# Tree Languages

- **Alphabet with arity** is a finite set $\Sigma$ of operators together with a function $\rho : \Sigma \to \mathbb{N}_0$, **arity**.

- $\Sigma_k = \{a \in \Sigma \mid \rho(a) = k\}$.

- The **homogeneous tree language** over $\Sigma$ is the following inductively defined set $T(\Sigma)$ :
  - $a \in T(\Sigma)$ for all $a \in \Sigma_0$;
  - Are $b_1, \ldots, b_k$ in $T(\Sigma)$ and is $f \in \Sigma_k$, so is $f(b_1, \ldots, b_k) \in T(\Sigma)$.

Example:
$\Sigma = \{a, cons, nil\}$,
$\rho(a) = \rho(nil) = 0$, $\rho(cons) = 2$.
Some trees over $\Sigma$:
$a$, $cons(nil, nil)$, $cons(cons(a, nil), nil)$.

# Patterns, Substitutions

$V$ infinite set of variables (arity 0).

- $p \in T(\Sigma \cup V)$ is called a **pattern** over $\Sigma$,
- $p$ is **linear** if no variable occurs twice in $p$.
- A **Substitution** $\Theta$ maps variables to patterns, $\Theta : V \to T(\Sigma \cup V)$.
- $\Theta$ extended to $\Theta : T(\Sigma \cup V) \to T(\Sigma \cup V)$ by $t\Theta = x\Theta$, if $t = x \in V$ and $t\Theta = a(t_1\Theta, \ldots, t_k\Theta)$, if $t = a(t_1, \ldots, t_k)$.

Let $V = \{X\}$.
$X$, $cons(nil, X)$, $cons(X, nil)$ are patterns over $\Sigma$.

# Regular Tree Grammars

## Regular Tree Grammar (RTG)

$G = (N, \Sigma, P, S)$ consists of

- $N$, finite set of **non–terminals**,
- $\Sigma$, finite alphabet (with arity) of **terminals** (operators labeling nodes)
- $P$, finite set of **rules** $X \to s$ where $X \in N$ and $s \in T(\Sigma \cup N)$,
- $S \in N$, the start symbol.

Notions:

- $p : X \to Y$ **chain rule**,
- $p : X \to s$ has **type** $(X_1, \ldots, X_k) \to X$, if $j$-th occurrence of a non–terminal in $s$ (counted from the left) is $X_j$.
- $\tilde{s}$ results from $s$ by replacing non–terminal $X_j$ by variable $x_j$.

# Why "Regular"?

- Path words form a regular word language,
- Regular tree languages are closed under union, intersection, and complement,
- Emptiness and therefore containment are decidable.

# Example: Lists

- $G_1 = (N_1, \Sigma, P_1, L)$
- $\Sigma = \{a, cons, nil\}$
  where $\rho(a) = \rho(nil) = 0, \rho(cons) = 2$
- $N_1 = \{E, L\}$ and
- $P_1 = \{ \begin{array}{lcl} L & \to & nil, \\ L & \to & cons(E, L), \\ E & \to & a\} \end{array}$

$L(TG_1)$ is the language of linear lists of $a$'s including the empty list,
i.e. $L(G_1) = \{nil, cons(a, nil), cons(a, cons(a, nil)), \ldots\}$.

# Example: Machine Grammar

- $G_m = (N_m, \Sigma, P_m, REG)$;
- $\Sigma = \{const, m, plus, REG\}$
  where $\rho(const) = 0; \rho(m) = 1, \rho(plus) = 2$,
- $N_m = \{REG\}$
- $P_m = \{$

$$
\begin{array}{lllll}
addmc: & REG & \rightarrow & plus(m(const), REG), \\
addm: & REG & \rightarrow & plus(m(REG), REG), \\
add: & REG & \rightarrow & plus(REG, REG), \\
ldmc: & REG & \rightarrow & m(const), \\
ldc: & REG & \rightarrow & const, \\
ld: & REG & \rightarrow & REG\}
\end{array}
$$

$G_m$ describes a subset of an instruction set of a simple processor, rules are marked with names of instructions.

The first three instructions add

- the contents of a memory cell, whose address is given by a constant,
- the contents of a memory cell, whose address is in a register, resp.,
- the contents of a register

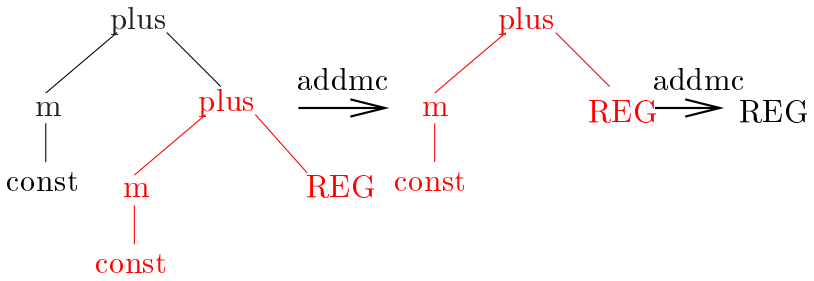
to the contents of a register and put the result into a register.

The last three rules describe load instructions, which load

- the contents of a memory cell, whose address is given by a constant,
- a constant, resp.,
- the contents of a register

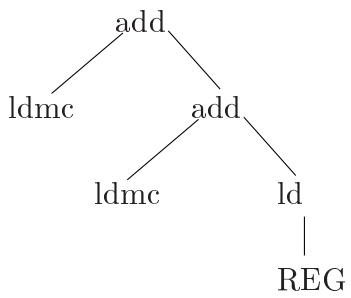
into a register.

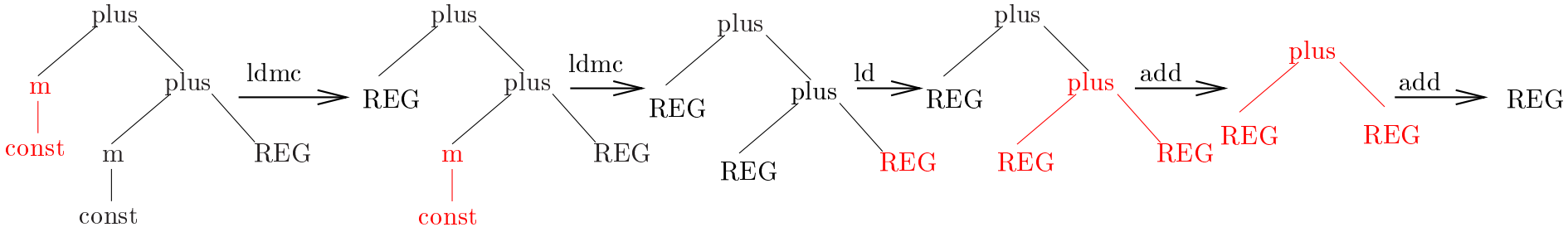
# Example Derivations




Derivation Tree

Derivation tree

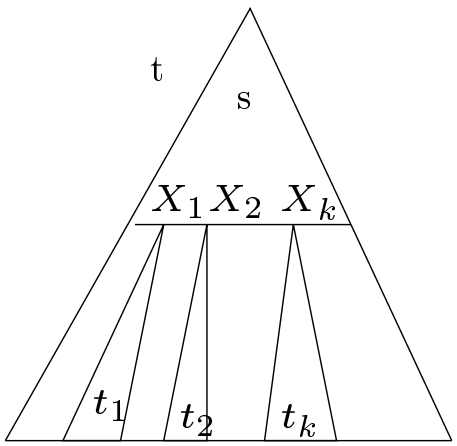


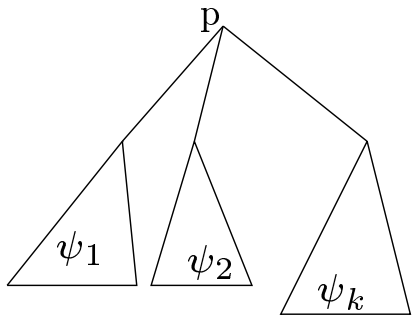



# Derivation Tree

An $X$**–derivation tree** for tree $t \in T(\Sigma \cup N)$ according to tree grammar $G$ is a tree $\psi \in T(P \cup N)$, such that

- Is $\psi \in N$, then $\psi = X = t$.
- Is $\psi \notin N$, then $\psi = p(\psi_1, \ldots, \psi_k)$ for a rule $p : X \to s \in P$ of type $(X_1, \ldots, X_k) \to X$, such that $t = \tilde{s}\{x_1/t_1, \ldots, x_k/t_k\}$ and $\psi_j$ are $X_j$-derivation trees for the $t_j$.




The **generated language** $L(TG) = \{t \in T(\Sigma) \mid \exists \psi \in T(P \cup N) : \psi \text{ is } S\text{-derivation tree for } t\}$.

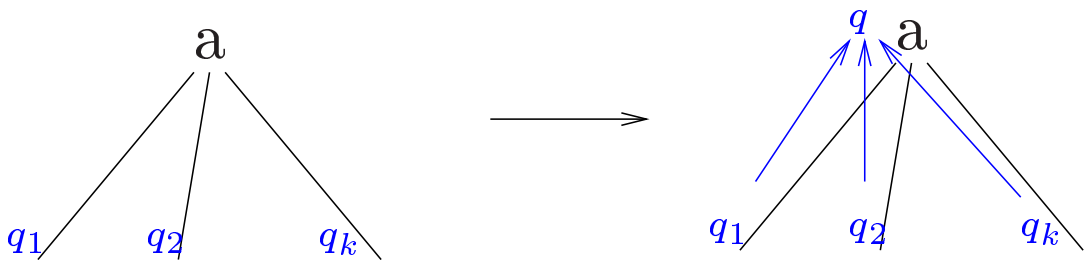
# The Tree Analysis Problem

- An instance of the **tree analysis problem** consists of an RTG $G$ and a tree $t$.
- A solution consists of the set of all derivation trees of $t$ according to $G$,
- A **Tree Analyzer** for $G$ solves the tree analysis problem for $G$ and all its trees,
- A **Tree Analyzer Generator** generates a tree analyzer for each RTG.

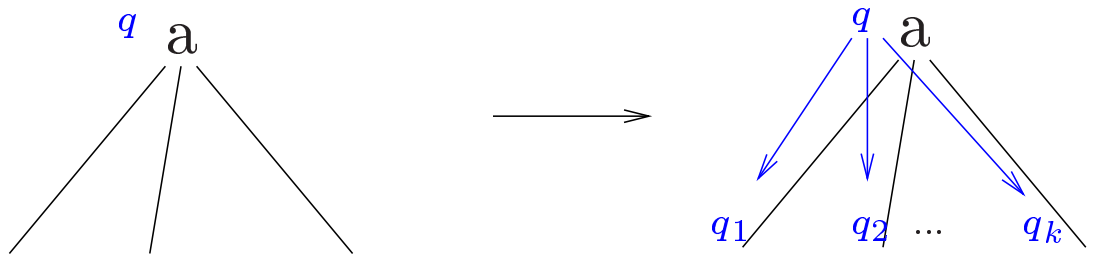
# Finite Tree Automata, Intuition

- Generalization of finite word automata to trees,
- Transitions $(q, a, q_1, \ldots, q_k)$, where
  $a \in \Sigma_k$, q state at node $n$ labeled $a$,
  $q_1, \ldots, q_k$ state at children of $n$,
- Traversal strategies,

  **bottom up:**




  **top down:**



# Finite Tree Automata, Definition

**Finite tree automaton (FTA)**
$A = (Q, \Sigma, \delta, Q_F)$, where

- $Q$, finite set of **states**,
- $Q_F \subseteq Q$, **final states**,
- $\Sigma$, input alphabet (with arity),
- $\delta \subseteq \bigcup_{j \geq 0} Q \times \Sigma_j \times Q^j$, **transition relation**.
- $A$ is **top down deterministic**, if
  - exactly one final state, and
  - at most one transition $(q, a, q_1 \ldots, q_k) \in \delta$ for all $a$ and $q$.
- $A$ is **bottom up deterministic**, if
  at most one transition $(q, a, q_1 \ldots q_k) \in \delta$ for all $a$ and all $q_1, \ldots, q_k$.
  In this case, we write $\delta$ as partial function:
  $\delta : \bigcup_{j \geq 0} \Sigma_j \times Q^j \to Q$

# Computation

- $A$ annotates the nodes with states;
  hence new alphabet $\Sigma \times Q = \{\langle a, q\rangle \mid a \in \Sigma,\ q \in Q\}$,
  where $\rho(\langle a, q\rangle) = \rho(a)$.

- $q$-**computation** $\phi$ of $A$ on tree $t = a(t_1, \ldots, t_m)$:
  a tree $\langle a, q\rangle(\phi_1, \ldots, \phi_m) \in T(\Sigma \times Q)$, where
  $\phi_j$ are $q_j$-computations for the $t_j$, $j = 1, \ldots, m$,
  $(q, a, q_1 \ldots q_m)$ is a transition.

- Is $q \in Q_F$, then $\phi$ is **accepting**.

- The language $L(TA)$ consists of the trees with accepting computations.

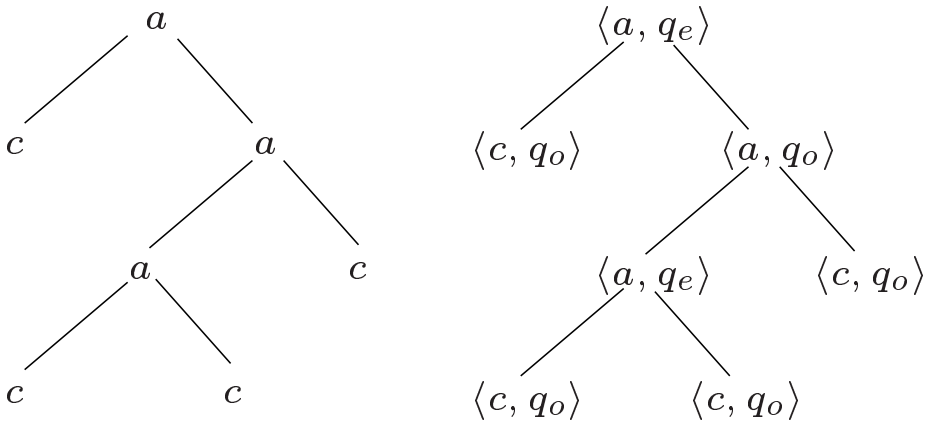
- A state resp. transition is **superfluous** if it does not occur in any accepting computation.

# Example Computation

DFTA $A_b = (Q_b, \Sigma_b, \delta_b, Q_{F,b})$ with
states $Q_b = \{q_e, q_o\}$,
alphabet $\Sigma_{b,0} = \{c\}$ and $\Sigma_{b,2} = \{a\}$,
final states $Q_{F,b} = \{q_e\}$
transitions: $\delta_b = \{\ (q_o, c)$
$(q_e, a, q_o, q_o)$
$(q_o, a, q_e, q_o)$
$(q_o, a, q_o, q_e)$
$(q_e, a, q_e, q_e)\}$

Accepts trees with even number of $c$'s.

Tree and $q_e$–computation



# Determinism – Non–determinism

- Bottom up NFTAs and Top down NFTAs are equivalent,

- Bottom up DFTAs and Top down DFTAs are not equivalent;
  example language cannot be recognized by top down DFTA.

- NFTAs are equivalent to bottom up DFTAs (powerset construction).

(Bottom up) DFTA:

- At most one computation for each tree,

- At most one state at each node,

- $\delta$ extended to a partial function $\delta : T(\Sigma) \to Q$ by: $\delta(t) = \delta(a, \delta(t_1) \ldots \delta(t_k))$, if $t = a(t_1, \ldots, t_k)$.

- $\delta(t) = q$ iff there is a $q$–computation for $t$.

# Generating Pattern Matchers

- Be $\tau$ a linear pattern in $B(\Sigma \cup V)$. The FTA $A_\tau$ recognizes whether $\tau$ matches a given input tree.

- Intuitively: If a pattern matches a subtree there is a region near the root of the tree where the pattern 'covers' the subtree precisely, i.e. a region where the operators of the subtree correspond precisely to the operators of the pattern. Outside this region $A_\tau$ takes an unspecific state $\perp$.

- We ignore the numbering of variables and replace them by $\perp$, i.e. suppose that $\tau \in B(\Sigma \cup \{\perp\})$.

- $A_\tau = (Q_\tau, \Sigma, \delta_\tau, Q_{\tau F})$, where $Q_\tau = \{s \mid s \text{ subtree of } \tau\} \cup \{\perp\}$, $Q_{\tau F} = \{\tau\}$ and $\delta_\tau$:
  - $(\perp, a, \perp \ldots \perp) \in \delta_\tau$
  - if $s \in Q_\tau$ and $s = a(s_1, \ldots, s_k)$, then $(s, a, s_1 \ldots s_k) \in \delta_\tau$.

- A $q$-computation of $A$ corresponds to a $q$-derivation tree.

# Generating Tree Parsers

The generation (and the explanation) process:

Input: $G$

1. Generate NFTA $A_G$,

2. Apply powerset construction to obtain DFTA $P(G)$.

Later: Consider variant with costs.

# $A_G$, Definition

$A_G = (Q_G, \Sigma, \delta_G, \{S\})$, where

- $Q_G = N \cup \{s' \mid \exists (X \to s) \in P,$
  where $s'$ is proper subpattern of $s\}$.

- Transition relation $\delta_G$: transitions of the forms
  $\{(s, a, s_1 \ldots s_k) \mid s = a(s_1, \ldots, s_k) \in Q_G\}$ and
  $\{(X, a, s_1 \ldots s_k) \mid \exists (X \to s) \in P$ and
  $s = a(s_1, \ldots, s_k)\}$.

Problem with chain rules:

- $A_G$ would have to "step on the spot" doing chain reductions.
  However, $A_G$ has to consume at least one terminal per step,

- Chain reductions are precomputed and integrated into $\delta$.

$$\begin{aligned}\delta_G := \quad & \{(s, a, s_1 \ldots s_k) \mid s = a(s_1, \ldots, s_k) \in Q_G\} \cup \\ & \{(X, a, s_1 \ldots s_k) \mid \exists (X' \to s) \in P : \\ & \exists X\text{–derivation tree for } X' \\ & \text{ and } s = a(s_1, \ldots, s_k)\}\end{aligned}$$

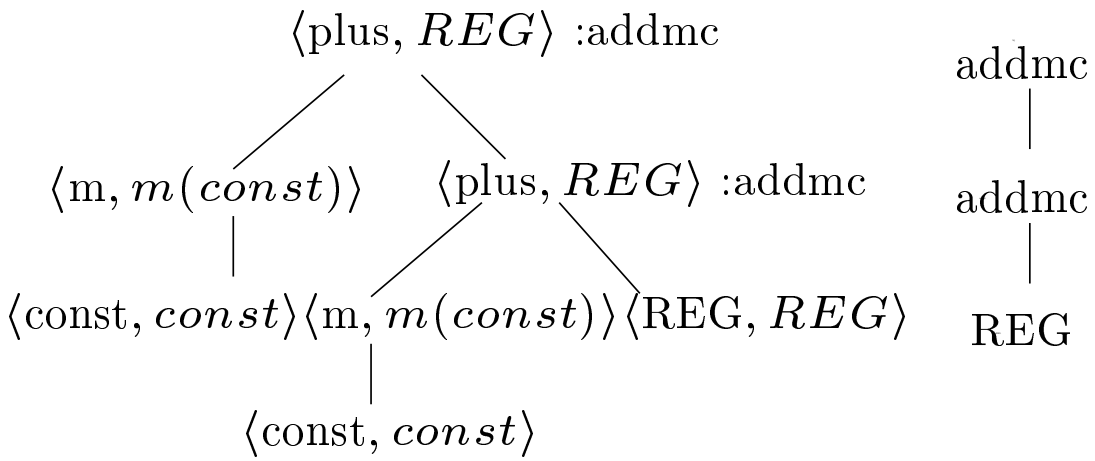
# Example $A_{G_m}$

$A_{G_m} = (Q_{G_m}, \Sigma_{G_m}, \delta_{G_m}, Q_{F,G_m})$ for $G_m$ has the state set
$Q_{G_m} = \{const, REG, m(const), m(REG)\}$

and the transitions

$$\begin{aligned}\delta_{G_m} = \{\ & (const, const, \epsilon)\\ & (REG, const, \epsilon)\\ & (REG, REG, \epsilon)\\ & (m(const), m, const)\\ & (REG, m, const)\\ & (m(REG), m, REG)\\ & (REG, plus, m(const)\ REG)\\ & (REG, plus, m(REG)\ REG)\\ & (REG, plus, REG\ REG)\}\end{aligned}$$

# Example Computation of $A_{G_m}$



# Properties

$G$ RTG and $t$ input tree.

- Exists $X$–derivation tree for $t$ according to $G$ iff exists an $X$–computation for $t$ in $A_G$.

  In particular: $L(G) = L(A_G)$.

# Principle of the Powerset Construction

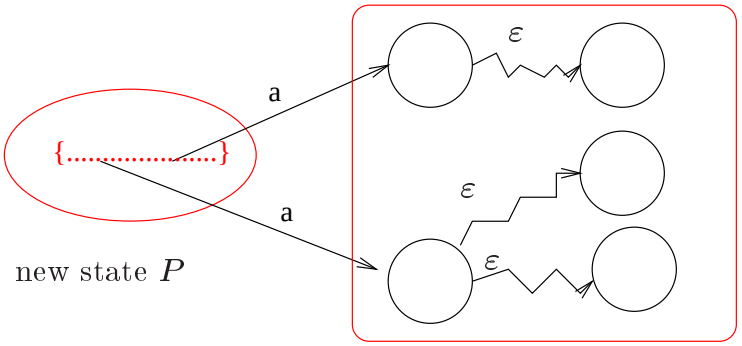
**Finite Word Automata:**

old states $q_1, q_2$ and word $w$ such that $(q_0, w) \vdash_M^* (q_1, \varepsilon)$ and $(q_0, w) \vdash_M^* (q_2, \varepsilon)$ $\Longrightarrow$ $\exists$ new state $Q$ such that $q_1, q_2 \in Q$ and $\delta_d(q_d, w) = Q$

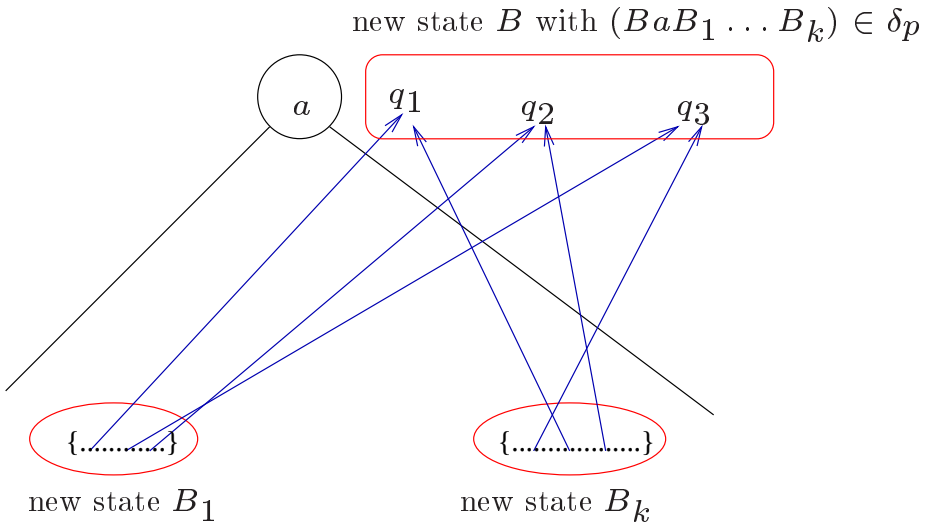
**Finite Tree Automata:**

old states $q_1, q_2$ and tree $t$ such that $\exists q_1 -$ and $q_2 -$ computations for $t$ $\Longrightarrow$ $\exists$ new state $B$ such that $q_1, q_2 \in B$

## Word automata:




## Tree automata:



# Powerset Construction

Powerset automaton $P(A)$ is built iteratively,
$Q_p^{(n)}$ and $\delta_p^{(n)}$ occur in computations on trees of height $\leq n-1$.

Let $A = (Q, \Sigma, \delta, Q_F)$ be a NFTA.
Its **powerset automaton** is the DFTA
$P(A) = (Q_p, \Sigma, \delta_p, Q_{p,F})$ where

- $Q_p = 2^Q$,
- $Q_{p,F} := \{B \in Q_p \mid B \cap Q_F \neq \emptyset\}$,
- states and transitions are computed in the iteration: $Q_p := \bigcup_{n \geq 0} Q_p^{(n)}$ and $\delta_p := \bigcup_{n \geq 0} \delta_p^{(n)}$, where:
  - $Q_p^{(0)} = \emptyset$;
  - Be $n > 0$. For $a \in \Sigma_k$ and $B_1, \ldots, B_k \in Q_p^{(n-1)}$ let $B := \{q \in Q \mid \exists q_1 \in B_1, \ldots, q_k \in B_k : (q, a, q_1 \ldots q_k) \in \delta\}$.
    Is $B \neq \emptyset$,
    then $B \in Q_p^{(n)}$ and $(B, a, B_1 \ldots B_k) \in \delta_p^{(n)}$.

# Example

The powerset automaton for $A_{G_m}$ has the state set $Q_{G_m} = \{q_1, q_2, q_3, q_4\}$ where

$$\begin{array}{lcl} q_1 & = & \{REG\} \\ q_2 & = & \{const, REG\} \\ q_3 & = & \{m(REG)\} \\ q_4 & = & \{m(const), REG, m(REG)\} \end{array}$$

and the transition function $\delta_{G_m}$:

| $state$ | $operator$ | $children\ state(s)$ | |
|---|---|---|---|
| $q_1$ | $REG$ | $\varepsilon$ | |
| $q_2$ | $const$ | $\varepsilon$ | |
| $q_3$ | $m$ | $q_1$ | |
| $q_4$ | $m$ | $q_2$ | |
| $q_1$ | $plus$ | $q_1$ | $q_1$ |
| $q_1$ | $plus$ | $q_4$ | $q_1$ |
| $q_1$ | $plus$ | $q_3$ | $q_1$ |

# Properties

1. For each $t \in T(\Sigma)$:
   - Is $\delta_p(t)$ defined, then $\delta_p(t) = \{q \mid \exists q\text{–computation on } t\}$.
   - Is $\delta_p(t)$ undefined, then there is no $q \in Q$ with a $q$–computation of $A$ for $t$.
   - $\delta_p(t) \cap N = \{X' \in N \mid \exists X'\text{–derivation tree for } t\}$.

2. $L(A) = L(P_p(A))$.

3. For each state $B \in Q_r$ there exists a tree $t$, such that $\delta_p(t) = B$.

# Tree Automata with Costs

- Machine grammars usually are ambiguous. Thus among all possible derivation trees the cheapest should be selected.

- Required: cost measure for derivation trees.

- Assume: rules of the grammar are annotated with cost functions.

- For each rule $p$ of type $(X_1, \ldots, X_k) \to X$: $k$-ary function $C(p) : \mathbb{N}_0^k \to \mathbb{N}$.

- The cost measure $C$ can be extended to a function which assigns a cost $C(\Psi)$ to every derivation tree $\Psi$.

- $C$ is called *monotonic*, or *additive*, if $C(p)$ is monotonic, or of the form $C(p) = c_p + x_1 + \ldots + x_k,\ c_p \in \mathbb{N}_0$, for all $p \in P$.

# Tree Automata with Costs (c'ed)

- Examples:
  - number of required clock cycles
  - number of referenced memory cells

- Translation of the cost annotation $C$ of the grammar $G$ into a cost annotation $C^*$ of the associated automaton $A_G$. Assumption: $C$ additive $\Rightarrow$ cost of each rule can be described by a constant, i.e. $C' : P \to \mathbb{N}$. The cost function $C^*$ for the transitions of $A_G$:
  - If $\tau = (s, a, s_1 \ldots s_k)$ with $s = a(s_1, \ldots, s_k)$, then $C^*(\tau) = 0$.
  - if $\tau = (X, a, s_1 \ldots s_k)$ then $C^*(\tau)$ corresponds to the minimal cost of an $X$-derivation tree for $a(s_1, \ldots, s_k)$.

- Bottom-up computation: For each node $(a, B)$ of a computation $\Phi$ of the subset automaton $A_r$ we tabulate the costs $c_q$ and the transitions $d_q$ for all $q \in B$. Let $t$ be the subtree of the input tree for a node of $\Phi$. Then $c_q$ is the cost of a cheapest $q$-computation of $t$ and $d_q$ is the corresponding transition of $A$.

# Tree Automata with Costs: Example

Be $G_m = (N_m, \Sigma, P_m, REG)$ the tree grammar of the previous examples with cost annotations:

$$
\begin{array}{lllll}
addmc: & REG & \rightarrow & plus(m(const), REG) & Cost: 3 \\
addm: & REG & \rightarrow & plus(m(REG), REG) & Cost: 3 \\
add: & REG & \rightarrow & plus(REG, REG) & Cost: 2 \\
ldmc: & REG & \rightarrow & m(const) & Cost: 2 \\
ldc: & REG & \rightarrow & const & Cost: 1 \\
ld: & REG & \rightarrow & REG & Cost: 1
\end{array}
$$

The NFTA $A = (Q, \Sigma, \delta, Q_F)$ has the states $Q = \{const, REG, m(const), m(REG)\}$ and the transitions $\delta = \{$

$$
\begin{array}{ll}
(const, const, \epsilon) & Cost: 0 \\
(REG, const, \epsilon) & Cost: 1 \\
(REG, REG, \epsilon) & Cost: 0 \\
(m(const), m, const) & Cost: 0 \\
(REG, m, const) & Cost: 2 \\
(m(REG), m, REG) & Cost: 0 \\
(REG, plus, m(const)\ REG) & Cost: 3 \\
(REG, plus, m(REG)\ REG) & Cost: 3 \\
(REG, plus, REG\ REG) & Cost: 2\}
\end{array}
$$

# Powerset Construction with Costs

Usually for machine grammars $G_m = (N_m, \Sigma, P_m, S)$ the cost differences of $X$ derivation trees with minimal costs are bounded by a constant. The finite many cost differences can be directly integrated in the states of the subset automaton $A_C = (Q_C, \Sigma, \delta_C, Q_{CF})$. A cost difference is assigned to each reachable state $q \in Q$, i.e. $B \subseteq \{(q, d) \mid q \in Q \wedge d \in \mathbb{N}_0\}$. For $(q, d) \in B$ the value $d$ describes the cost difference between a $q$-computation of $A$ to a cheapest computation.

# Powerset Construction with Costs (c'ed)

Be $A = (Q, \Sigma, \delta, Q_F)$ a finite tree automaton and $C : \delta \to \mathbb{N}_0$ a cost function assigning to each transition from $\delta$ a cost in $\mathbb{N}_0$. The associated (reduced) subset automaton with integrated cost is $P_C(A) = (Q_C, \Sigma, \delta_C, Q_{CF})$ with $Q_{CF} = \{B \in Q_C \mid (q, d) \in B \wedge q \in Q_F\}$ whose states and transitions are computed iteratively by $Q_C = \bigcup_{n \geq 0} Q_C^{(n)}$, and $\delta_C = \bigcup_{n \geq 0} \delta_c^{(n)}$ where

- $Q_C^{(0)} = \emptyset$
- Be $n > 0$. For $a \in \Sigma_k$ and $B_1, \ldots, B_k \in Q_C^{(n-1)}$: $B := \{(q, d) \mid \exists (q_1, d_1) \in B_1, \ldots, (q_k, d_k) \in B_k$ and $\tau = (q, a, q_1 \ldots q_k) \in \delta$ so that $d = C(\tau) + d_1 + \ldots + d_k$ is minimal $\}$.

  If $B \neq \emptyset$, $norm(B) \in Q_C^{(n)}$ and $(norm(B), a, B_1 \ldots B_k) \in \delta_c^{(n)}$ where $norm(B) = \{(q, (d - \epsilon)) \mid (q, d) \in B\}$, $\epsilon = \min\{d \mid (q, d) \in B\}$.