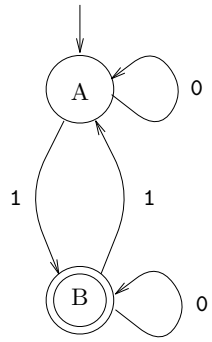
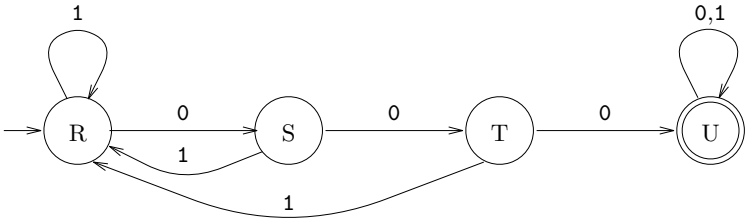
Handout 2: The Product Construction for Automata

2.1 Example. Check if the input contains an odd number of 1's. We remember in state A that the processed portion of the input contains an even number of 1's, and in state B that the processed portion of the input contains an odd number of 1's. State diagram:



2.2 Example. Check if the input contains the substring 000. We remember in state R that we have not yet seen 000 and we need to see 3 more 0's to accept; in state S, that we have not yet seen 000 and we need to see 2 more 0's to accept; in state T, that we have not yet seen 000 and we need to see 1 more 0 to accept; in state U, that we have seen 000. State diagram:



2.3 Intersection and union of finite automata. Since a language is a set of words, we can take the intersection and the union of two languages L_1 and L_2. It would be useful if we can construct finite automata for $L_1 \cap L_2$ and $L_1 \cup L_2$ from finite automata for L_1 and L_2, no matter what L_1 and L_2. For instance, this would give us a systematic way of constructing from the automata of Examples 2.1 and 2.2 an automaton that checks "if the input contains an odd number of 1's and the substring 000," as well as an automaton that checks "if the input contains an odd number of 1's or the substring 000." We cannot first run the automaton for L_1 over the input, then back up and run the automaton for L_2 over the same input, because a finite automaton cannot "back up." However, we can run both automata "in parallel," by remembering the states of both automata while reading the input. This is achieved by the *product construction*.

Let $M_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$ be two finite automata with the same input alphabet. We define a finite automaton $M_\cap$ such that $L(M_\cap) = L(M_1) \cap L(M_2)$ and a finite automaton $M_\cup$ such that $L(M_\cup) = L(M_1) \cup L(M_2)$ as follows:

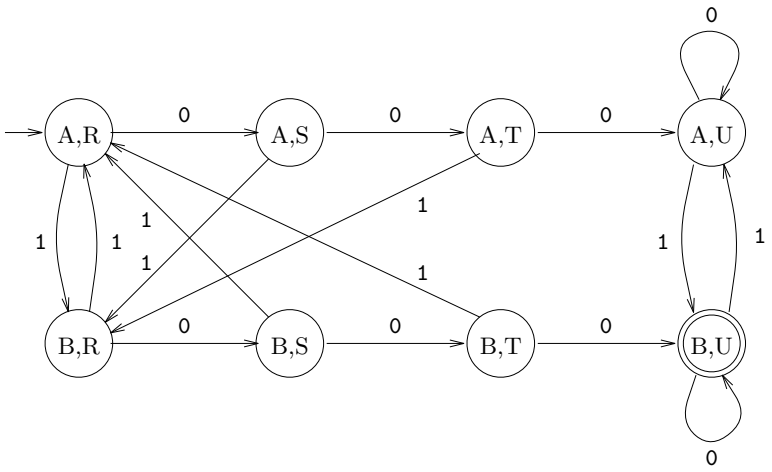
$M_\cap = (Q, \Sigma, \delta, q_0, F_\cap)$,
$M_\cup = (Q, \Sigma, \delta, q_0, F_\cup)$,

where

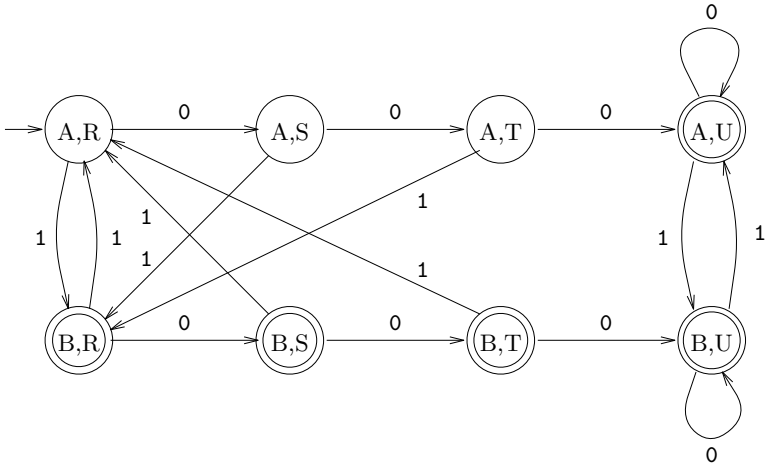
$Q = Q_1 \times Q_2$,
$\delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$,
$q_0 = (q_{01}, q_{02})$,
$(q_1, q_2) \in F_\cap$ iff $q_1 \in F_1$ and $q_2 \in F_2$,
$(q_1, q_2) \in F_\cup$ iff $q_1 \in F_1$ or $q_2 \in F_2$.

Each state in Q is a pair consisting of a state from Q_1 and a state from Q_2. In state (q_1, q_2) on input a, the automata $M_\cap$ and $M_\cup$ proceed by executing M_1 from q_1, and in parallel, executing M_2 from q_2. If M_1 has n_1 states and M_2 has n_2 states, then $M_\cap$ and $M_\cup$ each have $n_1 \cdot n_2$ states.

2.4 Example. Check if the input contains an odd number of `1`'s and the substring `000`. State diagram:



2.5 Example. Check if the input contains an odd number of `1`'s or the substring `000`. State diagram:



2.6 Merge of finite automata. For two languages L_1 and L_2, we write $L_1||L_2$ for the set of words that result from merging a word in L_1 with a word in L_2. For instance:

$\{\texttt{a,ab}\}||\{\texttt{01}\} = \{\texttt{a01,0a1,01a,ab01,a0b1,0ab1,a01b,0a1b,01ab}\}$

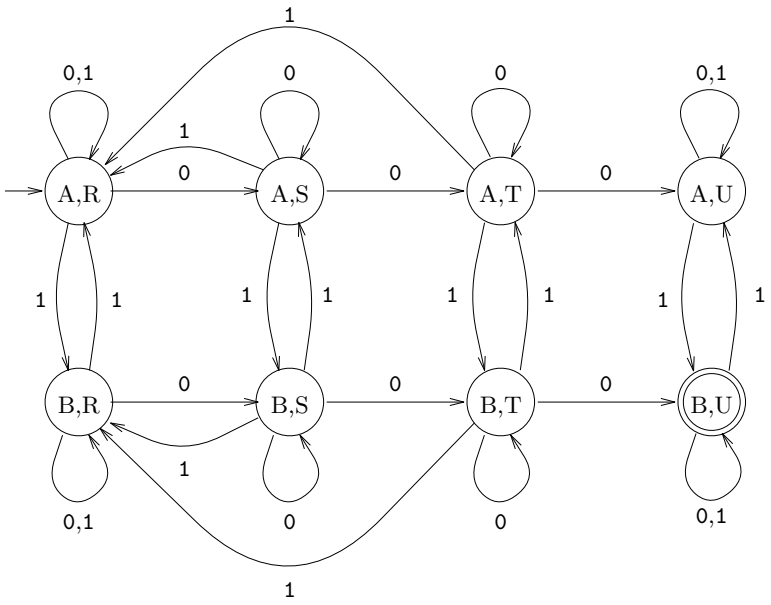
Let $M_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$ be two finite automata with the same input alphabet. We define a finite automaton $M_{||}$ such that $L(M_{||}) = L(M_1)||L(M_2)$ as follows:

$$M_{||} = (Q, \Sigma, \delta_{||}, q_0, F_\cap)$$

where

$Q = Q_1 \times Q_2$,
$\delta_{||}((q_1, q_2), a) = \{(\delta_1(q_1, a), q_2), (q_1, \delta_2(q_2, a))\}$,
$q_0 = (q_{01}, q_{02})$,
$(q_1, q_2) \in F_\cap$ iff $q_1 \in F_1$ and $q_2 \in F_2$.

In state (q_1, q_2) on input a, the automaton $M_{||}$ has two choices: it can either execute M_1 from q_1, leaving q_2 unchanged, or it can execute M_2 from q_2, leaving q_1 unchanged. Such a choice is called *nondeterminism*. State diagram for merging to the two automata from Examples 2.1 and 2.2:



2.7 Nondeterminism for finite automata. A *nondeterministic finite automaton* $M = (Q, \Sigma, \delta, q_0, F)$ has the same components as a deterministic finite automaton, except that δ: $Q \times \Sigma \to \mathcal{P}(Q)$. Equivalently, $\delta \subseteq Q \times \Sigma \times Q$; that is, nondeterministic automata have transition relations instead of transition functions. While according to a transition function, every state has precisely one successor on each input symbol, according to a transition relation, a state may have zero, one, or more successors on an input symbol. If there are zero successors, then no run results; if there are several successors, then several runs may result. The automaton can be thought of as pursuing all possibilities in parallel (say, by replicating itself), and accepting if at least one of the parallel copies of the automaton accepts. Equivalently, a nondeterministic automaton can be thought of as magically guessing a sequence of choices that will lead to an accept state if such a sequence exists.

A *run* of M is a sequence

$$p_0 \overset{a_0}{\to} p_1 \overset{a_1}{\to} \cdots \overset{a_{n-1}}{\to} p_n$$

with $p_0, \ldots, p_n \in Q$ and $a_0, \ldots, a_{n-1} \in \Sigma$ such that

(1) $p_0 = q_0$,
(2) $p_{i+1} \in \delta(p_i, a_i)$ for all $i \in [0..n-1]$.

The run *accepts* the input word $a_0 \ldots a_{n-1}$ if

(3) $p_n \in F$.

The *language* of M is

$$L(M) = \{w \in \Sigma^* \mid w \text{ is accepted by some run of } M\}.$$

In a deterministic automaton, each input word corresponds to precisely one run. In a nondeterministic automaton, each input word may correspond to zero, one, or more runs.