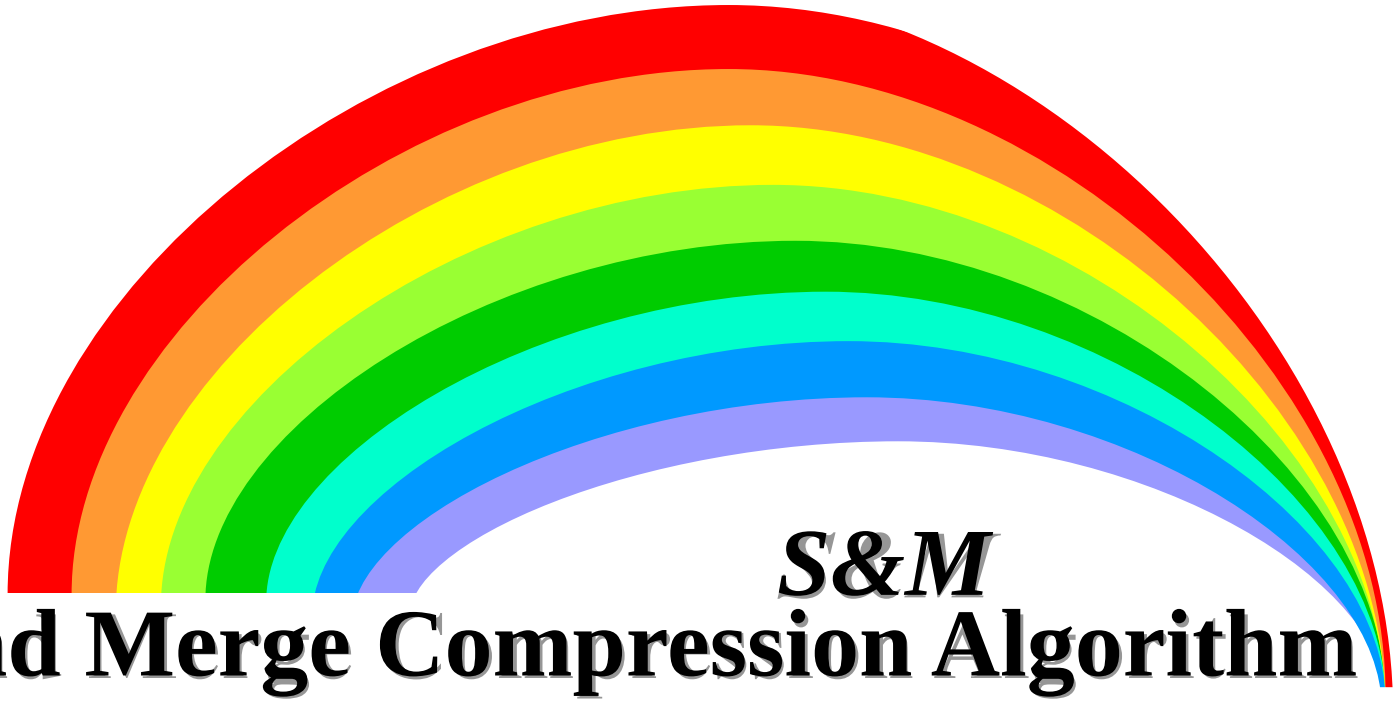


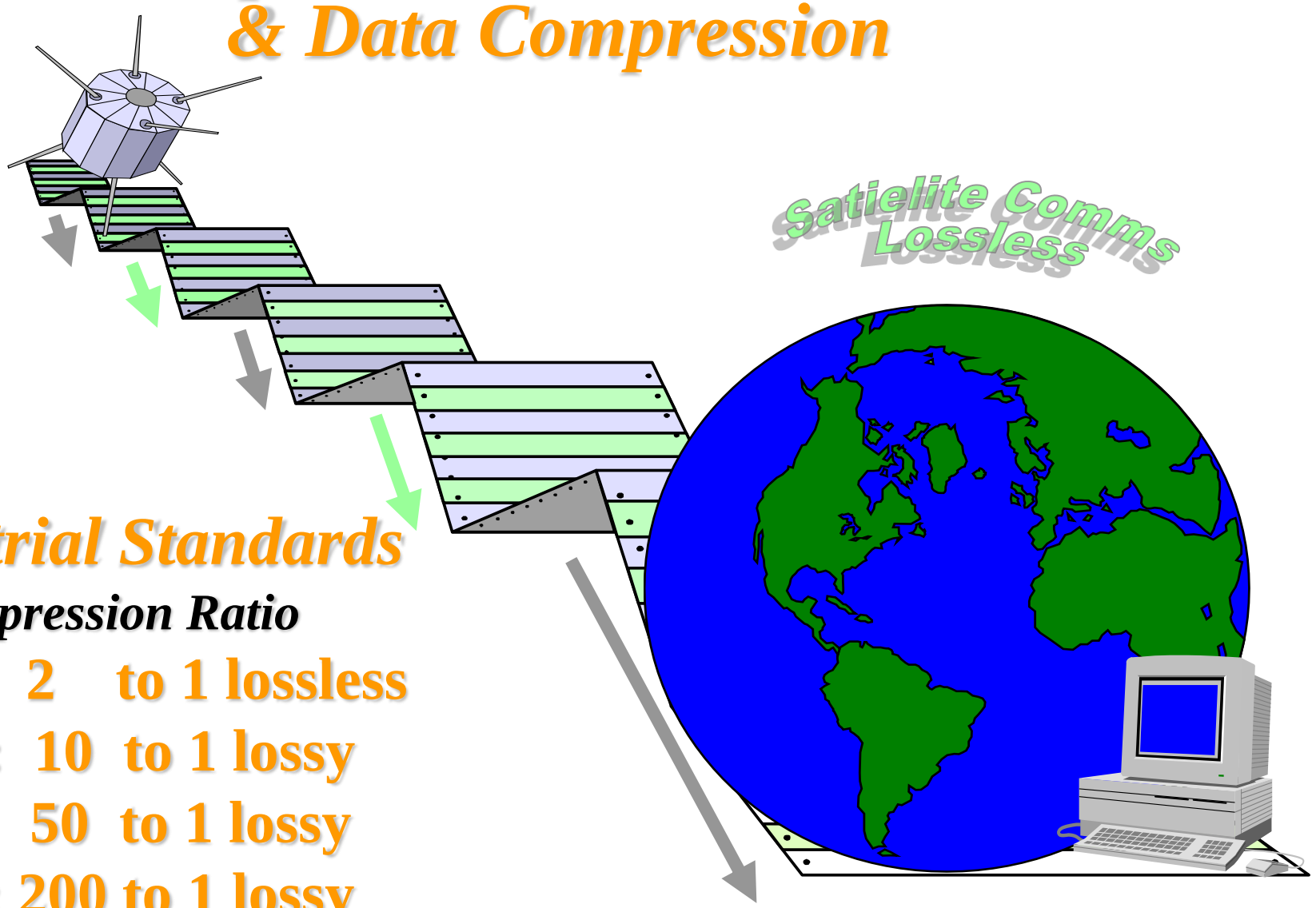
S&M
Split and Merge Compression Algorithm

By

Abdullah Hashim

Compression - Overall -

IT Age
Computers, Communications & Data Compression



Industrial Standards

Compression Ratio

Text : 2 to 1 lossless

Sound: 10 to 1 lossy

Image: 50 to 1 lossy

Video : 200 to 1 lossy

Compression Fundamentals
-Information Theory-

Hartley (1928). (*Transmission of Information*)

Shannon (1948). (*Information Theory*)

Information content (I) in bits of a symbol (α_i) with a probability (p_i) is given by the expression:

$$I_i = \log_2(1 / p_i) = - \log_2(p_i) \quad \text{bits}$$

The measure of the average information per symbol of N symbols source (s_i) is called the entropy (H) of the source is given by the following expression:

$$H = \sum_{i=0}^{N-1} (p_i I_i) \quad \text{bits per symbol}$$

Compression Fundamentals

Examples of Data Source Entropies

Sample Type	Sample Entropy	Comments
English Text	**4.03 bits per symbol**	**Shannon (1951)**
Portuguese Text	**3.92 bits per symbol**	**Manfrino (1969)**
C++ Code	**5.29 bits per symbol**	**Measured**
Executable Code	**5.80 bits per symbol**	**Measured**

Source is called memoryless if

i^{th+1} event is independent of the i^{th} event.

Compression Fundamentals

Variable Length Coding

When the average codeword length $L_{average}(\alpha_i)$ of source of (N) symbols equal to the source entropy the code is said to be ***optimal code***:

$$L_{average}(\alpha_i) = \sum_{i=0}^{N-1}[p_i\, L(\alpha_i)] \Rightarrow \sum_{i=0}^{N-1}(p_i\, I_i) = H$$

Compression Ratio $R = \log_2(N) / L_{average}(a_i)$

Prefix code consists of comma less unique codewords, i.e. no codeword may be a prefix of any other codeword

Note: *R* decreases to a great extent with slight changes of probability set p_i from that of the assumed values.

To ensure robustness against probability set p_i variation, codewords length are bounded to a given value say (L), where $L < (N - 1)$, N-1 is the longest possible length of codewords.

Compression Fundamentals

Entropy of a binary source with two symbol α and θ

Note:

when: $p_\alpha = 0$ **and**

$p_\theta = 1$

the entropy is minimum

$H_{min} \rightarrow 0$

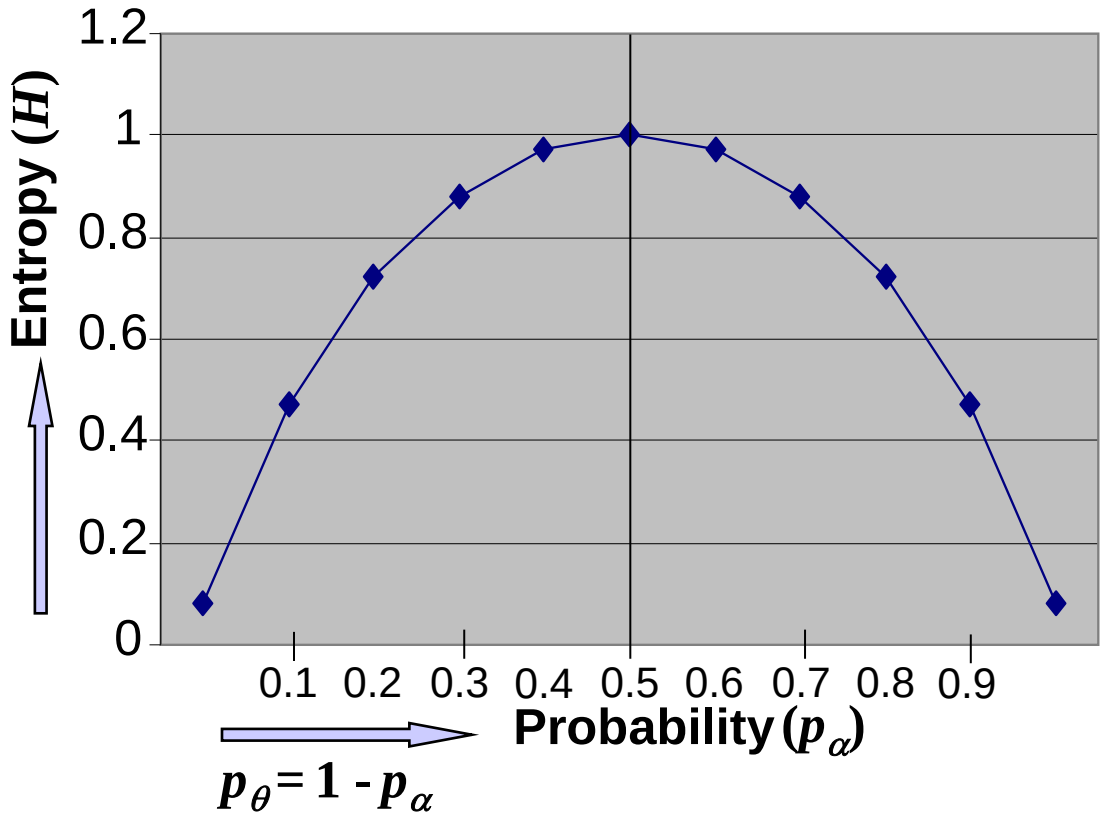
when: $p_\alpha = p_\theta = 1/N = 1/2$

the entropy is maximum

$H_{max} = log_2(N)$

where N **is the number of symbols in the source**

$$0 < H \leq log_2(N)$$



Note:

Source with N **equiprobable symbols is called a random source, and its entropy is equal to its symbol codeword length of** $[log_2(N)]$**. Random source is coded optimally with equal length binary codewords.**

Compression Fundamentals

Practical implementation of Huffman code

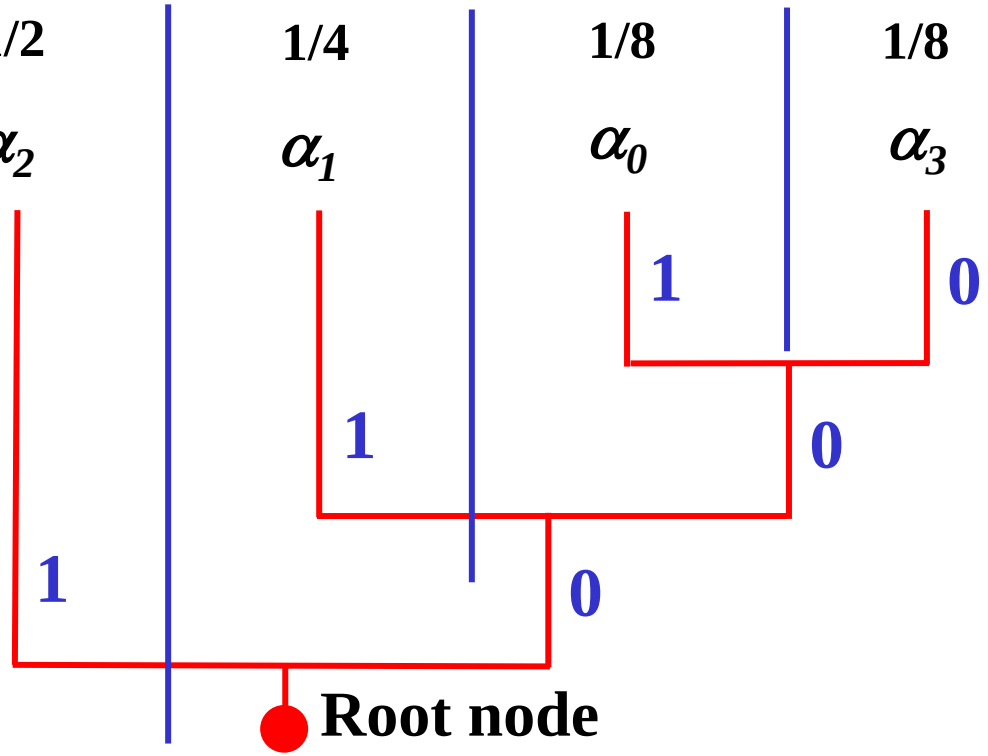
Consider a message of N symbols, α_0, α_1, α_2 and α_3, N=4, symbols probabilities are: $p(\alpha_2) = 1/2$, $p(\alpha_1) = 1/4$, $p(\alpha_0) = 1/8$, $p(\alpha_3) = 1/8$.

Probabilities 1/2 1/4 1/8 1/8

Symbols α_2 α_1 α_0 α_3

Coding

$\alpha_0 \rightarrow 001$, $\alpha_1 \rightarrow 01$,
$\alpha_2 \rightarrow 1$, and $\alpha_3 \rightarrow 000$



The longest possible length of codeword is: (N-1) = 3

Compression Ratio R = $\log_2(N)$ / $L_{average}(\alpha_i)$ = 2.0 / 1.75 = 1.1428

Compression Fundamentals

Source With Memory

- Real information sources exhibit local dependence between message symbols.
- Conditional Probability is given by:

$$P_{i/j} = P_i \cdot P_{j/i}$$

$$I_{i/j} = I_i - I_{j/i}$$

$$H_{i/j} = H_i - H_{j/i}$$

$$H_{1st\ order} \geq H_{2nd\ order} \geq \ldots H_{ith\ order} \geq H_{ith+1\ order} \geq \ldots$$

for very large value of i; $H_{ith\ order} \rightarrow 0$

Compression Fundamentals

Source With Memory

Pair (α_i, α_j)	$P_{j/i}$	I_j (bits)	$I_{j/i}$ (bits)
(e,)	**0.0341**	**2.77**	**1.77**
(,t)	**0.0264**	**3.76**	**2.48**
(t,h)	**0.0239**	**4.55**	**1.63**
(h,e)	**0.0223**	**3.11**	**1.94**
(s,)	**0.0197**	**2.77**	**1.57**

$$H_{1st\ order} \geq H_{2nd\ order} \geq \ldots H_{ith\ order} \geq H_{ith+1\ order} \geq \ldots$$

for very large value of i; $H_{ith\ order} \rightarrow 0$

Compression Fundamentals

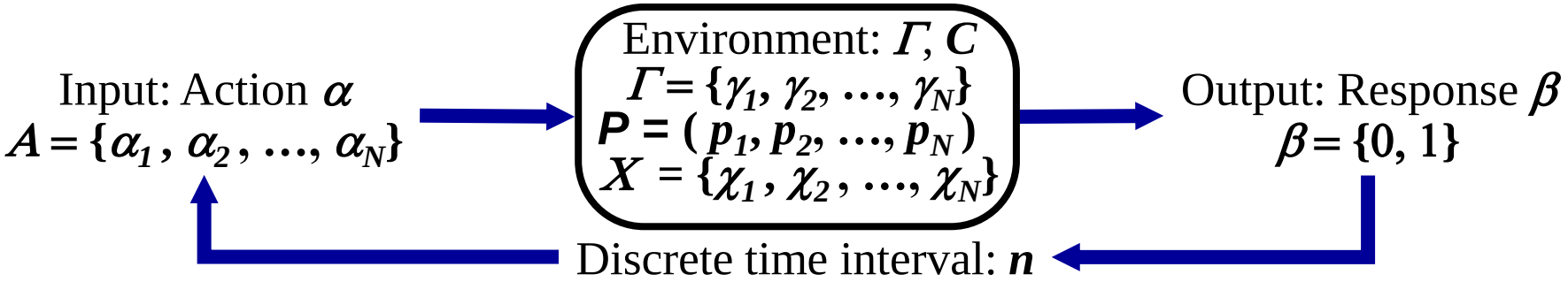
Sample entropy per symbol

Sample type	1st order H_i	Joint $H_{i,j}$	Conditional $H_{j/i}$
Arabic	**4.21**	**3.99**	**3.77**
English	**4.03**	**3.67**	**3.32**
TV Signal	**4.39**	**3.15**	**1.91**
Average	**4.21**	**3.61**	**3.00**

$$H_{1st\ order} \geq H_{2nd\ order} \geq \ldots H_{ith\ order} \geq H_{ith+1\ order} \geq \ldots$$

for very large value of i; $H_{ith\ order} \rightarrow 0$

Stochastic Learning Automaton
-The Concept-

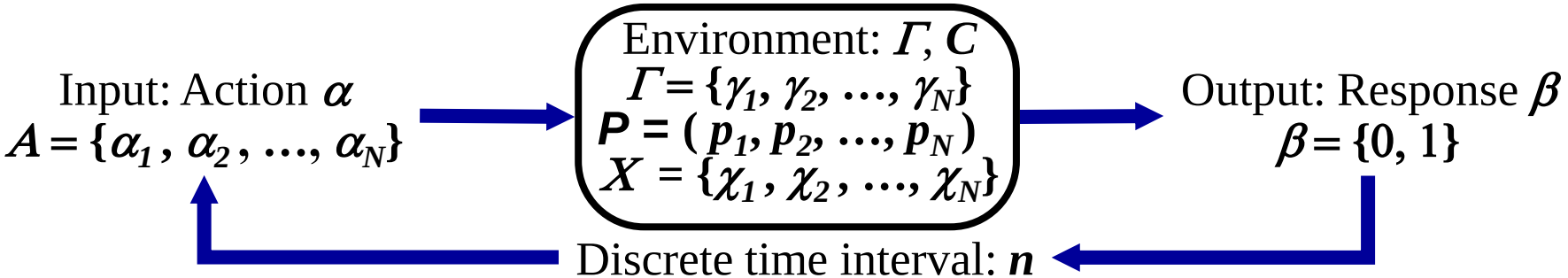


Action: α is random variable of a stationary environment Θ over a language with alphabet set A. The prior probabilities distribution and the statistical parameters of alphabet A are not known explicitly.

Output: For every action α_i, the environment, provides an output response: $\beta = \{0, 1\}$, the response is therefore, either favorable **(0)** or unfavorable, "penalty response", **(1)** .

Environment: γ is a random variable of a set of alphabet Γ, where $\Gamma = A$. However, the state probability vector of set Γ, $P = (p_1, p_2, \ldots, p_N)$ is determined by a reward-penalty function of (β). X is the penalty probability set: $X = \{\chi_1, \chi_2, \ldots, \chi_N\}$, where χ_i is the probability that action α_i, will result in unfavourable "penalty" response $(\beta_i = 1)$ from the environment.

Stochastic Learning Automaton
-The Strategy-



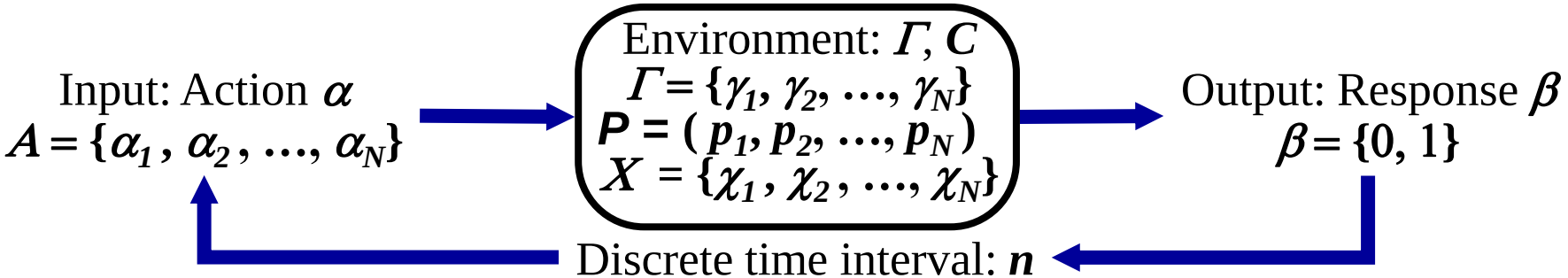
Action: At every discrete interval n, $(n = 1, 2, ..., L)$, where L is the size of the input string, the action input to the environment a single character $\alpha(n)$.

Environment: The environment select randomly a single character $\gamma(n)$. If character $\gamma(n) \neq \alpha(n)$ then, it respond with "penalty response" $\beta = 1$. The n-th interval penalty probability χ_i is mathematically define by the expression: $\chi_i = \mathbf{Prob}\{\beta(n) = 1 \mid \alpha(n) = \alpha_i\}$. $\chi_{min} = \mathbf{Min}_i\{\chi_i\}$.

Transition: The probability of each character in set Γ is updated by a pre-defined updating scheme. The scheme should ensure expediency and asymptotic convergence of the environment probability vector P to that of the real probability distribution of the actions.

A variety of linear, nonlinear and hybrid schemes exists for stochastic learning automata [Narendra 1989].

Stochastic Learning Automaton
-The Norms-



Average Penalty: Average penalty for a given action $\alpha(n)$ is the quantity $M(n)$ defined by the expression: $M(n) = \sum_{i=1}^{N} \chi_i p_i(n)$.

Pure-Chance Automation: When $p_1(0) = p_2(0) =, \ldots, = p_N(0) = 1/N$, $M(n)$ is a constant denoted by $M_0 = \frac{1}{N} \sum_{i=1}^{N} \chi_i$. This condition is known as pure-chance automation.

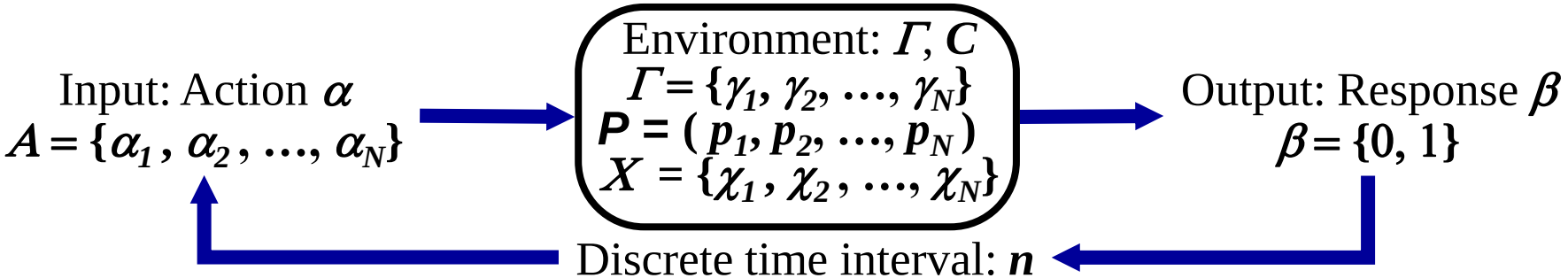
Expediency: A learning automaton is ***expedient*** if $\lim_{n \to \infty} E[M(n)] < M_0$, and said to be ***absolutely expedient*** if: $E[M(n+1)] < E[M(n)]$.

Optimality: A learning automaton is said to be ***optimal*** if:

$\lim_{n \to \infty} E[M(n)] = c_{\min}$ and said to be ***e-optimal*** if: $\lim_{n \to \infty} E[M(n)] = c_{\min} + e$.

Where e is arbitrarily small positive number.

Stochastic Learning Automaton
-The Norms-



Mean Square Errors: If action α_i is equiprobable random variable, then; at interval n, the mean square error $\varepsilon(n)$ is given by the expression:

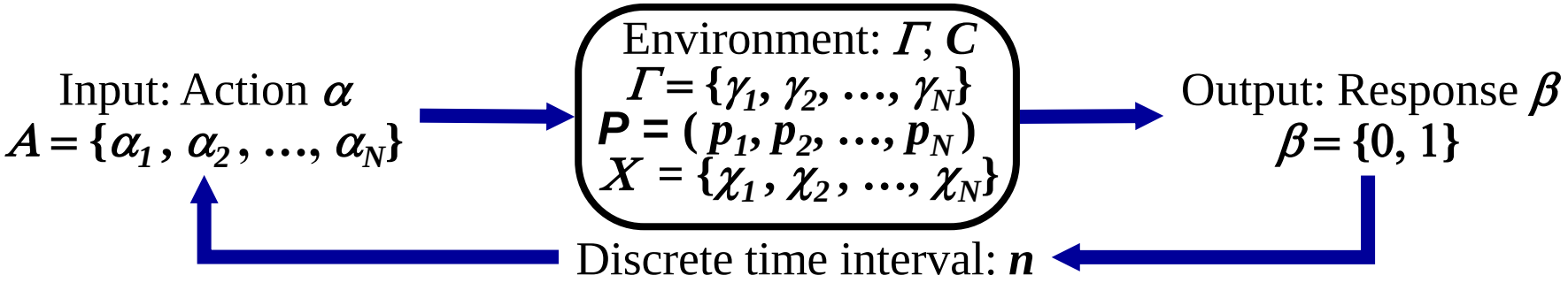
$$\varepsilon(n) = \frac{1}{N}\sum_{i=1}^{N}\left(p_i(n) - \frac{1}{N}\right)^2 = \frac{1}{N}\sum_{i=1}^{N} p_i(n)^2 - \frac{2}{N^2}\sum_{i=1}^{N} p_i(n) + \frac{1}{N^2}$$

Variable Quantity

$$= \frac{1}{N}\sum_{i=1}^{N} p_i(n)^2 - \frac{1}{N^2}$$

The ***MSE variable*** $(Q(n)) = \sum_{i=1}^{N} p_i(n)^2$, then $Q(n)$ has a maximum value of **1** when one action has a probability of **1** and all other $(N\text{-}1)$ actions have probability **0**, and a minimum value when all N actions are equiprobable. An equivalent condition to the state vector $p(n)\}$ converging to $(1/N, 1/N, \ldots, 1/N)$ is that the sum $Q(n)$ converges to $1/N$, its minimum value.

Stochastic Learning Automaton
-The Norms-



Let p_{max} denote the maximum of the component of the state probabilities P:

$$Q(n) = \sum_{i=1}^{N} (p_i^2)$$

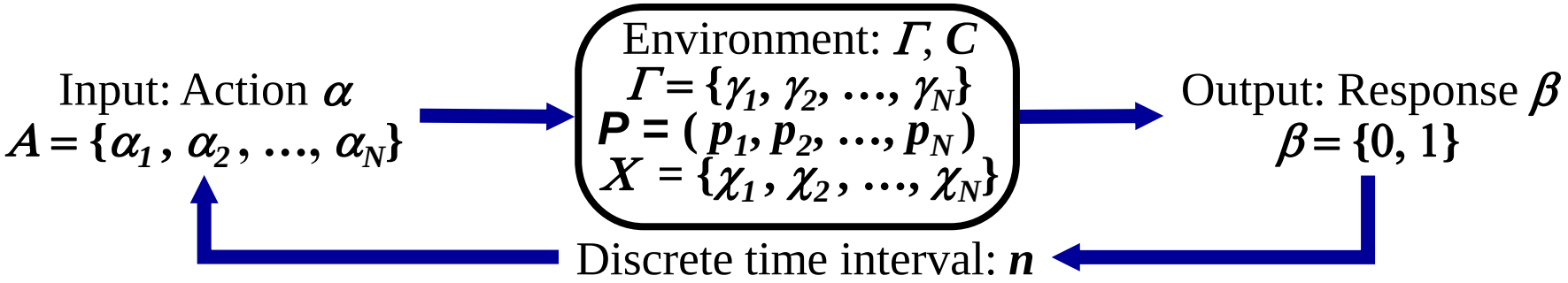
$$Q(n) <= \sum_{i=1}^{N} (p_{max} \cdot p_i)$$

$$<= p_{max} \cdot (p_1 + p_2 + \ldots + p_N) <= p_{max} \cdot 1 <= p_{max}$$

$$Q(n) \leq \text{Max}\,(p_i) \leq p_{max}$$

$$\sum_{i=1}^{N} (p_i^3) \leq p_{max} \cdot \sum_{i=1}^{N} (p_i^2) \leq p_{max} \cdot Q(n)$$

Stochastic Learning Automaton
-The Norms-



Mean Square Errors:

$$\varepsilon(n) = \frac{1}{N}\sum_{i=1}^{N}\left(p_i(n) - \frac{1}{N}\right)^2 = \frac{1}{N}\sum_{i=1}^{N} p_i(n)^2 - \frac{1}{N^2}$$

When $\varepsilon(n) = 0$; $\sum_{i=1}^{N}(p_i(n) = \frac{1}{N}$

Expediency: A learning automaton is said to be ***absolutely expedient*** if:

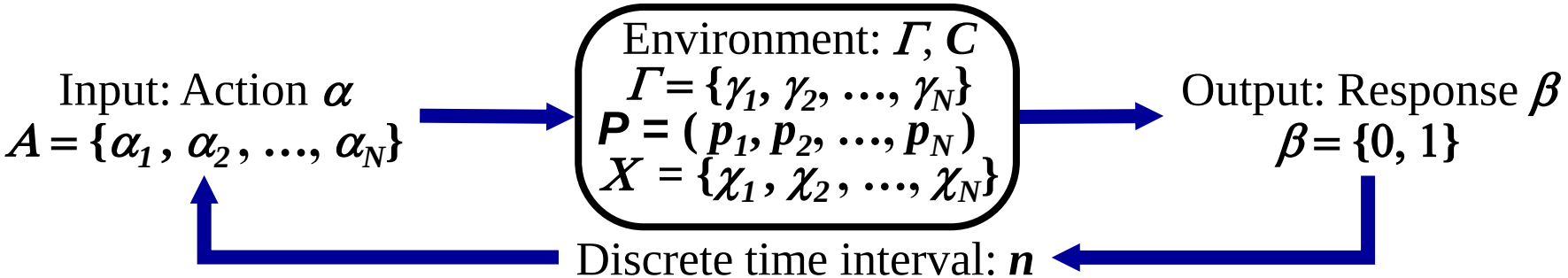
$$E[Q(n+1)] < E[Q(n)].$$

Optimality: A learning automaton is said to be ***optimal*** if:

$\lim_{n\to\infty} E[Q(n)] = 1/N$ and said to be ***e-optimal*** if: $\lim_{n\to\infty} E[Q(n)] = 1/N + e$.

Where ***e*** is arbitrarily small positive number.

Stochastic Learning Automaton
-The Norms-



The Entropy: At interval n, The entropy $H(n)$ is given by the expression:

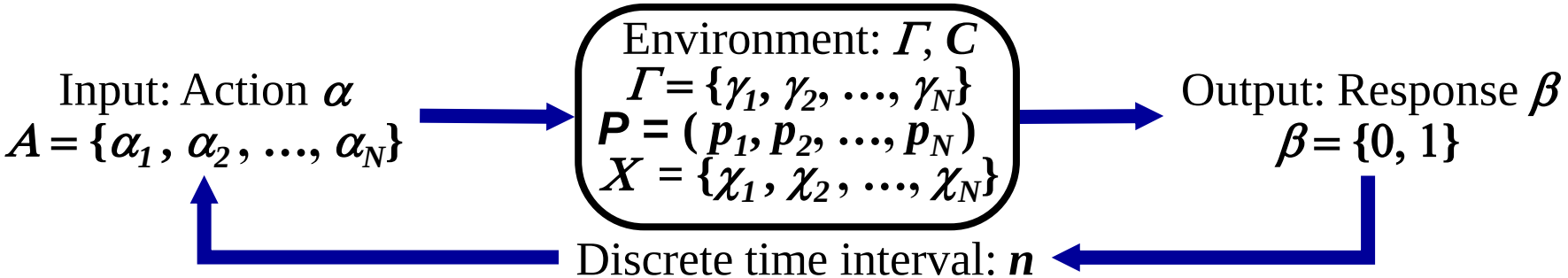
$$H(n) = \sum_{i=1}^{N} [p_i(n) \log_2(1/p_i)]$$ = Average codeword length of the alphabet Γ.

= $L_{average}(n)$ of the alphabet Γ.

If action α_i is equiprobable random variable, then; all the characters of alphabet A have equal codeword length of $\log_2(N)$.

$H(n)$ has a minimum value of **0** when one action has a probability of **1** and all other $(N-1)$ actions have probability **0**, and a maximum value when all N actions are equiprobable. An equivalent condition to the state vector $P(n)$ converging to $(1/N, 1/N, \ldots, 1/N)$ is that the sum $H(n)$ converges to $\log_2(N)$, its maximum value.

Stochastic Learning Automaton
-The Norms-



The Entropy:

$$H(n) = \sum_{i=1}^{N} [p_i(n) \log_2(1/p_i)] = L_{average}(n) \text{ of the alphabet } \Gamma.$$

Optimum $H(n)$ is equal to the average codeword of $\alpha = \log_2(N)$

i.e. The average codeword length of α is equal to that of γ

Expediency: A learning automaton is said to be ***absolutely expedient*** if:

$$E[H(n+1)] > E[H(n)].$$

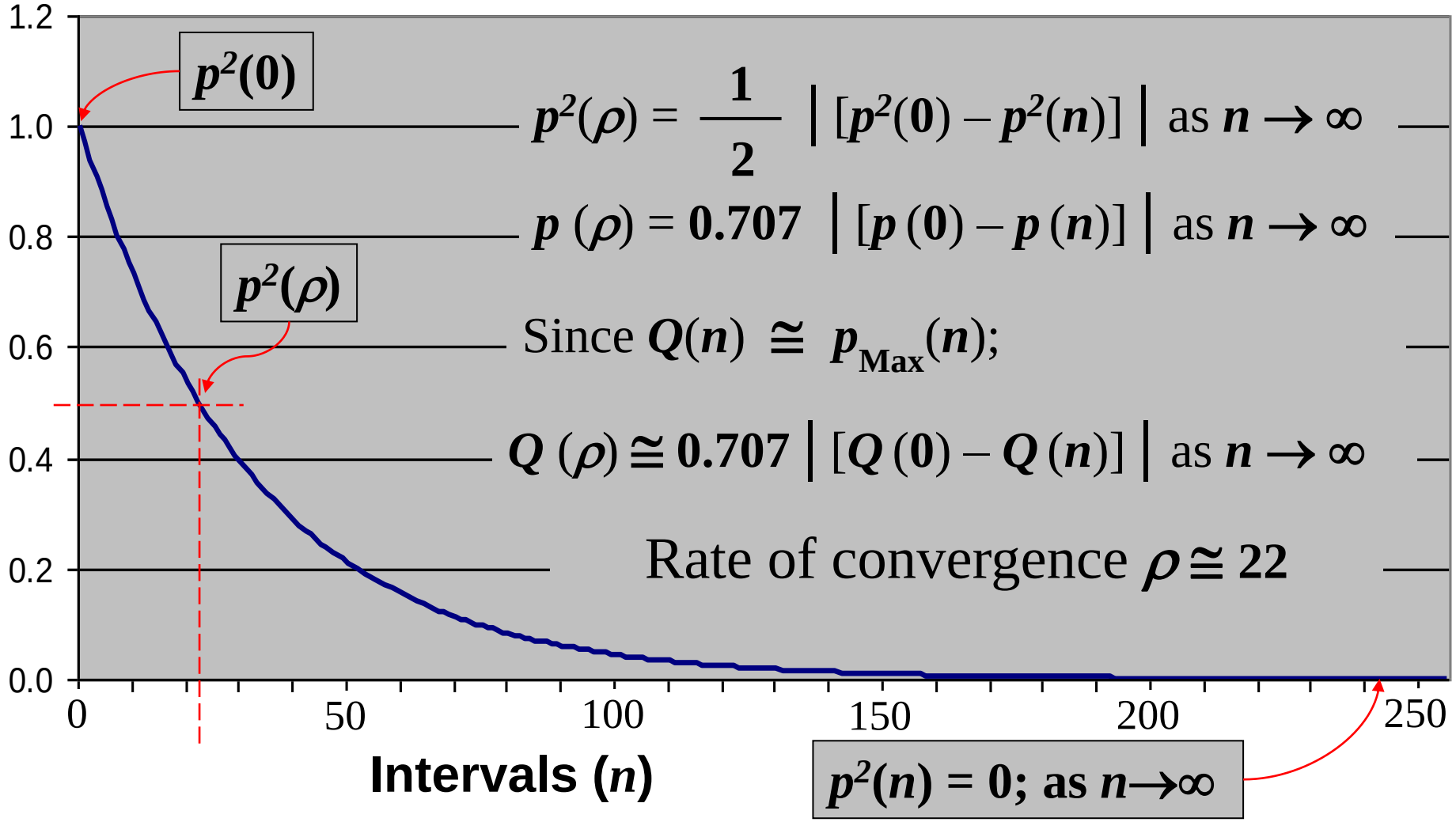
Optimality: A learning automaton is said to be ***optimal*** if:

$\lim_{n\to\infty} E[H(n)] = \log_2(N)$ and said to be ***e-optimal*** if: $\lim_{n\to\infty} E[H(n)] = \log_2(N) + e$.

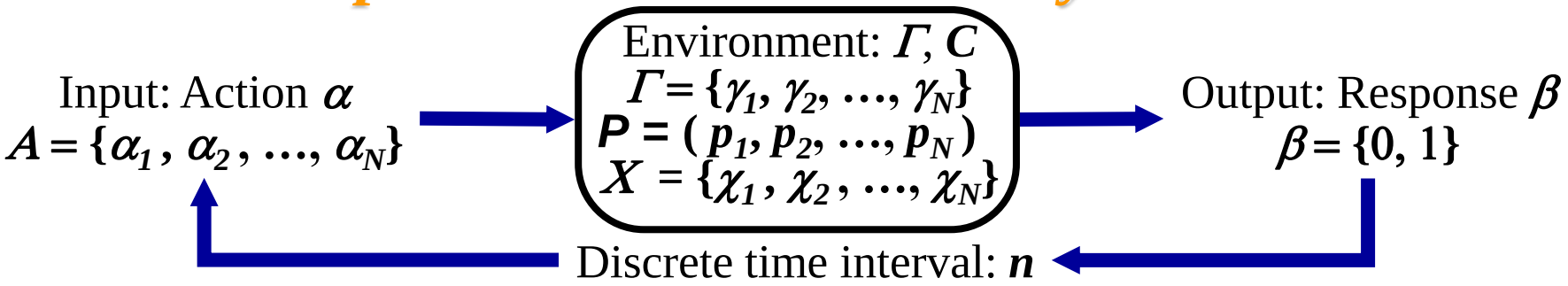
Where e is arbitrarily small positive number.

Stochastic Learning Automaton
-The Norms-

Rate of Convergence: The rate of convergence **(ρ)** is a measure of the speed of learning, it may be defined as the half power point interval:



Stochastic Learning Automaton
-Simple Linear Reward-Penalty Scheme-



Consider a simple linear scheme, where α is binary random variable, at interval n, $\alpha(n) = \alpha_i = \{0, 1\}$.

when $\beta = 0$

$$\begin{cases} p_i(n+1) = p_i(n) + a \cdot [1 - p_i(n)]; \\ p_j(n+1) = 1 - P_i(n+1) \end{cases}$$

Where $0<a<1$, $j \neq i$

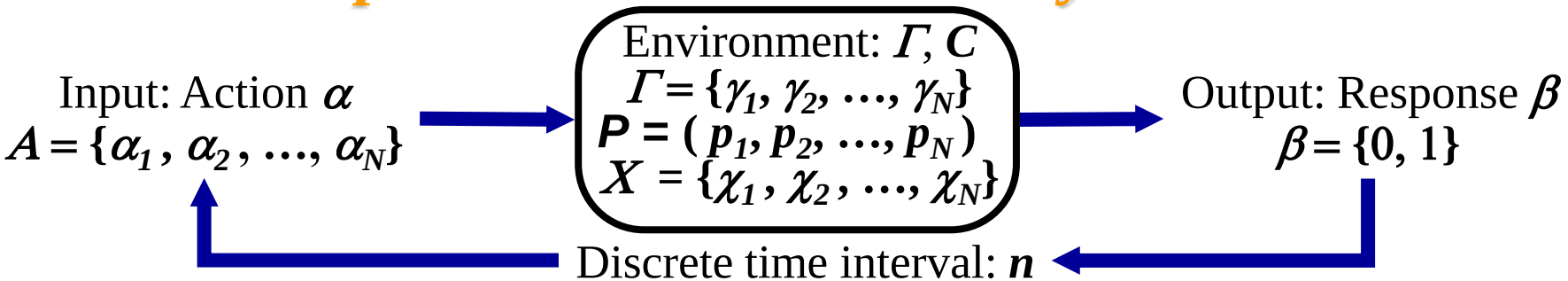
when $\beta = 1$

$$\begin{cases} p_i(n+1) = p_i(n) - b \cdot P_i(n); \\ p_j(n+1) = 1 - p_i(n+1) \end{cases}$$

Where $0<b<1$, $j \neq i$

If the learning parameters a and b are equal, the scheme is called the linear reward-penalty L_{R-P} Scheme.

Stochastic Learning Automaton
-Simple Linear Reward-Penalty Scheme-



$$Q(n) = p_i^2 + p_j^2 + \{\ldots\}$$

when $\beta = 0$; $Q(n+1) = (p_i + a \cdot p_j)^2 + (1-a)^2 \cdot p_j^2 + \{\ldots\}$

$$\begin{aligned} Q(n) - Q(n+1) &= p_i^2 + p_j^2 - (p_i + a \cdot p_j)^2 - (1-a)^2 \cdot p_j^2 \\ &= p_i^2 + p_j^2 - p_i^2 - 2a p_i p_j - a^2 p_j^2 - p_j^2 + 2a p_j^2 - a^2 p_j^2 \\ &= -2a p_i p_j - a^2 p_j^2 + 2a p_j^2 - a^2 p_j^2 \\ &= -2a p_i p_j - 2a^2 p_j^2 + 2a p_j^2 \\ &= -2a p_i p_j + 2a p_j^2 \end{aligned}$$

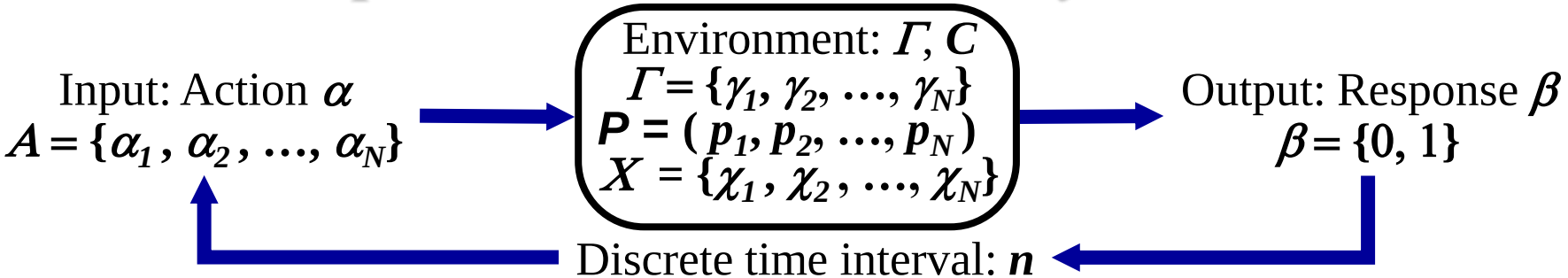
For expediency, $E[Q(n) - Q(n+1)] > 0$; then:

$$E(2a p_j^2) > E(2a p_i p_j)$$

$$E(p_j) > E(p_i)$$; This is true when $p_j > p_i$

$$\text{As } n \to \infty,\ Q(n) \le p_{max} \le 1/N = \tfrac{1}{2}$$

Stochastic Learning Automaton
-Simple Linear Reward-Penalty Scheme-



$$Q(n) = p_i^2 + p_j^2 + \{\ldots\}$$

when $\beta = 1$; $Q(n+1) = (1-b)^2 . p_i^2 + (p_j + b . p_i)^2 + \{\ldots\}$

$$\begin{aligned} Q(n) - Q(n+1) &= p_i^2 + p_j^2 - (1-b)^2 . p_i^2 - (p_j + b . p_i)^2 \\ &= p_i^2 + p_j^2 - p_i^2 + 2\,b\,p_i^2 - b^2 p_i^2 - p_j^2 - 2\,b\,p_i p_j - b^2 p_i^2 \\ &= + 2\,b\,p_i^2 - b^2 p_i^2 - 2\,b\,p_i p_j - b^2 p_i^2 \\ &= + 2\,b\,p_i^2 - 2b^2 p_i^2 - 2\,b\,p_i p_j \\ &= + 2\,b\,p_i^2 - 2\,b\,p_i p_j \end{aligned}$$

For expediency, $E[Q(n) - Q(n+1)] > 0$; then:

$$E(2\,b\,p_i^2) > E(2\,b\,p_i p_j)$$

$$E(p_i) > E(p_j)\text{; This is true when } p_i > p_j$$

$$\text{As } n \to \infty,\ Q(n) \le p_{max} \le 1/N = \tfrac{1}{2}$$

Stochastic Learning Automaton
-Practical Simulation-

Practical simulation of the simple binary stochastic learning automaton proposed in previous slides.

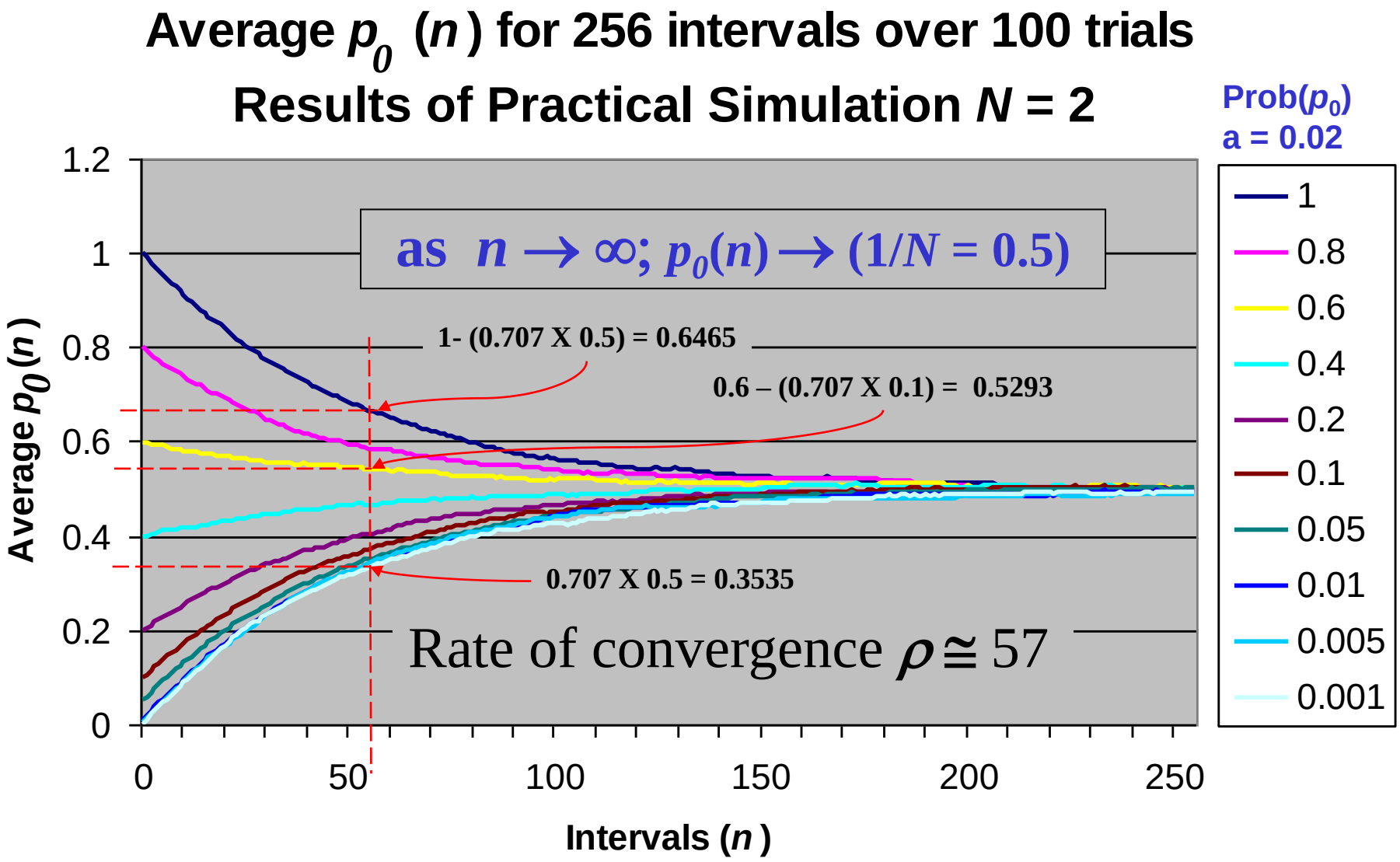
Initial Conditions: The initial environment probability $p_1(0)$ is set to one of ten values in the range of $[0.001 \leq p_1(0) \leq 1]$ and $p_2(0)$ is made to be equal to $[1 - p_1(0)]$.

Results: The behaviour of the algorithm is determined by plotting the average values of $p_1(n)$, $M(n)$, $Q(n)$ and $H(n)$ over hundred **(100)** trials, for every one of the predetermined ten different set of initial probability distributions.

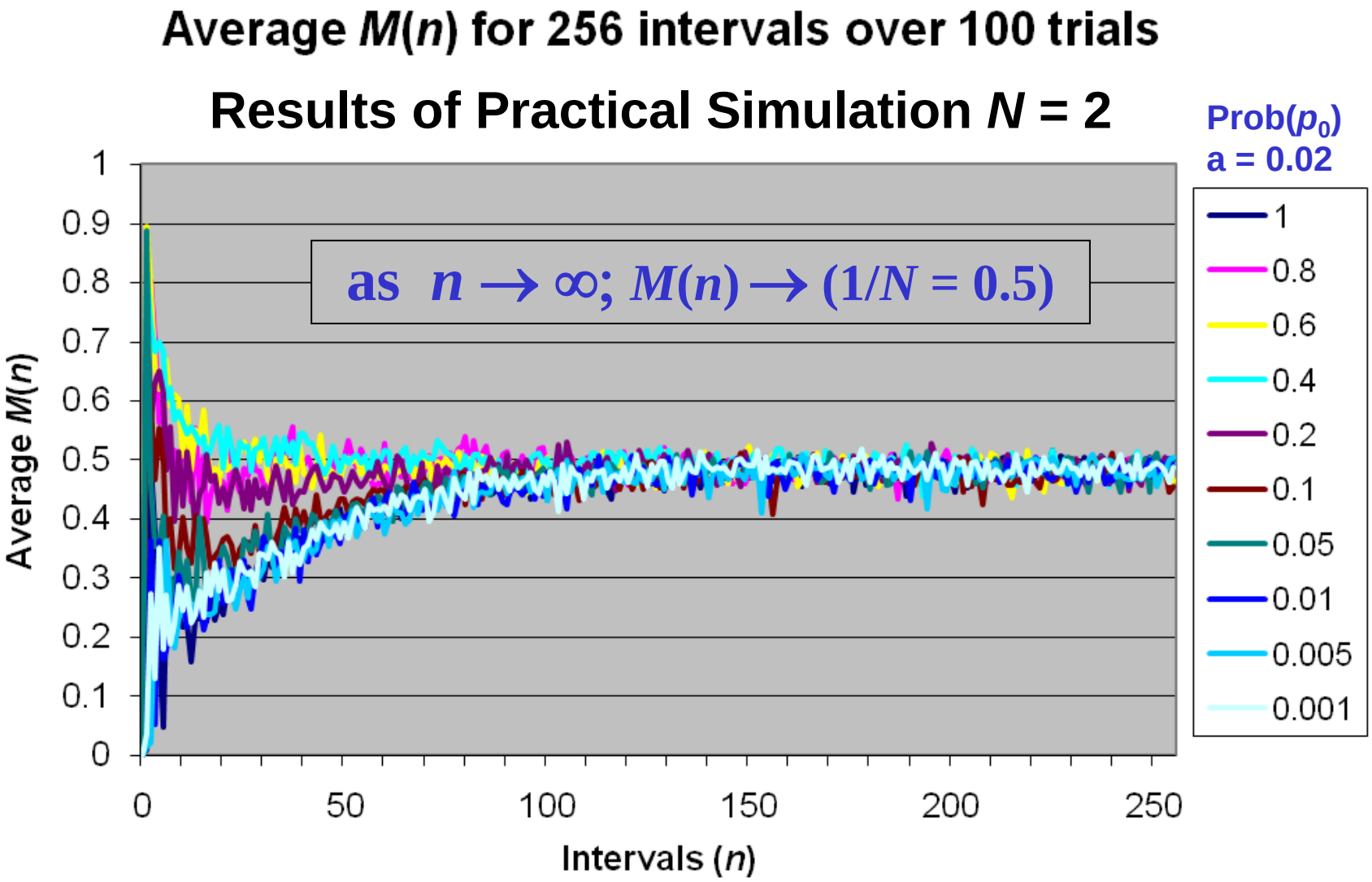
Where $M(n)$, $Q(n)$ and $H(n)$ are given by the expressions:

$$M(n) = \sum_{i=1}^{N} c_i p_i(n); \quad Q(n) = \sum_{i=1}^{N} p_i^2(n); \quad H(n) = \sum_{i=1}^{N} p_i(n) \log_2[1/p_i(n)]$$

Stochastic Learning Automaton
-Simple Linear Reward-Penalty Scheme-

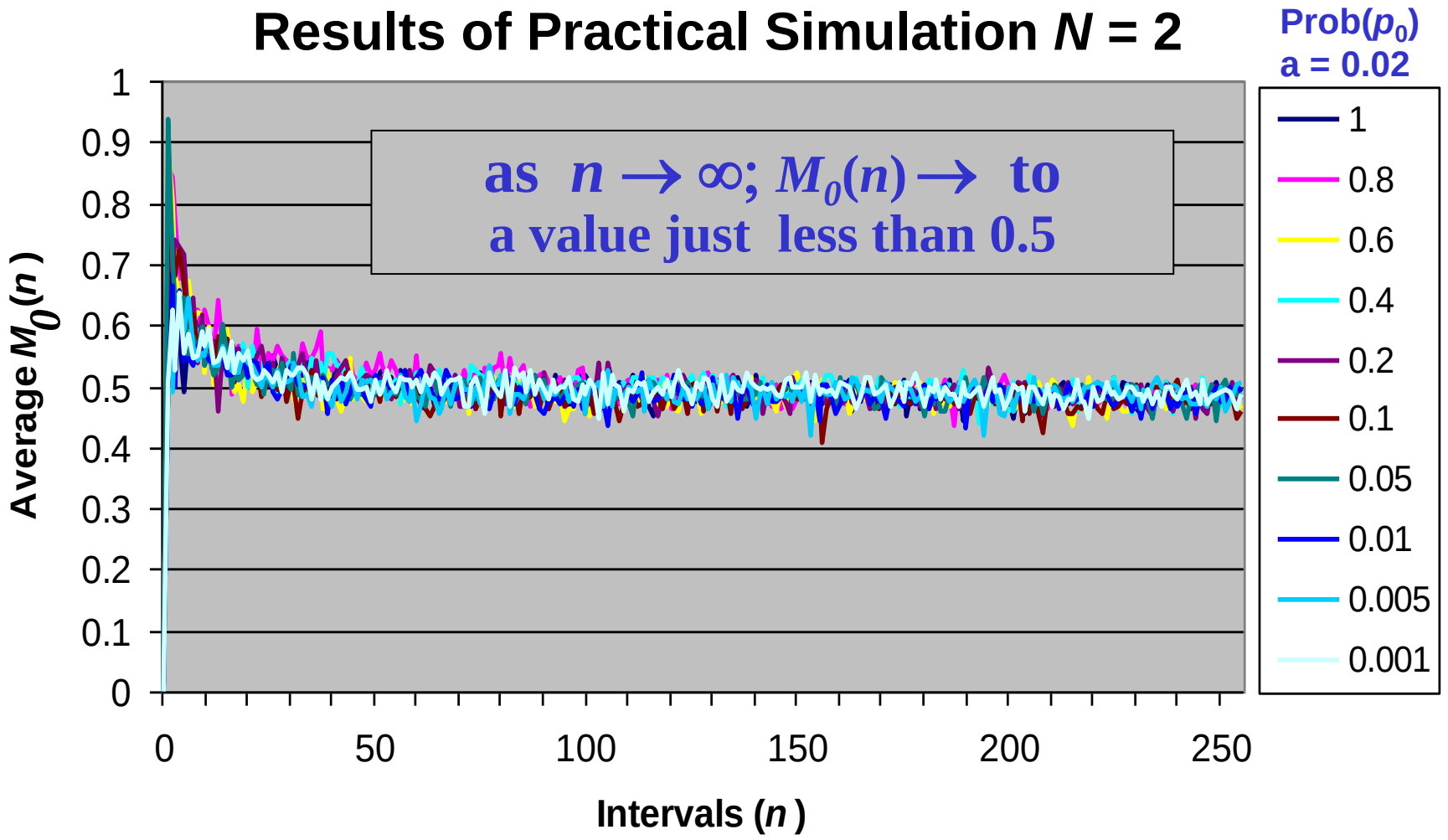


Stochastic Learning Automaton
-Simple Linear Reward-Penalty Scheme-

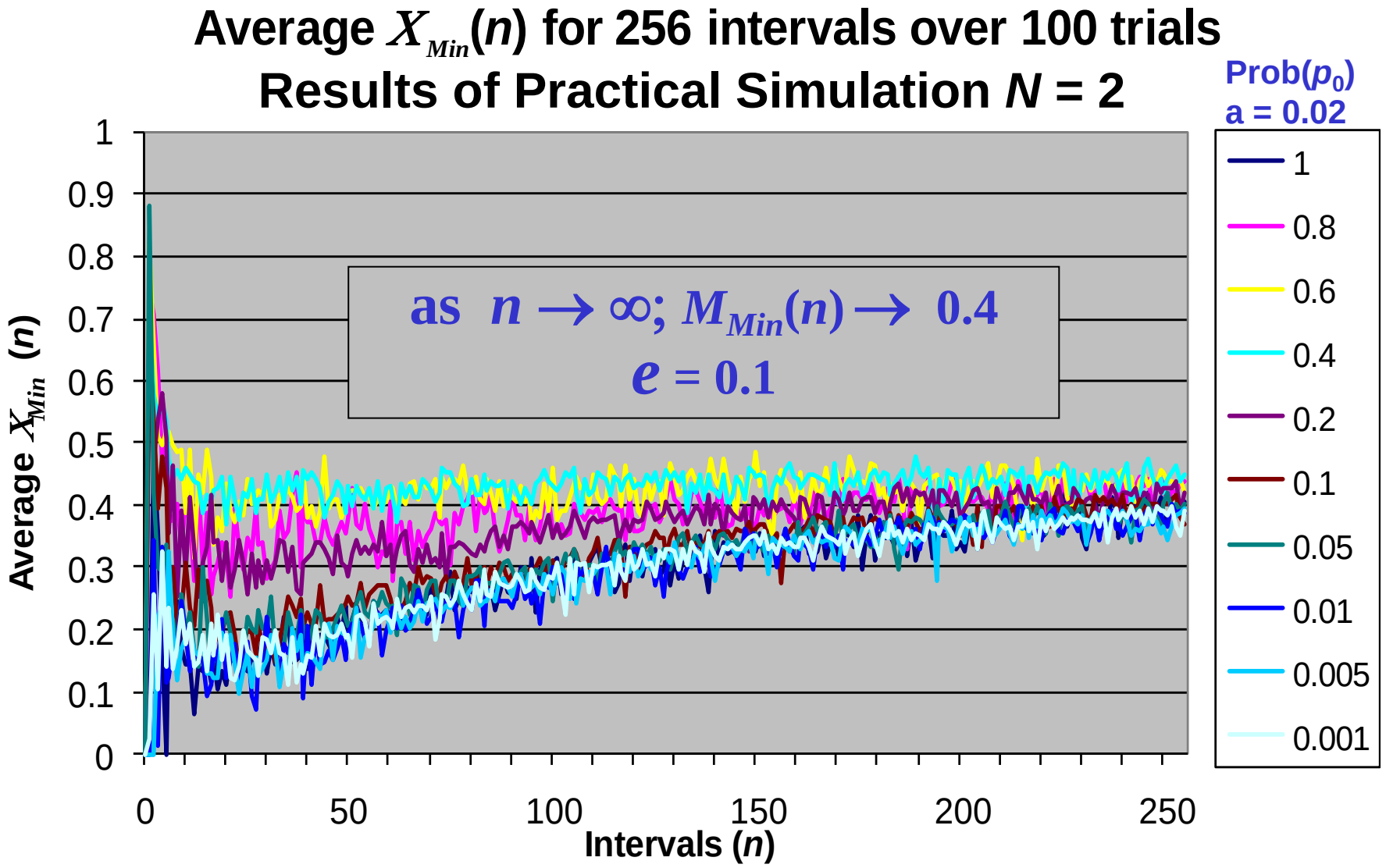


Stochastic Learning Automaton
-Simple Linear Reward-Penalty Scheme-

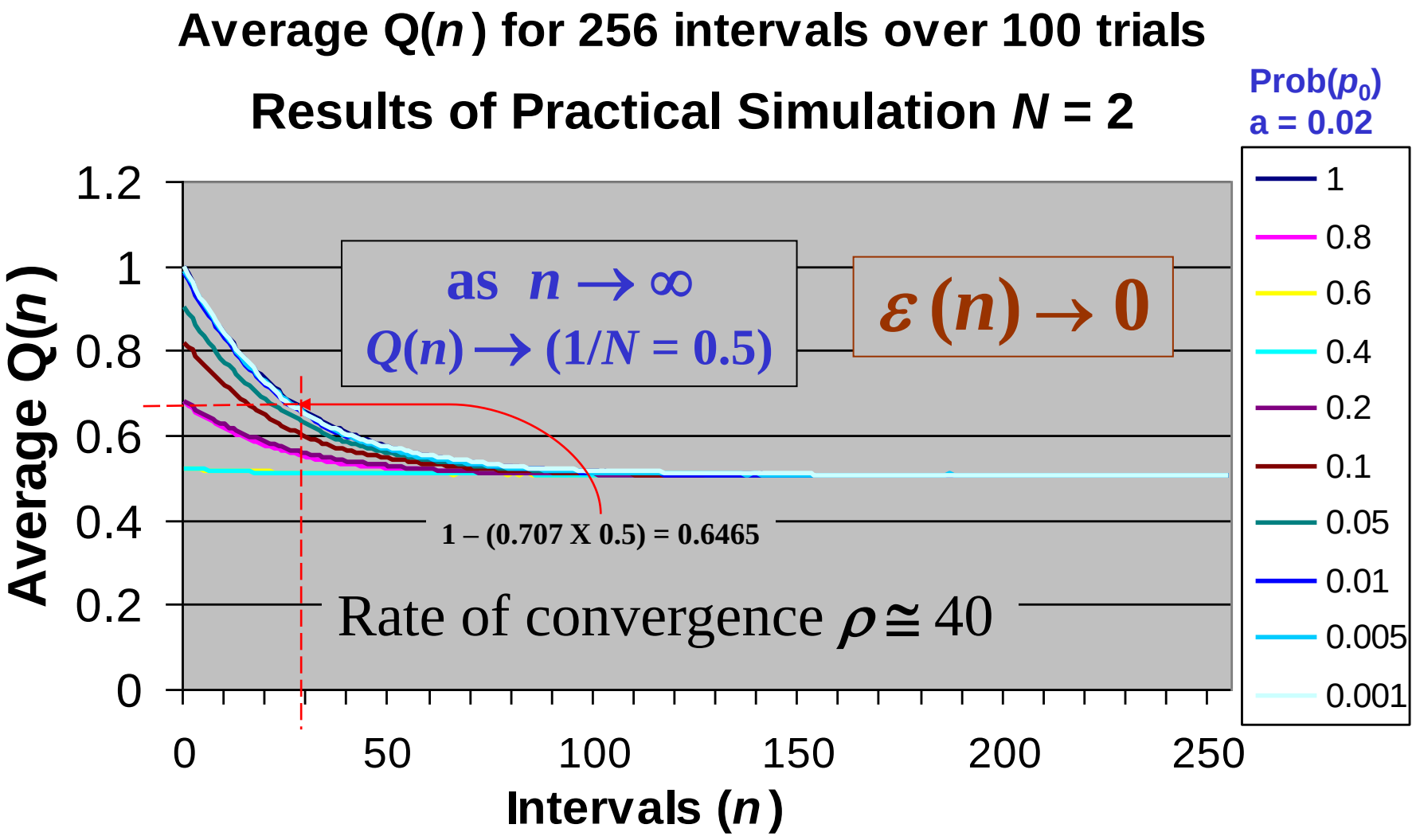
Average $M_0(n)$ for 256 intervals over 100 trials



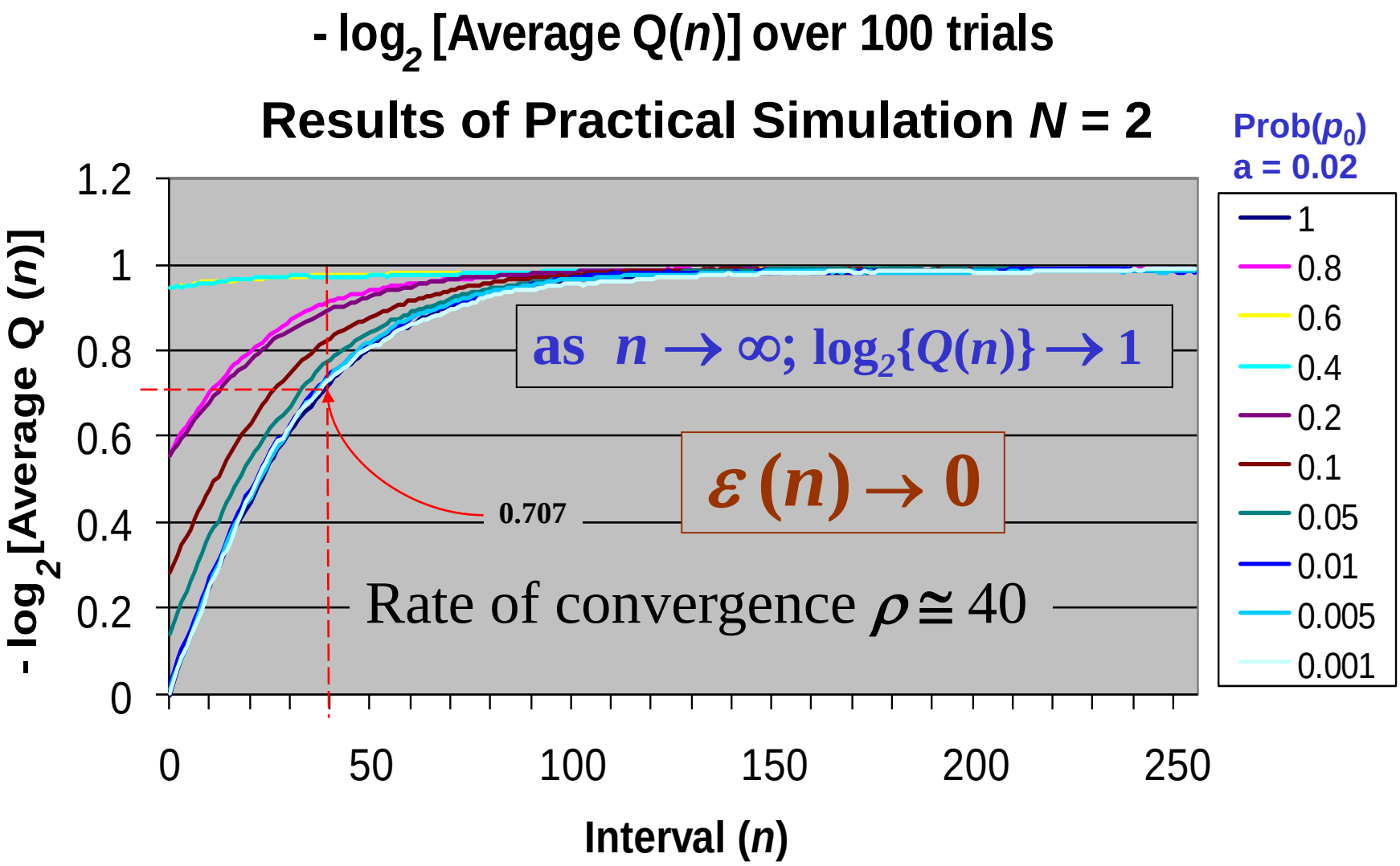
Stochastic Learning Automaton
-Simple Linear Reward-Penalty Scheme-



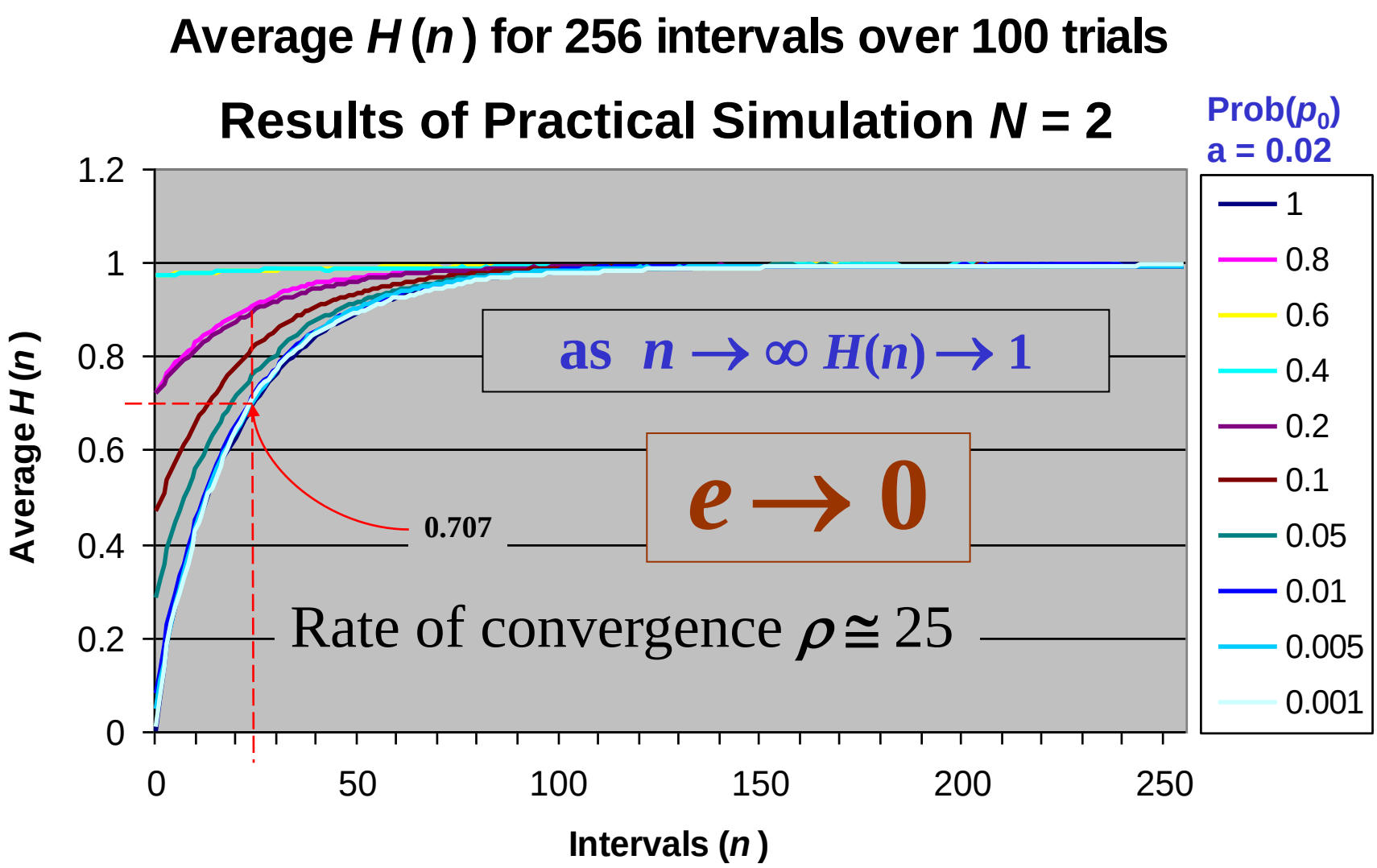
Stochastic Learning Automaton
-Simple Linear Reward-Penalty Scheme-



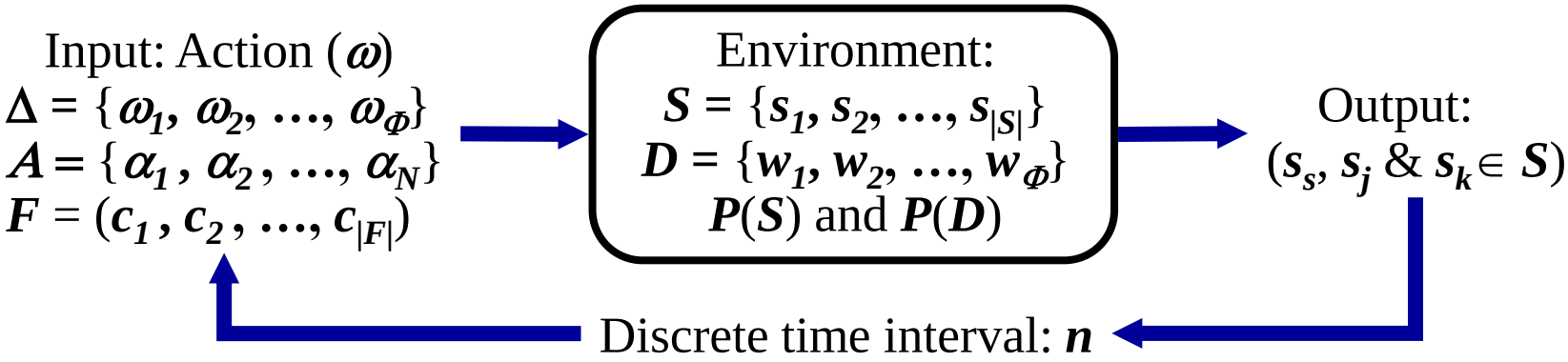
Stochastic Learning Automaton
-Simple Linear Reward-Penalty Scheme-



Stochastic Learning Automaton
-Simple Linear Reward-Penalty Scheme-



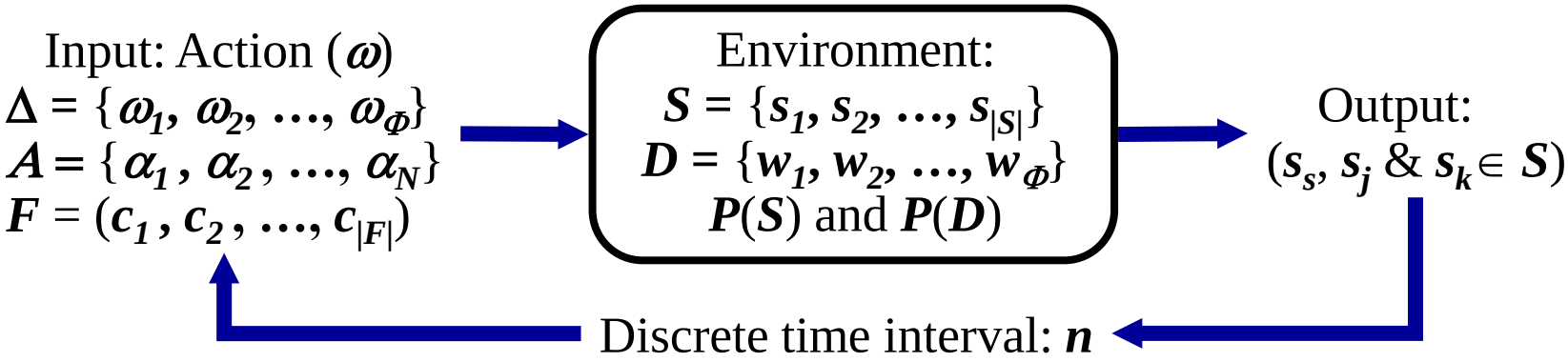
S&M algorithm: The Concept
-Action-



Action (ω): ω is random variable of a stationary environment Θ over a language with alphabet set (A); $A = \{\alpha_1, \alpha_2, \ldots, \alpha_N\}$, N is the number of characters in the alphabet. The i^{th} action (ω_i) is a string, (word), in the source dictionary (Δ); $\Delta = \{\omega_1, \omega_2, \ldots, \omega_\Phi\}$, Φ is the file size of the dictionary ($|\Delta|$); i.e. the number of words in Δ. The dictionary must contains at least all the characters in alphabet A; (Min ($|\Delta|$) = N). The prior probabilities distribution and the statistical parameters of the words in Δ are not known explicitly.

The input source file (F) contains a sequence of symbols, (characters) of the alphabet A. At interval n, an input string of source symbols is matched with the longest string in the dictionary $(\omega(n))$. $\omega(n)$ is known as the n^{th} action.

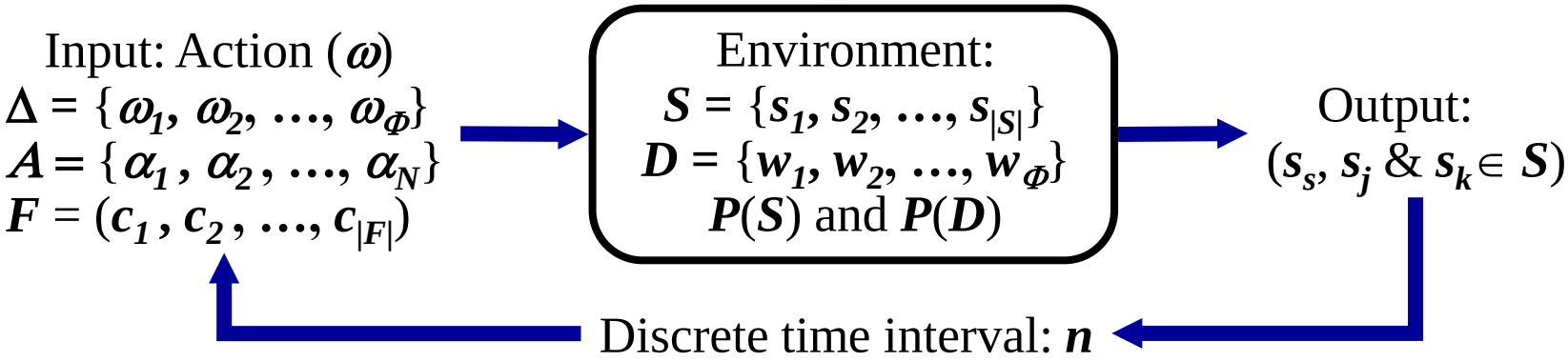
S&M algorithm: The Concept
-Environment-



Environment (S, D): Set S is a set of $|S|$ mutually exclusive sub sets, $s_1, s_2, .., s_{|S|}$. Each sub set contains a number of unique words (w_i) of a dictionary (D), (where, s_i is a subset of D, and $s_i = \{w_{i1}, w_{i2}, \ldots, |s_i|\}$. $P(S)$ is the set state probability vector, $P(S) = (\, p(s_1), p(s_2), \ldots, p(s_{|S|}) \,) \mid p(s_i) = \sum_{j=1}^{|S_i|} p(w_{ij})$. $P(D)$ is the dictionary state probability vector $P(D) = (\, p(w_1), p(w_2), \ldots, p(w_\Phi) \,)$.

The dictionary $D = \Delta$, however, the state probability vector of D is updated by a pre-defined updating scheme. Since the action $\omega_i = w_i \mid \omega(n) = \omega_i$ and $w(n) = w_i$, $\omega_i \in \Delta$ and $w_i \in D$, the environment has no reward-penalty response.

S&M algorithm: The Concept
-Output-



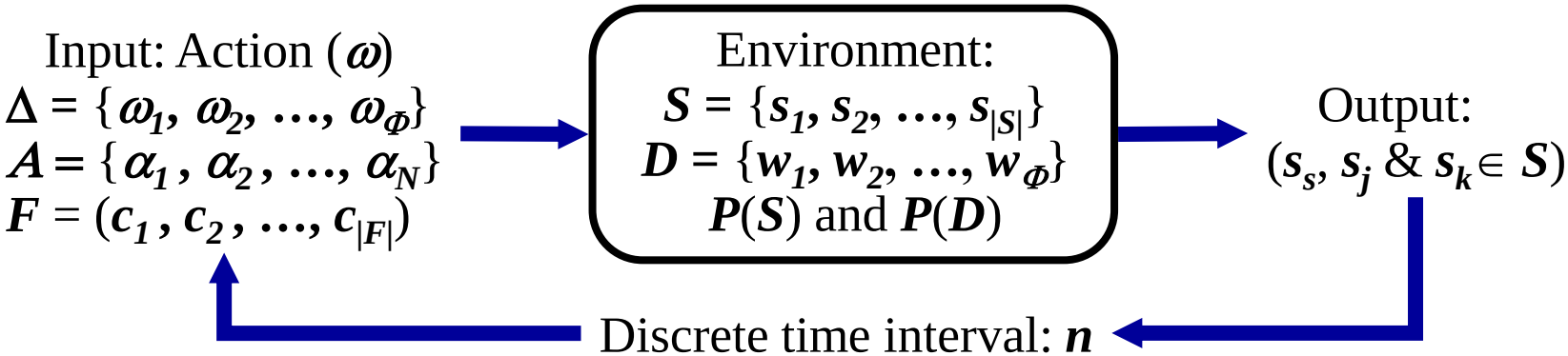
Output (s_s, s_j, s_k): For every action ($\omega(n)$), the environment respond with three duple output ((s_s, s_j and $s_k \in S$). The first is a set ($s_s \in S$), called (***the split set***): The split set is that set contains the word, which, matches the action $\omega(n)$.

If $w_s = \omega(n) \mid \omega(n) = \omega_s$, and $w_s \in s(n) \mid s(n) = s_s$.

The two, other sets (s_j & $s_k \in S$), called (***the merger sets***), are selected randomly, from the $|S|$ subsets of S, such that:

$j < k$, and $j \neq k \neq s$.

S&M algorithm: The Concept
-Transition-

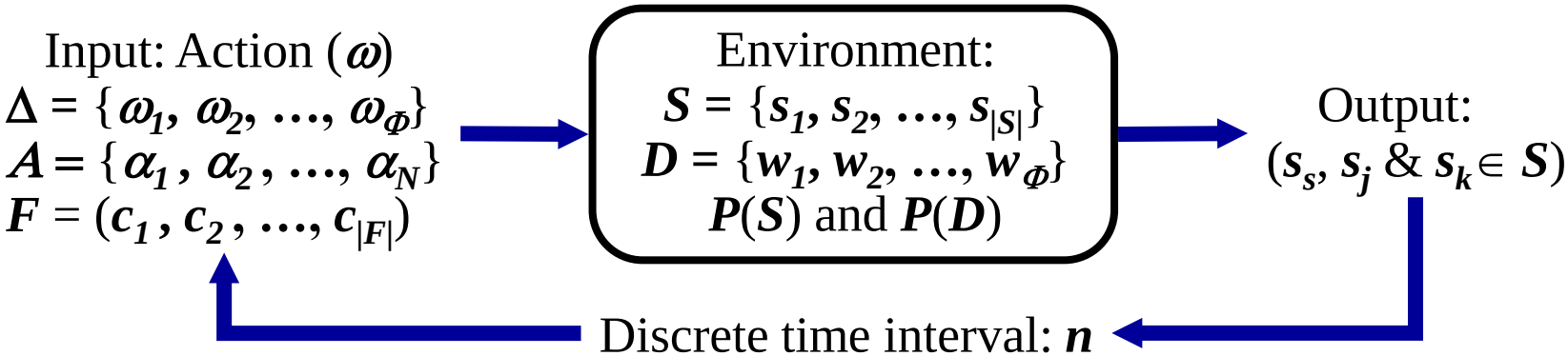


Transition **($P(S)$, $P(D)$)**: The state, set probability vectors **($P(S)$)** and the state dictionary probability vector **($P(D)$)** are updated by a pre-defined updating scheme. The scheme should ensure expediency and asymptotic convergence of:

1) the set state probability vector $P(S)$ to $(1/|S|, 1/|S|, \ldots, 1/|S|)$. i. e. the set probability $\lim_{n\to\infty} P(s_i) = 1 / |S|$, for $i = 1, 2, \ldots, |S|$.

2) the dictionary state probability vector $P(D)$ to the real probability vector of the source dictionary Δ.

S&M algorithm: The Concept
The Action-Norms

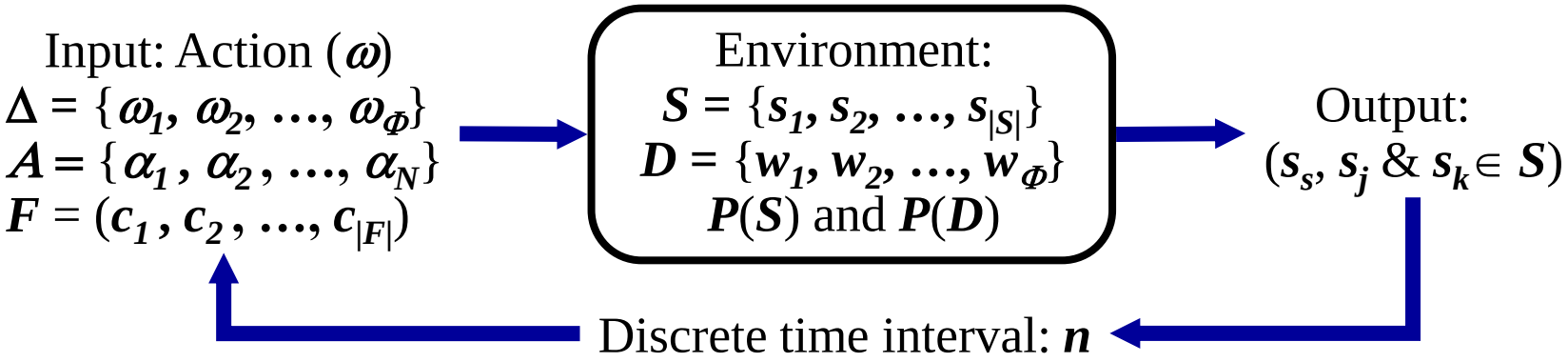


Action Norms (H_{α}, Q_{α}): The norms used as a datum reference for the automaton learning capability is called the ***Action-Norms***, or (***α-Norms***) of the probability vectors $P(\omega)$. H_α is the action entropy and the ***MSE variable*** (Q_α), is the sum of, the square of, the word probabilities, for all words in Δ.

$$H_\alpha(n) = \sum_{i=1}^{|S|} \left[\sum_{j=1}^{|s_i|} (-\, p\,(\omega_{ij}) \mid \omega_{ij} \in s_i) \right] \cdot \log_2 \left[\sum_{j=1}^{|s_i|} (p\,(\omega_{ii}) \mid \omega_{ij} \in s_i) \right]$$

$$Q_\alpha(n) = \sum_{i=1}^{|S|} \sum_{j=1}^{|s_i|} \left[(p\,(\omega_{ij}) \mid \omega_{ij} \in s_i) \right]^2$$

S&M algorithm: The Concept
The Set-Norms

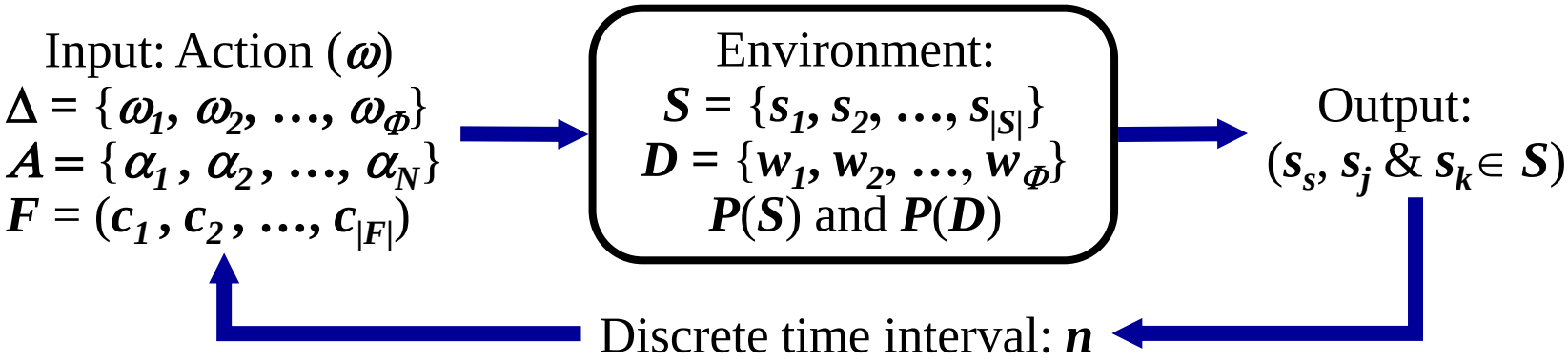


Set Norms (H_S, Q_S): The norms used to test the expediency and optimality of the set state probability vectors $P(S)$ are called the ***Set-Norms***, or (***S-Norms***). H_S is the set real entropy, Q_S is the set real ***MSE Variable***, while the state set entroy equal to $\log_2(|S|)$ and the state set ***MSE variabl*** equal to $(1 / |S|)$.

$$H_S(n) = \sum_{i=1}^{|S|} \left[\sum_{j=1}^{|s_i|} (-\, p(\omega_{ij}) \mid \omega_{ij} \in s_i) / FBL(s_i)\right] \cdot \log_2 \left[\sum_{j=1}^{|s_i|} (\, p(\omega_{ii}) \mid \omega_{ij} \in s_i) / FBL(s_i)\right]$$

$$Q_S(n) = \sum_{i=1}^{|S|} \left[\sum_{j=1}^{|s_i|} (\, p\,(\omega_{ij}) \mid \omega_{ij} \in s_i) / FBL(s_i)\right]^2$$

S&M algorithm: The Concept
The Word-Norms



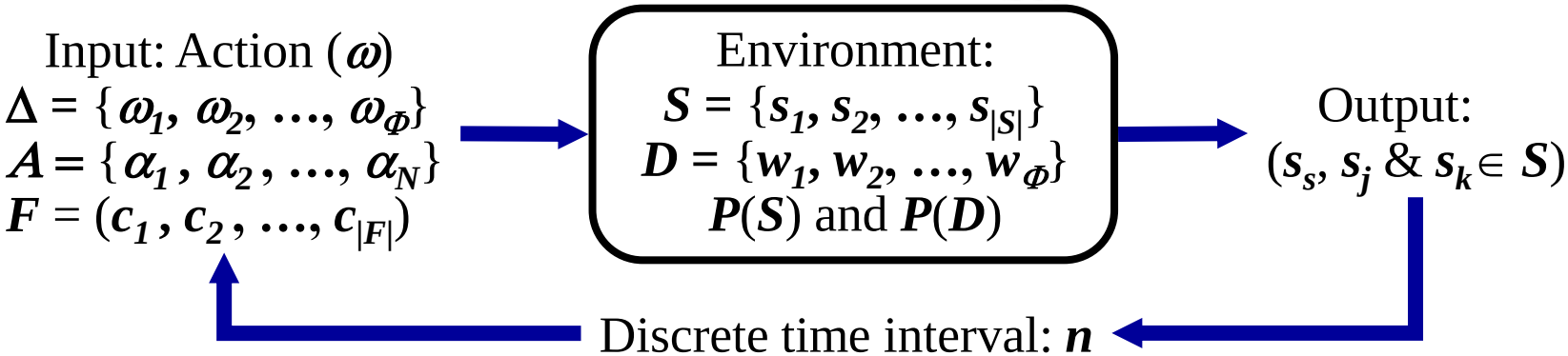
Word Norms (H_W, Q_W): The norms used to test the expediency and optimality of the set state probability vectors $P(w)$ are called ***Word-Norms***, or (***W-Norms***). H_W is the state word entropy and the state word ***MSE variable*** (Q_W) is the sum of, the square of, word probabilities, for all words in D.

$$H_w(n) = \sum_{i=1}^{|S|} \left[\sum_{j=1}^{|s_i|}(-p_{inset}(w_{ij})|w_{ij}\in s_i)\, FBL(s_i)/|S|\right] \cdot \log_2 \left[\sum_{j=1}^{|s_i|}(p_{inset}(w_{ii})|w_{ij}\in s_i)\, FBL(s_i)/|S|\right]$$

$$Q_W(n) = \sum_{i=1}^{|S|}\sum_{j=1}^{|s_i|}\left[\left(p_{inset}(w_{ij}) \mid w_{ij} \in s_i\right) \cdot FBL(s_i) / |S|\right]^2$$

S&M algorithm: The Norms

The Set-Norm: $H_{Set}(n)$



The Entropy: Sets have equiprobable state probability, then:

$$H_{set}(n) = \sum_{i=1}^{|S|} [p_{s_i}(n) \log_2(1/p_{s_i})] = L_{average} = \text{Average set codeword length.}$$

Optimum $H_{Set}(n)$ is equal to the average codeword of $s_i = \log_2(|S|)$

Expediency: The algorithm is said to be ***absolutely expedient*** if:

$$E[H_{Set}(n+1)] > E[H_{Set}(n)].$$

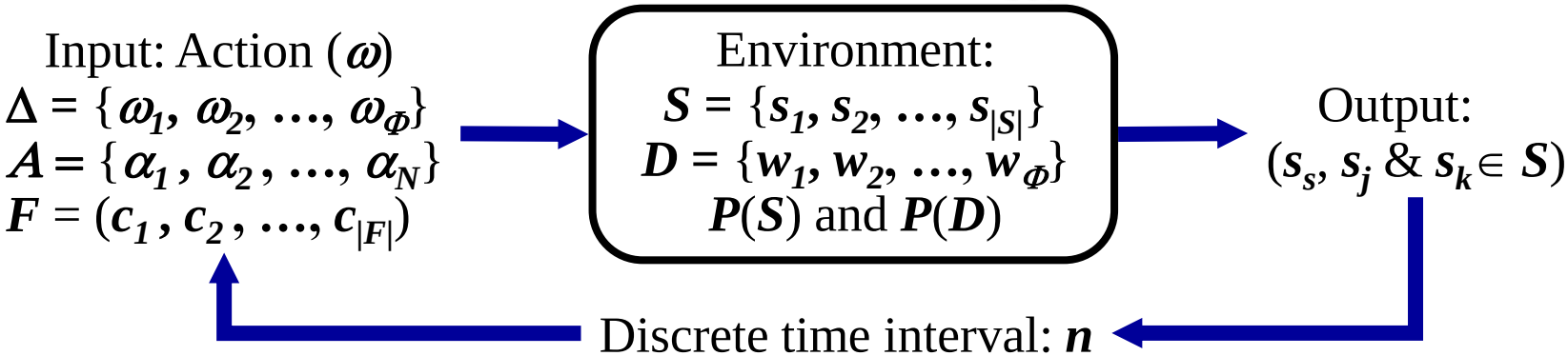
Optimality: The algorithm is said to be ***optimal*** if:

$\lim_{n\to\infty} E[H_{Set}(n)] = log_2(|S|)$ and said to be ***e-optimal*** if: $\lim_{n\to\infty} E[H_{Set}(n)] = \log_2(|S|) + e$.

Where e is arbitrarily small positive number.

S&M algorithm: The Norms

The Set-Norm: $Q_{set}(n)$



Sets ***MSE vector***: Sets have equiprobable state probability, then:

$$\varepsilon(n) = \frac{1}{|S|}\sum_{i=1}^{|S|}\left(p_{s_i}(n) - \frac{1}{|S|}\right)^2 = \frac{1}{|S|}\sum_{i=1}^{|S|}p_{s_i}(n)^2 - \frac{2}{|S|^2}\sum_{i=1}^{|S|}p_{s_i}(n) + \frac{1}{|S|^2}$$

When $\varepsilon(n) = 0$; $\sum_{i=1}^{N}(p_{s_i}(n) = \frac{1}{|S|}$

Expediency: The algorithm is said to be ***absolutely expedient*** if:

$$E\,[\,Q_{Set}(n+1)\,] < E\,[\,Q_{Set}(n)\,].$$

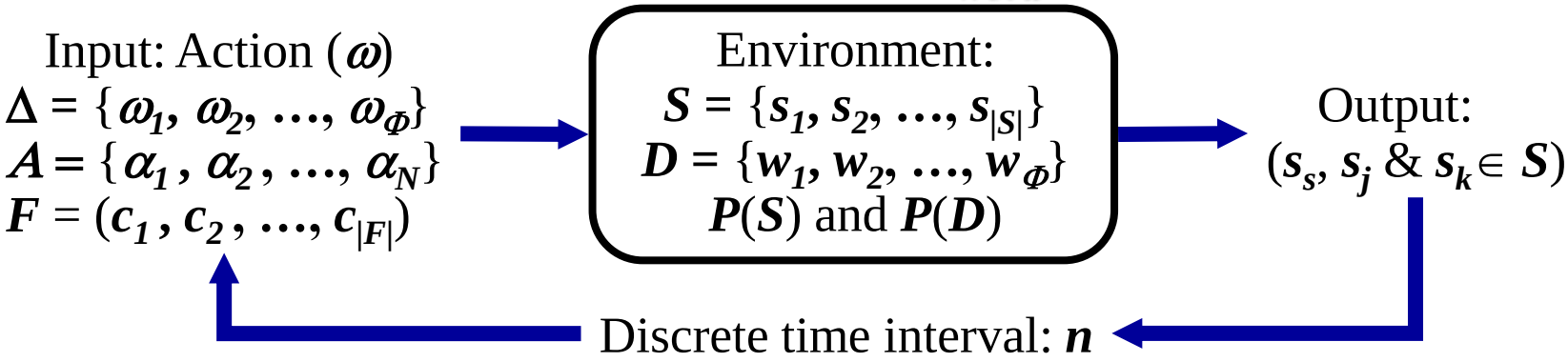
Optimality: The algorithm is said to be ***optimal*** if:

$\lim_{n\to\infty} E\,[Q_{Set}(n)] = 1/|S|$ and said to be ***e-optimal*** if: $\lim_{n\to\infty} E\,[Q_{Set}(n)] = 1/|S| + e$.

Where ***e*** is arbitrarily small positive number.

S&M algorithm: The Norms

Word-Norm: $H_{word}(n)$



The Entropy: Assume all words in the source dictionary Δ are equiprobable, then; all source words have equal codeword length of $\log_2(|\Phi|)$.

At interval n, The entropy $H_{word}(n)$ is given by the expression:

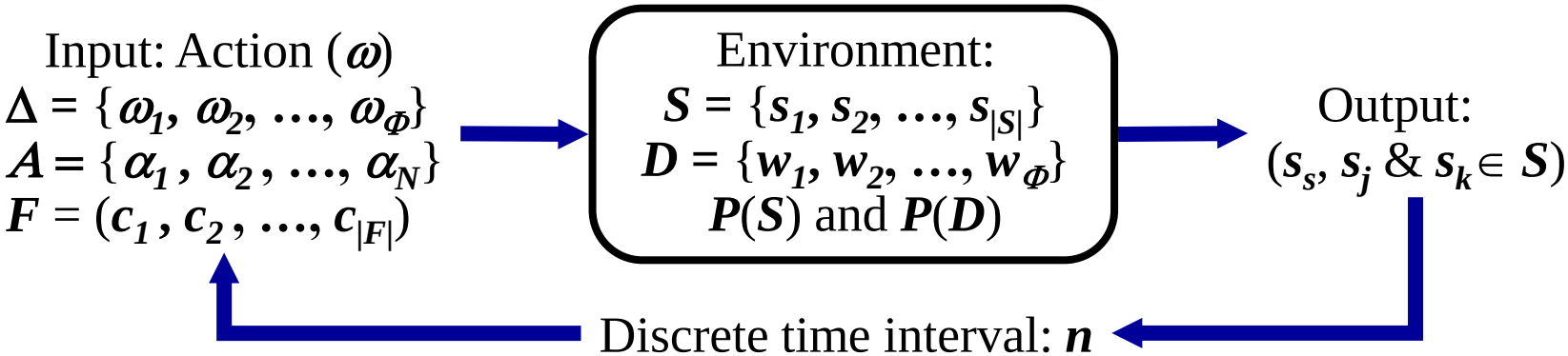
$$H_{word}(n) = \sum_{i=1}^{|\Phi|} [p[w_i(n)] \log_2\{1/p[w_i(n)]\}$$ = Average codeword length of all words in the dictionary D.

$$= L_{average}\{w_i(n)\}.$$

$H_{word}(n)$ has a minimum value of **0** when one word has a probability of **1** and all other $(|\Phi|-1)$ words have probability **0**, and a maximum value of $\log_2(|\Phi|)$,when all words of D, are equiprobable.

S&M algorithm: The Norms

The Word-Norm: $Q_{word}(n)$



Word ***MSE vector***: Assume the action is equiprobable random variable, then:

$$\varepsilon(n) = \frac{1}{|\Phi|}\sum_{i=1}^{|\Phi|}\left\{p[w_i(n)] - \frac{1}{|\Phi|}\right\}^2 = \frac{1}{|\Phi|}\sum_{i=1}^{|\Phi|}\{p[w_i(n)]\}^2 - \frac{1}{|\Phi|^2}$$

When $\varepsilon(n) = 0$; $\sum_{i=1}^{|\Phi|}\{p[w_i(n)]\}^2 = \frac{1}{|\Phi|}$

Expediency: The algorithm is said to be ***absolutely expedient*** if:

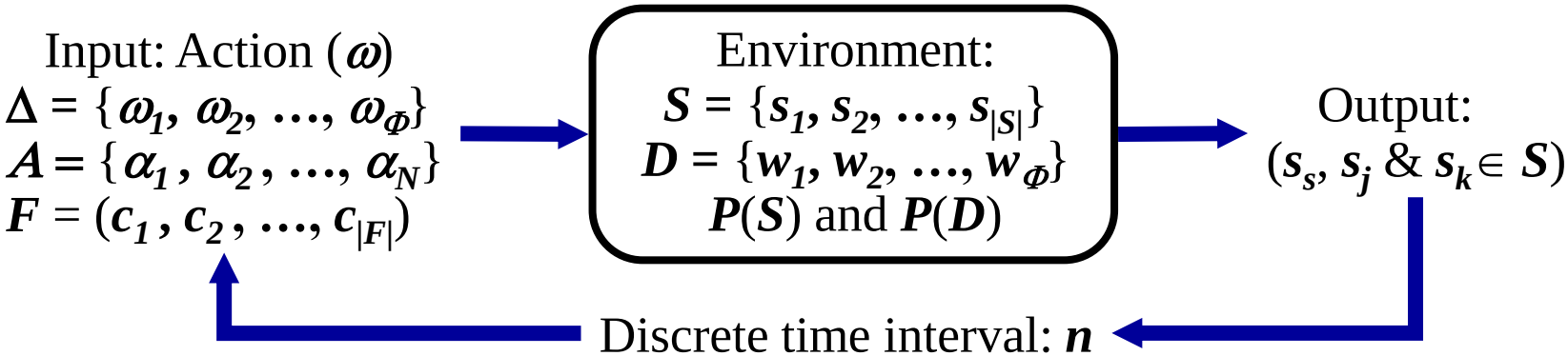
$$E[Q_{word}(n+1)] < E[Q_{word}(n)].$$

Optimality: The algorithm is said to be ***optimal*** if:

$\lim_{n\to\infty} E[Q_{word}(n)] = 1/|S|$ and said to be ***e-optimal*** if: $\lim_{n\to\infty} E[Q_{word}(n)] = 1/|\Phi| + e$.

Where e is arbitrarily small positive number.

S&M algorithm: The Strategy



The Strategy:

At every interval n, $(n = 1, 2, ..., |F|)$, the element of set $s_k(n)$ is merged with element of set $s_j(n)$:

$$s_j(n+1) = s_j(n) \cup s_k(n) \text{ and}$$
$$p[s_j(n+1)] = p[s_j(n)] + p[s_k(n)].$$

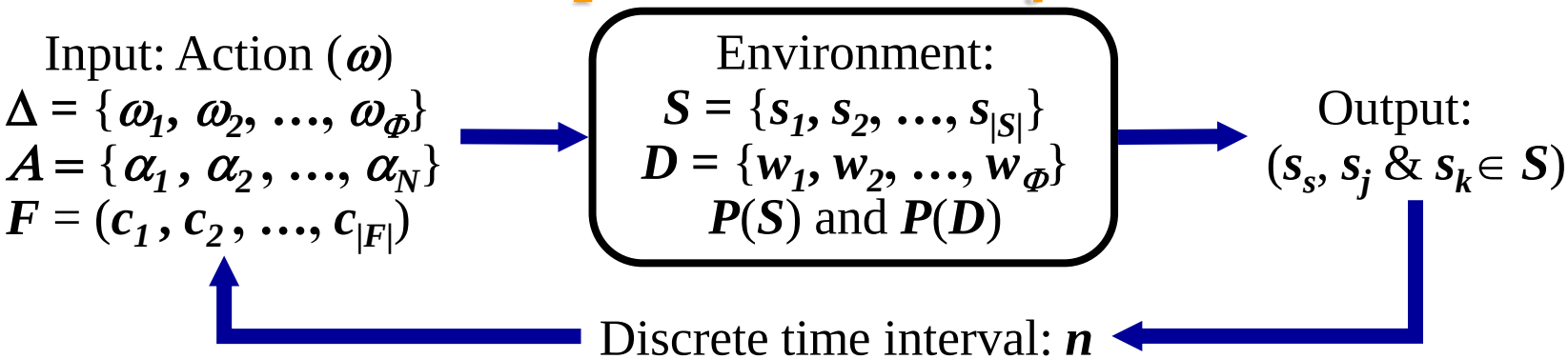
The element of the split set $s_s(n)$ is divided into two sets (s_{s1}) and (s_{s2}) such that:

$$p(s_{s1}) = a \cdot p(s_s) \text{ and } p(s_{s2}) = (1-a) \cdot p(s_s).$$

$$0 \le a \le 1$$

$$s_s(n+1) = s_{s1} \quad \text{and} \quad s_k(n+1) = s_{s2}.$$

S&M algorithm: The Strategy
-Optimum value of a-



$Q_S(n) = \{\ldots\} + p_{s_s}^2 + p_{s_j}^2 + p_{s_k}^2$; Split p_{s_s} into $a\,p_{s_s}$ and $(1-a)\,p_{s_s}$.

$Q_S(n+1) = \{\ldots\} + a^2 p_{s_s}^2 + (1-a)^2 P_{s_s}^2 + (p_{s_j} + p_{s_k})^2$

$= \{\ldots\} + a^2 p_{s_s}^2 + p_{s_s}^2 - 2a\,p_{s_s}^2 + a^2 p_{s_s}^2 + p_{s_j}^2 + 2\,p_{s_j} p_{s_k} + p_{s_k}^2$

$= \{\ldots\} + p_{s_s}^2 + p_{s_j}^2 + p_{s_k}^2 + 2a^2 p_{s_s}^2 - 2a\,p_{s_s}^2 + 2\,p_{s_j} p_{s_k}$

$Q_S(n) - Q_S(n+1) = -2a^2 p_{s_s}^2 + 2a\,p_{s_s}^2 - 2\,p_{s_j} p_{s_k}$

$d\{Q_S(n) - Q_S(n+1)\} / d(a) = -4a\,p_{s_s}^2 + 2\,p_{s_s}^2$

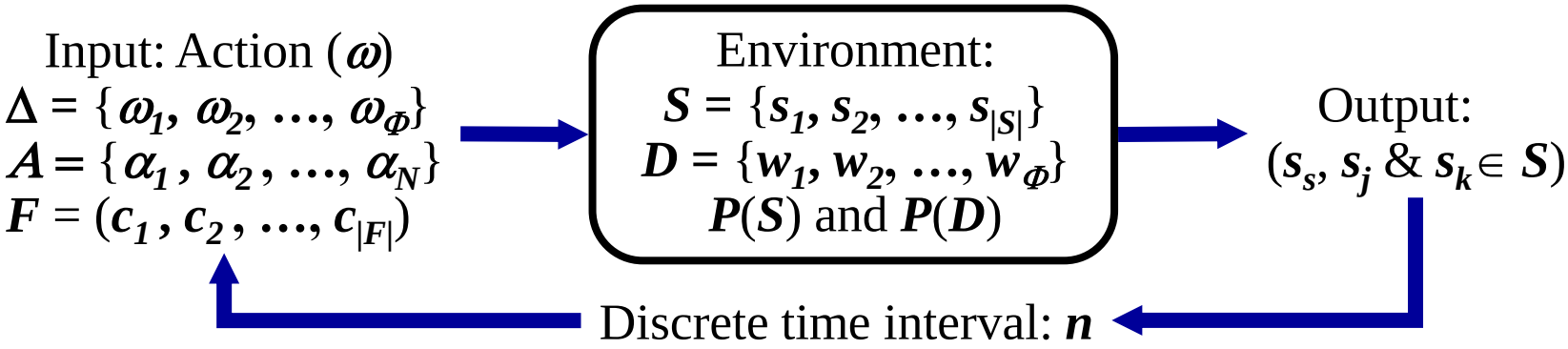
$= 0$; when $a = ½$.

$d^2\{Q_S(n) - Q_S(n+1)\} / d(a)^2 = -4\,p_{s_s}^2$; Maximum value, when $a = ½$.

Optimum value of splitting, when $a = ½$.

S&M algorithm: Set Norms

The Set-Norm: $H_S(n)$



H_S Norm: We have established that: $I_S(p_{max}) \leq H_S$, and

$$\lim_{n\to\infty} \sqrt{(2/3)} / |S| \leq Q_S(n) \leq p_{max} \leq 2 / |S|;$$ Let $|S| = 256$, then:

Maximum value of H_S:

$$\lim_{n\to\infty} p_{max} \leq \sqrt{(2/3)} / 256 = 0.816/256,\ I_S = -\log_2(0.816/256) = 8.29,$$

Maximum: $\lim_{n\to\infty} H_S < 8.29$

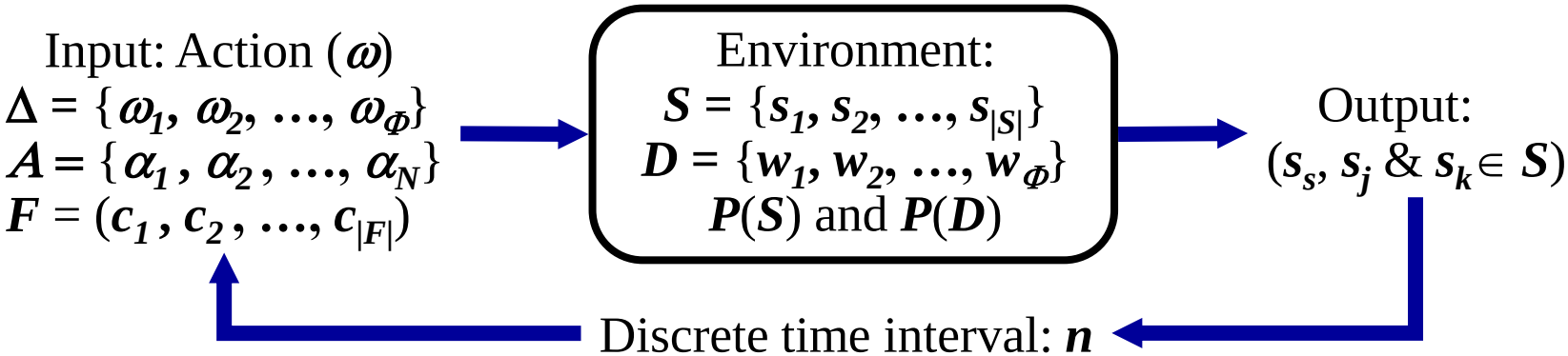
Minimum value of H_S: $\lim_{n\to\infty} p_{max} \leq 2 / 256,\ I_S = -\log_2(2/256) = 7,$

Minimum: $\lim_{n\to\infty} H_S > 7.$

$$\lim_{n\to\infty} 7 < H_S(n) < 8.2;\ e \to 0$$

S&M algorithm: Set Norms

The Set-Norm: $Q_S(n)$



$Q_S(n) = \{\ldots\} + p_{s_s}^2 + p_{s_j}^2 + p_{s_k}^2$; split set s_s into two half and merge s_j with s_k:

$$Q_S(n+1) = \{\ldots\} + + 1/4 \,.\, p_{s_s}^2 + 1/4 \,.\, P_{s_s}^2 + (p_{s_j} + p_{s_k})^2$$

$$Q_S(n+1) = \{\ldots\} + + 1/2 \,.\, p_{s_s}^2 + p_{s_j}^2 + 2\, p_{s_j}\, p_{s_k} + p_{s_k}^2$$

$$Q_S(n) - Q_S(n+1) = p_{ss}^2 + p_{s_j}^2 + p_{s_k}^2 - 1/2 \,.\, p_{s_s}^2 - p_{s_j}^2 - 2\, p_{s_j}\, p_{s_k} - p_{s_k}^2$$

$$= 1/2 \,.\, p_{s_s}^2 - 2\, p_{s_j}\, p_{s_k}$$

For expediency, $E\,[Q_S(n) - Q_S(n+1)] > 0$;

then: $E\,(1/2 \,.\, p_{s_s}^2) > E\,(2\, p_{s_j}\, p_{s_k})$

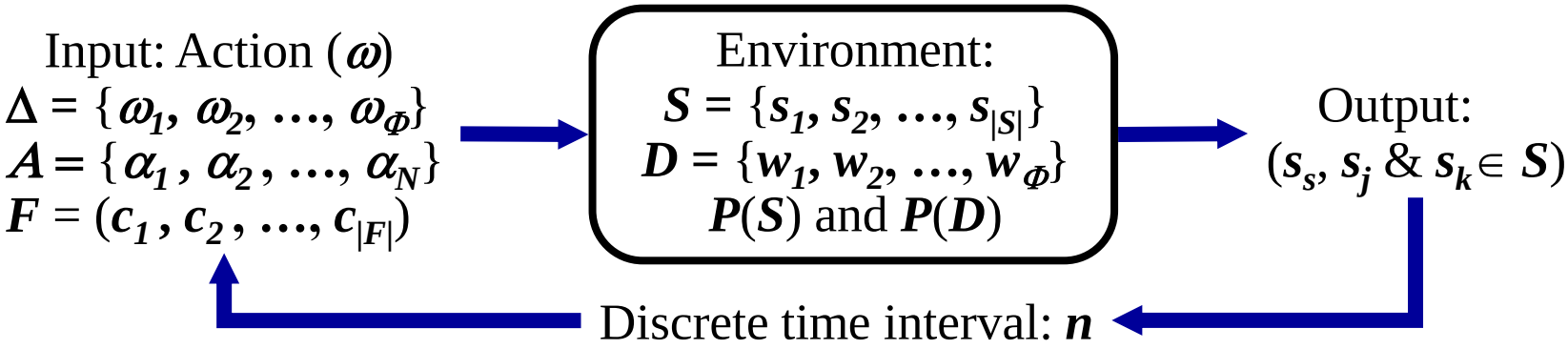
$E\,(1/4 \,.\, p_{s_s}^2) > E\,\{[1/(1-|s|)] \,.\, [1/(1-|s|)]\}$

$E\,(p_{s_s}^2) > E\,(4 / |s|^2)$

$$\text{As } n \to \infty,\ Q_S(n) \le p_{max} \to 2 / |S|$$

S&M algorithm: Set Norms

The Set-Norm: $Q_S(n)$



For simplicity, assume, s_s, s_j and s_k are singleton sets. Let p_1, p_2 and p_3 equal to $p(w_1)$, $p(w_2)$ and $p(w_3)$ respectively, where $w_1 \in s_s$, $w_2 \in s_j$ and $w_3 \in s_k$. In the process of splitting, p_1 will doubled, while, p_2 and p_3 will be halved in the merging process.

$$Q_S(n) = \{\ldots\} + p_1^2 + p_2^2 + p_3^2$$

$$Q_S(n+1) = \{\ldots\} + (2 \cdot p_1)^2 + (1/2 \cdot p_2 + 1/2 \cdot p_3)^2$$

$$= \{\ldots\} + 4 p_1^2 + 1/4 p_2^2 + 2/4 p_2 p_3 + 1/4 p_3^2$$

$$Q_S(n+1) - Q_S(n) = + 3 p_1^2 - 3/4 p_3^2 - 2/4 p_2 p_3 - 3/4 p_3^2$$

For expediency, $E[Q_S(n+1)) - Q_S(n] > 0$;

then: $E(3 p_1^2) > E(3/4 p_2^2) + E(2/4 p_2 p_3) + E(3/4 p_3^2)$

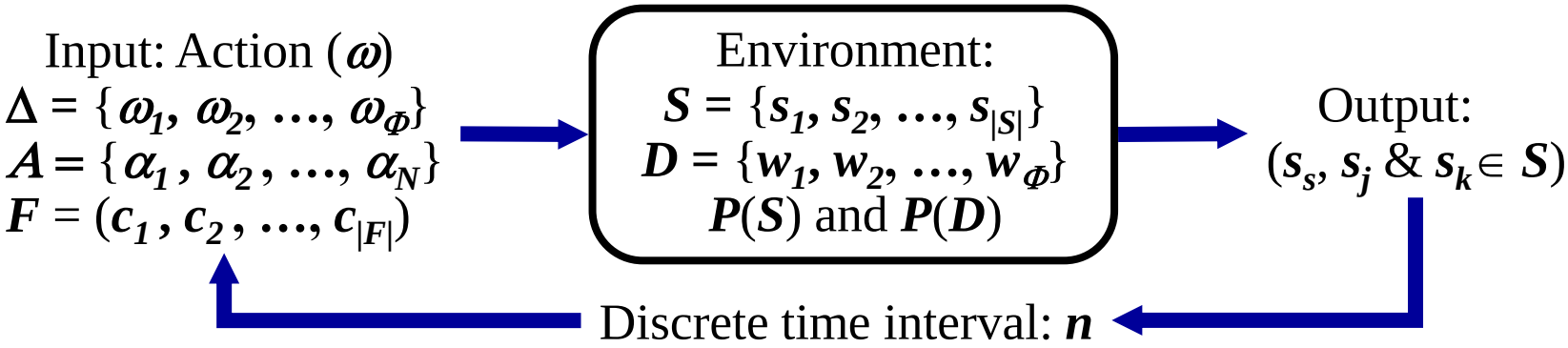
$E(3 p_1^2) > E\{[3/4(1-|S|)^2] + [2/4(1-|S|)^2] + [3/4(1-|S|)^2]\}$

$E(p_1^2) > E(2/3 |S|^2)$

$$\text{As } n \to \infty,\ 2/|S| \le Q_S(n) \le p_{max} \le \sqrt{(2/3)}/|S|$$

S&M algorithm: Dictionary Norms

The Word-Norm: $H_W(n)$



H_W Norm: We have established that: $I_D\,(p_{max}) \le H_D$, and

$\lim_{n\to\infty} \sqrt{(1/2)} / |\Phi| \le p_{max}$; Let $|\Phi| = 256$, then:

Maximum value of H_W:

$\lim_{n\to\infty} p_{max} \le \sqrt{(1/2)} / 256 = 0.707/256$, $I_S = -\log_2(0.707/256) = 8.5$,

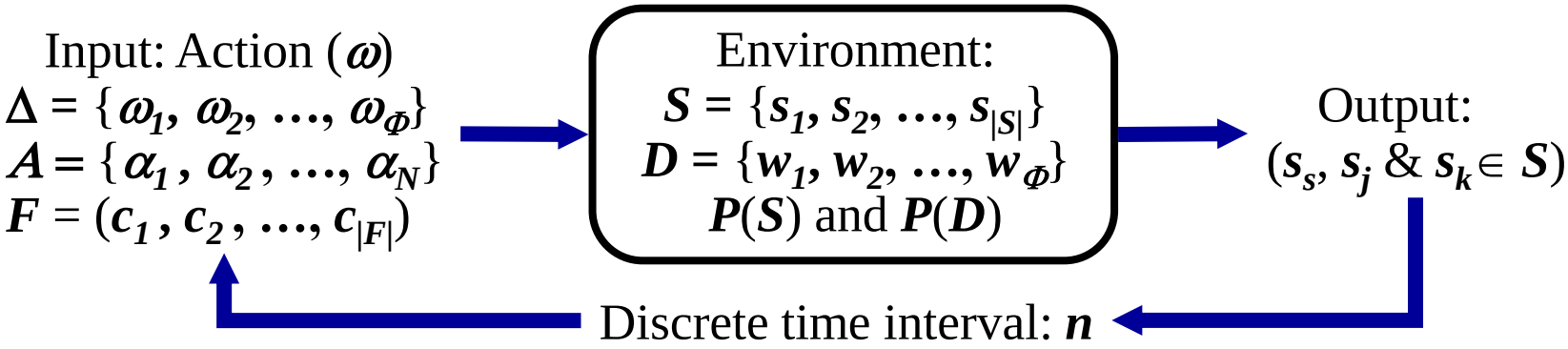
Maximum: $\lim_{n\to\infty} H_W < 8.5$

Minimum value of H_W: $\lim_{n\to\infty} p_{max} \le 2 / 256$, $I_S = -\log_2(2/256) = 7$,

Minimum: $\lim_{n\to\infty} H_S > 7$.

$$\lim_{n\to\infty} 7 < H_W(n) < 8.5;\ e \to 0$$

S&M algorithm: Dictionary Norms
The Word-Norm: $Q_W(n)$



For simplicity, assume, s_s, s_j and s_k are singleton sets. Let p_1, p_2 and p_3 equal to $p(w_1)$, $p(w_2)$ and $p(w_3)$ respectively, where $w_1 \in s_s$, $w_2 \in s_j$ and $w_3 \in s_k$. In the process of splitting, p_1 will doubled, while, p_2 and p_3 will be halved in the merging process.

$$Q_W(n) = \{\ldots\} + p_1^2 + p_2^2 + p_3^2$$

$$Q_W(n+1) = \{\ldots\} + (2 \cdot p_1)^2 + (1/2 \cdot P_2)^2 + (1/2 \cdot p_3)^2$$

$$= \{\ldots\} + 4\,p_1^2 + 1/4\,p_2^2 + 1/4\,p_3^2$$

$$Q_W(n+1) - Q_W(n) = + 3\,p_1^2 - 3/4\,p_3^2 - 3/4\,p_3^2$$

For expediency, $E\,[Q_W(n+1)) - Q_D(n] > 0$;

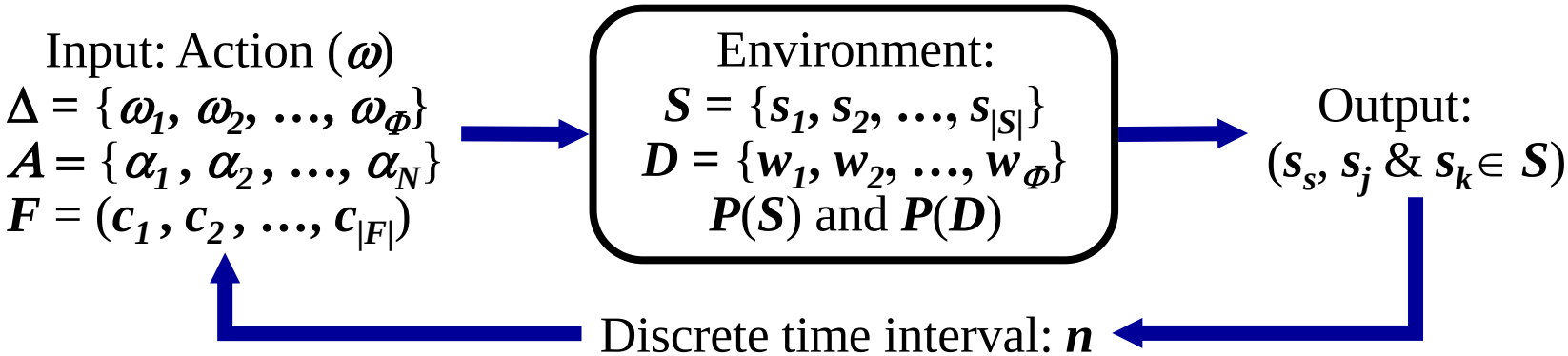
then: $E\,(3\,p_1^2) > E\,(3/4\,p_2^2) + E\,(3/4\,p_3^2)$

$E\,(3\,p_1^2) > E\,\{[3/4(1-|\Phi|)^2] + [3/4(1-|\Phi|)^2]\}$

$E\,(p_1^2) > E\,(1/2\,|\Phi|^2)$

$$\text{As } n \to \infty,\ Q_W(n) \le p_{max} \to \sqrt{(1/2)} / |\Phi|$$

S&M algorithm: the asymptotic case
-The Practical Simulation-



Assumptions: In the asymptotic case we assume that: $|s_i| >> |F|$, (i =1, …, $|S|$). Throughout the process, the value of $|s_i|$ remain greater than unity.

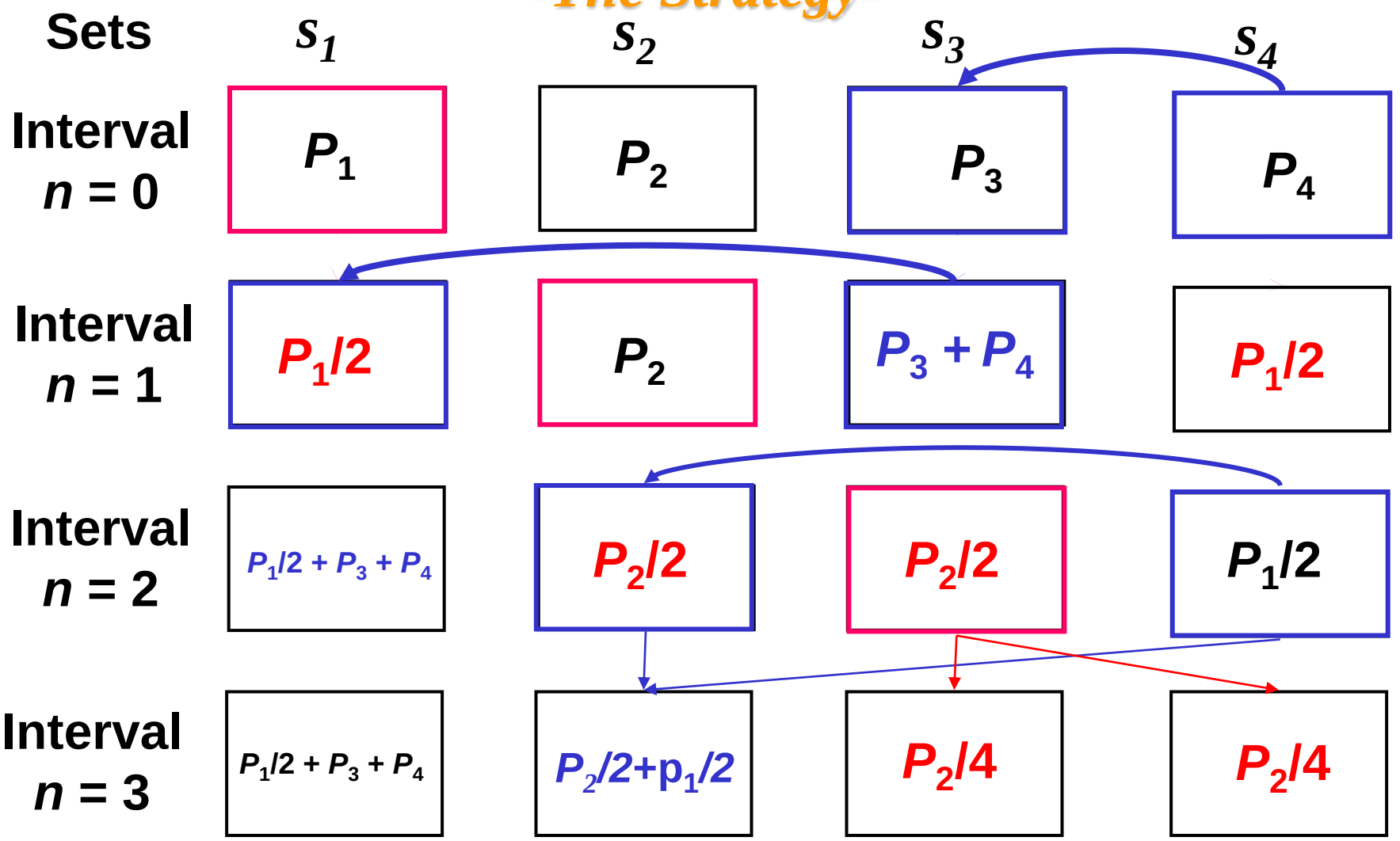
Initial Conditions: The initial $p[s_1(0)]$ is fixed to one of ten values in the range of $\{0.001 \leq p[s_1(0)] \leq 1\}$ and all other set's probabilities is made to be equal to:

$$\{1 - p[s_1(0)]\} / (1 - |S|).$$

Results: Since we assumed that, $|s_i| >> |F|$, then: ***D-Norms*** is not applicable. The behaviour of the algorithm is evaluated by ***S-Norms*** only, by plotting the average values of $p(s_1(n))$, $Q_S(n)$ and $H_S(n)$ over hundred **(100)** trials, for every one of the ten predetermined different set of initial probabilities. Where:

$$Q_S(n) = \sum_{i=1}^{|S|} p[s_i(n)]^2; \qquad H_S(n) = \sum_{i=1}^{|S|} p[s_i(n)] \log_2(1/p[s_i(n)])$$

S&M algorithm: the asymptotic case
-The Strategy-



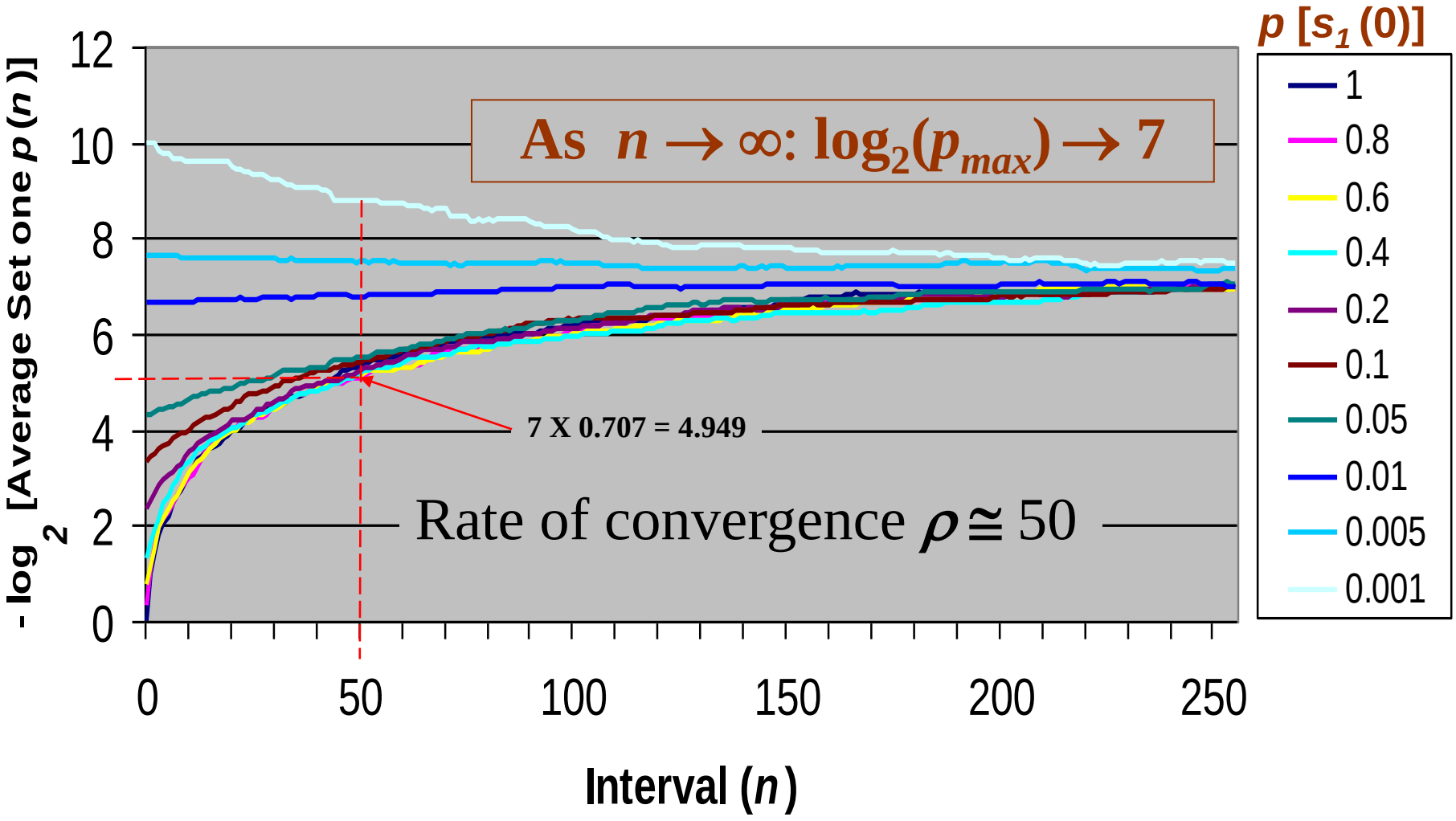
For large values of *n*, set probabilities will converge to $2/|S|$.

S&M algorithm: the asymptotic case

The S-Norms

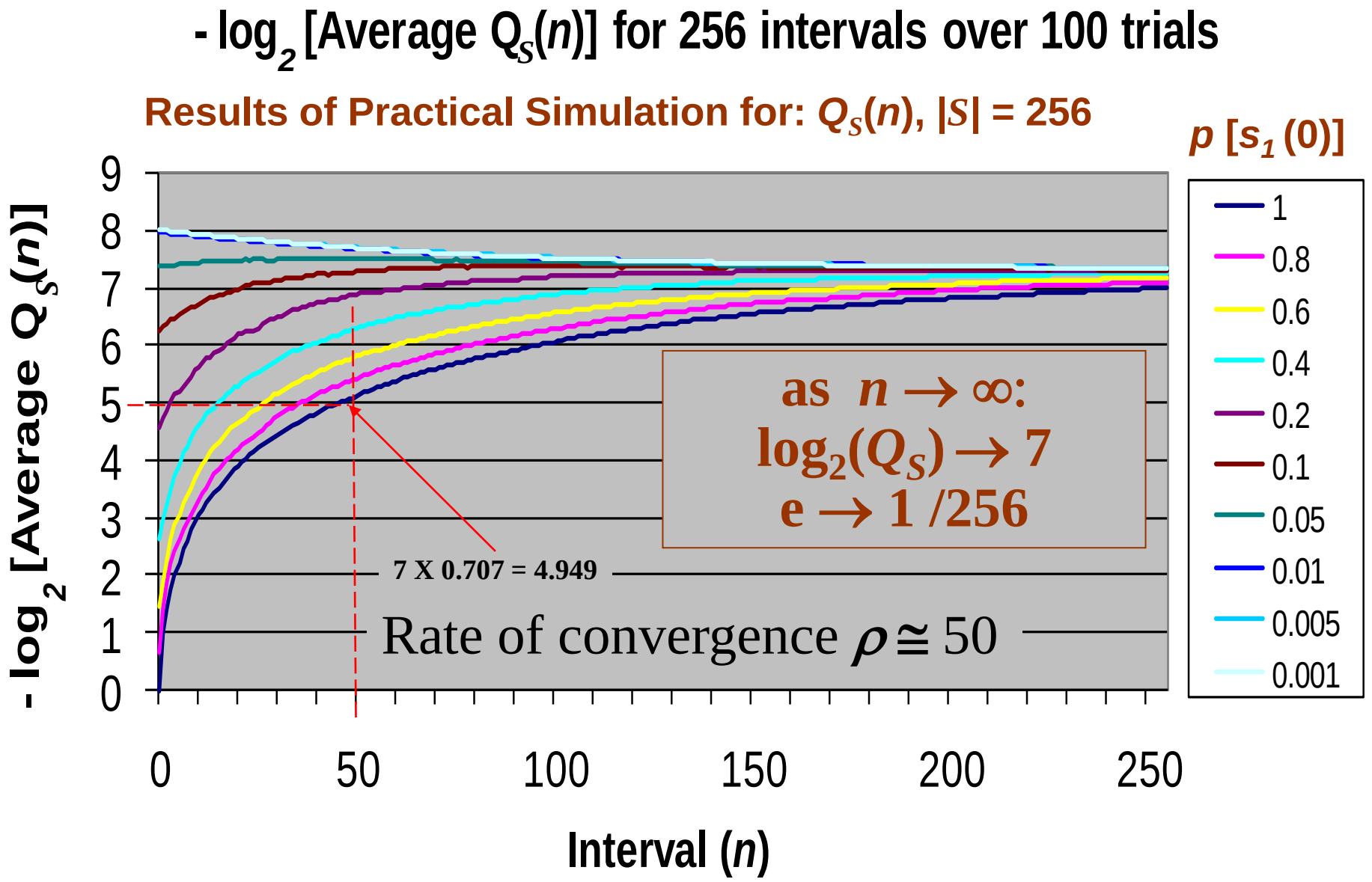
- $\log_2$[Average Set One $p(n)$] for 256 intervals over 100 trials

Results of Practical Simulation for: First Set Prob, $|S| = 256$



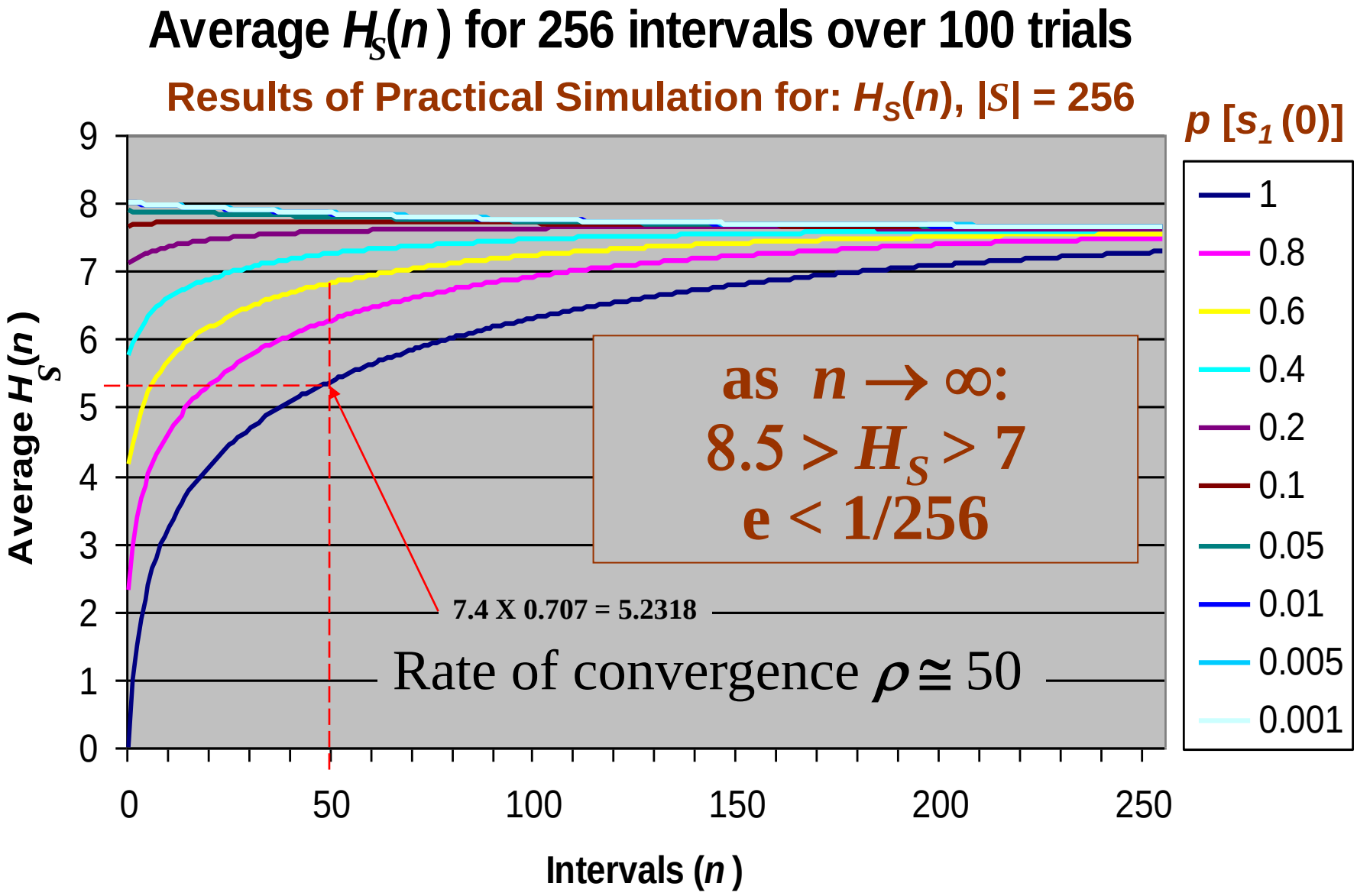
S&M algorithm: the asymptotic case

The S-Norms



S&M algorithm: the asymptotic case

The S-Norms

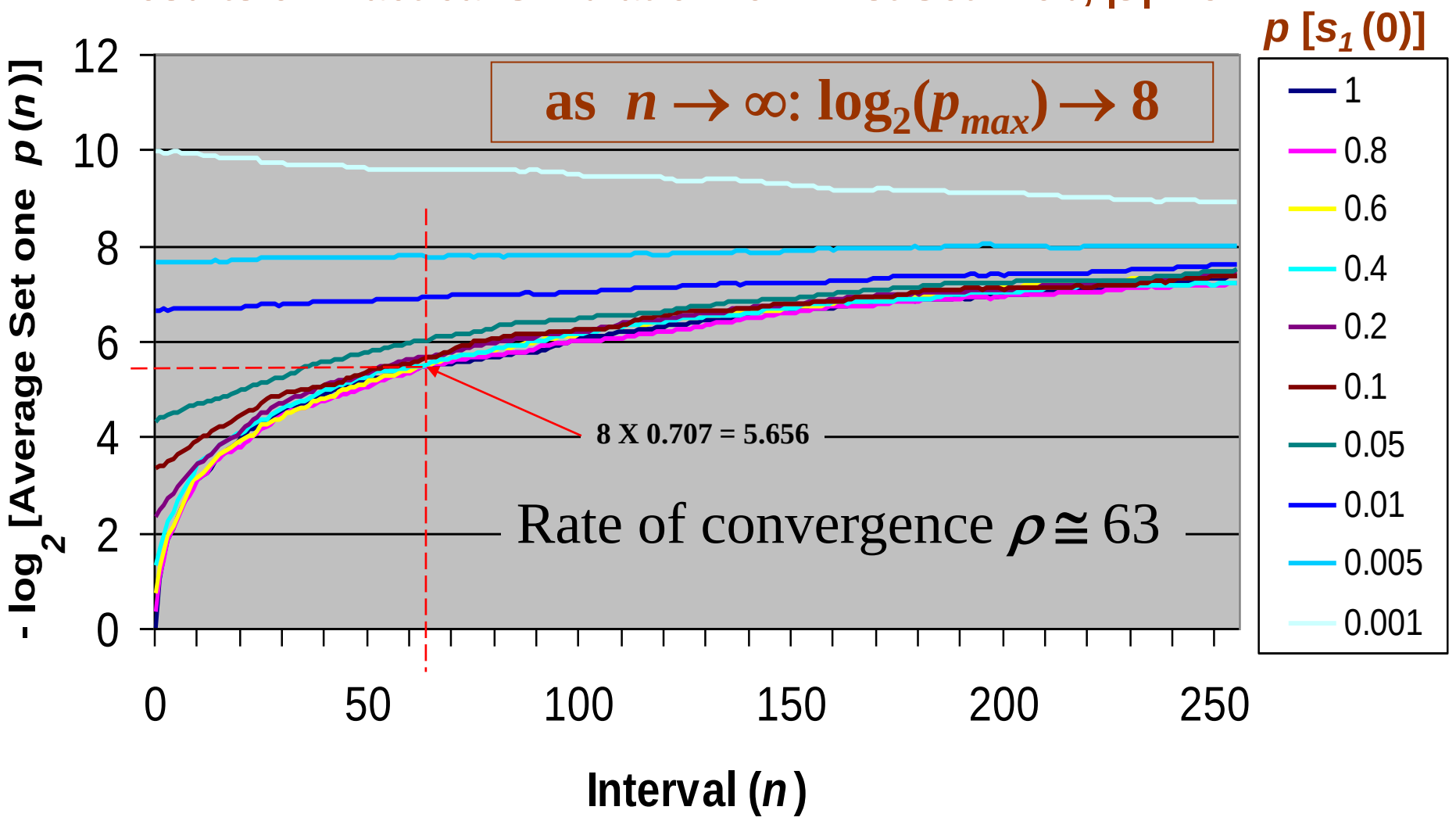


S&M algorithm: the asymptotic case

The S-Norms

- $\log_2$ [Average Set One $p(n)$] for 256 intervals over 100 trials

Results of Practical Simulation for: First Set Prob, $|S| = 512$

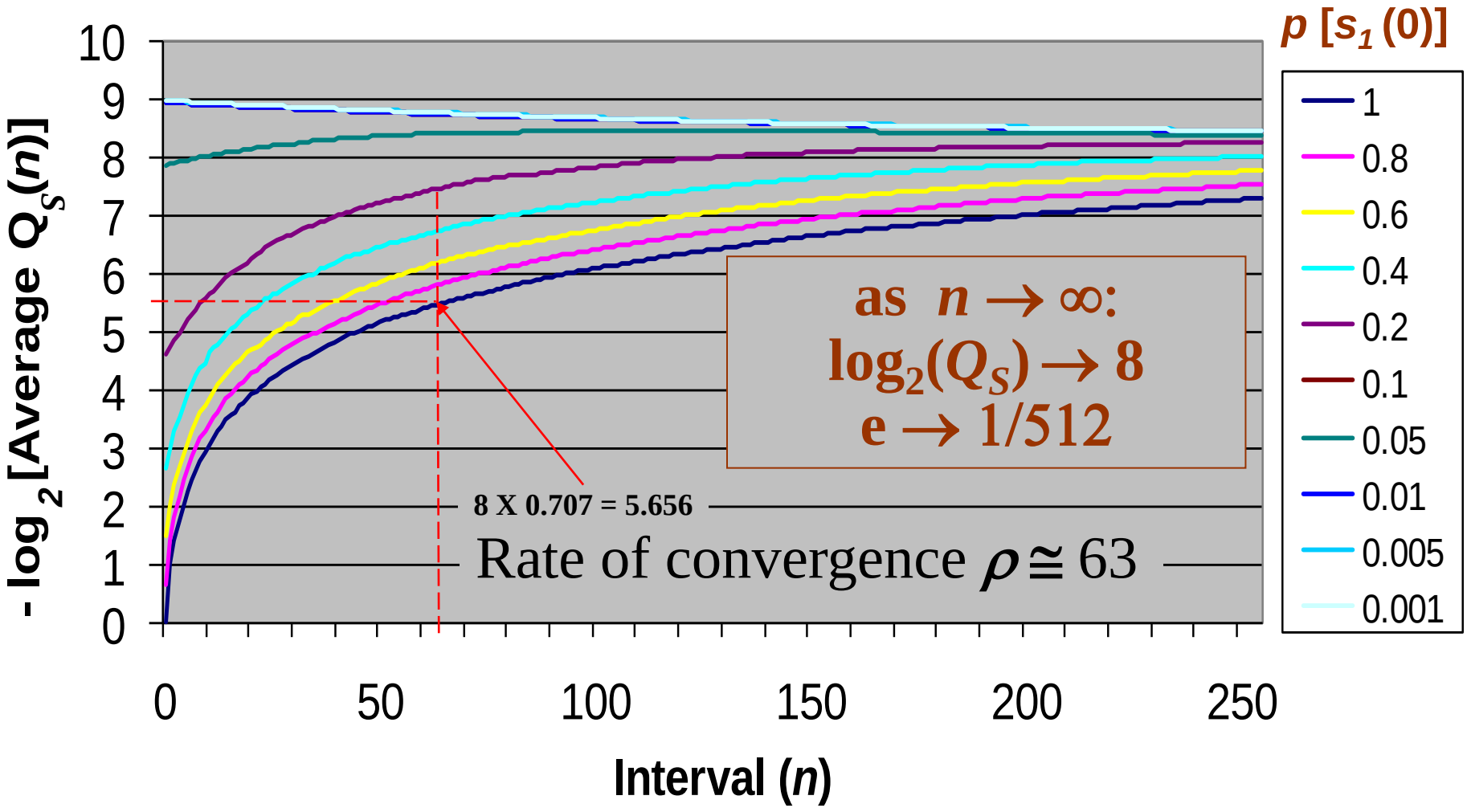


S&M algorithm: the asymptotic case

The S-Norms

- $\log_2$ [Average $Q_S(n)$] for 256 intervals over 100 trials

Results of Practical Simulation for: $Q_S(n)$, $|S| = 512$

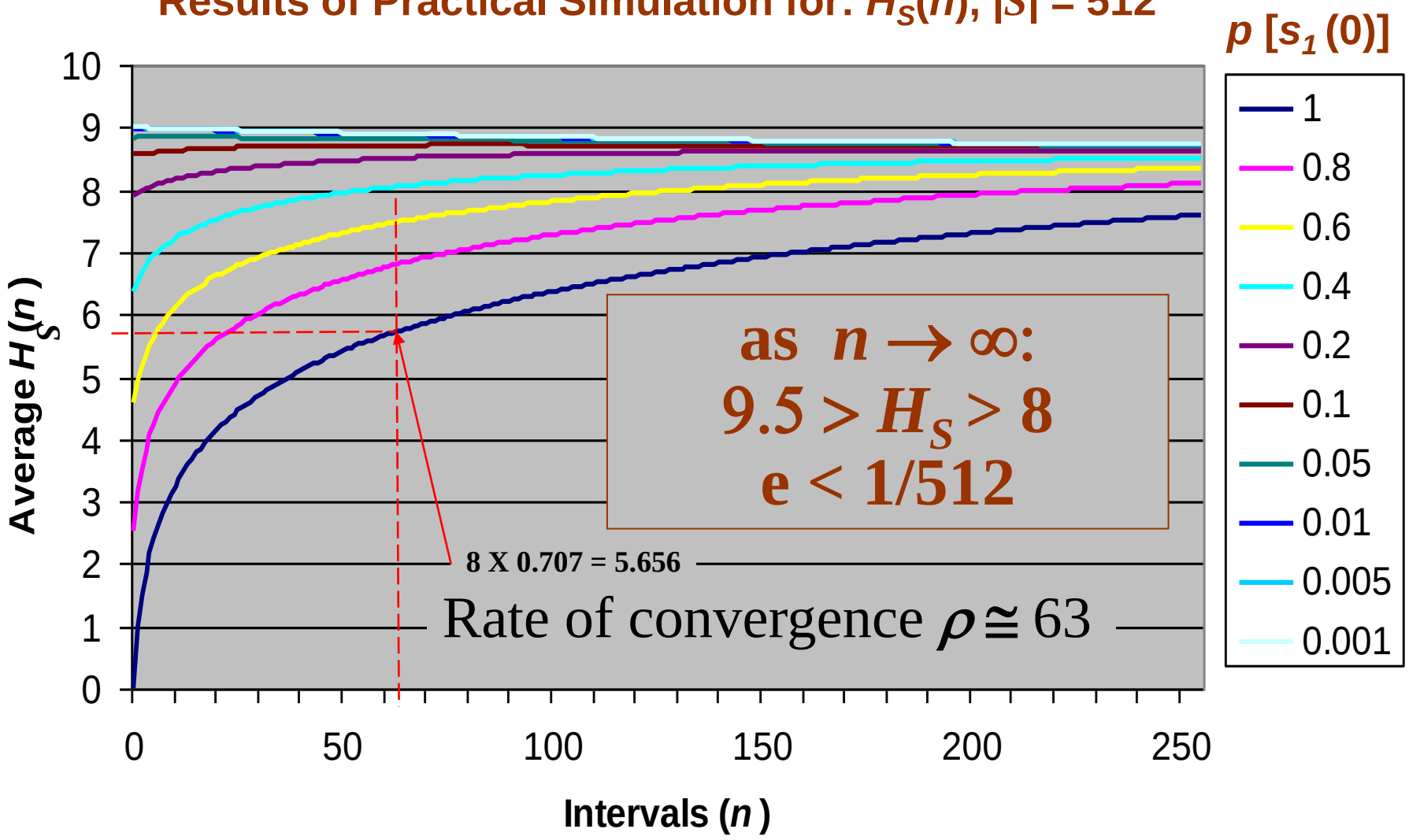


S&M algorithm: the asymptotic case

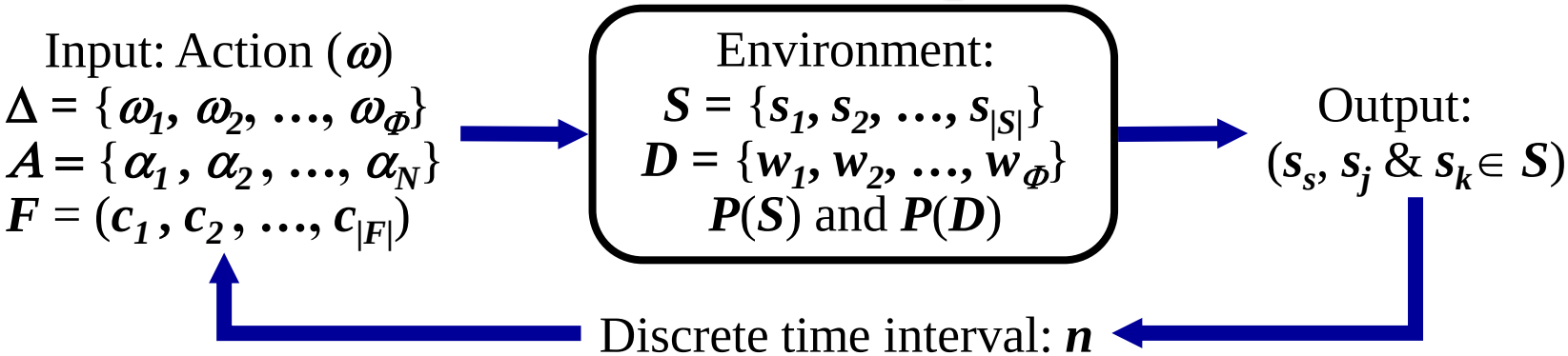
The S-Norms

Average $H_S(n)$ for 256 intervals over 100 trials

Results of Practical Simulation for: $H_S(n)$, $|S| = 512$



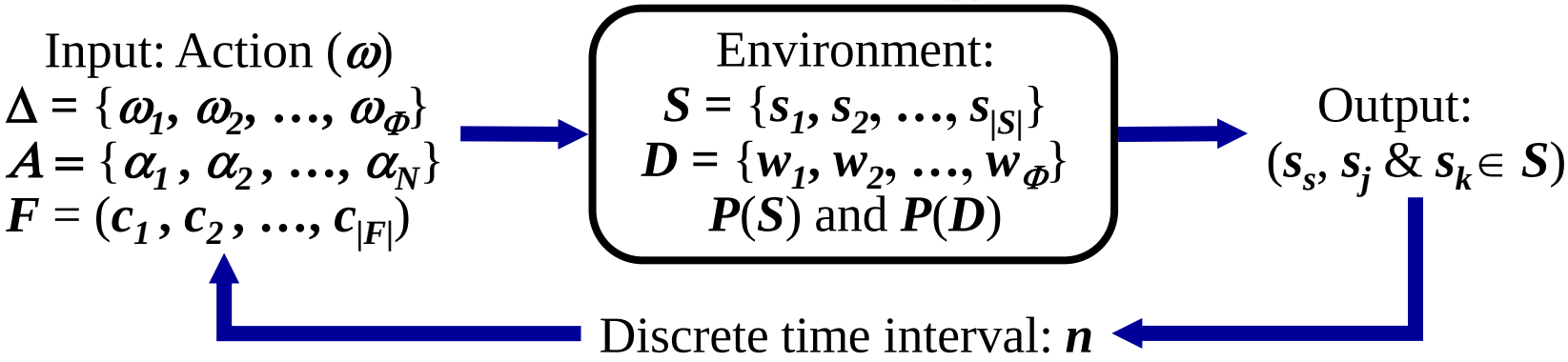
S&M algorithm: the finite case
-The Concept-



Satellite Sets ({}): In the asymptotic case, it was assumed that, the size of the sub sets $|s_i| >> |F|$, (i =1, …, $|S|$), therefore, the process of splitting can continued with no limitation. However, practically $|s_i|$ have a finite size, usually < $|F|$. The splitting process therefore is halted when $|s_i| = 1$. To overcome this problem, the concept of satellite set is introduced. A satellite is a data-empty set, carries no information, (a ***null*** set denoted by ***{}***) corresponding to an empty word, (a ***null*** word denoted by Λ), in the dictionary D. A set with one word ($|s_i| = 1$) known as a ***singleton*** set. A singleton set with satellites called parent singleton of a group of satellite. The number of satellites in the group called ***block length*** of s_i and denoted by $SBL(s_i)$ is equal to (2^r-1), (where r is a positive integer). $\{SBL(s_i) + 1\}$ is called the family block length of s_i and denoted by $\{FBL(s_i)\}$,

where: $1 \leq FBL(s_i) \leq (|S|/2); FBL(s_i) = 2^r, r = 0, 1, 2, \ldots, \log_2(|S|/2)$.

S&M algorithm: the finite case
-The Strategy-



Matching, Merging and Splitting:

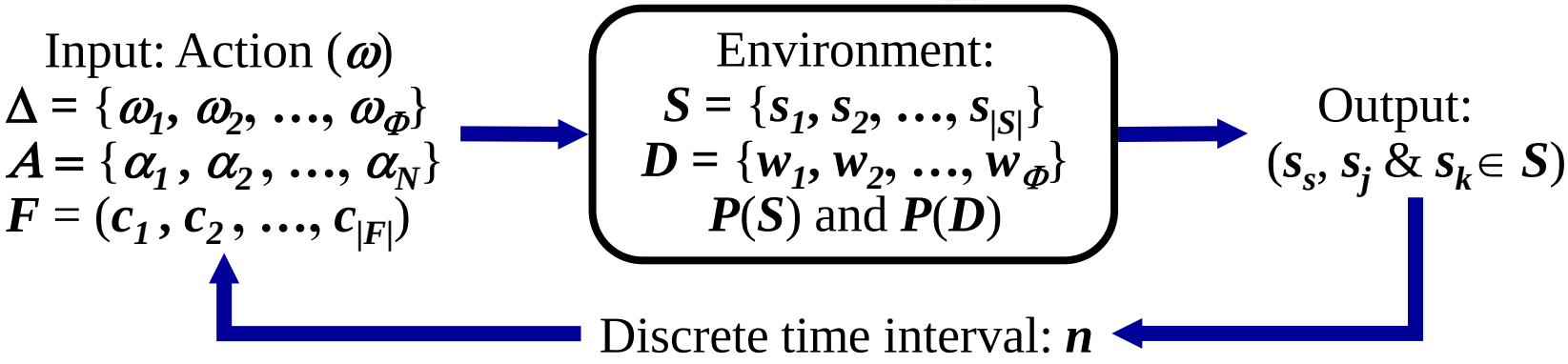
1. **Matching**: At interval n, match the n^{th} action $\omega_s(n)$ of the input string with the longest word in the dictionary, denoted as $w_s(n)$. $w_s(n) \in s_s(n)$.

2. **Merging**: two sets are randomly selected (say, s_j and s_k, $j \neq k \neq s$). Satellite sets take the identity of its parent singleton (Family Leader). The merger is preformed as follows:

   i. Both s_j and s_k are non-empty sets and with no satellites: replace s_j by the union set ($s_j \cup s_k$).

   ii. s_j is a singleton with satellite/s: remove $FBL(s_j)/2$ satellites from its block, s_k remain unchanged.

S&M algorithm: the finite case
-The Strategy-



iii $\underline{s_k \text{ is alone an empty-set or a singleton with satellite/s}}$: remove $FBL(S_k)/2$ satellites from its block, s_j remain unchanged.

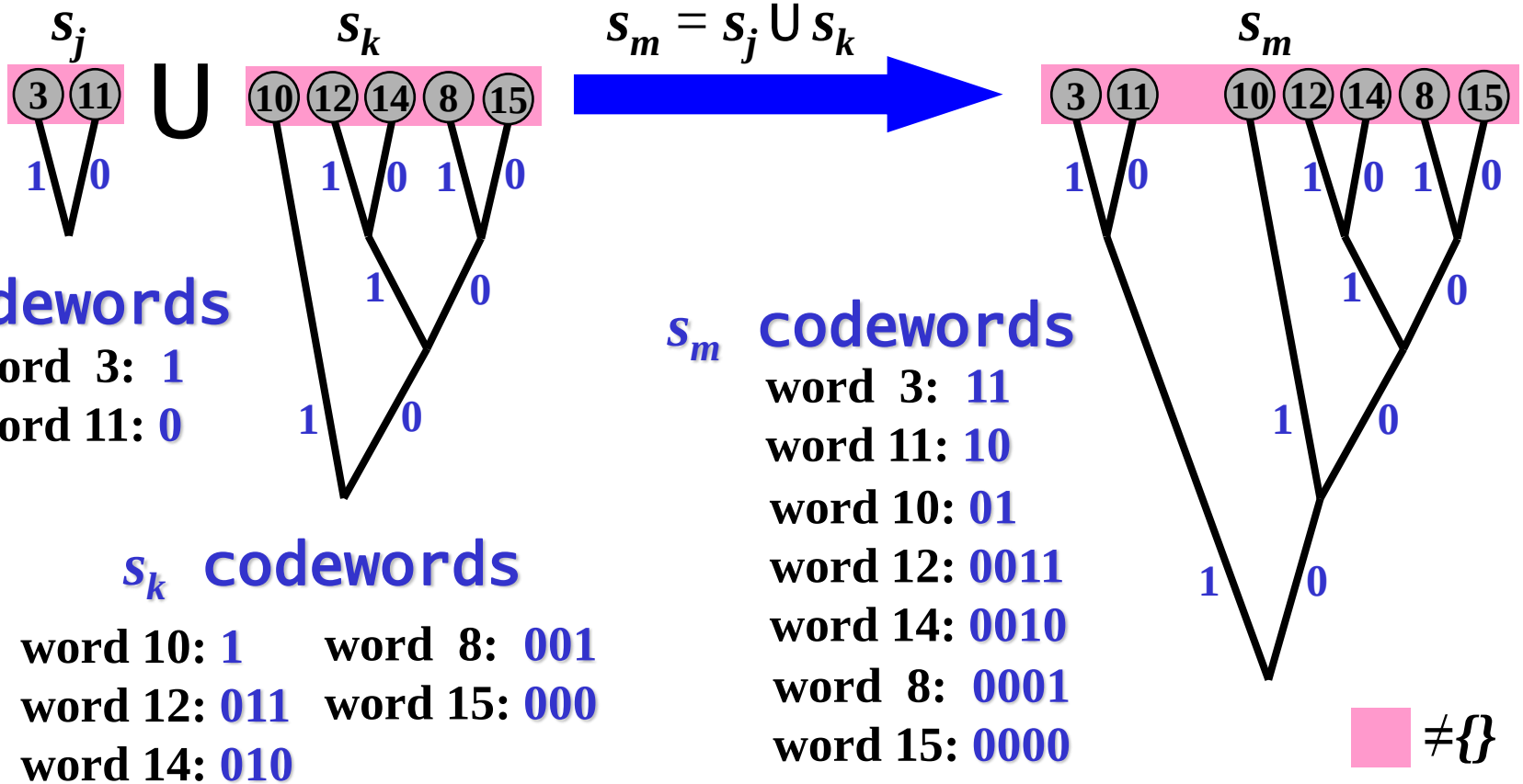
4 Splitting s_s:

i. $\underline{\text{Set size } |s_s| > 1}$: split s_s into two sets along its highest word link. (See diagram)

ii. $\underline{\text{Set } s_s \text{ is singleton with family block length } FBL(s_s)}$: add $FBL(s_s)$ satellite set . If $FBL(s_s)$ is greater than the available free set number, (free set number at the n^{th} interval is equal to the maximum number of sets minus the actual number of sets after the last merging step), then splitting procedure at this interval will be ignored.

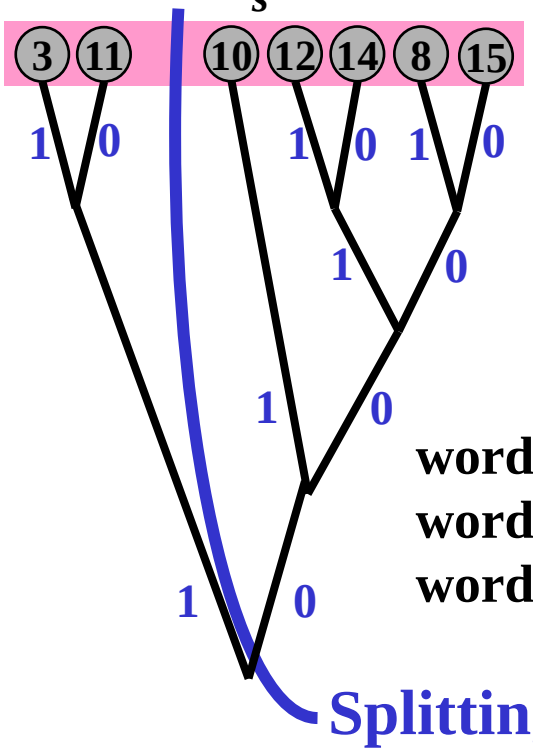
S&M algorithm: the finite case
-The Merger-

Non Empty Sets Merger: Merging two non-empty sets s_j and s_k into one set s_m, is a process of constructing the union of the two sets and updating the probabilities and codewords of all words within the set.



S&M algorithm: the finite case
-The Splitting-

Multi Word Sets Splitting: Splitting a multi word set s_s into two sets s_{s1} and set s_{s2}, along the highest splitting plane in the set, therefore most significant digit of s_s is removed, and the word probabilities been doubled.



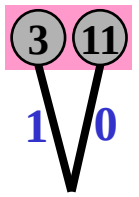
Splitting s_s into set s_{s1} & set s_{s2}

s_s codewords

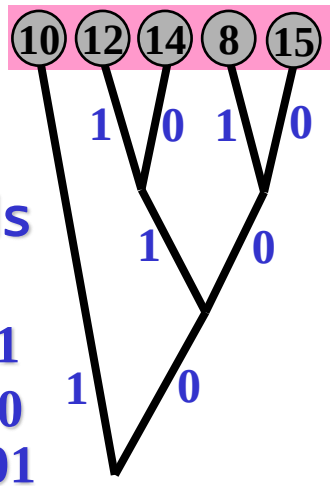
word 3: 11
word 11: 10
word 10: 01
word 12: 0011
word 14: 0010
word 8: 0001
word 15: 0000

s_{s1} codewords

word 3: 1
word 11: 0



&

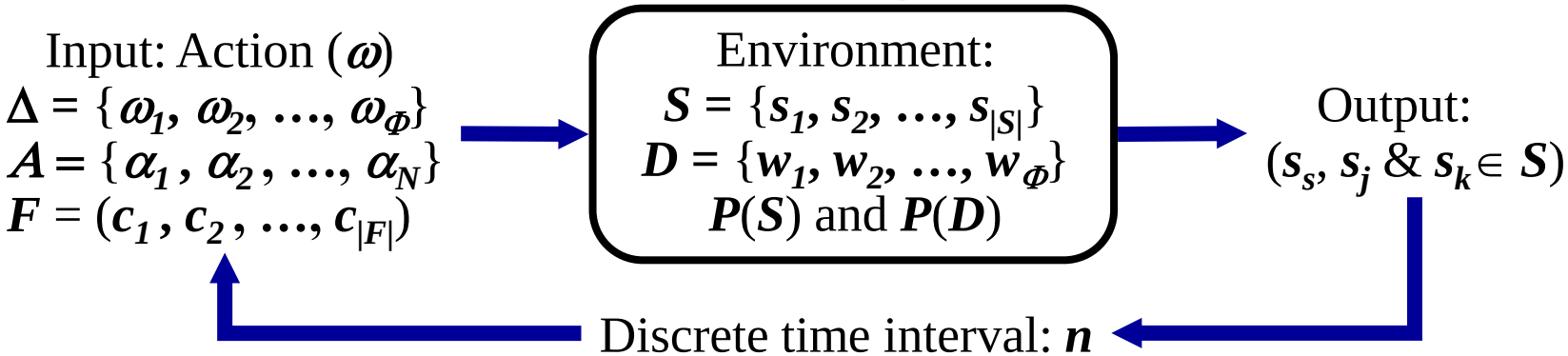


s_{s2} codewords

word 10: 1
word 12: 011
word 14: 010
word 8: 001
word 15: 000

$\neq\{\}$

S&M algorithm: the finite case
-The Probability Vectors-



The probability vector of the source dictionary Δ is:

$$(p(\omega_1),\ p(\omega_2),\ \ldots,\ p(\omega_\Phi)).$$

The state probability vector of the environment dictionary D, is:

$$(p(w_1),\ p(w_2),\ \ldots,\ p(w_\Phi))$$

The real probability of the set s_i is given by: $p_{real}(s_i) = \sum_{j=1}^{|S_i|} p(\omega_{ij}) / FBL(s_i)$.

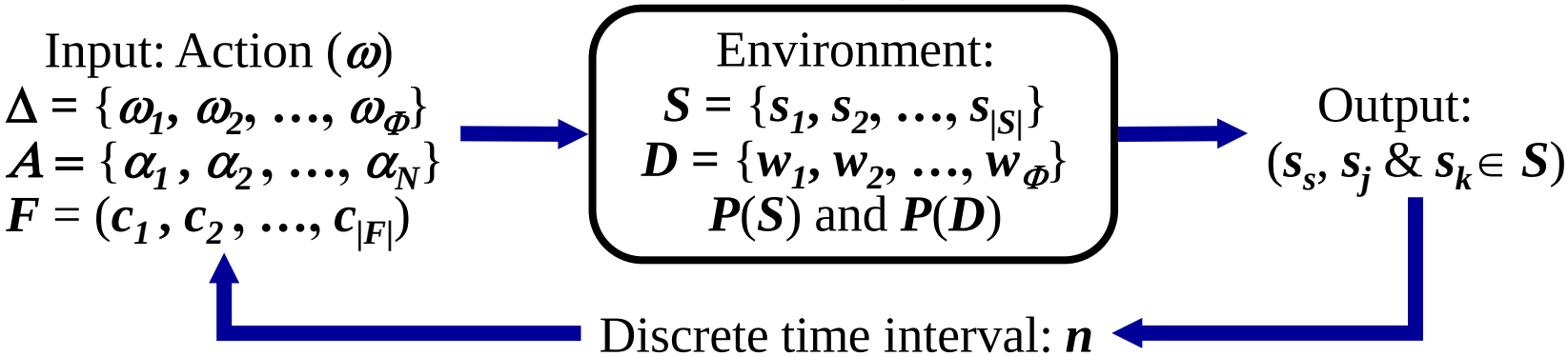
The state probability of the set s_i is given by: $p(s_i) = \sum_{j=1}^{|S_i|} p(w_{ij})$.

Set state probabilities $p(s_i)$ are equiprobable, equal to $(1 / |S|)$.

Since empty sets have zero frequency of accordance, their state probabilities, is therefore, added to their parent singleton.

Non empty set state probability is therefore: $p(s_i) = FBL(s_i) / |S|$.

S&M algorithm: the finite case
-The Probability Vectors-



The ***in-set*** probability vector $P_{in\text{-}set}$: is the state probabilities of words within a set s_i:

$$P_{in\text{-}set} = (p_{in\text{-}set}(w_{i1}),\ p_{in\text{-}set}(w_{i2}),\ \ldots,\ p_{in\text{-}set}(w_{i|s_i|})) \mid p_{in\text{-}set}(s_i) = \sum_{j=1}^{|S_i|} p_{in\text{-}set}(w_{ij}) = 1.$$

At the n^{th} interval the words ***in-set*** probabilities $p_{in\text{-}set}(w_i)$ are computed as follows:

1- Single word set: $p_{in\text{-}set}(w_1) = 1$.

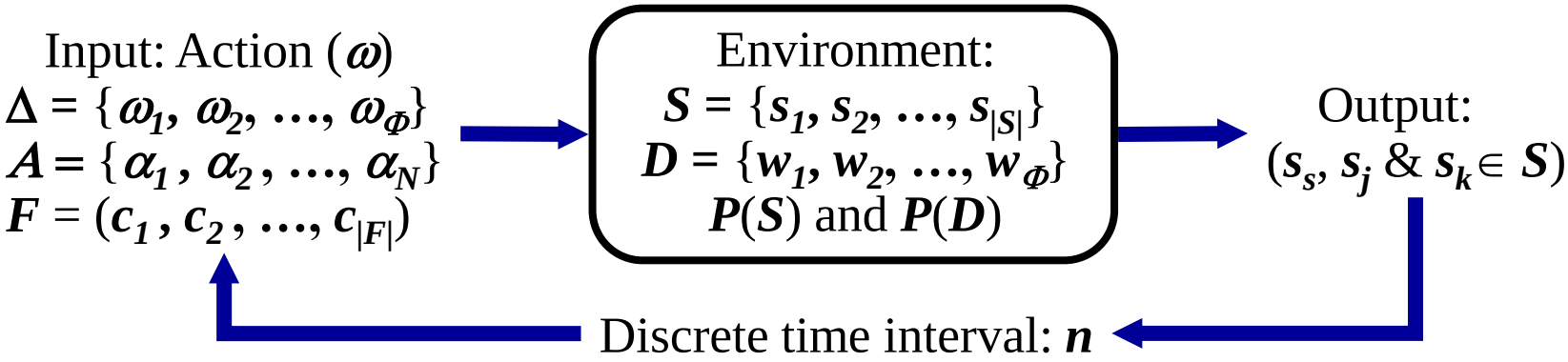
2- Two words set: $p_{in\text{-}set}(w_1) = p_{in\text{-}set}(w_2) = ½$.

3- More than two words in a set, their probabilities depend on the history of the set merger as shown in the diagram.

The state probability of a word $p(w_j) \in s_i$ is known as the ***out-set*** state probability vector given by the expression:

$$p(w_j) = \{ p_{in\text{-}set}(w_{ij}) \cdot FBL(s_i) \ / |S| \mid w_j \in s_i \}.$$

S&M algorithm: the finite case
-Initial Conditions-



Initial State: When $n = 0$, the environment have the following initial conditions:

1) The dictionaries contains single character words only, $\Phi = N$.

2) Each sub-set of S contains a single word of the dictionary $\Delta \mid \omega_i \in s_i$. $|S| = \Phi$.

3) $p_{real}(\omega_i)$ is determined by the type of the input source.

4) Set family $FBL(s_i) = 1$, where $i = 1,2, \ldots, |S|$, $p(s_i) = 1/\Phi$.

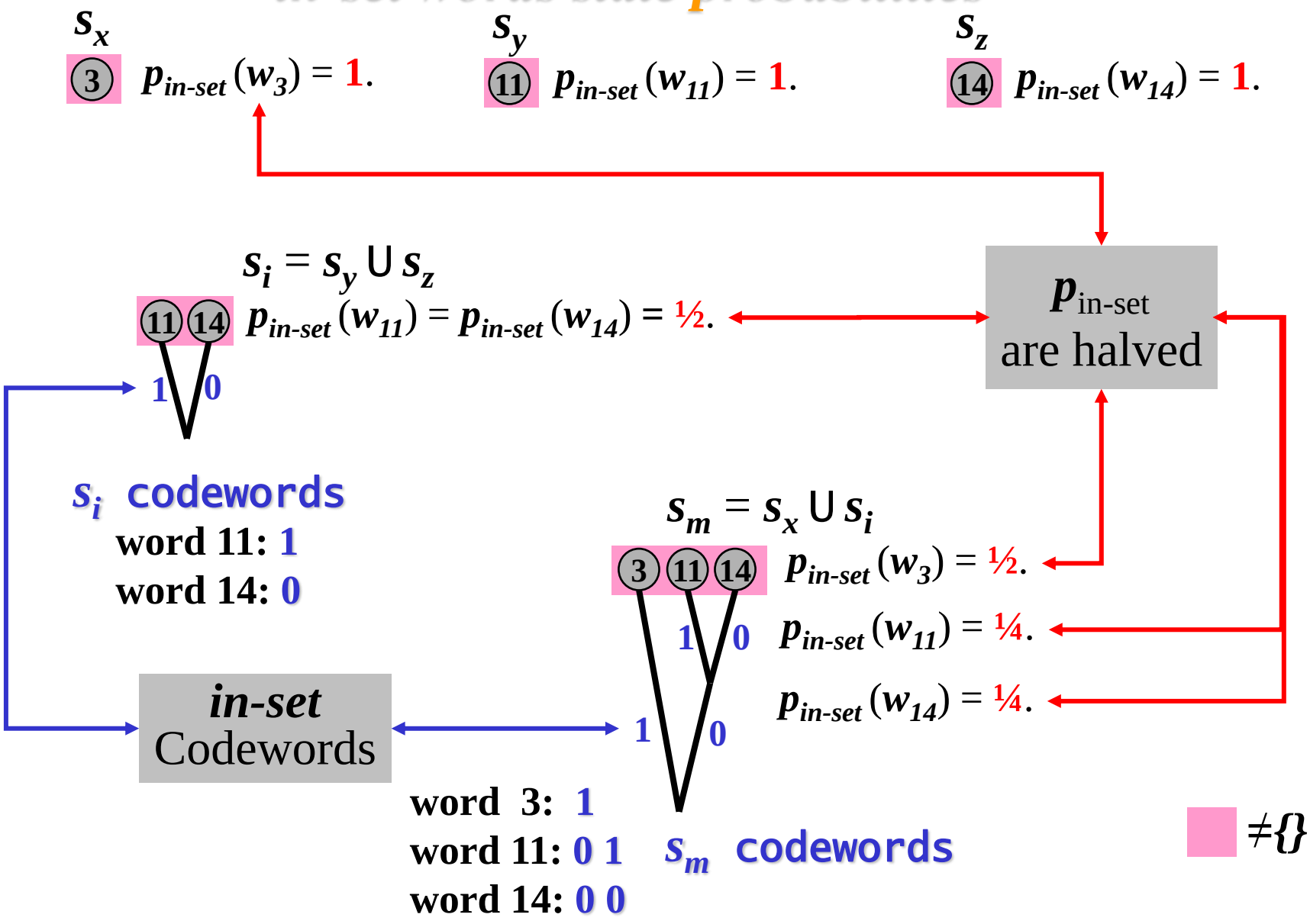
5) Since all words within the dictionaries at $n = 0$, all equiprobable, then:

$$Q_S(0) = 1 / N$$

$$H_S(0) = \log_2 (N)$$

$Q_W(0)$ and $H_W(0)$: are determined by the source statistical parameters.

S&M algorithm: the finite case
-in-set words state probabilities-



S&M algorithm: the finite case
-Word Coding-

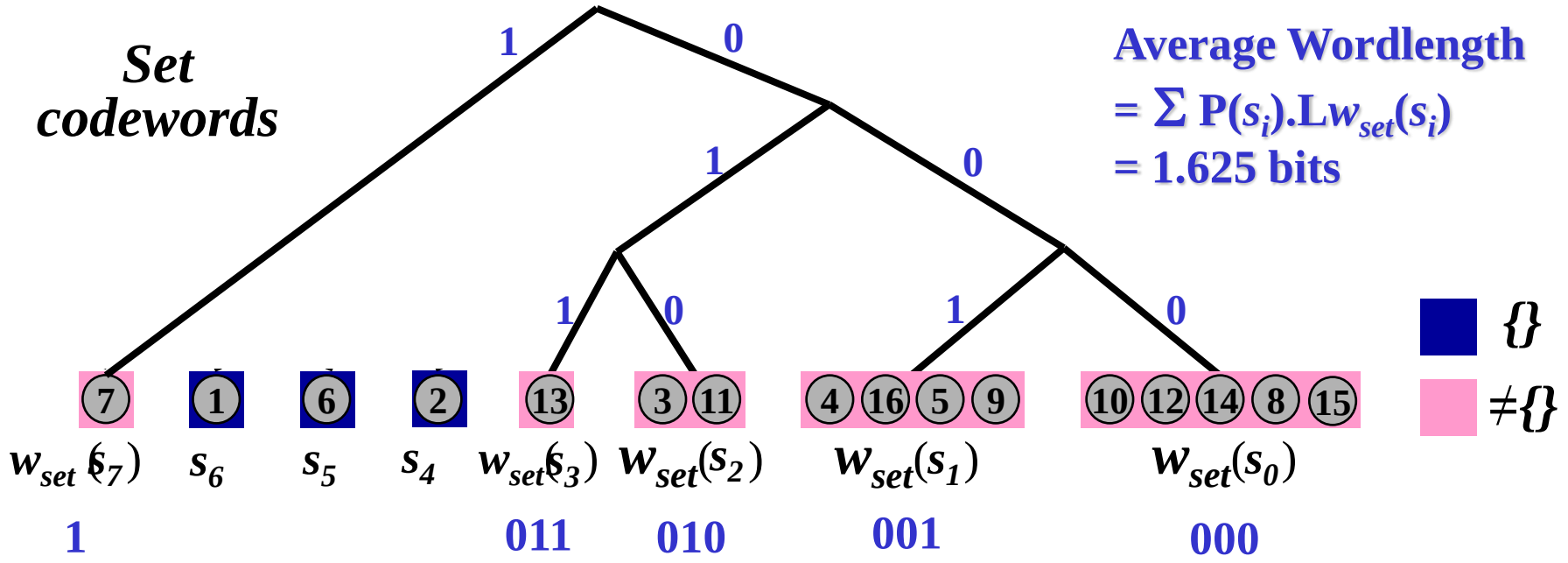
Set Coding:

Let ***D*** contains the following words:

$<w_7, w_1, w_6, w_2, w_{13}, w_3, w_{11}, w_4, w_{16}, w_5, w_9, w_{10}, w_{12}, w_{14}, w_8, w_{15}>$ partitioned into 8 sets: $(s_7, s_6, s_5, s_4, s_3, s_2, s_1, s_0)$; where:

$s_7 = \{w_7\}$; $s_6 = \{w_1\} = \{\}$; $s_5 = \{w_6\} = \{\}$; $s_4 = \{w_2\} = \{\}$; $s_3 = \{w_{13}\}$;

$s_2 = \{w_3, w_{11}\}$; $s_1 = \{w_4, w_{16}, w_5, w_9\}$; and. $s_0 = \{w_{10}, w_{12}, w_{14}, w_8, w_{15}\}$.



S&M algorithm: the finite case
-Words coding-

The word codeword

$w_{word} = w_{set}(s_i) + w_{inset}(v_i)$

String addition

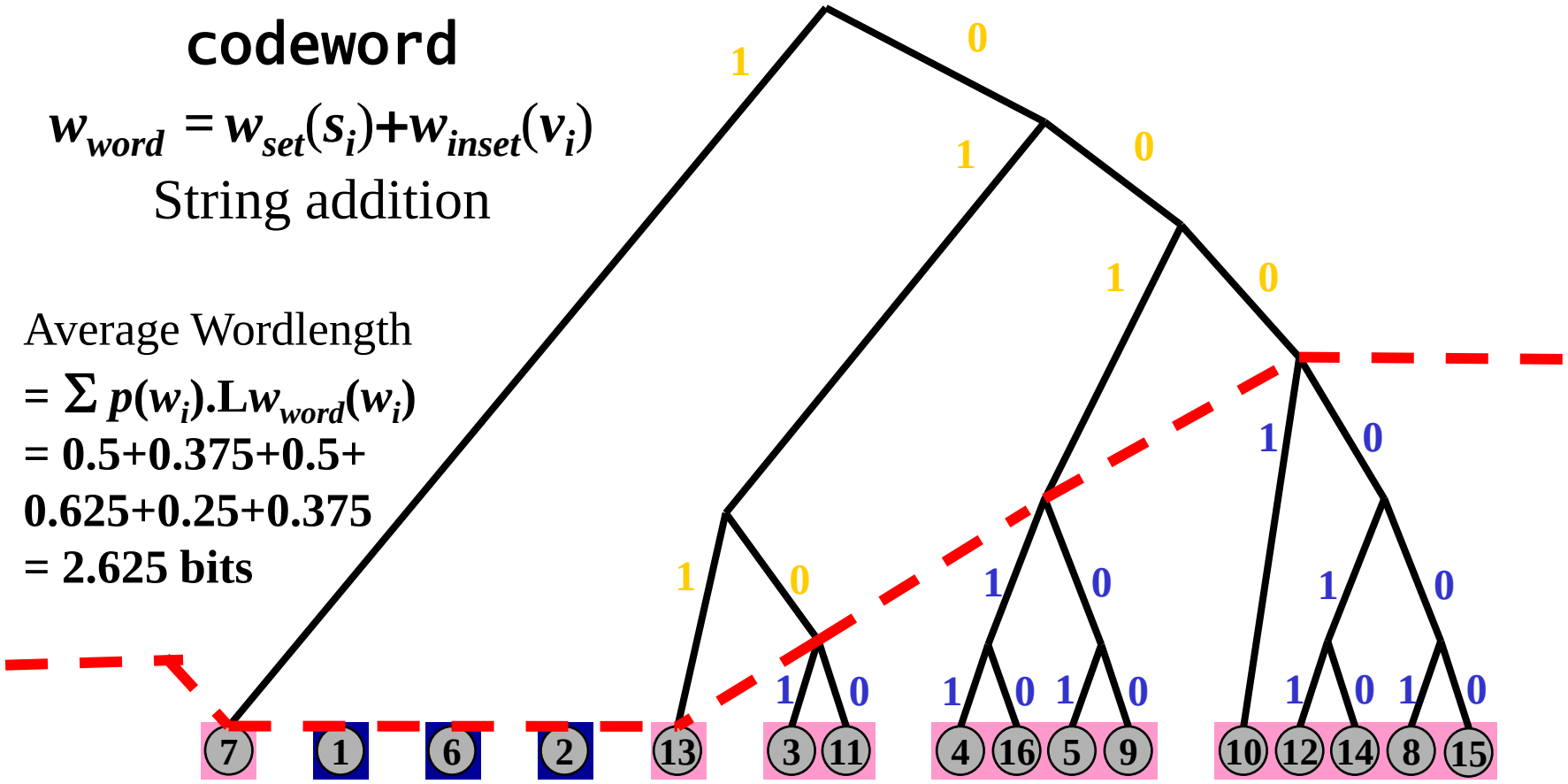
Average Wordlength

$= \sum p(w_i).Lw_{word}(w_i)$

= 0.5+0.375+0.5+

0.625+0.25+0.375

= 2.625 bits



s_7 s_6 s_5 s_4 s_3 s_2 s_1 s_0

word 7: 1
word 13: 011
word 3: 010 1
word 11: 011 0

word 4: 001 11
word 16: 001 10
word 5: 001 01
word 9: 001 00

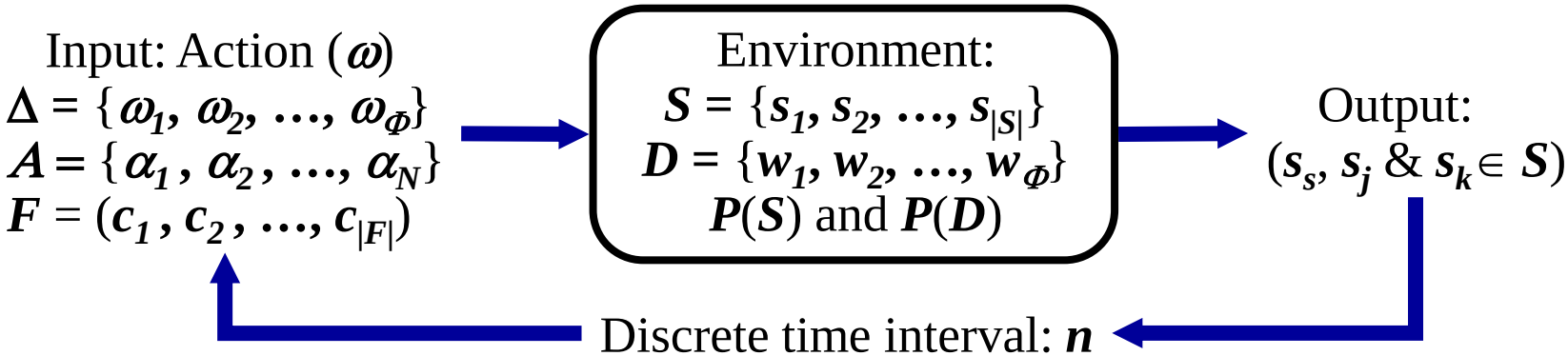
word 11: 000 1
word 12: 000 011
word 14: 000 010
word 8: 000 001

word 15: 000 000

{}

≠{}

S&M algorithm: the finite case
-The Practical Simulation-



Constants: In the finite case we assume:

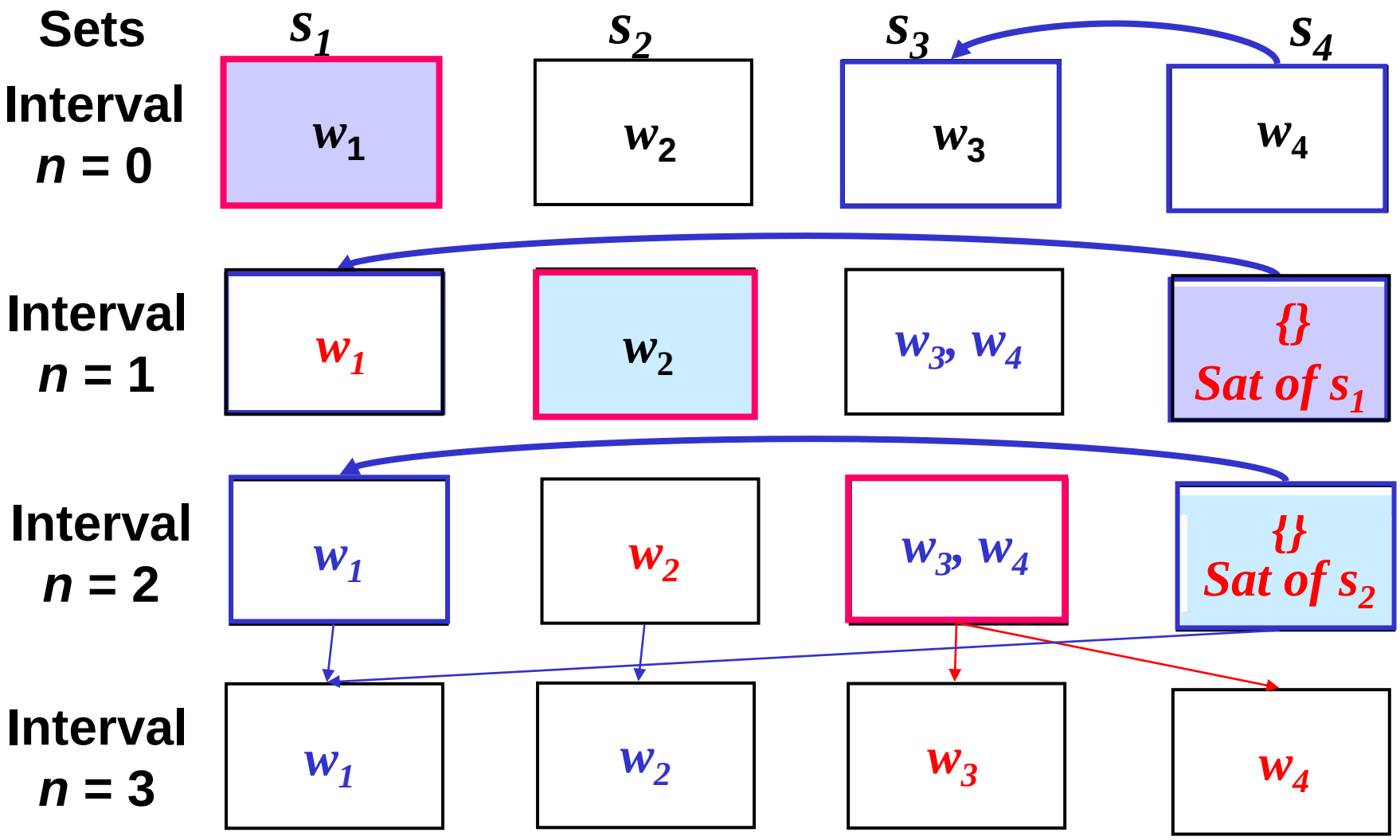
1) The word size in the dictionaries is one. i. e. $\Phi = N = 256$.
2) The algorithm tested for two sizes of sets: $|S| = 256$ and $|S| = 512$.
3) Two maximum number of intervals are used, $|F| = 256$ and 512.

Initial Conditions: The initial $p[s_1(0)]$ is set to one of ten values in the range of $\{0.001 \leq p[s_1(0)] \leq 1\}$ and all other set's probabilities is made to be equal to:

$$\{1 - p[s_1(0)]\} / (1 - |S|).$$

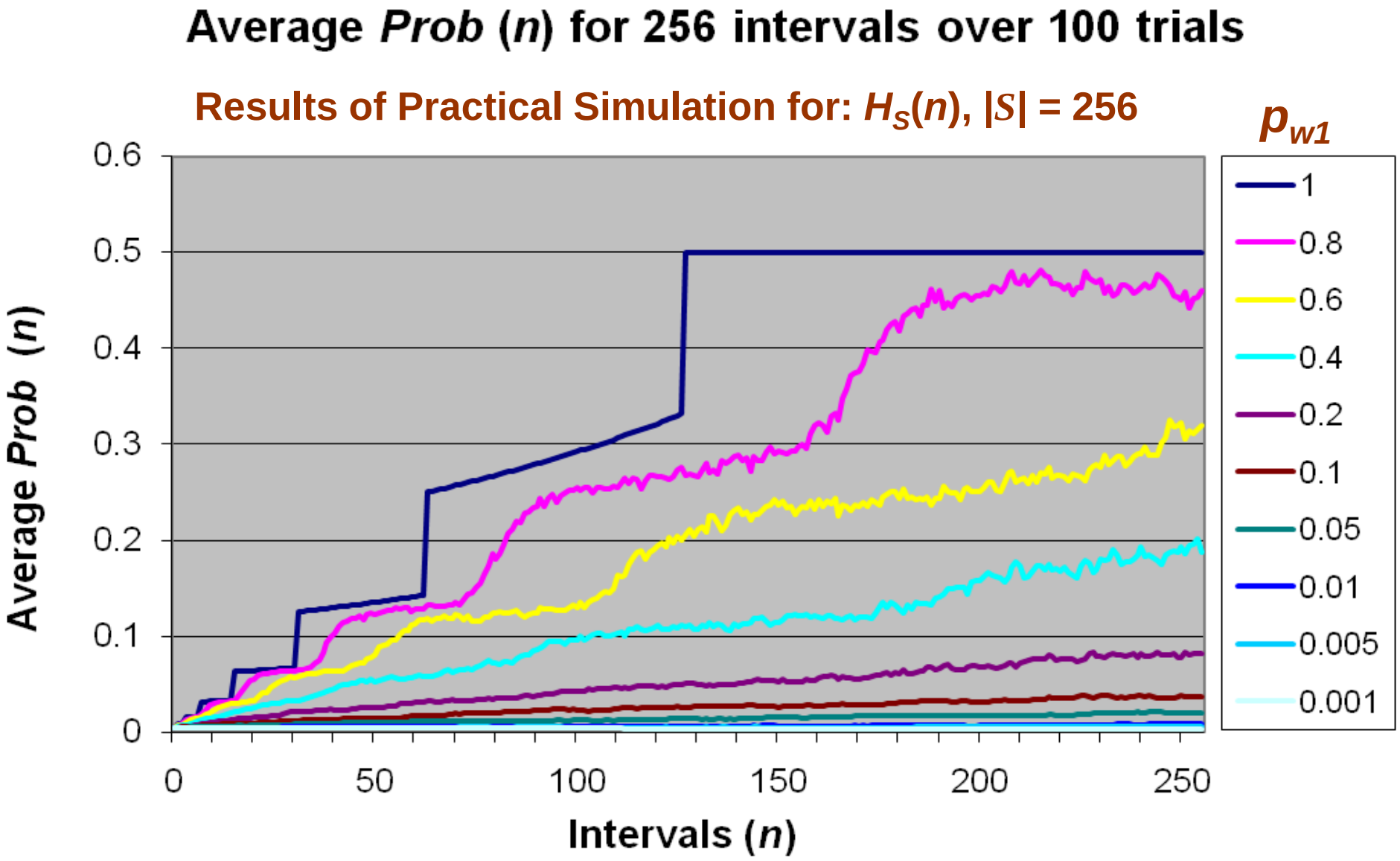
Results: The behaviour of the algorithm is determined by plotting the average values of $p(s_1(n))$, $Q_S(n)$, $H_S(n)$, $Q_D(n)$ and $H_D(n)$ over hundred **(100)** trials, for every one of the ten predetermined different set of initial probabilities.

S&M algorithm: the finite case

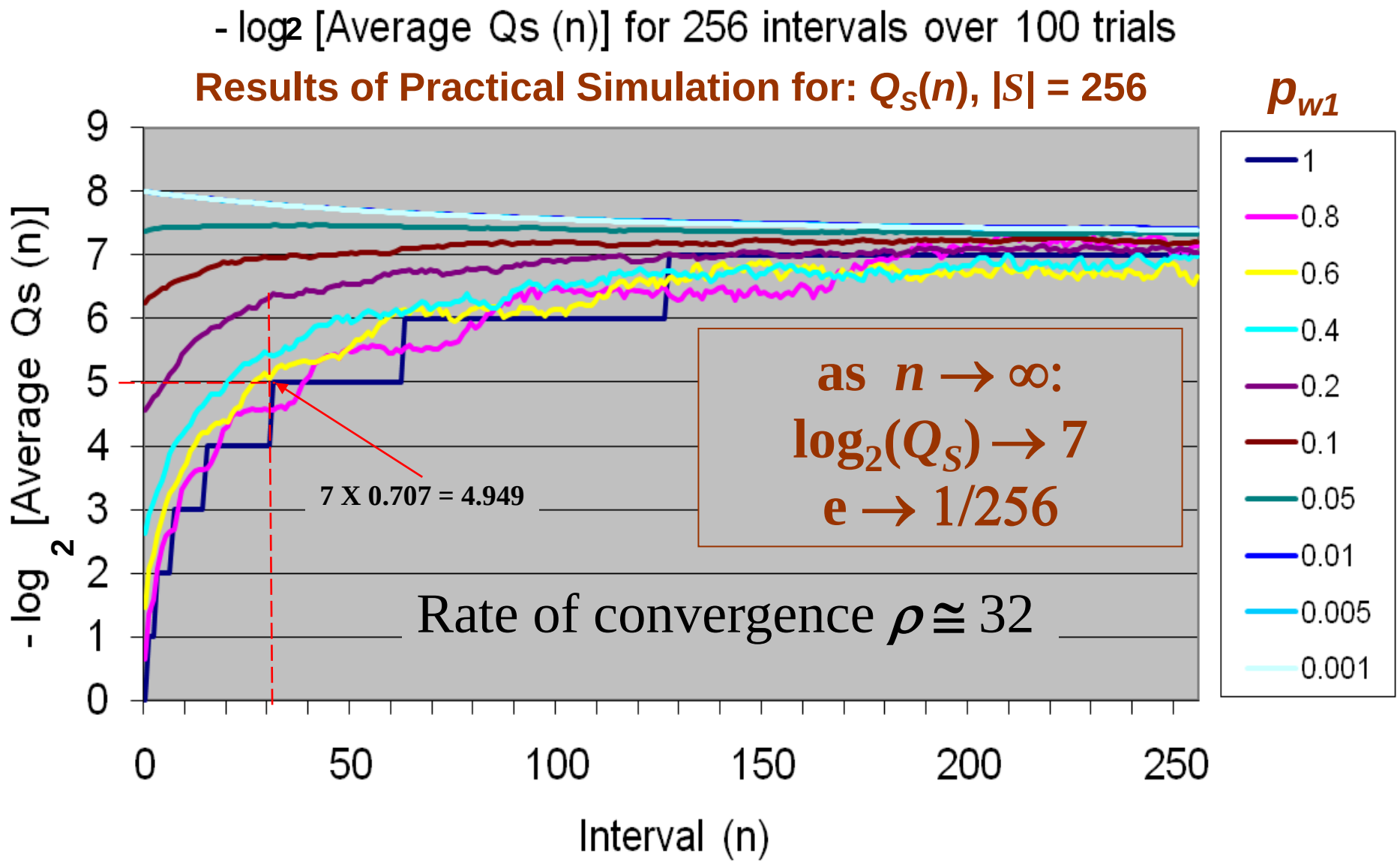


For large values of *n*, set probabilities will converge to $1/|S|$

S&M algorithm: the finite case
The S-Norms



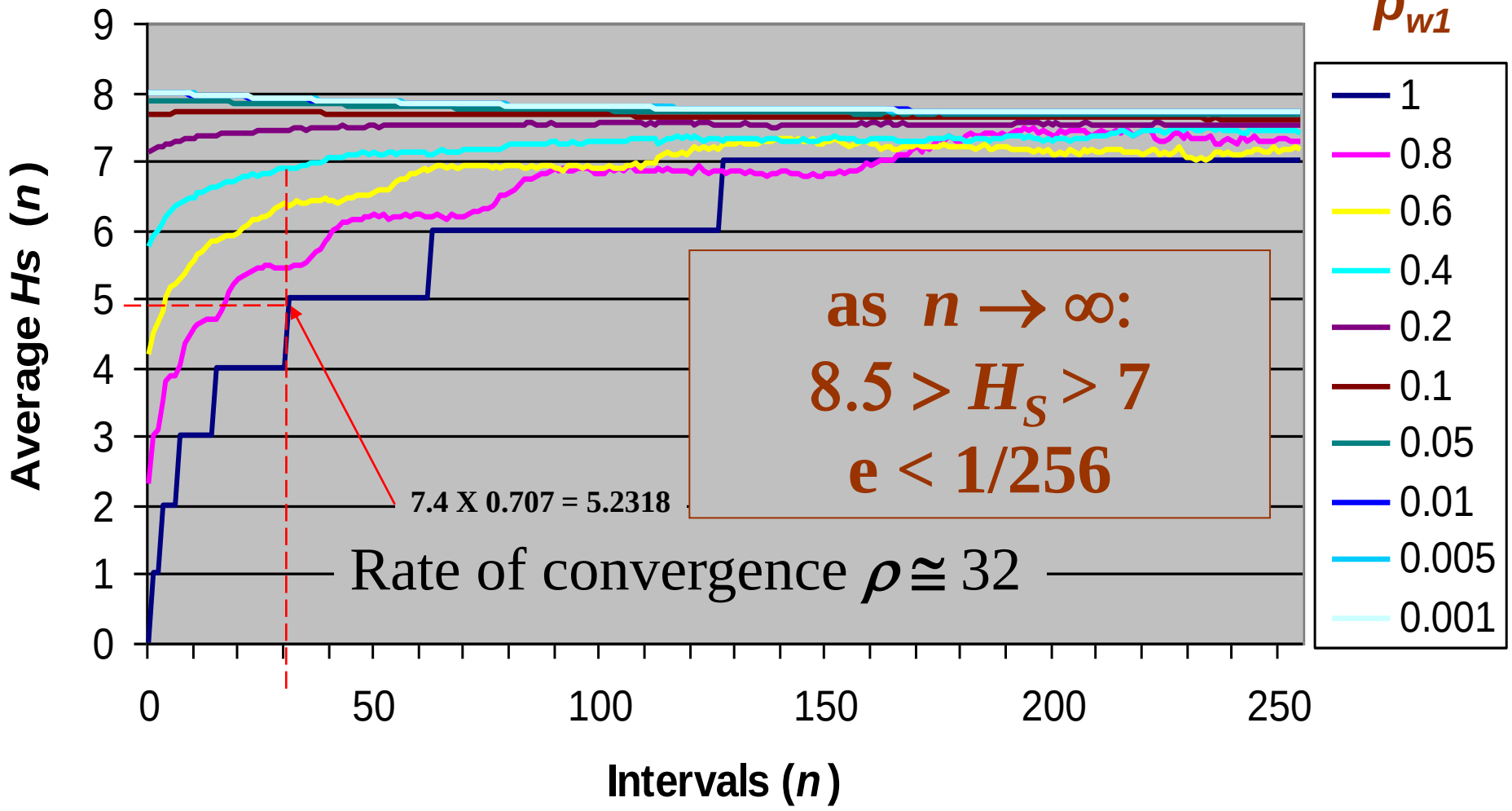
S&M algorithm: the finite case
The S-Norms



S&M algorithm: the finite case
The S-Norms

Average *Hs* (*n*) for 256 intervals over 100 trials

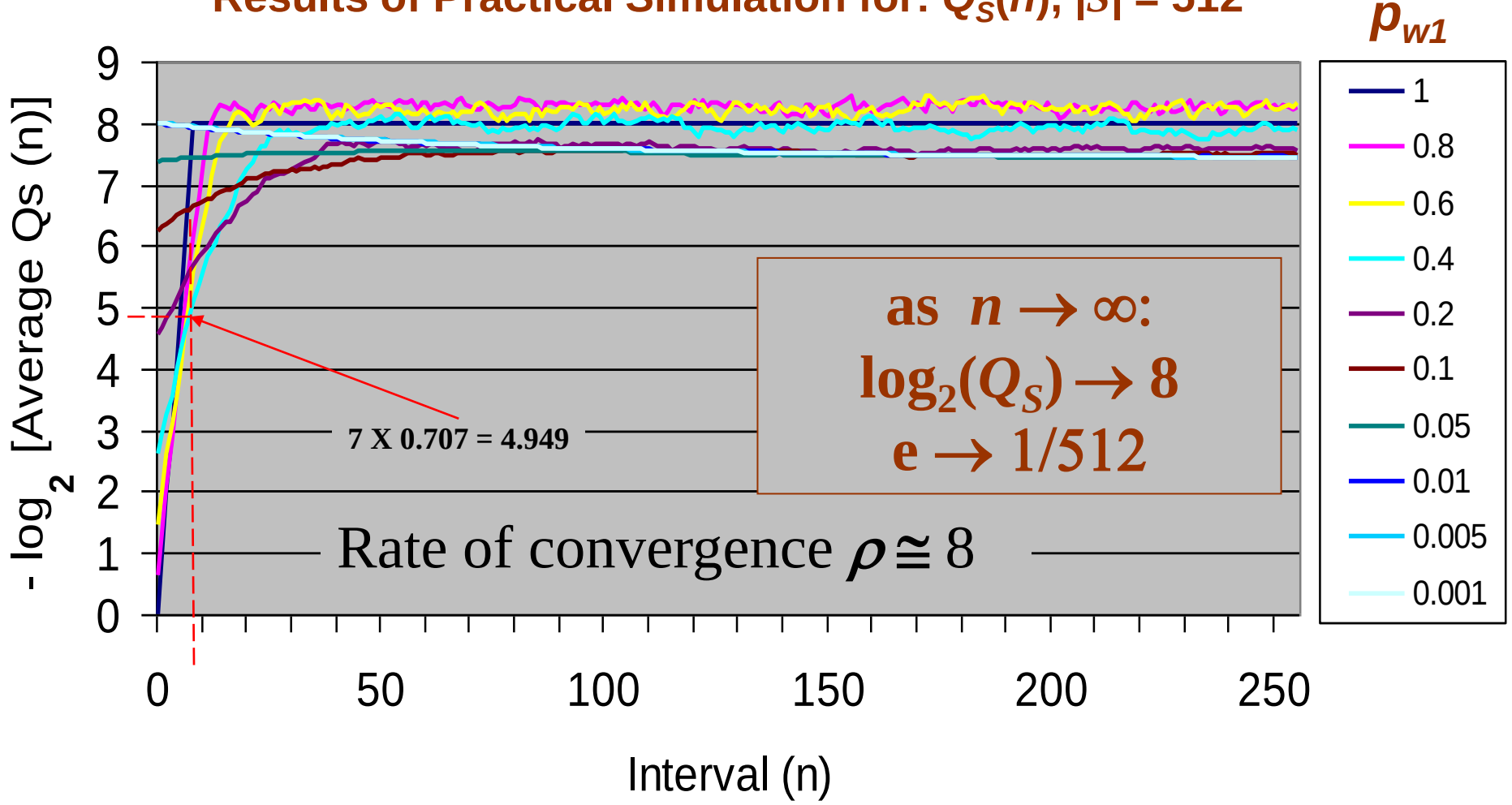
Results of Practical Simulation for: $H_S(n)$, $|S| = 256$



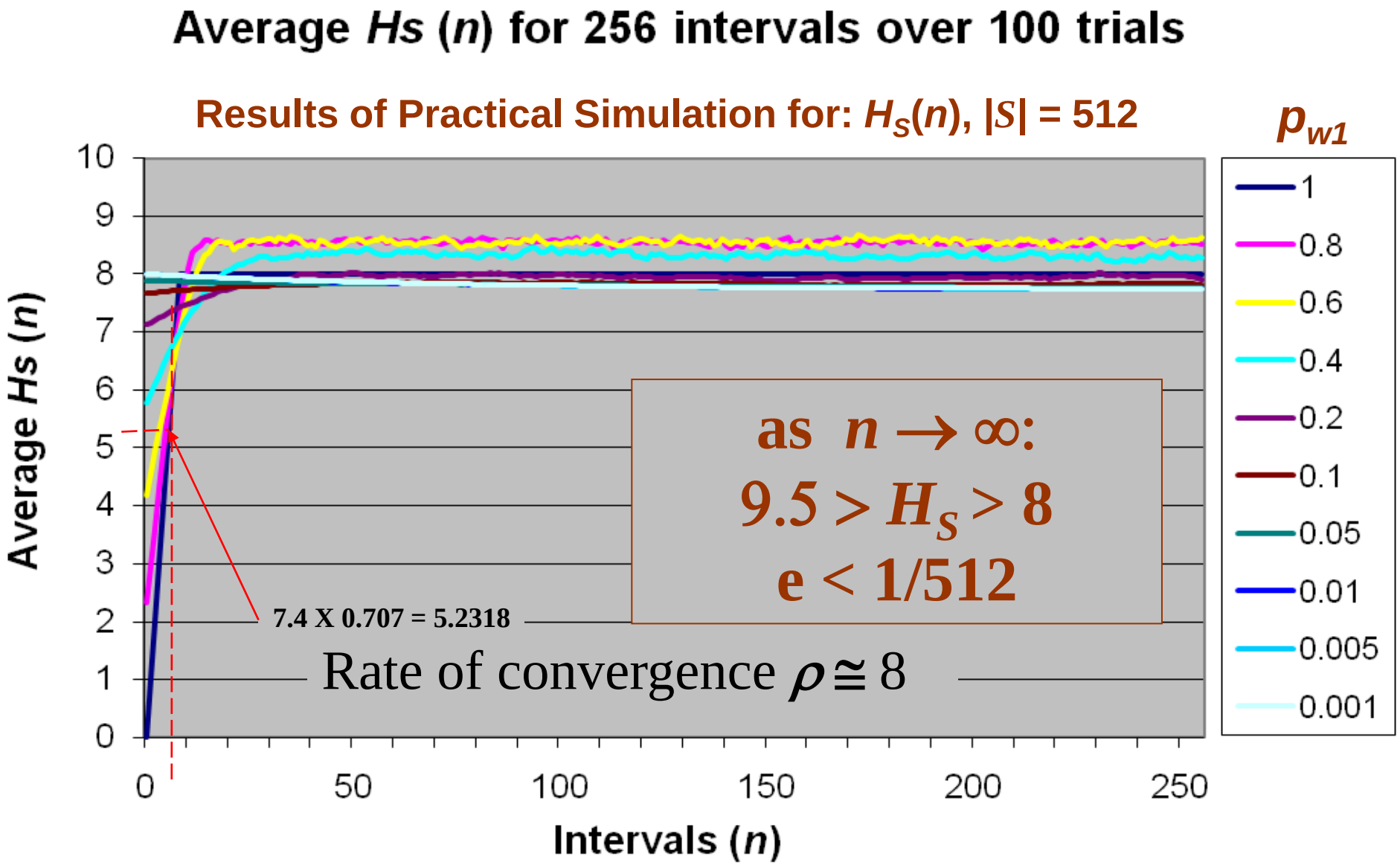
S&M algorithm: the finite case
The S-Norms

- $\log_2$[Average Qs (n)] for 256 intervals over 100 trials

Results of Practical Simulation for: $Q_S(n)$, $|S| = 512$



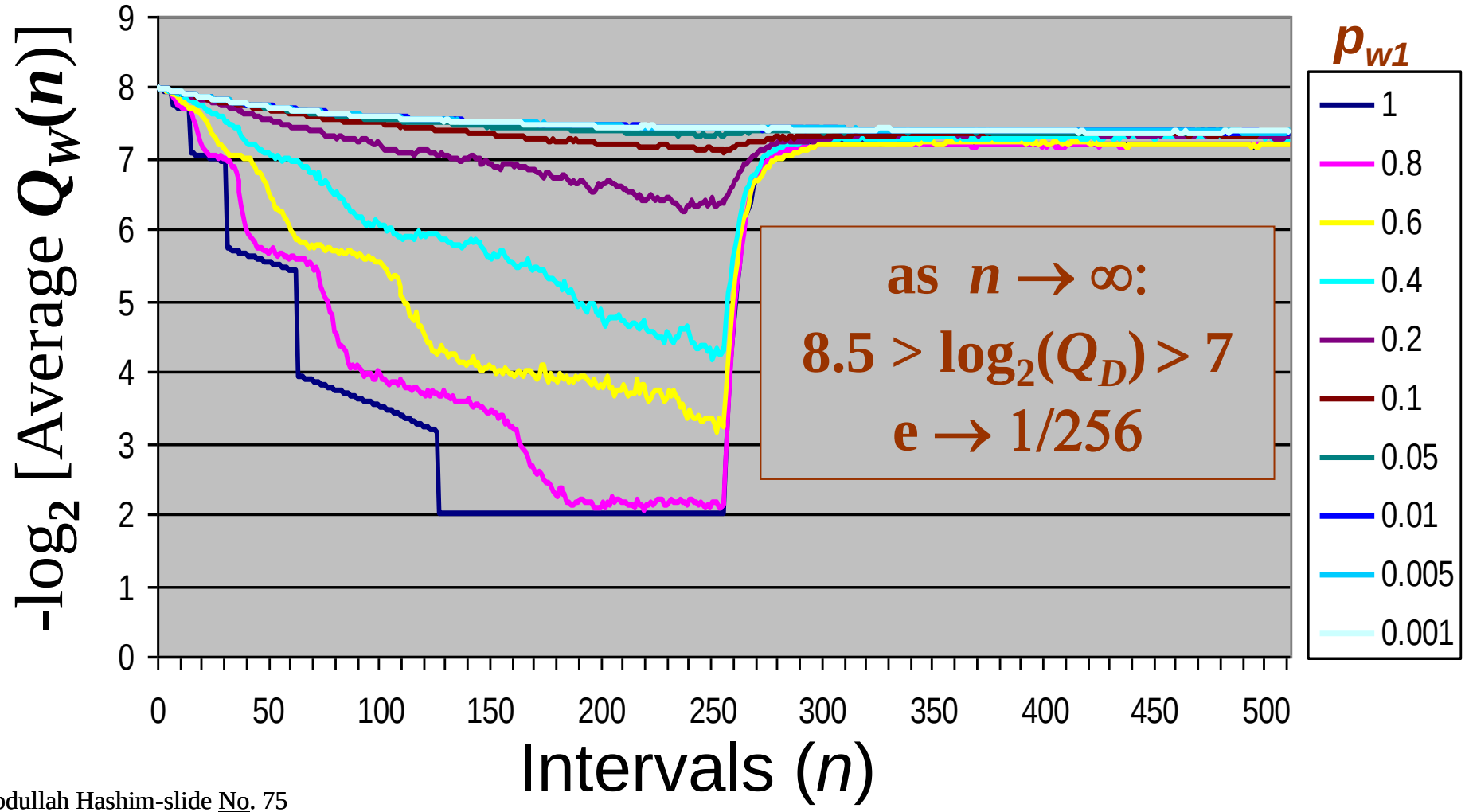
S&M algorithm: the finite case
The S-Norms



S&M algorithm: the finite case
The W-Norms

n = 0; Source probabilities are set in the given ten values.
At ***n*** = 256 the source become completely random.

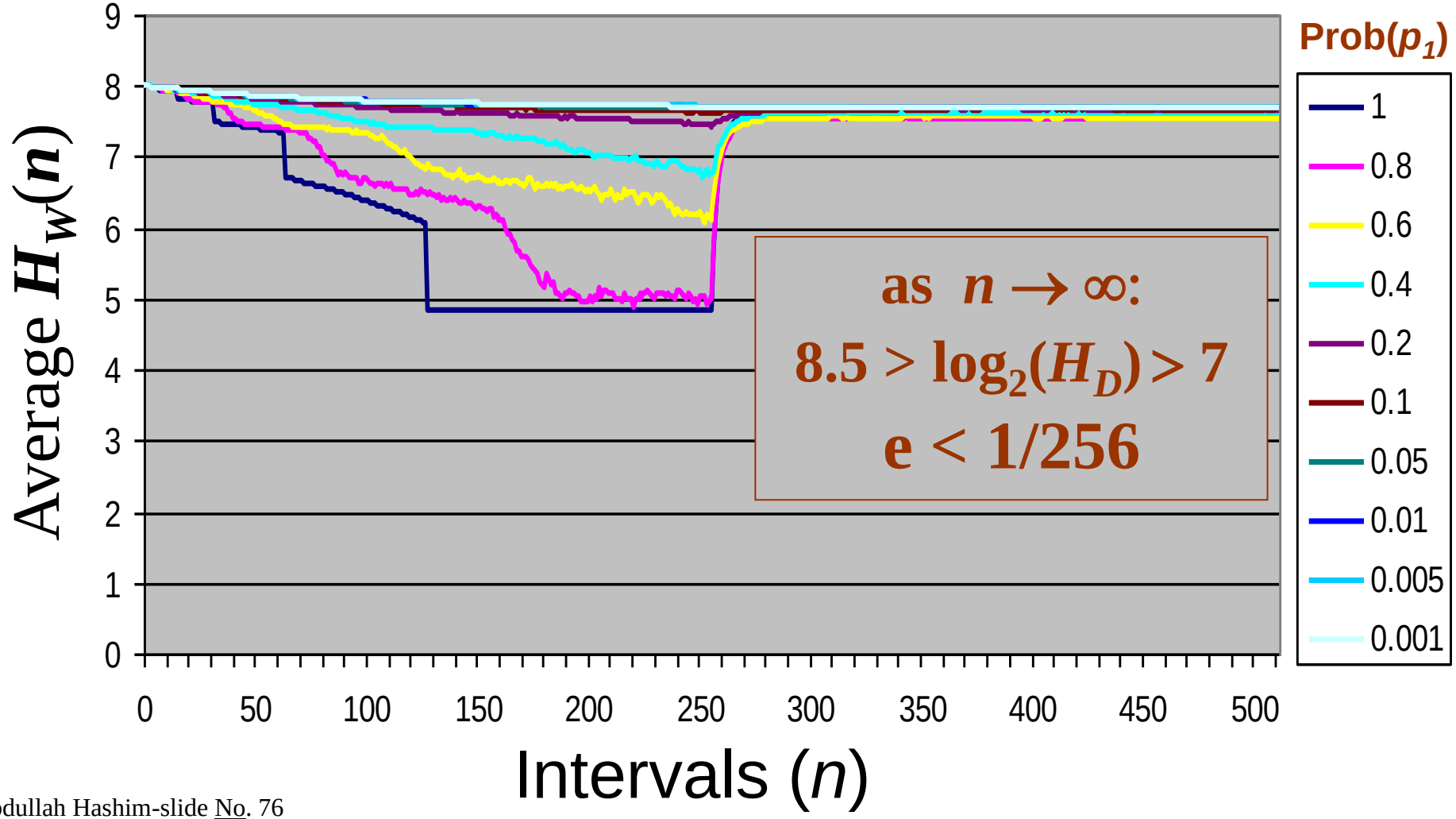
Results of Practical Simulation for: $Q_W(n)$, $|S| = 256$, over 100 trials



S&M algorithm: the finite case
The W-Norms

n = 0; Source probabilities are set in the given ten values.
At ***n*** = 256 the source become completely random.

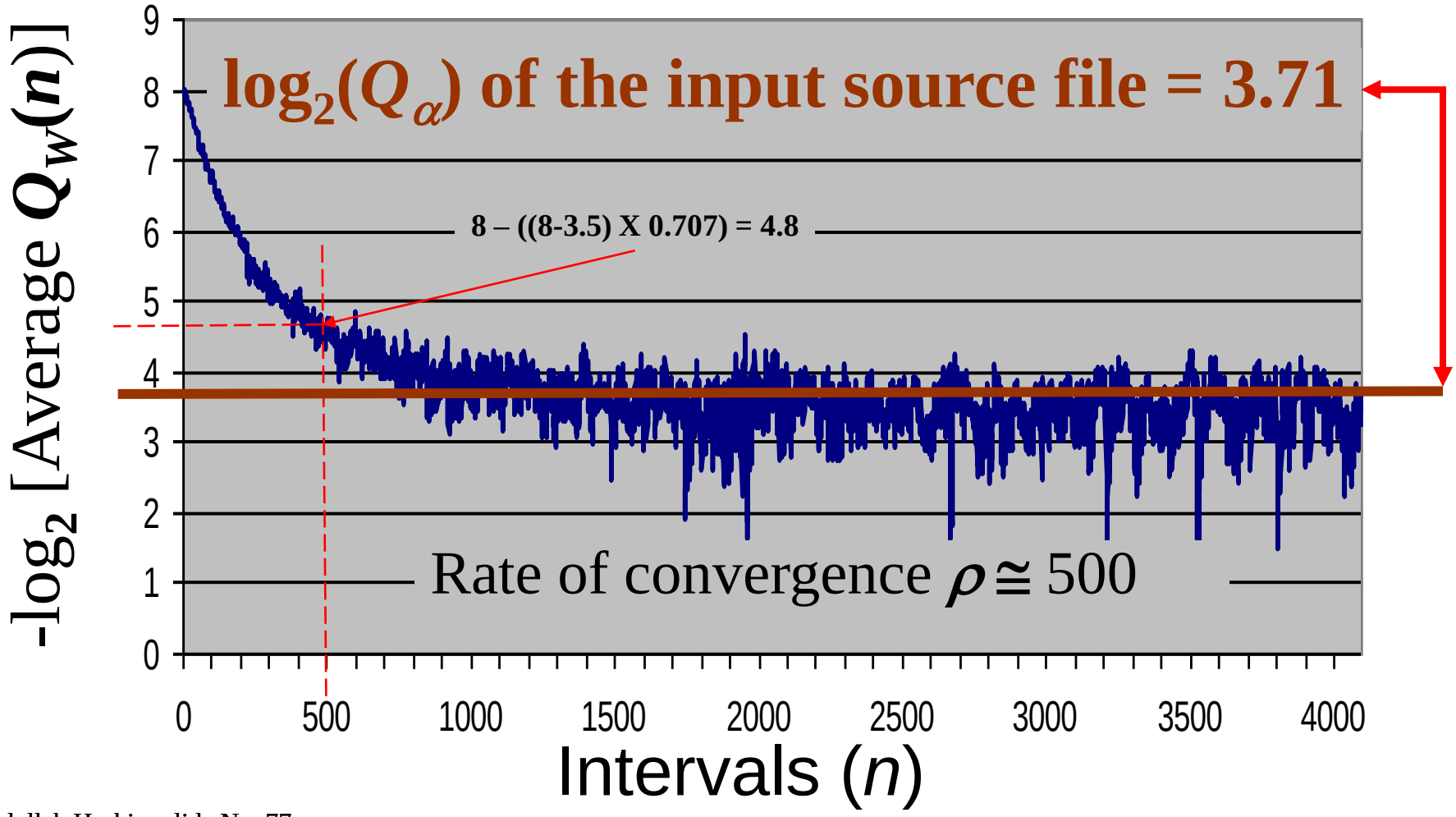
Results of Practical Simulation for: $H_W(n)$, $|S| = 256$, over 100 trials



S&M algorithm: the finite case
The W-Norms

Practical Input Source
4096 character string from Obama Inaugural Speech

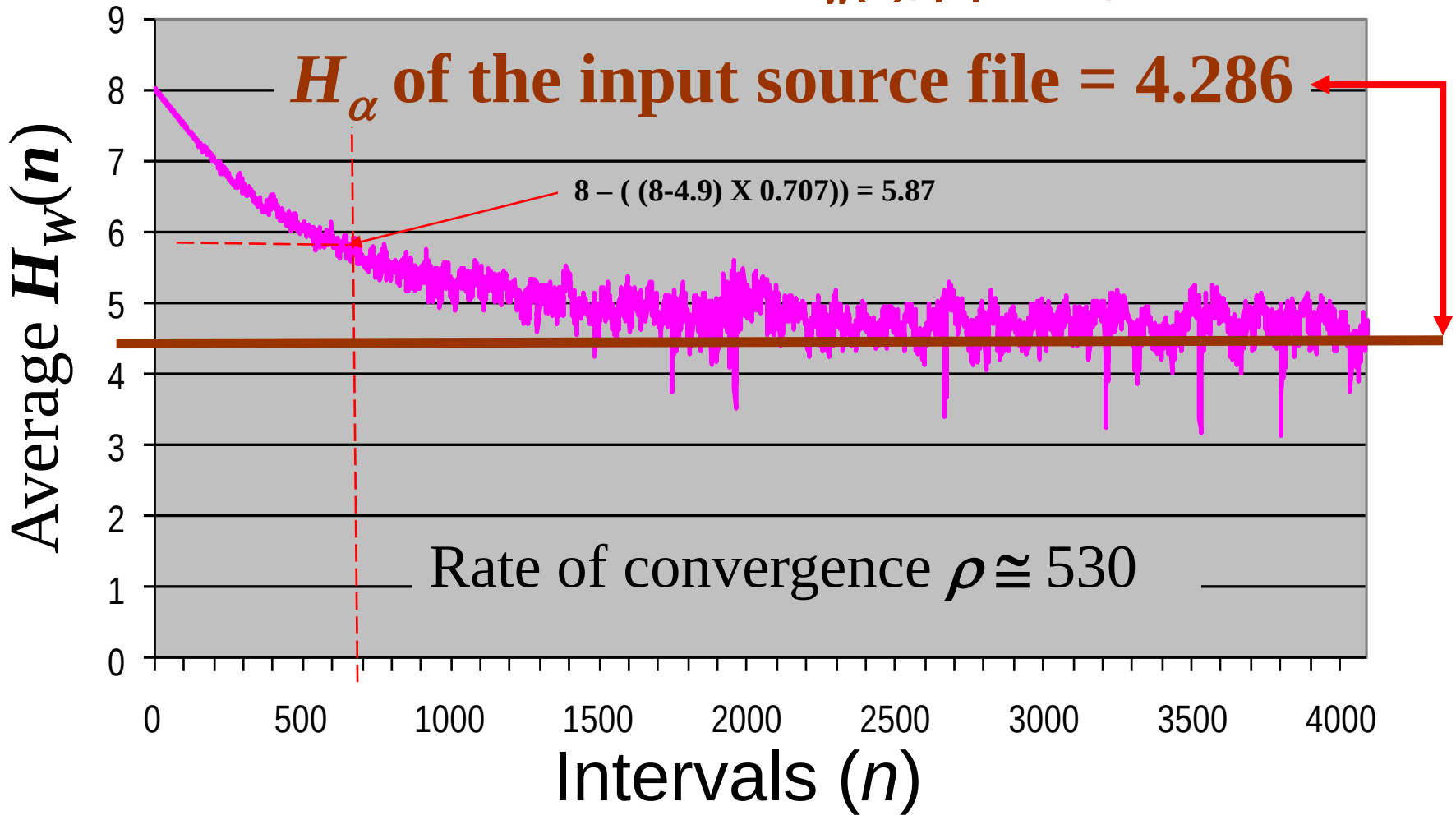
Results of Practical Simulation for: $Q_W(n)$, $|S| = 256$, over 100 trials



S&M algorithm: the finite case
The W-Norms

Practical Input Source
4096 character string from Obama Inaugural Speech

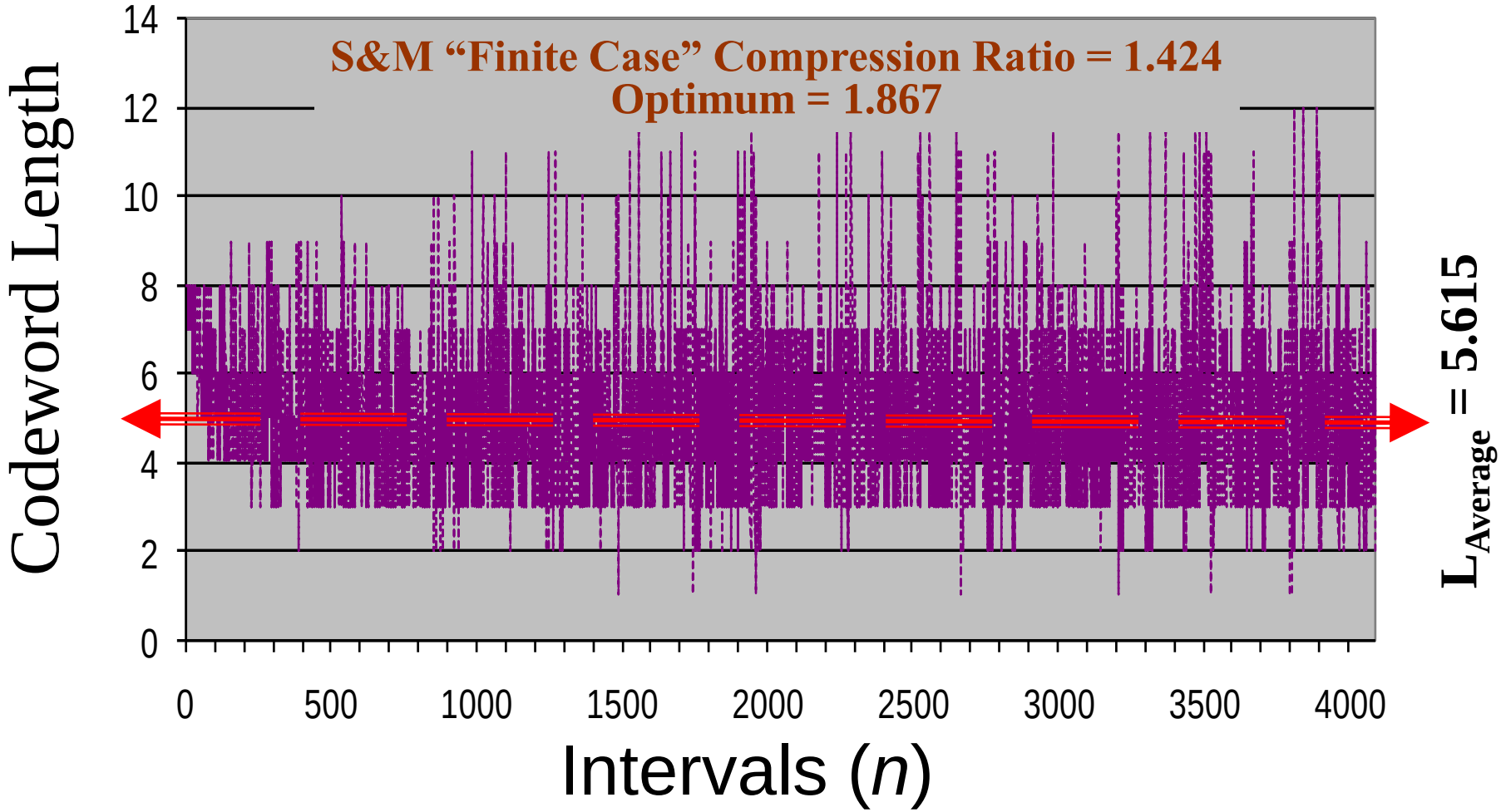
Results of Practical Simulation for: $H_W(n)$, $|S| = 256$, over 100 trials



S&M algorithm: the finite case
The D-Norms

Practical Input Source
4096 character string from Obama Inaugural Speech

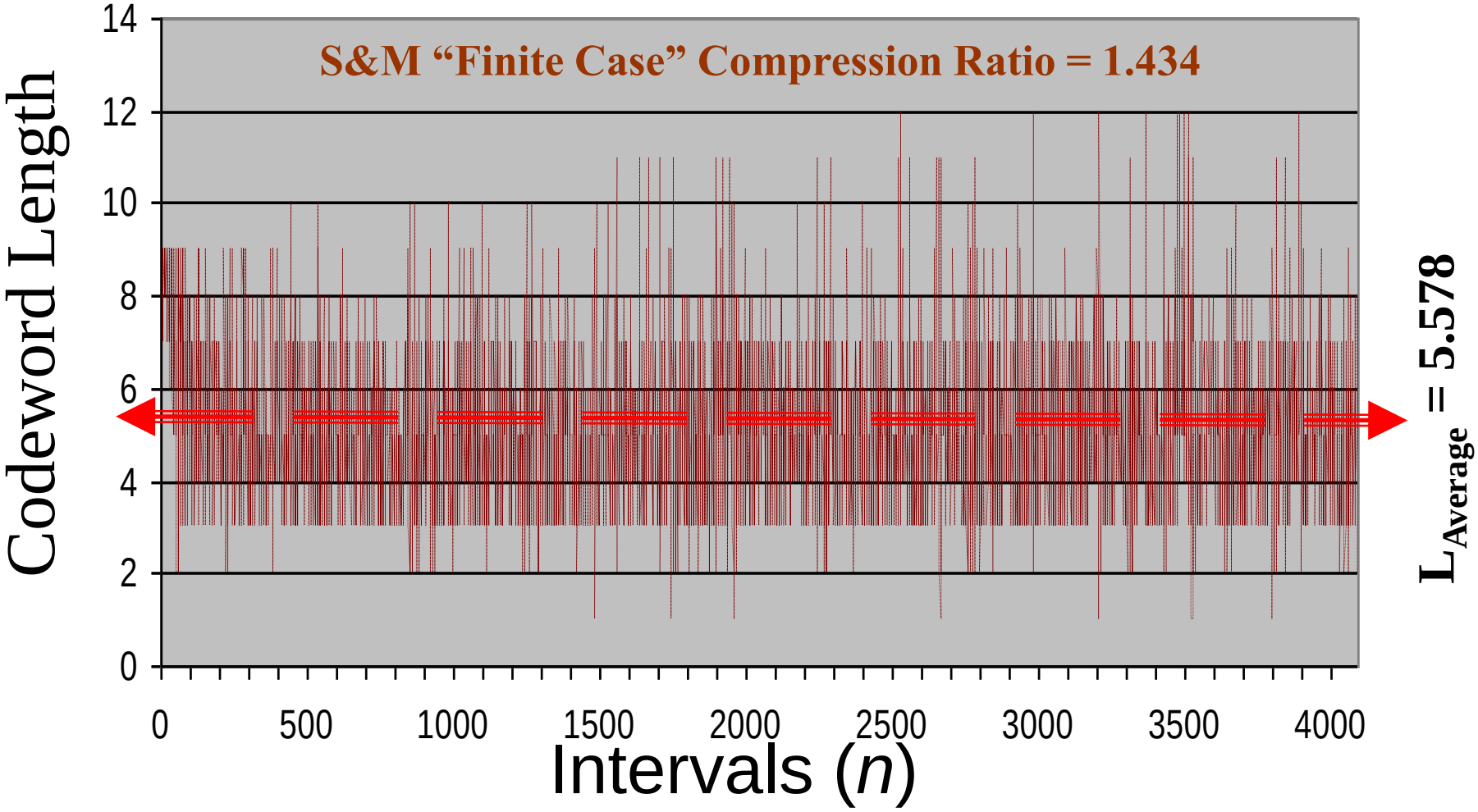
Results of Practical Simulation for: $|S| = 256$, over 100 trials



S&M algorithm: the finite case
The D-Norms

Practical Input Source
4096 character string from Obama Inaugural Speech

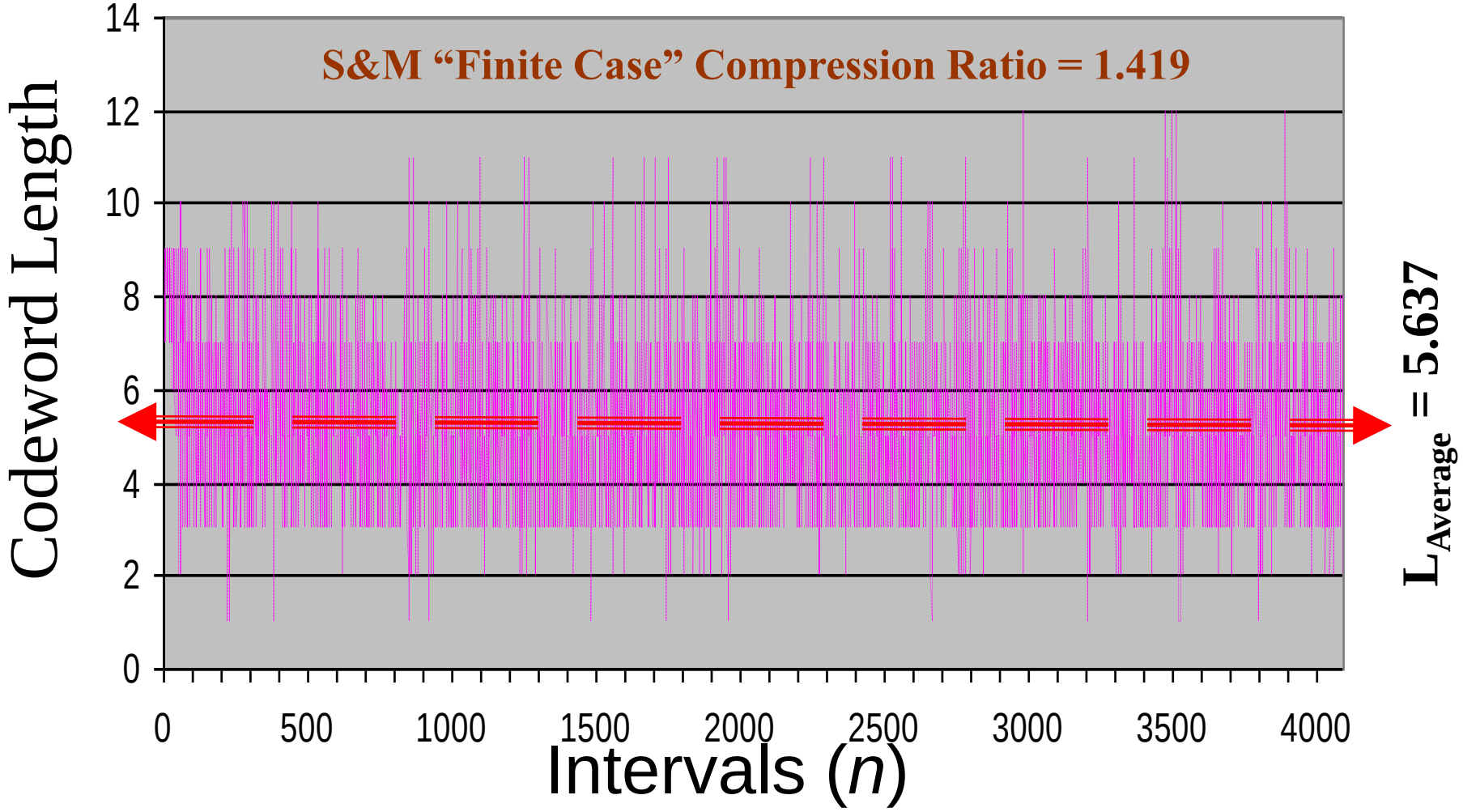
Results of Practical Simulation for: $|S| = 512$, over 100 trials



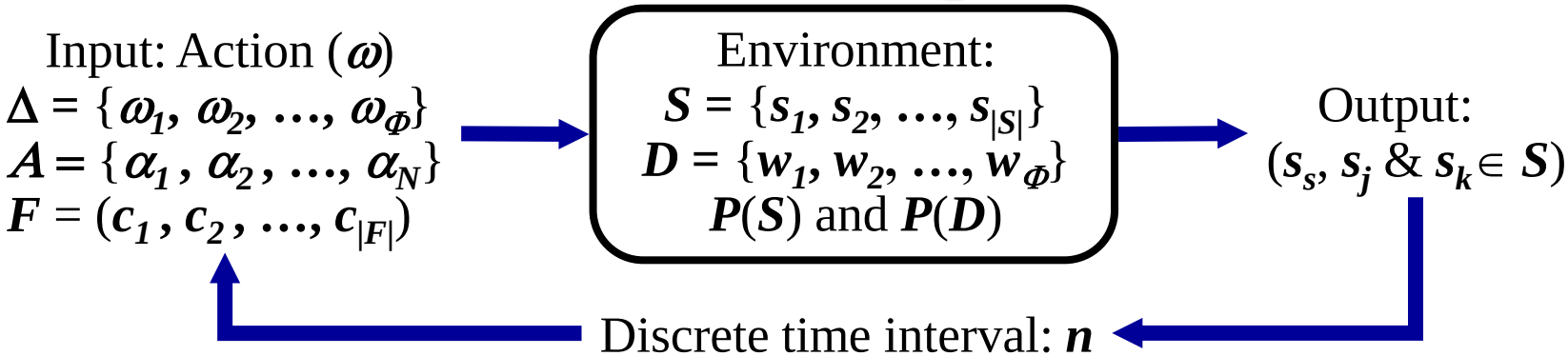
S&M algorithm: the finite case
The D-Norms

Practical Input Source
4096 character string from Obama Inaugural Speech

Results of Practical Simulation for: $|S| = 1024$, over 100 trials



S&M algorithm: the dictionary case
-The Concept-



Assumptions: In the finite case we assume that: $|D| = |\Delta| = N$. In the dictionary case the size of the dictionary is allowed to grow. Asymptotically,

$$\lim_{\Phi \to \infty} H(w) \to H(\omega) \to 0.$$ *Lempel and Ziv, 1981.*

Practically, the size of the dictionary is limited to Φ, at this stage a word with lowest probability will be pruned to give a space for the new word to be added to the dictionary.

Dictionary Tree: The dictionary is best described and implemented through a tree data structure. Tree data structure offers a good foundation for understanding the behaviour and the practical implementation of the dictionary.

S&M algorithm: The Tree Structure

Dictionary tree (***DT***): A ***tree*** (***DT***) is an abstract data structure used here to represent the words in ***D***. It consists of a ***root*** node with ***links*** (branches) to its ***children*** nodes. These nodes in turn have links to their children. A ***leaf*** is a node with no children. The number of links in a path from the root to a node v_i is called the ***depth*** (d_i) of the node v_i. The maximum node depth of tree is called the ***tree depth*** (d_t). The maximum number of nodes in a dictionary tree is (Φ=1). Each node of ***DT*** contains a single character from the alphabet, except that the root contains no data (i.e. contains the empty word Λ).

Data-node	A node containing a single character of the alphabet.
No-data node	A node containing the null character (Λ).

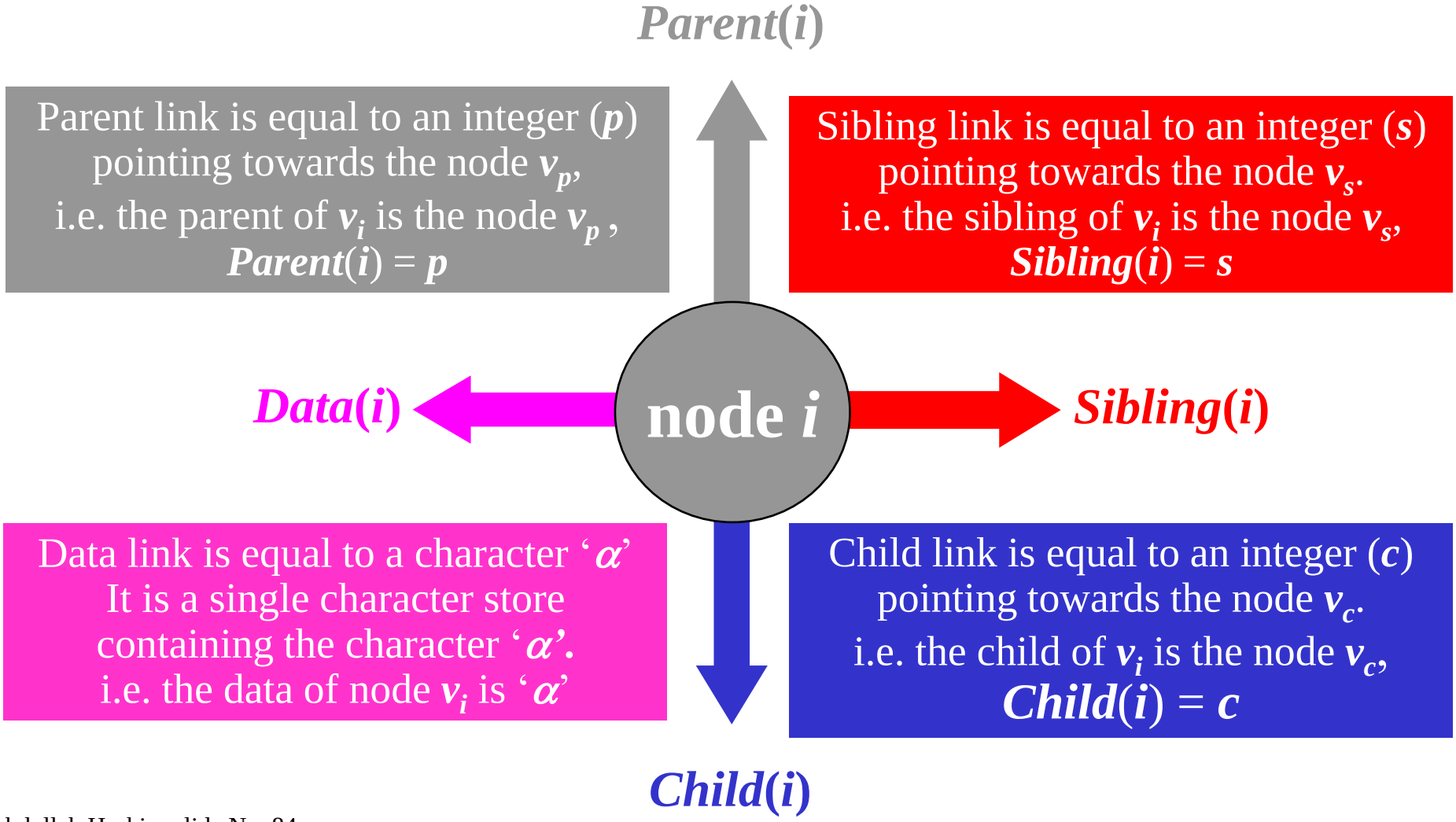
The root of the ***DT*** has α children. Each node v_i, except the root node, corresponds to a word (w_i) in ***D***.

The corresponding word of a given node may be constructed by reading the data Λ of the root node v_0 and concatenated to the single character data from the ancestor nodes along the path in sequence, terminating with the character of the given node.

S&M algorithm: The Tree Structure

DT links:

If a link has a value equal to -1, the link is a null link, leading to no node; the node does not exist.



S&M algorithm: The Tree Structure

Let $D = \{0, 1, 2, \ldots, 253, 254, 255, ab, ae, abc, fc, fr, fce\}$

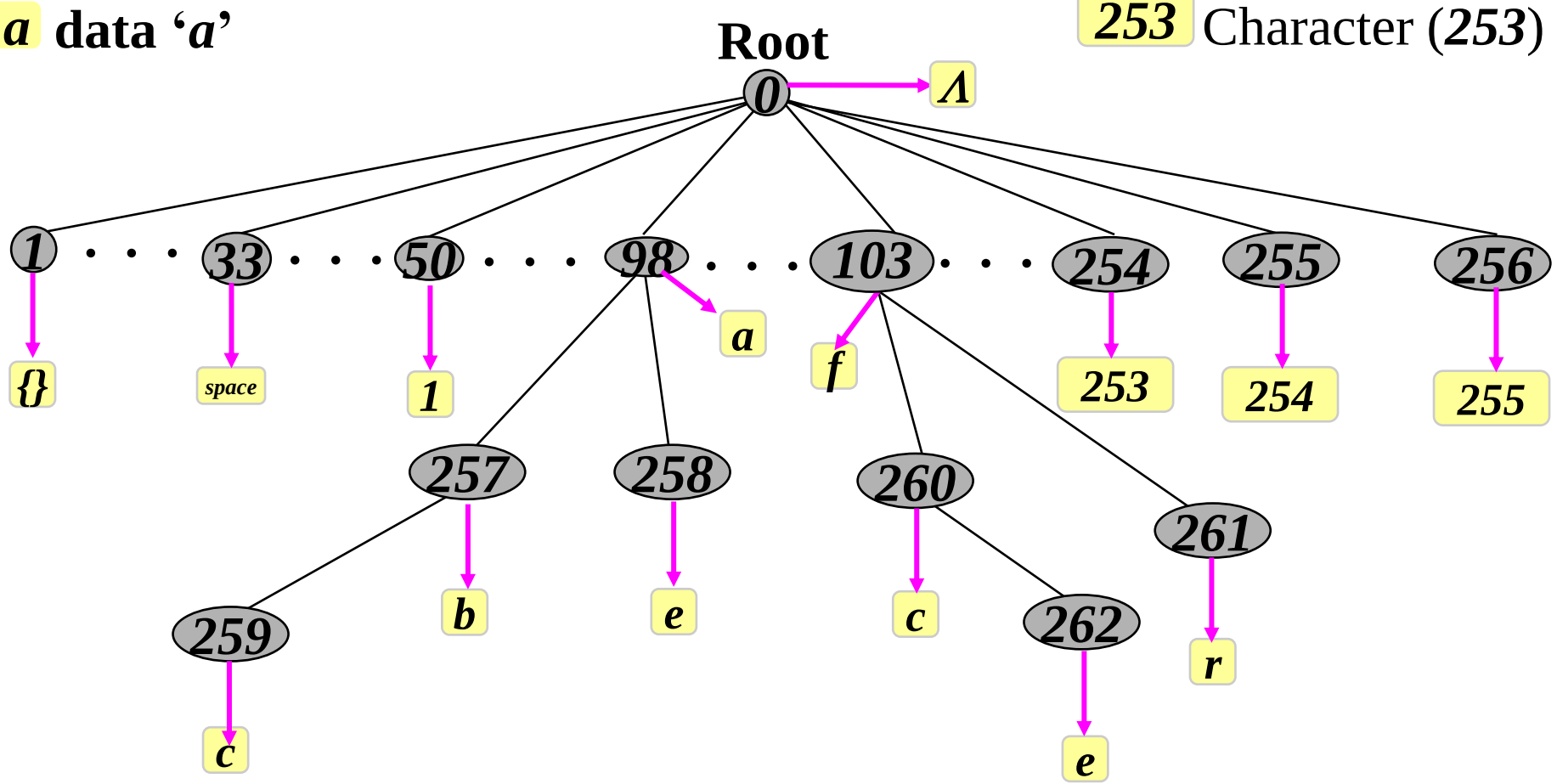
D may be represented graphically as follows:

Key: | **data link** Λ **null data** (0) **node** *0*

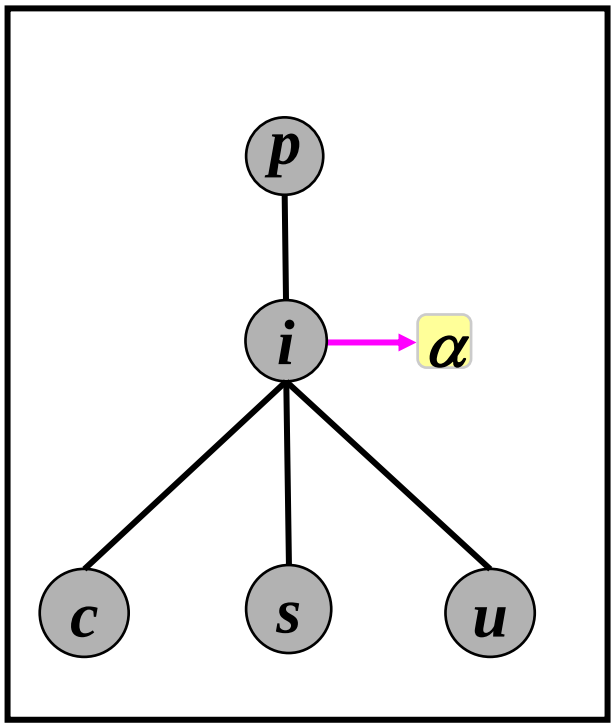
a **data 'a'**

i ASCII Character (**Val *i***)

253 ASCII Character (*253*)



S&M algorithm: The Tree Structure



node *P*

Parent (*i*) link = *p*

KEY

each node i has 4 links:

1. *Parent (i) link*
2. *Data(i) link*
3. *Child (i) link*
4. *Sibling(i) link*

node *i*

Data (*i*) Link =‘*α*’

ASCII character or -1 if no data in v_i

Child(i) link = c

node *c*

Sibling (c) link = s

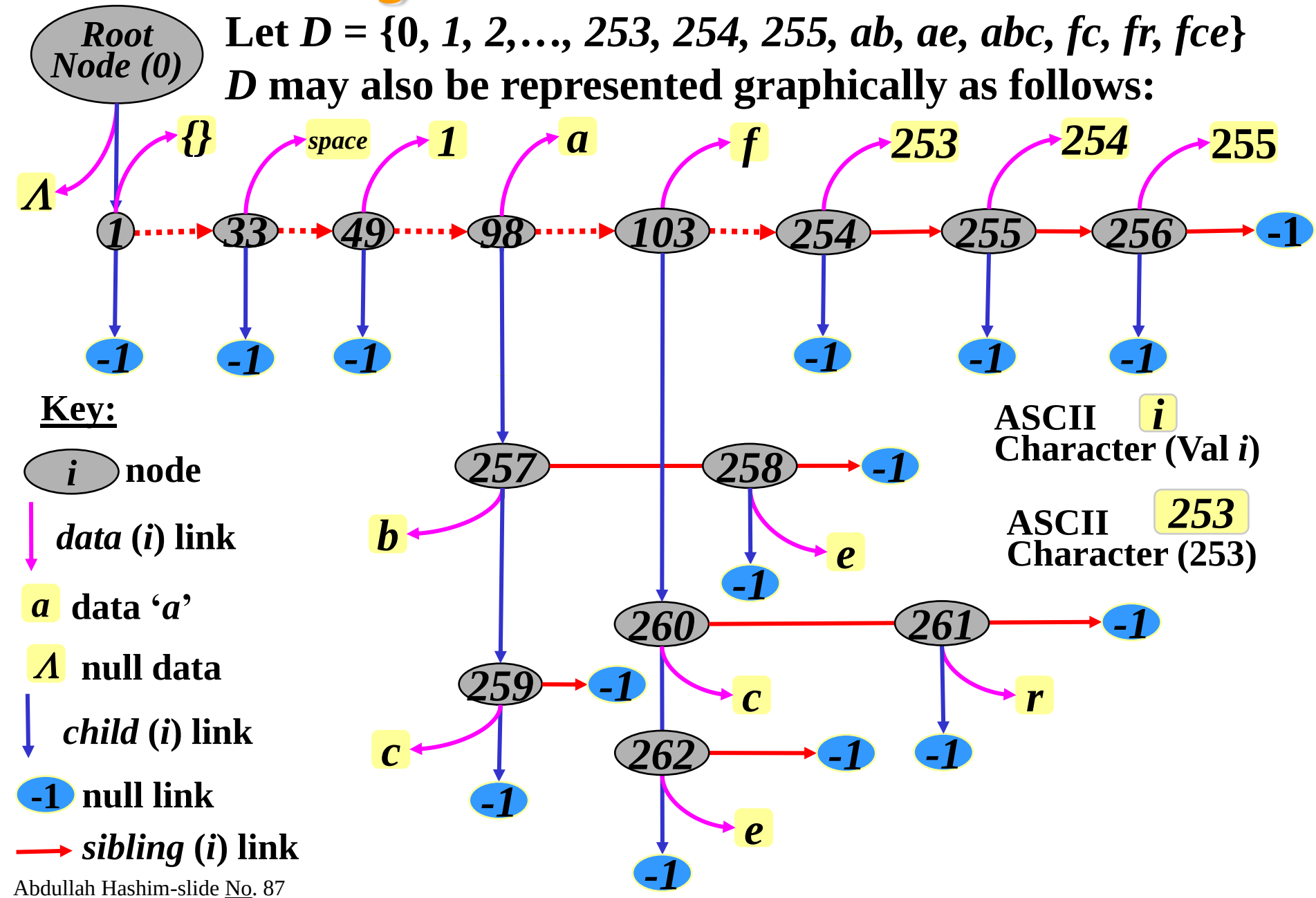
node *s*

Sibling (c) link = u

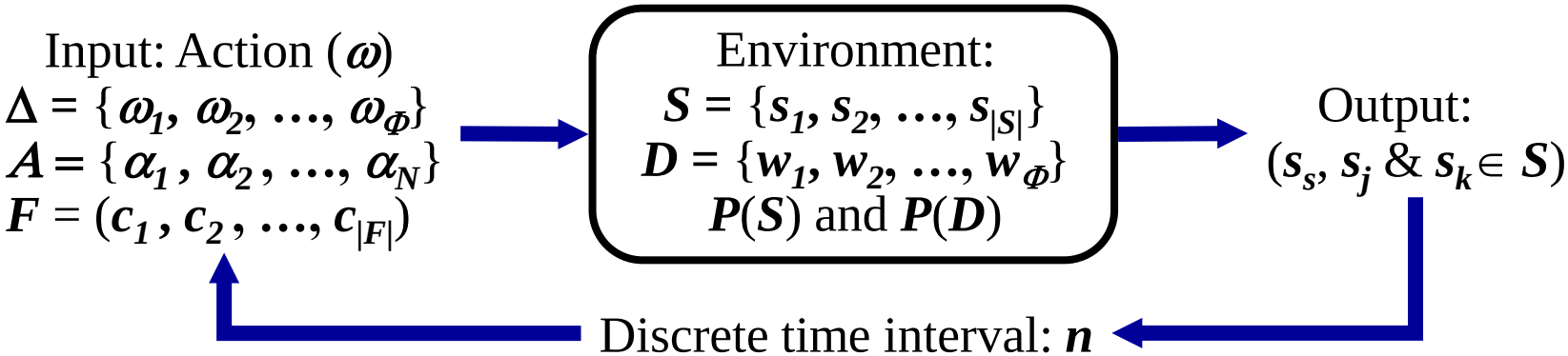
node *u*

-1

S&M algorithm: The Tree Structure



S&M algorithm: The Tree Structure



ED*: *Extended Dictionary*: *ED is a dictionary containing:

1. All single ASCII characters, $A = \{\alpha_1, \alpha_2, \ldots, \alpha_N\}$.

2. All multiple character words, $W = \{w_1, w_2, \ldots w_M\}$.

3. All command "control" words, $C = \{\delta_1, \delta_2, \ldots, \delta_C\}$, usually $|C|$ is in the range of two to three characters.

4. All the null words, $\{\} = \{\Lambda_1, \Lambda_2, \ldots, \Lambda_i, \ldots, \Lambda_B\}$.

Therefore: total words in ***ED*** is $|ED|_{max} = N + M + C + B$

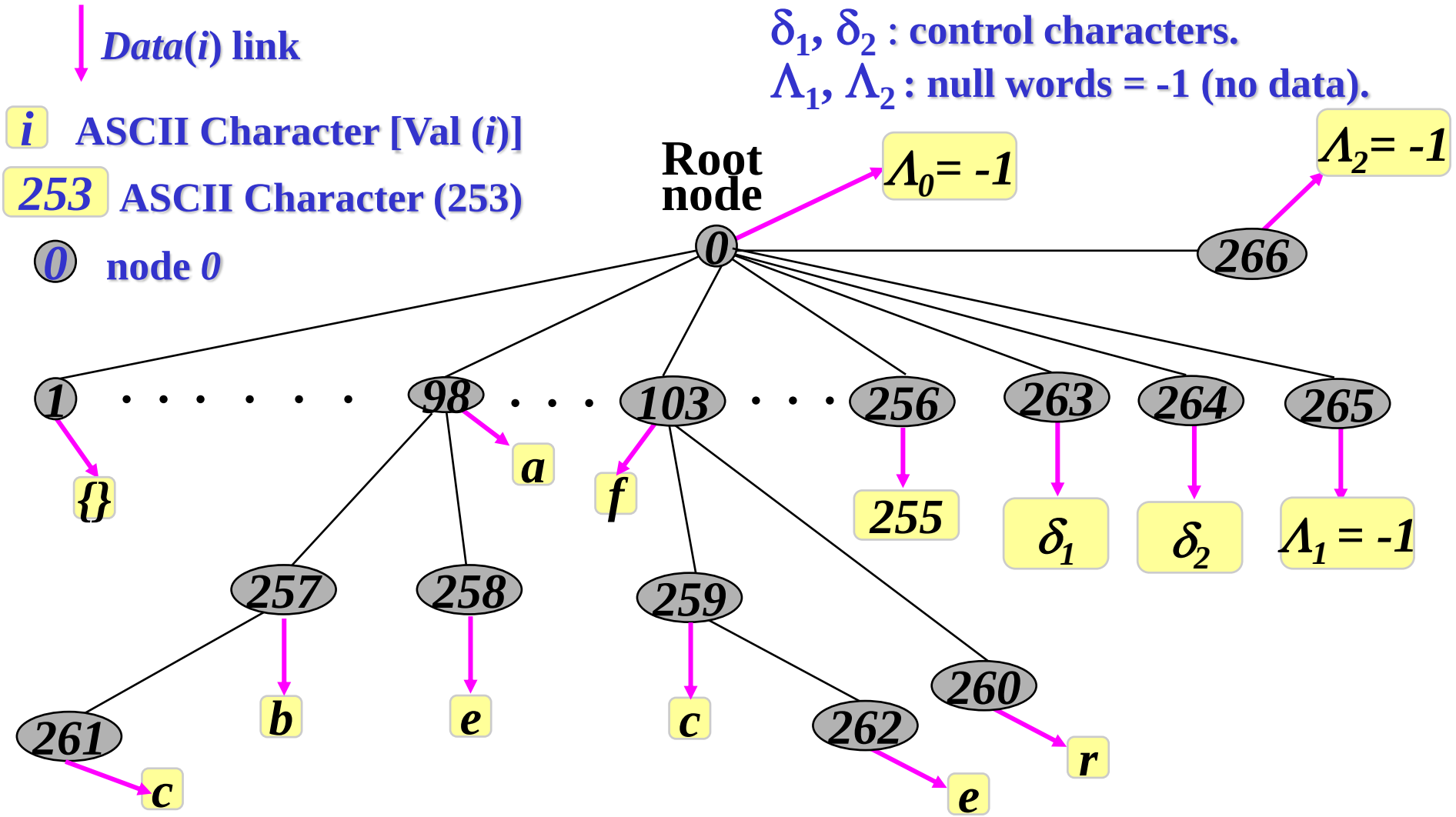
S&M algorithm: The Tree Structure

For example let the extended dictionary be given by the set *ED*:

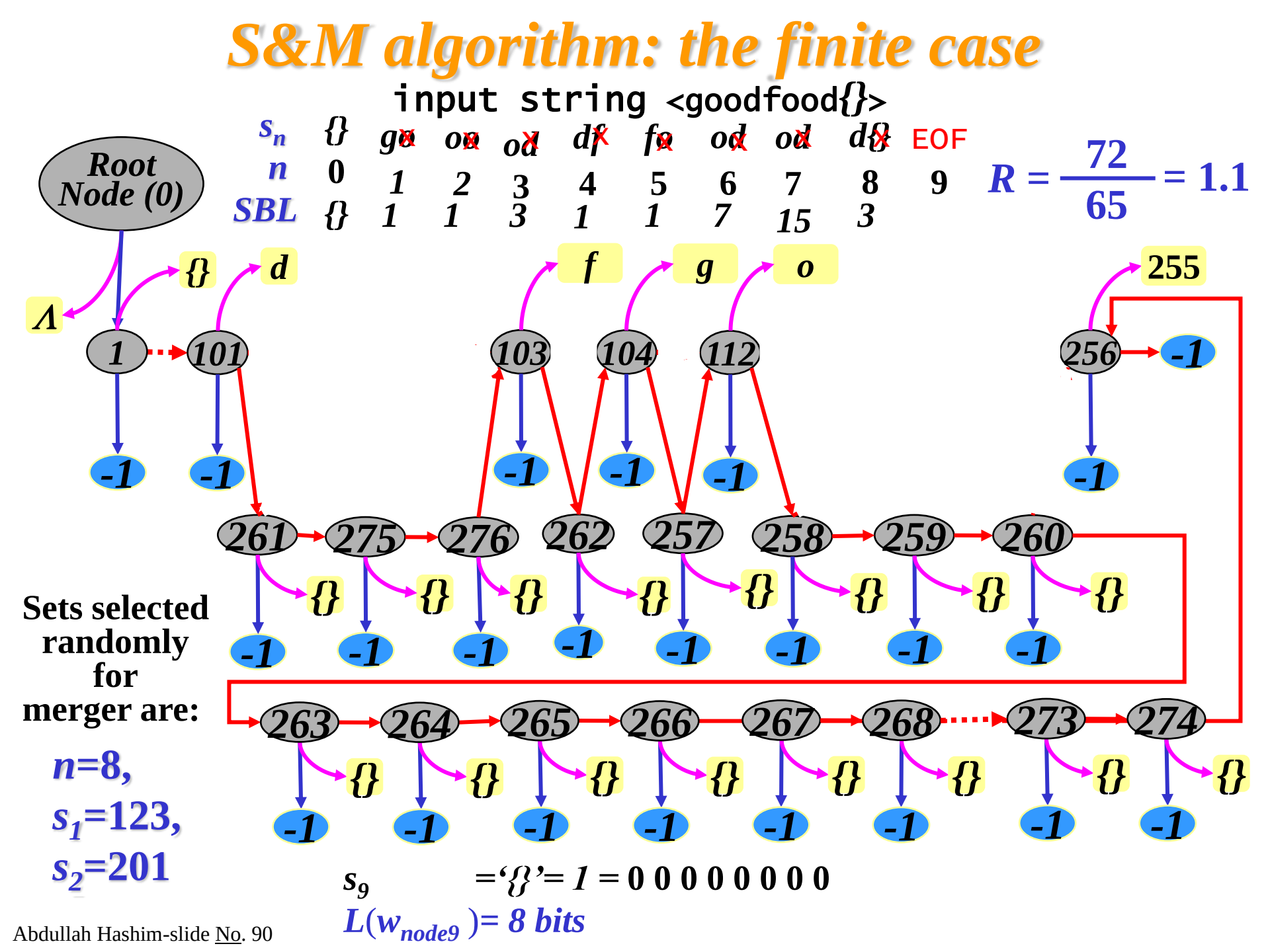
{0, 1, 2, …, 253, 254, 255, δ_1, δ_2, ab, ae, abc, fc, fr, fce, Λ_1, Λ_2}

δ_1, δ_2 : control characters.

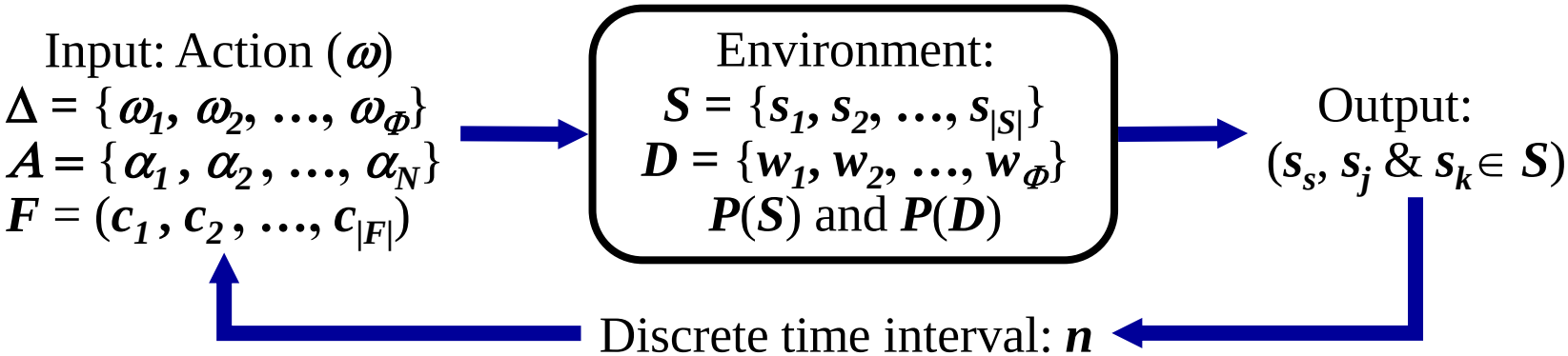
Λ_1, Λ_2 : null words = -1 (no data).



S&M algorithm: the finite case



S&M algorithm: the dictionary case
-The Strategy-



The new word w_{new}: Let word ω_n , be an input string from file F, at interval n, match ω_n to the longest word w_n in the dictionary D:

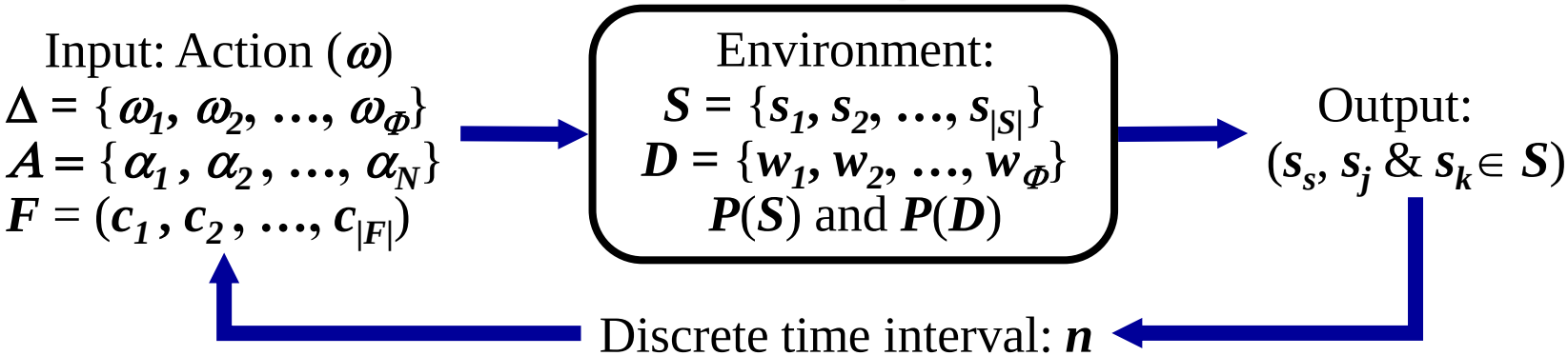
$$\omega_n = w_n = <\alpha_1\alpha_2\ldots\alpha_i\ \ldots\alpha_j>.$$

If α_{j+1} is the unmatched character resulting from the matching process, then the new word (w_{new}) is defend as:

$$w_{new} = <a_1a_2\ldots a_j a_{j+1}>.$$

The strategy of the dictionary case is identical to that, of the finite case, however, a singleton set is splitted by adding a new singleton set containing the new word (w_{new}). If (w_{new}) does not exist, then the process will be identical to that, of the finite case.

S&M algorithm: the finite case
-The Probability Vectors-

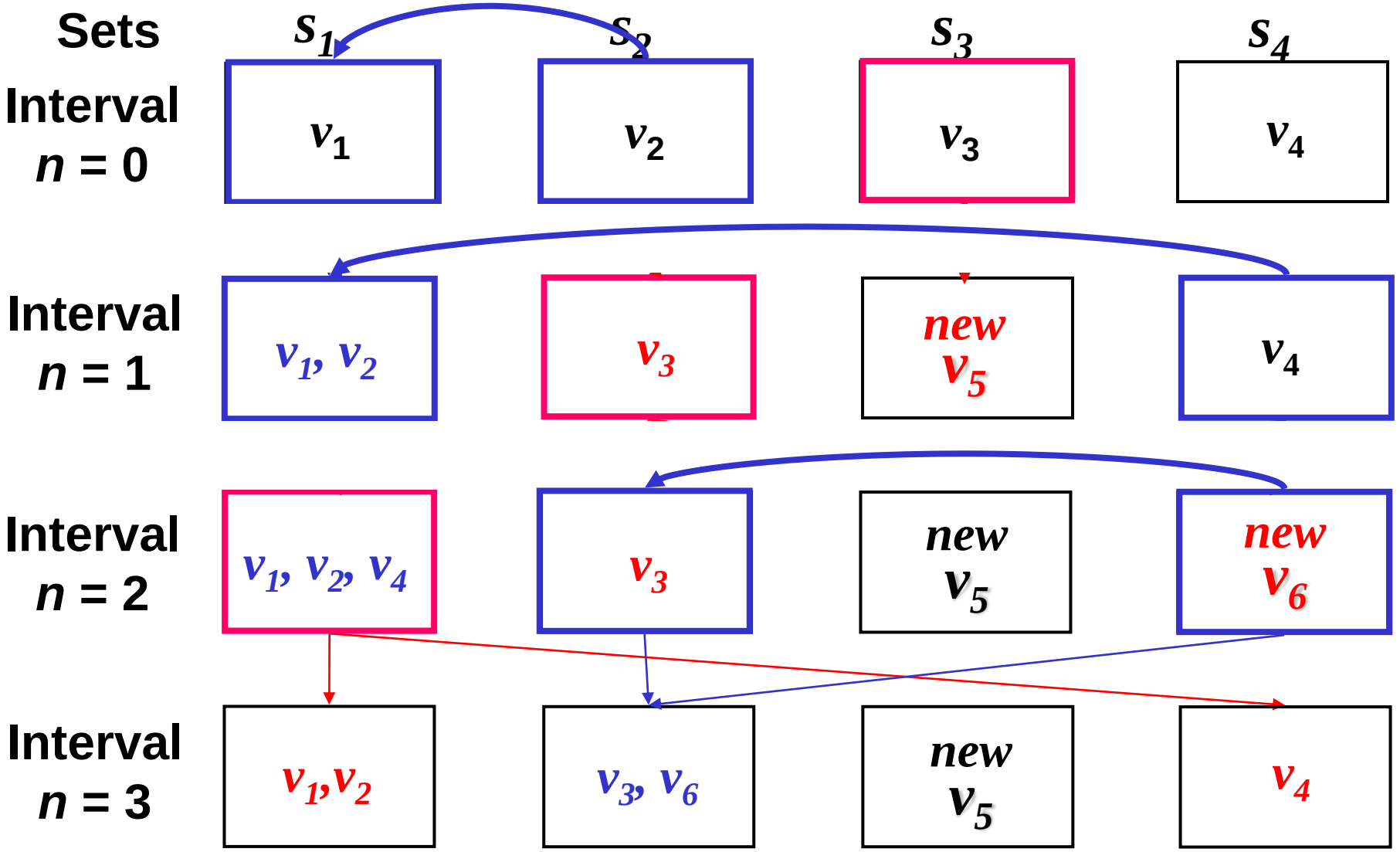


Child Word Probability: A singleton parent node probability will be halved by adding a new child (word). Similarly, ***FBL*** of a singleton with satellites will be halved and the child word ***FBL*** will equal to that of its parent after the process of splitting.

Word Real Probabilities: Word (node) real probability is equal to the sum of all the ***real*** probabilities of the given node and its decendent; Parent word and its decendent are in Δ.

Word State Probabilities: Word (node) state probability is equal to the sum of all the ***outset*** probabilities of the given node and its decendent; Parent word and its decendent are in ***D***.

S&M algorithm: the dictionary case



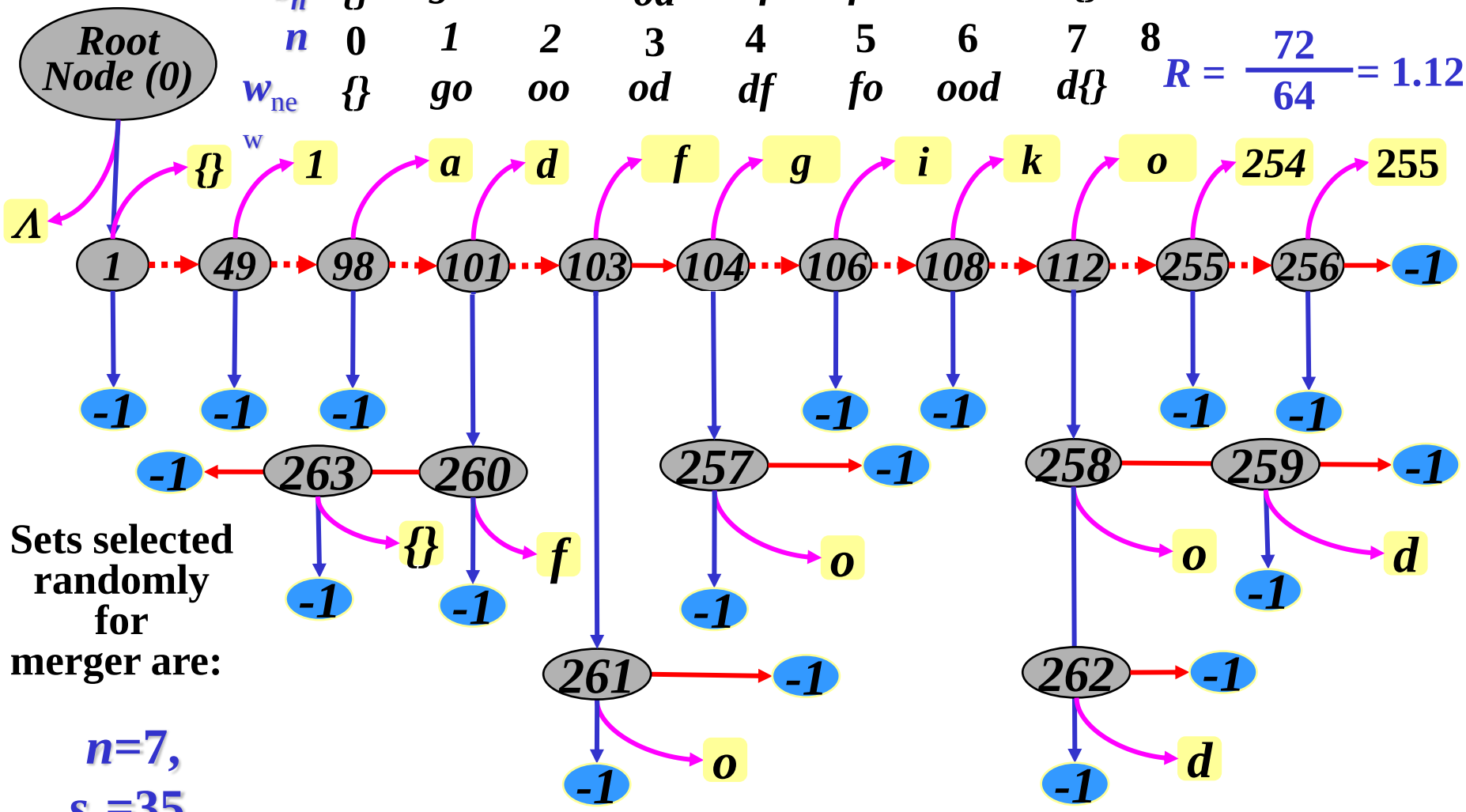
For large values of *n*, set probabilities will converge to $2/|S|$

S&M algorithm: the dictionary case.

input string <goodfood{}>

s_n	{}	go	oo	od	df	fo	ood	d{}	EOF
n	0	1	2	3	4	5	6	7	8
w_{ne}	{}	go	oo	od	df	fo	ood	d{}	

$R = \frac{72}{64} = 1.125$



Sets selected randomly for merger are:

$n=7$,
$s_1=35$,
$s_2=9$

s_8 ='{}'=1 = 0 0 0 0 0 0 0 0

$L(w_{word8})$ *is 8 bits*

S&M algorithm: the dictionary case, Δ=0.

results of practical simulation

Prob(Set1)

$$2/N \leq p(s_i) \leq 1/N^{1/2}$$

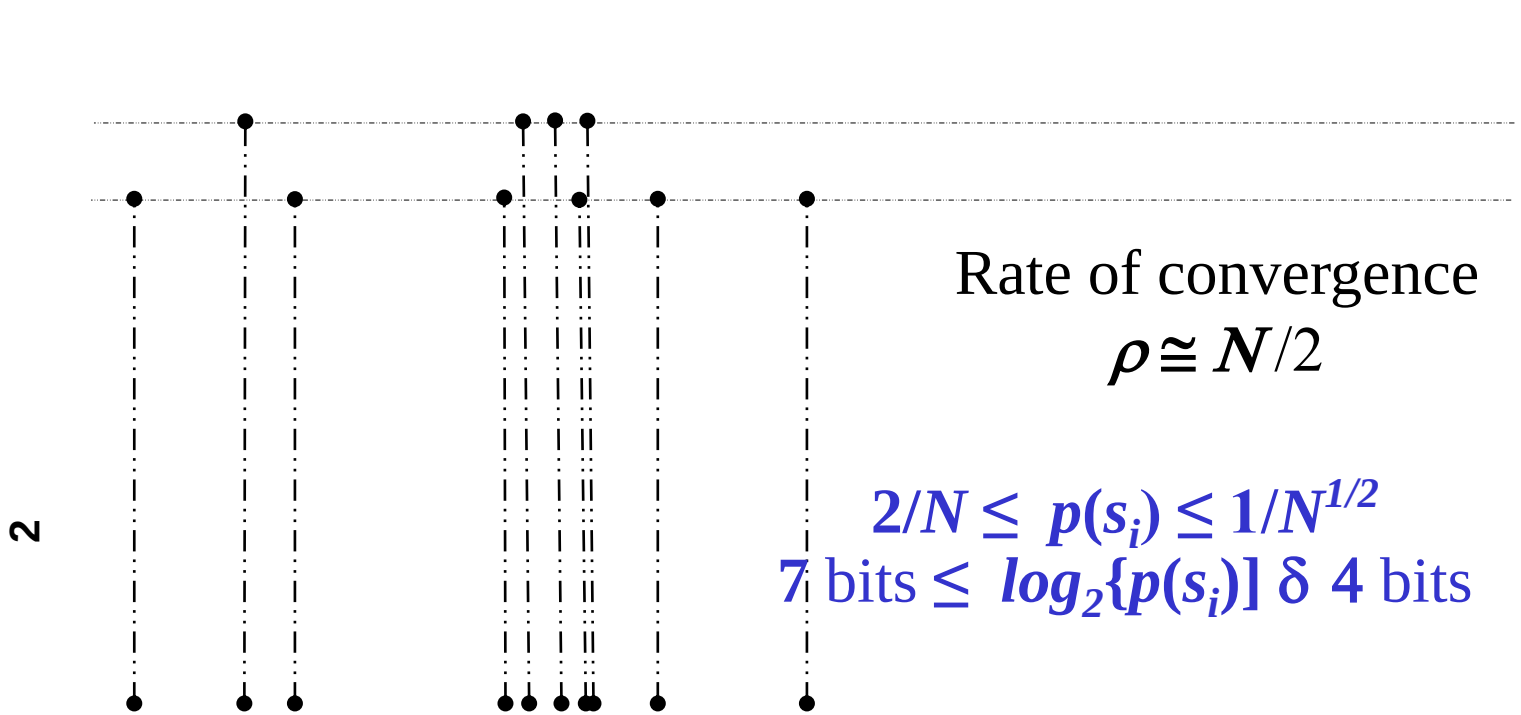
$$0.0078 \leq p(s_i) \leq 0.067$$

Rate of convergence

$$\rho \cong N/2$$

S&M algorithm: the dictionary case, Δ=0.

2Results of Practical Simulation



Rate of convergence

$\rho \cong N/2$

$2/N \leq p(s_i) \leq 1/N^{1/2}$

7 bits $\leq log_2\{p(s_i)] \leq 4$ bits

3 db points

S&M algorithm: The Tree Structure

OD *Ordered Dictionary*. The ordered dictionary is a list of words. The words are ordered according to some parameter or function; the leftmost word (w_1) has the highest rank while the rightmost word (w_M) has the lowest rank.
$OD = < w_1, w_2, \ldots, w_j, \ldots, w_{(M-1)}, w_M >$

ODT *Ordered Dictionary Tree*. A graphical presentation of ***OD***. Let $ODT = < v_1, v_2, \ldots, v_j, \ldots, v_{(M-1)}, v_M >$, where v_j is a node in ***ODT*** corresponding to the word w_j in the ***OD***.

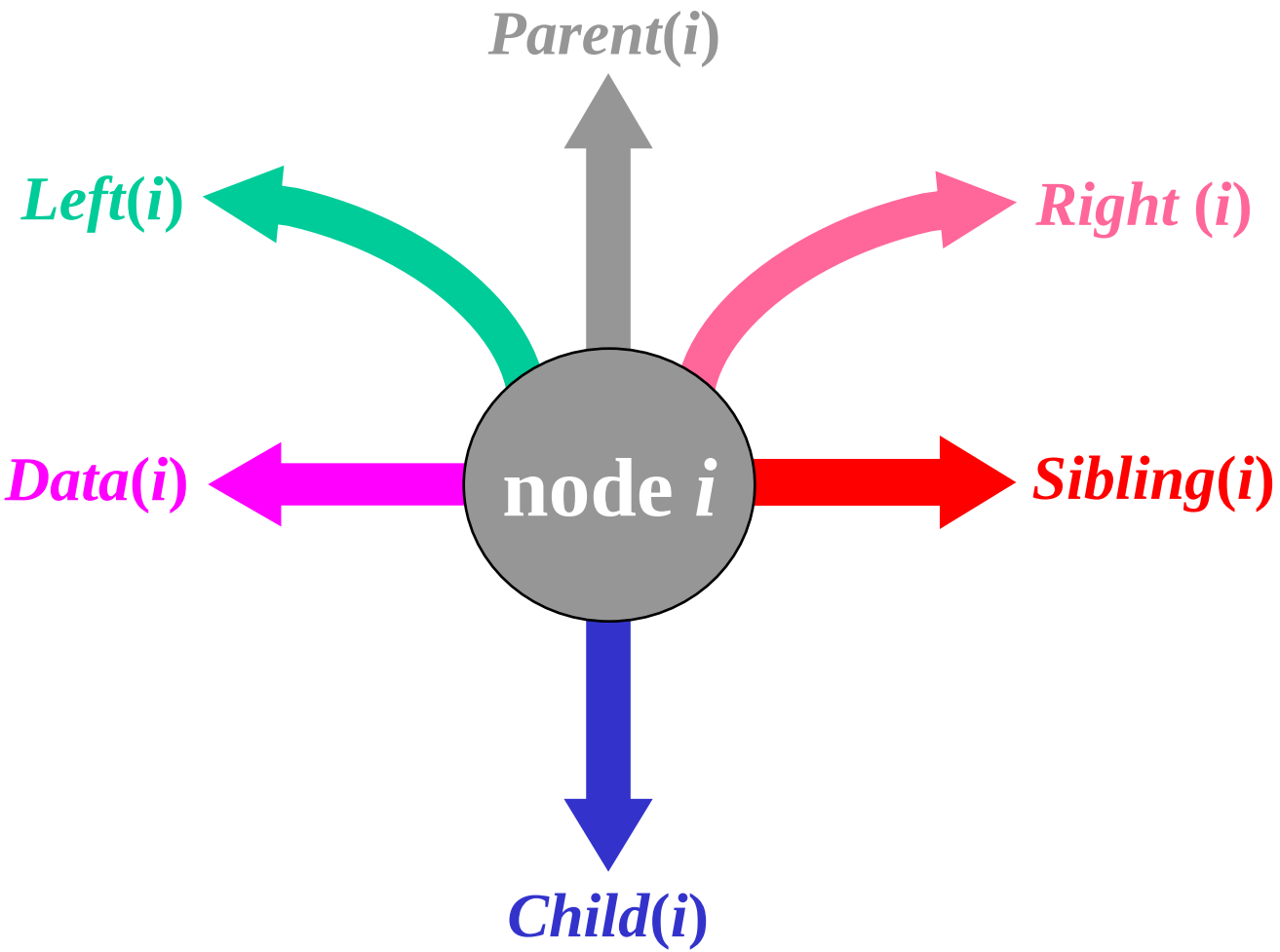
To represent ***ODT*** graphically, two extra links will be needed. The first is left node link of node ***i***, the second is right node link of node ***i***. The links of node ***i*** leading to named nodes are:

1. ***Parent***(***i***) link; 2. ***Child***(***i***) link; 3. ***Sibling***(***i***) link;

4. ***Data***(***i***) link; 5. ***Left*** (***i***) link and 6. ***Right(i)*** link.

S&M algorithm: The Tree Structure

ODT links:
With left and right links
If a link has a value equal to -1, the link is a null link.

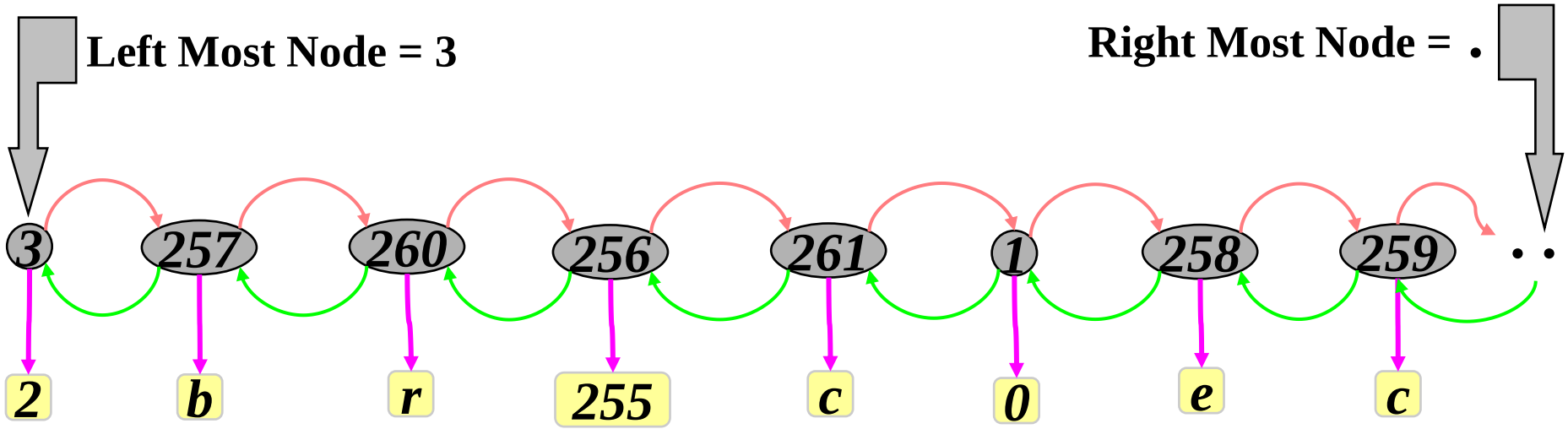


S&M algorithm: The Tree Structure

Graphical Representation of *ODT*

Let ***OD*** = *<3 , 257, 260, 256, 261, 1, 258, 259, ……>*

Data(i) link ***Right***(i) link ***Left***(i) link



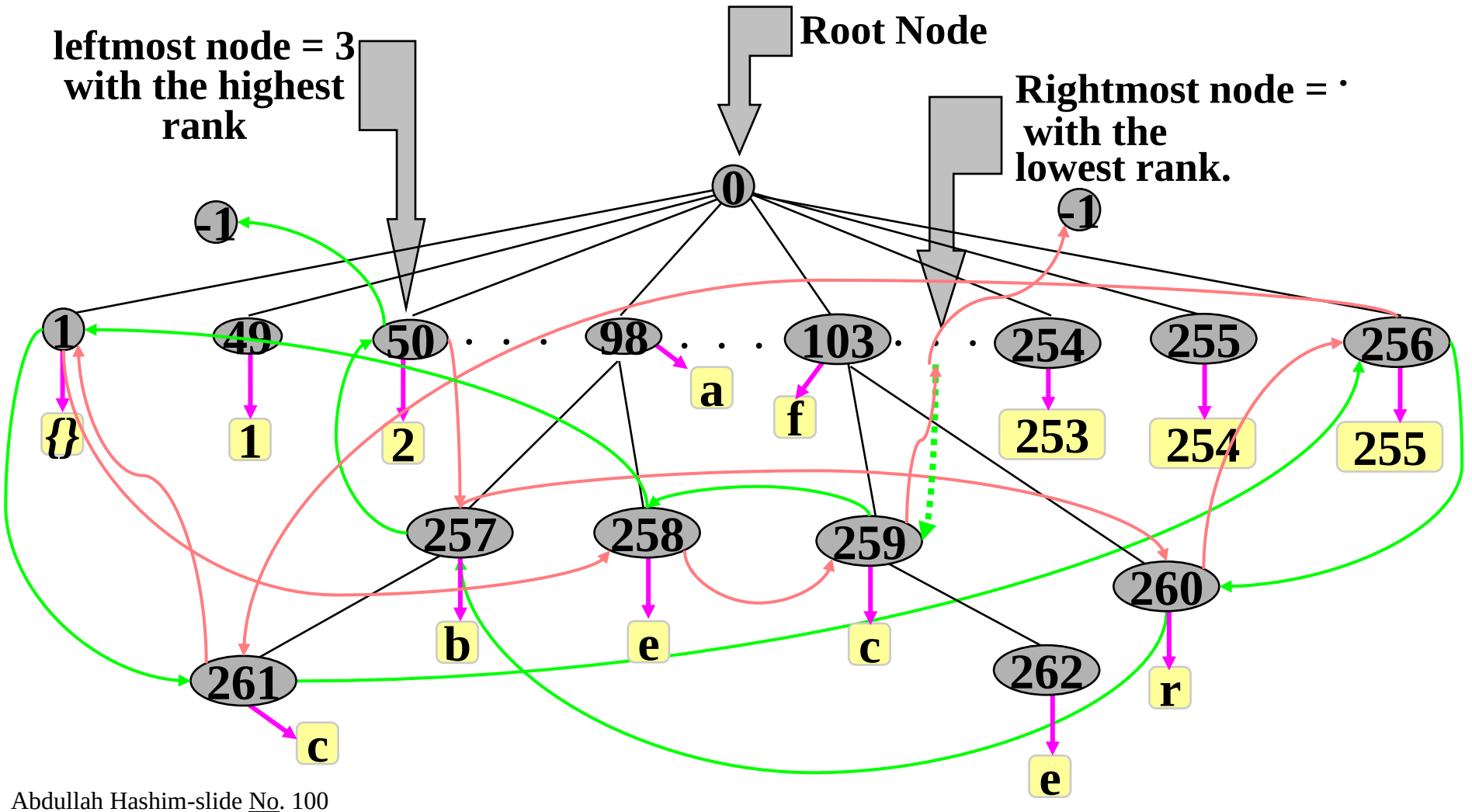
Node **3** has a higher rank than node **257**, node **257** has a higher rank than node **260, 260** has higher rank than **256**, ……etc.

S&M algorithm: The Tree Structure

Graphical Representation of *ODT*

Let ***OD*** = <*3 , 257, 260, 256, 261, 1, 258, 259, ……*>

Data(***i***) link ***Right***(***i***) link ***Left***(***i***) link



S&M algorithm: The Tree Structure

Set s_i: In the proposed ***S&M*** algorithm all nodes of ***EDT*** except the root node are partitioned into sets $s_1, s_2, \ldots, s_{|S|}$, for lossless ***S&M*** system $(\alpha + \chi) \leq |S|$ and usually $|S| = 2^r$, ***r*** a positive integer, while for lossy ***S&M*** system $|S| = 1, 2, \ldots$ depending on the given degradation factor. Each set of nodes s_i corresponds to a set of words W_i in ***ED***. All node sets s_i are mutually exclusive, (i.e. no node is in two sets). The union of the $|S|$sets is equal to the set of all nodes in the ***EDT*** excluding the root node. A set s_i may contains a single ***data-node***, multiple ***data-nodes*** or ***no-data node***. ***Singleton*** is a set with a single ***data-node***.

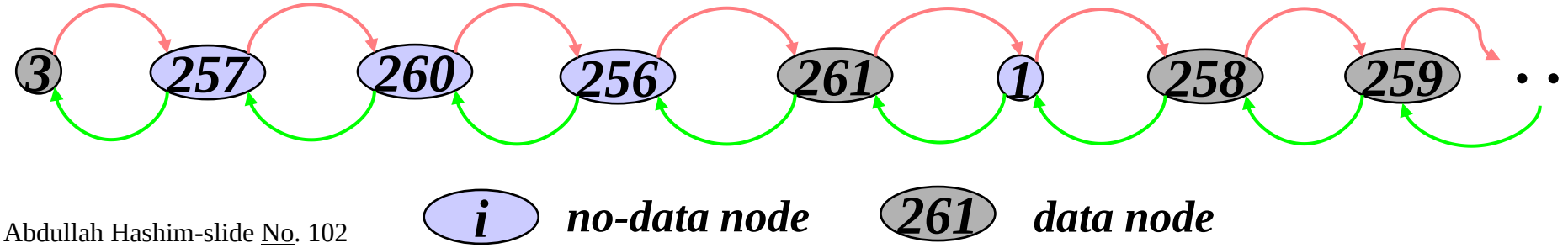
Null set: is a sets with ***no-data*** nodes may be called ***data-empty*** set and denoted by ***{}***.

Multiple data-nodes set: is a set with multiple data nodes, the set frequency of occurrence is equal to the sum of all the frequencies of occurrence of the node in the set.

S&M algorithm: The Tree Structure

Parent singleton set: is a one data-node set (s_i), leading a group of empty-sets called ***satellites*** of s_i. The number of satellites in the group called ***block length*** of s_i and denoted by $SBL(s_i)$ is equal to (2^r-1), (where ***r*** is a positive integer). The satellite block is located in an ordered sequence to the right of s_i in an ***OEDT.*** The last satellite in the block reside in an odd and the parent singleton in an even position of the ***OEDT***. $\{SBL(s_i) + 1\}$ is called the family block length of s_i and denoted by $\{FBL(s_i)\}$, where:
$1 \leq FBL(s_i) \leq (|S|/2)$; $FBL(s_i) = 2^r$, $r = 0, 1, 2, \ldots, \log_2(|S|/2)$.

Let nodes ***257***, ***260***, ***256***, and ***1*** to be ***no-data*** nodes of an ***OEDT***, then nodes ***257***, ***260*** and ***256*** are ***satellites*** of node ***3***, and said to be the ***satellite block*** of node ***3***, $SBL(s_3) = 3$. Node ***1*** is ***satellite*** of node ***261*** and are said to be the ***satellite block*** of node ***261***, $SBL(s_{261}) = 1$. Node ***3*** is said to be of higher ***rank*** than node ***261***.



S&M algorithm: The Tree Structure

Γ_i *is the* ***rank*** *of the* ***non-empty*** *node set* s_i *and its corresponding word set* W_i *in* ***D,*** *while the rank of* ***null satellite*** *node sets are equal to the* ***rank*** *of its parent* ***singleton*** *node* ***rank***.

The ***rank*** of set s_i is directly proportional to its ***satellite block*** length and inversely proportional to its ***set size***. The ***ranks*** of sets with same bock length and size is directly proportional to the maximum wordlength of W_i. It is a positive integer given by the expression:-

$$\Gamma_i = L_{max}(D) \,.\, (|D|_{\max} + SBL(s_i) - |s_i|) + L_{max}(W_i)$$

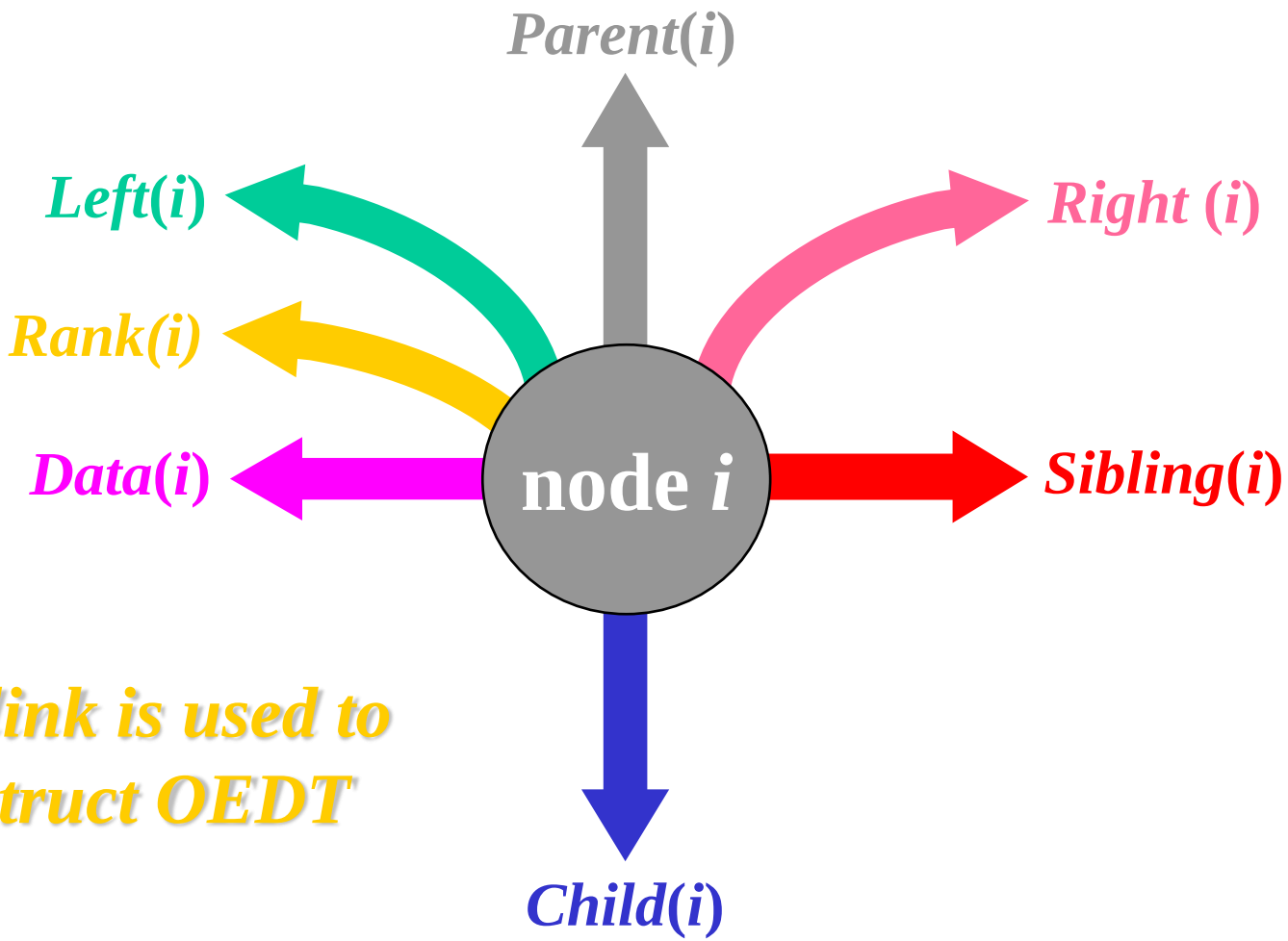
Note:

1. Since satellites have ***rank*** equal to that of its parent ***singleton***, therefore ***satellite block*** sets in ***ODT*** are always positioned immediately on the right of its parent ***singleton*** set.
2. ***Singleton*** sets are ordered in decreasing scale of their ***satellite block*** length and increasing scale of their ***size***.
3. Sets of the same ***satellite block*** length and ***size*** are ordered in decreasing scale of the largest wordlength in W_i; i.e. $L_{max}(W_i)$.

S&M algorithm: The Tree Structure

ODT links:
With rank link

If a link has a value equal -1, the link is a null link



Rank link is used to Construct OEDT

S&M algorithm: The Tree Structure

Coding: is a process of naming the events of an environment Θ by a unique binary codeword.

To code each node v_i of ***EDT***, firstly each set s_i of ***EDT*** is coded by a prefix codeword called ***set codeword*** denoted by $w_{set}(s_i)$. Secondly each node v_i in s_i is coded by a prefix codeword called ***node codeword*** denoted $w_{node}(v_i)$. Each word w_i in the ***ED*** is coded by a ***word codeword*** $w_{word}(w_i)$ which is determined by concatenating the two strings of the set and node codewords.

$w_{word}(w_i) = w_{set}(s_i) + w_{node}(v_i)$, this is a string sum.

The length of the word codeword is therefore equal to the sum of the set and node codeword lengths. Code efficiency is determined by the average codeword length.

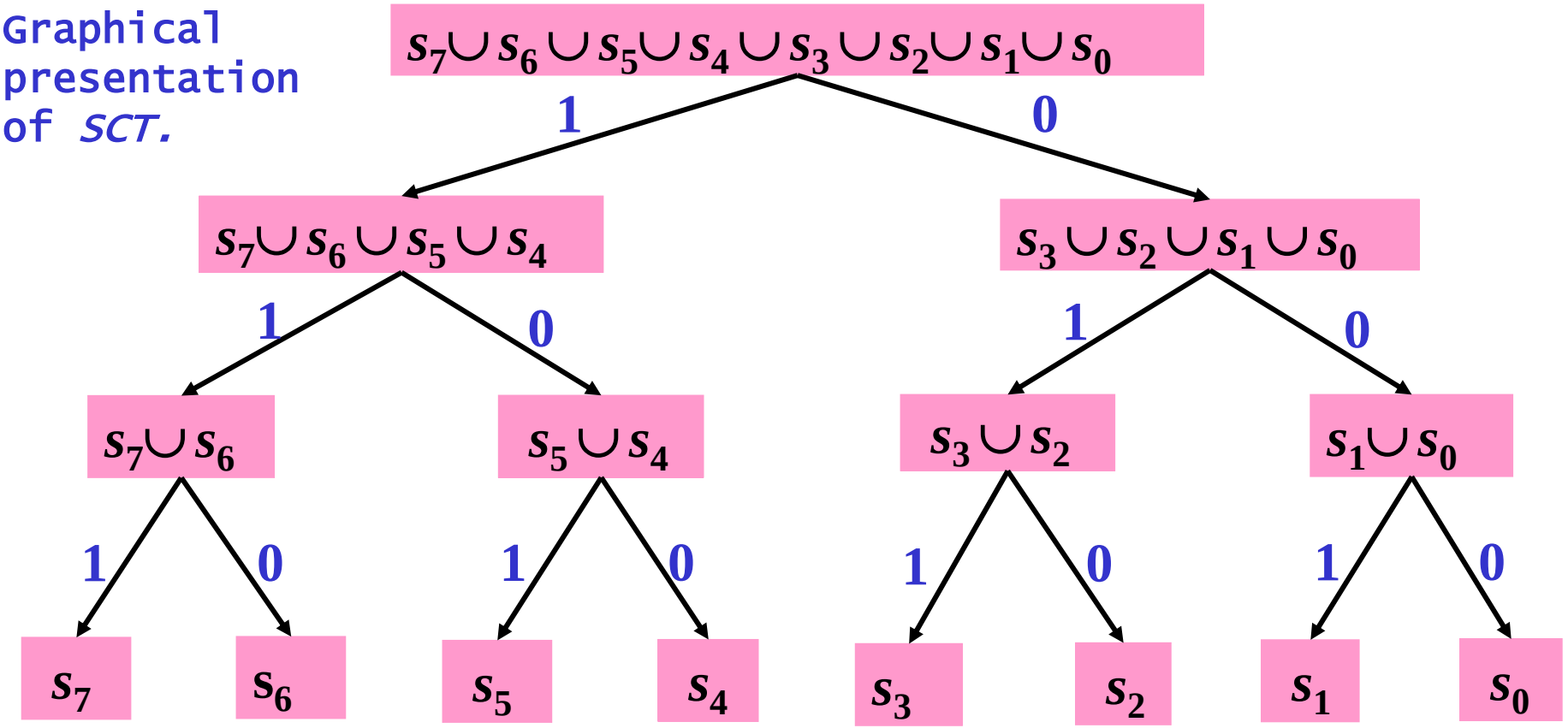
$$L_{average}(ED) = (1 / \Phi) \sum_{over\ \Phi} \mathrm{L}[w_{word}(w_i)]$$ and that of a

compressed file s_{CF}, $L_{average}(s_{CF}) = (1 / |s_{SF}|) \sum_{over\ |s_{CF}|} \mathrm{L}[w_{word}(w_i)]$

S&M algorithm: The Tree Structure

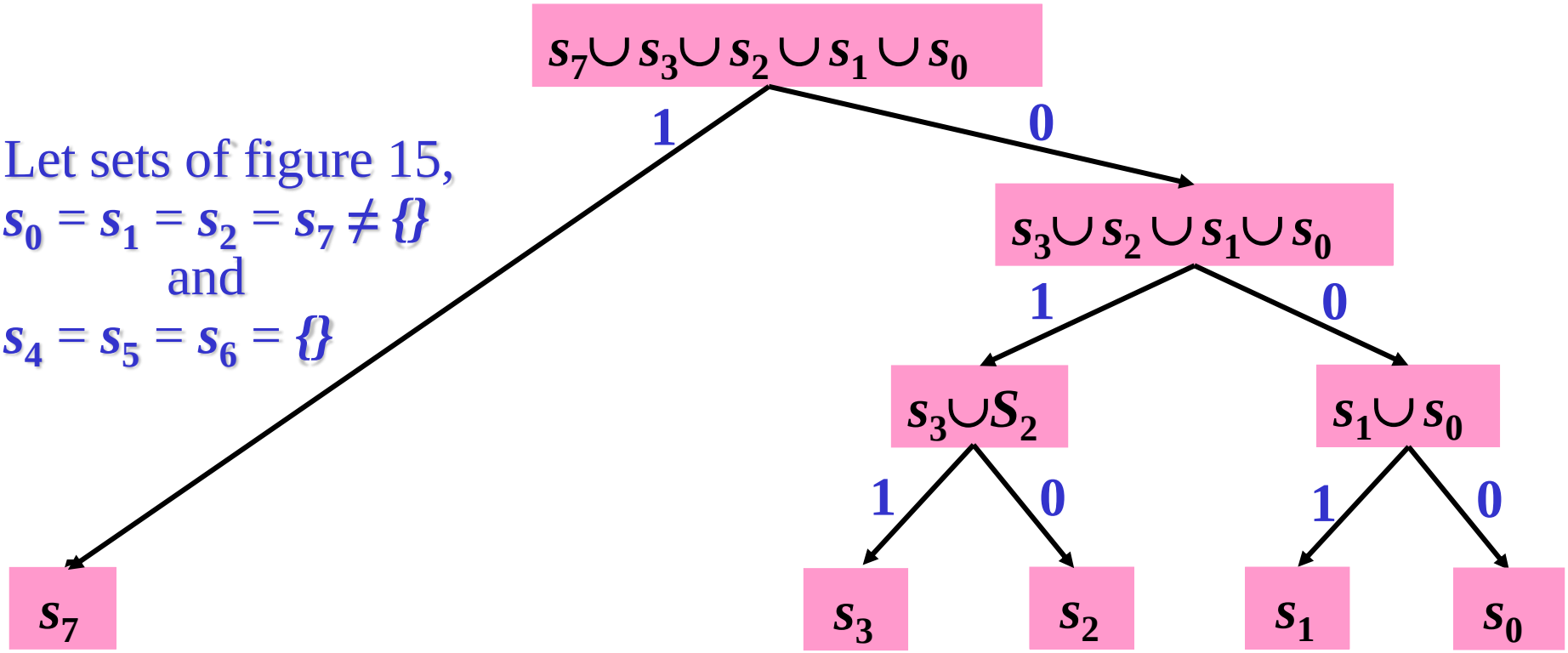
Set-Coding Tree (***SCT***): set-coding tree is a graphical presentation of the ***EDT*** node sets. The links of ***SCT*** presents prefix codeword of sets. Each of the ***SCT*** node contains a set equal to the union of all the sets of its children sets. The root node of ***SCT*** is therefore contains all the sets of ***EDT***. Let ***SCT*** = $<s_7, s_6, s_5, s_4, s_3, s_2, s_1, s_0>$

Graphical presentation of *SCT*.



S&M algorithm: The Tree Structure

Empty sets carries no information, nodes and links of empty sets are therefore redundant and should be removed from the ***SCT***.



Let sets of figure 15,
$s_0 = s_1 = s_2 = s_7 \neq \{\}$
and
$s_4 = s_5 = s_6 = \{\}$

s_7 may be coded by the following variable binary code **1**
s_3 may be coded by the following variable binary code **011**
s_2 may be coded by the following variable binary code **010**
s_1 may be coded by the following variable binary code **001**
s_0 may be coded by the following variable binary code **000**

S&M algorithm: The Tree Structure

Set Coding:

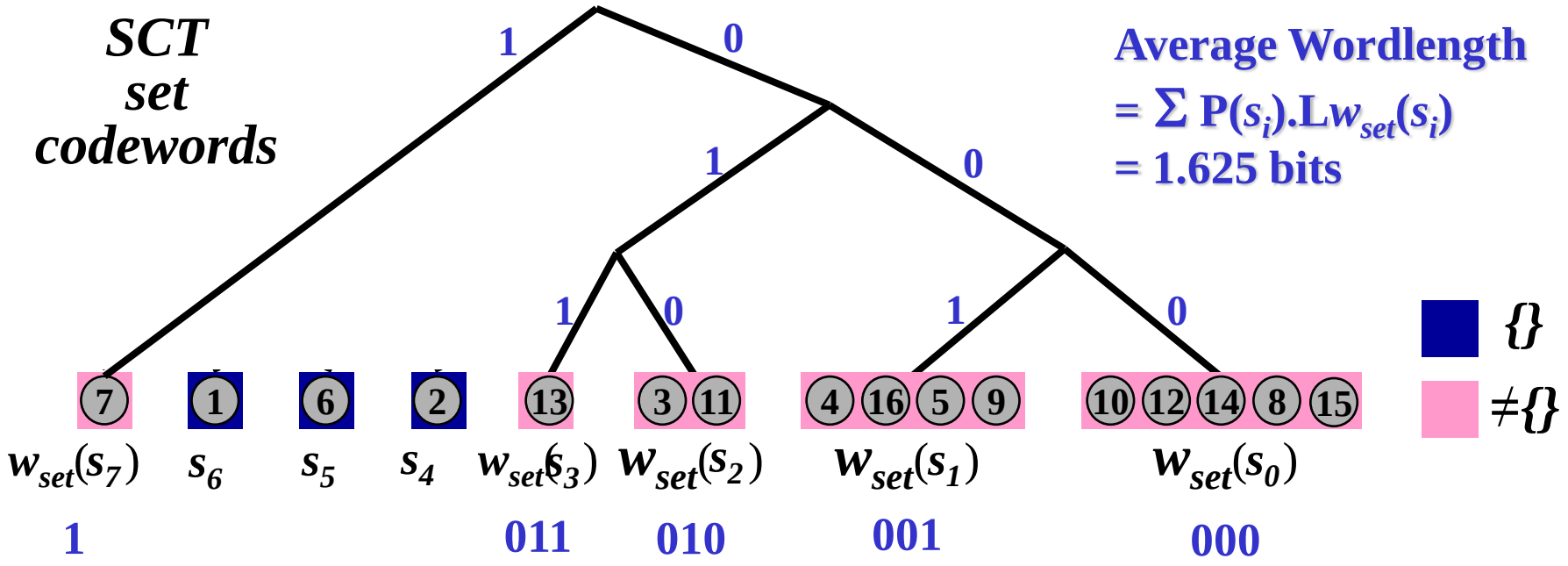
Let an ***OED*** consist of the following words:

$<w_7, w_1, w_6, w_2, w_{13}, w_3, w_{11}, w_4, w_{16}, w_5, w_9, w_{10}, w_{12}, w_{14}, w_8, w_{15}>$ corresponding to the following nodes in ***OEDT*:**

$<v_7, v_1, v_6, v_2, v_{13}, v_3, v_{11}, v_4, v_{16}, v_5, v_9, v_{10}, v_{12}, v_{14}, v_8, v_{15}>$ partitioned into 8 sets: $<s_7, s_6, s_5, s_4, s_3, s_2, s_1, s_0>$; where:

$s_7 = \{v_7\}$; $s_6 = \{v_1\} = \{\}$; $s_5 = \{v_6\} = \{\}$; $s_4 = \{v_2\} = \{\}$; $s_3 = \{v_{13}\}$;

$s_2 = \{v_3, v_{11}\}$; $s_1 = \{v_4, v_{16}, v_5, v_9\}$; and. $s_0 = \{v_{10}, v_{12}, v_{14}, v_8, v_{15}\}$.



S&M algorithm: The Tree Structure

Identifying sets: *Set links* (***SL***) are used to identify a set s_i and determine its codeword $\{w_{set}(s_i)\}$. ***SL*** connects the first node say (***m***) to the last node say (***n***) of a given set or the union of (2^r) adjacent sets in an ***OEDT***, (where ***r*** is a positive integer). If node ***n*** is on the right of node ***m*** it is called a right set-link (***RSL***). If node ***n*** is on the left of node ***m*** the link is said to be a left set-link (***LSL***). Nodes surrounded by a link are the nodes of a given set or union of sets. The set-link is said to be of height (***h***) and is denoted by ***SL***(***h***) if it surrounds the nodes of the union of two set-links of height (***h*-1)** denoted by ***SL***(***h* -1**). The set-link of height zero ***SL***(**0**) surrounds all the nodes of a single set s_i. The number of set-links of height ***h*** is twice the number of set-links of height ***h*+1**. The ***i***-*th* right set-link denoted by ***RSL***$(h)_i$ and the corresponding ***i***-*th* left set-link denoted ***LSL***$(h)_i$ surrounds the same nodes. Each of the set-links is coded by a single binary digit. The zero height set-link has the least significant digit and the largest height set-link has the most significant digit in a set's codeword $w_{set}(s_i)$. Set-links are of two types. The first (***type* 1**) contains at least one data node and it is coded by a single binary digit. The second (***type* 0**) is redundant and contains only no-data, and surround a ***null*** nodes.

S&M algorithm: The Tree Structure

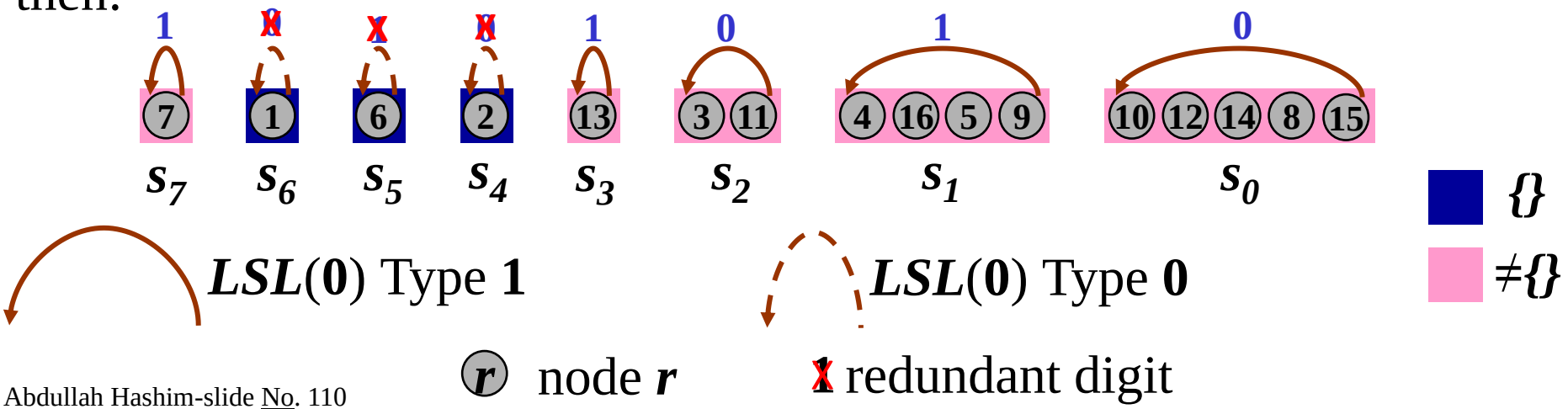
Left set-links of height zero LSL(0)$_i$: is a set link connecting the rightmost to the leftmost node of a single set s_i. ***RSL(0)$_i$*** connect the leftmost to the rightmost node of s_i.

Let an ***OED*** consist of the following words:

$<w_7, w_1, w_6, w_2, w_{13}, w_3, w_{11}, w_4, w_{16}, w_5, w_9, w_{10}, w_{12}, w_{14}, w_8, w_{15}>$ corresponding to the following nodes in ***OEDT*:**

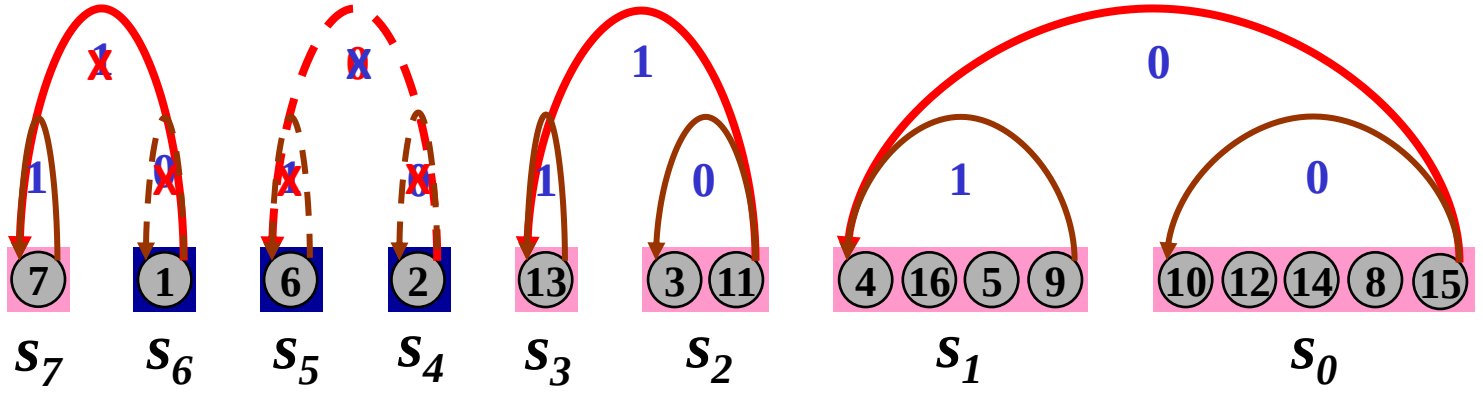
$<v_7, v_1, v_6, v_2, v_{13}, v_3, v_{11}, v_4, v_{16}, v_5, v_9, v_{10}, v_{12}, v_{14}, v_8, v_{15}>$ partitioned into 8 sets: $<s_7, s_6, s_5, s_4, s_3, s_2, s_1, s_0>$; where:

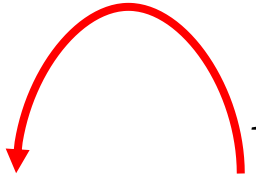
$s_7 = \{v_7\}$; $s_6 = \{v_1\} = \{\}$; $s_5 = \{v_6\} = \{\}$; $s_4 = \{v_2\} = \{\}$; $s_3 = \{v_{13}\}$; $s_2 = \{v_3, v_{11}\}$; $s_1 = \{v_4, v_{16}, v_5, v_9\}$; and. $s_0 = \{v_{10}, v_{12}, v_{14}, v_8, v_{15}\}$; then:



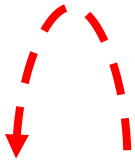
S&M algorithm: The Tree Structure

Left set-links of height one $\{LSL(1)_i\}$: is a set-link connecting the rightmost node of the (***2i***)-***th*** left-set-link $\{LSL(0)_{2i}\}$ to the leftmost node of the (***2i+1***)-*th* left-set-link $\{LSL(0)_{2i+1}\}$. If $LSL(0)_{2i+1}$ is a null-link then $LSL(1)_i$ surrounds only nodes of $LSL(0)_{2i}$ and (in this case) $LSL(0)_{2i}$ becomes redundant. $RSL(1)_i$ surrounds the same nodes of $LSL(1)_i$. If one of the two $SL(0)$ set-links is of type **0**, then the $SL(0)$ type **1** becomes redundant. $SL(1)$ type **1** surrounds at least one $SL(0)$ of type **1**. $SL(1)$ type **0** surrounds only type **0** $\{SL(0)\}$ set-links.





$LSL(1)$ Type **1**



$LSL(1)$ Type **0**

(r) node ***r***

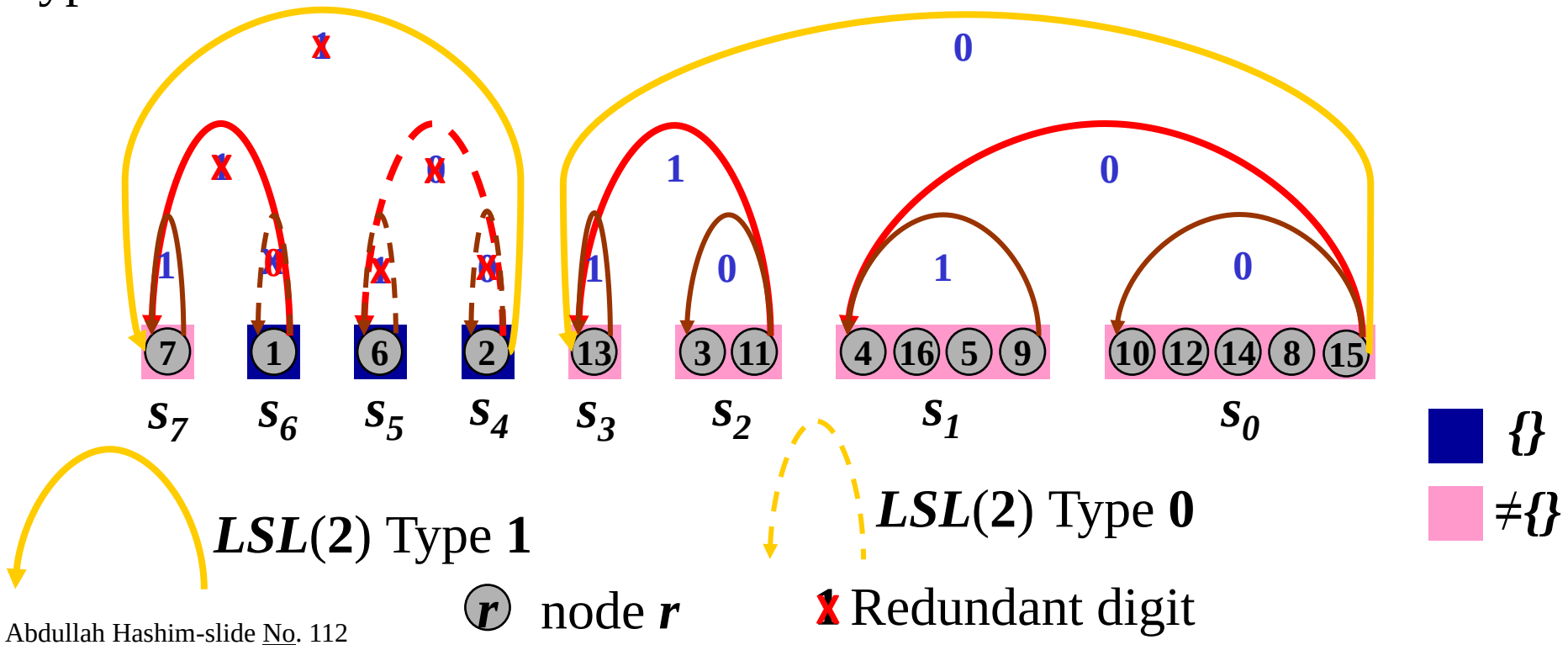
X redundant digit

{}

≠{}

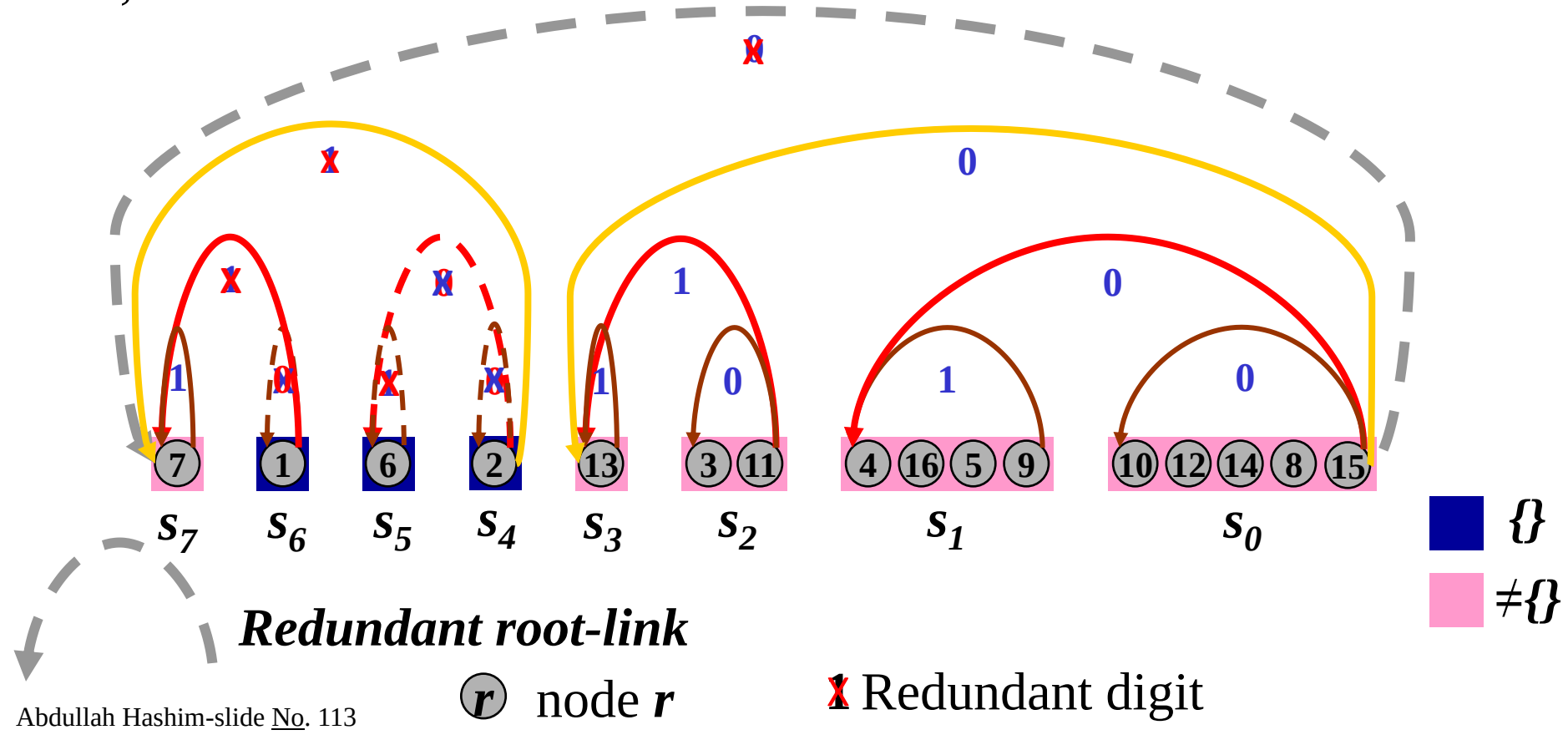
S&M algorithm: The Tree Structure

Left set-links of height two $\{LSL(2)_i\}$**:** is a set-link connecting the rightmost node of the (***2i***)-***th*** left-set-link $\{LSL(1)_{2i}\}$ to the leftmost node of the (***2i+1***)-*th* left-set-link $\{LSL(1)_{2i+1}\}$. If $LSL(1)_{2i+1}$ is a null-link then $LSL(2)_i$ surrounds only nodes of $LSL(1)_{2i}$ and (in this case) $LSL(1)_{2i}$ becomes redundant. $RSL(2)_i$ surrounds the same nodes of $LSL(2)_i$. If one of the two $SL(1)$ set-links is of type **0**, then the $SL(1)$ type **1** becomes redundant.



S&M algorithm: The Tree Structure

Left set-links of height three $\{LSL(3)_i\}$**:** is a set-link connecting the rightmost node of the (***2i***)-***th*** left-set-link $\{LSL(2)_{2i}\}$ to the leftmost node of the (***2i+1***)-***th*** left-set-link $\{LSL(2)_{2i+1}\}$. If $LSL(1)_{2i+1}$ is a null-link then $LSL(3)_i$ surrounds only the nodes of $LSL(2)_{2i}$ and (in this case) $LSL(2)_{2i}$ becomes redundant. A link surrounding all nodes of all sets, it is redundant an called root set-link.



S&M algorithm: The Tree Structure

Notes on Set Coding Tree *SCT* :

1 $SL(h)_i$ is the i-th set-link of height h, where $i = 1, 2, \ldots, r$; $r = \lfloor N / 2^h \rfloor$, and N is the number of sets in SCT. $SL(h)_I$ surrounds the nodes of set-links $SL(h\text{-}1)_{2i}$ and $SL(h\text{-}1)_{2i+1}$. If $SL(h\text{-}1)_{2i+1}$ is a null-link then $SL(h)_i$ surrounds only the nodes of $SL(h\text{-}1)_{2i}$ and (in this case) $SL(h\text{-}1)_{2i}$ becomes redundant.

2 Links pointing from left to right are called right links, and links pointing from right to left are called left links.

3 $LSL(h)_i$ and the $RSL(h)_i$ surrounds the same nodes.

4 set-links have a single binary digit code, the i-th set-link is coded by binary zero if i is an even integer and binary one if i is an odd integer. The zero height set-link has the least significant digit and the largest height set-link has the most significant digit in the set's codeword. A set-link holding no data is redundant and has no code.

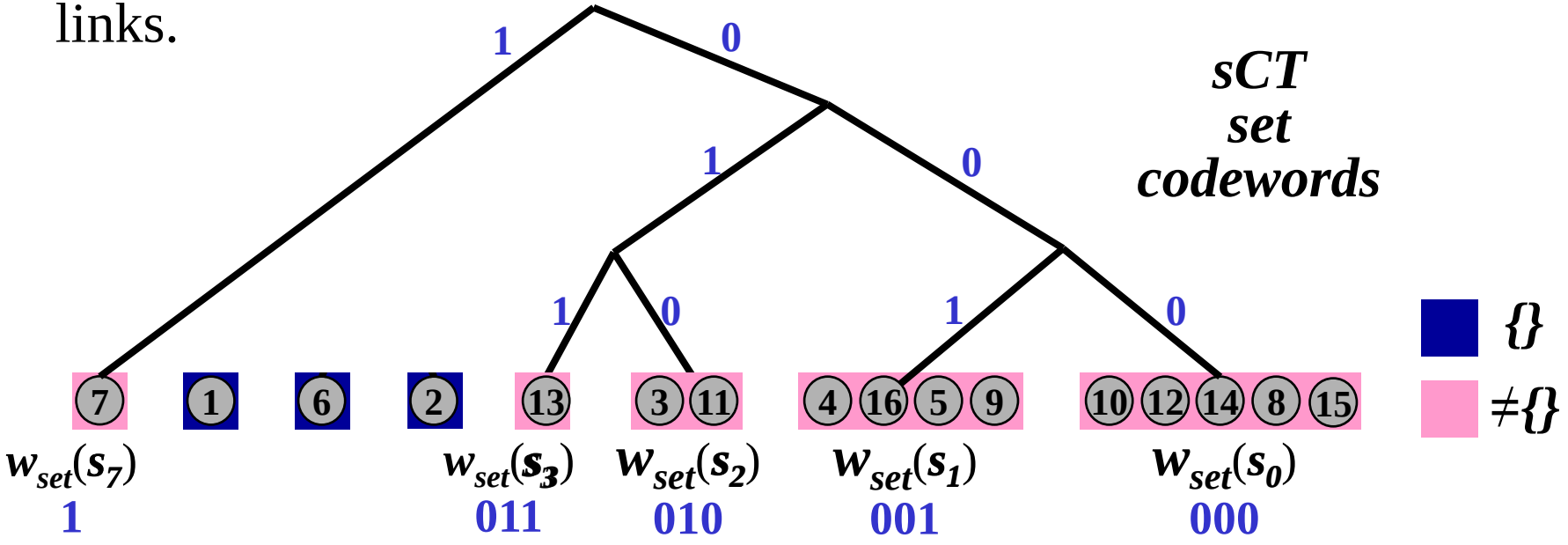
5 A link surrounding all nodes of all sets is called root set-link and is redundant. [$SLd(SCT)$], where $d(SCT)$ is the depth of SCT denoted by the letter (t); $t = \lfloor \log_2 N \rfloor$ is known as the root-link.

S&M algorithm: The Tree Structure

6. ***SL*(*0*)** surrounds the nodes of a single set.

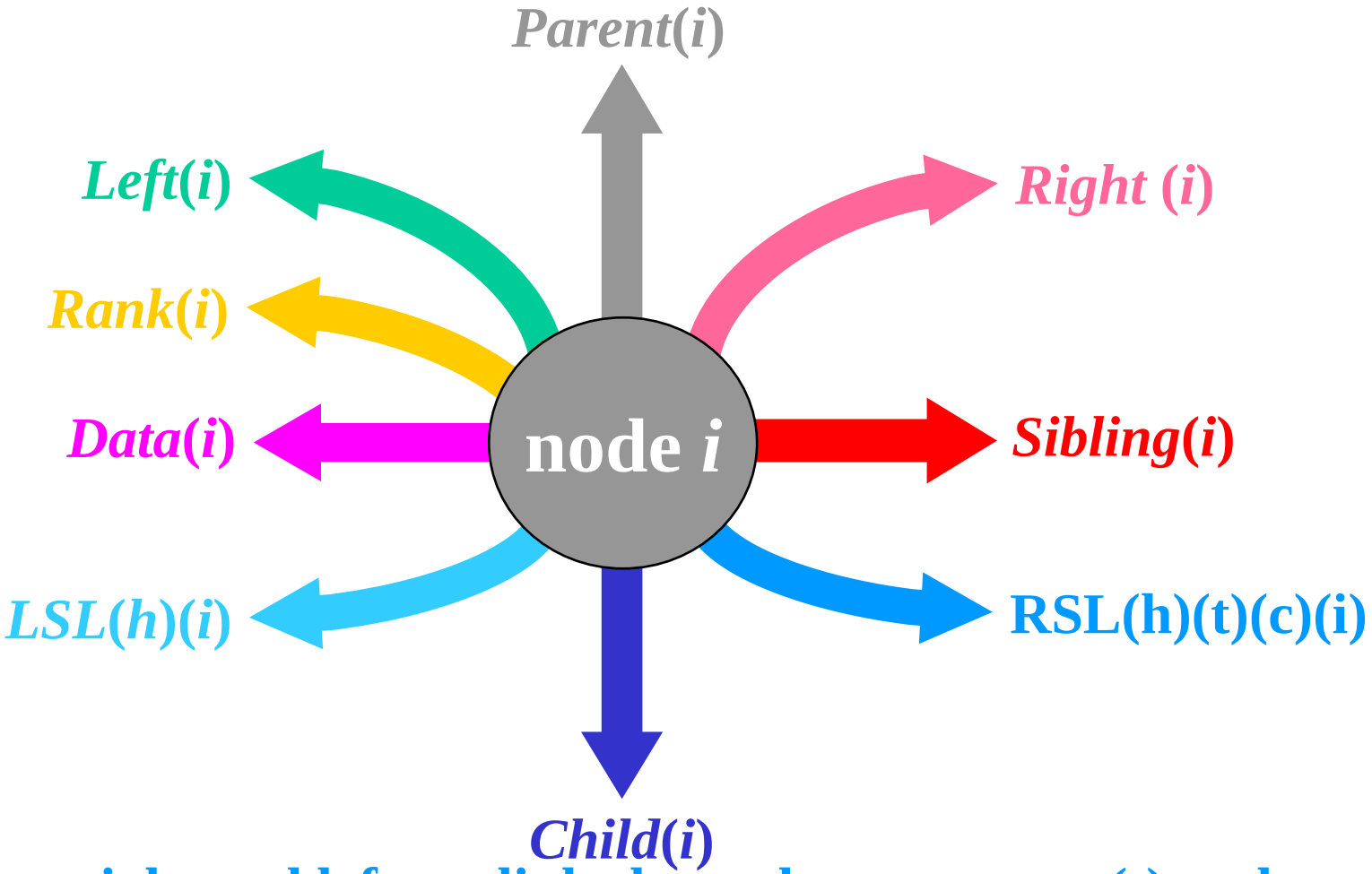
7. set-links are of two types, type **1** surrounds at least one data node, while type **0** surrounds only no-data nodes. A type **1** set-link is coded by a single binary digit, while type **0** is a redundant link.

8. If ***SL*(*h*+1)** surrounds all nodes of two adjacent ***SL*(*h*)** set-links and if one of the two ***SL*(*h*)** set-links is of type **0**, then the ***SL*(*h*)** type **1** becomes redundant. ***SL*(*h*+1)** type **1** surrounds at least one ***SL*(*h*)** of type **1**. ***SL*(*h*+1)** type **0** surrounds only type **0** ***SL*(*h*)** set-links.



S&M algorithm: The Tree Structure

ODT* links**: The coding digit (c***) may take three values, 0, 1 for binary zero and one respectively and 2 for redundant digit.

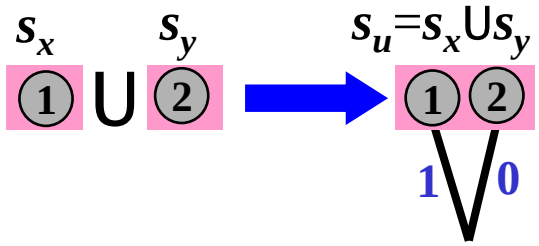


Since right and left set-links have the same type (t) and coding digit (c), left set-link type and code digit is ignored.

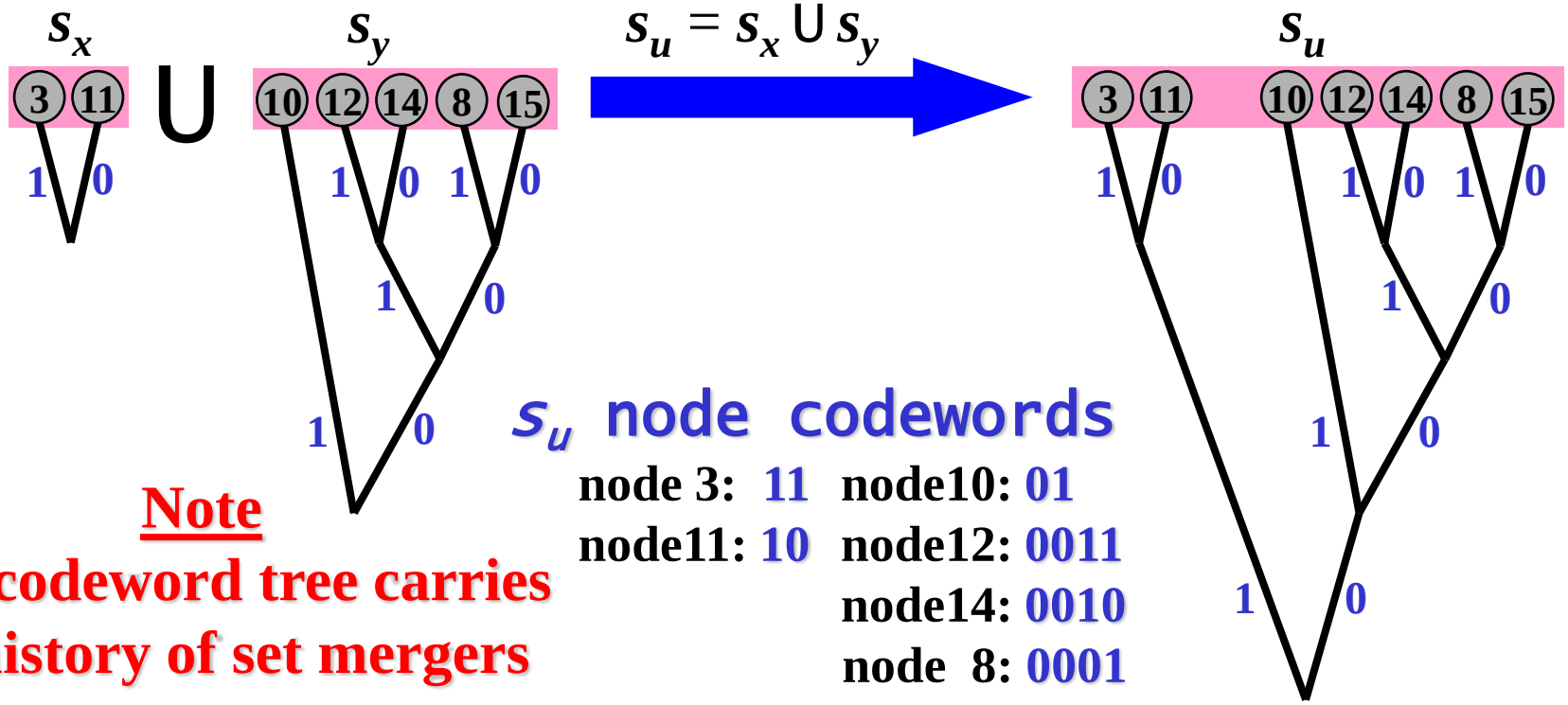
S&M algorithm: The Tree Structure

Node Coding:

When two singleton s_x and s_y sets merge into one set s_u, the two nodes in s_u are coded by a single binary digit (1) and (0) respectively.



similarly, the merger of two multi nodes sets, s_x and s_y into set (s_u), $s_u = s_x \cup s_y$, if $|s_x| = 2$, $|s_y| = 5$, then $|s_u| = 7$, nodes of s_x coded by binary digit (0) and that of s_y by (1), as shown below.



Note

node codeword tree carries the history of set mergers

s_u node codewords

node 3: 11 **node10: 01**
node11: 10 **node12: 0011**
node14: 0010
node 8: 0001
node05: 0000

S&M algorithm: The Tree Structure

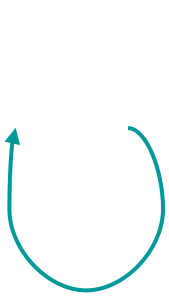
Node coding: Initially the dictionary contains α single character words corresponding to the alphabet in A and χ single character words corresponding to control elements in C. All sets in the initial tree have zero length node codewords. Multiple node sets are formed by the successive merging process. Assuming the two sets to be merged have equal probability and a single binary digit will be appended to the merged set node codewords. The node codewords of the first set of the merger are appended by one, while those of the second set of the merger are appended by zero. This process ensures that the merged set of nodes are optimally coded. To ensure robustness, in sets with node codewords of length equal to a given Ψ have their NCT code bounded to keep the maximum height within the bound.

Node codewords (w_{node}) are appended to their corresponding set codewords (w_{set}) to form the compressed word codewords (w_{word}).

S&M algorithm: The Tree Structure

Node Coding: ***node-links (NL)*** are used to determine node codeword $w_{node}(v_i)$ for a node v_i in set s_i. ***NL*** is a link between nodes within a set. It has the same structure as the ***SL***; it consists of right and left node-links, {***RNL*(*h*)**} and {***LNL*(*h*)**} respectively, of height (***h***) (where $h = 0, 1, \ldots, h_{max}$, and $\lfloor \log_2 (|s_i|_{max}) \rfloor \leq h_{max} \leq (\Phi - N)$). $RNL(h)_i$ and $LNL(h)_i$ surrounds the same nodes. Node-links have no type unlike the case of ***SL***. A link containing all nodes in a set s_i and its corresponding node-link $NL(d_{NCT})$, (where d_{NCT} is the binary ***NCT*** depth) are redundant. $NL(d_{NCT})$ is known as the root node-link. All single node sets have only root node-links.

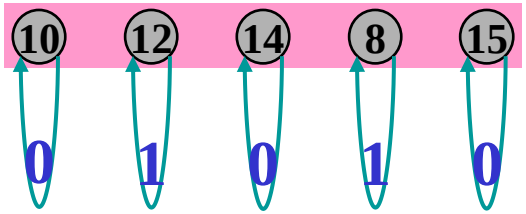
Node-links of height zero **{*NL*(0)}:** is a node-link surrounding a single node. The ***LNL*(0)** and ***RNL*(0)** surrounds the same single node. Consider set s_0 of the previous examples: $s_0 = \{v_{10}, v_{12}, v_{14}, v_8, v_{15}\}$.



***LNL*(0)**

node ***r***

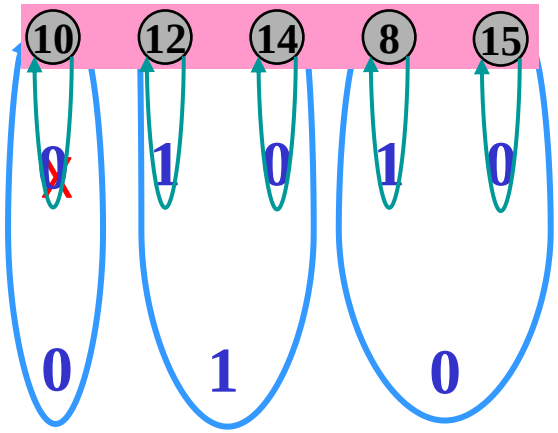
***NCT* of s_0; *h* = 0**



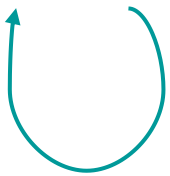
S&M algorithm: The Tree Structure

Left node-link of height One $\{LNL(1)_i\}$**:** is a node-link connecting the rightmost node of the (***2i***)-***th*** left-node-link $\{LNL(0)_{2i}\}$ to the leftmost node in the (***2i+1***)-***th*** left-node-link $\{LNL(0)_{2i+1}\}$. If $LNL(0)_{2i+1}$ is a null-link then $LNL(1)_i$ surrounds the only single node of $LNL(0)_{2i}$ and (in this case) $LNL(0)_{2i}$ becomes redundant. $RNL(1)_i$ and $LNL(1)_i$ contain the same nodes.

***NCT* of s_0; h = 0 and 1**



1 Redundant digit



$LNL(1)$

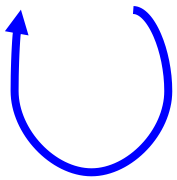
r node ***r***

S&M algorithm: The Tree Structure

Left node-link of height two $\{LNL(2)_i\}$**:** is a node-link connecting the rightmost node of the (***2i***)-**th** left-node-link $\{LNL(1)_{2i}\}$ to the leftmost node in the (***2i+1***)-**th** left-node-link $\{LNL(1)_{2i+1}\}$. If $LNL(1)_{2i+1}$ is a null-link then $LNL(2)_i$ surrounds all nodes of $LNL(1)_{2i}$ and (in this case) $LNL(1)_{2i}$ becomes redundant. $RNL(2)_i$ and $LNL(2)_i$ surrounds the same nodes.

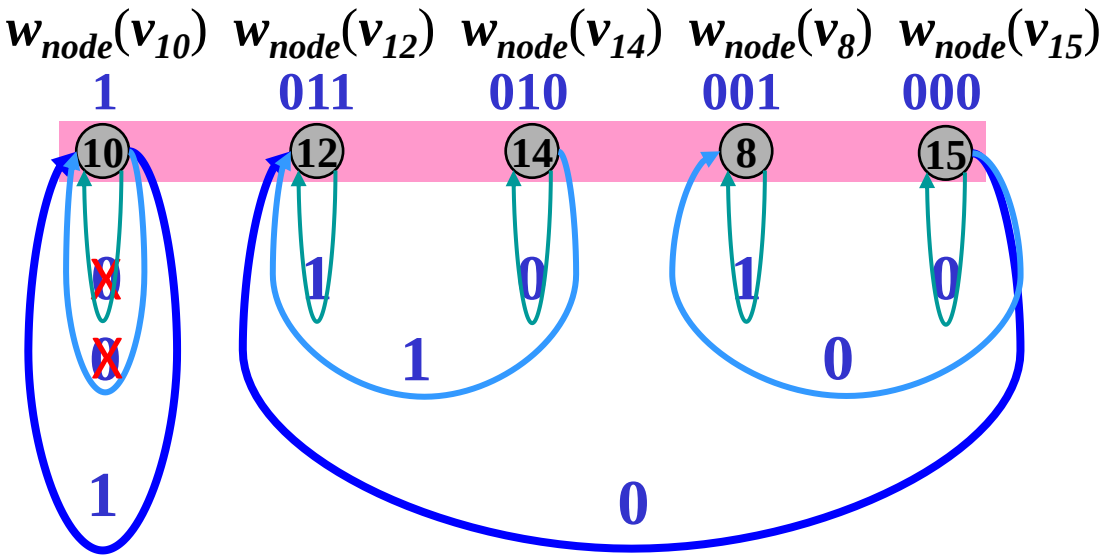
Node	Codeword
10	1
12	011
14	010
8	001
15	000

~~1~~ Redundant digit



$LNL(2)$

Set s_1: node codewords $w_{node}(v_i)$

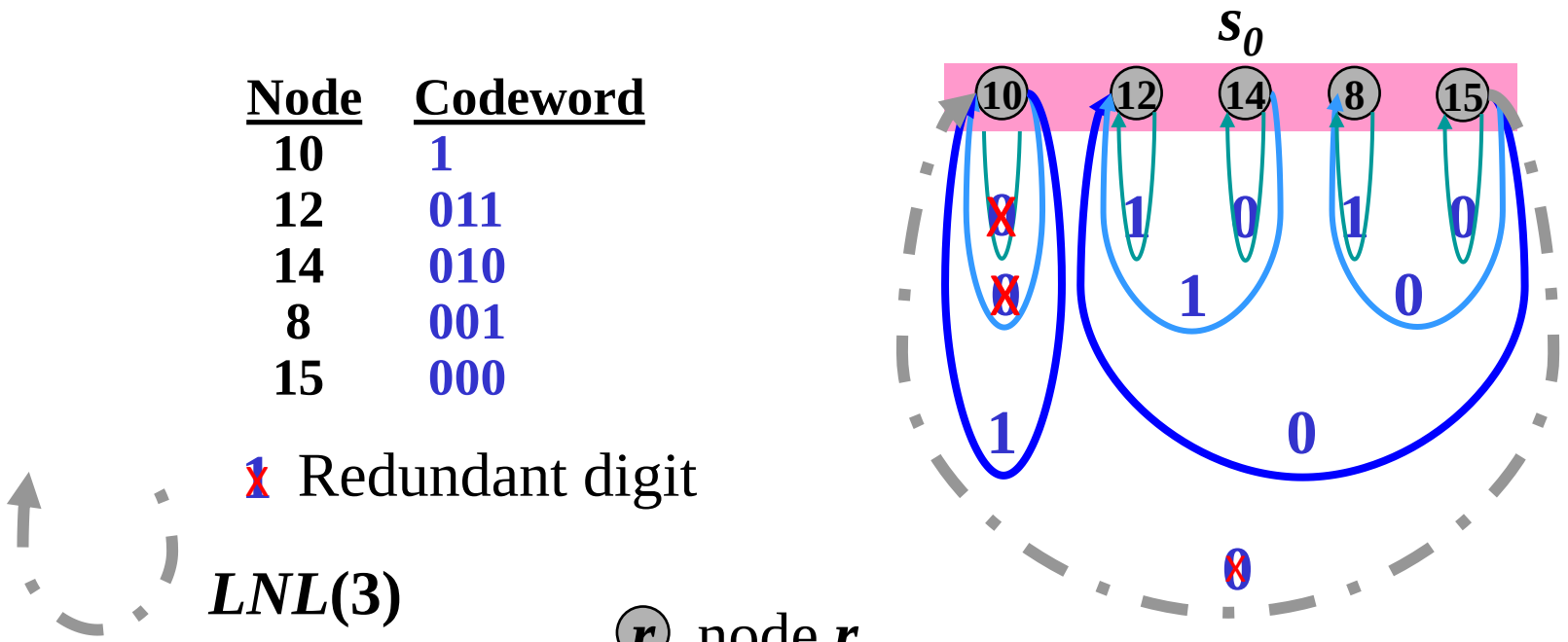


(r) node ***r***

S&M algorithm: The Tree Structure

Left node-link of height three $\{LNL(3)_i\}$**:** is a node-link connecting the rightmost node of the (***2i***)**-th** left-node-link $\{LNL(2)_{2i}\}$ to the leftmost node of the (***2i+1***)-th left-node-link $\{LNL(2)_{2i+1}\}$. If $LNL(2)_{2i+1}$ is a null-link then $LNL(3)_i$ surrounds all nodes of $LNL(2)_{2i}$ and (in this case) $LNL(2)_{2i}$ becomes redundant. Right node-links $RNL(3)_i$ surrounds the same nodes of left node-links $LNL(3)_i$. A link containing all nodes of a set is redundant, known as the root node-link.

Node	Codeword
10	1
12	011
14	010
8	001
15	000



S&M algorithm: The Tree Structure

Notes on Node Coding Tree *NCT*:

1. The i-th node-link ($NL(h)_i$) surrounds the nodes in a set (s_i) is said to be of height h, where $i = 0, 1, \ldots, m-1$; $m = \lfloor |s_i|/2^h \rfloor$. $NL(h)_i$ surrounds the nodes of node-links $NL(h-1)_{2i}$ and $NL(h-1)_{2i+1}$. If $NL(h-1)_{2i+1}$ is a null-link then $NL(h)_i$ surrounds only the nodes of $NL(h-1)_{2i}$ and (in this case) $NL(h-1)_{2i}$ becomes redundant.

2 Links pointing from left to right are called right links, and links pointing from right to left are called left links.

3 $LNL(h)_i$ and the $RNL(h)_i$ surrounds the same nodes.

4 Node-links have a single binary digit code, the i-th node-link is coded by binary zero if i is an even integer and binary one if i is an odd integer. The zero height node-link has the least significant digit and the largest height node-link has the most significant digit in the node codeword $w_{node}(s_i)$.

S&M algorithm: The Tree Structure

Notes on Node Coding Tree *NCT* (Continued):

5 A link surrounding all nodes of a set s_i is denoted by ($NL(d_{NCT})$, (where d_{NCT} is the binary ***NCT*** depth) are redundant. $NL(d_{NCT})$ is known as the root node-link.

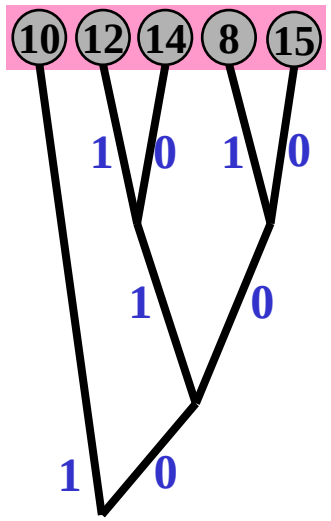
6. ***NL*(0)** surrounds a single node.

7. A node in a ***singleton*** set has a redundant root node-link ***NL*(0)**.

8. ***NCT*** is constructed from right to left (***i*** = 0, 1, 2, …).

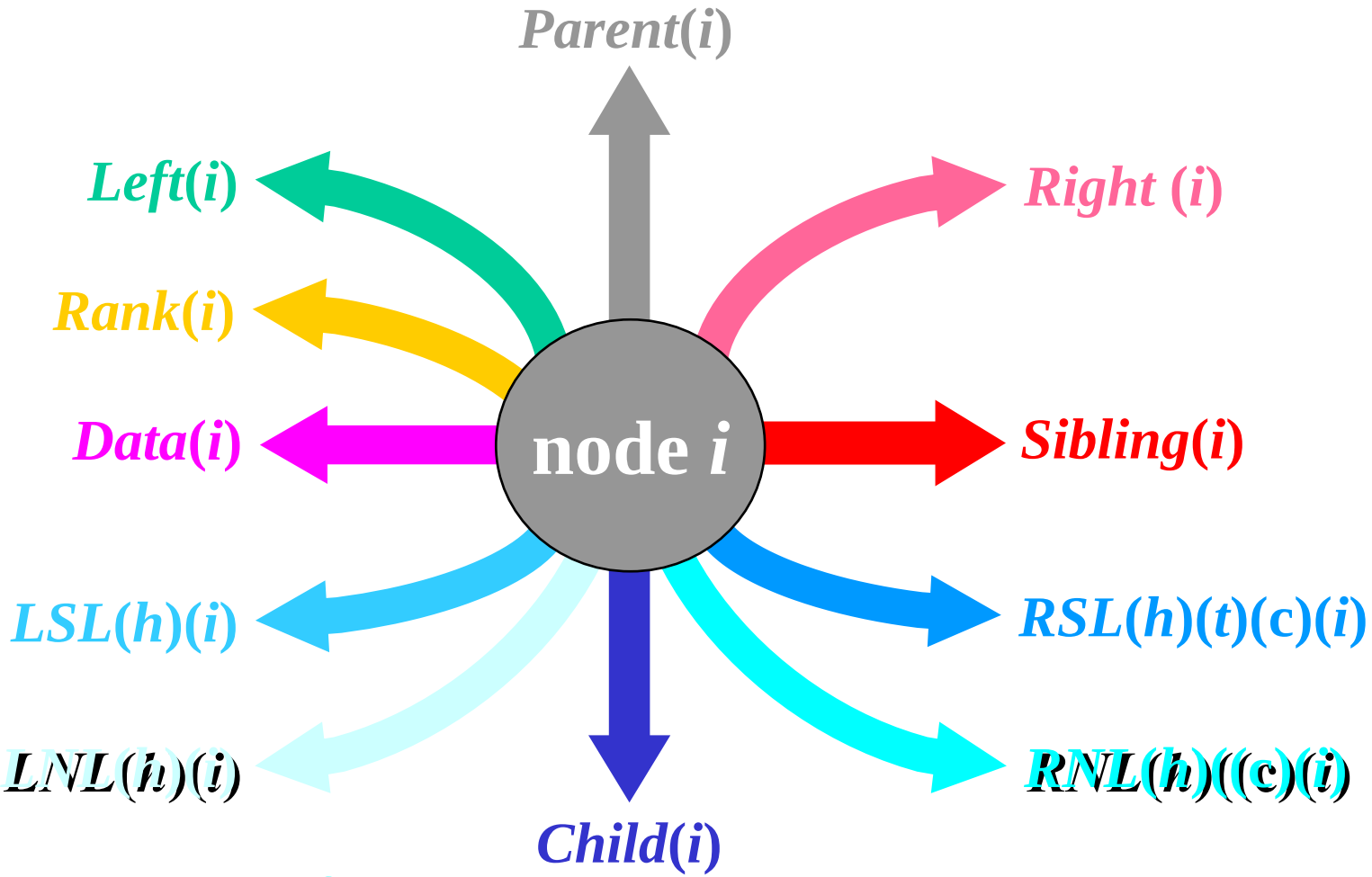
NCT node codewords

Node	Codeword
10	1
12	011
14	010
8	001
15	000



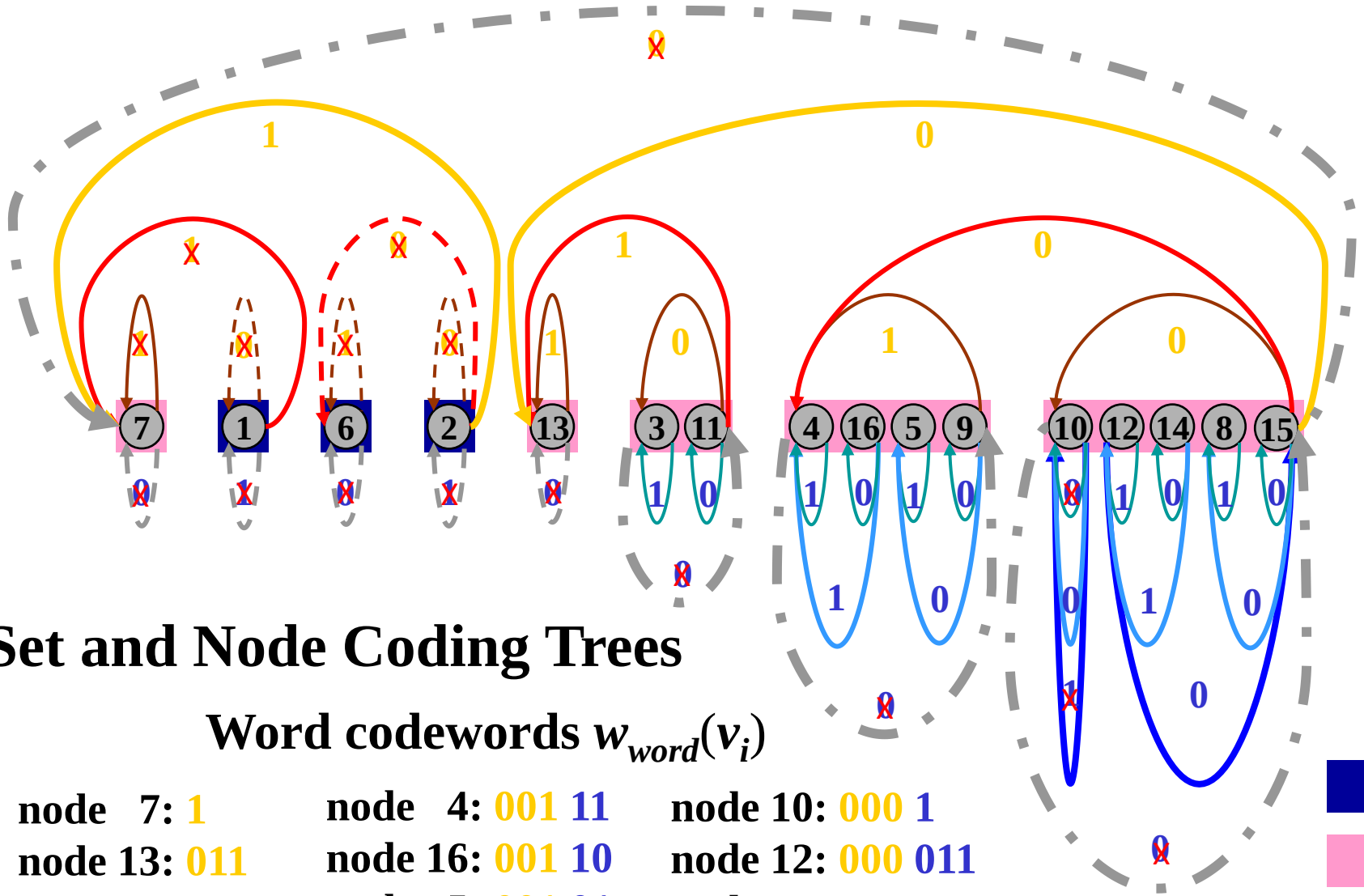
S&M algorithm: The Tree Structure

ODT* links**: The coding digit (c***) may take three values, 1, 1 for binary zero and one respectively and 2 for redundant digit.



Since right and left node-links have the same coding digit (c), left node-link code digit is ignored.

S&M algorithm: The Tree Structure



Set and Node Coding Trees

Word codewords $w_{word}(v_i)$

node 7: 1
node 13: 011
node 3: 010 1
node 11: 011 0
node 4: 001 11
node 16: 001 10
node 5: 001 01
node 9: 001 00
node 10: 000 1
node 12: 000 011
node 14: 000 010
node 8: 000 001
node 15: 000 000

{}
≠{}

S&M algorithm: The Tree Structure

The word codeword

$w_{word} = w_{set}(s_i) + w_{node}(v_i)$

String addition

Average Wordlength

$= \Sigma P(v_i).Lw_{word}(v_i)$

= 0.5+0.375+0.5+

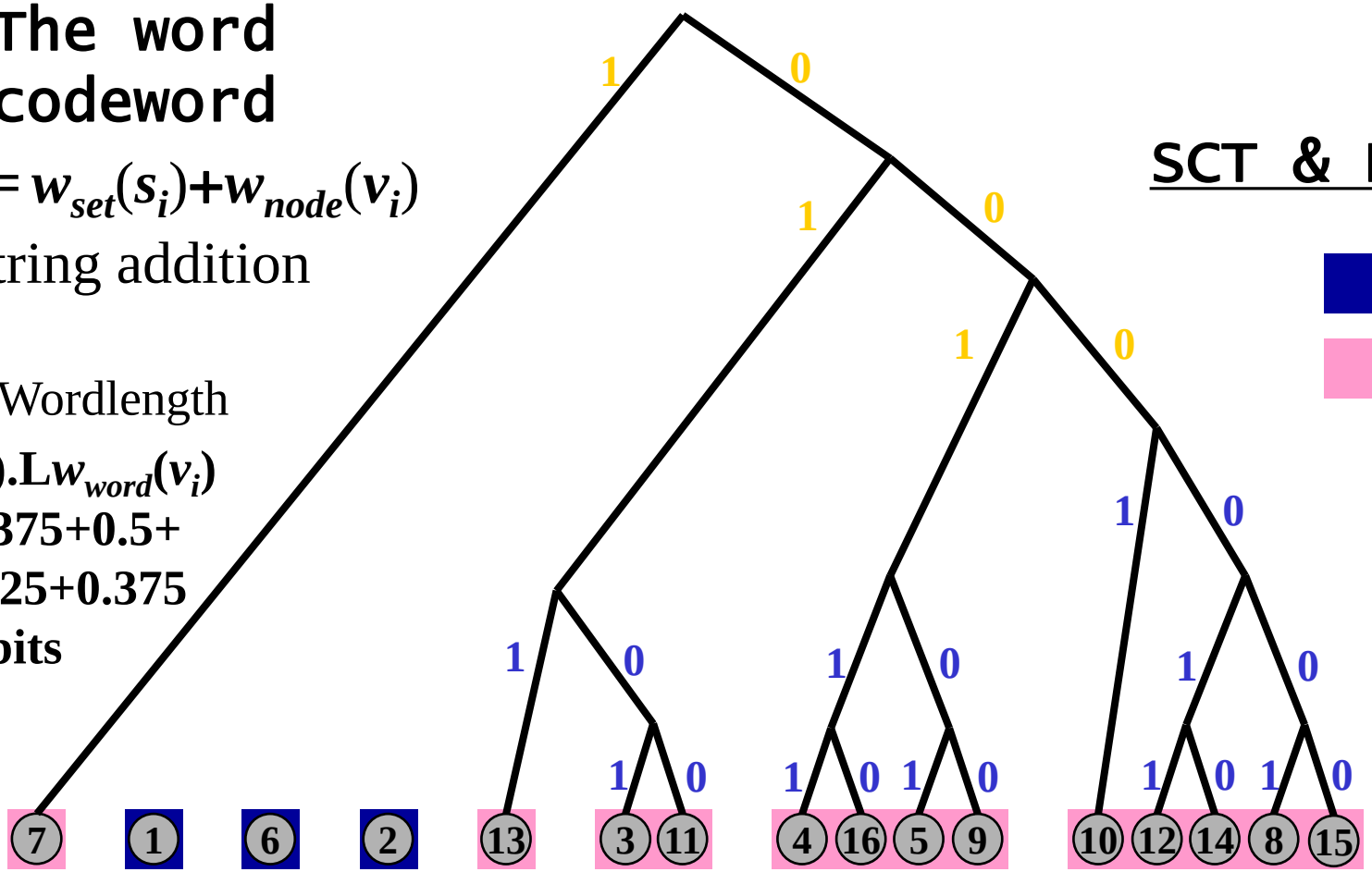
0.625+0.25+0.375

= 2.625 bits

SCT & NCT

{}

≠{}



s_7 s_6 s_5 s_4 s_3 s_2 s_1 s_0

node 7: 1
node 13: 011
node 3: 010 1
node 11: 011 0

node 4: 001 11
node 16: 001 10
node 5: 001 01
node 9: 001 00

node 11: 000 1
node 12: 000 011
node 14: 000 010
node 8: 000 001

node 15: 000 000

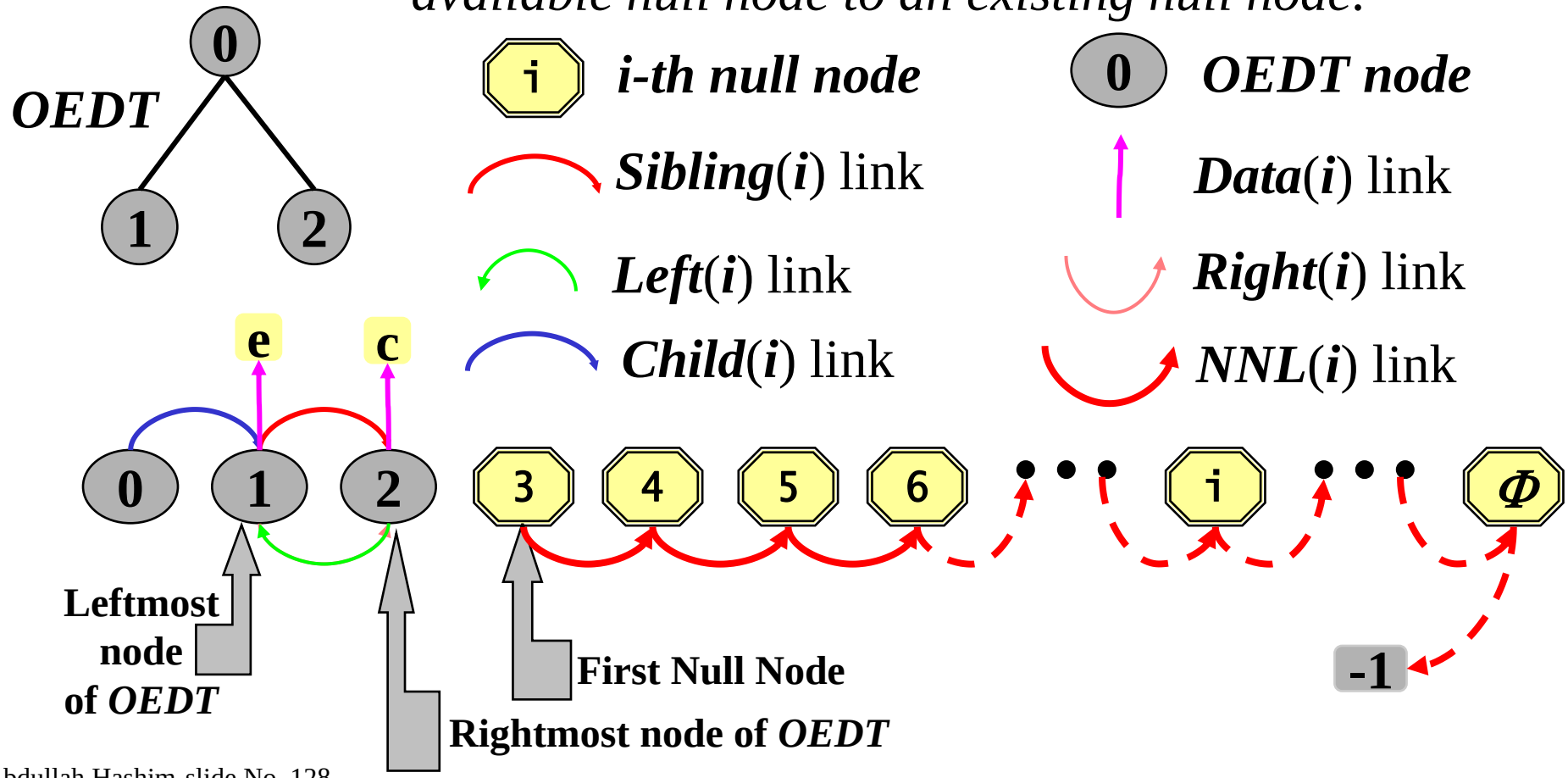
S&M algorithm: The Tree Structure

EDT nodes organisation:

***Null Node*:** *is a node with only null links.*

***Null nodes array*:** *node[i]; i = 0,1,2,.., **Φ**, where **Φ** is the maximum number of words in the **ED**.*

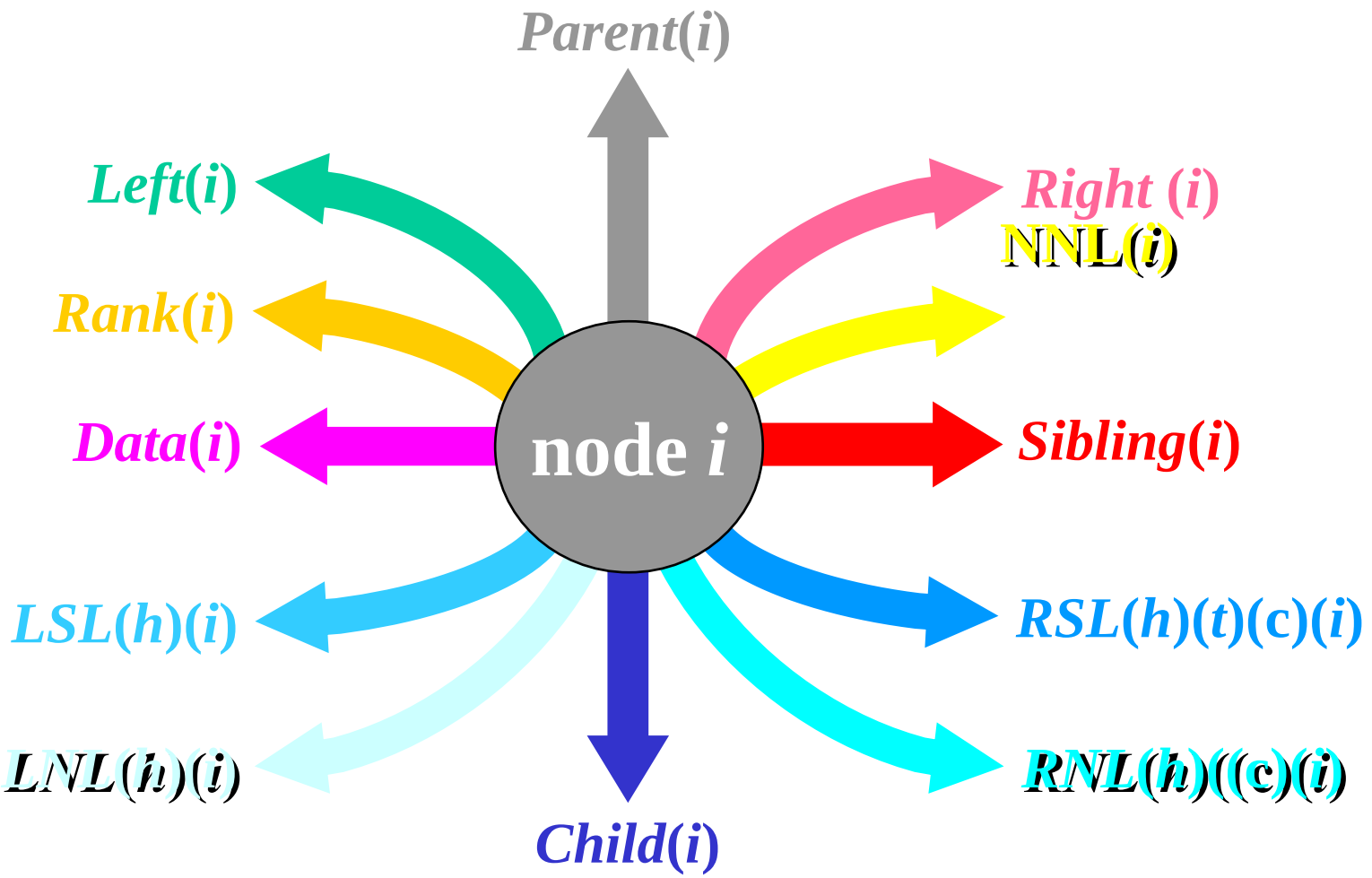
***NNL(i) link*:** *is the **next null node-link** to identify the next available null node to an existing null node.*



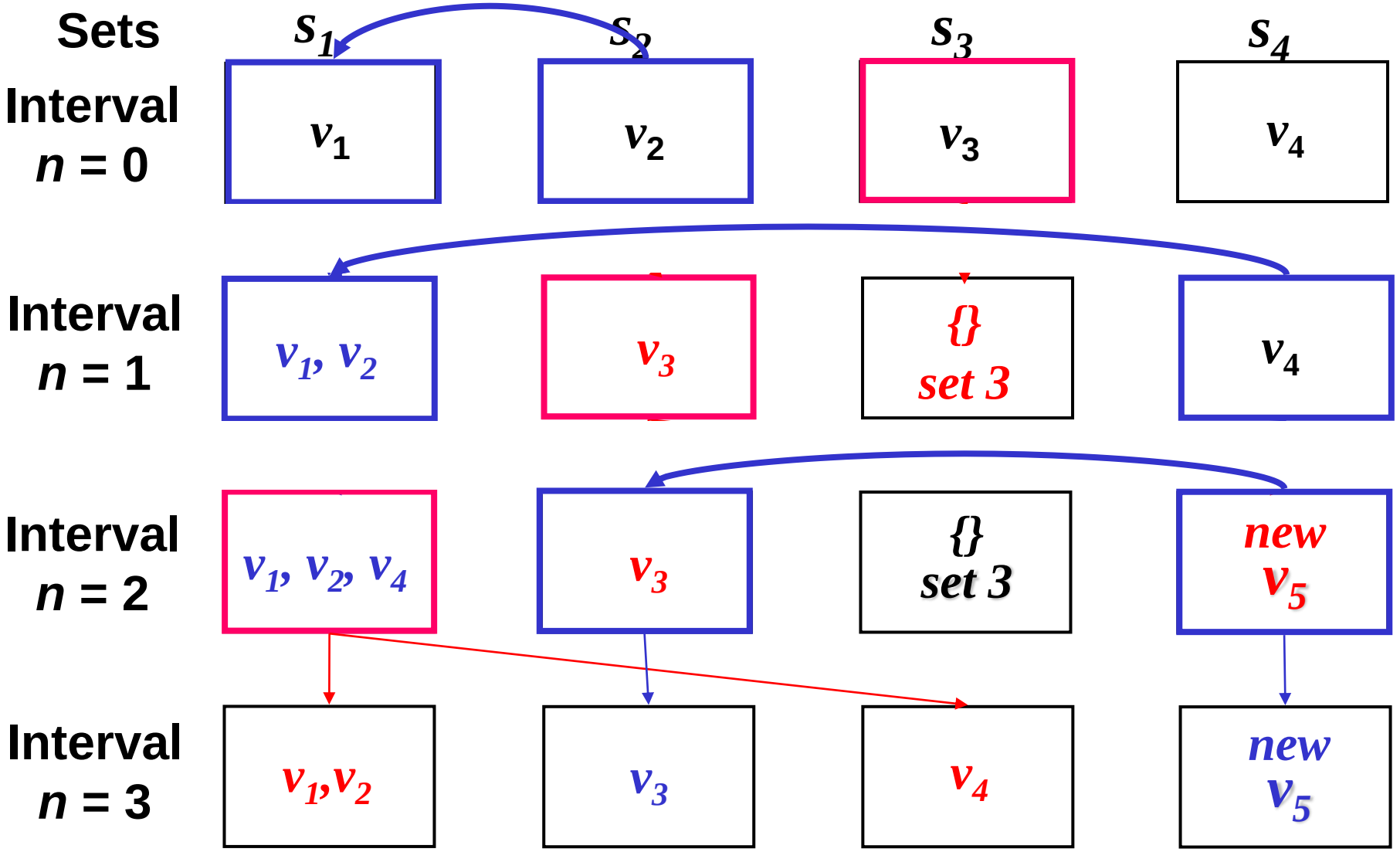
S&M algorithm: The Tree Structure

ODT links:

If a link has a value equal -1, the link is a null link.



S&M algorithm: the dictionary case, Δ=1.



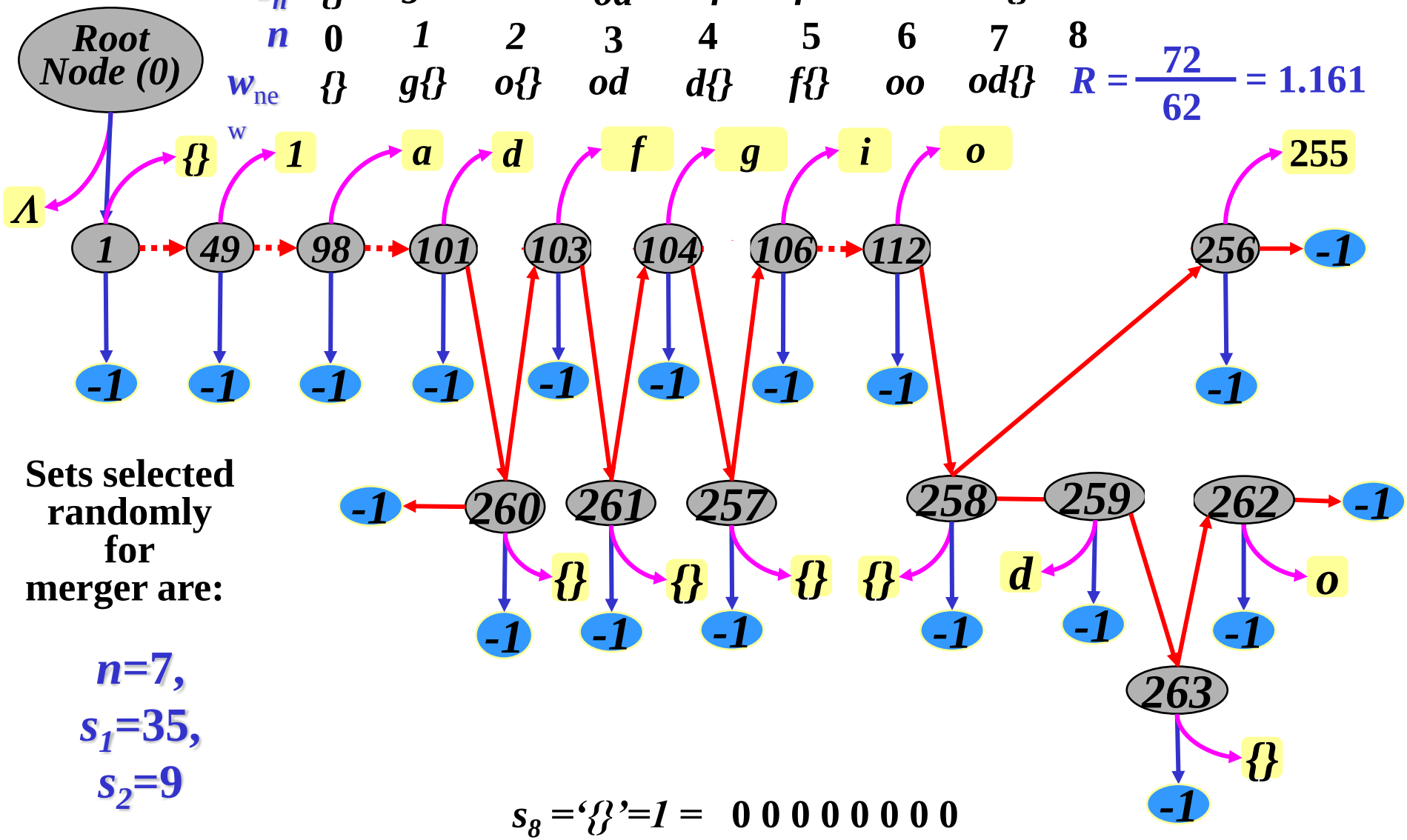
For large values of *n*, set probabilities will converge to values in the range of $2/N \; \delta \; p(s_i) \leq 1/N^{1/2}$

S&M algorithm: the dictionary case, Δ=1.

input string <goodfood{}>

s_n	{}	g	o	o	d	f	o	o	d{} EOF
n	0	1	2	3	4	5	6	7	8
w_{ne}	{}	g{}	o{}	od	d{}	f{}	oo	od{}	

$R = \frac{72}{62} = 1.161$



Sets selected randomly for merger are:

$n=7$,

$s_1=35$,

$s_2=9$

s_8 ='{}'=1 = 0 0 0 0 0 0 0 0

$L(w_{word8})$ *is 8 bits*

S&M algorithm: the dictionary case, Δ=1.

Results of Practical Simulation

Prob(Set1)

$$2/N \leq p(s_i) \leq 1/N^{1/2}$$

$$0.0078 \leq p(s_i) \delta\ 0.067$$

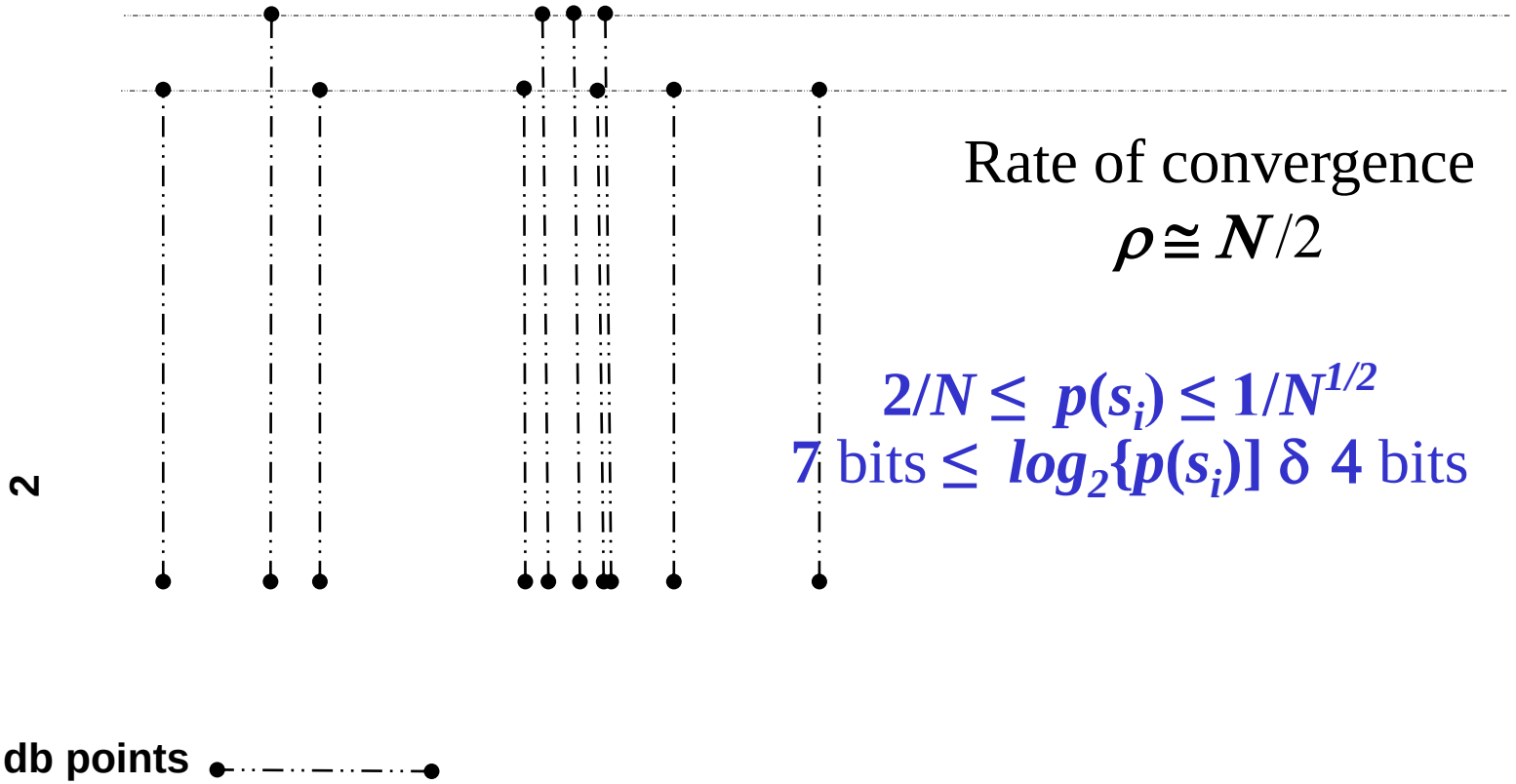
Rate of convergence

$$\rho \cong N/2$$

Abdullah Hashim-slide No. 132

S&M algorithm: the dictionary case, Δ=1.

Results of Practical Simulation



Rate of convergence

$\rho \cong N/2$

$2/N \leq p(s_i) \leq 1/N^{1/2}$

7 bits $\leq log_2\{p(s_i)] \leq 4$ bits

Abdullah Hashim-slide No. 133

Lempel & Ziv algorithm

Lempel and Ziv in their original paper (On the Complexity of Finite sequences. IEEE Trans. IT-22.1 Jan 1976, pp75-81) have shown that if an initial dictionary of single character words of an alphabet is allowed to grow indefinitely by adding a new word at every successive interval in time and the dictionary words are coded by equal length codewords, asymptotically the compression ratio of such a coding scheme will tend to infinity as the dictionary size tends to infinity. Many practical implementations of this basic theory have been reported in the literature of different approaches and complexity. However, all suffer from the same basic inherent feature that once a practical limit is imposed on the dictionary size and its wordlength, the compression ratio will be degraded rapidly. In fact all practical implementations of the dictionary result in an average codeword length even higher than second order entropy. The low compression ratios are due to the practical implementations of the scheme which fill the dictionary mainly with two-character words of low frequency of occurrence instead of long words with high frequency.

Lempel & Ziv algorithm

Let string s_n be an input string from a file of characters in alphabet A over language Θ, at interval n. Match s_n to the longest word w_n in the dictionary given by: $s_n = w_n = <e_1 e_2 \ldots e_i \ldots e_p>$. If e_{p+1} is the unmatched character resulting from the matching process, then the new word added to the dictionary at interval n is $w_{new} = <e_1 e_2 \ldots e_p e_{p+1}>$.

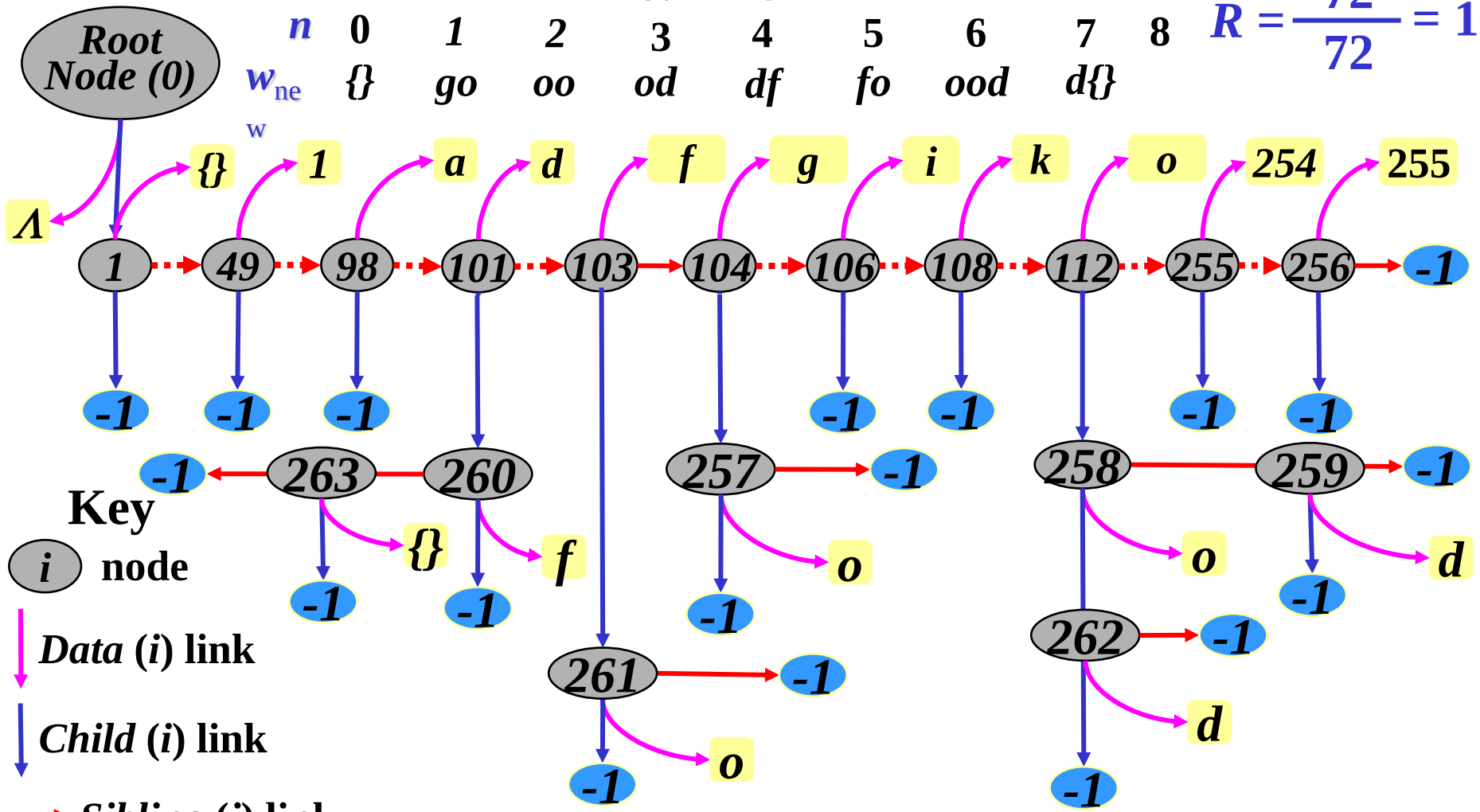
1. At interval n, an input string of source symbols is matched with the longest string in the dictionary (w_n).
2. The matched word w_n is coded by a binary codeword to form the compressed word [$w_{word}(w_n)$].
3. The input string $s_n = w_n$ is appended to the unmatched character resulting from the matching process in 2 above to form the new word (w_{new}).
4. The new word w_{new} is added to the dictionary.
5. The process is repeated starting from step one until EOF.

Lempel & Ziv algorithm

A practical simulation for input string <goodfood{}>

s_n	{}	g	o	o	d	f	o	o	d	{}	EOF
n	0	1	2	3	4	5	6	7	8		
w_{ne}	{}	go	oo	od	df	fo	ood	d{}			

$$R = \frac{72}{72} = 1$$



s_8 ='{}'=1 = 0 0 0 0 0 0 0 0

w_{word8} ='{}'=1 = 0 0 0 0 0 0 0 0 1

Lempel & Ziv algorithm

To study the practical behaviour of the ***L&Z*** algorithm in the context of predicting the properties of the environment Θ, a dictionary of 512 words is used the first 256 words are the single ASCII character of the alphabet ***A***, the second 256 words are the multi-cheater words added to the dictionary over the 256 intervals. The initial probability distribution at interval n=0 of the 256 characters were assigned ten different sets of values, by fixing the initial probability of a leading character (c_1) to one of the ten values in the range $\{0.001 \leq p(c_1) \leq 1\}$, other character initial probabilities where made to be equal to $\{[(1- p(c_1)] / 255\}$. At each interval n, a new word added to the dictionary according to the initial probability distribution of the environment Θ. All words assumed to be equiprobable and the probability of each characters were recomputed. Let at interval n, the words w_1, w_2, … ,and w_k contains z_1, z_2, … , and z_k of the character α, the probability of character α at the n-th interval is given by:

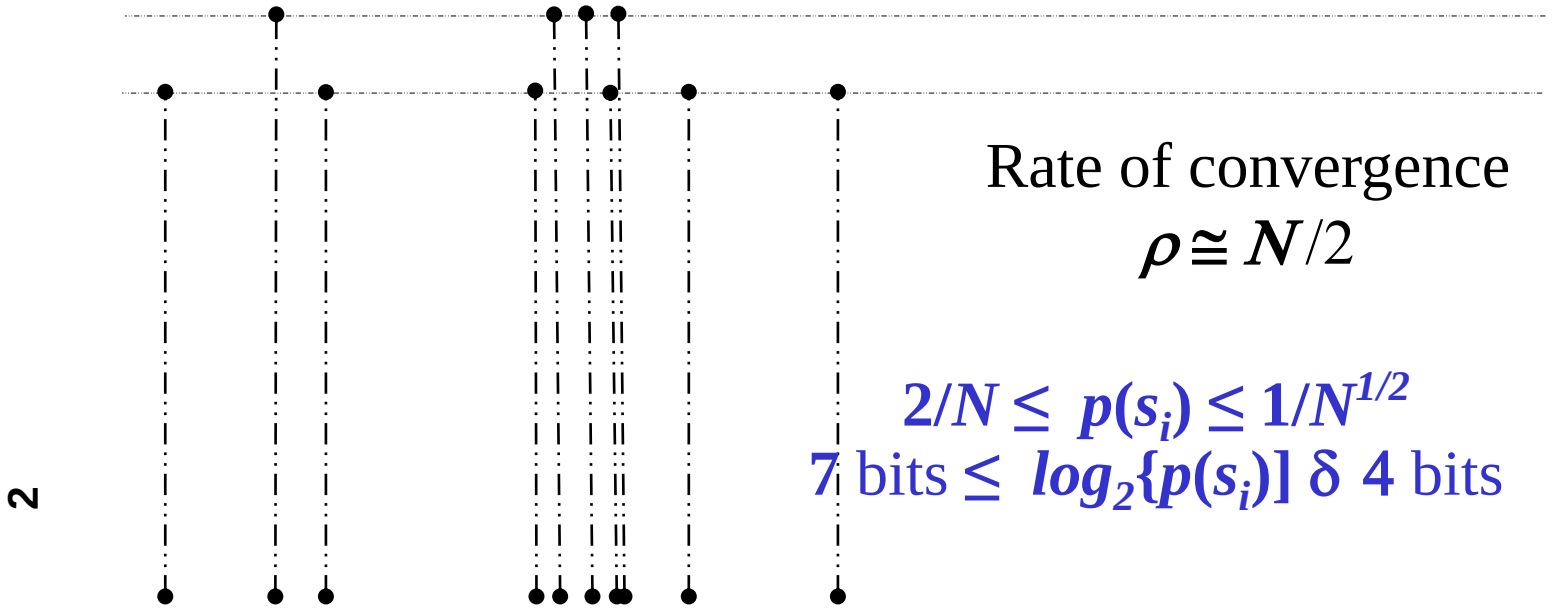
$$p(\alpha) = (1/n) \sum_{i=1}^{k} (z_i / |w_i|)$$

The behaviour of the algorithm is determined by plotting the average $Q(n)$ value over hundred (100) trials, for each of the ten different set of initial probabilities, against the intervals n, where $Q(n)$ is given by the expression:

$$Q(n) = \sum_{i=1}^{256} p^2(s_i) = p^2(s_1) + p^2(s_2) + \ldots + p^2(s_{256})$$

Lempel & Ziv algorithm

2 Results of Practical Simulation



Rate of convergence

$\rho \cong N/2$

$2/N \leq p(s_i) \leq 1/N^{1/2}$

7 bits $\leq log_2\{p(s_i)] \leq 4$ bits

3 db points

S&M algorithm: the dictionary case

Note:

The proposed ***S&M-Dictionary case*** differs fundamentally from that of ***Lempel and Ziv*** scheme in that a new word will not be added to the extended dictionary (at the ***n***-th interval) unless the set is a singleton with Δ satellite sets. This condition ensures that only words of high frequency of occurrence are added to the dictionary and in turn increases the inclusion of longer words of high frequency at subsequent intervals. It is common practice that one of the major conditions of the dictionary pruning process is that the pruned word is of low wordlength and frequency of occurrence.

S&M algorithm: the lossy case

In the lossless ***S&M*** algorithm the word codeword is given by:

$w_{word}(w_i) = w_{set}(s_i) + w_{node}(v_i)$, this is a string sum;

In lossy case the aim is to identify the set containing the compressed node only; nodes within a set are predicted either randomly or in accordance to its frequency of occurrence within the set. High probability singleton and low size sets therefore, are compressed without errors. Sets with size higher than given value, depending on the acceptable level of errors (degradation), will give rise to errors. The magnitude of the errors increases with set size. To keep the compression and decompression processors in synchronisation, the node codeword, in the lossy case, is replaced by a synch codeword $\{w_{synch}(w_i)\}$, the synch codeword is used to identify the control characters and the length of the compressed words (w_i) uniquely.

$$w_{word}(w_i) = w_{set}(w_i) + w_{synch}(w_i);\ L(w_{word}(w_i)) = L(w_{set}(w_i) + L(w_{synch}(w_i))$$

Consider the following compression schemes:

1) The finite case: Sets contains only single character words, if a set contains no control characters then the synch codeword has zero length.

2) The dictionary case: Sets contains multi-characters words a synch word is needed to identify the length of the compressed word (w_i).

S&M algorithm: the lossy case

Synch Coding: in a lossy ***S&MC*** algorithm w_{synch} is used in place of w_{node} for sets of a size $|s_p| > \zeta$. Since the construction rules applied to ***NCT*** and ***SynchCT*** are identical for the processes of splitting, pruning, merging, code bounding and enlarging, the resulting w_{synch} has the same optimal inherent properties of w_{node}. w_{synch} is shorter than w_{node} due to the fact the majority of links in ***SynchCT*** are redundant. The general case of ***S&MC*** algorithm can be a lossless system if $\zeta > \eta$ and can have the maximum compression ratio with maximum degradation when $\zeta = 2$. The level of degradation can be monitored at the encoder and the value of ζ may be varied accordingly during the compression process for optimal operation. The predicator in the ***S&M*** algorithm, requires no prior information about the type, properties and statistics of the input file to predict the maximum likely outcome. This makes the algorithm work for all types of files used in storage, transmission and networked communications.

S&M algorithm: the lossy case

In lossy system the following rules may be followed to keep synchronisation between the compression and decompression processors:

1. the set codewords is constructed in the same way as in the lossless system.
2. the node codewords of sets with size less than a given value say ζ are constructed in the same way as in the lossless system.
3. sets with size higher than ζ node codeword may be considered redundant and the value of the node may be predicted in identical manner in the compression and decompression processors. This will lead to shorter word codewords and higher compression ratio. To keep both processors synchronized we should make sure that this rule does not applied to:
    i. nodes corresponding to control characters
    ii. the word size of the corresponding compressed node should be made available to the decompression processor.

S&M algorithm: the lossy case

Set lead-node*:** If a multiple node set s_i of an ***OEDT is partitioned into subsets of equal depth data-nodes (word size $|w_i|$), then the rightmost node (v_k) in such a subset (say Q_s) is called the ***Set lead-node*** of Q_s.

Synch Coding Tree (*SynchCT*): A *SynchCT* is used in a lossy ***S&M*** algorithm in place of ***NCT*** for sets in the ***OEDT*** of size $|s_i| > \zeta$. Synch codeword $w_{synch}(s_i)$ is used to identify only the lead and control-nodes within a set s_i. ***SynchCT*** has the same structure as ***SCT*** and ***NCT***, The synch-links denoted by (***SyL***), left and right links correspond to left and right synch-links. The ***i***-th synch-link ($SyL(h)_i$) of set (s_i) is said to be of height ***h***, where $i = 0, 1, \ldots, m-1$; $m = \lfloor |s_i|/2^h \rfloor$. $SyL(h)_i$ surrounds the nodes of $SyL(h-1)_{2i}$ and $SyL(h-1)_{2i+1}$. If $SyL(h-1)_{2i+1}$ is a null-link then $SyL(h)_i$ surrounds only the nodes of $SyL(h-1)_{2i}$ and (in this case) $SyL(h-1)_{2i}$ becomes redundant. The ***i***-th right synch-link $\{RSyL(h)_i\}$ and the left synch-link $\{LSyL(h)_i\}$ surrounds the same nodes. Synch-links differ from node-links in that they have two types. If $SL(0)$ surrounds a single word or control character it said to be of type **1**,

S&M algorithm: the lossy case

all other ***SyL*(0)** are said to be of ***type* 0**. Synch-links have a single binary digit code, the ***i*-th** synch-link is coded by binary zero if ***i*** is an even integer and binary one if ***i*** is an odd integer. The zero height synch-link has the least significant digit and the largest height synch-link has the most significant digit in the synch codeword $\boldsymbol{w_{synch}(s_i)}$. Type **1** synch-links are coded by a binary digit, while type **0** synch-links are not coded as they are redundant and carry no information. If ***SyL*(*h*+1)** surrounds all nodes of two adjacent ***SyL*(*h*)** and if one of the two ***SyL*(*h*)** is of type **0**, then the ***SyL*(*h*)** type **1** becomes redundant. ***SyL*(*h*+1)** type **1** surrounds at least one ***SyL*(*h*)** of type **1**. ***SyL*(*h*+1)** type **0** surrounds only type **0** ***SyL*(*h*)** synch-links. A link surrounding all nodes of a set $\boldsymbol{s_i}$ and its corresponding synch-link ($\boldsymbol{SyL(d_{SynchCT})}$, where $\boldsymbol{d_{SynchCT}}$ is the binary depth of ***SynchCT*** of set $\boldsymbol{s_i}$) is redundant. $\boldsymbol{SL(d_{SynchCT})}$ is known as the root synch-link. Set $\boldsymbol{s_i}$ has the same links in both synch and node coding trees, however many of the links in ***SynchCT*** are redundant while their corresponding links in the ***NCT*** are not redundant. This feature of ***SynchCT*** leads to a low synch codeword

S&M algorithm: the lossy case

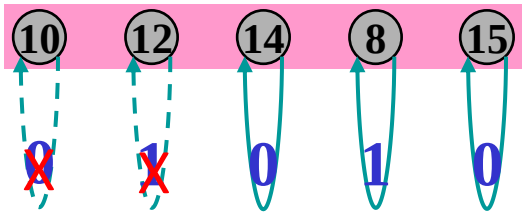
average-length $L_{average}[w_{synch}(s_i)]$ than that of the node codeword $L_{average}[w_{node}(s_i)]$.

***Synch-link of height zero SyL*(0):** is a link surrounding a single node. All zero height synch-links are of type **0** except synch-links surrounding a lead or a control-node are of type **1**. Consider set W_0 of the previous examples: $s_0 = \{\nu_{10}, \nu_{12}, \nu_{14}, \nu_8, \nu_{15}\}$; ν_{15} is a control-node; ν_8 and ν_{14} are lead-nodes; $L(s_8) = L(s_{12}) = L(s_{10}) = 3$ and $L(s_{14}) = 1$. The fourth synch-link $SyL(0)_4$ and the third synch-link $SyL(0)_3$ are of type **0**, surrounding node ν_{10} and ν_{12} respectively, while $SyL(0)_2$, $SyL(0)_1$ and $SyL(0)_0$ are of type **1** surrounding node ν_{14}, ν_8 and ν_{15} respectively.

***LSyL*(0)** type **0**

***LSyL*(0)** type **1**

***SynchCT* links of s_0, $h = 0$.**



~~1~~ redundant digit

r node ***r***

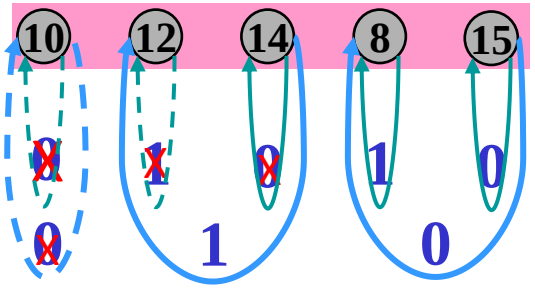
S&M algorithm: the lossy case

Left synch-link of height One LSyL*(1) :** is a link connecting the rightmost node of the (2i***)-th left-synch-link {$LSyL(0)_{2i}$} to the leftmost node of the (***2i+1***)-th left-synch-link {$LSL(0)_{2i+1}$}. If $LSyL(0)_{2i+1}$ is a null-link then $LSyL(1)_i$ surrounds the single node of $LSL(0)_{2i}$ and (in this case) $LSL(0)_{2i}$ becomes redundant. $RSL(1)_i$ and $LSL(1)_i$ surrounds the same nodes. If one of the two ***SyL*(0)** is of type **0**, then the ***SyL*(0)** type **1** becomes redundant. ***SyL*(1)** type **1** surrounds at least one ***SyL*(0)** of type **1**. ***SyL*(1)** type **0** surrounds only type **0** ***SL*(0)** synch-links.

***LSyL*(1) type 0**

***LSyL*(1) type 1**

***SynchCT* links of s_0**
***h* = 0 and 1**

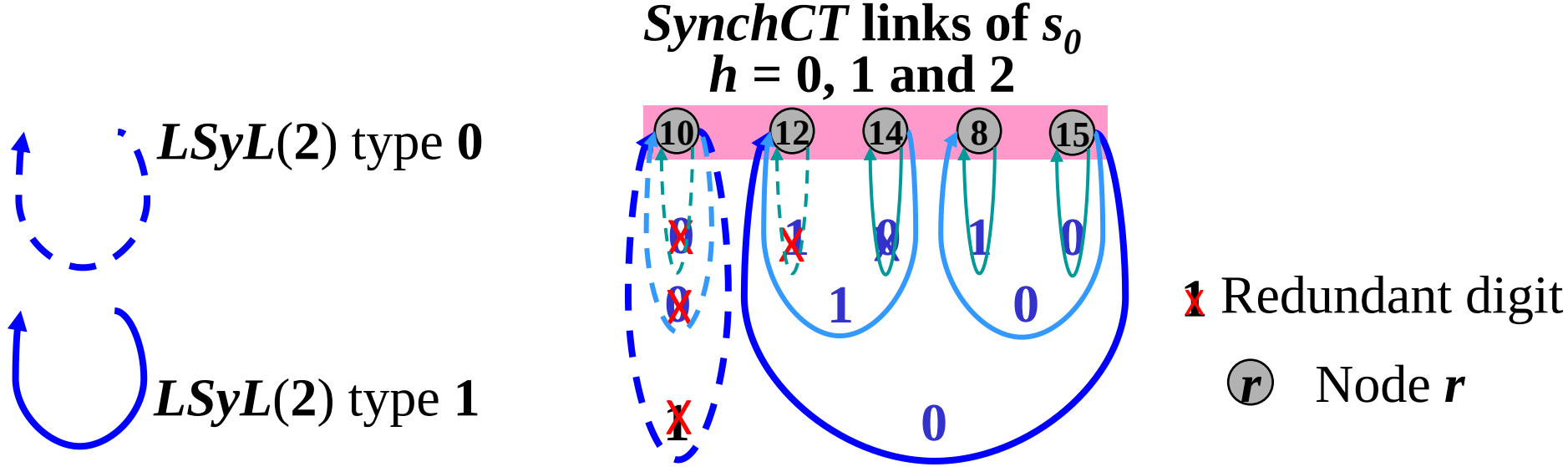


~~1~~ redundant digit

r node ***r***

S&M algorithm: the lossy case

Left synch-link of height two LSyL*(2):** is a link connecting the rightmost node of the (2i***)-th left-synch-link {$LSL(1)_{2i}$} to the leftmost node of the (***2i+1***)-th left-synch-link {$LSL(1)_{2i+1}$}. If $LSyL(1)_{2i+1}$ is a null-link then $LSyL(2)_i$ surrounds all nodes of $LSyL(1)_{2i}$ and (in this case) $LSyL(1)_{2i}$ becomes redundant. $RSyL(2)_i$ and $LSyL(2)_i$ surrounds the same nodes. If one of the two ***SyL*(1)** is of type **0**, then the ***SyL*(1)** type **1** becomes redundant. ***SyL*(2)** type **1** surrounds at least one ***SL*(1)** of type **1**. ***SyL*(2)** type **0** surrounds only type **0** ***SL*(1)** synch-links.



S&M algorithm: the lossy case

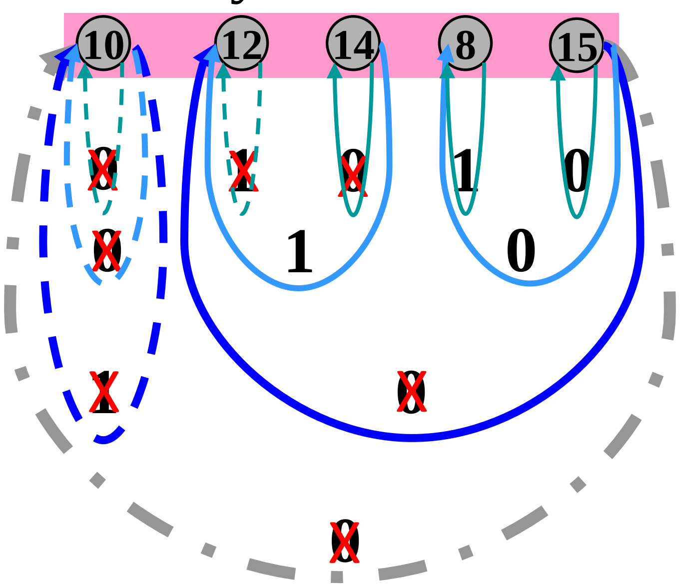
Left synch-link of height three {***LSyL***(3)}**:** is a link connecting the rightmost node of the (***2i***)-th left-synch-link {***LSyL***$(2)_{2i}$} to the leftmost node in the (***2i+1***)-th left-synch-link {***LSyL***$(2)_{2i+1}$}. If ***LSyL***$(2)_{2i+1}$ is a null-link then ***LSyL***$(3)_i$ surrounds a single node of ***LSyL***$(2)_{2i}$ and (in this case) ***LSL***$(2)_{2i}$ becomes redundant. If one of the two ***SyL***(2) is of type **0**, then the ***SyL***(2) type **1** becomes redundant. ***SyL***(3) type **1** surrounds at least one ***SyL***(2) of type **1**. ***SyL***(3) type **0** surrounds only type **0** ***SyL***(2) synch-links. ***RSyL***$(3)_i$ and ***LSyL***$(3)_i$ surrounds the same nodes. If a link surrounds all nodes of s_i it is a redundant link, known as the root synch-link.

Synch Codewords

node 15 codeword = **00**

nodes 8,12 & 10 = **01**

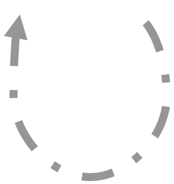
node 14 codeword = **1**



Synch-links of set s_0

1 Redundant digit

r node ***r***



LSyL(3) type **0**

S&M algorithm: the lossy case

Notes on Synch Coding Tree *SynchCT* :

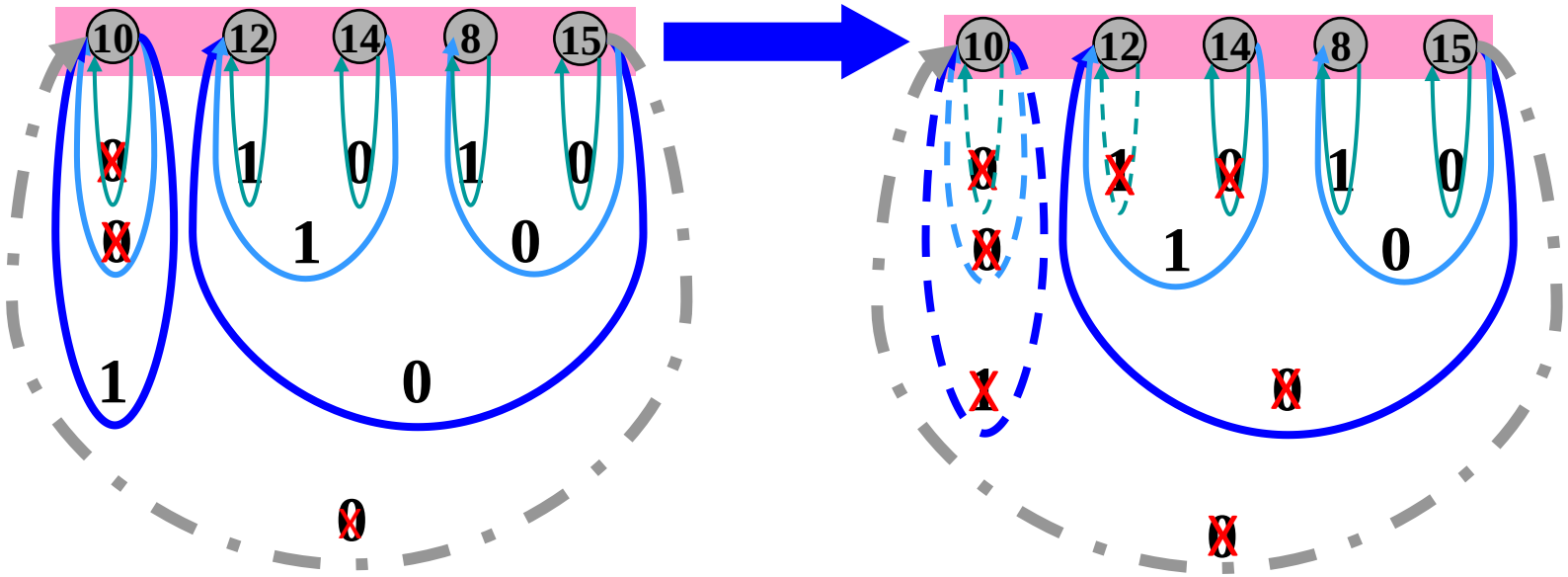
1. In ***S&M*** lossy algorithm, node codeword $w_{node}(s_i)$ is used for set size $|s_i| \leq \zeta$ and synch codeword $w_{synch}(s_i)$ for set size $|s_p| > \zeta$.

s_0 node codewords

$w_{node}(v_{15}) = 000$; $w_{node}(v_8) = 001$
$w_{node}(v_{14}) = 010$; $w_{node}(v_{12}) = 011$
and $w_{node}(v_{10}) = 1$

s_0 synch codewords

$w_{syncg}(v_{15}) = 11$; $w_{synche}(v_8) = 11$
$w_{synch}(v_{14}) = 1$; $w_{synch}(v_{12}) = \{\}$
and $w_{synche}(v_{10}) = \{\}$



S&M algorithm: the lossy case

Notes on Synch Coding Tree SynchCT :

2. s_i has the same links in both node and synch coding trees, however many of the links in ***SynchCT*** are redundant.

3. The *i*-th synch-link ($SL(h)_i$) of a subset of nodes in a set (s_p) is said to be of height ***h***, where $i=0, 1, \ldots, m-1$; $m = \lfloor |s_p|/2^h \rfloor$. $SL(h)_i$ surrounds the elements of synch-link $SL(h\text{-}1)_{2i}$ and $SL(h\text{-}1)_{2i+1}$. If $SL(h\text{-}1)_{2i+1}$ is a null-link then $SL(h)_i$ surrounds only the nodes of $SL(h\text{-}1)_{2i}$ and (in this case) $SL(h\text{-}1)_{2i}$ becomes redundant.

4 Links pointing from left to right are called right links, and links pointing from right to left are called left links.

5 $LSL(h)_i$ and the $RSL(h)_i$ surrounds the same nodes.

6 A synch-link has a single binary digit, the ***i***-th synch-link is coded by binary zero if ***i*** is an even integer and binary one if ***i*** is an odd integer. The zero height synch-link has the least significant digit

S&M algorithm: the lossy case

Notes on Synch Coding Tree *SynchCT* (continued):

and the largest height synch-link has the most significant digit in the synch codeword $\boldsymbol{w_{synch}(s_i)}$. A synch-link carrying no data is redundant and has no codeword.

7. A link surrounding all nodes of set $\boldsymbol{s_i}$ and its corresponding synch-link ($\boldsymbol{SL(d_{SynchCT})}$, where $\boldsymbol{d_{SynchCT}}$ is the binary depth of ***SynchCT*** of set $\boldsymbol{s_p}$) are redundant. $\boldsymbol{SL(d_{SynchCT})}$ is known as the root synch-link.

8. $\boldsymbol{SL(0)}$ surrounds a single node in set $\boldsymbol{s_i}$.

9. Synch-links are of two types. All zero height synch-links are of ***type* 0** except synch-links surrounding a lead or a control-node are of ***type* 1**. A type one synch-link is coded by a single binary digit, while type zero is a redundant synch-link.

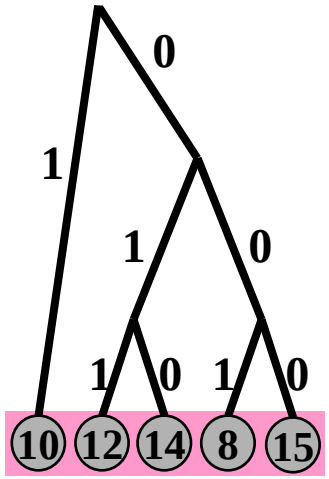
S&M algorithm: the lossy case

Notes on Synch Coding Tree *SynchCT* (continued):

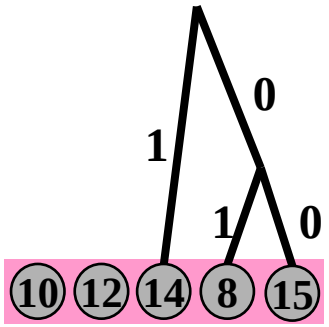
10. A ***SL*(*h*+1)** surrounds all nodes of two adjacent ***SL*(*h*)** synch-links. If one of the two ***SL*(*h*)** synch-links is of type 0, then the ***SL*(*h*)** type 1 becomes redundant. ***SL*(*h*+1)** type 1 surrounds at least one ***SL*(*h*)** of type 1. ***SL*(*h*+1)** type 0 surrounds only type 0 ***SL*(*h*)** synch-links.

11. For a given set s_i the links in both synch and node coding trees are identical, they differ only in coding. Many of the synch-links are redundant while their corresponding node-links are not. Sync codeword $w_{synch}(v_i)$ is shorter than the corresponding node codeword $w_{node}(v_i)$.

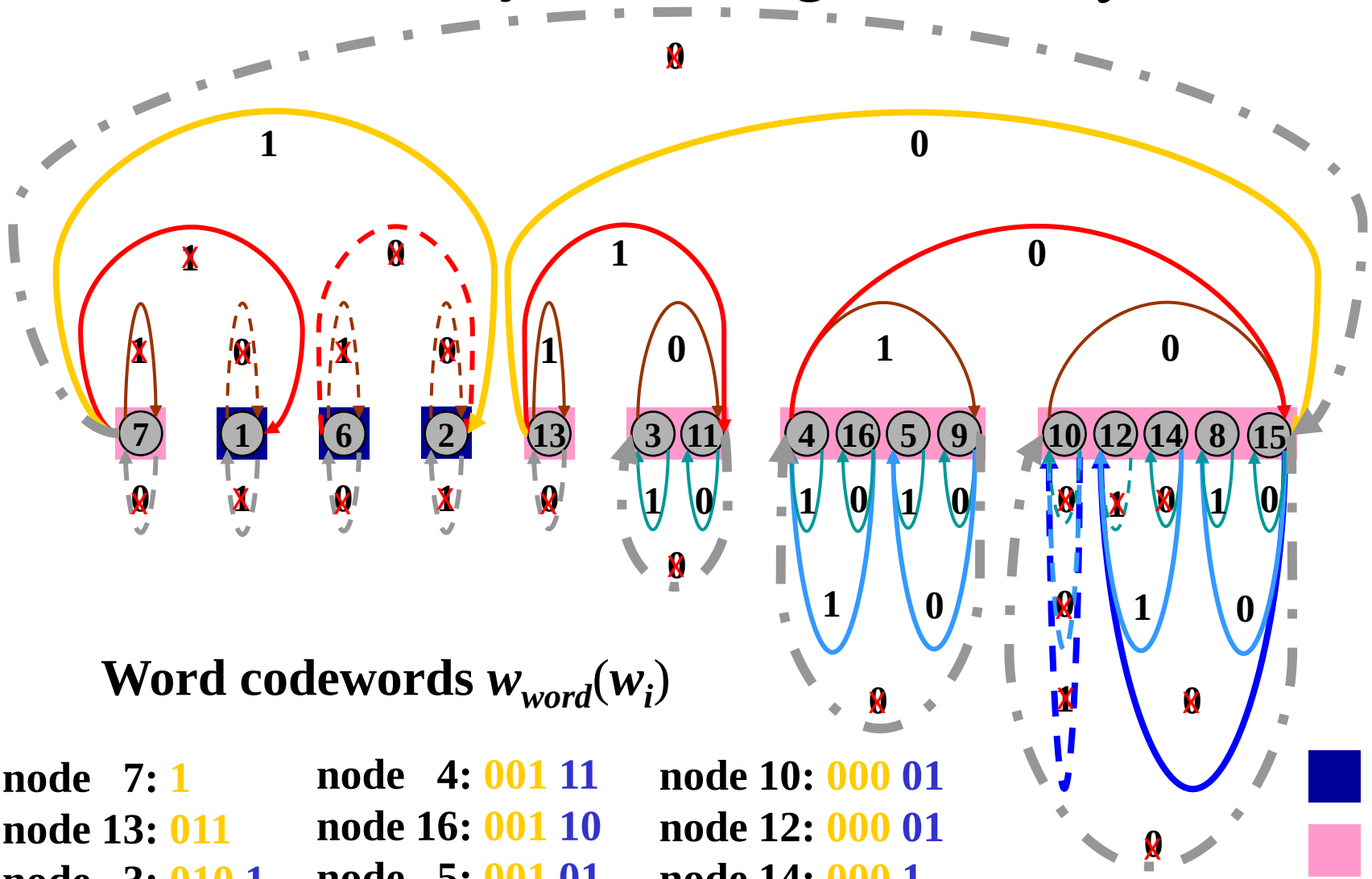
Set s_0 Node Coding Tree



Set s_0 Sync Coding Tree



Set, Node and Synch Coding Trees for ζ = 4



Word codewords $w_{word}(w_i)$

node 7: 1
node 13: 011
node 3: 010 1
node 01: 010 0

node 4: 001 11
node 16: 001 10
node 5: 001 01
node 9: 001 00

node 10: 000 01
node 12: 000 01
node 14: 000 1
node 8: 000 01

node 15: 000 00

The word codeword

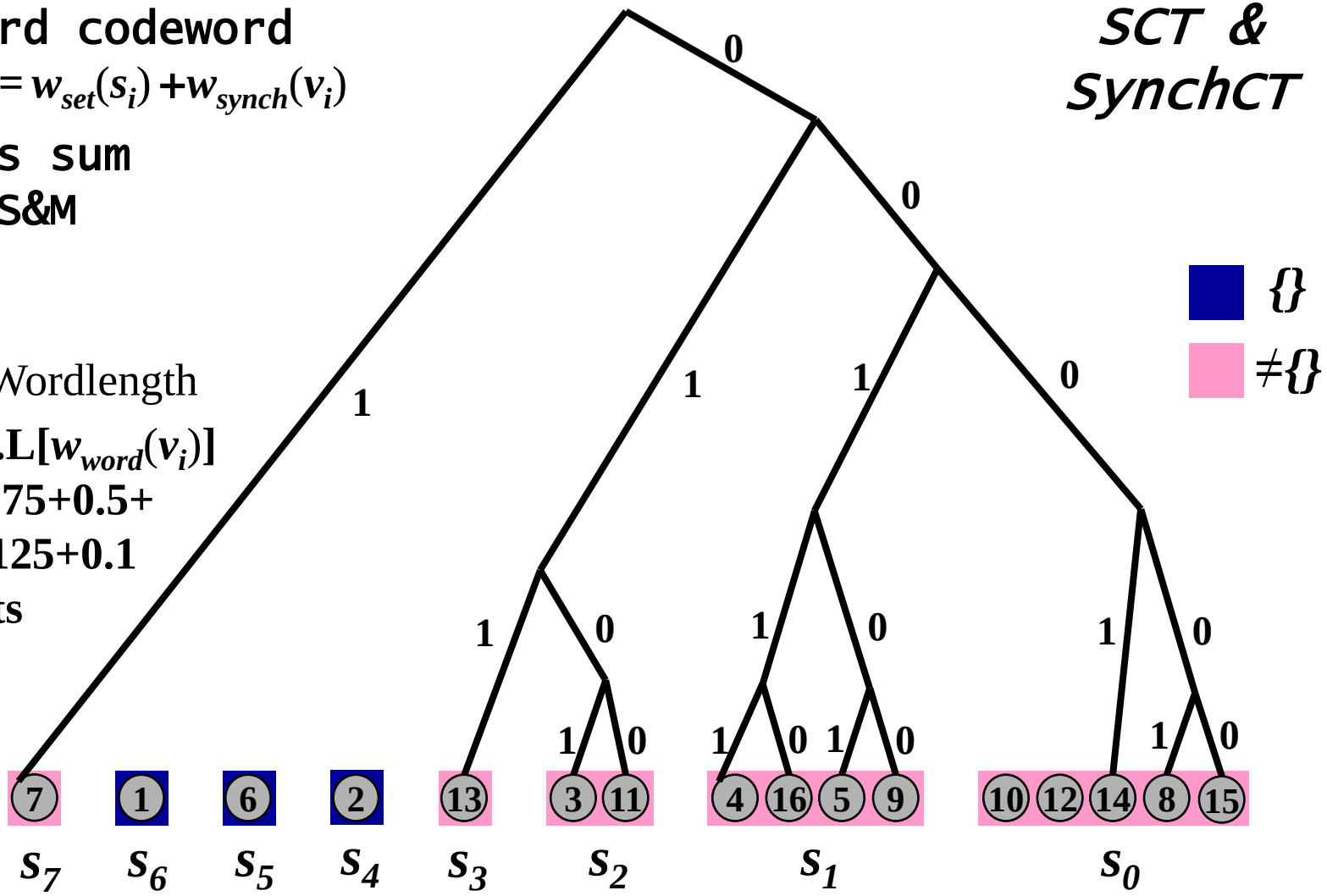
$w_{word}(v_i) = w_{set}(s_i) + w_{synch}(v_i)$

Strings sum

Lossy S&M

$\zeta = 4$

SCT & SynchCT



{}

≠{}

Average Wordlength

$= \Sigma P(v_i).L[w_{word}(v_i)]$

= 0.5+0.375+0.5+

0.625+0.125+0.1

= 2.25 bits

node 7: 1

node 13: 011

node 3: 010 1

node 11: 010 0

node 4: 001 11

node 16: 001 10

node 5: 001 01

node 9: 001 00

node 00: 000 01

node 12: 000 01

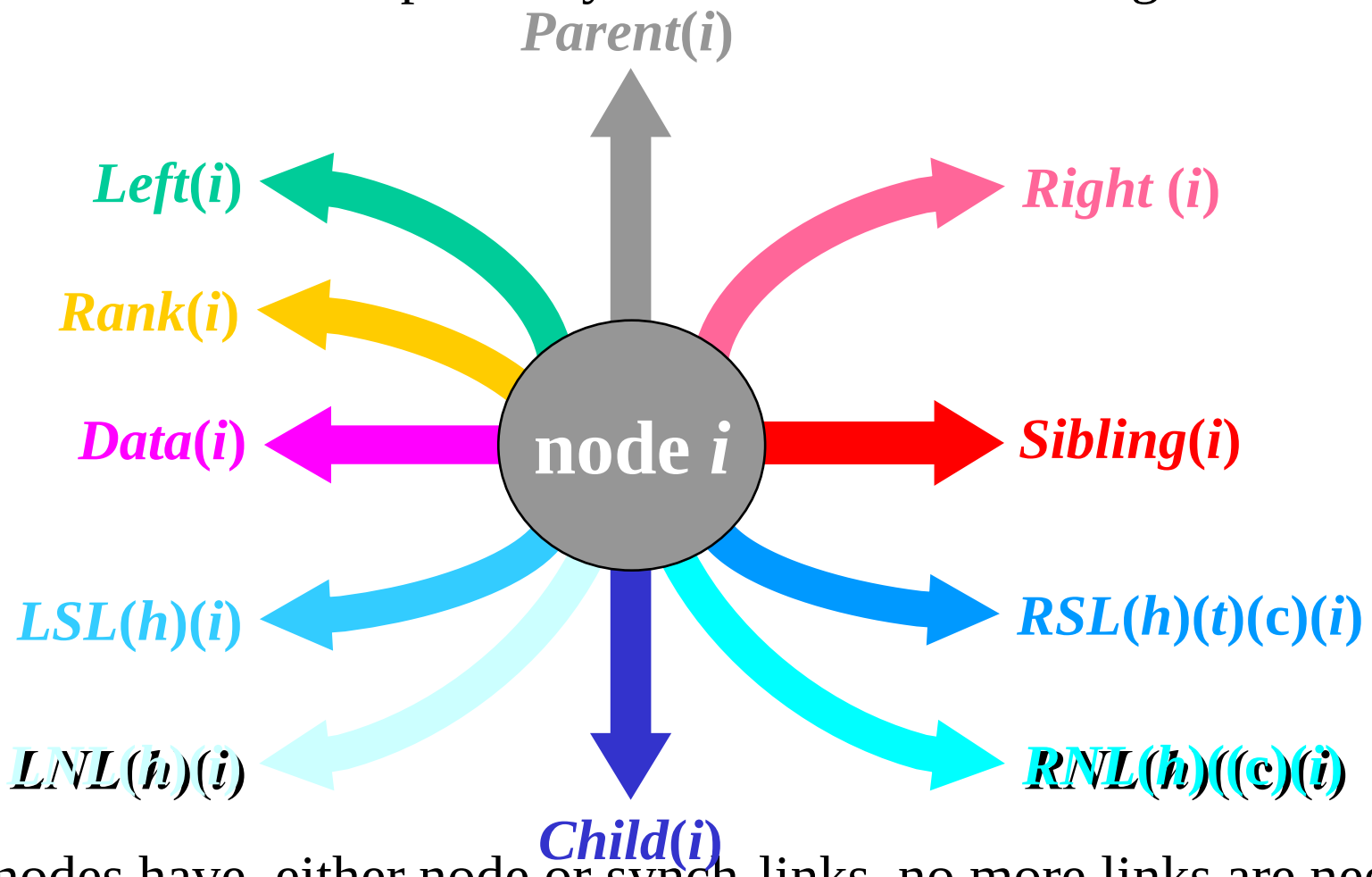
node 14: 000 1

node 8: 000 01

node 15: 000 00

S&M algorithm: the lossy case

ODT* links**: The coding digit (c***) may take three values, 0, 1 for binary zero and one respectively and 2 for redundant digit.



Since nodes have either node or synch-links, no more links are needed.
Since right and left synch-links have the same type(t) and coding digit (c), left node-link code digit is ignored.

S&M algorithm: practical implementation

***Code Bounding*:** is a process of keeping Lw_{node} and Lw_{synch} within a given bound.

In the ***S&M*** algorithm node codewords w_{node} and synch codewords w_{synch} may have lengths between **0** and $(\Phi - N)$, where Φ is the maximum number of allowed words in an ***ED***, and N is the number of sets in an ***OEDT***.

To ensure robustness a limit is usually fixed for both node and synch codeword lengths such that:

$$\lfloor \log_2 \eta \rfloor \leq \text{limit} \leq (\Phi - N),$$

where $\eta = \Phi - N + 1$ is the maximum number of nodes in a set.

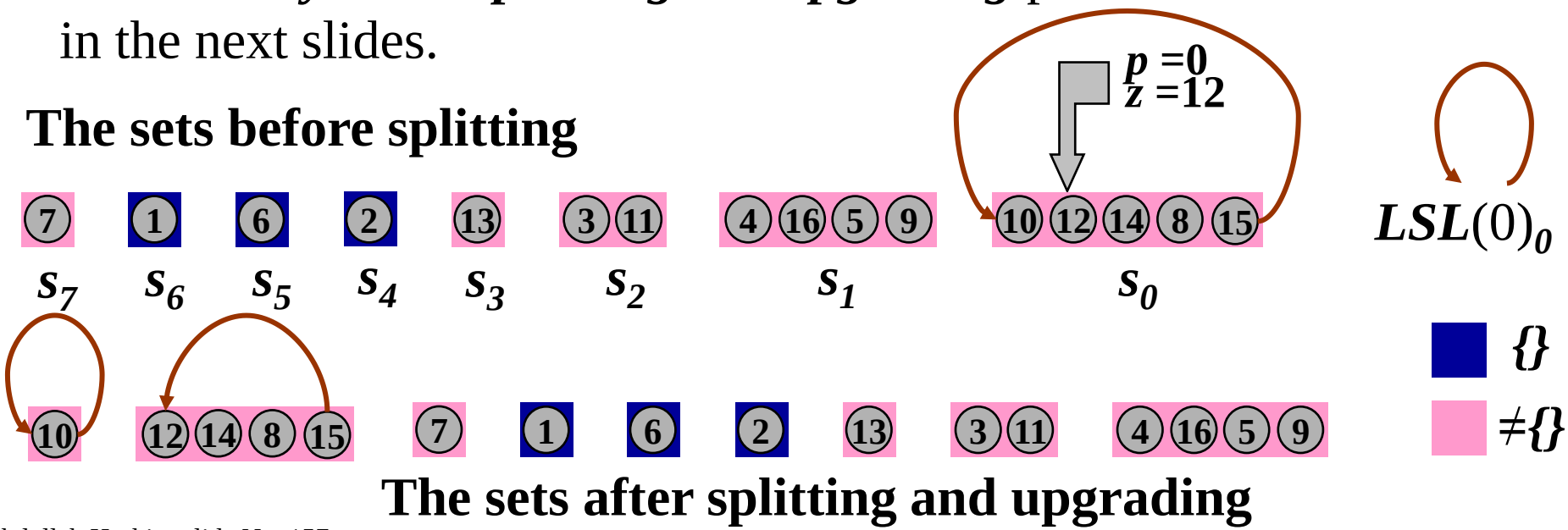
To achieve a robust coding the value of the bound (Ψ) on the links height is usually set to, or near to the value $\lfloor \log_2 \eta \rfloor + 1$.

This condition for coding robustness is achieved by setting a limit Ψ on the root synch-link and root node-link heights (h_{max}), (i.e. $h_{max} < \Psi$, the maximum node and synch-links height).

Definitions:

Splitting*:** The process of splitting consists of the following steps: Let the node corresponding to s_{in} be node v_z located in set S_p of an ***OEDT.

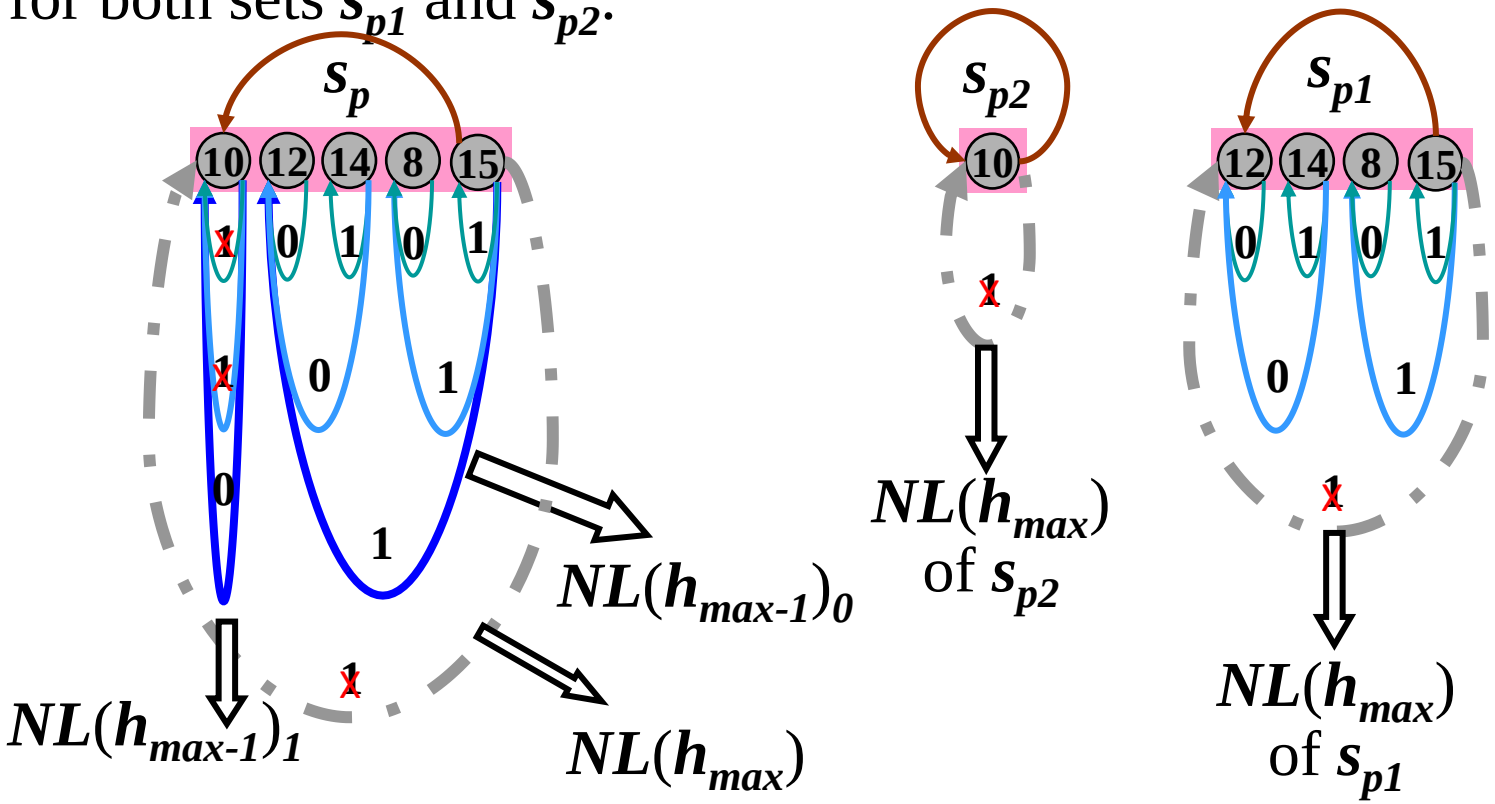
1. If $|s_p| > 1$ then, for a ***NCT*** split s_p into two disjoint equal probability sets s_{p1} and s_{p2} ; s_{p1} containing all nodes in the node-link $NL(h_{max}\text{-}1)_0$ and s_{p2} containing all nodes in the node-link $NL(h_{max}\text{-}1)_1$ of s_p. Similarly for ***SynchCT***, s_{p1} containing all nodes in $SL(h_{max}\text{-}1)_0$ and s_{p2} containing all nodes in $SL(h_{max}\text{-}1)_1$ of set s_p. Prune the ***NCT*** or ***SynchCT*** and up grade s_{p1} and s_{p2} . ***NCT*** and ***SynchCT pruning*** and ***upgrading*** procedures are defined in the next slides.



Definitions:

***NCT pruning*:** The process of pruning consists of the following steps: Suppose set s_p has been split into two sets s_{p1} and s_{p2}.

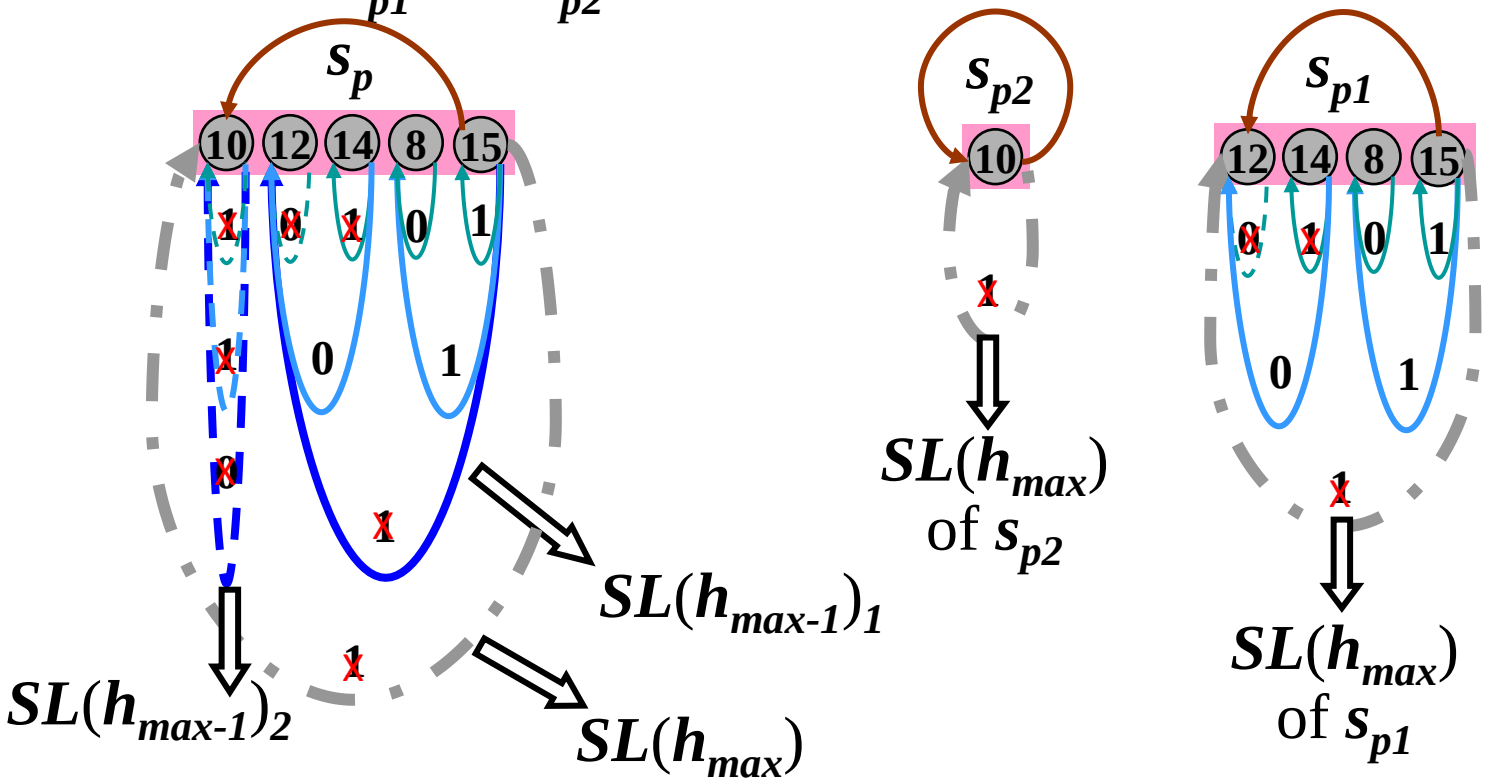
1. Delete $NL(h_{max})$ of s_p .
2. $NL(h_{max}\text{-}1)_0$ of s_p becomes equal to the redundant root node-link $NL(h_{max})$ of s_{p1}. Likewise for $NL(h_{max}\text{-}1)_1$ and s_{p2}.
3. Delete all duplicate root node-links except the innermost root-link for both sets s_{p1} and s_{p2}.



Definitions:

***SynchCT pruning*:** The process of pruning consists of the following steps: Suppose set s_p has been split into two sets s_{p1} and s_{p2}.

1. Delete $SL(h_{max})$ of s_p .
2. $SL(h_{max}\text{-}1)_0$ of s_p becomes equal to the redundant root synch-link $SL(h_{max})$ of s_{p1}. Likewise for $SL(h_{max}\text{-}1)_1$ and s_{p2}.
3. Delete all duplicate root synch-links except the innermost root-link for both sets s_{p1} and s_{p2}.



***Splitting* (continued):**

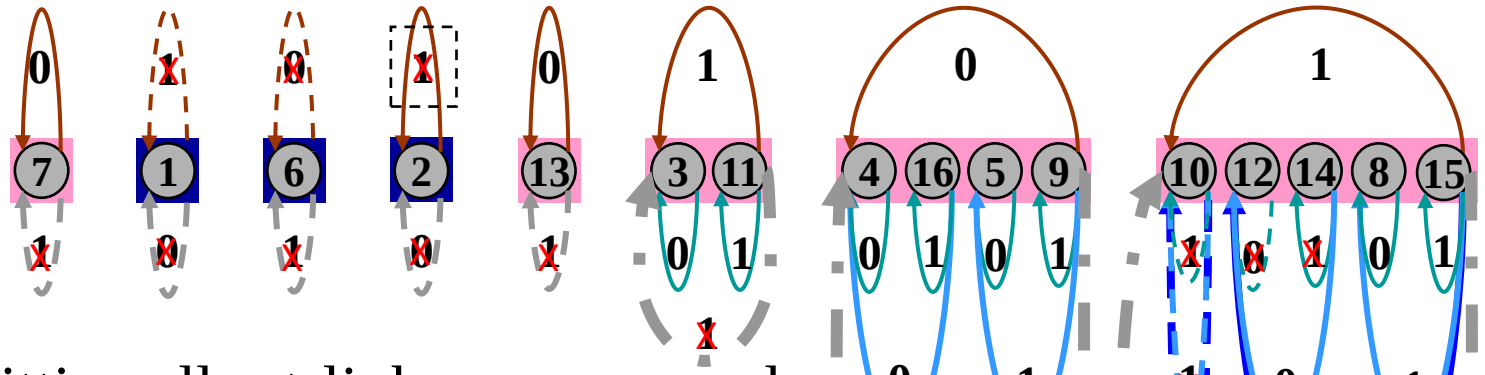
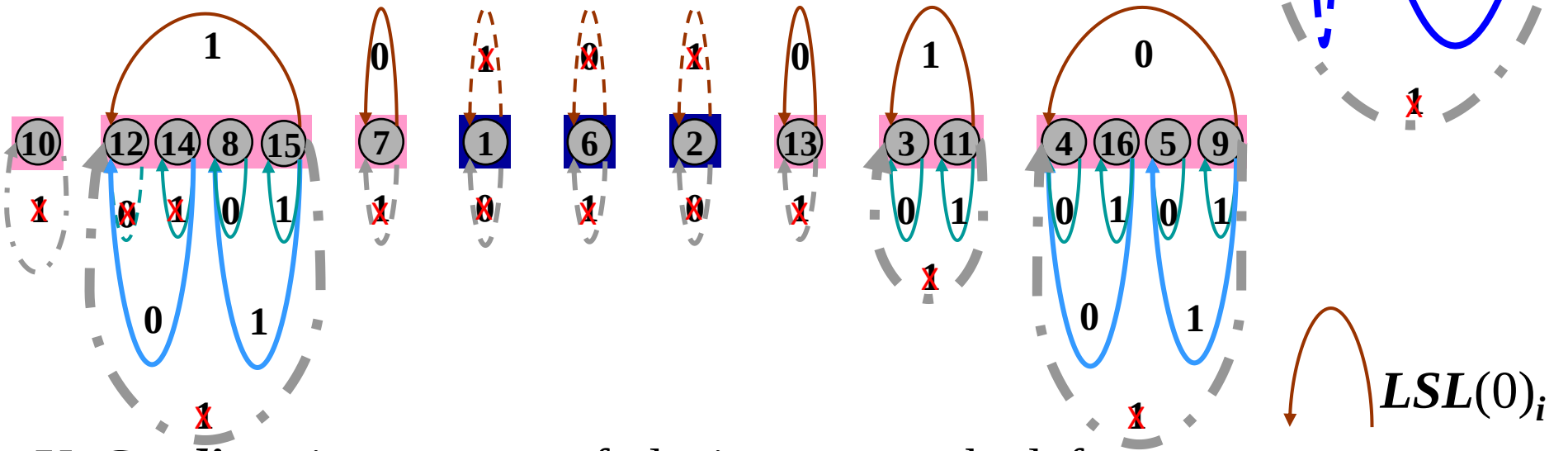


Figure: 39

1.1. Before splitting all set-links are removed except of left set-links of height zero. Left set-link is used to identify the new two sets.



UpGrading*:** is a process of placing a set at the leftmost position of the ***OEDT. The new two sets generated after the splitting process are upranked (i.e. Placed at the left most position of the ***OEDT***).

***Splitting* (continued)**:

2. If $|s_p| = 1$ and $FBL_i < N/2$, then perform the following steps:

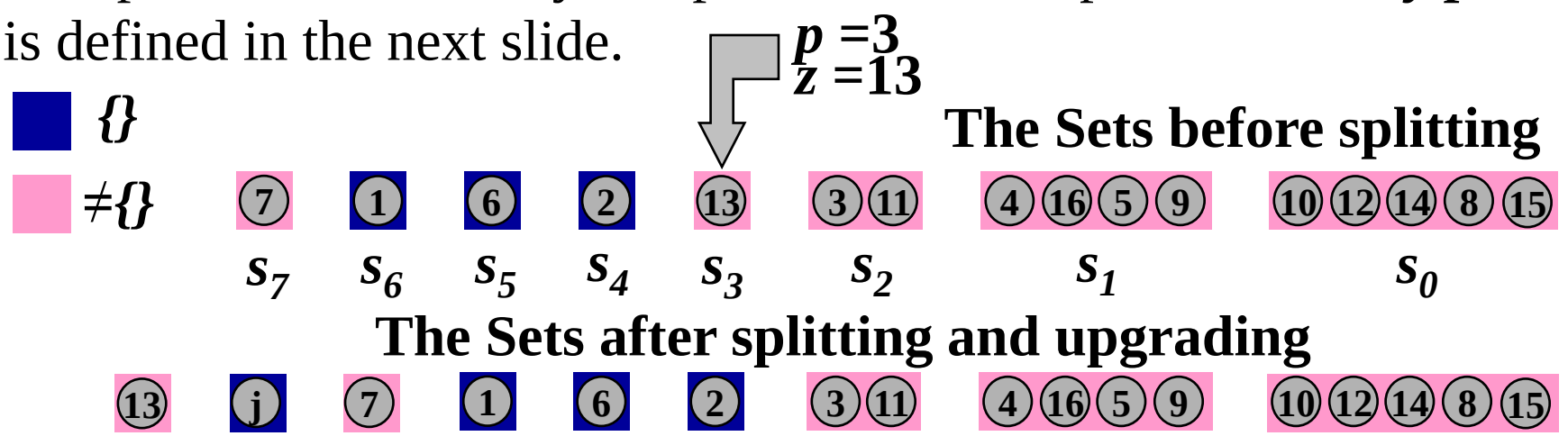
Let s_i is the parent set of the satellite block, s_p is s_i or one of its satellites:

i) up grade s_i and its satellites,

ii) create a set s_{new} containing a new node j corresponding to a new word w_{new}. Put $w_{new}=\Lambda$, $s_{new}=\{\}_{new}$, place it in the right of the rightmost satellite of s_i.

iii) repeat step (ii) for FBL_i times.

Note: If the extended dictionary is full, upgrade s_i and its satellites then prune the dictionary and proceed from step 2. ***Dictionary pruning*** is defined in the next slide.



Definitions:

***Dictionary Pruning*:**

Is a process of removing words from ***ED*** to keep the dictionary size constant. A word w_r may be removed from the dictionary and its corresponding node v_r removed from ***EDT*** if they satisfy the following four conditions:

1. $w_r \neq (\delta_i)$. δ_i is a control character in ***C*** .
2. $w_r \neq (c_i)$. c_i *is a character of a the alphabet* ***A***. (i.e. w_r is a single character word).
3. v_r has no child. (i.e. v_r is a leaf node).

If node v_r is in a singleton without satellite then remove the singleton and v_r. If node v_r is in an empty satellite set or in a parent set s_i of a family block then remove (FBL_i / 2) rightmost empty sets of the satellite block. (i.e. the remaining family block will consist of the parent set s_i and a the new satellite block of length SBL_i =(FBL_i/2)-1).

Note:

1. All children of the root node (except the no-data nodes of

Pruning (continued)

Notes:

the satellite sets ***{}***) cannot be removed from the ***ED***.

2. Only a leaf node may be removed.
3. A word which satisfies the three pruning conditions and with the lowest frequency of occurrence may be selected for removal from the ***ED***.
4. If an empty word is removed, the corresponding empty set***{}*** should also be removed from the ***EDT***.
5. If a node in a single node set is removed, the set should also be removed from the ***EDT***.
5. If a node is in a single node set of a family block then FBL_i/2 rightmost satellite should be removed from the ***EDT***.
6. If a node in a multiple node set is removed, the ***NCT*** or ***SynchCT*** must be ***reconstructed***.

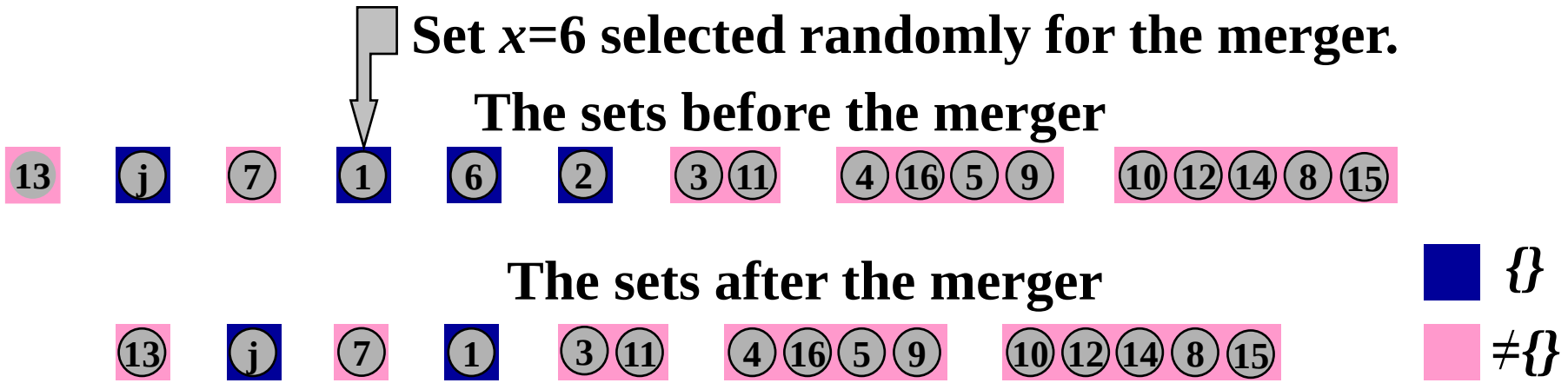
To satisfy note **3**, the process of pruning scans the ***OEDT*** from right to left to locate the lowest frequency of occurrence node v_r and its corresponding word w_r for pruning.

Definitions:

Merging*:** The merging process is applied only when the number of sets in the ***EDT is greater than ***N***. Merging is performed only on the sets not involved in the previous splitting process (i.e. the singleton parent set and its satellites, or the new two sets generated by the previous splitting process). Merging process is consists of two steps:

1. Select a set say s_x randomly.

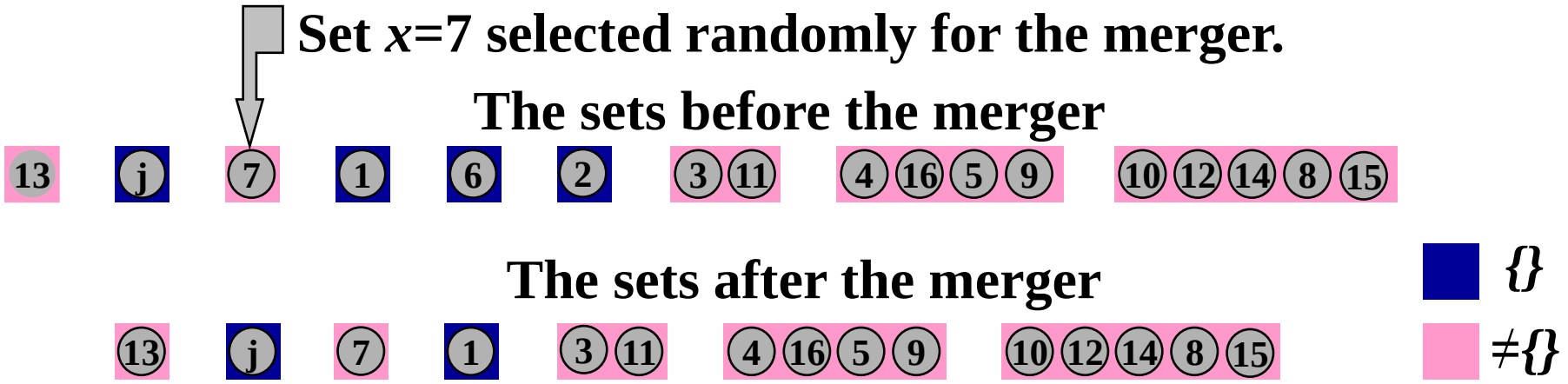
2.i If the set $s_x = \{\}$, then delete (FBL_i / 2) rightmost satellites of its family block. Update rank of the family sets. In the example shown below $x = 6$.



Merging (continued)

2.ii. If the set $s_x \neq \{\}$ and if s_x has satellite sets, then delete (FBL_i / 2) rightmost satellites of its family block. Update rank of the family sets.

In the following example below x = 7.



2.iii. If the set $s_x \neq \{\}$ and has no satellite set, select another set say s_y randomly. If the set $s_y = \{\}$, then delete (FBL_i / 2) rightmost satellites of its family block. Update rank of the family sets.

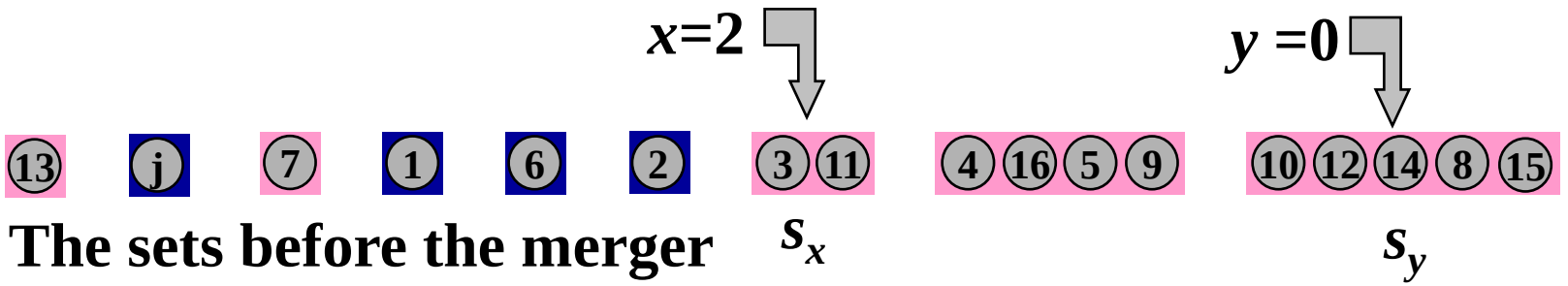
Merging (continued)

2.iv. If the set $s_x \neq \{\}$ and has no satellite set, select another set say s_y randomly. If the set $s_y \neq \{\}$ and s_y has satellite sets, then delete (FBL_i / 2) rightmost satellites of its family block. Update rank of the family sets.

2.v If the set $s_x \neq \{\}$ and has no satellite sets, select randomly another set say s_y. If the set $s_y \neq \{\}$ and also has no satellite sets, then merge the two sets to form a new set equal to the union of the two sets s_u. If $|s_y| > |s_x|$, then $s_u = s_x \cup s_y$. Otherwise $s_u = s_y \cup s_x$. Upgrade set s_u. Code bound s_x, s_y, and s_u and ***enlarge*** the ***NCT*** or ***SynchCT***. The processes of ***code bounding*** and ***enlarging*** are defined in the following slides.

Merging (continued)

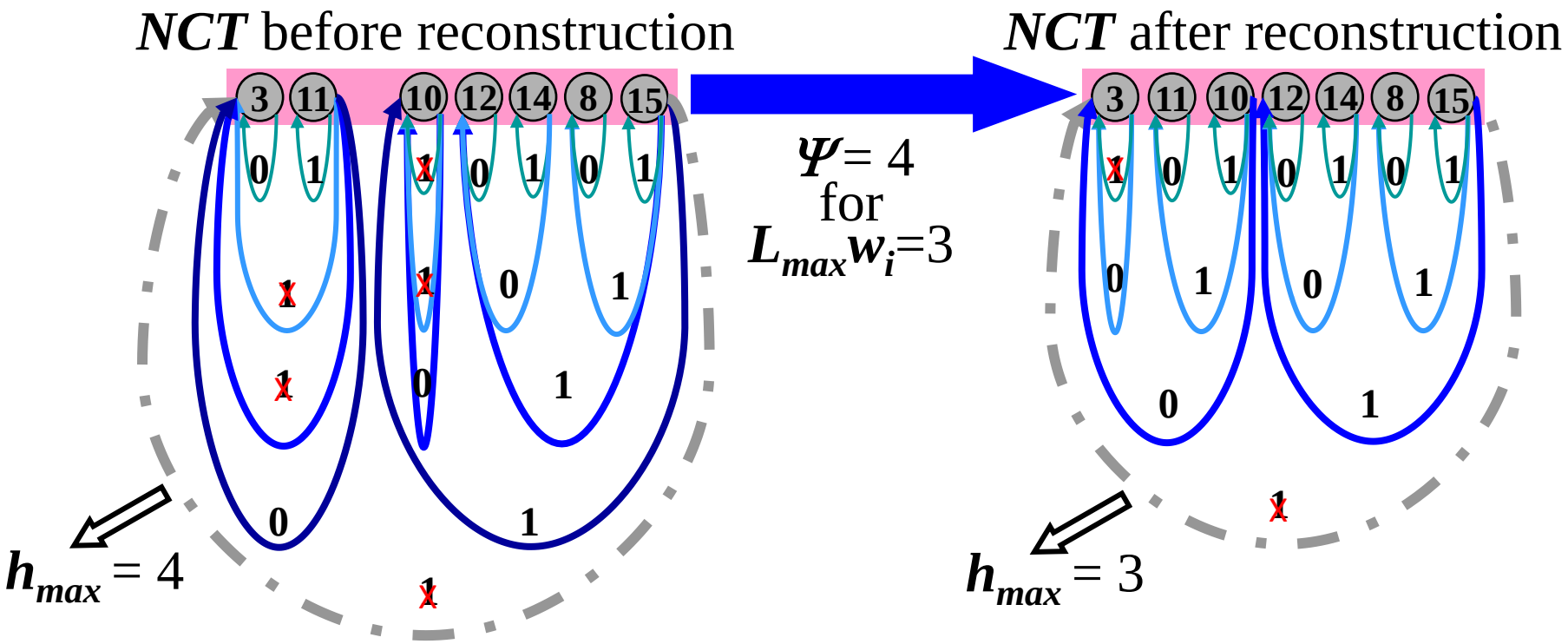
2.v The union of set 2 and set 0.:



Definitions:

***Code Bounding*:** The process of code bounding consists of the following steps:

1. If the root node-link or root synch-link height ($h_{x:max}$) of set s_x is equal to Ψ then its ***NCT*** or ***SynchCT*** must be reconstructed.
2. If the root node-link or root synch-link height ($h_{y:max}$) of set s_y is equal to Ψ, then its ***NCT*** or ***SynchCT*** must be reconstructed.



Definitions:

Note on code bounding: If the previous code bounding process on sets s_x and s_y results in either or both $h_{x:max}$ and $h_{y:max}$ equal to Ψ, then apply the code bounding process on set s_u.

NCT Enlarging*:** The enlarging process consists of adding the following node-links to the new ***NCT of set s_u:

Let $h_{y:max} > h_{x:max}$, and $\Delta = (h_{y:max} - h_{x:max})$.

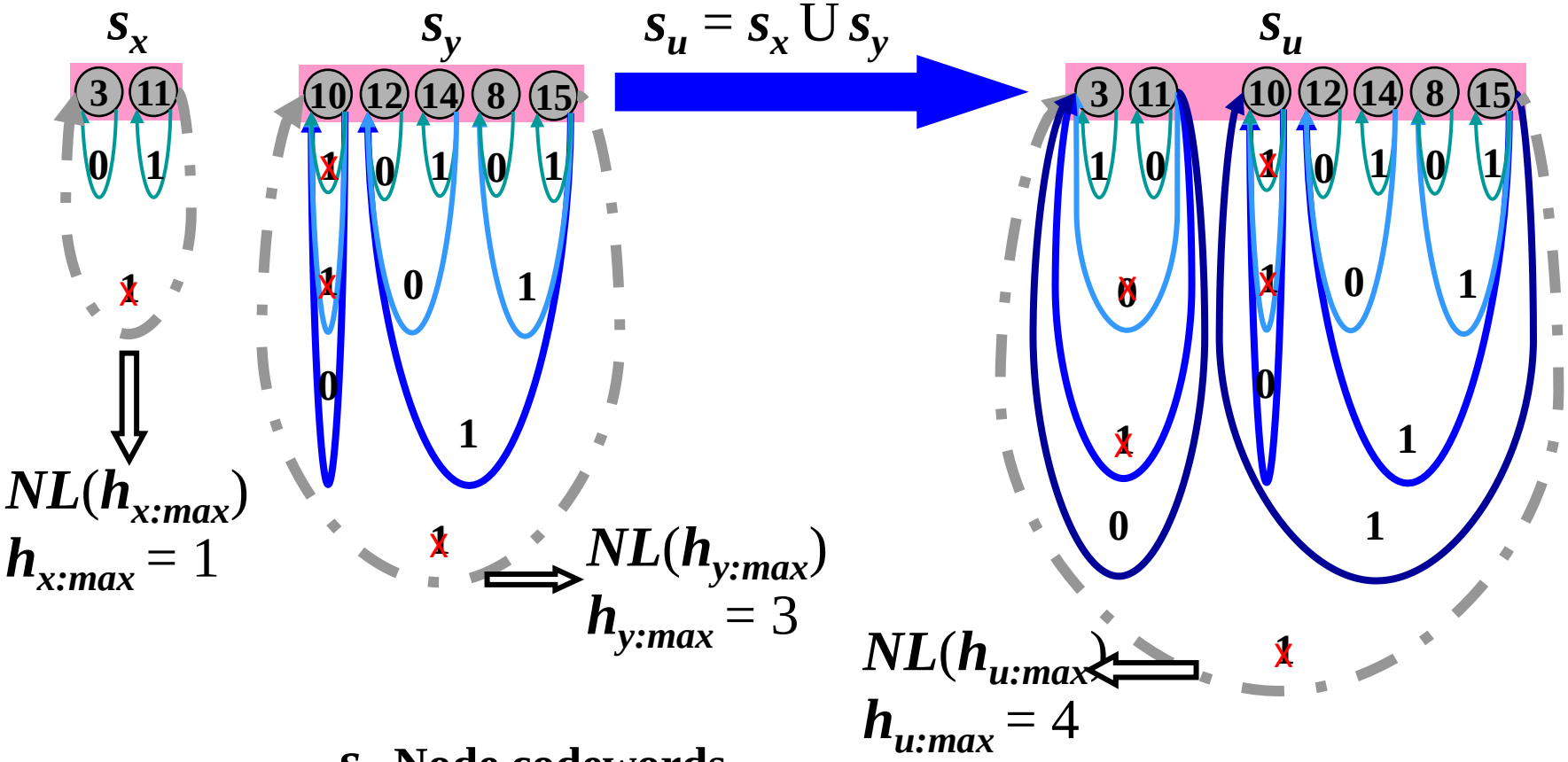
1. $NL(h_{x:max}+j)$, where j=0, 1, …, Δ, containing all nodes of s_x.
2. $NL(h_{u:max}-1)_0$ containing all node of s_y.
3. The root node-link $NL(h_{u:max})$ containing all nodes of s_u, $h_{u:max} = h_{y:max} + 1$.

Apply construction rules and conditions of ***NCT*** in the process of adding new links, and in the process of coding all node-links starting with $NL(0)_0$.

Figure 46 shows an example of constructing the ***NCT*** of set s_u.

Merging (continued)

$|s_x| = 2$, $|s_y| = 5$, $|s_u| = 7$, $\Delta = 2$, $\zeta > 7$, $h_{x:max} = 1$, $h_{y:max} = 3$, and $h_{u:max} = 4$.



s_u Node codewords

Node15: 1111 **Node14: 1101** **Node10: 10** **Node 3: 01**

Node 8: 1110 **Node12: 1100** **Node11: 00**

Definitions:

SynchCT Enlarging*:** The enlarging process consists of adding the following links and their corresponding synch-links to the new ***SynchCT of set s_u:

Let $h_{y:max} > h_{x:max}$, and $\Delta = (h_{y:max} - h_{x:max})$.

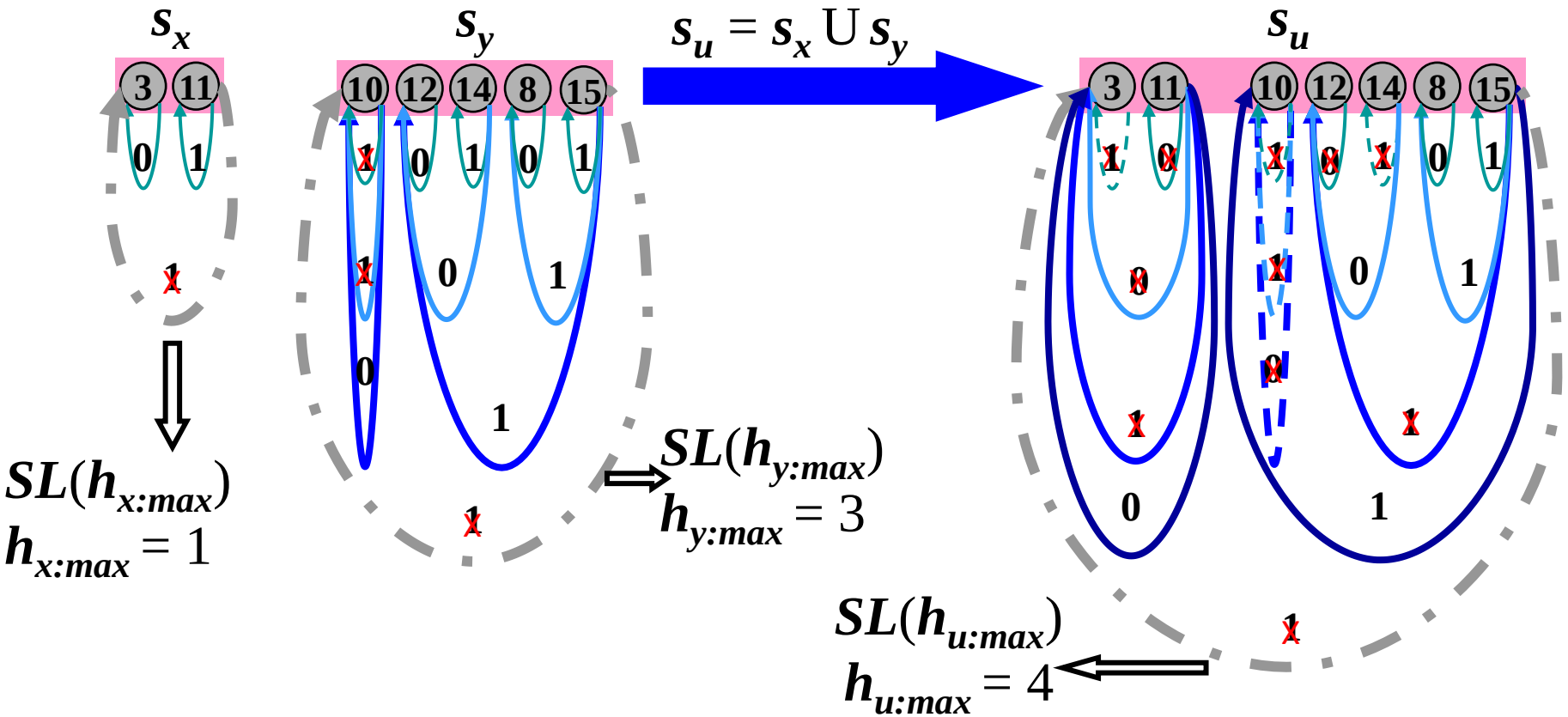
1. $SL(h_{x:max}+j)$, where j=0, 1, …, Δ, containing all nodes of s_x.
2. $SL(h_{u:max}-1)_0$ containing all node of s_y.
3. The root synch-link $SL(h_{u:max})$ containing all nodes of s_u, $h_{u:max} = h_{y:max} + 1$.

Apply construction rules and conditions of ***SynchCT*** in the process of adding new links, and in the process of coding all synch-links starting with $SL(0)_0$.

Figure 47 shows an example of constructing the ***SynchCT*** of set s_u.

Merging (continued)

$|s_x| = 2$, $|s_y| = 5$, $|s_u| = 7$, $\Delta = 2$, $\zeta = 6$ or 7, $h_{x:max} = 1$, $h_{y:max} = 3$, and $h_{u:max} = 4$. Let v_{15}, v_8, v_{12} and v_{11} be lead or control nodes, then the ***SynchCT*** of s_u is shown below.



s_u Synch codewords

Node15: 111

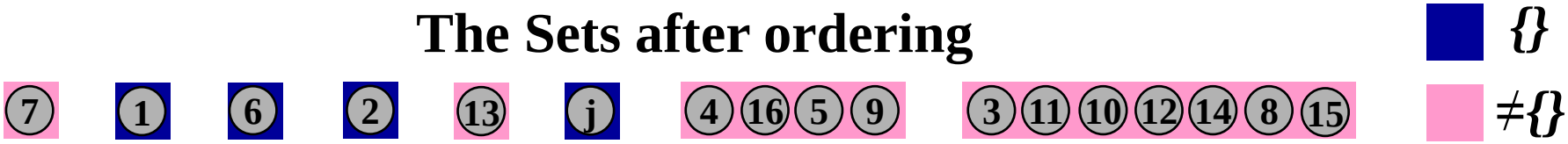
Node 8: 110 **Node12: 10** **Node11: 0**

Merging (continued)

2.v.1. The remaining sets of the ***OEDT*** should be ordered according to rank as shown below.

The Sets before ordering

The Sets after ordering



Definitions:

***Dictionary updating*:**

The process of updating the extended dictionary consists of three steps. The first step is testing the conditions for a new word to exist, the second step is updating the data of the corresponding node in the ***OEDT*** and the third is replacing an empty word in the ***OED*** with the new word.

***Dictionary updating* (*continued*):**

The dictionary updating process is performed before every matching process and the ordering process is performed immediately after it.

Let the string $s_n=(e_1e_2\ldots e_i\ldots e_p)$ be the input string from a file of characters in the alphabet A at the time interval n, which has been matched to the longest word (w_n), in the ED and let the corresponding input string and word at interval (n-1) be $s_{n-1}= w_{n-1}= (c_1c_2\ldots c_i\ldots c_t)$, where e_i and c_i are characters in A. ν_n and ν_{n-1} are the corresponding nodes of words w_n and w_{n-1} in $OEDT$. Then the new word is given by: $w_{new} = (c_1c_2\ldots c_te_1)$, and $Lw_{new} = Ls_{n-1} + 1 = t + 1$.

Step 1: There are four conditions under which a new word does not exist: **1.** If ν_{n-1} is a node in a multiple node set.

2. If ν_{n-1} is in a singleton set without any satellites.

3. w_{new} is already in the ED.

4. Lw_{new} > the maximum allowed word length.

If any one of the above conditions is satisfied put $w_{new} = \Lambda$. If $w_{new} \neq \Lambda$

***Dictionary updating* (*continued*):**

and s_s is the middle satellite of ν_{n-1} family, let node ν_s be the data-empty node in s_s corresponding to an empty word Λ_s in the ***OED***.

Step 2: 1. Replace the null character (-1) of ν_s data by the character e_1.

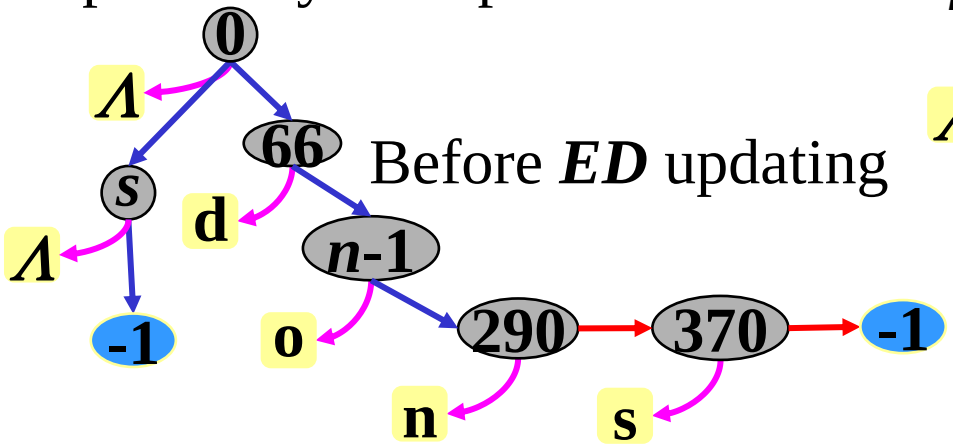
2. Replace the parent node of ν_s by ν_{n-1} in the ***OEDT***.

3. Replace the sibling node of ν_s by the child node of ν_{n-1} in the ***OEDT***.

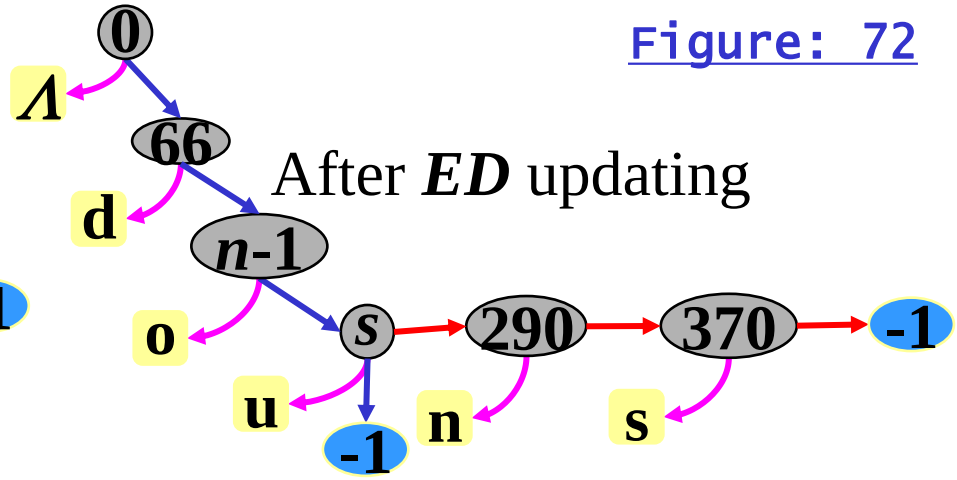
4. Replace the child node of ν_{n-1} by ν_s in the ***OEDT***.

Step 3: 1. Replace the empty word Λ_s by w_{new} in the ***OED***.

Note: In the case of a lossy ***S&MC*** algorithm w_n and w_{n-1} may be replaced by their predicted values $\hat{w}_n$ and $\hat{w}_{n-1}$. Update rank of the sets.



Before ***ED*** updating



After ***ED*** updating

Figure: 72

Definitions:

***OEDT* Initialisation**:

In the initial state the extended dictionary ***ED*** contains only α single character words corresponding to the alphabet ***A*** and χ single character words corresponding to the control elements in ***C***. The ***OEDT*** consists of a root node with no data and ($\alpha + \chi$) children nodes corresponding to all the single character words of the extended dictionary. Each node except the root node is put in a node set to form ($\alpha + \chi$) singleton sets. Left set links are constructed first and then their corresponding right links. Since all sets are singleton they are all redundant links of the root set-links.

Node zero of the tree usually identifies the root node, node one to node α identify the single word characters of the alphabet ***A*** and node (α +1) to node ($\alpha + \chi$ +1) identify the control functions. All these nodes cannot be pruned and therefore will keep their identifiers during the entire compression process. No pruning or merging will be performed until the extended dictionary ***ED*** is filled with Φ words and the tree has N sets respectively.

Definitions:

Lossless process: If the output strings of a decoder are identical to the input strings to the encoder the compression procedure is known as a lossless process.

Lossy process: If the output strings of a decoder differ from that of the input strings to the encoder the compression procedure is known as a lossy process.

Only lossless algorithms are used for text compression, while sound, images and video compression may use lossy algorithms. A lossy compression algorithm may result in a degradation of the output strings in favour of increase in the compression ratio.

Compression ratio: Compression ratio ($\boldsymbol{R}$) is defined as the ratio of $\log_2(\alpha)$ times the number of character in the input file ($|s_{SF}|$) to the number of binary digits in the output (compressed) file ($|s_{OP}|$).

$$R = (\log_2(\alpha)\,|s_{SF}|) / |s_{OP}| = L_{average} w_i \text{ in } D \cdot \log_2(\alpha) / L_{average} w_{word}$$

Definitions:

Lossless Split and Merge Compression Algorithm:

The encoder output is a binary word codeword resulting from the string concatenation of set and node binary codewords. The decoder at any time interval (***n***) can identify correctly the set (s_n), the node (ν_n) in s_n and the corresponding word (w_n) in the dictionary. The decoder output is a string of characters in the alphabet ***A*** identical to the input string to the encoder.

Conditions for successful synchronisation between the encoder and the decoder of ***S&MC*** lossless algorithm are:-

3.1. The random generators used to select the two sets for merger at the encoder and decoder must generate the same sequence of random numbers.

3.2. The decoder receives uncorrupted set codewords.

3.3. The decoder receives uncorrupted node codewords.

The ***S&MC*** lossless algorithm is shown to have a higher compression ratio compared with other practical systems available on the market.

Definitions:

Lossy Split and Merge Compression Algorithm:

Conditions for ***S&MC*** algorithm are said to be lossy if for $|s_n| \leq \zeta$, (where s_n is the set containing the node (ν_n) corresponding to the longest dictionary word (w_n) which has been matched to the input string ($s_n = w_n$) at the time interval (n)), the encoder output is a binary word codeword resulting from the string concatenation of set and node codewords, while for $|s_n| > \zeta$ the encoder generates a word codeword equal to the string concatenation of set and synch codewords. Sync codewords are constructed to identify only the control-nodes. Other nodes are identified by their corresponding wordlength. For $|s_n| \leq \zeta$ a lossy ***S&MC*** algorithm decoder generates output strings without loss or degradation equal to s_n. However for $|s_n| > \zeta$ the decoder generate a prediction (w_n) of w_n as the most likely word from a set of words (corresponding to nodes in s_n) of length equal to Lw_n. This may result in a degradation of the output strings in favour of increase in the compression ratio.

Lossy Split and Merge Compression Algorithm (*continued*):

Conditions for successful synchronisation between the encoder and the decoder of ***S&MC*** lossy algorithm are:-

4.1. The conditions of lossless ***S&MC*** algorithm are satisfied for set size $|s_n| \leq \zeta$.

4.2. In the absence of the node codeword for set size $|s_n| > \zeta$, the decoder receives an uncorrupted synch codeword.

S&MC predictor:

The input to the predictor is a set of equal length words. The predictor output is a prediction (w_n) of w_n. There are a large variety of possibilities suggested in the literature to predict a given word from previously known ones depending on the type and properties of the input file. Here, it is suggested that a word is picked randomly in proportion to the probabilities of the input words. The probability of each of these words may be estimated from the ***NCT*** links. Consider set 1 of the previous examples: $s_0 = \{\nu_{10}, \nu_{12}, \nu_{14}, \nu_8, \nu_{15}\}$, with $Lw_8 = Lw_{12} = Lw_{10} = 3$, $Lw_{14} = 1$ and ν_{15} is a control-node.

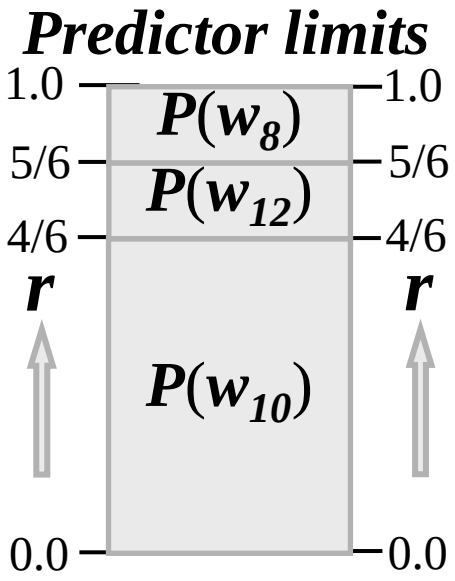
***S&MC predictor* (continued):**

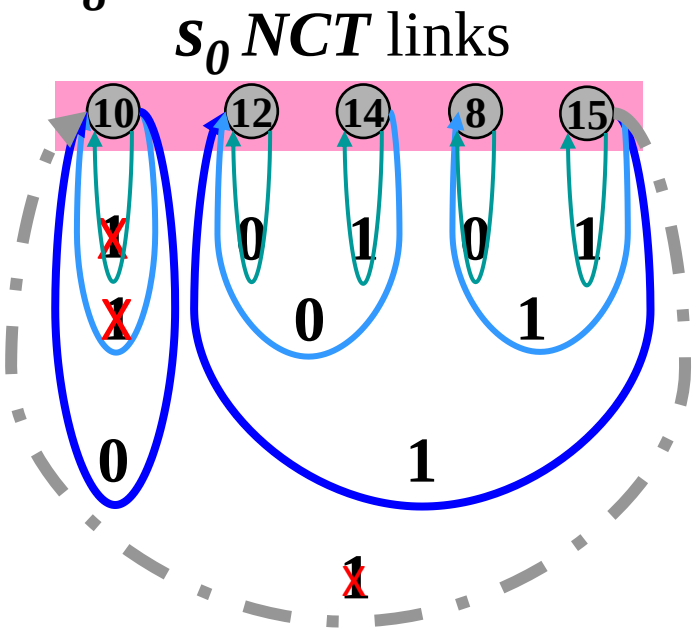
If the decoder identifies node v_8, then w_8, w_{12} and w_{10} are the input words to the predictor from the ***NCT*** links. Their probabilities may be estimated as $P(w_8) = 1/6$, $P(w_{12}) = 1/6$ and $P(w_{10}) = 4/6$.

A scale from **0** to **1** is constructed given by:

0, $P(w_{10}) = \underline{4/6}$, $P(w_{10})+P(w_{12}) = \underline{5/6}$ and $P(w_{10})+P(w_{12})+P(w_8) = \underline{1}$.

Let r be a random number between **0** and **1**, generated at interval n by a random generator. If $r \leq 4/6$ then $\hat{w}_n = w_{10}$, if $4/6 < r \leq 5/6$ then $\hat{w}_n = w_{12}$, otherwise assume $\hat{w}_n = w_8$.





***S&MC predictor* (continue):**

Word probabilities Estimation: Let W_s be a subset of words corresponding to nodes in s_p consisting of the k words input to the predictor $W_s = \{w_1, w_2, \ldots, w_i, \ldots, w_k\}$ and let their node-codeword lengths be $Lw_{node:1}, \ldots, Lw_{node:i}, \ldots, Lw_{node:k}$ respectively. Then the probability of the i-th word is given by:

$$P(w_i) = ((1/2) \wedge Lw_{node:i}) \Big/ \sum_{i=1}^{k} ((1/2) \wedge Lw_{node:i})$$

Node-codeword lengths of W_s may be computed from the *NCT* of set s_p. *NCT* of s_p may be constructed from the *SynchCT* by making all zero height synch-links as type 1 links and then coding the new tree.

Consider s_1 of the previous example: $s_1 = \{\nu_{10}, \nu_{12}, \nu_{14}, \nu_8, \nu_{15}\}$, with $Lw_8 = Lw_{12} = Lw_{10} = 3$, $Lw_{14} = 1$ and ν_{15} is a control-node.

If $W_s = \{w_8, w_{12}, w_{10}\}$, $Lw_{node:8} = Lw_{node:12} = 3$, and $Lw_{node:10} = 1$. The probability of each word is therefore:

$P(w_8) = P(w_{12}) = (1/2)^3 / ((1/2)^3 + (1/2)^3 + (1/2)^1) = 1/6$ and

$P(w_{10}) = (1/2)^1 / ((1/2)^3 + (1/2)^3 + (1/2)^1) = 4/6.$

Note: If $W_s = \{w_{14}\}$ then $P(w_{14}) = 1$ and if $W_s = \{w_{15}\}$ then $P(w_{15}) = 1$.

S&M algorithm: The Tree Structure

α_i *is the* i^{th} *character of a finite alphabet* $\boldsymbol{A}$ *of* ($\boldsymbol{N}$) *characters.*
For an 8-bit character (ASCII character) set, the number of characters in $\boldsymbol{A}$ is equal to $\boldsymbol{N}$ = 256. A negative value (usually **-1**) denotes a null character.

δ_i *is the i-th control character of a finite set* $\boldsymbol{X}$ *of* (χ) *control characters. (usually in the range of 3-5 characters).*
Control characters are used to communicate control functions between the compression and the decompression processors.

$\boldsymbol{w_i}$ *the i-th word.*
$\boldsymbol{w_i} = (\alpha_1\alpha_2...\alpha_{L(w_i)})$, where $\boldsymbol{L}(\boldsymbol{w_i})$ is the wordlength of word $\boldsymbol{w_i}$.

Λ *An empty* (***null***) *word*, a word with no characters $\boldsymbol{L}(\Lambda) = \boldsymbol{0}$.

$\boldsymbol{W_i}$ *The i-th set of words* $\boldsymbol{W_i} = \{\boldsymbol{w_1}, \boldsymbol{w_2}, ..., \boldsymbol{w_{|Wi|}}\}$ where $|\boldsymbol{W_i}|$ is the number of words in $\boldsymbol{W_i}$ (***set size***). $|\boldsymbol{W_i}| = 1, 2, \ldots, |\boldsymbol{W_i}|_{\max}$, where $|\boldsymbol{W_i}|_{\max}$ is the maximum allowable number of words in a given set of words. i.e. maximum set side.

s&M algorithm: The Tree structure

$L_{max}(W_i)$	*is the maximum (longest) wordlength of words in set* W_i.
$L_{average}(W_i)$	*is the average wordlength of words in set* W_i.
$\{\}$	*an empty (**null**) set of elements, a set containing no word, sets or characters; length of null set is zero;* $L(\{\}) = 0$.
D	*Dictionary: a set of* M *words.* $D = \{w_1, w_2, \ldots, w_M\}$. If D contains ASCII alphabet then $256 \le M \le \Phi$, where Φ is the maximum allowable number of words in D; $\Phi = \lvert D \rvert_{max}$
λ	*is* $L_{max}(D)$ *the maximum wordlength of words in* D.
ϖ	*is* $L_{average}(D)$ *the average wordlength of words in* D.
s_i	*is a set of nodes in a **tree** (**t**),* $s_i = \{v_1, v_2, v_3, \ldots\}$, *where* v_i *is the i-th node in the **tree** (**t**).*
$\lfloor x \rfloor$	*smallest integer* $\ge x$.
$\lceil x \rceil$	*greatest integer* $\le x$.

S&M algorithm: The Tree Structure

s_{SF} *source file character string*: is the input file to be compressed, which is a character string consisting of a sequence of Ls_{SF} characters in the alphabet.

The s_{SF} could be a binary file, or a text file of 8-bit ASCII characters, or an image file of G-bit grey-level pixels, or a sound file of Q-bit pulse code modulation samples, or a video file of G-bit grey-level pixels, or a combination of the above formats.

s_{in} *is the Input Character String of length* $L(s_{in})$, where $L(s_{in}) = 1, 2, \ldots, L_{max}(s_{in})$; $L_{max}(s_{in})$ is the maximum string length of s_{in}; $1 \leq L_{max}(s_{in}) \leq L(s_{SF})$. $\mathbf{s_{in}}$ may be formed by reading characters from file $\mathbf{s_{SF}}$ in sequence and concatenating with Λ.

EOF *end of file*: A negative value (**-1**) denoting a null character.

S&M algorithm: The Tree Structure

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

The Greek Alphabet:

A	A	α	Alpha	N	N	n	Nu
B	B	β	Beta	X	X	x	Xi
G	Γ	γ	Gamma	O	o	o	Omicron
D	Δ	δ	Delta	P	Π	π	Pi
E	E	ε	Epsilon	R	P	ρ	Rho
Z	Z	ζ	Zeta	S	Σ	σ	Sigma
H	H	η	Eta	T	T	τ	Tau
Q	Θ	θ	Theta	U	Y	υ	Upsilon
I	I	ι	Iota	F	Φ	ϕ	Phi
K	K	κ	Kappa	C	X	χ	Chi
L	Λ	λ	Lambda	U	Ψ	ψ	Psi
M	M	μ	Mu	W	Ω	ω	Omega