5.2 From inclusion to universality

Goal: · Previous problem was $\Sigma^\omega \overset{(\subseteq)}{=} \mathcal{L}(A)$

· Solve more general problem $\mathcal{L}(A) \subseteq \mathcal{L}(B)$ for ω-Büchi NBAs A, B (above fixes $\Sigma^\omega = \mathcal{L}(\Sigma \odot^\omega \Sigma)$)

Approach: Reduce inclusion to universality.

Claim:

Let A, B two NBAs over Σ.

There is an NBA C over an alphabet Δ so that

$\mathcal{L}(A) \subseteq \mathcal{L}(B)$ iff $\mathcal{L}(C) = \Delta^\omega$.

Moreover, the size of C is polynomial in the sizes of A and B.

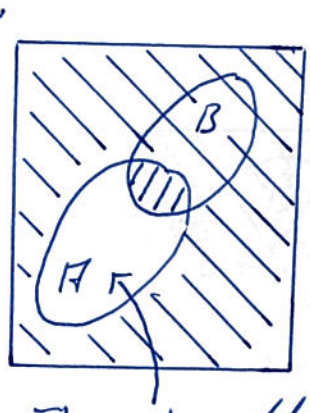
Approach:

Consider two languages L_1 and L_2 over an alphabet Γ. Then

$L_1 \subseteq L_2$

iff $(L_1 \cap L_2) = L_1$

iff $(L_1 \cap L_2) \cup \overline{L_1} = \Gamma^\omega$



This part would be missing if inclusion failed.

Problem:

For a word, a single run does not reveal whether or not the word is accepted (there may be another run that is accepting)

Idea:

But for every run, one can of course decide whether or not it is accepting.

⇒ base technical development on runs of A (instead of words)

⇒ let automaton C work on letter + states of A (new alphabet)

Turn to the technicalities.

Let $\mathcal{A} = (Q_{\mathcal{A}}, q_0^{\mathcal{A}}, \to_{\mathcal{A}}, Q_F^{\mathcal{A}})$ and $\mathcal{B} = (Q_{\mathcal{B}}, q_0^{\mathcal{B}}, \to_{\mathcal{B}}, Q_F^{\mathcal{B}})$.

To let $\subseteq$ work on the runs of $\mathcal{A}$, extend the alphabet to

$$\Delta := \Sigma \times Q_{\mathcal{A}}$$

We then have

$$L(\mathcal{A}) \subseteq L(\mathcal{B})$$

iff every accepting run of $\mathcal{A}$

$$r_{\mathcal{A}} = q_0^{\mathcal{A}} \xrightarrow{a_0}_{\mathcal{A}} q_1^{\mathcal{A}} \xrightarrow{a_1}_{\mathcal{A}} \cdots$$

yields an accepting run of $\mathcal{B}$

$$r_{\mathcal{B}} = q_0^{\mathcal{B}} \xrightarrow{a_0}_{\mathcal{B}} q_1^{\mathcal{B}} \xrightarrow{a_1}_{\mathcal{B}} \cdots$$

Rewriting the implication, this can be equivalently expressed as

for every sequence $q_0^{\mathcal{A}} \xrightarrow{a_0}_{\mathcal{A}} q_1^{\mathcal{A}} \xrightarrow{a_1}_{\mathcal{A}} \cdots$ in Δ^ω we have

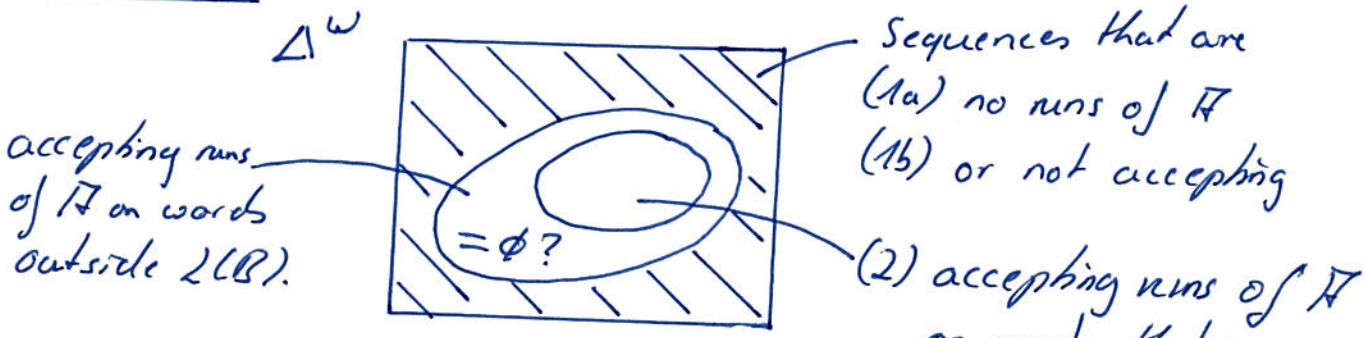
(1) the sequence is no accepting run of $\mathcal{A}$ (because

(1a) it is no run of $\mathcal{A}$

or (1b) it is not accepting)

(2) $a_0 a_1 \ldots \in L(\mathcal{B})$.

Illustration:



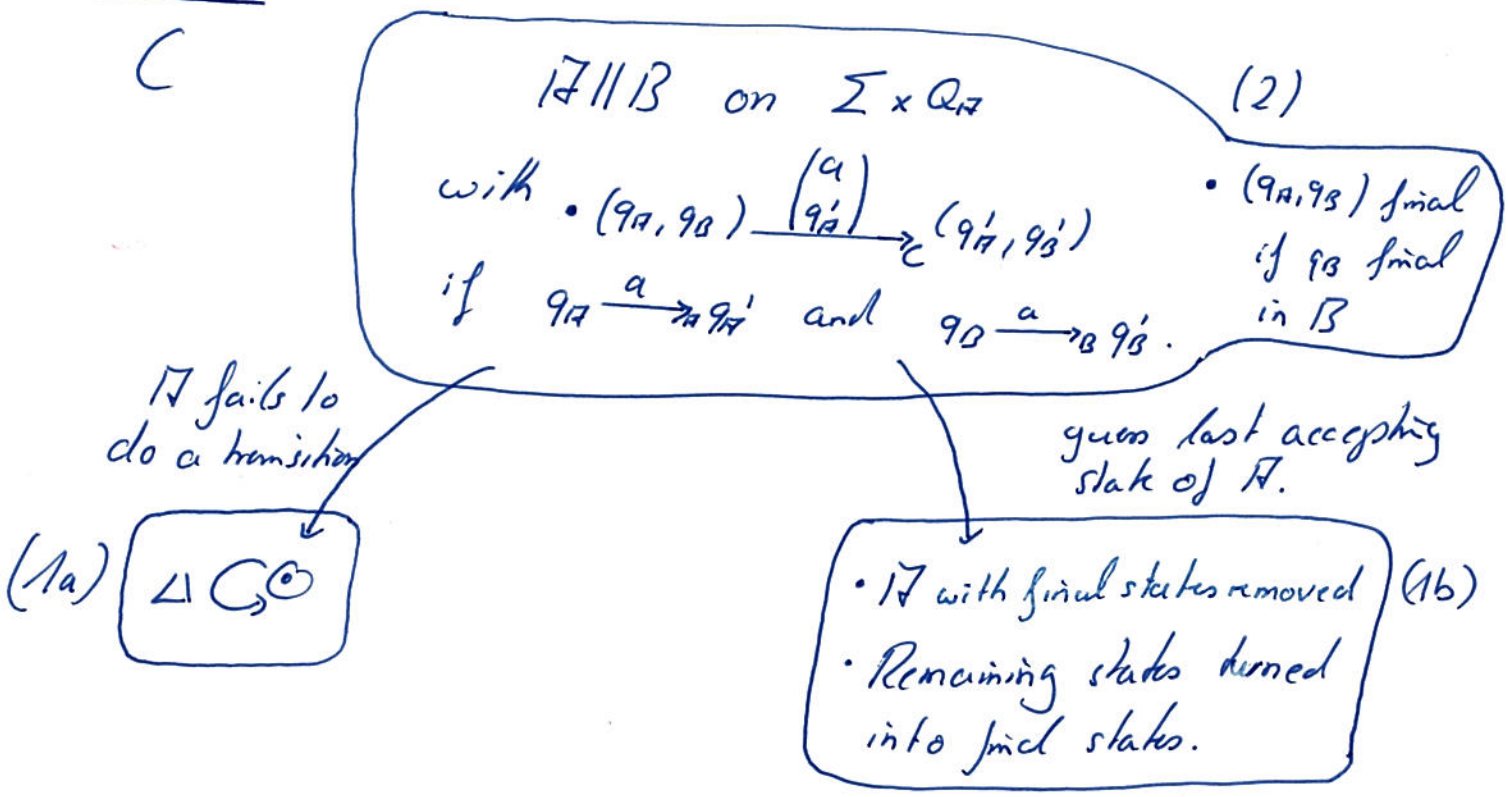
Inclusion holds iff $= \emptyset$ is true, i.e., (1a), (1b), (2) cover Δ^ω.

Construct C so that it accepts the Δ^ω words that satisfy (1a), (1b), or (2).

This means (with the above equivalences)

$$\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B}) \quad \text{iff} \quad \mathcal{L}(C) = (\Sigma \times Q_{\mathcal{A}})^\omega = \Delta^\omega.$$

Construction:



Behaviour:

C guesses run of $\mathcal{A}$ and $\mathcal{B}$ on ω:

↳ Check that it respects transitions of $\mathcal{A}$.

⇒ No: accept by (1a).

⇒ Yes: Guess whether $\mathcal{A}$ accepts

⇒ $\mathcal{A}$ accepts: $\mathcal{B}$ has to accept (2)

⇒ $\mathcal{A}$ does not accept:

↳ Guess last accepting state of $\mathcal{A}$

↳ Accept from now on (1b).

5.3 Inclusion, the standard way

Alternative approach to inclusion checking: $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$ by emptiness checking.

Remember: NBA languages are (effectively) closed under complementation.
Therefore, all operations used in the following equivalences are effective:

$$\mathcal{L}(A) \subseteq \mathcal{L}(B)$$

(Set theory) $\Leftrightarrow \mathcal{L}(A) \cap \overline{\mathcal{L}(B)} = \emptyset$

(construction $\bar{B}$) $\Leftrightarrow \mathcal{L}(A) \cap \mathcal{L}(\bar{B}) = \emptyset$

(construction $\|$) $\Leftrightarrow \mathcal{L}(A \| \bar{B}) = \emptyset$.

Lemma:

For a given NBA A, it is decidable (in polynomial time) whether $\mathcal{L}(A) = \emptyset$.

Proof:

We have $\mathcal{L}(A) = \emptyset$ iff there is a state $q \in Q$ that

- is reachable from q_0
- lies on a cycle of length ≥ 1 (edges) which contains a final state.

□

6. Linear-time temporal logic

- Specification language for model checking:
  In $A \models \varphi$, formula φ is typically in LTL.
- Used in industry (PSL = property specification language, variant of LTL, like stakmachines in UML are derived from finite automata).
- Found by Amir Pnueli '77, Turing award 1996

Idea of LTL:

↳ subset of FO useful for specification
↳ No quantifiers, more complex and intuitive operators
↳ View word as a sequence of (sets of) actions over time
↳ Interpret formula at a single moment/point in the word

$w = \alpha \overset{a}{|} \beta$

$\Rightarrow$ a is now

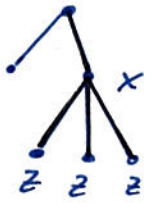
$\Rightarrow$ β is future

↳ Operators make claims about the future.

Remark:

- LTL is a linear-time temporal logic
  that talks about words
- CTL is a branching-time temporal logic
  that talks about computation trees:

$$E\bigcirc(x \wedge A\bigcirc z)$$



- CTL* unifies (and generalises)
  LTL and CTL.

Goal:

↳ Translate LTL into NBA for model checking

↳ LTL can be seen as a subset of MSO

$\Rightarrow$ We know that it can be done

↳ But it is strictly less expressive than MSO

$\Rightarrow$ We obtain a faster and easier algorithm.

6.1 Syntax and semantics of LTL

Recall: For translation of WMSO formulas $\varphi(X_1, \dots, X_n)$
we had NFAs over $\{0,1\}^n$, vectors of Booleans.

In LTL:
- Finite set of propositions $p, q, \dots \in \mathcal{P}$
  ↳ Imitate second-order variables X_i
  ↳ Finite in every system
- Define alphabet $\Sigma = \mathbb{P}(\mathcal{P})$
  ↳ Letters again yield vectors:
  $a \in \Sigma$ means $a \in \mathcal{P}$ with $a = \begin{pmatrix} 1 \text{ if } p_1 \in a \\ 0 \text{ if } p_2 \notin a \\ \vdots \\ 1 \text{ if } p_n \in a \end{pmatrix}$

↳ but we can use set notation, $p \in a$.

- Practically, system
  - does more than one action at a time
  - has components that are in one state each

Definition (Syntax of LTL):

Let $S = \{p, q, \ldots\}$ a finite set of propositions.

Formulas in LTL over $\Sigma = \mathbb{P}(S)$ are defined by

$$\varphi ::= p \mid \varphi \vee \psi \mid \neg\varphi \mid \underbrace{\bigcirc\varphi}_{\text{"next"}} \mid \underbrace{\varphi \,U\, \psi}_{\text{"until"}}$$

with $p \in S$. Use following abbreviations (besides standard abbreviations for Boolean operators):

$$\underbrace{\Diamond\varphi}_{\text{"eventually"}} = \text{true}\; U\, \varphi \qquad \underbrace{\Box\varphi}_{\text{"globally"}} = \neg\Diamond\neg\varphi$$

$$\underbrace{\varphi \,R\, \psi}_{\text{"release"}} = \neg(\neg\varphi \,U\, \neg\psi)$$

Intuition:

- p = proposition p holds at the current position
- $\bigcirc\varphi$ = the next position satisfies φ
- $\varphi U \psi$ = ↳ φ holds in all positions until ψ holds
  - ↳ And ψ definitely holds some time later (or already now)
- $\Diamond\varphi$ = there is some future moment in which φ holds
- $\Box\varphi$ = from now on, φ holds in all moments in the future
- $\varphi R \psi$ = dual of until ↳ ψ holds as long as it is not released by φ
  - ↳ ψ may hold forever or
  - ↳ there is a moment with ψ and φ.

Definition (Satisfaction relation):

Let $\omega = a_0 a_1 \ldots \in \Sigma^\omega = \mathbb{P}(S)^\omega$. The <u>satisfaction relation</u> $\models$ is defined inductively as follows (for all $i \in \mathbb{N}$):

$\omega, i \models p$ if $p \in a_i$:

$\omega, i \models \varphi \vee \psi$ if $\omega, i \models \varphi$ or $\omega, i \models \psi$

$\omega, i \models \neg \varphi$ if not $\omega, i \models \varphi$

$\omega, i \models \bigcirc \varphi$ if $\omega, i+1 \models \varphi$

$\omega, i \models \varphi \mathcal{U} \psi$ if there is $k \geq i$ so that
- for all $i \leq j < k$ we have $\omega, j \models \varphi$
- $\omega, k \models \psi$.

An LTL-formula φ <u>defines a language</u> $L(\varphi) \subseteq \Sigma^\omega$ by interpreting it in the first position of a word:

$$\omega \models \varphi \text{ if } \omega, 0 \models \varphi$$

and

$$L(\varphi) := \{\omega \in \Sigma^\omega \mid \omega \models \varphi\}.$$

<u>Examples:</u>

- Infinitely often φ: $\Box\Diamond\varphi$
- Finitely often φ: $\Diamond\Box\neg\varphi$
- If there are infinitely many positions with p, then there are infinitely many positions with q:
$$\Box\Diamond q \vee \Diamond\Box\neg p$$
- Every request is followed by an acknowledge:
$$\Box(req \rightarrow \Diamond ack).$$

<u>Note:</u>

- Assume your system creates entities (object-oriented programs with new)
- Then LTL is not sufficient to express correctness.

$$\forall \text{components } c: \Box(req@c \Rightarrow ack@c).$$

Defining such a logic + basic decision procedures could be a Master's thesis.