h \quad \neg c$
4
$(\neg h)\,\mathbf{U}\,c$
$h \quad \neg c$
5
$\neg((\neg h)\,\mathbf{U}\,c)$
$h \quad \neg c$
6
$\neg((\neg h)\,\mathbf{U}\,c)$
$\neg h \quad c$
7
$\neg((\neg h)\,\mathbf{U}\,c)$
$\neg h \quad \neg c$
8
$\neg((\neg h)\,\mathbf{U}\,c)$
$h \quad c$

From LTL formula φ to GBA $\mathcal{A}_\varphi$

$\mathcal{A}_\varphi = (\mathcal{P}(AP), \mathbf{Q}, \mathbf{\Delta}, \mathbf{Q}^0, \mathbf{F})$

- $\mathbf{Q} \subseteq \mathcal{P}(cl(\varphi))$ is the set of all the good sets in $cl(\varphi)$.

Each state of $\mathcal{A}_\varphi$ is **labelled** with **a set of properties** that should be satisfied **on all paths starting at that state**

From LTL formula φ to GBA $\mathcal{A}_\varphi$

$\mathcal{A}_\varphi = (\mathcal{P}(AP), \mathbf{Q}, \mathbf{\Delta}, \mathbf{Q}^0, \mathbf{F})$

- For q, q' ∈Q and $\sigma \subseteq AP$, $(q,\sigma,q') \in \mathbf{\Delta}$ if:
  1. $\sigma = q' \cap AP$
  2. $\boldsymbol{X}\varphi_1 \in q \Leftrightarrow \varphi_1 \in q'$
  3. φ_1 U $\varphi_2 \in$ q ⇔ either $\varphi_2 \in$ q or both $\varphi_1 \in$ q **and** φ_1 U $\varphi_2 \in$ q'

$$\varphi_1 U \varphi_2 \equiv \varphi_2 \vee (\varphi_1 \wedge X(\varphi_1 U \varphi_2))$$

Each state of $\mathcal{A}_\varphi$ is **labelled** with **a set of properties** that should be satisfied **on all paths starting at that state**

From LTL formula φ to GBA $\mathcal{A}_\varphi$

$\mathcal{A}_\varphi = (\mathcal{P}(AP), \mathbf{Q}, \mathbf{\Delta}, \mathbf{Q}^0, \mathbf{F})$

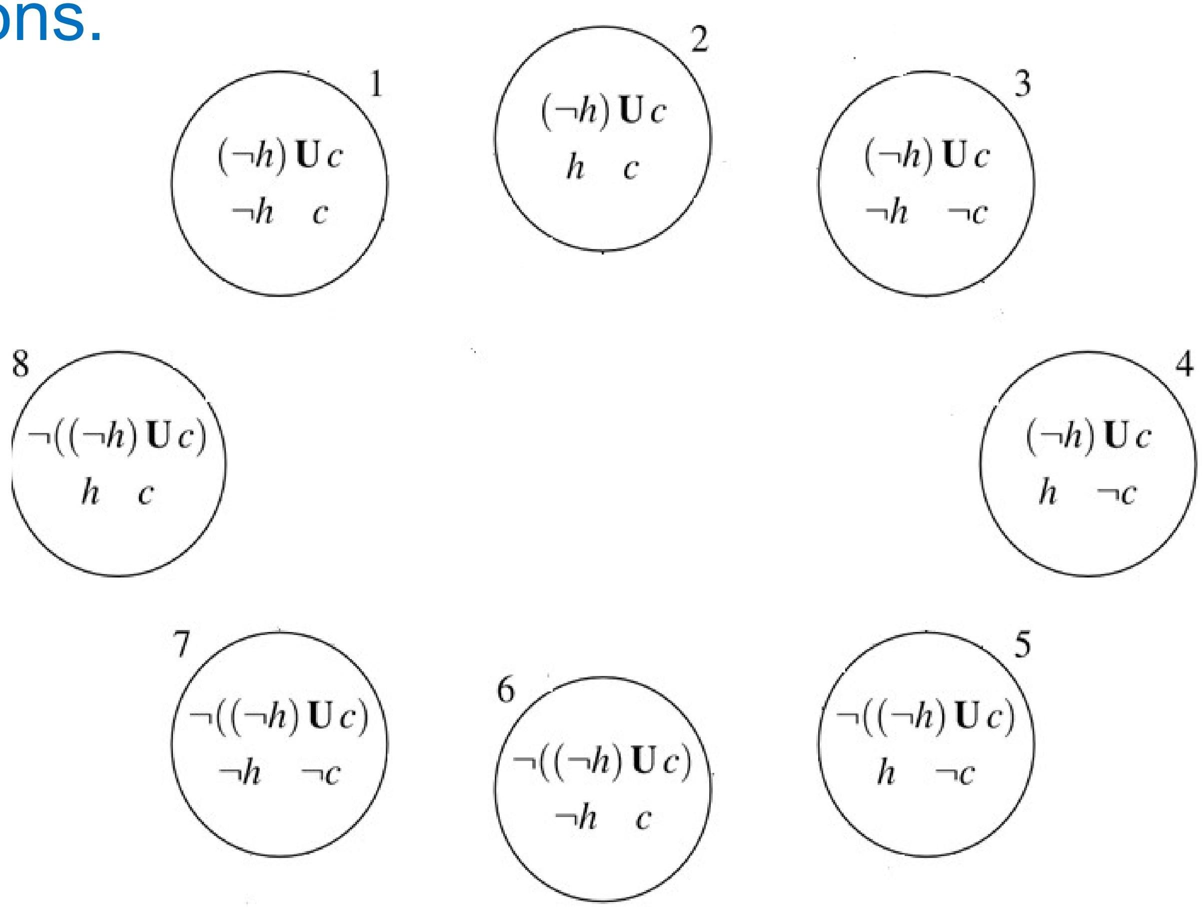
- For q, q' ∈Q and $\sigma \subseteq AP$, $(q,\sigma,q') \in \mathbf{\Delta}$ if:
  1. $\sigma = q' \cap AP$
  2. $X\varphi_1 \in q \Leftrightarrow \varphi_1 \in q'$
  3. $\varphi_1 \, U \, \varphi_2 \in q \Leftrightarrow$ either $\varphi_2 \in q$ **or** both $\varphi_1 \in q$ **and** $\varphi_1 \, U \, \varphi_2 \in q'$
  4. $\neg(\varphi_1 \, U \, \varphi_2) \in q \Leftrightarrow$ either $\neg\varphi_2 \in q$ **and** either $\neg \varphi_1 \in q$ **or** $\neg (\varphi_1 \, U \, \varphi_2) \in q'$

$$\neg(\varphi_1 U \varphi_2) = \neg\varphi_1 R \neg\varphi_2$$

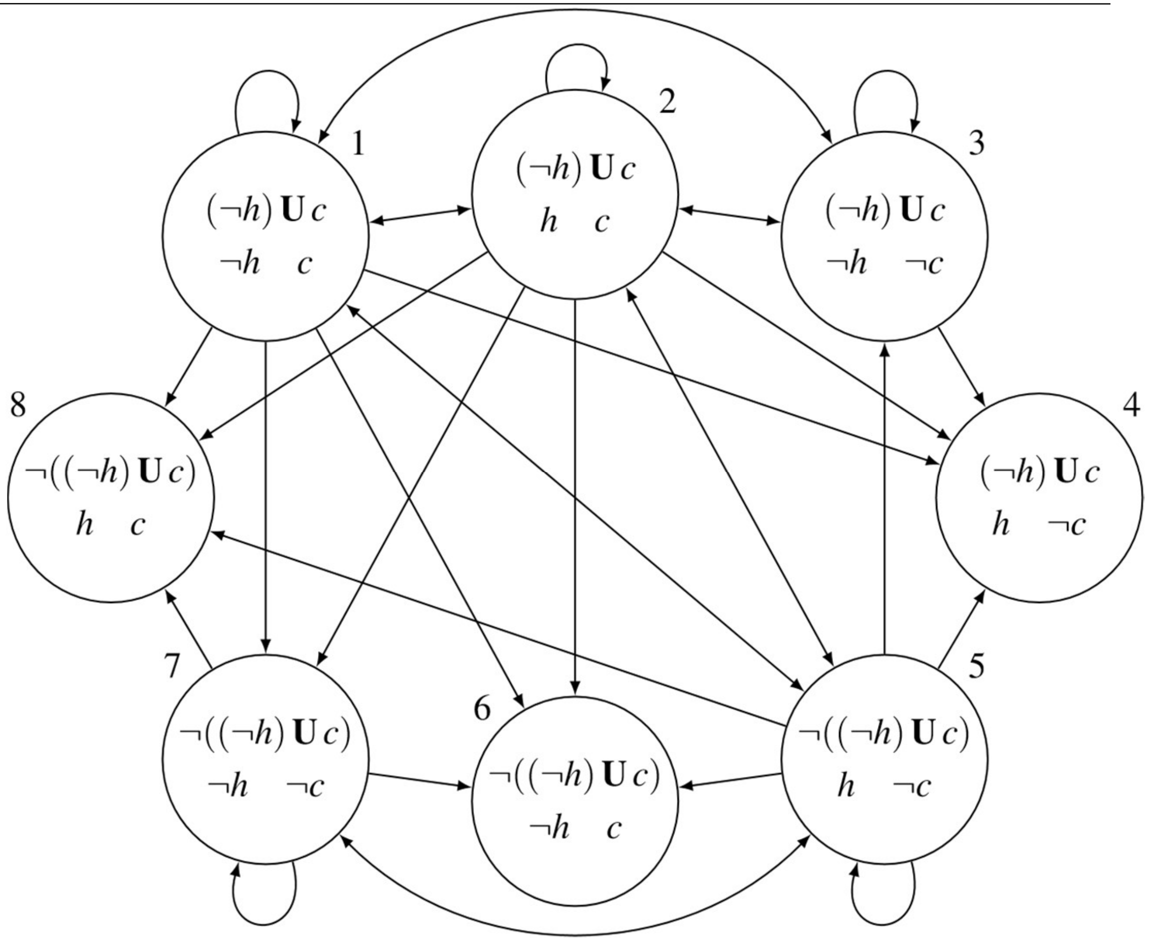
$$\varphi_1 R \varphi_2 \equiv \varphi_2 \wedge (\varphi_1 \vee X(\varphi_1 R \, \varphi_2))$$

$\varphi = (\neg h \; U \; c)$

Draw the tranistions.



$\varphi = (\neg h \; \mathrm{U} \; c)$



States 4, 6, 8 have no outgoing edges

From LTL formula φ to GBA $\mathcal{A}_\varphi$

$\mathcal{A}_\varphi = (\mathcal{P}(\mathrm{AP}), \mathbf{Q}, \boldsymbol{\Delta}, \mathbf{Q}^0, \mathbf{F})$

- What are the initial states?

Each state of $\mathcal{A}_\varphi$ is **labelled** with **a set of properties** that should be satisfied **on all paths starting at that state**

From LTL formula φ to GBA $\mathcal{A}_\varphi$

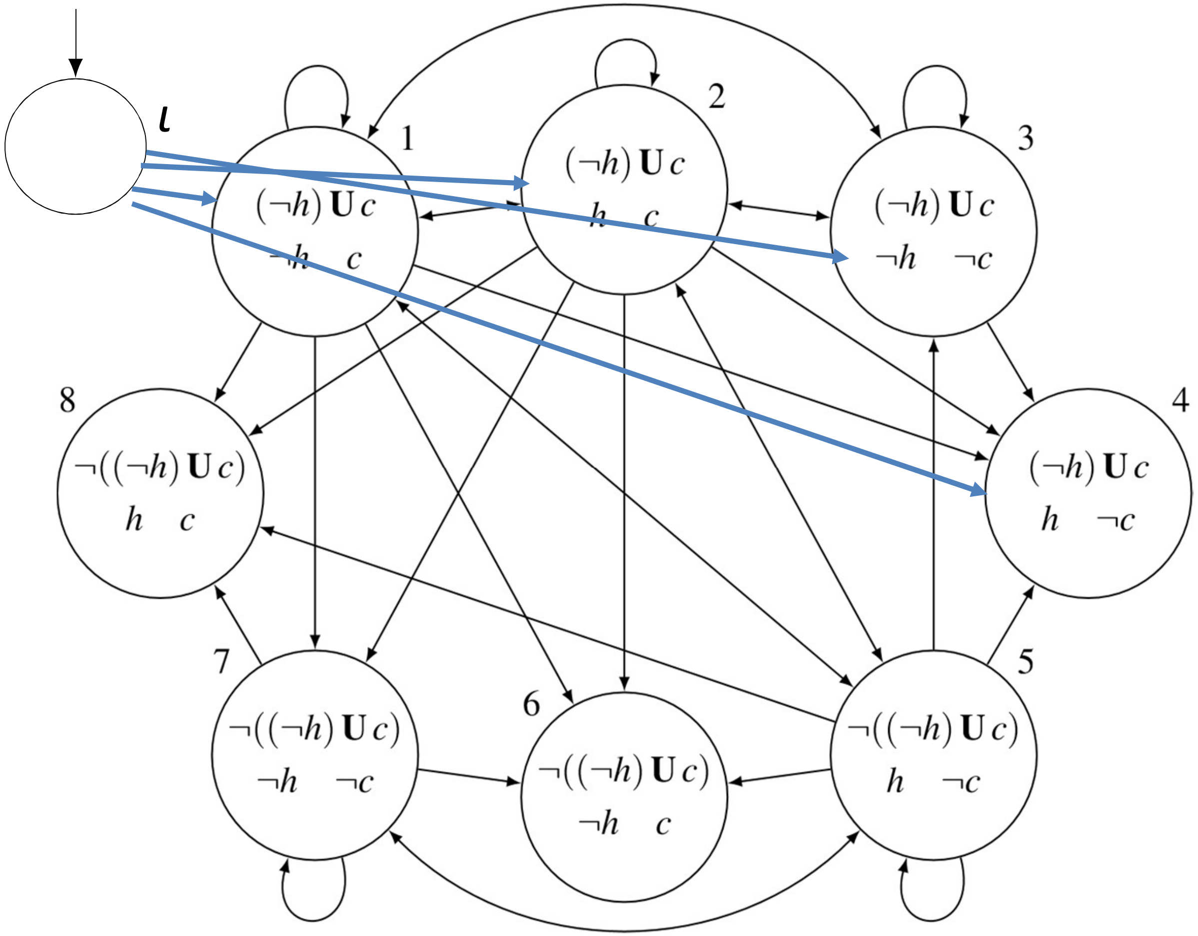
$\mathcal{A}_\varphi = (\mathcal{P}(AP), \mathbf{Q}, \mathbf{\Delta}, \{\iota\}, \mathbf{F})$

- $\mathbf{Q} \subseteq \mathcal{P}(cl(\varphi)) \cup \{\iota\}$ is the set of all the good sets in $cl(\varphi) \cup \{\iota\}$.

- $(\iota, \alpha, q) \in \mathbf{\Delta} \Leftrightarrow \varphi \in \mathbf{q}$ and $\sigma = q \cap AP$

Each state of $\mathcal{A}_\varphi$ is **labelled** with **a set of properties** that should be satisfied **on all paths starting at that state**

$\varphi = (\neg h \; \mathrm{U} \; c)$



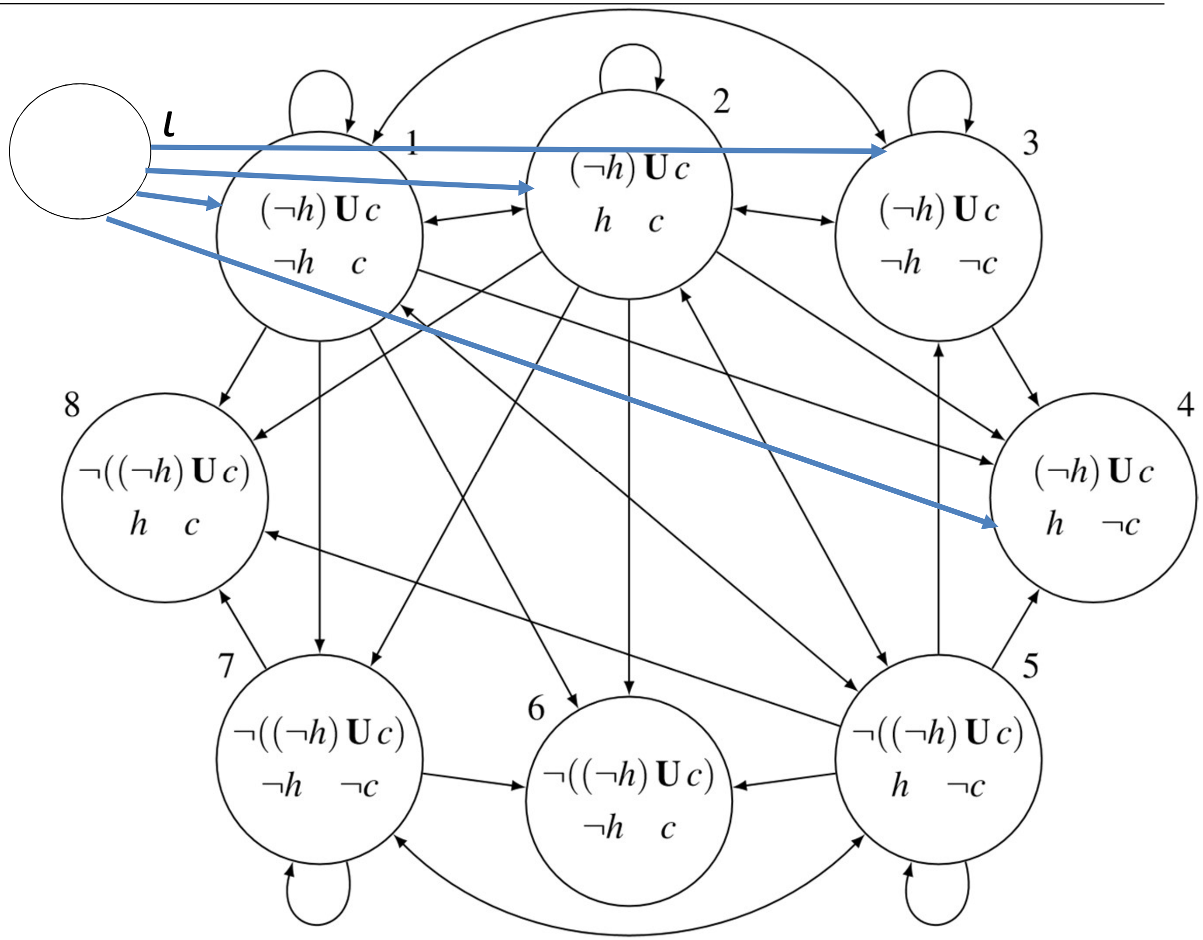
From LTL formula φ to GBA $\mathcal{A}_\varphi$

$\mathcal{A}_\varphi = (\mathcal{P}(AP), \mathbf{Q}, \boldsymbol{\Delta}, \{\boldsymbol{\iota}\}, \mathbf{F})$

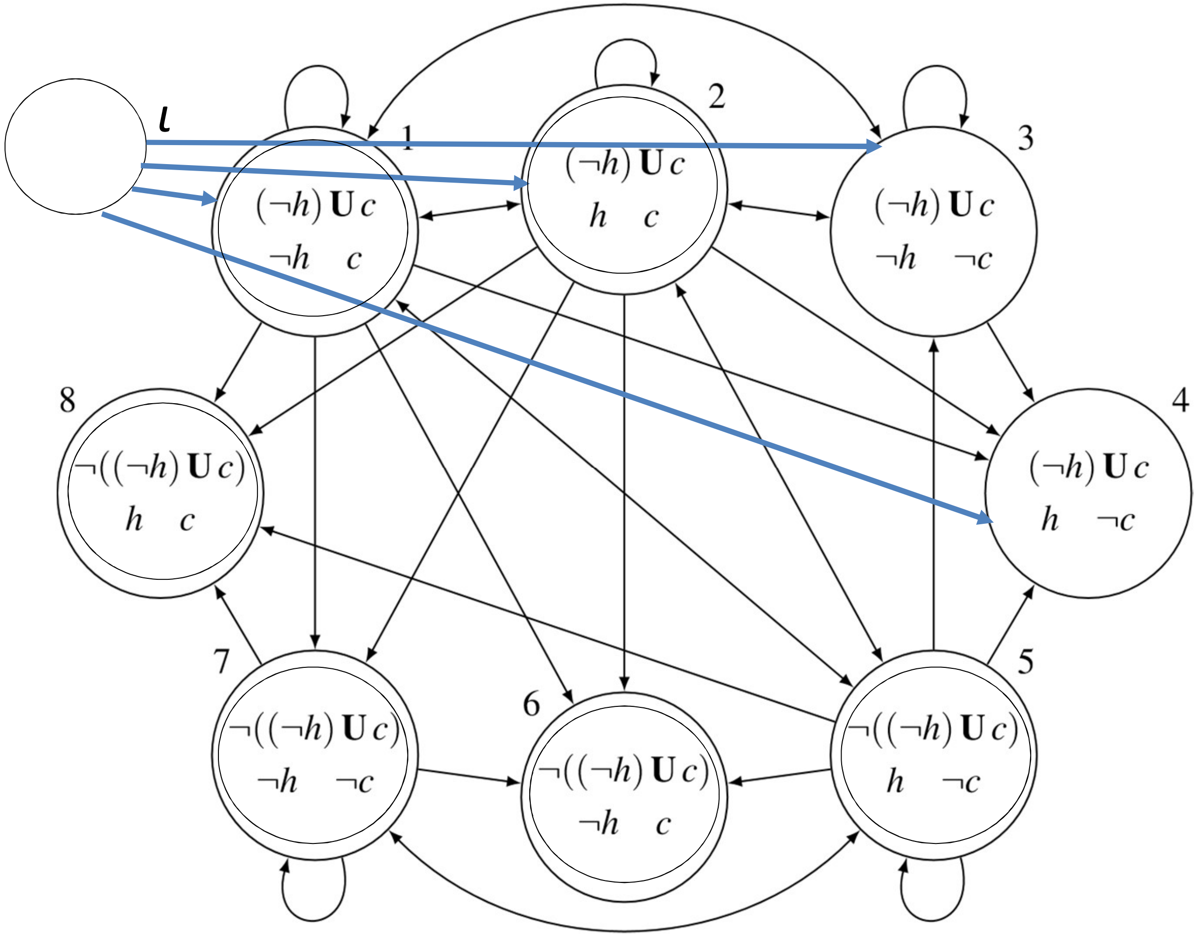
- $\mathbf{Q} \subseteq \mathcal{P}(cl(\varphi)) \cup \{\boldsymbol{\iota}\}$ is the set of all the good sets in $cl(\varphi)$ $\cup \{\boldsymbol{\iota}\}$.
  - $(\boldsymbol{\iota}, \alpha, q) \in \boldsymbol{\Delta} \Leftrightarrow \varphi \in \mathbf{q}$ and $\sigma = q \cap AP$
- For every $\varphi_1 \mathbf{U} \varphi_2 \in cl(\varphi)$, $\mathbf{F}$ includes the set
  - $F_{\varphi_1 \mathbf{U} \varphi_2} = \{q \in \mathbf{Q} \mid \varphi \in q$ or $\neg(\varphi_1 \mathbf{U} \varphi_2) \in q\}$.

$\varphi = (\neg h \text{ U } c)$

- What is F?



$\varphi = (\neg h \ \mathbf{U} \ c)$



- F = {{1, 2, 5, 6, 7, 8}}

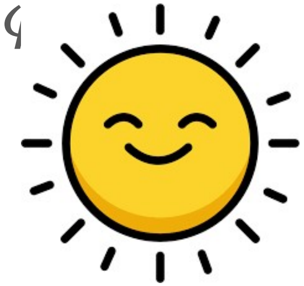
From LTL formula φ to GBA $\mathcal{A}_\varphi$

$\mathcal{A}_\varphi = (\mathcal{P}(AP), \mathbf{Q}, \mathbf{\Delta}, \{\iota\}, \mathbf{F})$

- $\mathbf{Q} \subseteq \mathcal{P}(cl(\varphi)) \cup \{\iota\}$ is the set of all the good sets in $cl(\varphi)$ $\cup \{\iota\}$.
  - $(\iota,\alpha,q) \in \mathbf{\Delta} \Leftrightarrow \varphi \in \mathbf{q}$ and $\sigma = q \cap AP$
- For every φ_1 U $\varphi_2 \in cl(\varphi)$, $\mathbf{F}$ includes the set
  - $F_{\varphi_1 \, U \, \varphi_2} = \{q \in \mathbf{Q} \mid \varphi \in q \text{ or } \neg(\varphi_1 \, U \, \varphi_2) \in q\}$.

- What is the complexity?

From LTL formula φ to GBA $\mathcal{A}_\varphi$



$\mathcal{A}_\varphi = (\mathcal{P}(AP), \mathbf{Q}, \mathbf{\Delta}, \{\mathbf{\iota}\}, \mathbf{F})$

- $\mathbf{Q} \subseteq \mathcal{P}(cl(\varphi)) \cup \{\mathbf{\iota}\}$ is the set of all the good sets in $cl(\varphi)$ $\cup \{\mathbf{\iota}\}$.
  - $(\mathbf{\iota}, \alpha, q) \in \mathbf{\Delta} \Leftrightarrow \varphi \in \mathbf{q}$ and $\sigma = q \cap AP$
- For every $\varphi_1 \, U \, \varphi_2 \in cl(\varphi)$, $\mathbf{F}$ includes the set
  - $F_{\varphi_1 U \varphi_2} = \{q \in \mathbf{Q} \mid \varphi \in q \text{ or } \neg(\varphi_1 \, U \, \varphi_2) \in q\}$.

- What is the complexity?
  - $\mathcal{A}_\varphi$ is **always exponential** in the size of φ.

Algorithm in the Book (7.9)

$\mathcal{A}_\varphi = (P(AP), \mathbf{Q}, \mathbf{\Delta}, \mathbf{Q}^0, \mathbf{F})$

- $\mathbf{Q} \subseteq P(cl(\varphi))$ is the set of all the good sets in $cl(\varphi)$.
- For q, q' ∈Q and $\sigma \subseteq AP$, $(q,\sigma,q') \in \mathbf{\Delta}$ if:
  1. **$\sigma = q \cap AP$ → Push labels forward**
  2. $\mathbf{X}\, \varphi_1 \in q \Leftrightarrow \varphi_1 \in q'$,
  3. $\varphi_1\, U\, \varphi_2 \in q \Leftrightarrow$ either $\varphi_2 \in q$ or both $\varphi_1 \in q$ and $\varphi_1\, U\, \varphi_2 \in q'$ ‘

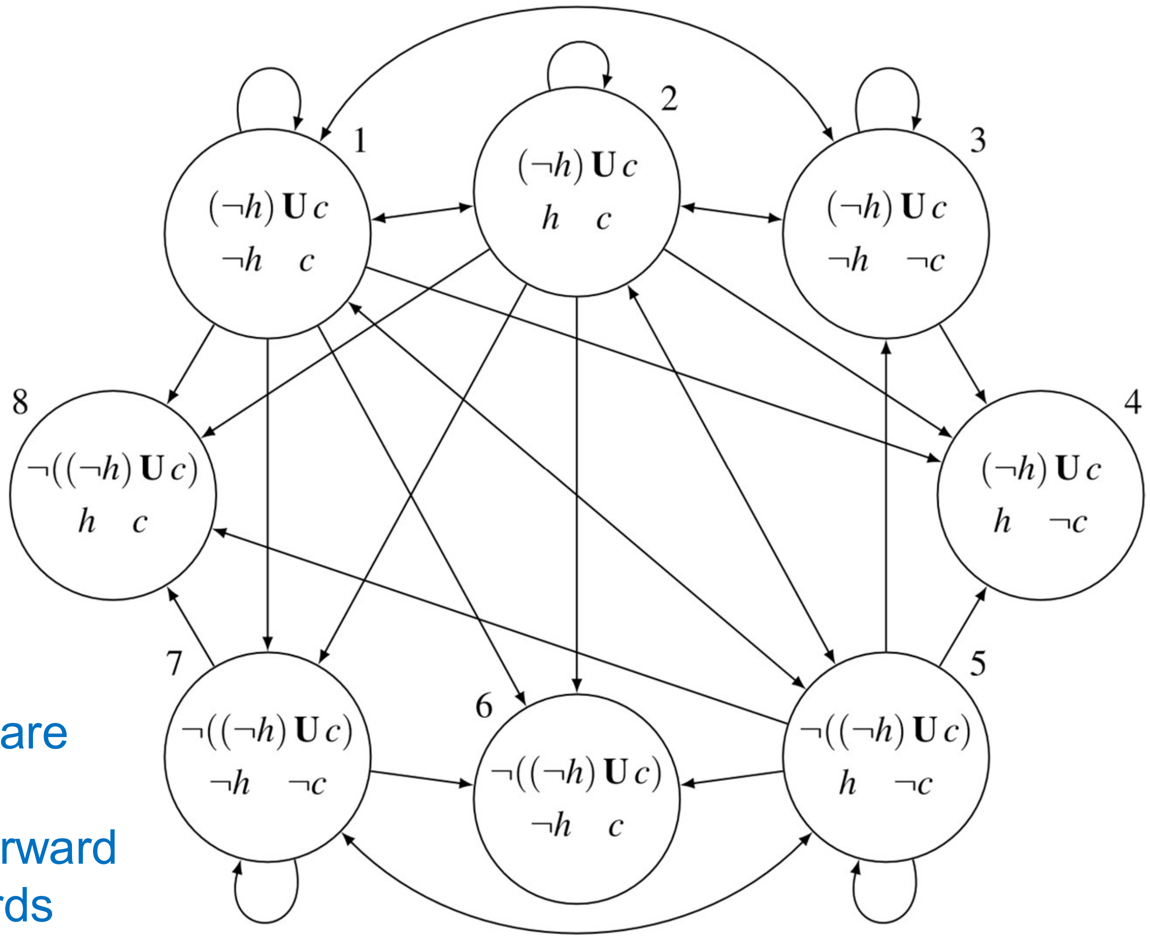
- $\mathbf{Q}^0$ is the set of all states $q \in \mathbf{Q}$ for which $\varphi \in q$.

- For every $\varphi_1\, U\, \varphi_2 \in cl(\varphi)$, $\mathbf{F}$ includes the set

$$F_{\varphi_1 U \varphi_2} = \{q \in \mathbf{Q} \mid \varphi \in q \text{ or } \neg(\varphi_1\, U\, \varphi_2) \in q\}.$$

Book: Fig. 7.10

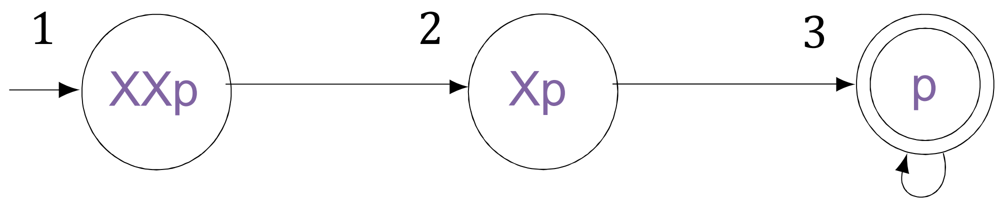
- Initial States: {1, 2, 3, 4}
- F = {{1, 2, 5, 6, 7, 8}}.



Homework:
Explain why both algorithm are correct.
Why does pushing labels forward and pushing labels backwards both work in this case?

Efficient translation of LTL to Büchi [Gerth, Peled, Vardi and Wolper]

- $\mathcal{A}_\varphi$ does not have to be always exponential in the size of φ (but sometimes it is).
- The idea: each state includes **only** subformulas that are required to be true for this state.



Example: φ **= X X** ***p***
Subformulas of φ: {X X p, X p, p}
number of subsets = $2^3 = 8$

But: in state 1 we care only about XXp, not about Xp or p
in state 2 we only care about Xp;
in state 3 we only care about p ⇒ we only need three states!

Translation of LTL to Büchi automata

Given an LTL formula φ, construct a generalized Büchi automaton $\mathcal{A}_\varphi$

1. Rewrite φ in **Negation Normal Form**
   - Apply Rewriting Rules
2. **New Efficient Translation**
   - Turn φ into generalized Büchi Automaton
3. Translate generalized Büchi to Büchi automaton

Rewriting

- Negated Normal Form
  - Negation appears only in front of literals
    - $\neg\neg\varphi = \varphi$
    - $\neg(X\varphi) = X\neg\varphi$
    - $\neg G\varphi = F\neg\varphi$
    - $\neg F\varphi = G\neg\varphi$
    - $\neg(\varphi U\psi) = \neg\varphi R\neg\psi$
    - $\neg(\varphi R\psi) = \neg\varphi U\neg\psi$

Rewriting

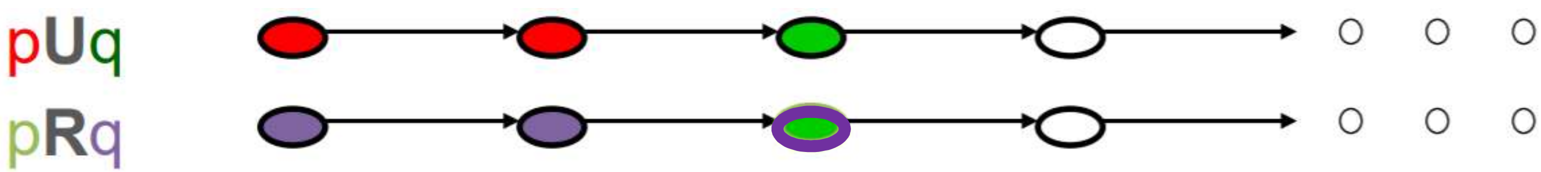
- Core Algorithm only handles
  - $\neg, \wedge, \vee, X, U, (R)$
- Use rewriting Rules$\neg G\varphi = F\neg\varphi$
  - $F\varphi = true\ U\varphi$
  - $G\varphi = \neg F\neg\varphi = false\ R\ \varphi$
  - $\neg(\varphi\ R\ \psi) = \neg\varphi\ U\neg\psi$

Efficient translation of LTL to Büchi

- φ is written in NNF
- Until and Release can be written as fixpoints:

$$\varphi_1 U \varphi_2 \equiv \varphi_2 \vee (\varphi_1 \wedge X(\varphi_1 U \varphi_2))$$

$$\begin{aligned}\varphi_1 R \varphi_2 &\equiv \varphi_2 \wedge (\varphi_1 \vee X(\varphi_1 R\, \varphi_2)) \\ &\equiv (\varphi_1 \wedge \varphi_2) \vee (\varphi_1 \vee X(\varphi_1 R\, \varphi_2))\end{aligned}$$



Efficient translation of LTL to Büchi

$$\varphi_1 U \varphi_2 \equiv \varphi_2 \vee (\varphi_1 \wedge X(\varphi_1 U \varphi_2))$$

$$\varphi_1 R \varphi_2 \equiv \varphi_2 \wedge \left(\varphi_1 \vee X(\varphi_1 R\, \varphi_2)\right)$$
$$\equiv (\varphi_1 \wedge \varphi_2) \vee (\varphi_1 \vee X(\varphi_1 R\, \varphi_2))$$

Two Observations

1. Requirements can be split

$$\varphi_1 U \varphi_2 \equiv \underbrace{\varphi_2}_{\text{Case 1}} \vee \underbrace{(\varphi_1 \wedge X(\varphi_1 U \varphi_2))}_{\text{Case 2}}$$

Efficient translation of LTL to Büchi

$$\varphi_1 U \varphi_2 \equiv \varphi_2 \vee (\varphi_1 \wedge X(\varphi_1 U \varphi_2))$$

$$\varphi_1 R \varphi_2 \equiv \varphi_2 \wedge \left(\varphi_1 \vee X(\varphi_1 R\, \varphi_2)\right)$$
$$\equiv (\varphi_1 \wedge \varphi_2) \vee (\varphi_1 \vee X(\varphi_1 R\, \varphi_2))$$

Two Observations

1. Requirements can be split
2. Requirements may refer to *current* and *next* states

$$\varphi_1 U \varphi_2 \equiv \varphi_2 \vee (\underbrace{\varphi_1}_{\text{Current State}} \wedge\, \underbrace{X(\varphi_1 U \varphi_2)}_{\text{Next State}})$$

Data Structure

- *Each node will store a set of properties that should be satisfied on paths starting at that state*
  - New: subformulas of φ that **need to be** processed; subformulas need to hold from **current** state q
  - Now: subformulas of φ that **have been** processed; subformulas need to hold from **current** state q
  - Next: subformulas that need to hold from **the next** state q'

- ID: Unique identifier of the node
- Incoming: incoming transitions for a node

Node

ID
Incoming:
New:
Now:
Next:

Open and Closed Nodes

Node

ID
Incoming:
New:
Now:
Next:

- *Each node will store a set of properties that should be satisfied on paths starting at that state*
  - New: subformulas of φ that **need to be** processed; subformulas need to hold from **current** state q
  - Now: subformulas of φ that **have been** processed; subformulas need to hold from **current** state q
  - Next: subformulas that need to hold from **the next** state q'

- Closed nodes: Set of all nodes, that are completely processed
  - New field is empty
  - Nodes in closed will be the states in $\mathcal{A}_\varphi$
- All nodes that must still be processed

procedure ***EfficientLTLBuchi***(φ)
Closed := ∅;
Open := ((n_0,{*init*},{φ}, ∅, ∅)) ; // Init
while *Open*≠ ∅ **do**
Choose *q* ∈ *Open*;
if *q*.*New* = 0 **then** // *q* is fully processed
Remove *q* from *Open*;
Update Closed(*q*);
else
Choose ψ ∈ q.New;
Move ψ from q.New to q.Now;
Update Split(q,ψ);
end if end while
define *F*; // GBA acceptance constraints
A := *Build Automaton*(*Closed*,*F*);
return A;
end procedure

Initialisation:

Single Node in Open:

ID: n_0

Incoming: {init}
New: { $(A\ U\ (B\ U\ C))$}
Now: ∅
Next: ∅

Nodes that will evolve from n_0 are the initial states of $\mathcal{A}_\varphi$

```
procedure EfficientLTLBuchi(φ)
Closed := ∅;
Open := ( (n0,{init},{φ}, ∅, ∅) ) ; // Init
while Open≠ ∅ do
  Choose q ∈ Open;
  if q.New = 0 then // q is fully processed
     Remove q from Open;
     Update Closed(q);
  else
     Choose ψ ∈ q.New;
     Move ψ from q.New to q.Now;
     Update Split(q,ψ);
  end if end while
define F; // GBA acceptance constraints
A := Build Automaton(Closed,F);
return A;
end procedure
```

Processing the Set Open

- For each node: process sub-formulas in New one by one
  - When we have $\varphi_1 \vee \varphi_2$ in the New list:
    - Split node: n1: New{φ_1} and n2: New{φ_2}

Processing the Set Open

- For each node: process sub-formulas in New one by one
  - When we have $\varphi_1 \vee \varphi_2$ in the New list:
    - Split node: n1: New{φ_1} and n2: New{φ_2}
  - When we have $\varphi_1 U \varphi_2$ in the New list we will use
    - $\varphi_1 U \varphi_2 \equiv \varphi_2 \vee (\varphi_1 \wedge X(\varphi_1 U \varphi_2))$
    - Split node: n1: New{φ_1 } Next{ $X(\varphi_1 U \varphi_2)$} and n2: New{φ_2}

Processing the Set Open

- For each node: process sub-formulas in New one by one
  - When we have $\varphi_1 \vee \varphi_2$ in the New list:
    - Split node: n1: New{φ_1} and n2: New{φ_2}
  - When we have $\varphi_1 U \varphi_2$ in the New list we will use
    - $\varphi_1 U \varphi_2 \equiv \varphi_2 \vee (\varphi_1 \wedge X(\varphi_1 U \varphi_2))$
    - Split node: n1: New{φ_1 } Next{ $(\varphi_1 U \varphi_2)$} and n2: New{φ_2}
  - When we have $\varphi_1 R\ \varphi_2$ in the New list we will use
    - $\varphi_1 R \varphi_2 \equiv (\varphi_1 \wedge \varphi_2) \vee (\varphi_1 \vee X(\varphi_1 R\ \varphi_2))$
    - Split node: n1: New{φ_2} Next{$(\varphi_1 R\ \varphi_2)$} and n2: New{φ_1, φ_2}

Processing the Set Open

- For each node: process sub-formulas in New one by one
  - When we have $\varphi_1 \vee \varphi_2$ in the New list:
    - Split node: n1: New{φ_1} and n2: New{φ_2}
  - When we have $\varphi_1 U \varphi_2$ in the New list we will use
    - $\varphi_1 U \varphi_2 \equiv \varphi_2 \vee (\varphi_1 \wedge X(\varphi_1 U \varphi_2))$
    - Split node: n1: New{φ_1 } Next{ $(\varphi_1 U \varphi_2)$} and n2: New{φ_2}
  - When we have $\varphi_1 R\ \varphi_2$ in the New list we will use
    - $\varphi_1 R \varphi_2 \equiv (\varphi_1 \wedge \varphi_2) \vee (\varphi_1 \vee X(\varphi_1 R\ \varphi_2))$
    - Split node: n1: New{φ_2} Next{$(\varphi_1 R\ \varphi_2)$} and n2: New{φ_1, φ_2}

procedure *Update_Split*(q, ψ)
  case of
    $\psi = p$ **or** $\psi = \neg p$: **skip**; // $p \in AP$
    $\varphi = \mathbf{X}\mu$: **add** μ **to** $q.Next$;
    $\varphi = \mu \vee \eta$: $q' := Split(q)$; **add** μ **to** $q.New$; **add** η **to** $q'.New$;
    $\varphi = \mu \wedge \eta$: **add** $\{\mu, \eta\}$ **to** $q.New$;
    $\varphi = \mu\, \mathbf{U}\, \eta$: $q' := Split(q)$; **add** η **to** $q.New$; **add** $\{\mu, \mathbf{X}(\mu\, \mathbf{U}\, \eta)\}$ **to** $q'.New$;
    $\varphi = \mu\, \mathbf{R}\, \eta$: $q' := Split(q)$; **add** $\{\mu, \eta\}$ **to** $q.New$; **add** $\{\eta, \mathbf{X}(\mu\, \mathbf{R}\, \eta)\}$ **to** $q'.New$;
  end case;
end procedure

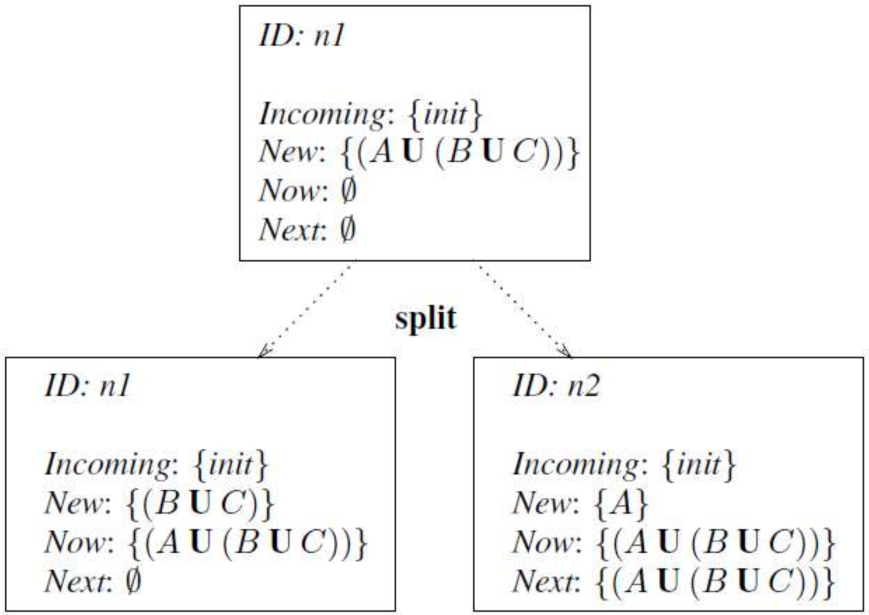
Processing the Set Open

- For each node: process sub-formulas in New one by one
  - When we have $\varphi_1 \vee \varphi_2$ in the New list:
    - Split node: n1: New{φ_1} and n2: New{φ_2}
  - When we have $\varphi_1 U \varphi_2$ in the New list we will use
    - $\varphi_1 U \varphi_2 \equiv \varphi_2 \vee (\varphi_1 \wedge X(\varphi_1 U \varphi_2))$
    - Split node: n1: New{φ_1 } Next{ $(\varphi_1 U \varphi_2)$} and n2: New{φ_2}
  - When we have $\varphi_1 R\ \varphi_2$ in the New list we will use
    - $\varphi_1 R \varphi_2 \equiv (\varphi_1 \wedge \varphi_2) \vee (\varphi_1 \vee X(\varphi_1 R\ \varphi_2))$
    - Split node: n1: New{φ_2} Next{$(\varphi_1 R\ \varphi_2)$} and n2: New{φ_1, φ_2}

procedure *Split*(q)
    create $q' = (freshID, q.Incoming, q.New, q.Now, q.Next)$;
    // q' identical to q except for ID
    return q';
end procedure

Processing the Set Open

- Process Node n1: When we have $\varphi_1 U \varphi_2$ in the New list we will use
  - $\varphi_1 U \varphi_2 \equiv \varphi_2 \vee (\varphi_1 \wedge X(\varphi_1 U \varphi_2))$
  - Split node: n1: New{φ_2} and n2: New{φ_1} Next: $(\varphi_1 U \varphi_2)$}

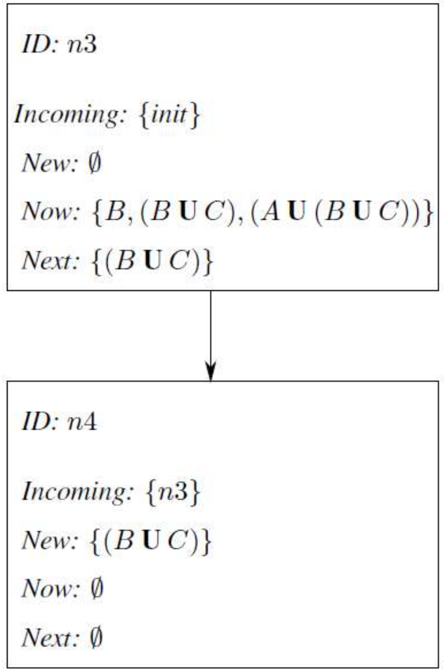


```
procedure EfficientLTLBuchi(φ)
Closed := ∅;
Open := ( (n0,{init},{φ}, ∅, ∅) ) ; // Init
while Open≠ ∅ do
  Choose q ∈ Open;
  if q.New = 0 then // q is fully processed
     Remove q from Open;
     Update Closed(q);
  else
     Choose ψ ∈ q.New;
     Move ψ from q.New to q.Now;
     Update Split(q,ψ);
  end if end while
define F; // GBA acceptance constraints
A := Build Automaton(Closed,F);
return A;
end procedure
```

Update_Closed(q)

- Applied if q.New is empty
- If a node q‘ with same values for Now and next exists:
  - Incomming edges of q are added to q‘
- Else
  - Insert q in Closed by. Create q‘ as possible successor.
  - q‘.New = q.Next

procedure *Update_Closed*(q)
    if there is $q' \in Closed$ **such that** $q.Now = q'.Now$ **and** $q.Next = q'.Next$ **then**
        $q'.Incoming := q'.Incoming \cup q.Incoming$;
    else
        add q **to** *Closed*;
        create $q' = (freshID, \{q\}, q.Next, \emptyset, \emptyset)$;
        // Node q' is a candidate successor of q
        add q' to *Open*
    end if
end procedure



procedure *EfficientLTLBuchi*(φ)
Closed := ∅;
Open := ((n_0,{*init*},{φ}, ∅, ∅)) ; // Init
while *Open*≠ ∅ **do**
  Choose *q* ∈ *Open*;
  if *q*.*New* = 0 **then** // *q* is fully processed
    Remove *q* from *Open*;
    Update Closed(*q*);
  else
    Choose ψ ∈ q.New;
    Move ψ from q.New to q.Now;
    Update Split(q,ψ);
  end if end while
define *F*; // GBA acceptance constraints
A := *Build Automaton*(*Closed*,*F*);
return A;
end procedure

Accepting States of GBA - Enforcing Eventualities

- *Multiple* accepting sets
  - One for each *Until* sub-formula (φ U ψ)
  - Nodes in Closed in which either
    - The *Now* field doesn't contain φ U ψ

  or

    - The *Now* field does contain ψ

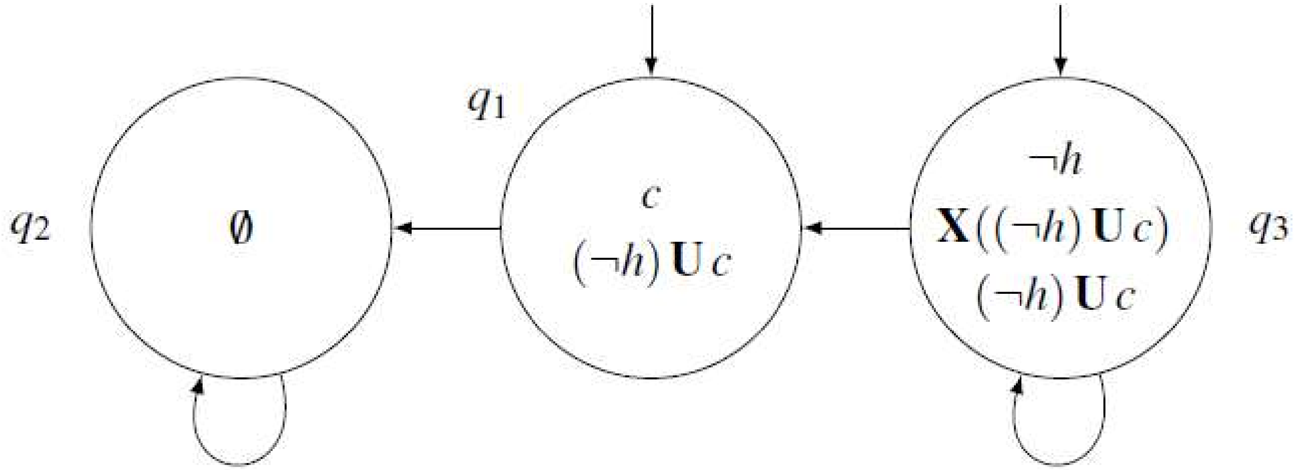
Construction of Kripke Structure

- Once open is empty
- For each node in closed
  - Create a new node with all the *Now* formulas
- Create edges between nodes using *Incoming*
- Use the set of sets of accepting states *F* from before

Construction of Kripke Structure

- The set of states S is the set of nodes in *Closed*.
- The set of initial states is $S_0 = \{q \in S \mid init \in q.Incoming\}$.
- The transition relation $R \subseteq S \times S$ is defined as follows: $(q, q') \in R$ if and only if $q \in q'.Incoming$.
- AP is the set of atomic propositions in φ. That is, $AP = \{p \mid p \in AP_\varphi\}$. Let $\overline{AP} = \{\neg p \mid p \in AP\}$.
- The labeling of states is $L(q) = q.Now$
- The generalized Büchi acceptance sets F which includes, for every subformula of φ of the form $\mu\,\mathbf{U}\,\eta$, a set $P_{\mu\mathbf{U}\eta} = \{q \mid \eta \in q.Now \text{ or } (\mu\,\mathbf{U}\,\eta) \notin q.Now\}$.

Construction of Kripke Structure



The Kripke structure resulting from algorithm EfficientLTLBuchi when given the formula (¬h)Uc


THANK
YOU!