Verification – Lecture 8
Progress under Justice

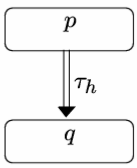
$$p \Rightarrow \Diamond q$$

Bernd Finkbeiner – Sven Schewe
Rayna Dimitrova – Lars Kuhtz – Anne Proetzsch

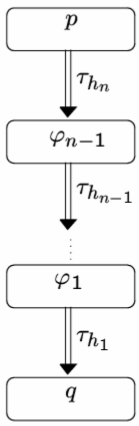
Wintersemester 2007/2008

Overview: 3 Rules

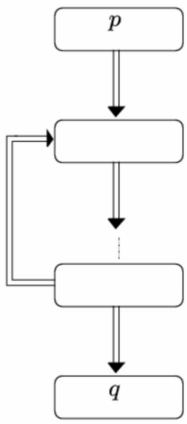
Review



1. Rule RESP-J
   (single-step response under justice)



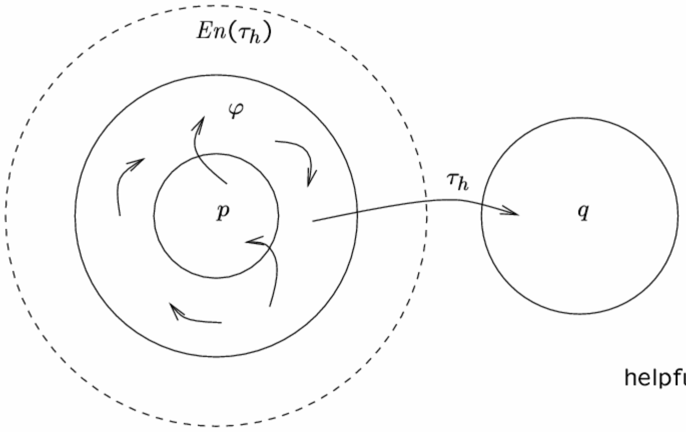
2. Rule CHAIN-J
   (chain rule under justice)



3. Rule WELL-J
   (well-founded response under justice)

Single-Step Rule (Motivation)

Review



helpful transition τ_h

Justice requirement: it is not the case that a just transition is continuously enabled but never taken.

Single-Step Rule

Review

For assertions p, q, φ, and transition $\tau_h \in \mathcal{J}$,

$$\begin{array}{ll} \text{J1.} & p \rightarrow q \vee \varphi \\ \text{J2.} & \{\varphi\}\ \mathcal{T}\ \{q \vee \varphi\} \\ \text{J3.} & \{\varphi\}\ \tau_h\ \{q\} \\ \text{J4.} & \varphi \rightarrow En(\tau_h) \\ \hline & p \Rightarrow \Diamond q \end{array}$$

Useful Rules

Review

- Monotonicity:

$$\frac{p \Rightarrow q \qquad q \Rightarrow \Diamond r \qquad r \Rightarrow t}{p \Rightarrow \Diamond t}$$

- Reflexivity:

$$p \Rightarrow \Diamond p$$

- Transitivity:

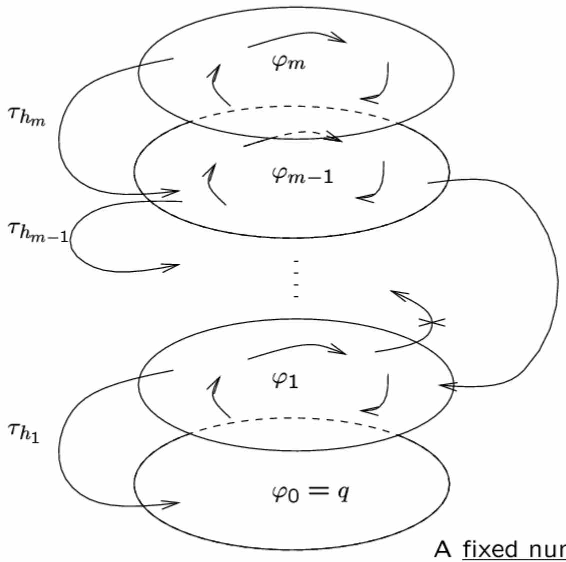
$$\frac{p \Rightarrow \Diamond q \qquad q \Rightarrow \Diamond r}{p \Rightarrow \Diamond r}$$

- Case analysis:

$$\frac{p \Rightarrow \Diamond r \qquad q \Rightarrow \Diamond r}{(p \vee q) \Rightarrow \Diamond r}$$

Chain Rule (Motivation)

Review



A <u>fixed number</u> of helpful transitions

Chain Rule

Review

For assertions p and $q = \varphi_0, \varphi_1, \ldots, \varphi_m$
and transitions $\tau_{h_1}, \ldots, \tau_{h_m} \in \mathcal{J}$

J1. $p \rightarrow \bigvee_{j=0}^{m} \varphi_j$

J2. $\{\varphi_i\}\mathcal{T}\left\{\bigvee_{j \leq i} \varphi_j\right\}$

J3. $\{\varphi_i\}\tau_{h_i}\left\{\bigvee_{j < i} \varphi_j\right\}$

J4. $\varphi_i \rightarrow En(\tau_{h_i})$

(J2–J4) for $i = 1, \ldots, m$

$p \Rightarrow \diamondsuit q$

Verification Diagrams

Review

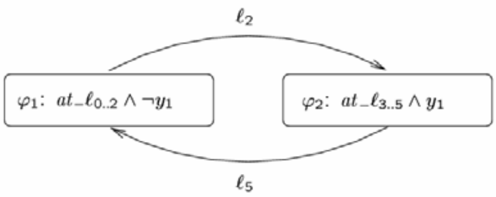
- INVARIANCE diagram valid for program MUX-PET1

local y_1, y_2: boolean where $y_1 = \text{F}, y_2 = \text{F}$
s : integer where $s = 1$

$P_1 ::$
ℓ_0 : loop forever do
- ℓ_1 : noncritical
- ℓ_2 : $(y_1, s) := (\text{T}, 1)$
- ℓ_3 : await $(\neg y_2) \vee (s = 2)$
- ℓ_4 : critical
- ℓ_5 : $y_1 := \text{F}$

||

$P_2 ::$
m_0 : loop forever do
- m_1 : noncritical
- m_2 : $(y_2, s) := (\text{T}, 2)$
- m_3 : await $(\neg y_1) \vee (s = 1)$
- m_4 : critical
- m_5 : $y_2 := \text{F}$



- Also,

(I2) $\underbrace{at\_\ell_0 \wedge \neg y_1 \wedge \cdots}_{\Theta} \rightarrow \underbrace{at\_\ell_{0..2} \wedge \neg y_1}_{\varphi_1} \vee \underbrace{\cdots}_{\varphi_2}$

(I1) $\underbrace{at\_\ell_{0..2} \wedge \neg y_1}_{\varphi_1} \rightarrow \underbrace{y_1 \leftrightarrow at\_\ell_{3..5}}_{q}$

$\underbrace{at\_\ell_{3..5} \wedge y_1}_{\varphi_2} \rightarrow \underbrace{y_1 \leftrightarrow at\_\ell_{3..6}}_{q}$

Therefore

$\Box(y_1 \leftrightarrow at\_\ell_{3..5})$

P-Valid Verification Diagrams

Review

Directed labeled graph with

Nodes – labeled by assertions

φ

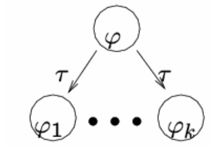
Edges – labeled by names of transitions

$\varphi_1 \overset{\tau}{\longrightarrow} \varphi_2$

Terminal Node ("goal") – no edges depart from it

φ_0

Verification conditions



$\Rightarrow \{\varphi\}\ \tau\ \{\varphi \vee \varphi_1 \vee \ldots \vee \varphi_k\}$

Definition: VD is P-valid iff all VCs associated with nodes in the diagram are P-state valid

Invariance Diagrams

Review

VDs with no terminal nodes (cycles OK)

Claim (invariance diagram):

A P-valid INVARIANCE diagram establishes that

$$\bigvee_{j=1}^{m} \varphi_j \Rightarrow \Box(\bigvee_{j=1}^{m} \varphi_j)$$

is P-valid.

If, in addition,

$$\text{(I1)} \quad \bigvee_{j=1}^{m} \varphi_j \rightarrow q$$

$$\text{(I2)} \quad \Theta \rightarrow \bigvee_{j=1}^{m} \varphi_j$$

are P-state valid, then

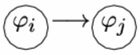
$\Box q$ is P-valid

Wait Diagrams

Review

VDs with nodes $\varphi_m, \ldots, \varphi_0$ such that:

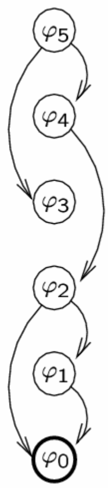
- weakly acyclic, i.e.,

  if 

  then $i \geq j$

- φ_0 is a terminal node



Proofs with Wait Diagrams

Review

A P-valid WAIT diagram establishes that

$$\bigvee_{j=0}^{m} \varphi_j \Rightarrow \varphi_m \mathcal{W} \varphi_{m-1} \cdots \varphi_1 \mathcal{W} \varphi_0$$

is P-valid.

If, <u>in addition</u>,

(N1) $p \rightarrow \bigvee_{j=0}^{m} \varphi_j$

(N2) $\varphi_i \rightarrow q_i$ for $i = 0, 1, \ldots, m$

are P-state valid, then

$$p \Rightarrow q_m \mathcal{W} q_{m-1} \cdots q_1 \mathcal{W} q_0$$

is P-valid.

Chain Diagrams

Review

<u>Edges:</u> labeled by transitions

single-lined ——————
(represents a regular transition)

double-lined ══════
(represents a helpful transition)

<u>Nodes:</u> labeled by assertions (φ_i)

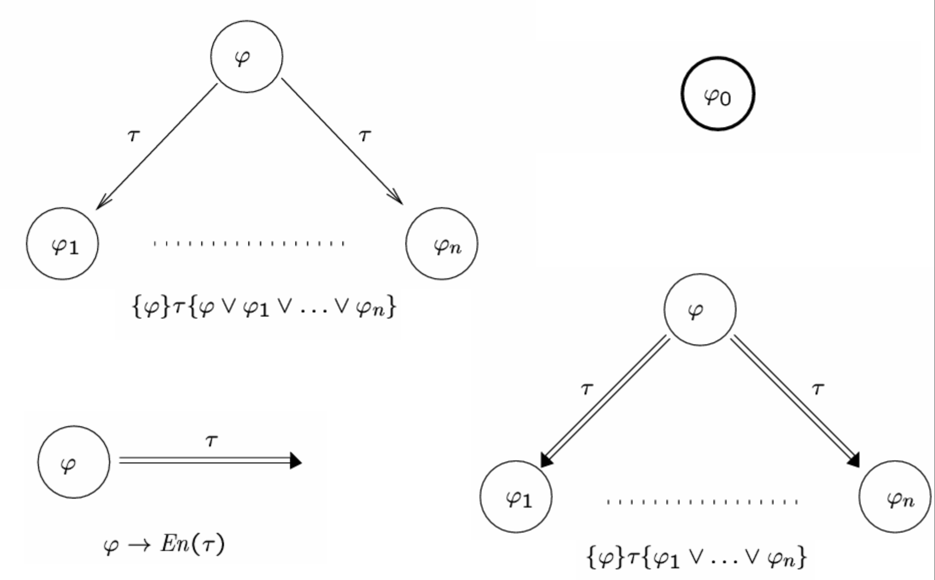
Terminal node (φ_0)

<u>well-formedness conditions:</u>

- weakly acyclic in $\longrightarrow$:
  if $(\varphi_i) \longrightarrow (\varphi_j)$ then $i \geq j$
- acyclic in $\Longrightarrow$:
  if $(\varphi_i) \Longrightarrow (\varphi_j)$ then $i > j$
- every nonterminal node has a double edge departing from it.

Verification Conditions

Review



Chain Diagram Validity

Review

A chain diagram is <u>P-valid</u>
if all the verification conditions associated with the diagram are P-valid.

<u>Claim:</u> A P-valid chain diagram establishes that

$$\bigvee_{j=0}^{m} \varphi_j \Rightarrow \Diamond \varphi_0$$

is P-valid.

With $\quad p \rightarrow \bigvee_{j=0}^{m} \varphi_j \quad$ and $\quad \varphi_0 \rightarrow q$,

we can conclude the P-validity of

$$\boxed{p \Rightarrow \Diamond q}$$

Example

Review

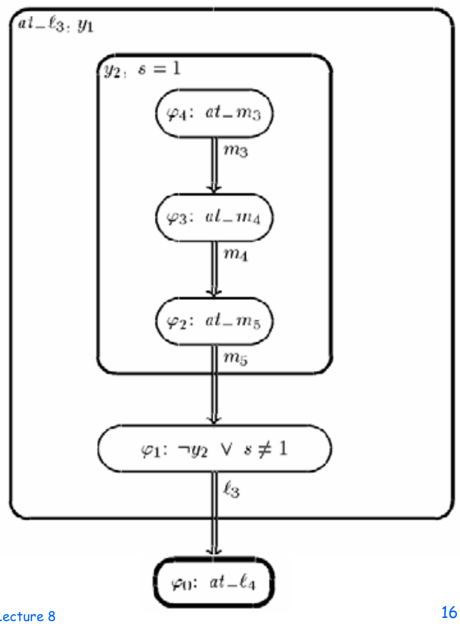
$$\boxed{at\_\ell_3 \Rightarrow \Diamond\, at\_\ell_4}$$

local y_1, y_2: boolean where $y_1 = \text{F}, y_2 = \text{F}$
$\quad s$: integer where $s = 1$

$P_1 ::$
ℓ_0 : loop forever do
- ℓ_1 : noncritical
- ℓ_2 : $(y_1,\ s) := (\text{T},\ 1)$
- ℓ_3 : await $(\neg y_2) \vee (s = 2)$
- ℓ_4 : critical
- ℓ_5 : $y_1 := \text{F}$

||

$P_2 ::$
m_0 : loop forever do
- m_1 : noncritical
- m_2 : $(y_2,\ s) := (\text{T},\ 2)$
- m_3 : await $(\neg y_1) \vee (s = 1)$
- m_4 : critical
- m_5 : $y_2 := \text{F}$



Program N

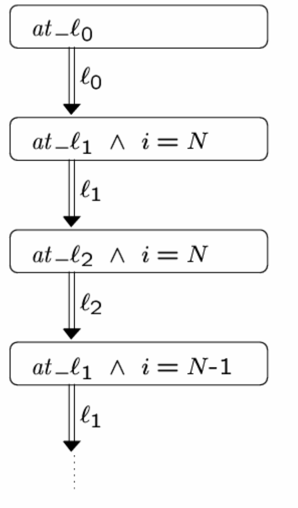
in N: integer where $N > 0$
local i: integer

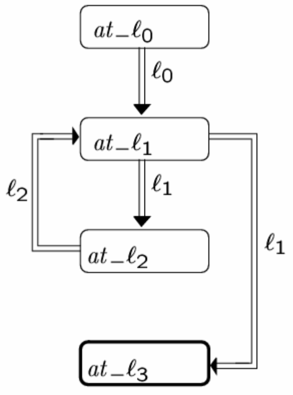
ℓ_0 : $i := N$
ℓ_1 : while $i > 0$ do
  ℓ_2 : $i = i - 1$
ℓ_3 :

We want to prove that for program N:

$$at\_\ell_0 \Rightarrow \Diamond\, at\_\ell_3$$

Attempts to use Chain Diagrams…





Well-Founded Domains

$(A, \succ)$

where A is a set and
$\succ$ is a well-founded order
(i.e., there does not exist an infinitely descending sequence $a_0 \succ a_1 \succ a_2 \ldots$)

Note: A well-founded order is transitive and irreflexive.

Examples:

$(\mathbb{N}, >)$ is well-founded:
$n > n\text{-}1 > n\text{-}2 > \ldots > 0$

$(\mathbb{Z}, >)$ is not well-founded:
$n > n\text{-}1 > \ldots > 0 > -1 > -2 \ldots$

$(\mathbb{Z}, |{>}|)$ with $x \,|{>}|\, y$ iff $|x| > |y|$ is well-founded:
$-7 \;|{>}|\; -3 \;|{>}|\; 2 \;|{>}|\; -1 \;|{>}|\; 0$

Lexicographic Product

Well-founded domains $(A_1, \succ_1)$ and $(A_2, \succ_2)$ can be combined into their

lexicographic product $(A_1 \times A_2, \succ)$

where

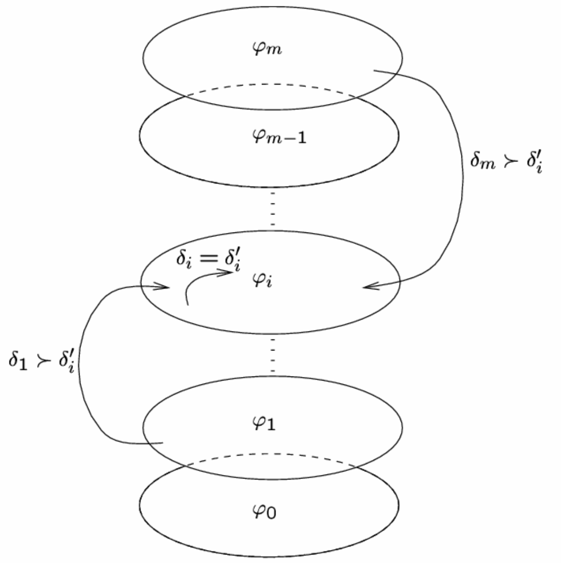
$(n_1, n_2) \succ (m_1, m_2)$

iff

$n_1 \succ m_1$ or $(n_1 = m_1$ and $n_2 \succ m_2)$.

$(A_1 \times A_2, \succ)$ is also a well-founded domain.

Rule Well-J (Motivation)



Rule WELL-J

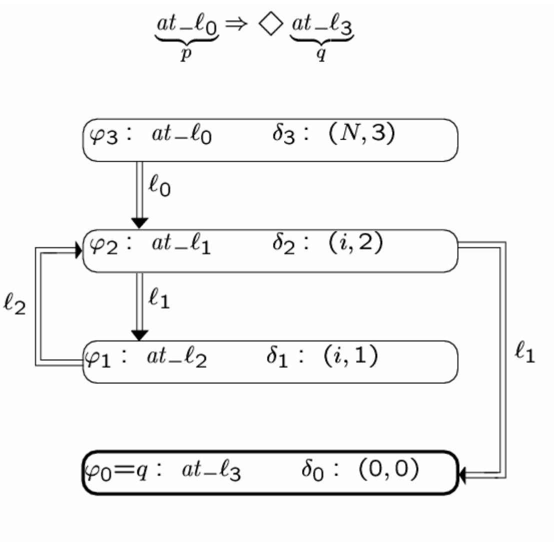
For assertions p and $q = \varphi_0, \varphi_1, \ldots, \varphi_m$,
transitions $\tau_1, \ldots, \tau_m \in \mathcal{J}$,
a well-founded domain $(\mathcal{A}, \succ)$, and
ranking functions $\delta_0, \ldots, \delta_m \colon \Sigma \mapsto \mathcal{A}$

$$\text{JW1.} \quad p \rightarrow \bigvee_{j=0}^{m} \varphi_j$$

$$\left.\begin{array}{l}
\text{JW2.} \quad \rho_\tau \wedge \varphi_i \rightarrow \begin{bmatrix} \bigvee_{j=0}^{m} (\varphi'_j \wedge \delta_i \succ \delta'_j) \\ \vee \ (\varphi'_i \wedge \delta_i = \delta'_i) \end{bmatrix} \\
\qquad\qquad\qquad\qquad \text{for every } \tau \in \mathcal{T} \\
\text{JW3.} \quad \rho_{\tau_i} \wedge \varphi_i \rightarrow \bigvee_{j=0}^{m} (\varphi'_j \wedge \delta_i \succ \delta'_j) \\
\text{JW4.} \quad \varphi_i \rightarrow En(\tau_i)
\end{array}\right\} \text{ for } i = 1, \ldots, m$$

$$p \Rightarrow \Diamond q$$

Rank Diagrams



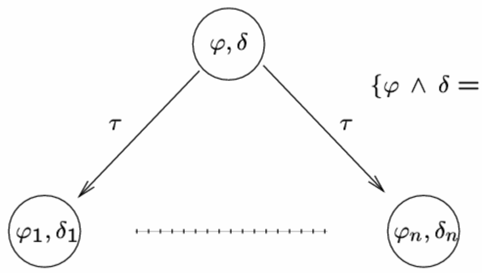
Nodes:

labeled by assertions and ranking functions

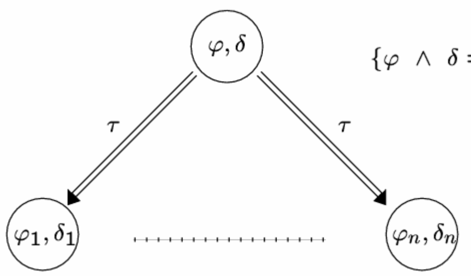
Well-formedness constraint:

Every node $\varphi_i,\ i > 0$, has a double edge departing from it.

Verification Conditions



$$\{\varphi \wedge \delta = u\} \quad \tau \quad \{(\varphi \wedge u \succeq \delta) \vee (\varphi_1 \wedge u \succ \delta_1) \vee \ldots \vee (\varphi_n \wedge u \succ \delta_n)\}$$



$$\{\varphi \wedge \delta = u\} \quad \tau \quad \{(\varphi_1 \wedge u \succ \delta_1) \vee \ldots \vee (\varphi_n \wedge u \succ \delta_n)\}$$

$$\varphi \rightarrow En(\tau)$$

Example: Program UP-DOWN

local x, y: **integer where** $x = y = 0$

$$P_1 :: \left[\begin{array}{l} \ell_0\text{: } \textbf{while } x = 0 \textbf{ do} \\ \quad \ell_1\text{: } y := y + 1 \\ \ell_2\text{: } \textbf{while } y > 0 \textbf{ do} \\ \quad \ell_3\text{: } y := y - 1 \\ \ell_4\text{:} \end{array}\right] \quad || \quad P_2 :: \left[\begin{array}{l} m_0\text{: } x := 1 \\ m_1\text{:} \end{array}\right]$$

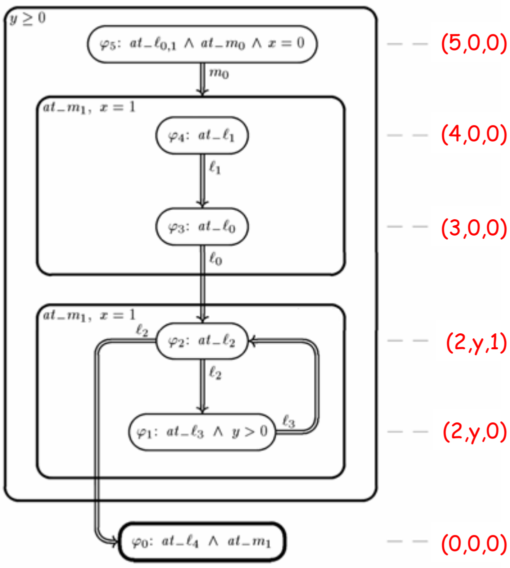
$$at\_l_0 \wedge at\_m_0 \wedge x{=}y{=}0 \quad \Rightarrow \quad \Diamond \quad at\_l_4 \wedge at\_m_1$$

UP-DOWN

local x, y: **integer where** $x = y = 0$

$$P_1 :: \left[\begin{array}{l} \ell_0\text{: } \textbf{while } x = 0 \textbf{ do} \\ \quad \ell_1\text{: } y := y + 1 \\ \ell_2\text{: } \textbf{while } y > 0 \textbf{ do} \\ \quad \ell_3\text{: } y := y - 1 \\ \ell_4\text{:} \end{array}\right]$$

$$|| \quad P_2 :: \left[\begin{array}{l} m_0\text{: } x := 1 \\ m_1\text{:} \end{array}\right]$$



Completeness

For a program P (with $\mathcal{C} = \emptyset$, $\mathcal{J} = \{\tau_1, \ldots, \tau_m\}$):
for every two state assertions p, q, such that

$$p \Rightarrow \Diamond q$$

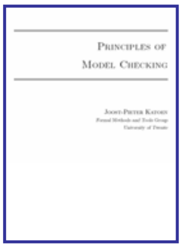
is P-valid, there exist

- assertions $q = \varphi_0, \varphi_1, \ldots, \varphi_m$,
- transitions $\tau_1, \tau_1, \ldots, \tau_m$,
- a well-founded domain $(A, \succ)$, and
- ranking functions $\delta_1, \delta_1, \ldots, \delta_m$

such that the premises of WELL-J are provable from state validities.

Proof: *later*

Finite-State Model Checking

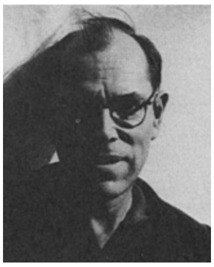


Principles of Model Checking

by Christel Baier and
Joost-Pieter Katoen

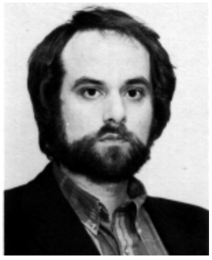
To appear in Spring 2008

(we'll distribute selected chapters in class.)



J. Richard Büchi

Edmund M. Clarke



E. Allen Emerson

Review: Finite-State Programs

For a computation σ,

$$\sigma: s_0, s_1, s_2, \ldots$$

state s_i is a accessible state.

A program is finite-state if the set of all accessible states is finite.

Peterson again!

local y_1, y_2: boolean where $y_1 = \text{F}, y_2 = \text{F}$
s : integer where $s = 1$

$P_1 ::$
ℓ_0 : loop forever do
- ℓ_1 : noncritical
- ℓ_2 : $(y_1,\ s) := (\text{T}, 1)$
- ℓ_3 : await $(\neg y_2) \vee (s = 2)$
- ℓ_4 : critical
- ℓ_5 : $y_1 := \text{F}$

||

$P_2 ::$
m_0 : loop forever do
- m_1 : noncritical
- m_2 : $(y_2,\ s) := (\text{T}, 2)$
- m_3 : await $(\neg y_1) \vee (s = 1)$
- m_4 : critical
- m_5 : $y_2 := \text{F}$

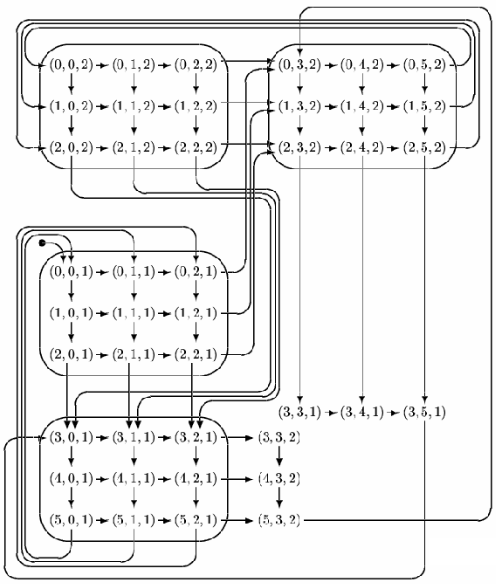
This is a finite-state program.

$s = 1, 2$

$y_1 = \text{T}, \text{F}$

$y_2 = \text{T}, \text{F}$

Peterson: State Transition Graph



(i, j, v) means $\pi\colon \{\ell_i, m_j\},\ s\colon v,$

$y_1 = \text{T}$ iff $3 \le i \le 5$
$y_2 = \text{T}$ iff $3 \le j \le 5$

Constructing the Transition Graph

- Initially
  Place as nodes in G_P all initial states (satisfy Θ)

- Repeat until no new nodes or new edges can be added to G_P

  For some $s \in G_P$, let $s_1, \ldots, s_k$ be its successors
  Add to G_P all new nodes in $\{s_1, \ldots, s_k\}$ and draw edges connecting s to s_i, $i = 1, \ldots, k$

Checking Invariance

For assertion q,
check validity of $\Box q$
over finite-state programs.

(= check that q is P-state valid)

Example: Peterson's Algorithm

Check assertions

φ_0: $\Box \neg(at\_\ell_4 \wedge at\_m_4)$

φ_1: $\Box(at\_\ell_3 \wedge \neg at\_m_3 \rightarrow s = 1)$

φ_2: $\Box(at\_m_3 \wedge \neg at\_\ell_3 \rightarrow s = 2)$

in the graph.

The assertions hold over all accessible states. Thus,

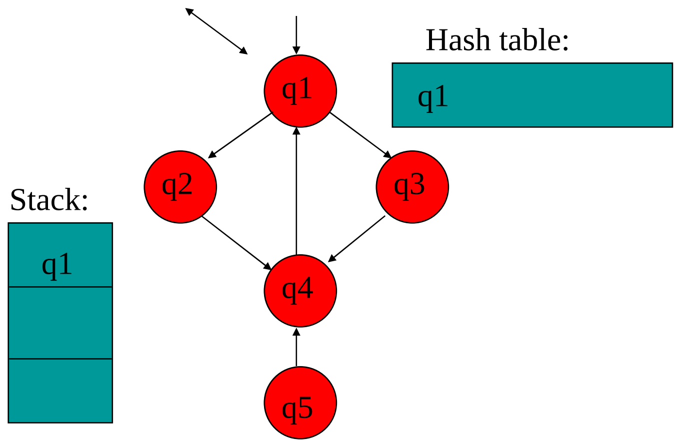
$$\Box\, \varphi_0, \ \Box\, \varphi_1, \ \Box\, \varphi_2$$

Depth First Search

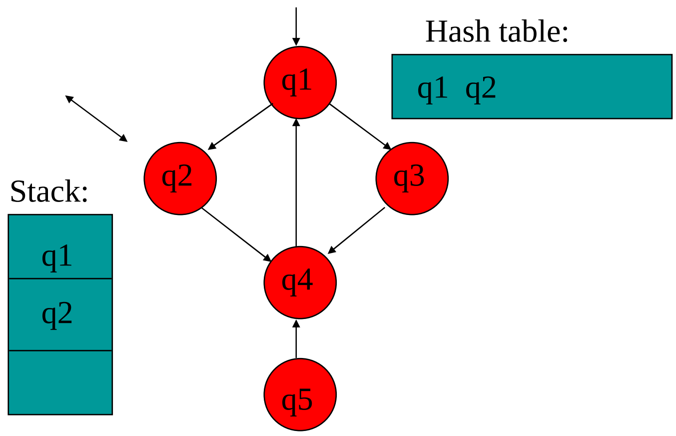
Program DFS
For each s such that s
satisfies θ do
dfs(s)
end DFS

Procedure dfs(s)
for each s' such that $s' \in \tau(s)$ do
If new(s') then dfs(s')
end dfs.

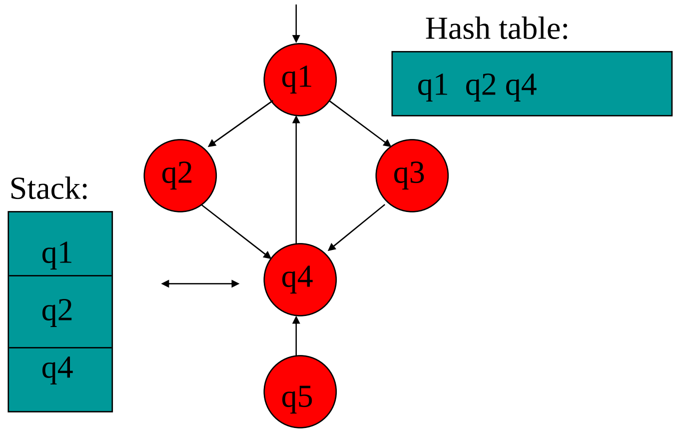
Start from an initial state



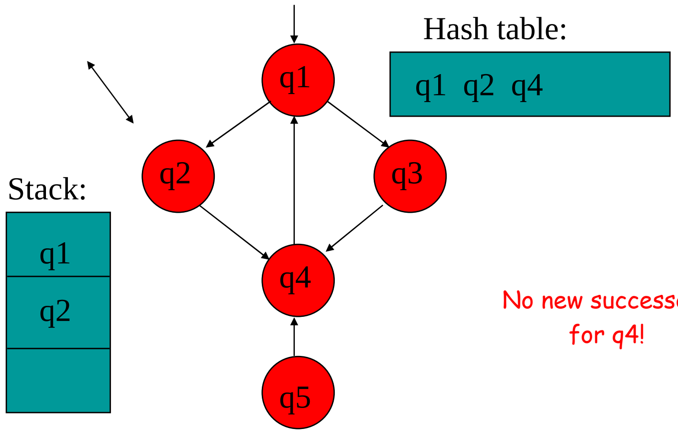
Continue with some successor…



One successor of q2…

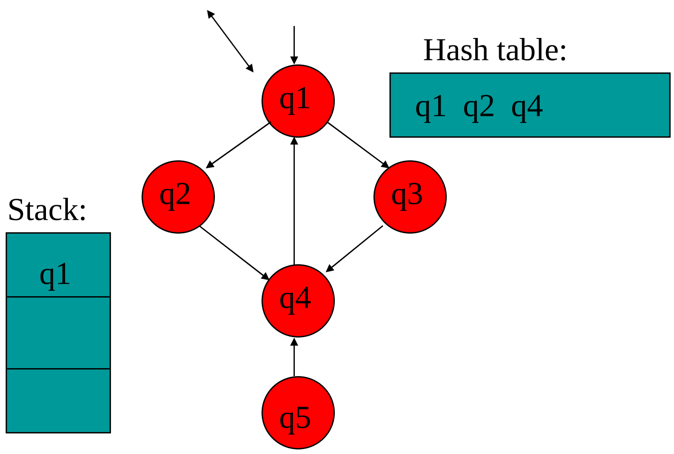


Backtrack

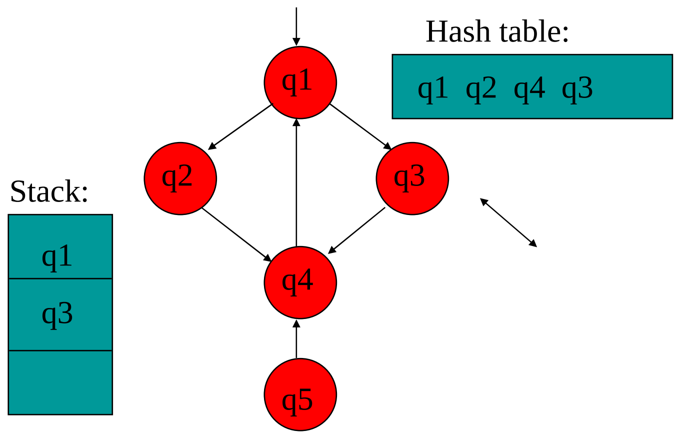


No new successors for q4!

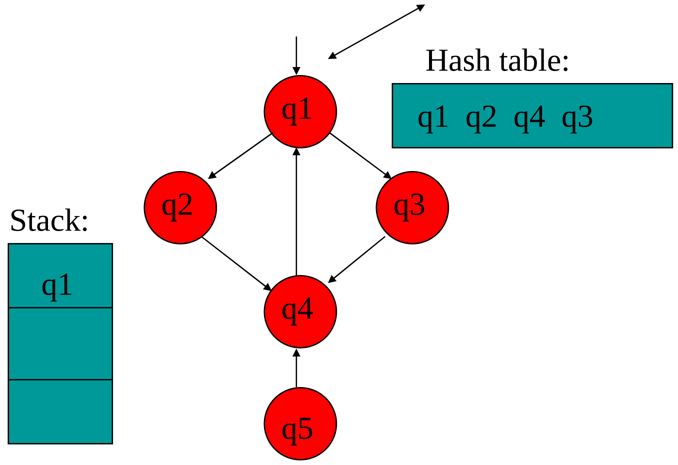
Backtrack...



Second successor



Backtrack



Beyond Invariance Checking

- Want to check more properties.
- Want to have a single algorithm that deals with all types of properties.

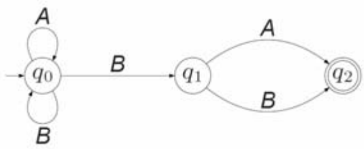
LTL formulas can be translated into graphs (finite automata).

Automata

Quick Review: Finite-State Automata

A *nondeterministic finite automaton* (NFA) $\mathcal{A}$ is a tuple $(Q, \Sigma, \delta, Q_0, F)$ where:

- Q is a finite set of states
- Σ is an alphabet
- $\delta : Q \times \Sigma \to 2^Q$ is a transition function
- $Q_0 \subseteq Q$ a set of initial states
- $F \subseteq Q$ is a set of accept (or: final) states



Language

- NFA $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$ and word $w = A_1 \ldots A_n \in \Sigma^*$
- A *run* for w in $\mathcal{A}$ is a finite sequence $q_0\, q_1\, \ldots\, q_n$ such that:
  - $q_0 \in Q_0$ and $q_i \xrightarrow{A_{i+1}} q_{i+1}$ for all $0 \leqslant i < n$
- Run $q_0\, q_1\, \ldots\, q_n$ is *accepting* if $q_n \in F$
- $w \in \Sigma^*$ is *accepted* by $\mathcal{A}$ if there exists an accepting run for w
- The *accepted language* of $\mathcal{A}$:

$$\mathcal{L}(\mathcal{A}) = \{ w \in \Sigma^* \mid \text{there exists an accepting run for } w \text{ in } \mathcal{A} \}$$

- NFA $\mathcal{A}$ and $\mathcal{A}'$ are *equivalent* if $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$

Extended Transition Function

Extend the transition function δ to $\delta^* : Q \times \Sigma^* \rightarrow 2^Q$ by:

$$\delta^*(q, \varepsilon) = \{ q \} \quad \text{and} \quad \delta^*(q, A) = \delta(q, A)$$

$$\delta^*(q, A_1 A_2 \ldots A_n) = \bigcup_{p \in \delta(q, A_1)} \delta^*(p, A_2 \ldots A_n)$$

$\delta^*(q, w)$ = set of states reachable from q for the word w

Then: $\mathcal{L}(\mathcal{A}) = \{ w \in \Sigma^* \mid \delta^*(q_0, w) \cap F \neq \varnothing \text{ for some } q_0 \in Q_0 \}$

Intersection

- Let NFA $\mathcal{A}_i = (Q_i, \Sigma, \delta_i, Q_{0,i}, F_i)$, with $i{=}1,2$
- The *product automaton*

$$\mathcal{A}_1 \otimes \mathcal{A}_2 = (Q_1 \times Q_2, \Sigma, \delta, Q_{0,1} \times Q_{0,2}, F_1 \times F_2)$$

where δ is defined by:

$$\frac{q_1 \xrightarrow{A}_1 q_1' \ \wedge \ q_2 \xrightarrow{A}_2 q_2'}{(q_1, q_2) \xrightarrow{A} (q_1', q_2')}$$

- Well-known result: $\mathcal{L}(\mathcal{A}_1 \otimes \mathcal{A}_2) = \mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$

Regular Expressions

For a regular expression R (over Σ)

- $\sigma \in R$ for every $\sigma \in \Sigma$
- If R_1, R_2 are regular expressions

$$R_1 + R_2 = \{x \mid x \in R_1 \text{ or } x \in R_2\}$$
$$R_1 \cdot R_2 = \{x \cdot y \mid x \in R_1 \text{ and } y \in R_2\}$$
$$R^* = \{\varepsilon\} \cup \{x \mid x \text{ obtained by concatenating a finite \# of words in } R\}$$

Examples

$\Sigma = \{a, b\}$

$abbaa$ is a word

$a^*ba^*ba^*$ – all words containing exactly 2 b's

ba^* – all words beginning with a $\underline{b}$ followed only by $\underline{a}$'s

$(a+b)^*$ – all words over $\{a, b\}$

$(a+b)^*(aa+bb)(a+b)^*$ – all words containing 2 consecutive a's or 2 consecutive b's

$(a^*b)^*$ – the empty word and all finite words over $\{a, b\}$ whose last letter is b

Deterministic Automata

Automaton $\mathcal{A}$ is called *deterministic* if

$$|Q_0| \leqslant 1 \quad \text{and} \quad |\delta(q, A)| \leqslant 1 \quad \text{for all } q \in Q \text{ and } A \in \Sigma$$

DFA $\mathcal{A}$ is called *total* if

$$|Q_0| = 1 \quad \text{and} \quad |\delta(q, A)| = 1 \quad \text{for all } q \in Q \text{ and } A \in \Sigma$$

any DFA can be turned into an equivalent total DFA

total DFA provide unique successor states, and thus, unique runs for each input word

Determinization

For NFA $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$ let $\mathcal{A}_{det} = (2^Q, \Sigma, \delta_{det}, Q_0, F_{det})$ with:

$$F_{det} = \{Q' \subseteq Q \mid Q' \cap F \neq \varnothing\}$$

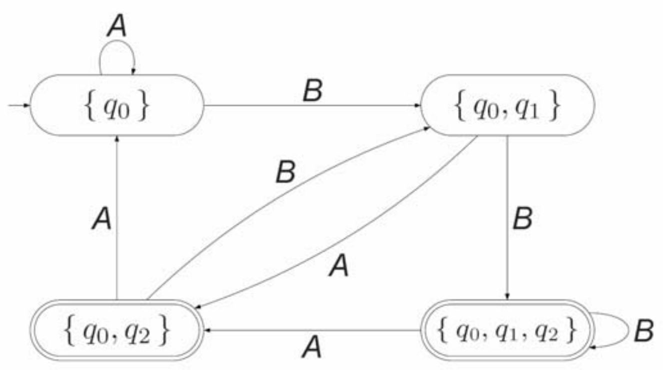
and the total transition function $\delta_{det} : 2^Q \times \Sigma \rightarrow 2^Q$ is defined by:

$$\delta_{det}(Q', A) = \bigcup_{q \in Q'} \delta(q, A)$$

$\mathcal{A}_{det}$ is a total DFA and, for all $w \in \Sigma^*$: $\delta^*_{det}(Q_0, w) = \bigcup_{q_0 \in Q_0} \delta^*(q_0, w)$

Thus: $\mathcal{L}(\mathcal{A}_{det}) = \mathcal{L}(\mathcal{A})$

Determinization



a deterministic finite automaton accepting $\mathcal{L}((A+B)^*B(A+B))$

Facts about NFAs

- They are as expressive as regular languages
- They are closed under $\cap$ and complementation
  - NFA $\mathcal{A} \otimes \mathcal{B}$ (= cross product) accepts $\mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{B})$
  - Total DFA $\overline{\mathcal{A}}$ (= swap all accept and normal states) accepts $\overline{\mathcal{L}(\mathcal{A})} = \Sigma^* \setminus \mathcal{L}(\mathcal{A})$
- They are closed under determinization (= removal of choice)
  - although at an exponential cost.....
- $\mathcal{L}(\mathcal{A}) = \varnothing$? = check for reachable accept state in $\mathcal{A}$
  - this can be done using a simple depth-first search
- For regular language $\mathcal{L}$ there is a unique minimal DFA accepting $\mathcal{L}$

Büchi Automata

- NFA (and DFA) are incapable of accepting infinite words
- Automata on infinite words
  - suited for accepting ω-regular languages
  - we consider nondeterministic Büchi automata (NBA)
- Accepting runs have to "check" the entire input word $\Rightarrow$ are infinite

  $\Rightarrow$ acceptance criteria for infinite runs are needed
- NBA are like NFA, but have a distinct *acceptance criterion*
  - one of the accept states must be visited infinitely often

Büchi Automata

A *nondeterministic Büchi automaton* (NBA) $\mathcal{A}$ is a tuple $(Q, \Sigma, \delta, Q_0, F)$ where:

- Q is a finite set of states with $Q_0 \subseteq Q$ a set of initial states
- Σ is an alphabet
- $\delta : Q \times \Sigma \rightarrow 2^Q$ is a transition function
- $F \subseteq Q$ is a set of accept (or: final) states

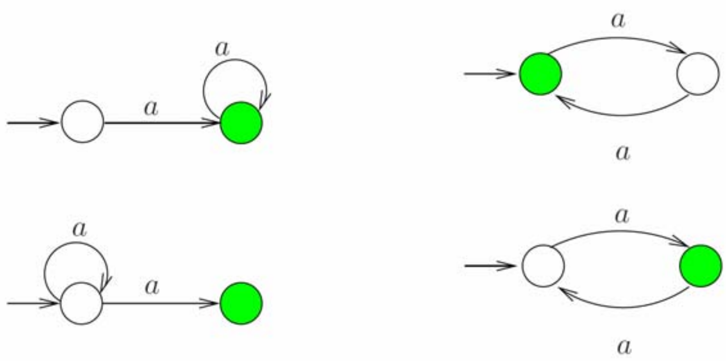
Language

- NBA $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$ and word $\sigma = A_0 A_1 A_2 \ldots \in \Sigma^\omega$
- A *run* for σ in $\mathcal{A}$ is an infinite sequence $q_0\, q_1\, q_2 \ldots$ such that:
  - $q_0 \in Q_0$ and $q_i \xrightarrow{A_i} q_{i+1}$ for all $0 \leqslant i$
- Run $q_0\, q_1\, q_2 \ldots$ is *accepting* if $q_i \in F$ for infinitely many i
- $\sigma \in \Sigma^\omega$ is *accepted* by $\mathcal{A}$ if there exists an accepting run for σ
- The *accepted language* of $\mathcal{A}$:

$$\mathcal{L}_\omega(\mathcal{A}) = \left\{ \sigma \in \Sigma^\omega \mid \text{ there exists an accepting run for } \sigma \text{ in } \mathcal{A} \right\}$$

- NBA $\mathcal{A}$ and $\mathcal{A}'$ are *equivalent* if $\mathcal{L}_\omega(\mathcal{A}) = \mathcal{L}_\omega(\mathcal{A}')$

NFA vs. NBA



finite equivalence $\not\Rightarrow$ ω-equivalence

$\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$, but $\mathcal{L}_\omega(\mathcal{A}) \neq \mathcal{L}_\omega(\mathcal{A}')$

ω-equivalence $\not\Rightarrow$ finite equivalence

$\mathcal{L}_\omega(\mathcal{A}) = \mathcal{L}_\omega(\mathcal{A}')$, but $\mathcal{L}(\mathcal{A}) \neq \mathcal{L}(\mathcal{A}')$

ω-Regular Expressions

For a regular expression R (where $\varepsilon \notin R$),

$\underline{R^\omega}$ is an ω-reg exp denoting the set of
all infinite words that can be represented
as the infinite concatenation

$$x_1 \cdot x_2 \cdot \ldots \cdot x_k \cdot \ldots$$

such that $x_i \in R$ for $i = 1, 2, \ldots$

> `Example:` $(a^*b)^\omega$
>
> denotes the set of
> all infinite words over $\{a, b\}$
> which contain infinitely many b's

ω-Regular Expressions (cont'd)

For regular expression R and
ω-regular expression O

$\underline{RO}$ is an ω-regular expression denoting
the set of all infinite words that can
be presented as the concatenation

$$xy$$

where $x \in R, y \in O$

> `Example:` $(a + b)^* b^\omega$
>
> denotes the set of
> all infinite words over $\{a, b\}$
> which contains finitely many a's

ω-Regular Expressions (cont'd)

For ω-regular expression O_1 and O_2

$O_1 + O_2$ is an ω-regular expression denoting
the union of the sets denoted by
O_1 and O_2.

Example: The ω-regular expression

$$(a+b)^*b^\omega \ + \ (a+b)^*a^\omega$$

denotes the set of
infinite words over $\{a, b\}$
which either contain finitely many a's
or finitely many b's.

NBA and ω-Regular Languages

The class of languages accepted by NBA
agrees with the class of ω-regular languages

(1) any ω-regular language is recognized by an NBA

(2) for any NBA $\mathcal{A}$, the language $\mathcal{L}_\omega(\mathcal{A})$ is ω-regular

For any ω-regular language there is an NBA

- How to construct an NBA for the ω-regular expression:

$$G = E_1.F_1^\omega + \ldots + E_n.F_n^\omega \; ?$$

where E_i and F_i are regular expressions over alphabet Σ; $\varepsilon \notin F_i$

- Rely on operations for NBA that mimic operations on ω-regular expressions:

(1) for NBA $\mathcal{A}_1$ and $\mathcal{A}_2$ there is an NBA accepting $\mathcal{L}_\omega(\mathcal{A}_1) \cup \mathcal{L}_\omega(\mathcal{A}_2)$
(2) for any regular language $\mathcal{L}$ with $\varepsilon \notin \mathcal{L}$ there is an NBA accepting $\mathcal{L}^\omega$
(3) for regular language $\mathcal{L}$ and NBA $\mathcal{A}'$ there is an NBA accepting $\mathcal{L}.\mathcal{L}_\omega(\mathcal{A}')$

Union

For NBA $\mathcal{A}_1$ and $\mathcal{A}_2$ (both over the alphabet Σ)
there exists an NBA $\mathcal{A}$ such that:
$\mathcal{L}_\omega(\mathcal{A}) = \mathcal{L}_\omega(\mathcal{A}_1) \cup \mathcal{L}_\omega(\mathcal{A}_2)$ and $|\mathcal{A}| = \mathcal{O}(|\mathcal{A}_1| + |\mathcal{A}_2|)$

The size of $\mathcal{A}$, denoted $|\mathcal{A}|$, is the number of states and transitions in $\mathcal{A}$:

$$|\mathcal{A}| \;=\; |Q| + \sum_{q \in Q} \sum_{A \in \Sigma} |\,\delta(q, A)\,|$$

ω-Operator (for NFA)

For each NFA $\mathcal{A}$ with $\varepsilon \notin \mathcal{L}(\mathcal{A})$ there exists an NBA $\mathcal{A}'$ such that:

$$\mathcal{L}_\omega(\mathcal{A}') = \mathcal{L}(\mathcal{A})^\omega \quad \text{and} \quad |\mathcal{A}'| = \mathcal{O}(|\mathcal{A}|)$$