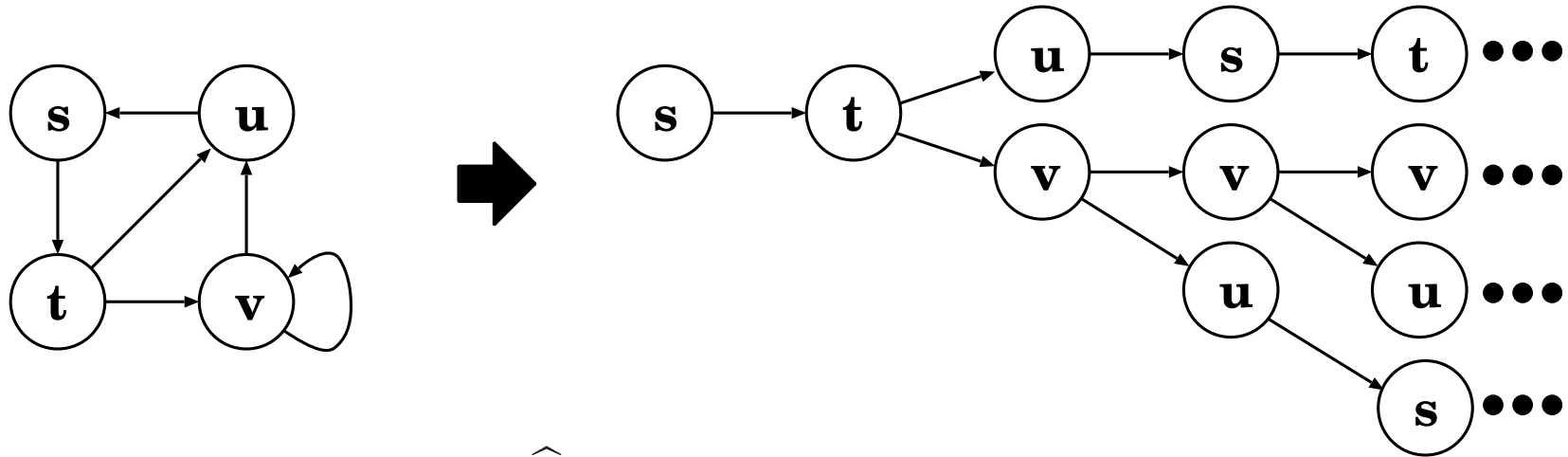

TEMPORAL LOGIC

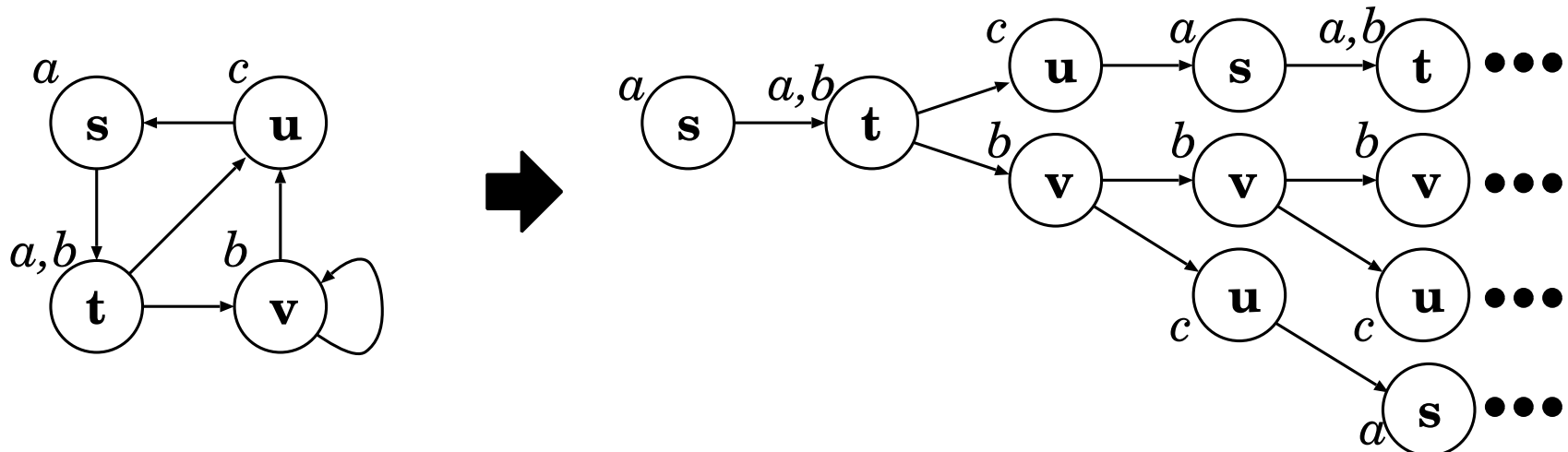
Kripke structures and computation trees

Given $(\hat{\mathcal{X}}, \mathcal{X}_{init}, \mathcal{N})$, we can **unwind** the graph into a **computation tree** rooted at each $i \in \mathcal{X}_{init}$



A **Kripke structure** is specified by $(\hat{\mathcal{X}}, \mathcal{X}_{init}, \mathcal{N}, \mathcal{A}, \mathcal{L})$

- \mathcal{A} is a set of **atomic properties**
- $\mathcal{L} : \hat{\mathcal{X}} \rightarrow 2^{\mathcal{A}}$ is a **labeling function** listing the atomic properties that **hold** in each state



A **temporal logic** talks about events and states occurring over **relative time**

- Is it possible to reach a state where both buffers are empty?
- Once both buffers are empty, can they ever both become full at the same time?
- Can we reach a stable set of states where race conditions cannot occur?
- Is it the case that, if race conditions occur, they cannot cause a deadlock?

Several classes of temporal logic exist, with different expressive power, for example:

- **CTL***
- **CTL**
- **LTL**

CTL* is **strictly more general** than either CTL or LTL (and even than their union)

CTL and LTL are **not comparable**

Assume a Kripke structure $(\hat{\mathcal{X}}, \mathcal{X}_{init}, \mathcal{N}, \mathcal{A}, \mathcal{L})$

CTL* formulas use

- **path quantifiers** to talk about the branching structure of the computation tree
- **temporal operators** to talk about properties of individual paths of the computation tree

CTL* formulas can be classified as

- **State formulas** (they may or may not hold in a state):
 - if $a \in \mathcal{A}$, a is a state formula
 - if p and p' are state formulas, $\neg p$, $p \vee p'$, $p \wedge p'$ are state formulas
 - if q is a path formula, $\text{E}q$ and $\text{A}q$ are state formulas
- **Path formulas** (they may or may not hold on a path):
 - if p is state formula, p is also a path formula
 - if q and q' are path formulas, $\neg q$, $q \vee q'$, $q \wedge q'$, $\text{X}q$, $\text{F}q$, $\text{G}q$, $q\text{U}q'$, $q\text{R}q'$ are path formulas

We assume infinite paths: simply redefine $\mathcal{N}(s) = \{s\}$ whenever $\mathcal{N}(s) = \emptyset$

If a state formula p holds in a state $\mathbf{s} \in \widehat{\mathcal{X}}$ we write $\mathbf{s} \models p$

If a path formula q holds on a path $\sigma = (\mathbf{s}_0, \mathbf{s}_1, \mathbf{s}_2, \dots)$ we write $\sigma \models q$

Given a path $\sigma = (\mathbf{s}_0, \mathbf{s}_1, \mathbf{s}_2, \dots)$ and an integer $i \in \mathbb{N}$, let $\sigma_{[i]} = (\mathbf{s}_i, \mathbf{s}_{i+1}, \dots)$

$$\mathbf{s} \models a \quad \Leftrightarrow \quad a \in \mathcal{L}(\mathbf{s})$$

$$\mathbf{s} \models \neg p \quad \Leftrightarrow \quad \mathbf{s} \not\models p$$

$$\mathbf{s} \models p \vee p' \quad \Leftrightarrow \quad \mathbf{s} \models p \text{ or } \mathbf{s} \models p'$$

$$\mathbf{s} \models p \wedge p' \quad \Leftrightarrow \quad \mathbf{s} \models p \text{ and } \mathbf{s} \models p'$$

$$\mathbf{s} \models \mathbf{E}q \quad \Leftrightarrow \quad \exists \sigma = (\mathbf{s}, \dots), \sigma \models q$$

exists a path

$$\mathbf{s} \models \mathbf{A}q \quad \Leftrightarrow \quad \forall \sigma = (\mathbf{s}, \dots), \sigma \models q$$

for all paths

$$\sigma \models p \quad \Leftrightarrow \quad \sigma = (\mathbf{s}, \dots) \text{ and } \mathbf{s} \models p$$

$$\sigma \models \neg q \quad \Leftrightarrow \quad \sigma \not\models q$$

$$\sigma \models q \vee q' \quad \Leftrightarrow \quad \sigma \models q \text{ or } \sigma \models q'$$

$$\sigma \models q \wedge q' \quad \Leftrightarrow \quad \sigma \models q \text{ and } \sigma \models q'$$

$$\sigma \models \mathbf{X}q \quad \Leftrightarrow \quad \sigma_{[1]} \models q$$

next

$$\sigma \models \mathbf{F}q \quad \Leftrightarrow \quad \exists i \in \mathbb{N}, \sigma_{[i]} \models q$$

finally, eventually

$$\sigma \models \mathbf{G}q \quad \Leftrightarrow \quad \forall i \in \mathbb{N}, \sigma_{[i]} \models q$$

generally, always

$$\sigma \models q \mathbf{U} q' \quad \Leftrightarrow \quad \exists i \in \mathbb{N}, \sigma_{[i]} \models q' \text{ and } \forall j < i, \sigma_{[j]} \models q$$

until

$$\sigma \models q \mathbf{R} q' \quad \Leftrightarrow \quad \forall j \geq 0, \text{ if } \forall i < j, \sigma_{[i]} \not\models q \text{ then } \sigma_{[j]} \models q'$$

releases

Given a Kripke structure $(\hat{\mathcal{X}}, \mathcal{X}_{init}, \mathcal{N}, \mathcal{A}, \mathcal{L})$

CTL has **state formulas** and **path formulas**

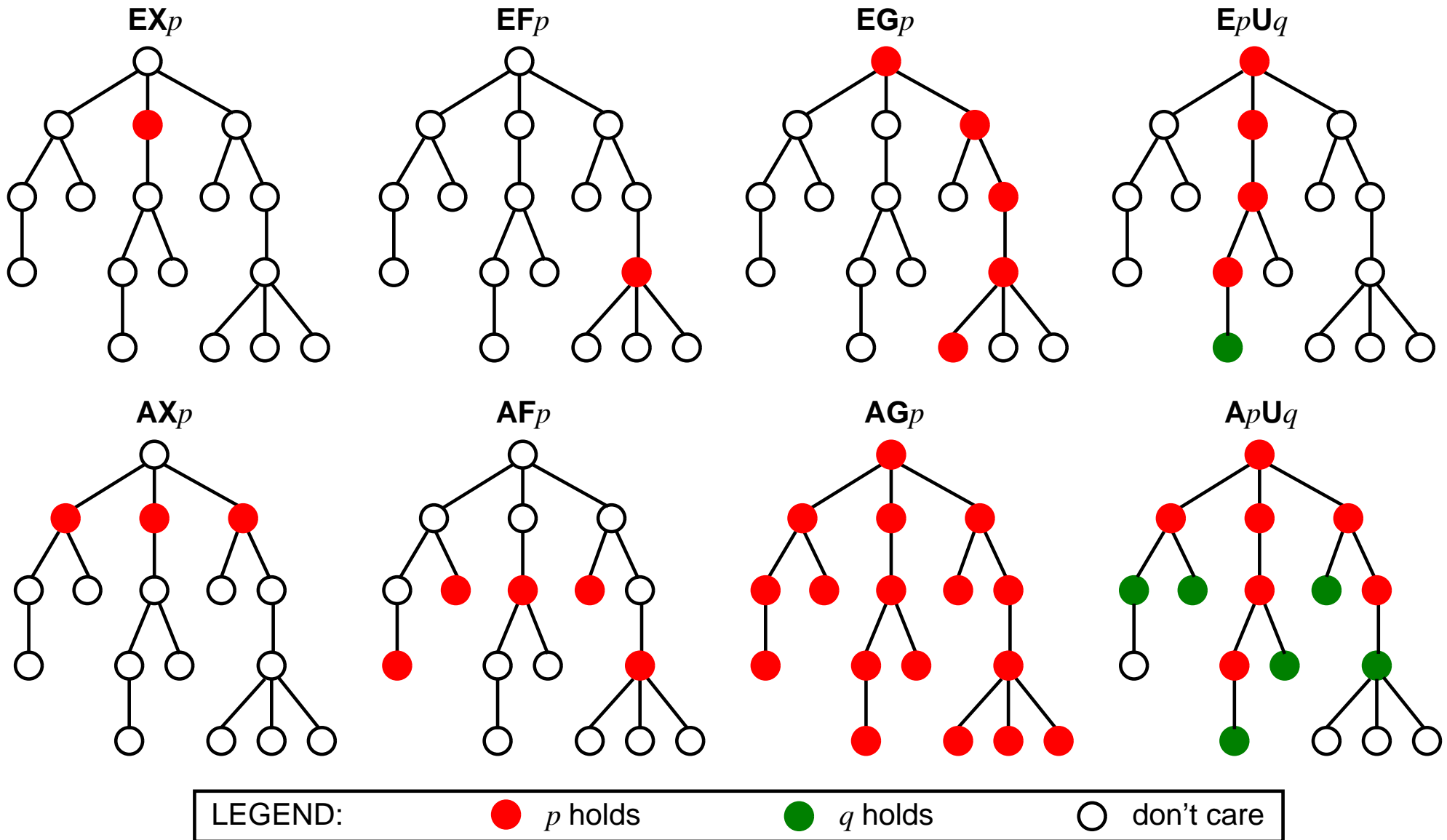
- State formulas:
 - if $a \in \mathcal{A}$, a is a **state formula** (a is an **atomic proposition**, true or false in each state)
 - if p and p' are **state formulas**, $\neg p$, $p \vee p'$, $p \wedge p'$ are **state formulas**
 - if q is a **path formula**, Eq , Aq are **state formulas**
- Path formulas:
 - if p and p' are **state formulas**, Xp , Fp , Gp , pUp' , pRp' are **path formulas**
 - **Note:** unlike CTL*, a state formula is **not** also a path formula

In CTL, operators occur in pairs:

- a **path quantifier**, E or A, must always immediately precede a **temporal operator**, X, F, G, U, R

Of course, CTL expressions can be **nested**: $p \vee E\neg pU(\neg p \wedge AXp)$

A CTL formula p identifies a set of model states (those satisfying p)



EX, EU, and EG form a **complete set** of CTL operators, since:

- $AXp = \neg EX\neg p$
- $EFp = EtrueUp$
- $AGp = \neg EF\neg p$
- $Eprq = \neg A\neg pU\neg q$
- $AFp = \neg EG\neg p$
- $ApUq = \neg(E\neg qU\neg p \wedge \neg q) \wedge \neg EG\neg q$
- $Aprq = \neg E\neg pU\neg q$

The system always reaches a stable state and remains in stable states after an initial startup period

- $initial \Rightarrow AF (AG \textit{ stable})$ or $initial \Rightarrow AF (A \textit{ stable U shutdown})$

At any point in the execution, it is possible to return to a reset state

- $AG EF \textit{ reset}$

If a process asks access to the critical region, it eventually obtains it

- $AG \textit{ request_critical} \Rightarrow AF \textit{ access_critical}$

Assume a Kripke structure $(\hat{\mathcal{X}}, \mathcal{X}_{init}, \mathcal{N}, \mathcal{A}, \mathcal{L})$

- $\hat{\mathcal{X}}$ set of (potential) states
- $\mathcal{X}_{init} \subseteq \hat{\mathcal{X}}$ set of initial states
- $\mathcal{N} : \hat{\mathcal{X}} \rightarrow 2^{\hat{\mathcal{X}}}$ next-state function
- \mathcal{A} set of atomic propositions
- $\mathcal{L} : \hat{\mathcal{X}} \rightarrow 2^{\mathcal{A}}$ labeling function

LTL formulas are of the type $A p$ where p is a **restricted path formula (RPF)**, i.e., without E or A

- if $a \in \mathcal{A}$, a is a RPF
- if p and p' are RPF, $\neg p$, $p \vee p'$, $p \wedge p'$, Xp , Fp , Gp , pUp' , pRp' are RPFs

The only path quantifier, **A**, is in the outermost position and can be omitted without ambiguity

We look for properties that hold on **all paths** starting from a state

The branching structure of the computation tree is not explicitly considered

CTL and LTL are not comparable

- An LTL formula not expressible in CTL:

$AFGp$

Try $AFAGp$ or $AFEGp$ and see why they are different from $AFGp$

- A CTL formula not expressible in LTL:

$AGEFq$

Try $AGFq$ and see why it is different from $AGEFq$

CTL and LTL are both strictly less expressive than CTL*

- A CTL* formula not expressible in CTL or in LTL:

$AFGp \vee AGEFq$

The EX algorithm for CTL (explicit version)

An algorithm to **label** all states that satisfy EXp

We assume that all states satisfying p have been correctly labeled already

LabelEX(p) is

1 $\mathcal{Y} \leftarrow \{\mathbf{i} \in \mathcal{X}_{reach} : p \in labels(\mathbf{i})\};$

initialize \mathcal{Y} with the states satisfying p

2 while $\mathcal{Y} \neq \emptyset$ do

3 pick and remove a state \mathbf{j} from \mathcal{Y} ;

4 for each $\mathbf{i} \in \mathcal{N}^{-1}(\mathbf{j})$ do

state \mathbf{i} can transition to state \mathbf{j}

5 $labels(\mathbf{i}) \leftarrow labels(\mathbf{i}) \cup \{EXp\};$

If we have $\mathcal{X}_{reach} \subseteq \hat{\mathcal{X}}$, we can assume $\mathcal{N} : \mathcal{X}_{reach} \rightarrow 2^{\mathcal{X}_{reach}}$ instead of $\mathcal{N} : \hat{\mathcal{X}} \rightarrow 2^{\hat{\mathcal{X}}}$

The EU algorithm for CTL (explicit version)

An algorithm to **label** all states that satisfy $EpUq$

We assume that all states satisfying p and all states satisfying q have been correctly labeled already

LabelEU(p, q) is

1 $\mathcal{Y} \leftarrow \{\mathbf{i} \in \mathcal{X}_{reach} : q \in labels(\mathbf{i})\};$

initialize \mathcal{Y} with the states satisfying q

2 for each $\mathbf{i} \in \mathcal{Y}$ do

3 $labels(\mathbf{i}) \leftarrow labels(\mathbf{i}) \cup \{EpUq\};$

4 while $\mathcal{Y} \neq \emptyset$ do

5 pick and remove a state \mathbf{j} from \mathcal{Y} ;

6 for each $\mathbf{i} \in \mathcal{N}^{-1}(\mathbf{j})$ do

state \mathbf{i} can transition to state \mathbf{j}

7 if $EpUq \notin labels(\mathbf{i})$ and $p \in labels(\mathbf{i})$ then

8 $labels(\mathbf{i}) \leftarrow labels(\mathbf{i}) \cup \{EpUq\};$

9 $\mathcal{Y} \leftarrow \mathcal{Y} \cup \{\mathbf{i}\};$

The EG algorithm for CTL (explicit version)

An algorithm to **label** all states that satisfy EGp

We assume that all states satisfying p have been correctly labeled already

The algorithm relies on finding the **(nontrivial) strongly connected components (SCCs)** of a graph

LabelEG(p) is

- 1 $\mathcal{Y} \leftarrow \{\mathbf{i} \in \mathcal{X}_{reach} : p \in labels(\mathbf{i})\};$ *initialize \mathcal{Y} with the states satisfying p*
- 2 build the set \mathcal{C} of SCCs in the subgraph of \mathcal{N} induced by \mathcal{Y} ;
- 3 $\mathcal{W} \leftarrow \{\mathbf{i} : \mathbf{i} \text{ is a state in a SCC of } \mathcal{C}\};$
- 4 for each $\mathbf{i} \in \mathcal{W}$ do
- 5 $labels(\mathbf{i}) \leftarrow labels(\mathbf{i}) \cup \{EGp\};$
- 6 while $\mathcal{W} \neq \emptyset$ do
- 7 pick and remove a state \mathbf{j} from \mathcal{W} ;
- 8 for each $\mathbf{i} \in \mathcal{N}^{-1}(\mathbf{j})$ do *state \mathbf{i} can transition to state \mathbf{j}*
- 9 if $EGp \notin labels(\mathbf{i})$ and $p \in labels(\mathbf{i})$ then
- 10 $labels(\mathbf{i}) \leftarrow labels(\mathbf{i}) \cup \{EGp\};$
- 11 $\mathcal{W} \leftarrow \mathcal{W} \cup \{\mathbf{i}\};$

All sets of states and relations over sets of states are encoded using DDs

An algorithm to build the DD encoding the set of states that satisfy EXp

Assume that the DD encoding the set \mathcal{P} of states satisfying p has been built already

BuildEXsymbolic(\mathcal{P}) is

1 return *RelationalProduct*($\mathcal{P}, \mathcal{N}^{-1}$); *perform one backward step in the transition relation*

Where

- \mathcal{N}^{-1} is the **inverse** or **backwards** transition relation:

$$\mathbf{i} \in \mathcal{N}^{-1}(\mathbf{j}) \iff \mathbf{j} \in \mathcal{N}(\mathbf{i})$$

- given a relation $\mathcal{R} : \mathcal{A} \rightarrow 2^{\mathcal{B}}$ and a set $\mathcal{A}' \subseteq \mathcal{A}$:

$$\textit{RelationalProduct}(\mathcal{A}', \mathcal{R}) = \mathcal{R}(\mathcal{A}') = \bigcup_{i \in \mathcal{A}'} \mathcal{R}(i) \subseteq \mathcal{B}$$

Two algorithms to build the DD encoding the set of states that satisfy $EpUq$

Assume that the DDs encoding the sets \mathcal{P} and \mathcal{Q} of states satisfying p and q have been built already

BuildEUsymbolic(\mathcal{P}, \mathcal{Q}) is

```

1  $\mathcal{Y} \leftarrow \emptyset;$ 
2  $\mathcal{U} \leftarrow \mathcal{Q};$            initialize the unexplored set  $\mathcal{U}$  with the states satisfying  $q$ 
3 repeat
4    $\mathcal{Y} \leftarrow \text{Union}(\mathcal{Y}, \mathcal{U});$            currently known states satisfying  $EpUq$ 
5    $\mathcal{W} \leftarrow \text{RelationalProduct}(\mathcal{U}, \mathcal{N}^{-1});$    perform one backward step in the transition relation
6    $\mathcal{Z} \leftarrow \text{Intersection}(\mathcal{W}, \mathcal{P});$            discard the states that do not satisfy  $p$ 
7    $\mathcal{U} \leftarrow \text{Difference}(\mathcal{Z}, \mathcal{Y});$            discard the states that are not new
8 until  $\mathcal{U} = \emptyset;$ 
9 return  $\mathcal{Y};$ 

```

BuildEUsymbolicAll(\mathcal{P}, \mathcal{Q}) is

```

1  $\mathcal{Y} \leftarrow \mathcal{Q};$            initialize the currently known result with the states satisfying  $q$ 
2 repeat
3    $\mathcal{O} \leftarrow \mathcal{Y};$            save the old set of states
4    $\mathcal{W} \leftarrow \text{RelationalProduct}(\mathcal{Y}, \mathcal{N}^{-1});$    perform one backward step in the transition relation
5    $\mathcal{Z} \leftarrow \text{Intersection}(\mathcal{W}, \mathcal{P});$            discard the states that do not satisfy  $p$ 
6    $\mathcal{Y} \leftarrow \text{Union}(\mathcal{Z}, \mathcal{Y});$            add to the currently known result
7 until  $\mathcal{O} = \mathcal{Y};$ 
8 return  $\mathcal{Y};$ 

```

An algorithm to build the DD encoding the set of states that satisfy EGp

Assume that the DDs encoding the set \mathcal{P} of states satisfying p has been built already

BuildEGsymbolic(\mathcal{P}) is

1 $\mathcal{Y} \leftarrow \mathcal{P};$	<i>initialize \mathcal{Y} with the states satisfying p</i>
2 repeat	
3 $\mathcal{O} \leftarrow \mathcal{Y};$	<i>save the old set of states</i>
4 $\mathcal{W} \leftarrow \text{RelationalProduct}(\mathcal{Y}, \mathcal{N}^{-1});$	<i>perform one backward step in the transition relation</i>
5 $\mathcal{Y} \leftarrow \text{Intersection}(\mathcal{Y}, \mathcal{W});$	
6 until $\mathcal{O} = \mathcal{Y};$	
7 return $\mathcal{Y};$	

This algorithm starts with a larger set of states and **reduces** it

This algorithm is not based on finding the strongly connected components of \mathcal{N}

Assume a Kripke structure M , a state s , and an LTL property Ap , where p is a restricted path formula

Instead of checking whether $M, s \models Ap$ we check the equivalent condition $M, s \models \neg E \neg p$

In other words, we only need to be able to check $M, s \models Eq$ where q is a restricted path formula

If $M, s \models Eq$ and $q = \neg p$ then it is not true that $M, s \models Ap$ and vice versa

We can restrict ourselves to restricted path formulas containing the temporal operators X and U , since

- $Fp \equiv true \ U \ p$ $true$ is an atomic proposition that holds in any state
- $Gp \equiv \neg F \neg p$
- $p \ R \ q \equiv \neg(\neg p \ U \ \neg q)$

Intersect the behaviors corresponding to the restricted path formula p with those of the Kripke structure

Define $CL(p)$ to be the closure of a restricted path formula p :

$$p \in CL(p)$$

$$q \in CL(p) \quad \Rightarrow \quad \neg q \in CL(p) \quad \text{we identify } \neg\neg q \text{ with } q$$

$$q \vee r \in CL(p) \quad \Rightarrow \quad q \in CL(p) \quad \text{and} \quad r \in CL(p)$$

$$\mathsf{X}q \in CL(p) \quad \Rightarrow \quad q \in CL(p)$$

$$\neg\mathsf{X}q \in CL(p) \quad \Rightarrow \quad \mathsf{X}\neg q \in CL(p)$$

$$q \mathsf{U} r \in CL(p) \quad \Rightarrow \quad q \in CL(p) \quad \text{and} \quad r \in CL(p) \quad \text{and} \quad \mathsf{X}(q \mathsf{U} r) \in CL(p)$$

Intuitively, $CL(p)$ contains all the formulas whose truth value affects the truth value of p

The size of $CL(p)$ is linear in the size of the restricted path formula p

Define a **tableau** for p : a graph where there is a path corresponding to p iff p is satisfiable

The set \mathcal{V}_T of tableau **vertices** contains each **maximal consistent subset** K of $CL(p)$, satisfying:

$$\forall q \in CL(p) \quad q \in K \quad \Leftrightarrow \quad \neg q \notin K$$

$$\forall q \vee r \in CL(p) \quad q \vee r \in K \quad \Leftrightarrow \quad q \in K \text{ or } r \in K$$

$$\forall \neg Xq \in CL(p) \quad \neg Xq \in K \quad \Leftrightarrow \quad X\neg q \in K$$

$$\forall q \cup r \in CL(p) \quad q \cup r \in K \quad \Leftrightarrow \quad r \in K \text{ or } q \in K \text{ and } X(q \cup r) \in K$$

The set \mathcal{N}_T of tableau **edges** contains the pairs (K_A, K_B) that satisfy:

$$\forall Xq \in CL(p) \quad Xq \in K_B \quad \Leftrightarrow \quad q \in K_C$$

The size of the tableau is **exponential in the size of the restricted path formula p**

Then, build the **product graph** of the Kripke structure $(\hat{\mathcal{X}}, \mathcal{X}_{init}, \mathcal{N}, \mathcal{A}, \mathcal{L})$ and the tableau $(\mathcal{V}_T, \mathcal{N}_T)$

Its size is **exponential in the size of the formula** but only **linear in the size of the Kripke structure**

The **vertices** of the product graph are the pairs of the form $B = (\mathbf{s}_B, K_B)$ where:

- $\mathbf{s}_B \in \hat{\mathcal{X}}$ (or $\mathbf{s}_B \in \mathcal{X}_{reach}$)
- $K_B \in \mathcal{V}_T$
- $\forall a \in \mathcal{A}, a \in \mathcal{L}(\mathbf{s}_B) \Leftrightarrow a \in K_B$

The **edges** are such that there is an edge from $B = (\mathbf{s}_B, K_B)$ to $C = (\mathbf{s}_C, K_C)$ iff:

- $\mathbf{s}_C \in \mathcal{N}(\mathbf{s}_B)$ in the Kripke structure
- $K_C \in \mathcal{N}_T(K_B)$ in the tableau graph

In practice, we can build the product graph directly, without explicitly building the tableau graph first

An **eventuality sequence** is an **infinite** path σ in the product graph such that

- if σ contains $B = (\mathbf{s}_B, K_B)$ in position m and $q \cup r \in K_B$,
- then σ contains $C = (\mathbf{s}_C, K_C)$ in position $n \geq m$ such that $r \in K_C$

Theorem: $M, \mathbf{s} \models \mathbf{E}p$

\Leftrightarrow there is an eventuality sequence starting at a vertex (\mathbf{s}, K) s.t. $p \in K$

A non-trivial SCC \mathcal{F} of the product graph is **self-fulfilling** if

$$\forall B = (\mathbf{s}_B, K_B) \in \mathcal{F}, \quad \forall q \cup r \in K_B, \quad \exists C = (\mathbf{s}_C, K_C) \in \mathcal{F}, \quad r \in K_C$$

Theorem: There is an eventuality sequence starting at vertex (\mathbf{s}, K)

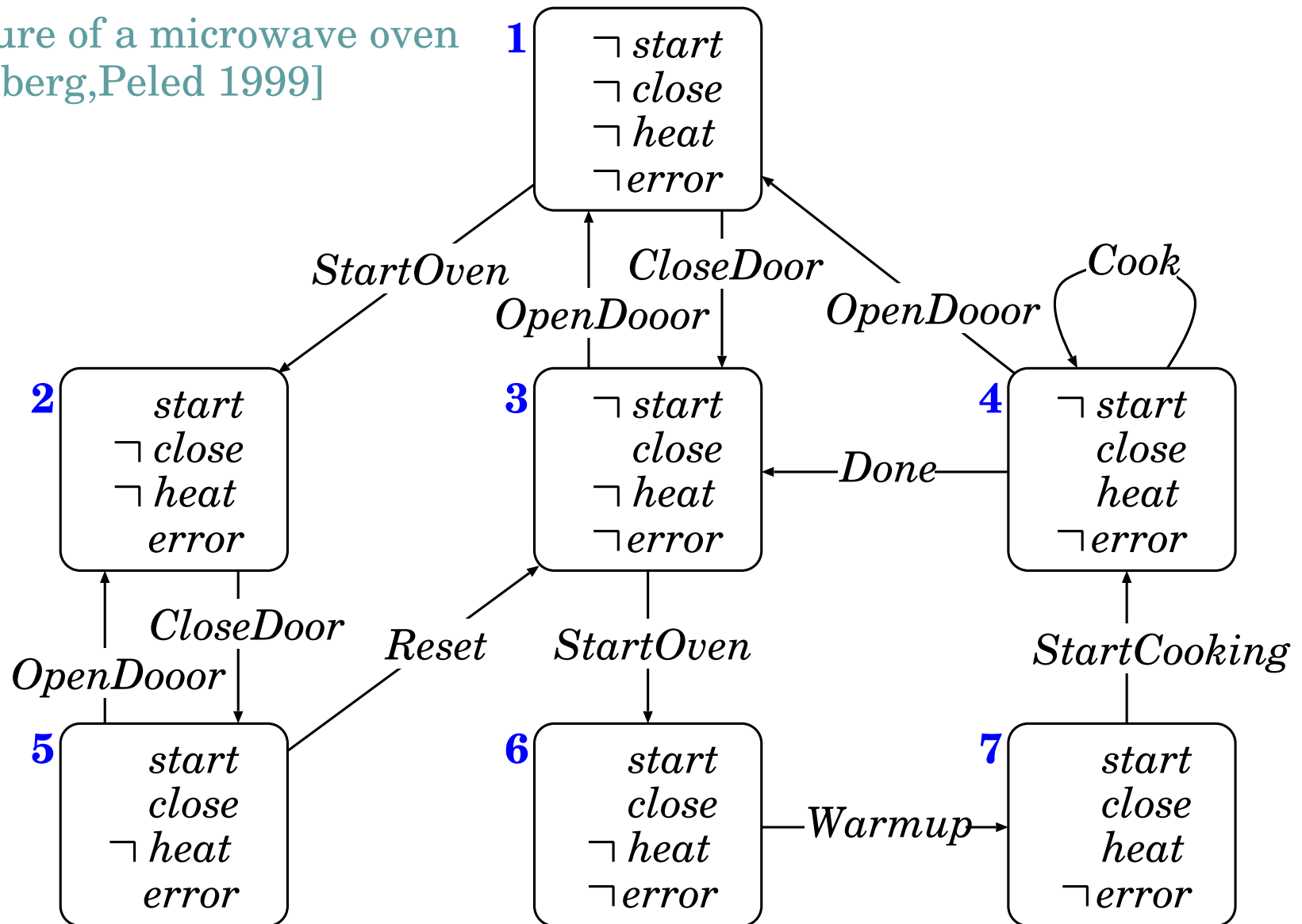
\Leftrightarrow the product graph contains a path from (\mathbf{s}, K) to a self-fulfilling SCC

Corollary: $M, \mathbf{s} \models \mathbf{E}p$

\Leftrightarrow there is a vertex (\mathbf{s}, K) s.t. $p \in K$ and there is a path from (\mathbf{s}, K) to a self-fulfilling SCC

Example of tableau-based LTL model checking

Kripke structure of a microwave oven
[Clarke, Grumberg, Peled 1999]



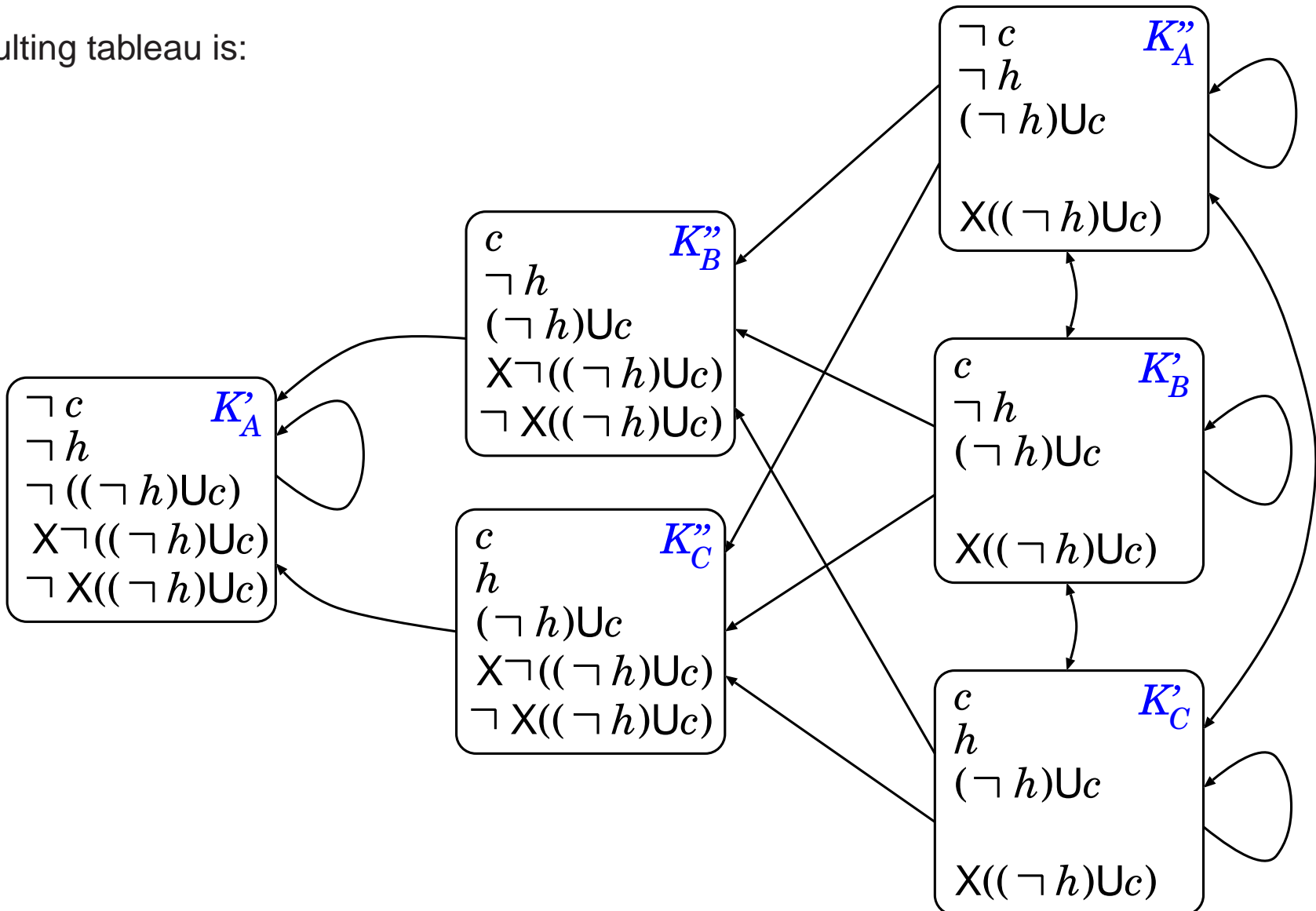
Verify that the LTL formula $A((\neg heat) U close)$ always holds
by checking that $E\neg((\neg heat) U close)$ is not satisfied by any state

Example of tableau-based LTL model checking (cont.)

Compute the closure of $p \equiv \neg((\neg heat) \cup close)$ (let h and c stand for $heat$ and $close$):

$$CL(p) = \{\neg((\neg h) \cup c), (\neg h) \cup c, \neg h, c, X((\neg h) \cup c), h, \neg c, \neg X((\neg h) \cup c), X\neg((\neg h) \cup c)\}$$

The resulting tableau is:



Example of tableau-based LTL model checking (cont.)

Build the vertices (s, K) of the product graph:

$$\text{vertices with } s = 1: \quad (1, K'_A) \equiv (1, \{\neg c, \neg h, \neg((\neg h)Uc), \neg X((\neg h)Uc), X\neg((\neg h)Uc)\})$$

$$(1, K''_A) \equiv (1, \{\neg c, \neg h, (\neg h)Uc, X((\neg h)Uc)\})$$

$$\text{vertices with } s = 2: \quad (2, K'_A) \equiv (2, \{\neg c, \neg h, \neg((\neg h)Uc), \neg X((\neg h)Uc), X\neg((\neg h)Uc)\})$$

$$(2, K''_A) \equiv (2, \{\neg c, \neg h, (\neg h)Uc, X((\neg h)Uc)\})$$

$$\text{vertices with } s = 3: \quad (3, K'_B) \equiv (3, \{c, \neg h, (\neg h)Uc, X((\neg h)Uc)\})$$

$$(3, K''_B) \equiv (3, \{c, \neg h, (\neg h)Uc, \neg X((\neg h)Uc), X\neg((\neg h)Uc)\})$$

$$\text{vertices with } s = 4: \quad (4, K'_C) \equiv (4, \{c, h, (\neg h)Uc, X((\neg h)Uc)\})$$

$$(4, K''_C) \equiv (4, \{c, h, (\neg h)Uc, \neg X((\neg h)Uc), X\neg((\neg h)Uc)\})$$

$$\text{vertices with } s = 5: \quad (5, K'_B) \equiv (5, \{c, \neg h, (\neg h)Uc, X((\neg h)Uc)\})$$

$$(5, K''_B) \equiv (5, \{c, \neg h, (\neg h)Uc, \neg X((\neg h)Uc), X\neg((\neg h)Uc)\})$$

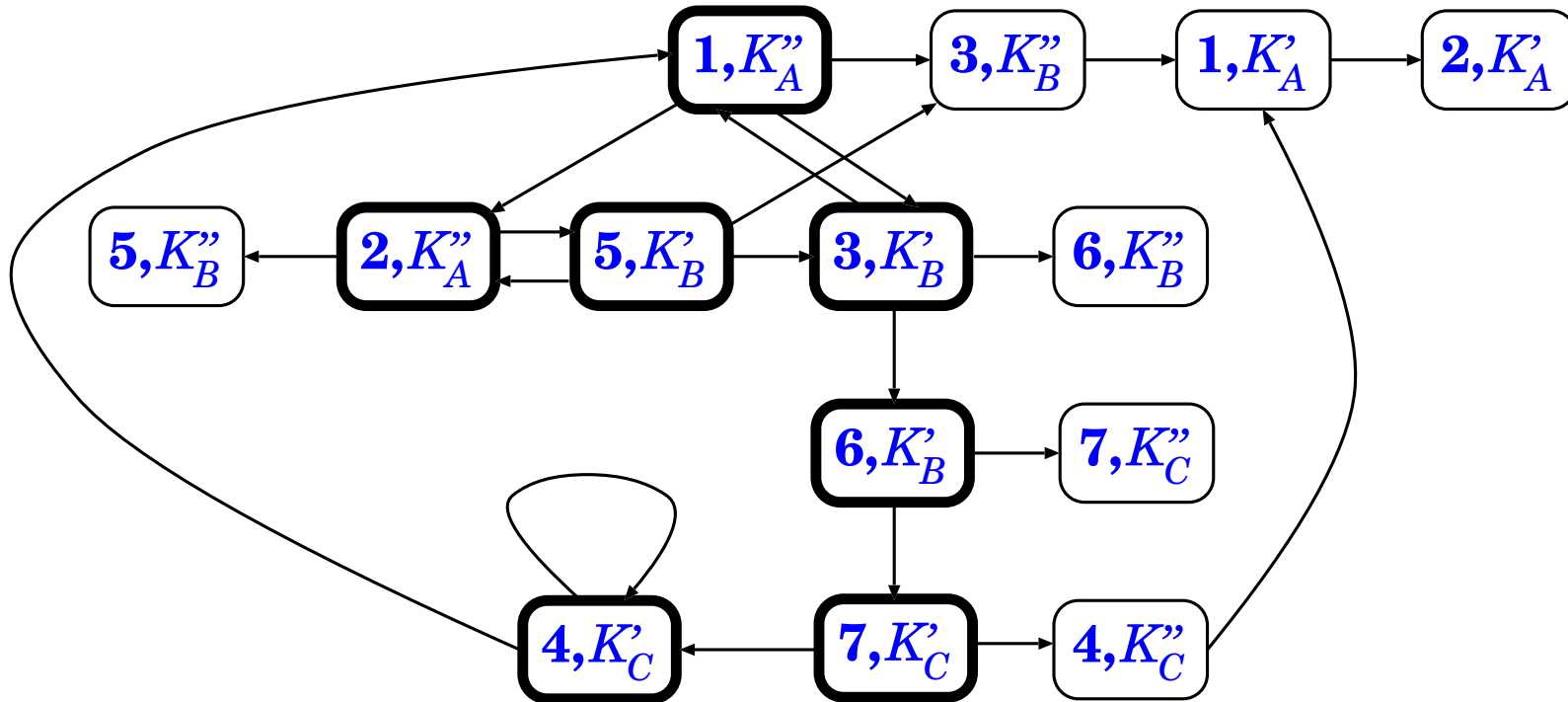
$$\text{vertices with } s = 6: \quad (6, K'_B) \equiv (6, \{c, \neg h, (\neg h)Uc, X((\neg h)Uc)\})$$

$$(6, K''_B) \equiv (6, \{c, \neg h, (\neg h)Uc, \neg X((\neg h)Uc), X\neg((\neg h)Uc)\})$$

$$\text{vertices with } s = 7: \quad (7, K'_C) \equiv (7, \{c, h, (\neg h)Uc, X((\neg h)Uc)\})$$

$$(7, K''_C) \equiv (7, \{c, h, (\neg h)Uc, \neg X((\neg h)Uc), X\neg((\neg h)Uc)\})$$

Build the product graph by adding the edges between vertices:



Then, $\mathbf{Ep} \equiv \mathbf{E}\neg((\neg\mathit{heat}) \mathbf{U} \mathit{close})$ is satisfied if there is a vertex (s, K) such that

1: $\neg((\neg\mathit{heat}) \mathbf{U} \mathit{close}) \in K$

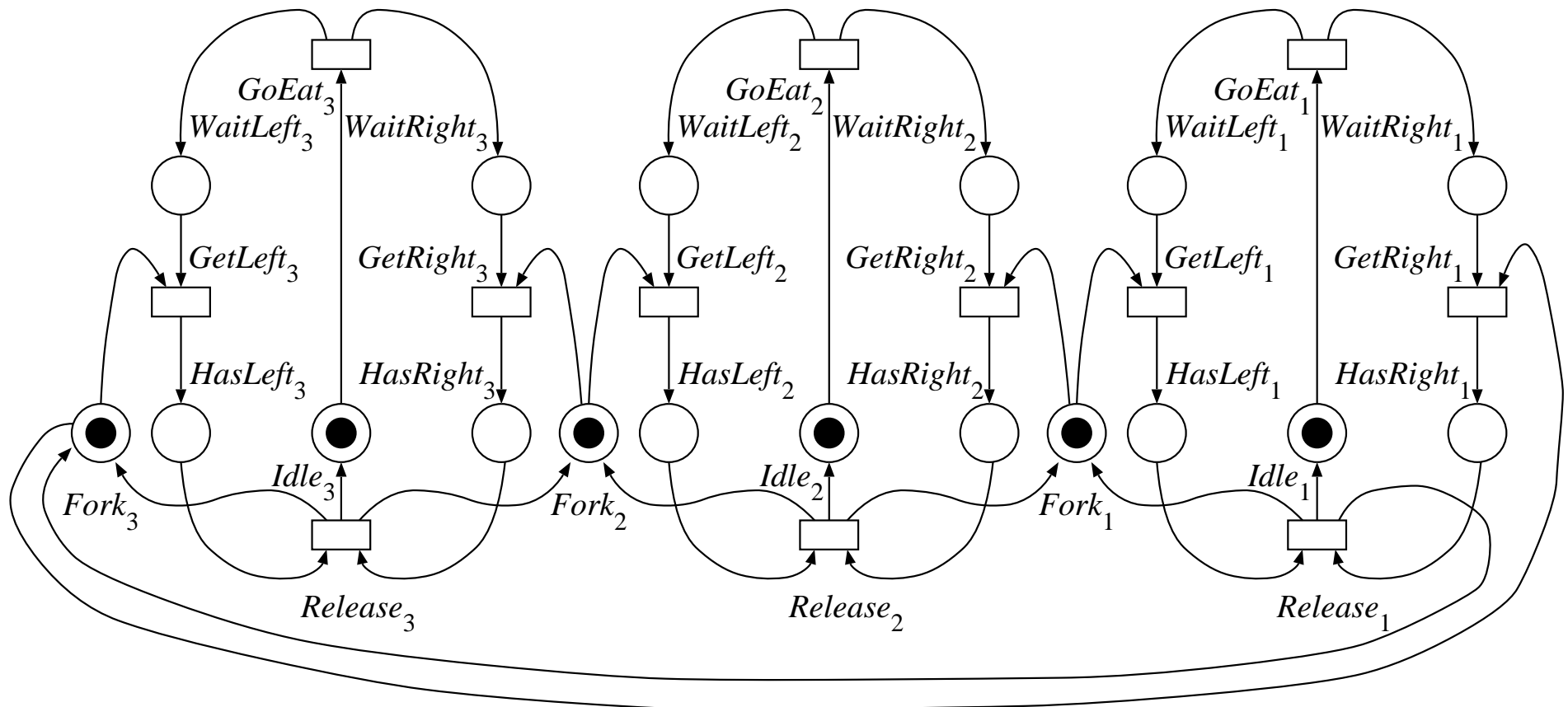
only K'_A contains $\neg((\neg h) \mathbf{U} c)$ and K'_A appears only in vertices $(1, K'_A)$ and $(2, K'_A)$

2: there is a path from (s, K) to a self-fulfilling SCC

neither $(1, K'_A)$ nor $(2, K'_A)$ reach an SCC

Thus, $\neg((\neg\mathit{heat}) \mathbf{U} \mathit{close})$ is never satisfied, i.e., $(\neg\mathit{heat}) \mathbf{U} \mathit{close}$ is always satisfied

Example: the dining philosophers (Petri net)



N subnets connected in a circular fashion

$N = 3$ in the figure

Example: the dining philosophers (SMART code)

```

spn phils(int N) := {
  for (int i in {0..N-1}) {
    place Idle[i], WaitL[i], WaitR[i], HasL[i], HasR[i], Fork[i];
    partition(i+1:Idle[i]:WaitL[i]:WaitR[i]:HasL[i]:HasR[i]:Fork[i]);
    trans GoEat[i], GetL[i], GetR[i], Release[i];
    init(Idle[i]:1, Fork[i]:1);
  }
  for (int i in {0..N-1}) {
    arcs(Idle[i]:GoEat[i], GoEat[i]:WaitL[i], GoEat[i]:WaitR[i],
        WaitL[i]:GetL[i], Fork[i]:GetL[i], GetL[i]:HasL[i],
        WaitR[i]:GetR[i], Fork[mod(i+1, N)]:GetR[i], GetR[i]:HasR[i],
        HasL[i]:Release[i], HasR[i]:Release[i], Release[i]:Idle[i],
        Release[i]:Fork[i], Release[i]:Fork[mod(i+1, N)]);
  }
  bigint num := card(reachable);
  stateset g := EF(initialstate);          bigint numg := card(g);
  stateset b := difference(reachable,g);    bool out      := printset(b);
};
# StateStorage MDD_SATURATION
int N := read_int("number of philosophers"); print("N=",N,"\n");
print("Reachable states: ",phils(N).num,"\n");
print("Good states:      ",phils(N).numg,"\n");
print("The bad states are:"); phils(N).out;

```

Example: the dining philosophers (results)

Reading input.

N=50

Reachable states: 22,291,846,172,619,859,445,381,409,012,498

Good states: 22,291,846,172,619,859,445,381,409,012,496

The bad states are:

State 0 : { WaitR[0]:1 HasL[0]:1 WaitR[1]:1 HasL[1]:1 WaitR[2]:1 HasL[2]:1 WaitL[2]:1

State 1 : { WaitL[0]:1 HasR[0]:1 WaitL[1]:1 HasR[1]:1 WaitL[2]:1 HasR[2]:1 WaitR[2]:1

true

Done.