Operations on relations

Definition 6.10 *A word relation $R \subseteq \Sigma^* \times \Sigma^*$ has* length $n \geq 0$ *if it is empty and $n = 0$, or if it is nonempty and for all pairs (w_1, w_2) of R the words w_1 and w_2 have length n. If R has length n for some $n \geq 0$, then we say that R is a* fixed-length *word relation, or that R* has fixed-length.

Definition 6.12 *The* master transducer *over the alphabet Σ is the tuple $MT = (Q_M, \Sigma \times \Sigma, \delta_M, F_M)$, where*

- *Q_M is is the set of all fixed-length relations;*
- *$\delta_M : Q_M \times (\Sigma \times \Sigma) \to Q_M$ is given by $\delta_M(R, [a, b]) = R^{[a,b]}$ for every $q \in Q_M$ and $a, b \in \Sigma$;*
- *$F_M = \{(\varepsilon, \varepsilon)\}$.*

With T_R as the "fragment" of MT with R as root we get:

Proposition 6.13 *For every fixed-length word relation R, the transducer T_R is the minimal deterministic transducer recognizing R.*

Storing minimal transducers

Like minimal DFA, minimal deterministic transducers are represented as tables of nodes. However, a remark is in order: since a state of a deterministic transducer has $|\Sigma|^2$ successors, one for each letter of $\Sigma \times \Sigma$, a row of the table has $|\Sigma|^2$ entries, too large when the table is only sparsely filled. Sparse transducers over $\Sigma \times \Sigma$ are better encoded as NFAs over Σ by introducing auxiliary states: a transition $q \xrightarrow{[a,b]} q'$ of the transducer is "simulated" by two transitions $q \xrightarrow{a} r \xrightarrow{b} q'$, where r is an auxiliary state with exactly one input and one output transition.

Computing joins

Equations:

- $\emptyset \circ R = R \circ \emptyset = \emptyset$;
- $\{(\varepsilon, \varepsilon)\} \circ \{(\varepsilon, \varepsilon)\} = \{(\varepsilon, \varepsilon)\}$;
- $R_1 \circ R_2 = \bigcup_{a,b,c \in \Sigma} [a, b] \cdot \left(R_1^{[a,c]} \circ R_2^{[c,b]}\right)$.

Input: transducer table T, states q_1, q_2 of T
Output: state recognizing $\mathcal{L}(q_1) \circ \mathcal{L}(q_2)$

$join[T](q_1, q_2)$
    if $G(q_1, q_2)$ is not empty **then return** $G(q_1, q_2)$
    if $q_1 = q_\emptyset$ **or** $q_2 = q_\emptyset$ **then return** $q_\emptyset$
    else if $q_1 = q_\epsilon$ **and** $q_2 = q_\epsilon$ **then return** q_ϵ
    else $/ * \; q_\emptyset \neq q_1 \neq q_\epsilon, q_\emptyset \neq q_2 \neq q_\epsilon \; * /$
        for all $(a_i, a_j) \in \Sigma \times \Sigma$ **do**
            $q_{a_i,a_j} \leftarrow union[T]\left(join\left(q_1^{[a_i,a_1]}, q_2^{[a_1,a_j]}\right), \ldots, join\left(q_1^{[a_i,a_m]}, q_2^{[a_m,a_j]}\right)\right)$
        $G(q_1, q_2) = make(q_{a_1,a_1}, \ldots, q_{a_1,a_m}, \ldots, q_{a_m,a_m})$
        return $G(q_1, q_2)$

Pre and Post

Pre and Post can be reduced to intersection and projection. Define:

$$emb(L) = \{[v_1, v_2] \in (\Sigma \times \Sigma)^n \mid v_2 \in L\}$$

$$pre_S(L) = \{w_1 \in \Sigma^n \mid \exists [v_1, v_2] \in S : v_1 = w_1 \text{ and } v_2 \in L\}$$

Then we have:

$$pre_S(L) = proj_1(S \cap emb(L))$$

We use this to derive equations.

Equations:

$$\textit{if } S = \emptyset \textit{ or } L = \emptyset, \textit{ then } pre_S(L) = \emptyset;$$

$$\textit{if } S \neq \emptyset \neq L \textit{ then } pre_S(L) = \bigcup_{a,b \in \Sigma} a \cdot pre_{S[a,b]}(L^b),$$

$$\textit{where } S^{[a,b]} = \{w \in (\Sigma \times \Sigma)^* \mid [a,b]w \in S\}.$$

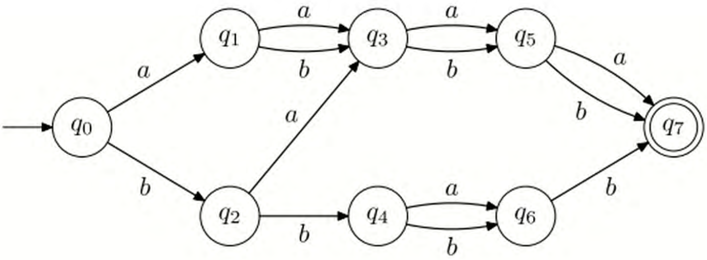
$$
\begin{aligned}
(pre_S(L))^a &= (proj_1(S \cap emb(L)))^a \\
&= \left(proj_1 \left(\bigcup_{b\in\Sigma} [a,b] \cdot (S \cap emb(L))^{[a,b]} \right) \right)^a \\
&= \left(\bigcup_{b\in\Sigma} proj_1 \left([a,b] \cdot (S \cap emb(L))^{[a,b]} \right) \right)^a \\
&= \left(\bigcup_{b\in\Sigma} a \cdot proj_1 \left((S \cap emb(L))^{[a,b]} \right) \right)^a \\
&= \bigcup_{b\in\Sigma} proj_1 \left((S \cap emb(L))^{[a,b]} \right) \\
&= \bigcup_{b\in\Sigma} proj_1 \left(S^{[a,b]} \cap emb\left(L^b\right) \right) \\
&= \bigcup_{b\in\Sigma} pre_{S^{[a,b]}}(L^b)
\end{aligned}
$$

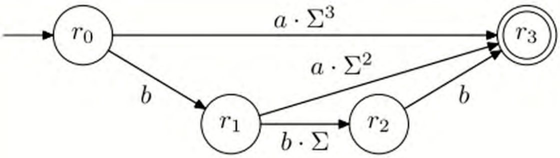
Input: transducer table TT, table T, state r of TT, state q of T

Output: state of T recognizing $pre_{\mathcal{L}(r)}(\mathcal{L}(q))$

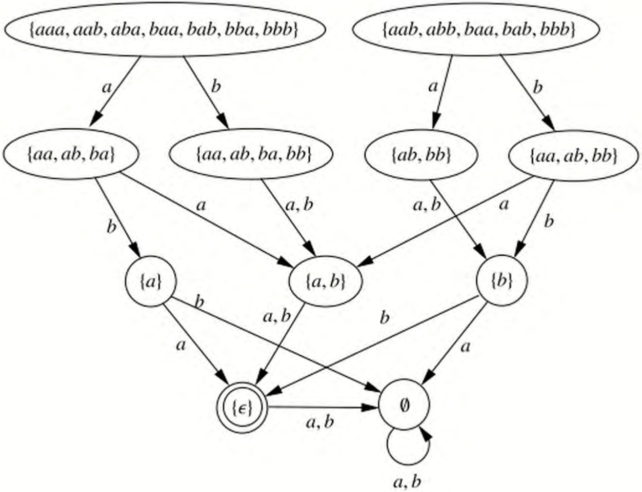
$pre[TT,T](r,q)$
  if $G(r,q)$ is not empty **then return** $G(r,q)$
  if $r = r_\emptyset$ **or** $q = q_\emptyset$ **then return** $q_\emptyset$
  else if $r = r_\epsilon$ **and** $q = q_\epsilon$ **then return** q_ϵ
  else
    for all $a_i \in \Sigma$ **do**
      $q_{a_i} \leftarrow union\left(pre[TT,T]\left(q^{[a_i,a_1]}, r^{a_1}\right), \ldots, pre[TT,T]\left(q^{[a_i,a_m]}, r^{a_m}\right)\right)$
    $G(r,q) \leftarrow make(q_{a_1}, \ldots, q_{a_m})$;
    return $G(r,q)$

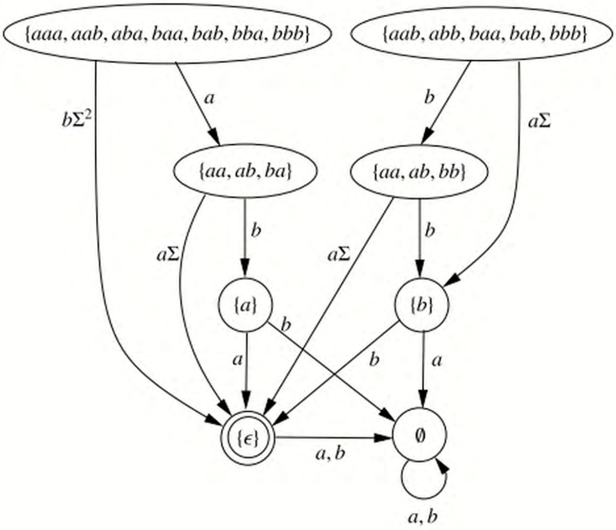
Binary Decision



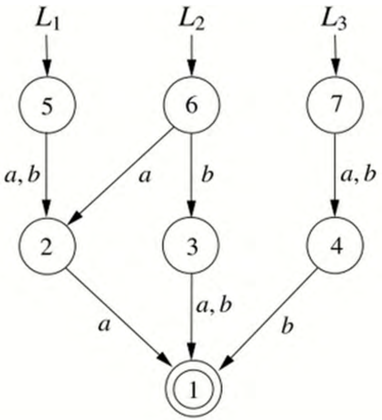


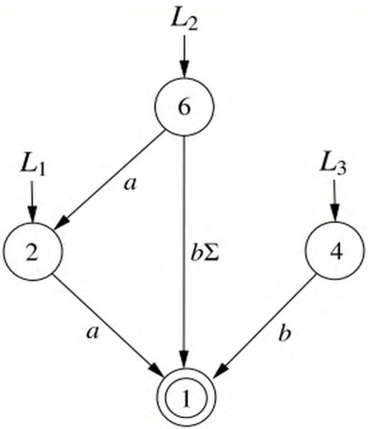
The master z-automaton



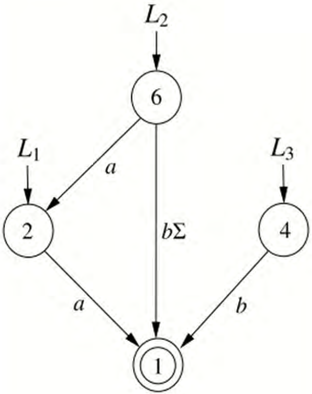

{aaa, aab, aba, baa, bab, bba, bbb}
{aab, abb, baa, bab, bbb}
a
b
bΣ²
aΣ
{aa, ab, ba}
{aa, ab, bb}
b
aΣ
aΣ
b
{a}
{b}
b
a
b
a
{ϵ}
∅
a, b
a, b

Length: 2

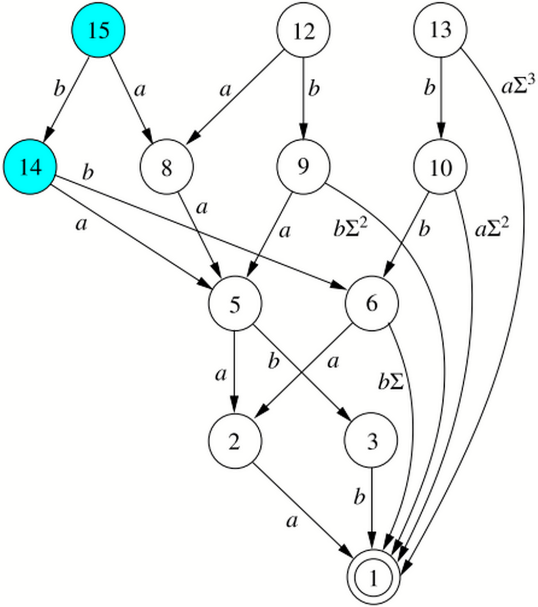


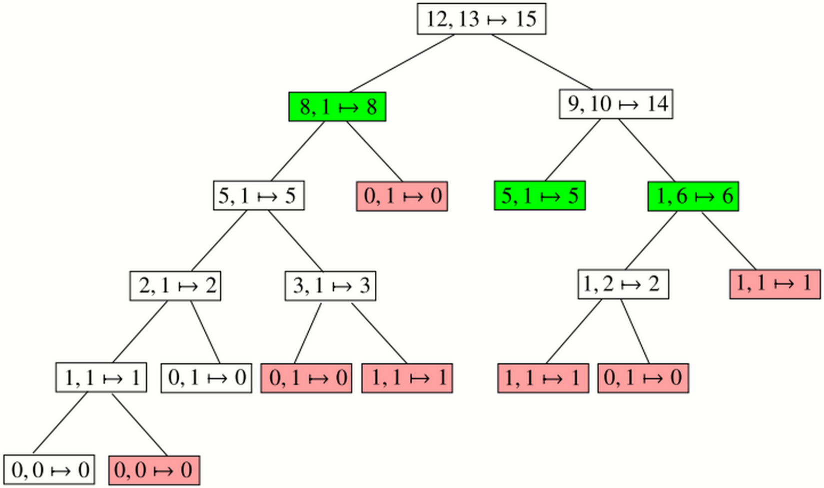


Data structure for z-automata



Ident.	Length	a-succ	b-succ
1	0	0	0
2	1	1	0
4	1	0	1
6	2	2	1


15
12
13
b
a
a
b
b
aΣ³
14
8
9
10
b
a
a
a
bΣ²
b
aΣ²
5
6
a
b
a
bΣ
2
3
a
b
1



12, 13 ↦ 15
8, 1 ↦ 8
9, 10 ↦ 14
5, 1 ↦ 5
0, 1 ↦ 0
5, 1 ↦ 5
1, 6 ↦ 6
2, 1 ↦ 2
3, 1 ↦ 3
1, 2 ↦ 2
1, 1 ↦ 1
1, 1 ↦ 1
0, 1 ↦ 0
0, 1 ↦ 0
1, 1 ↦ 1
1, 1 ↦ 1
0, 1 ↦ 0
0, 0 ↦ 0
0, 0 ↦ 0

8. Verification

- We use languages to describe the implementation and specification of a system.
- We reduce the verification problem to language inclusion between implementation and specification.

```
1  while x = 1 do
2      if y = 1 then
3          x ← 0
4      y ← 1 − x
5  end
```

- Configuration: triple $[l, n_x, n_y]$ where
  - l is the current value of the program counter, and
  - n_x, n_y are the current values of x, y

  Examples: [0,1,1], [5,0,1]

- Initial configuration: configuration with $l = 1$

- Potential execution: finite or infinite sequence of configurations

  Examples: [0,1,1][4,1,0]

  [2,1,0][5,1,0]

  [1,1,0][2,1,0][4,1,0][1,1,0]

```
1  while x = 1 do
2      if y = 1 then
3          x ← 0
4      y ← 1 − x
5  end
```

- Execution: potential execution starting at an initial configuration, and where configurations are followed by their „legal successors“ according to the program semantics.

  Examples: [1,1,1][2,1,1][3,1,1][4,0,1][1,0,1][5,0,1]
  [1,1,0][2,1,0][4,1,0][1,1,0]

- Full execution: execution that cannot be extended (either infinite or ending at a configuration without successors)

Verification as a language problem

- Implementation: set E of executions
- Specification:
  - subset P of the potential executions that satisfy a property , or
  - subset V of the potential executions that violate a property
- Implementation satisfies specification if :
  - $E \subseteq P$, or
  - $E \cap V = \emptyset$.
- If E and P regular: inclusion checkable with automata
- If E and V regular: disjointness checkable with automata

Verification as a language problem

- Implementation: set E of executions
- Specification:
  - subset P of the potential executions that satisfy a property , or
  - subset V of the potential executions that violate a property
- Implementation satisfies specification if :
  - $E \subseteq P$, or
  - $E \cap V = \emptyset$.
- If E and P regular: inclusion checkable with automata
- If E and V regular: disjointness checkable with automata

- How often is the case that E, P, V are regular?