Two Applications of Well-Quasi-Orderings and Well-Structuredness

1. Downward and Upward Closures of Formal Languages

Let Σ be a finite alphabet.
Since finite sets are wqo by equality, also $(\Sigma, =)$ is a wqo.
This means we can apply Higman's lemma
and obtain a wqo $\leq^*$ on Σ^*.
It corresponds to taking subwords,

$$u \leq^* v \quad \text{if} \quad u = a_1 \dots a_n$$
$$\text{and } v = v_1 a_1 v_2 \dots v_n a_n v_{n+1}, \quad a_1, \dots, a_n \in \Sigma, \; v_1, \dots, v_{n+1} \in \Sigma^*.$$

Example:

$$\{abc\}\downarrow = \{\varepsilon, a, b, c, ab, bc, ac, abc\}.$$

Goal: Show that taking downward closures wrt. $\leq^*$
of languages from Σ^*
corresponds to moving to a simple class of languages.

Theorem (Haines & Bouajjani):

Let $L \subseteq \Sigma^*$.
Then (Haines) $L\downarrow$ is regular
(Bouajjani) even simple regular.

Simple regular means the language can be represented
by an expression SRE of the following form:

$$e ::= (a + \varepsilon) \mid (a_1 + \dots + a_n)^*$$
$$p ::= e_1 \dots e_n$$
$$\text{SRE} ::= p_1 + \dots + p_m.$$

Lemma:

$L\uparrow$ is regular.

Proof:

Since $L\uparrow$ is upward closed wrt. a wqo,
there are finitely many words in $\min(L\uparrow)$.
We have

$$L\uparrow = \min(L\uparrow)\uparrow$$

$$(\text{Distributivity}) = \bigcup_{w \in \min(L\uparrow)} w\Sigma^*\uparrow$$

$$(\text{Definition of } \leq^*) = \bigcup_{\substack{w \in \min(L\uparrow) \\ w = a_1 \dots a_n}} \Sigma^* a_1 \Sigma^* \dots \Sigma^* a_n \Sigma^*.$$

Certainly, $\Sigma^* a_1 \Sigma^* \dots \Sigma^* a_n \Sigma^*$ is a regular language.
Regular languages are closed under finite union.
So $L\uparrow$ is regular. □

Proof (Haines' theorem):

The complement $\overline{L\downarrow}$ of $L\downarrow$ is upward closed.
Hence, by the previous lemma, it is regular.
Since regular languages are closed under complement,
also

$$\overline{\overline{L\downarrow}} = L\downarrow$$

is regular. □

Remark:

If representation (by an NFA or a regular expression) of $L\downarrow$ is typically not computable.

To see this, let $L = L(TM)$ be the language of a Turing machine where
- we label transitions, say by their operation,
- and accept upon reaching a halting state.

Now $L(TM)\downarrow$ is regular by Haines' theorem.
If we could compute an NFA for the language, we had

$$TM \text{ halts} \quad \text{iff} \quad L(TM)\downarrow \neq \emptyset$$

(Definition of downward closure) iff $\varepsilon \in L(NFA)$.

Since the halting problem is undecidable but $\varepsilon \in L(NFA)$ is decidable, we cannot compute this NFA.

Remark:

It is an important problem to understand when downward closures are computable.

Why:

1.) The problem generalizes reachability.

2.) The downward closure is what a component sees in an asynchronous interaction.

Asynchronous = shared variables, no acknowledgment, no atomic test-and-set.

Indeed, if the sender writes

$w(x,1).\ w(y,1).\ w(x,2).\ w(y,2)$,

a slow receiver may only see

$w(x,1).\ w(x,2).\ w(y,2)$.

See a recent (Concur '15) paper.

Similarly:

Read(LCS)↓ is not computable,

but can be approximated precise enough

to decide coverability / control-state reachability.

Classes of Languages with Computable Downward Closures:

Context-free languages: van Leeuwen '77.

Petri net languages: Habermehl, Meyer, Wimmel '10.

Higher-order languages of order 2 (indexed languages): Zetzsche '15 + mild conditions under which downward closures are computable.

Open: Orders > 2.

2. The Antichain Principle

Goal: Optimize language inclusion checks

$L(A) \subseteq L(B)$,

say A and B are NFAs.

Traditionally, this would be solved by computing

$\overline{A \times \det(B)}$

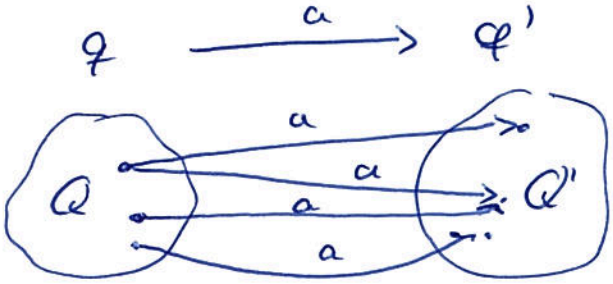
and checking the language for emptiness.

Problem: $\det(B)$ may be exponentially large.

Idea: Solve $\mathcal{L}(A \times \overline{\det(B)}) = \emptyset$ on-the-fly.

Technically:

- Consider a state of $A \times \overline{\det(B)}$.
  It takes the form
  (q, Q) with $q \in Q_A$ and $Q \subseteq Q_B$.
  Recall that in $(q, Q) \xrightarrow{a} (q', Q')$
  we consider every a-successor of every state in Q:



- The goal is to find a state that accepts in A and avoids F_B.
  So we are looking for (q_f, Q_{nf}) with
  $q_f \in F_A$ and $Q_{nf} \cap F_B = \emptyset$.

Key: If $Q \subseteq Q'$, it is easier to avoid F_B from (q, Q) than from (q, Q').

Idea: Store only states (q, Q) that are minimal wrt.

$$(q_1, Q_1) \leq (q_2, Q_2), \text{ if } q_1 = q_2 \text{ and } Q_1 \subseteq Q_2.$$

Claim:
This still yields a complete inclusion checker.

Argument:

- Since $\overline{det(B)}$ is deterministic, state q in (q, Q) determines the transition.
  Component Q will be stable to minimize it.
  (potentially going to $Q' = \emptyset$).

- This means:
  if $(q, \widetilde{Q}) \xrightarrow{a} (q', \widetilde{Q}')$, then
  $(q', \widetilde{Q}')$
  $\vee$
  (q, Q)
  $(q, Q) \xrightarrow{a} (q', Q')$.

So if we manage to avoid F-B from a larger state, we will be able to avoid F-B from the smaller state. □

Various Extensions:

- Büchi automata
- Visibly pushdown automata
- Context-free languages

=> ω-context-free languages ?