S&M
Split and Merge Compression Algorithm

By

Abdullah Hashim

S&M algorithm: the lossy case

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

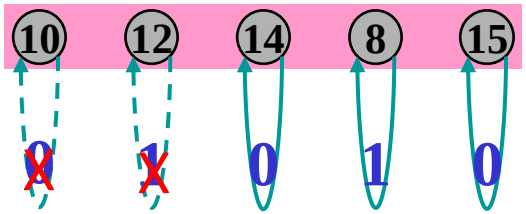
average-length $L_{average}[w_{synch}(s_i)]$ than that of the node codeword $L_{average}[w_{node}(s_i)]$.

***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*

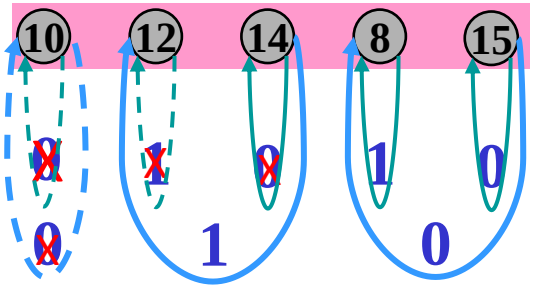
S&M algorithm: the lossy case

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

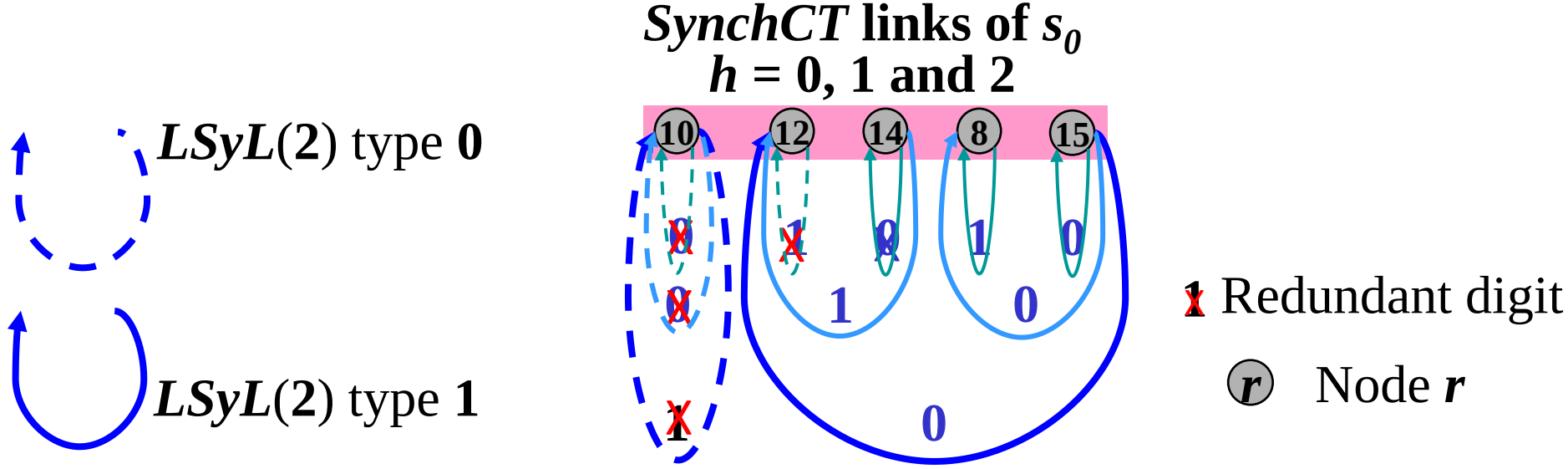


~~1~~ redundant digit

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

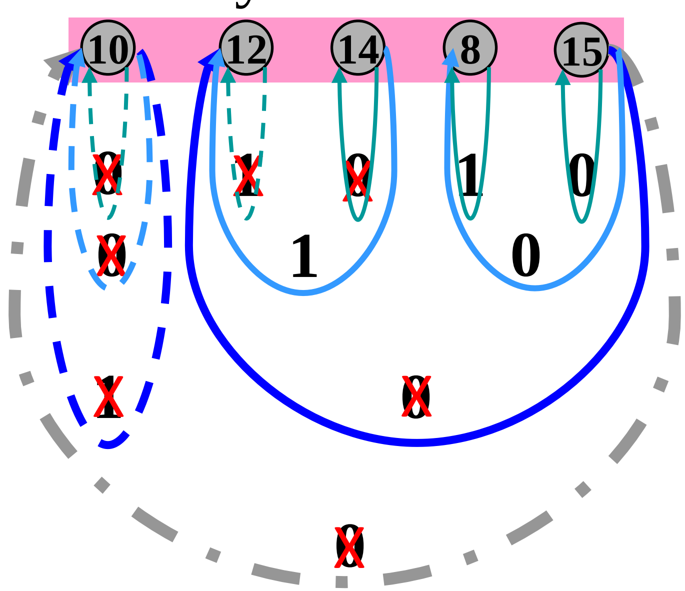
Left synch-link of height three {***LSyL***(3)}**:** is a link connecting the rightmost node of the (***2i***)-th left-synch-link {***LSyL***(2)$_{2i}$} to the leftmost node in the (***2i+1***)-th left-synch-link {***LSyL***(2)$_{2i+1}$}. If ***LSyL***(2)$_{2i+1}$ is a null-link then ***LSyL***(3)$_i$ 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

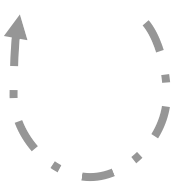
node 15 codeword = **00**

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

node 14 codeword = **1**



Synch-links of set s_0



LSyL(3) type **0**

1 Redundant digit

r node ***r***

S&M algorithm: the lossy case

Notes on Synch Coding Tree *SynchCT* :

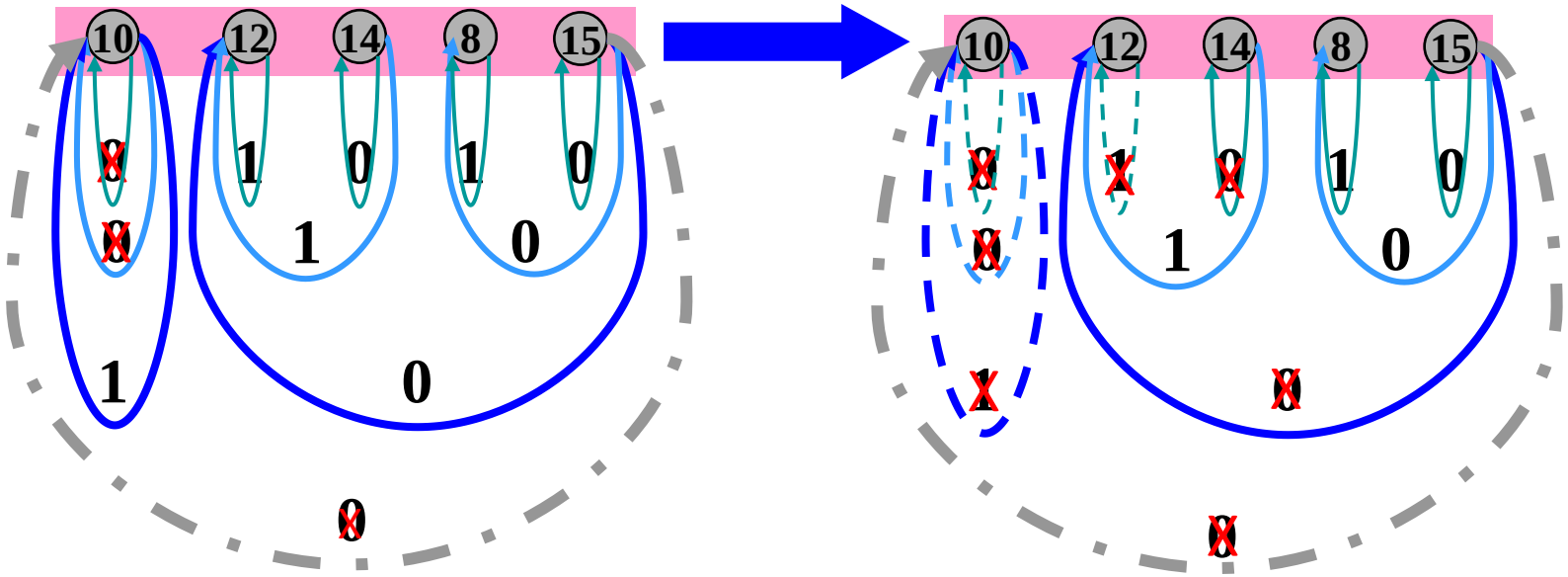
1. In ***S&M*** lossy algorithm, node codeword $w_{node}(s_i)$ is used for set size $|s_i| \leq \zeta$ and synch codeword $w_{synch}(s_i)$ for set size $|s_p| > \zeta$.

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.

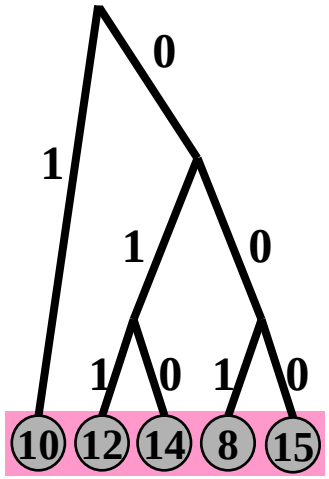
S&M algorithm: the lossy case

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

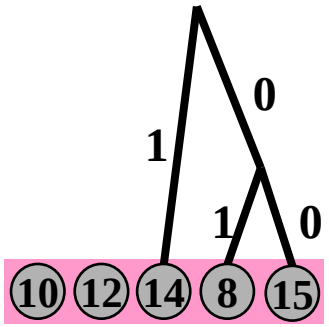
10. A ***SL*(*h*+1)** surrounds all nodes of two adjacent ***SL*(*h*)** synch-links. If one of the two ***SL*(*h*)** synch-links is of type 0, then the ***SL*(*h*)** type 1 becomes redundant. ***SL*(*h*+1)** type 1 surrounds at least one ***SL*(*h*)** of type 1. ***SL*(*h*+1)** type 0 surrounds only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

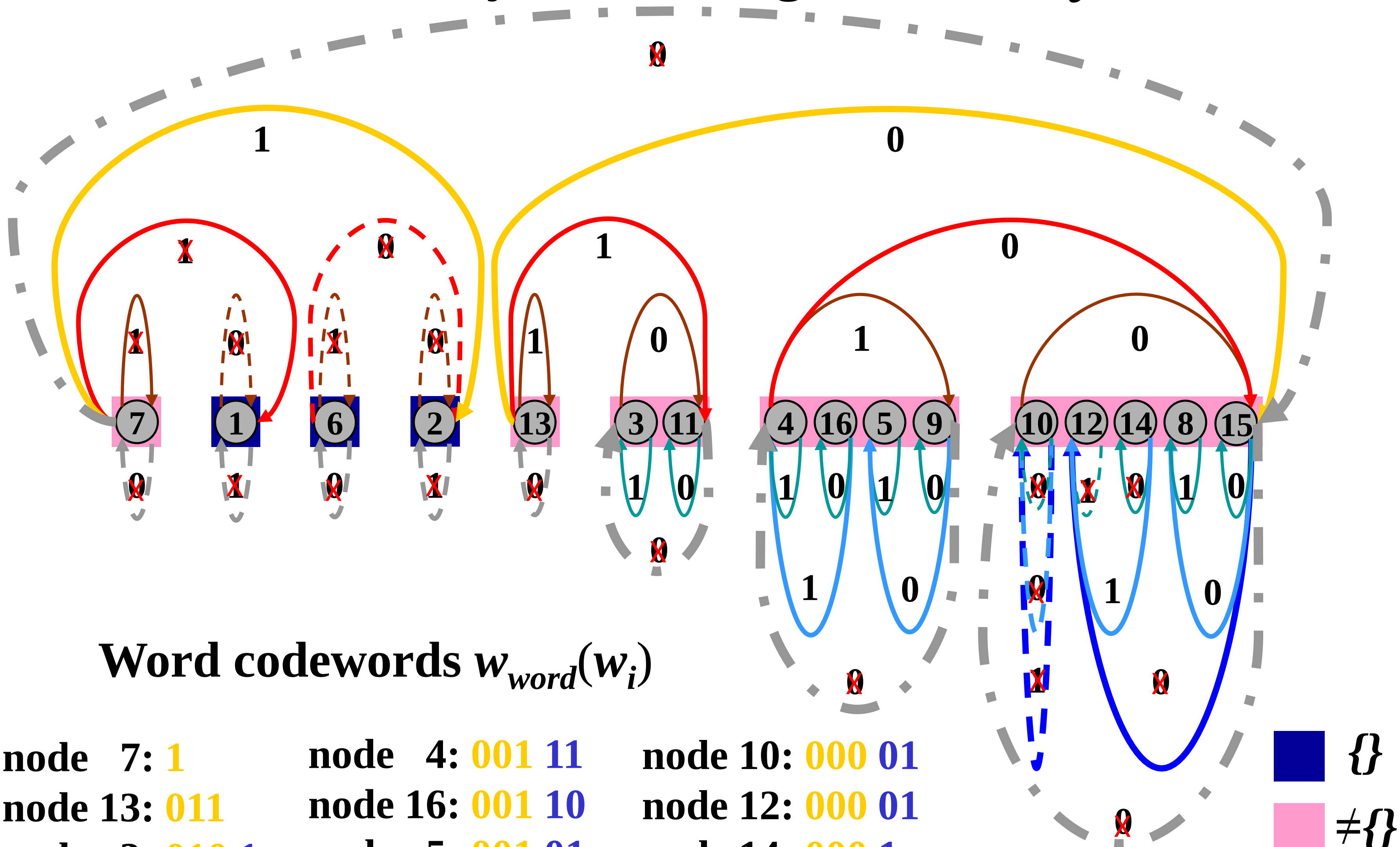
Set s_0 Node Coding Tree



Set s_0 Sync Coding Tree



Set, Node and Synch Coding Trees for $\zeta = 4$



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

{}

≠{}

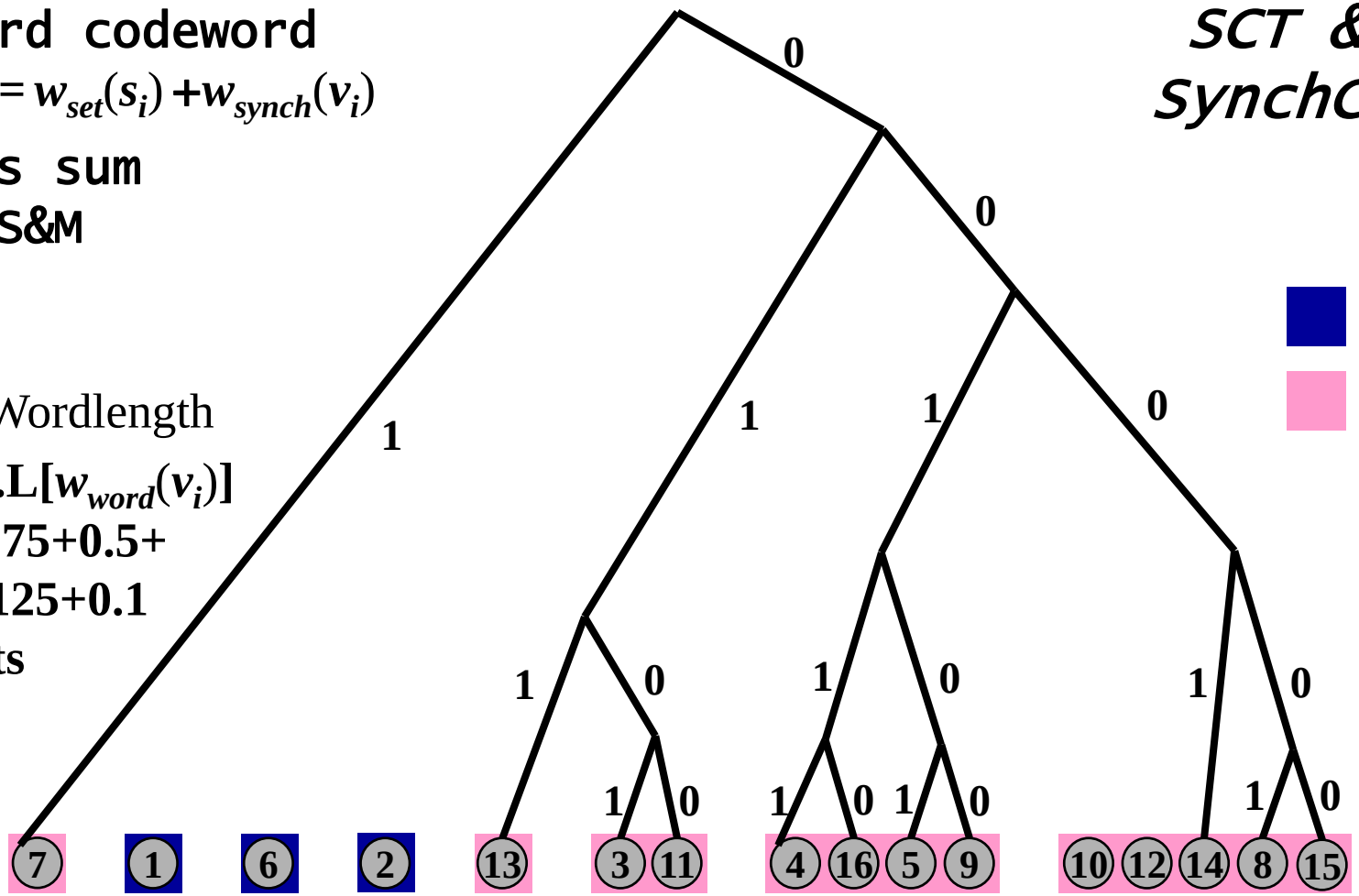
Average Wordlength

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

= 0.5+0.375+0.5+

0.625+0.125+0.1

= 2.25 bits



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: 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

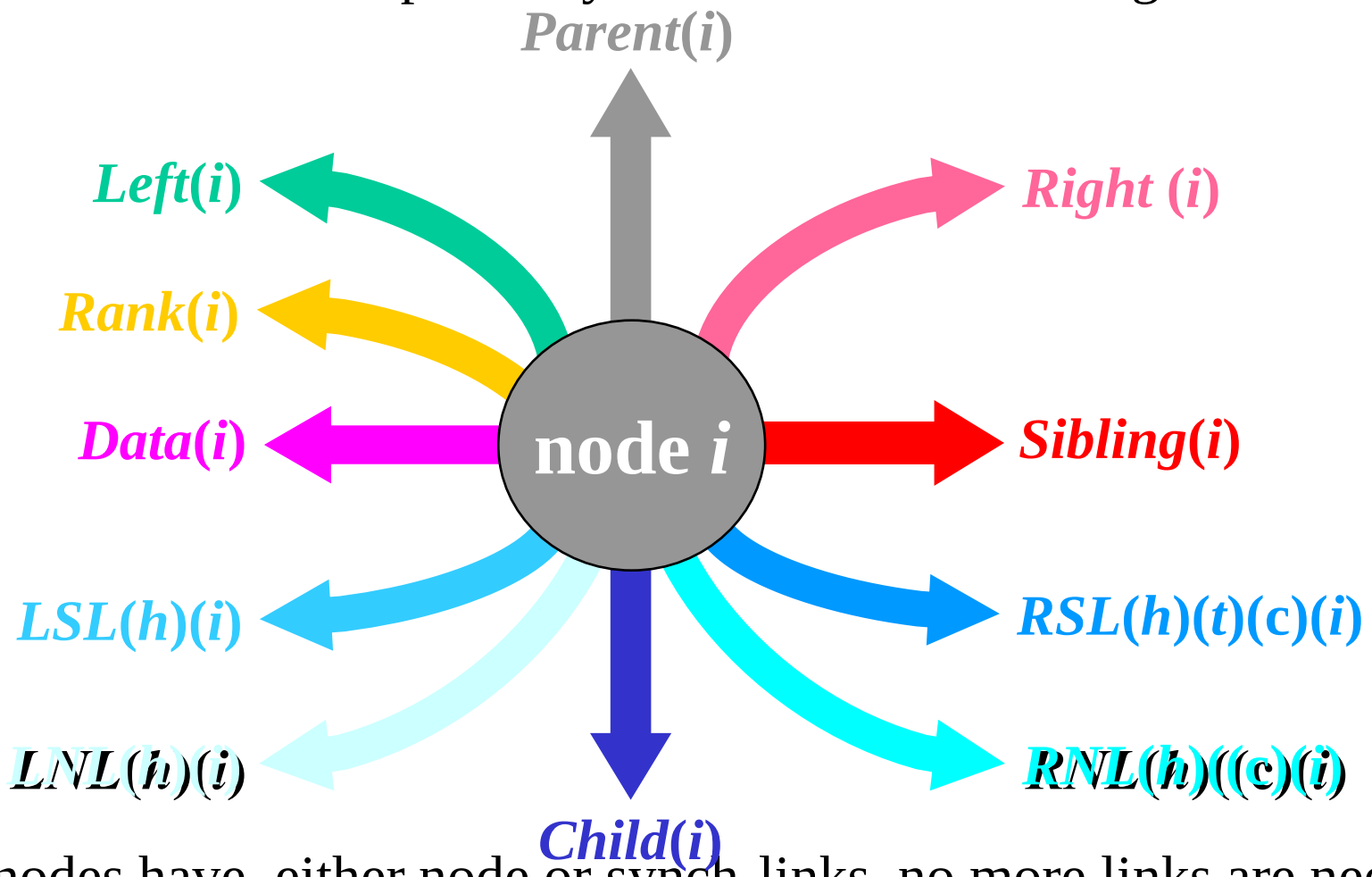
node 14: 000 1

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 (Φ **-** 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.

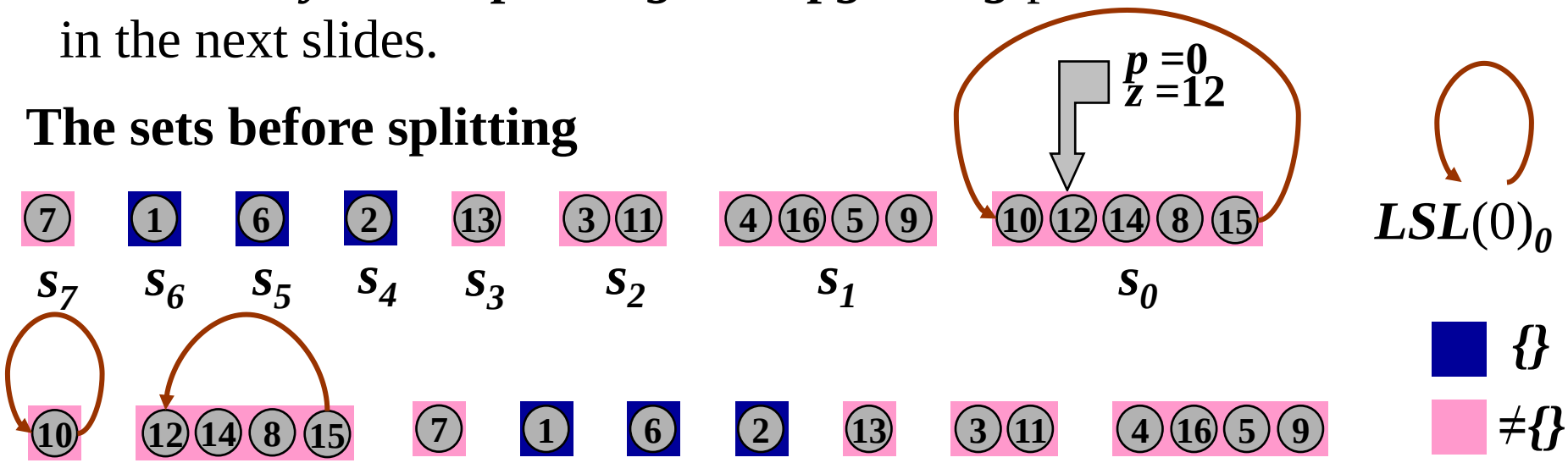
To achieve a robust coding the value of the bound (Ψ) on the links height is usually set to, or near to the value $\lfloor \log_2 \eta \rfloor + 1$.

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.

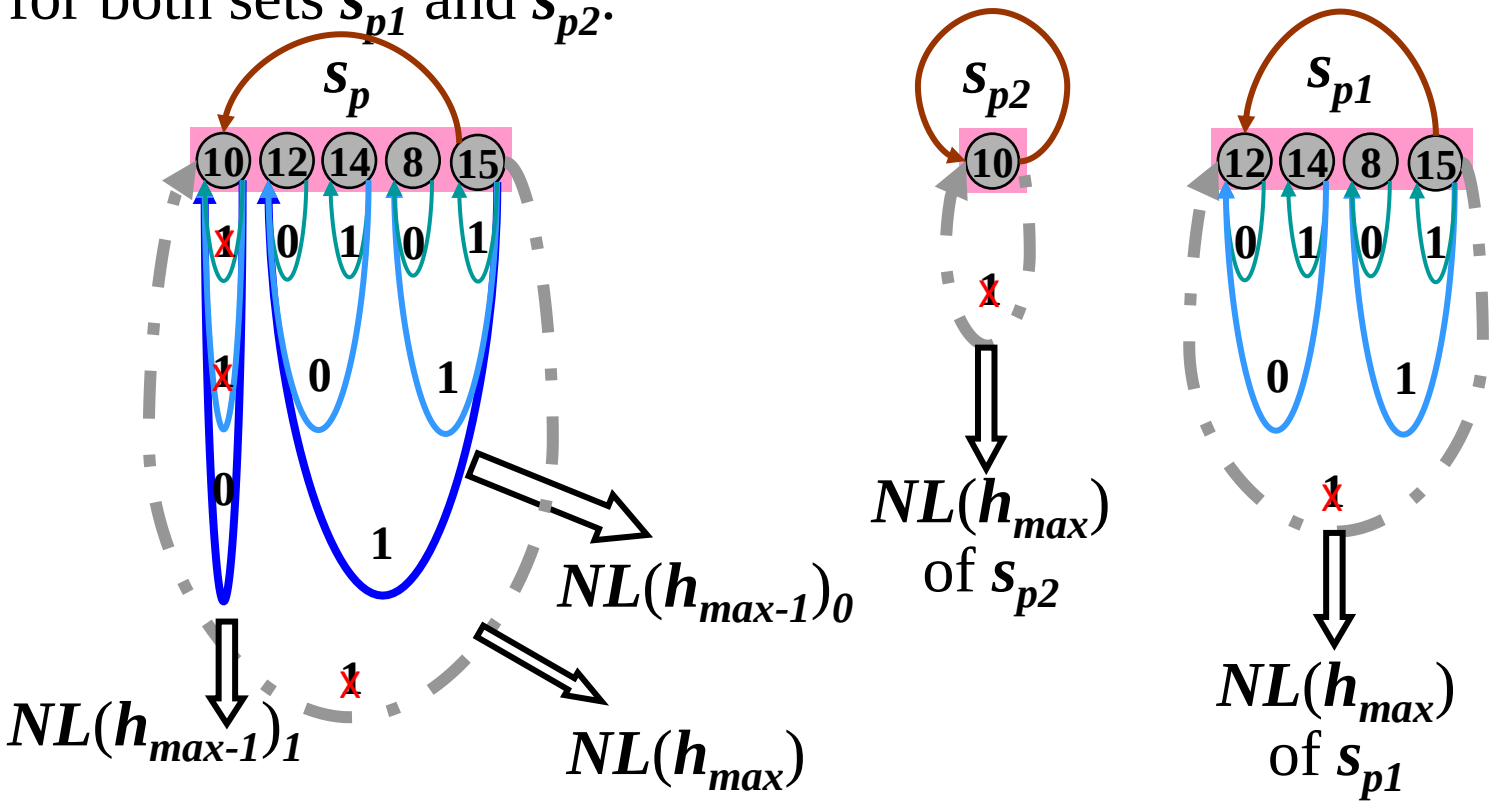
1. If $|s_p| > 1$ then, for a ***NCT*** split s_p into two disjoint equal probability sets s_{p1} and s_{p2} ; s_{p1} containing all nodes in the node-link $NL(h_{max}\text{-}1)_0$ and s_{p2} containing all nodes in the node-link $NL(h_{max}\text{-}1)_1$ of s_p. Similarly for ***SynchCT***, s_{p1} containing 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}.

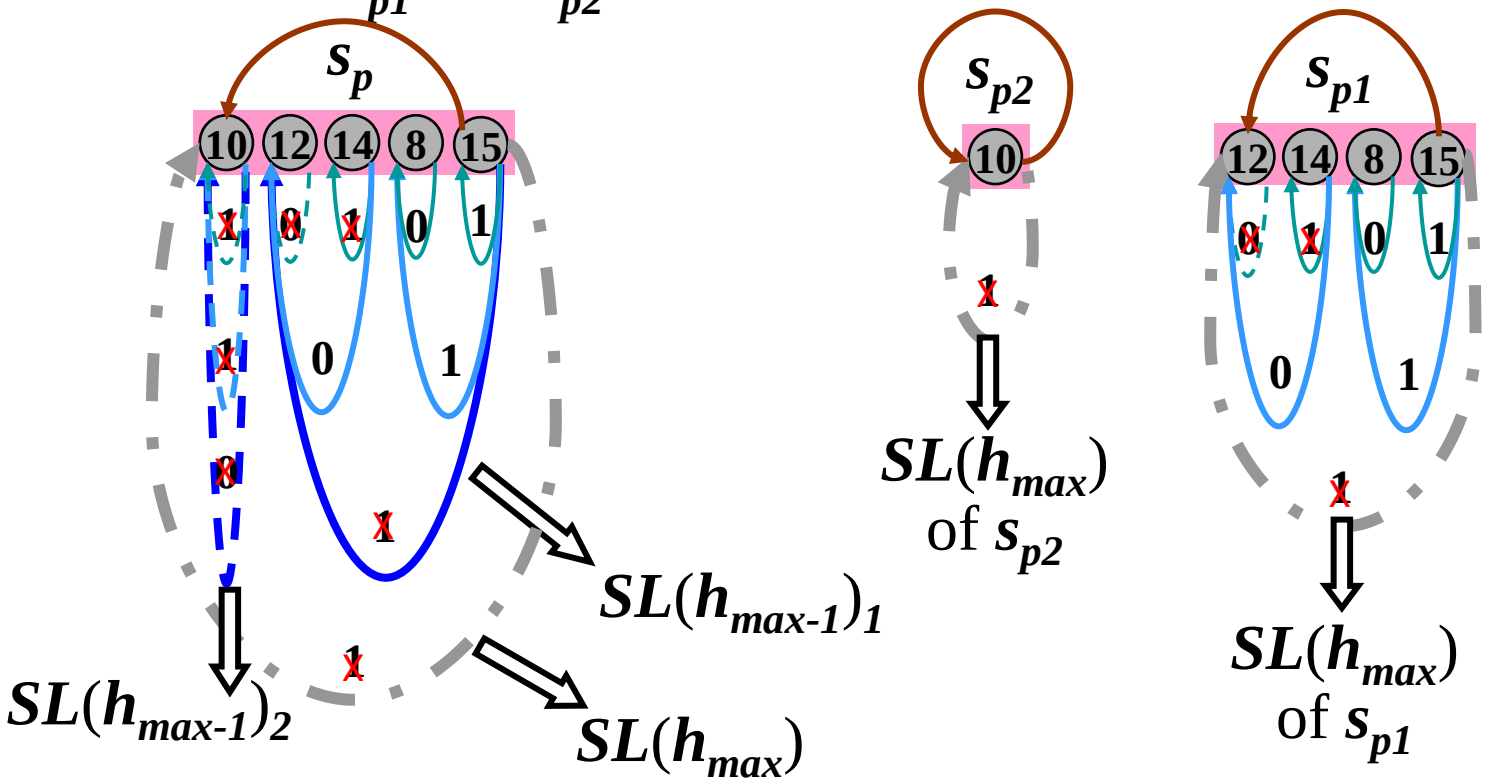
1. Delete $NL(h_{max})$ of s_p .
2. $NL(h_{max}\text{-}1)_0$ of s_p becomes equal to the redundant root node-link $NL(h_{max})$ of s_{p1}. Likewise for $NL(h_{max}\text{-}1)_1$ and s_{p2}.
3. Delete all duplicate root node-links except the innermost root-link for both sets s_{p1} and s_{p2}.



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):**

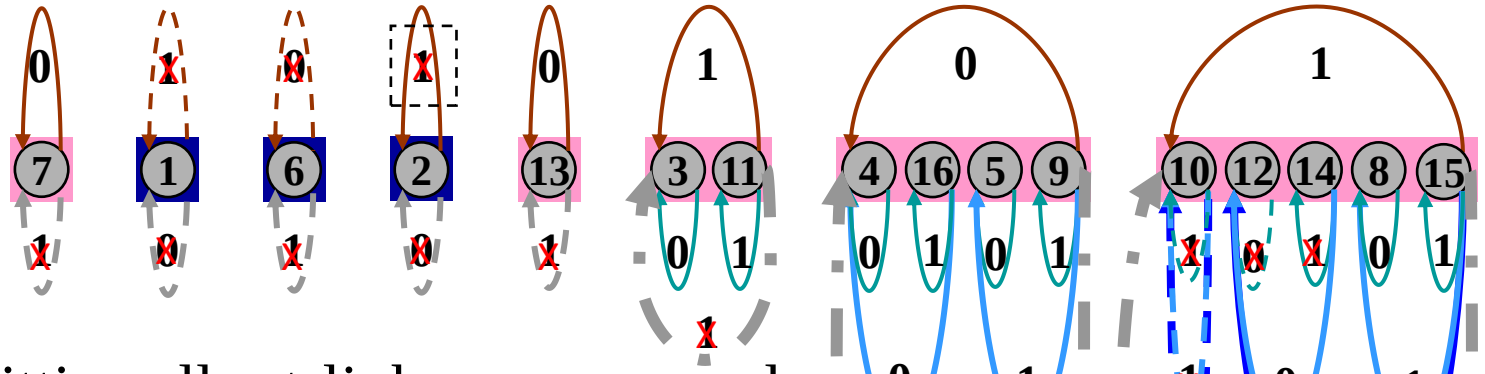
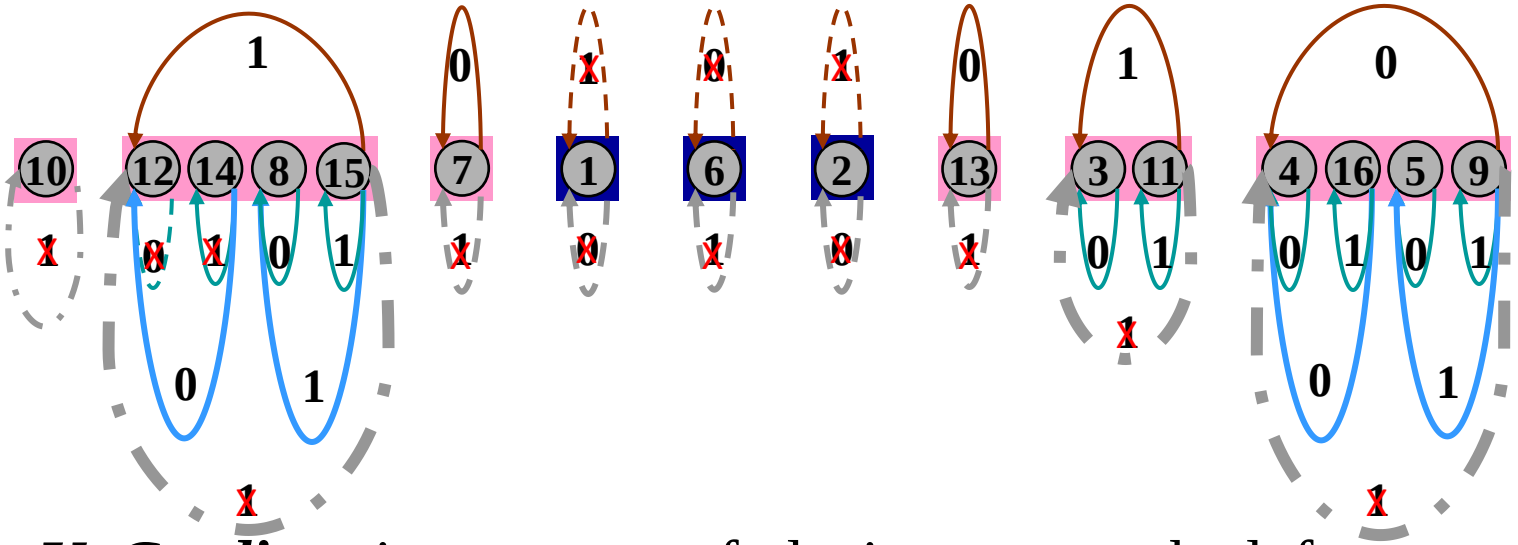


Figure: 39

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:

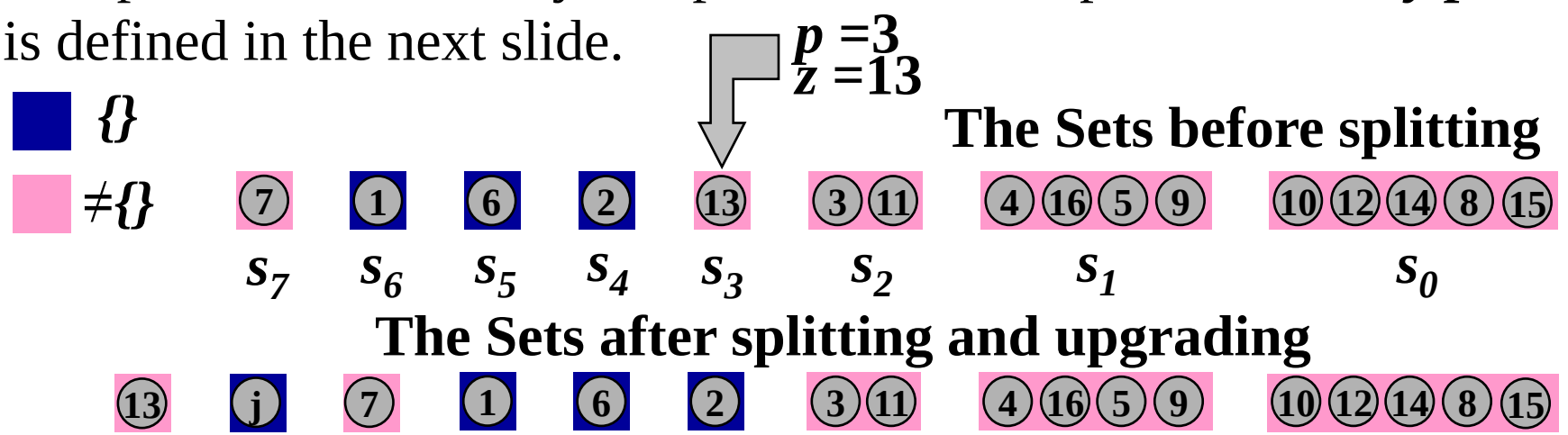
Let s_i is the parent set of the satellite block, s_p is s_i or one of its satellites:

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 ν_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. ν_r has no child. (i.e. ν_r is a leaf node).

If node ν_r is in a singleton without satellite then remove the singleton and ν_r. If node ν_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***.

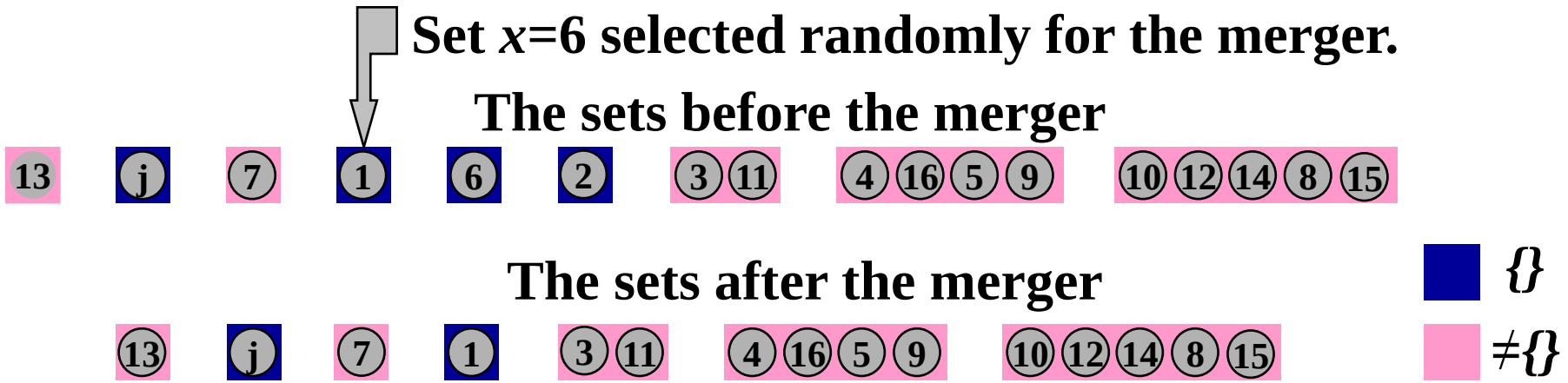
To satisfy note **3**, the process of pruning scans the ***OEDT*** from right to left to locate the lowest frequency of occurrence node v_r and its corresponding word w_r for pruning.

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.

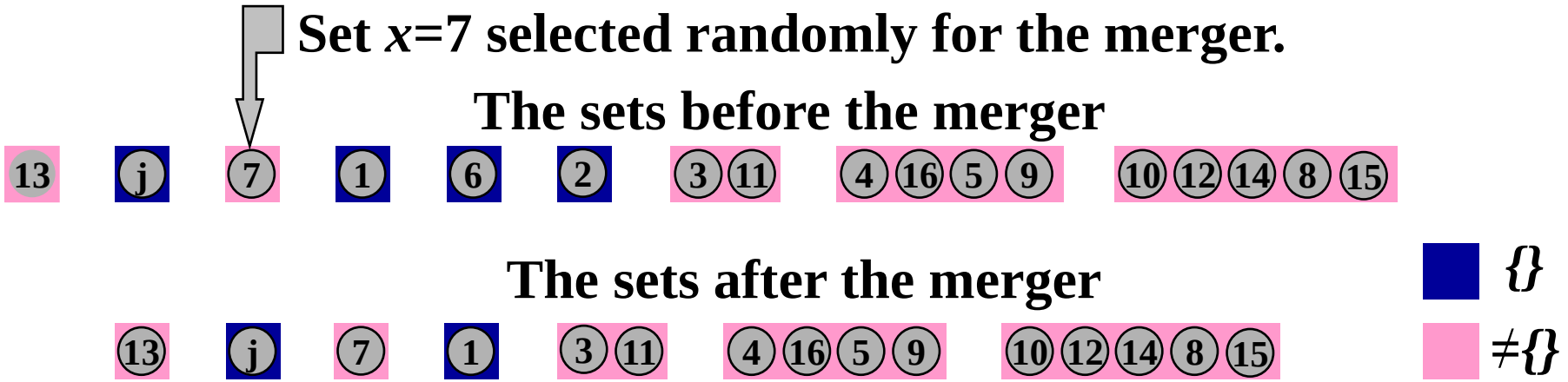
2.i If the set $s_x = \{\}$, then delete ($FBL_i / 2$) rightmost satellites of its family block. Update rank of the family sets. In the example shown below $x = 6$.



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.

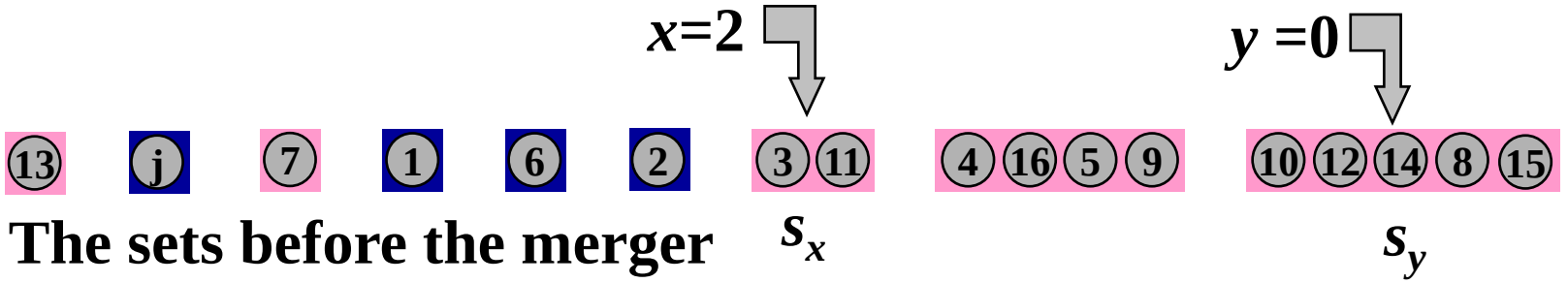
Merging (continued)

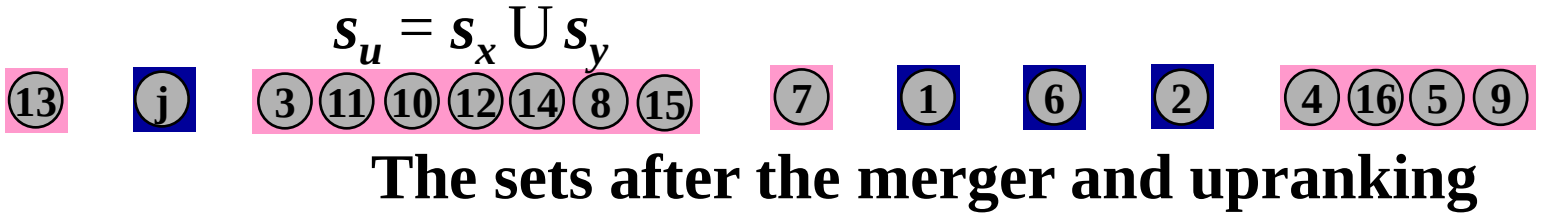
2.iv. If the set $s_x \neq \{\}$ and has no satellite set, select another set say s_y randomly. If the set $s_y \neq \{\}$ and s_y has satellite sets, then delete (FBL_i / 2) rightmost satellites of its family block. Update rank of the family sets.

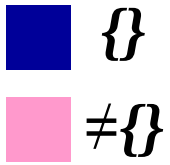
2.v If the set $s_x \neq \{\}$ and has no satellite sets, select randomly another set say s_y. If the set $s_y \neq \{\}$ and also has no satellite sets, then merge the two sets to form a new set equal to the union of the two sets s_u. If $|s_y| > |s_x|$, then $s_u = s_x \cup s_y$. Otherwise $s_u = s_y \cup s_x$. 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.:



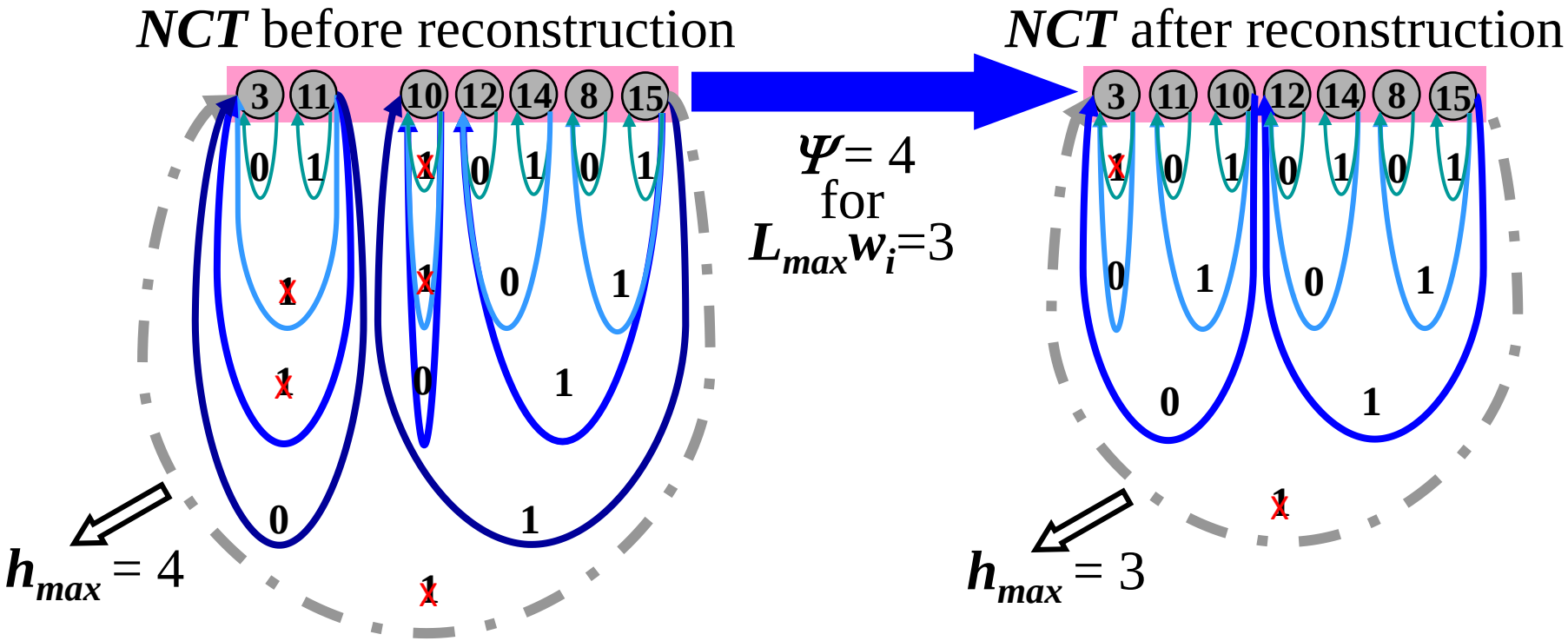




Definitions:

***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 $\Delta = (h_{y:max} - h_{x:max})$.

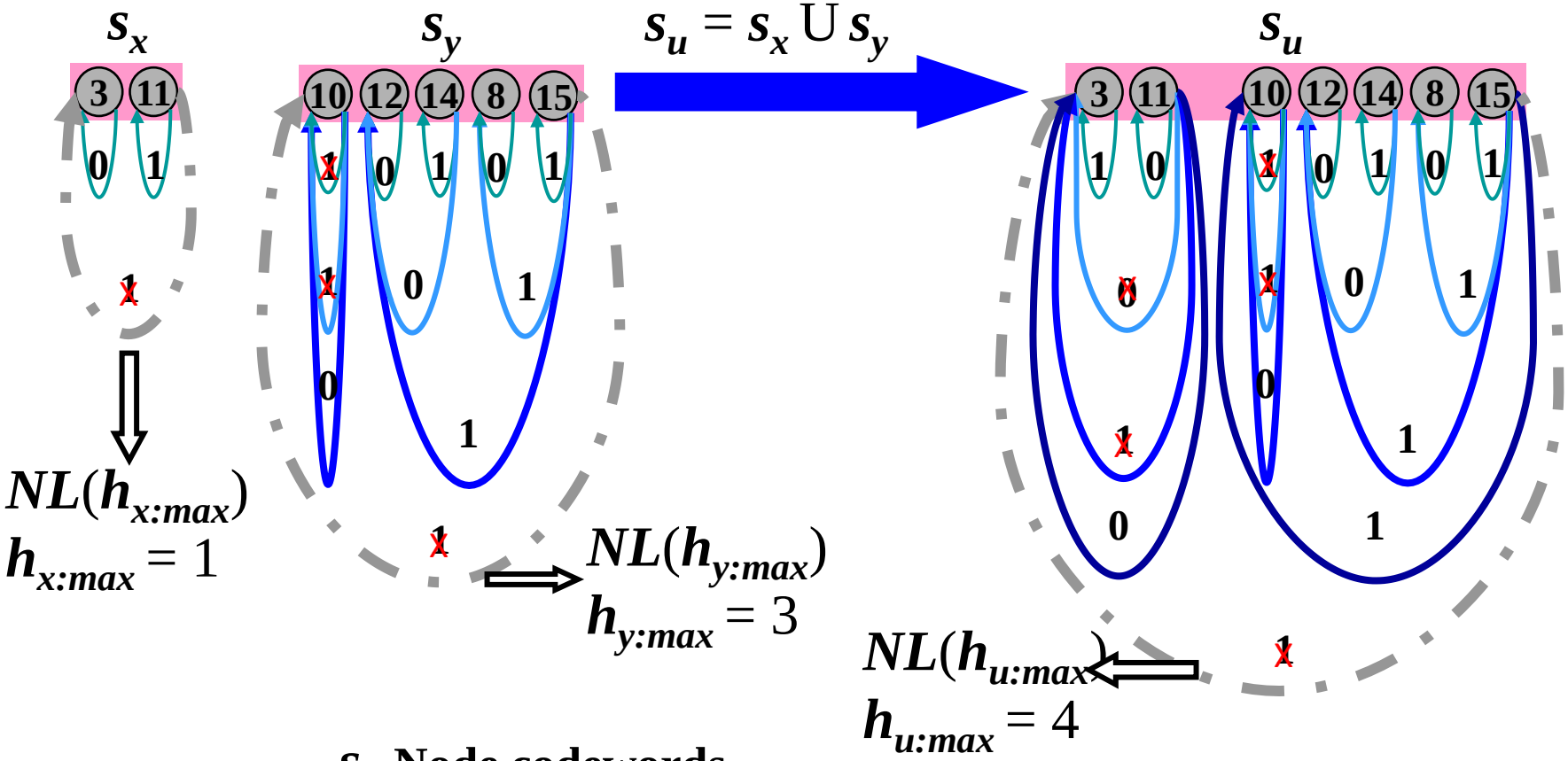
1. $NL(h_{x:max}+j)$, where j=0, 1, …, Δ, containing all nodes of s_x.
2. $NL(h_{u:max}-1)_0$ containing all node of s_y.
3. The root node-link $NL(h_{u:max})$ containing all nodes of s_u, $h_{u:max} = h_{y:max} + 1$.

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 $\Delta = (h_{y:max} - h_{x:max})$.

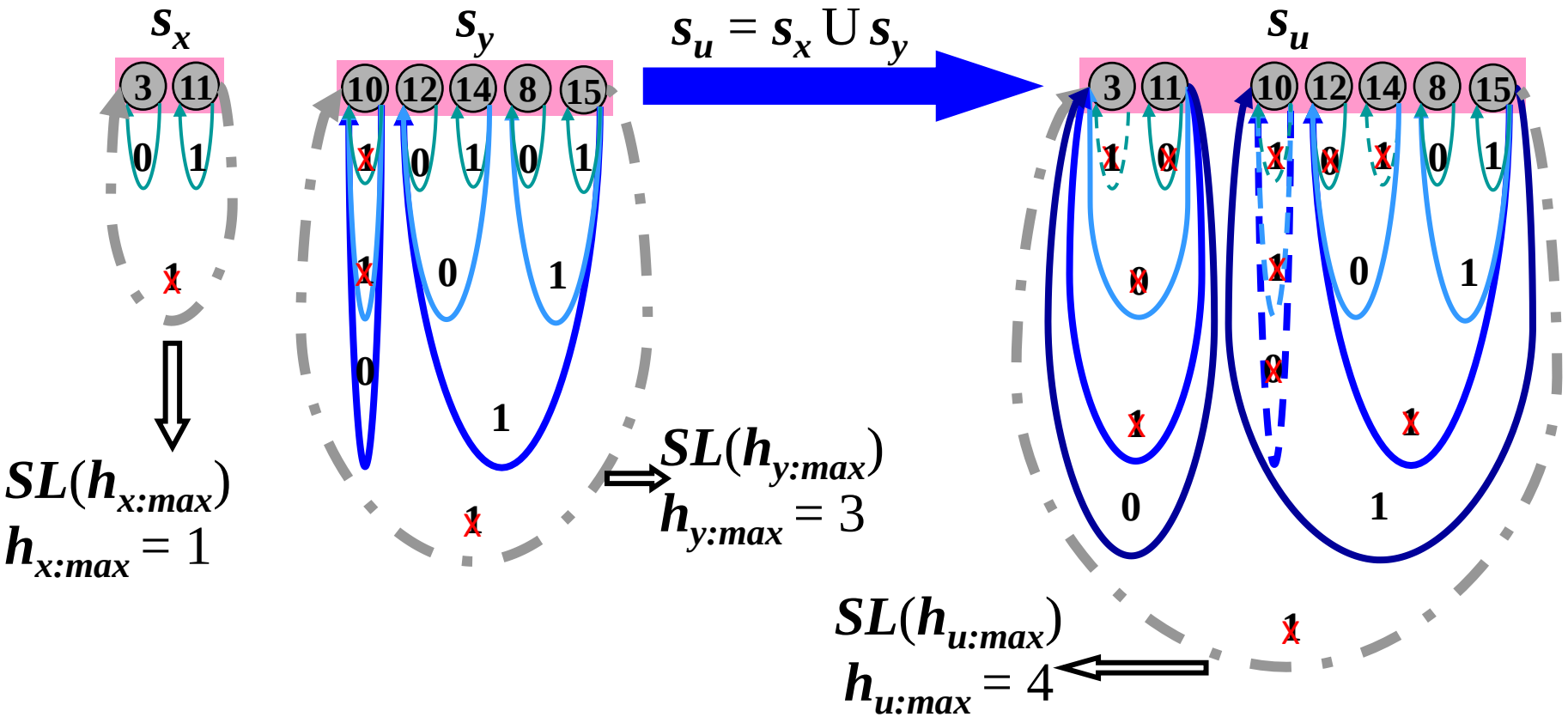
1. $SL(h_{x:max}+j)$, where j=0, 1, …, Δ, containing all nodes of s_x.
2. $SL(h_{u:max}-1)_0$ containing all node of s_y.
3. The root synch-link $SL(h_{u:max})$ containing all nodes of s_u, $h_{u:max} = h_{y:max} + 1$.

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

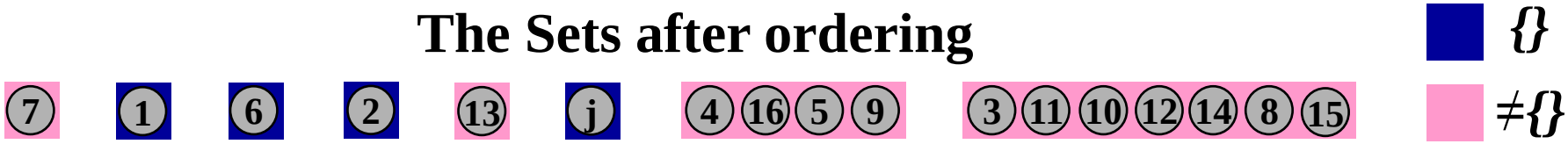
Node 8: 110 **Node12: 10** **Node11: 0**

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.

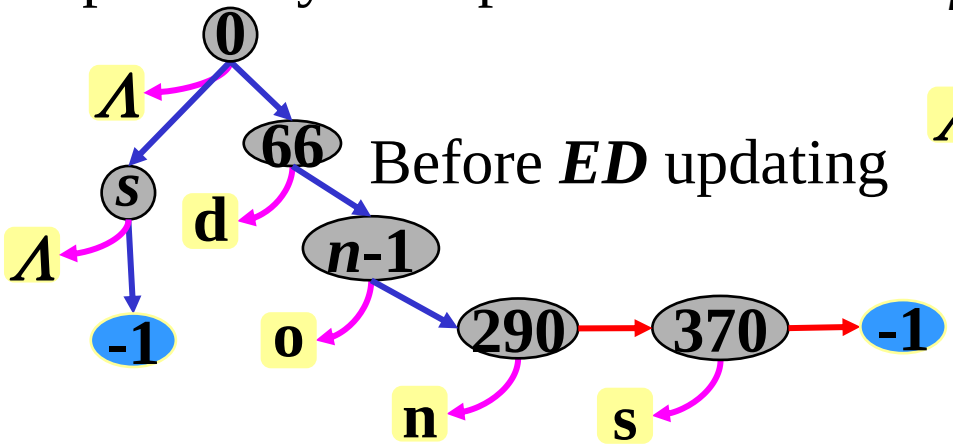
2. Replace the parent node of ν_s by ν_{n-1} in the ***OEDT***.

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

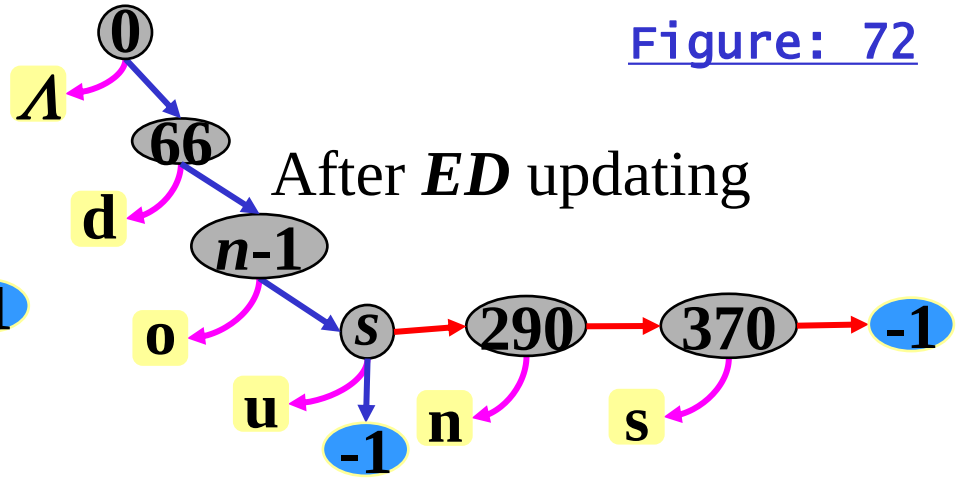
4. Replace the child node of ν_{n-1} by ν_s in the ***OEDT***.

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 ($|s_{OP}|$).

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

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.

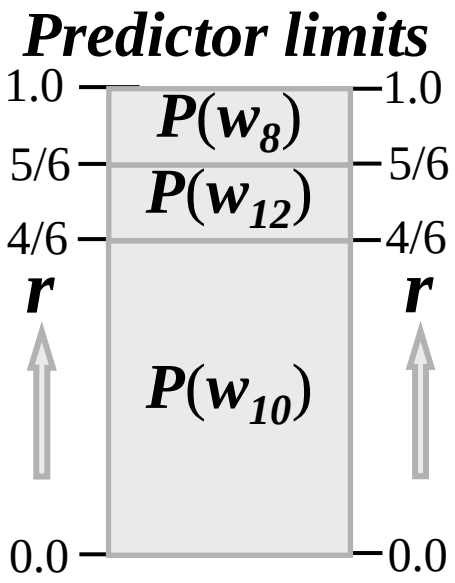
***S&MC predictor* (continued):**

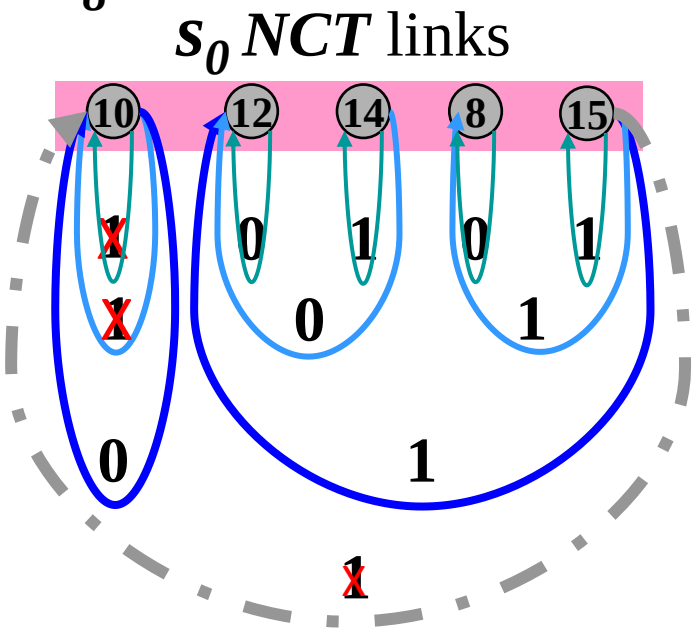
If the decoder identifies node v_8, then w_8, w_{12} and w_{10} are the input words to the predictor from the ***NCT*** links. Their probabilities may be estimated as $P(w_8) = 1/6$, $P(w_{12}) = 1/6$ and $P(w_{10}) = 4/6$.

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

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

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: $s_1 = \{v_{10}, v_{12}, v_{14}, v_8, v_{15}\}$, with $Lw_8 = Lw_{12} = Lw_{10} = 3$, $Lw_{14} = 1$ and v_{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***). $|\boldsymbol{W_i}| = \boldsymbol{1, 2, \ldots, |W_i|_{max}}$, where $|\boldsymbol{W_i}|_{\boldsymbol{max}}$ 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