CHREST Tutorial

Learning Mechanisms

Overview

- How is CHREST's long-term memory, the *discrimination network*, constructed?
  - What is a discrimination network?
  - *Recognition* processes
  - Learning through *familiarisation* and *discrimination*
  - Time parameters for all processes
  - Lateral links: linking nodes in the network
  - Templates

Long-Term Memory (LTM)

In CHREST:

- LTM is a *pool* of symbolic information
  - consists of production rules, templates, chunks, associations between chunks, etc
- LTM has an *index* to efficiently retrieve information from the pool
- the index is assumed to be a *discrimination network*
- the index is *constructed* by exposing the model to many examples from the domain
- an index of 300,000 nodes is typical for an expert

It is the discrimination-network index that gives CHREST model their distinctive nature

Efficient Indexing

- The human recognition system is very efficient: an object will be recognised as familiar and usually identified in a fraction of a second
- e.g. we 'instantly' recognise a familiar face
- Efficient indexing algorithms needed in other domains, e.g. dictionary algorithms
- The trie data structure (Fredkin, 1962) is an almost identical structure to that used inEPAM/CHREST, and today is found in yourmobile phones, for text prediction as you type

Symbolic Modelling

- Patterns are represented in the same style for objects *outside the mode*l and for objects *inside the model*
- Patterns are symbolic, as they are meaningful
- Patterns that are familiar to the model (that is, are stored within the LTM) are known as *chunks*
- Because patterns are symbolic, CHREST is an example of a *symbolic cognitive architecture*

Patterns

- All patterns are assumed to be *composite* objects, e.g.
  - a word is made up from a list of letters
  - a sentence is made up from a list of words
  - a chess position is made up from pieces on squares
- We represent the patterns as lists:
  - < w o r d >
  - < this is a sentence >
  - < <t h i s> <i s> <a> <s e n t e n c e> >
  - < [K e 1] [R f 1] [P f 2] [P g 2] [P h 2] >

End Markers for Patterns

- Sometimes we need to say ”This pattern consists of the items 'a' and 'b' and *nothing else*”
- To say 'and nothing else', we add an *end marker* to the pattern
- The end marker is some symbol that cannot otherwise appear in the pattern (denoted '$')
- e.g.
  - <a b $>
  - <this is a sentence $>

Operations on Patterns

- In our learning algorithm, we will need to manipulate patterns:
  - test two patterns to see if they *match* or are *equal*
  - *remove* one pattern from a second
- Examples of matching:
  - <a b c> matches <a b c d>
  - <a $> does not match <a b c>
  - <a b c> does not match <a c b>
- Two patterns are *equal* if they contain exactly the same items (including the end marker)

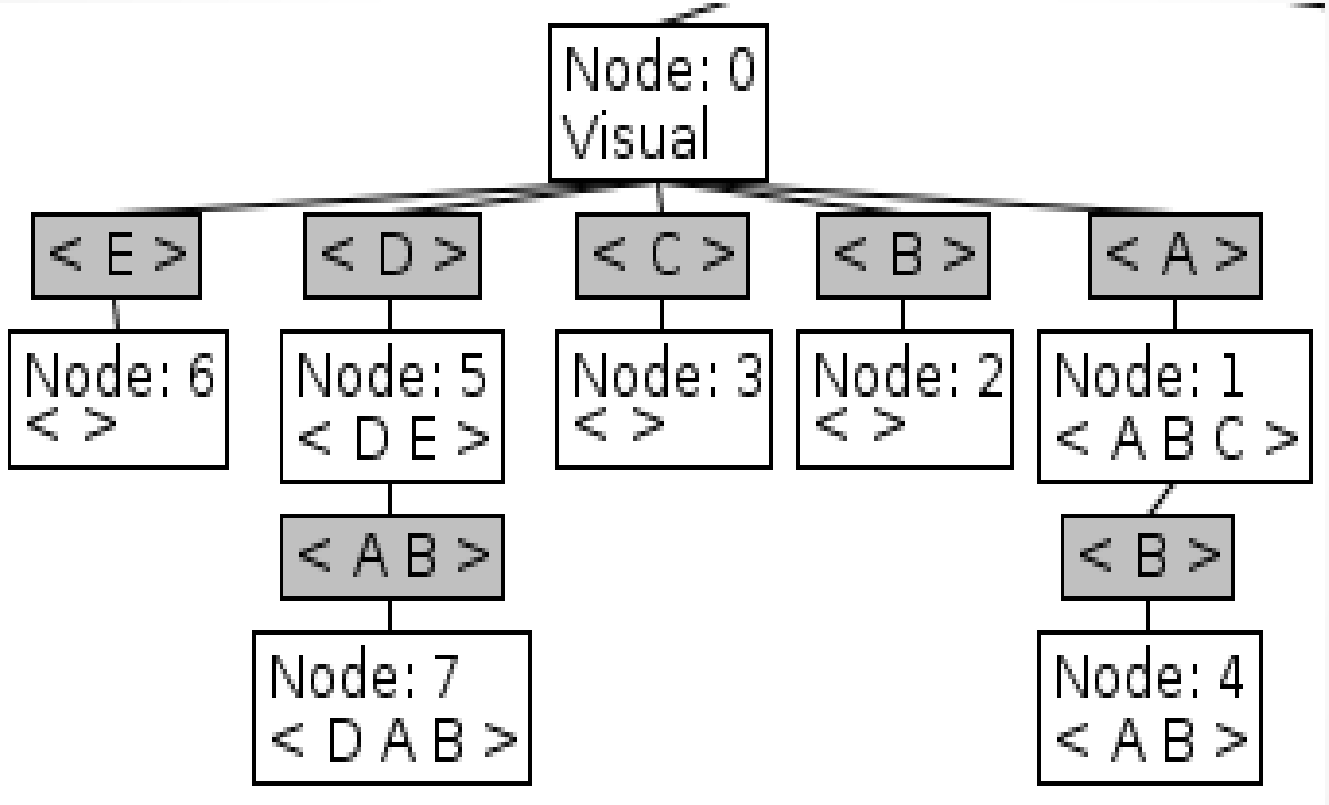
Removing Patterns

- The difference between two non-matching patterns is the part of the *second* pattern *after* the common part of the pattern is removed
- e.g.
  - <a b c> taken from <a b d c> = < d c >
  - <a b d c> taken from <a b c> = < c >
  - <a b c d> taken from <a c b d $> = < c b d $>

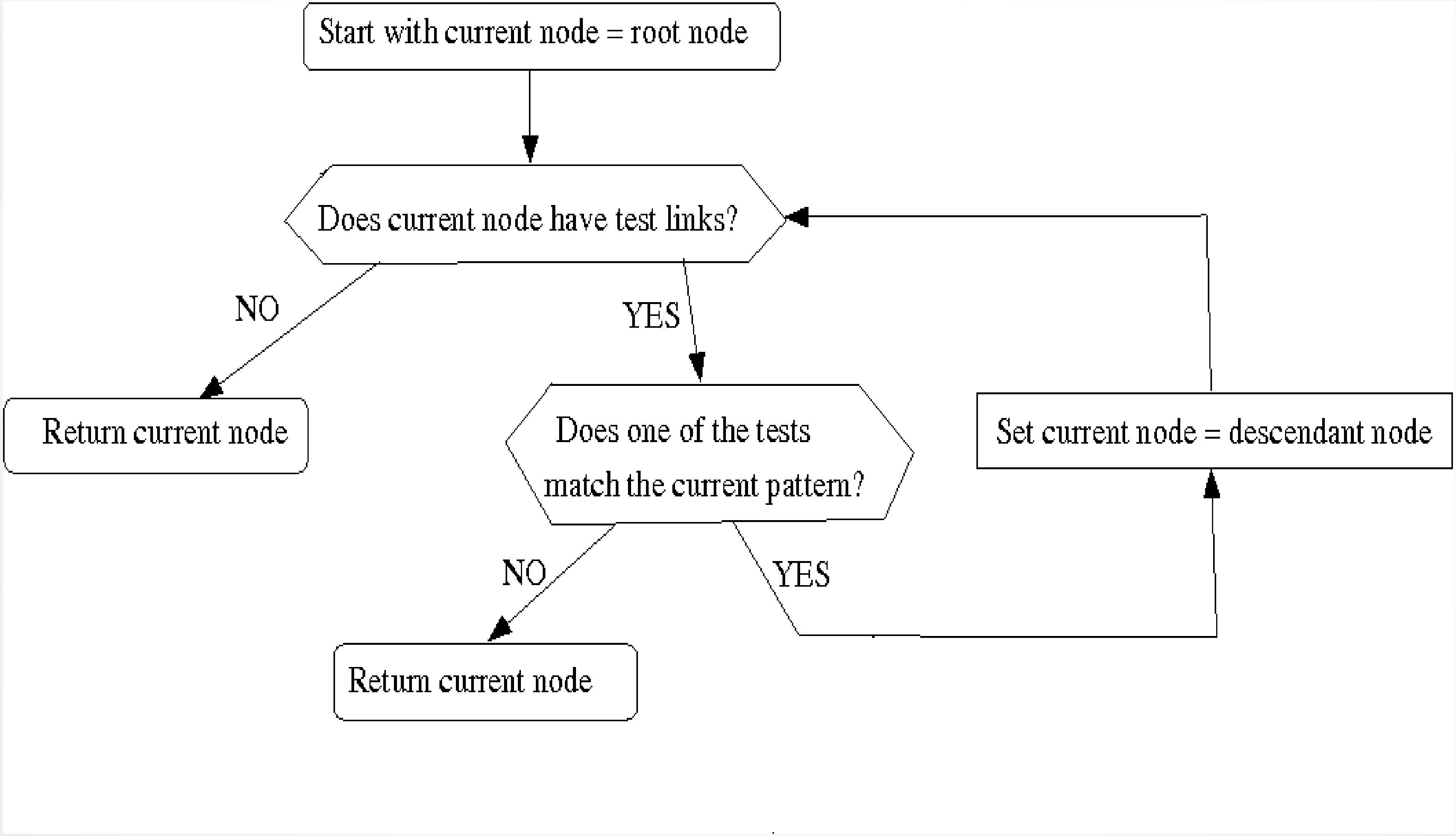
Discrimination Network: Nodes

- A network consists of *nodes* connected by test links
- The test links contain a pattern which is checked against the current input during retrieval
- The nodes contain *images*, which are the familiarised parts of patterns
    - images may be bigger, smaller or equal to the tests required to reach the node
- The series of tests to reach a node forms the *extrinsic description* of a chunk
- The image within the node forms the *intrinsic description*

Discrimination Network



Retrieval Process


Start with current node = root node
Does current node have test links?
NO
YES
Return current node
Does one of the tests match the current pattern?
Set current node = descendant node
NO
YES
Return current node

Learning Process

(1) The pattern is sorted to a node

(2) The pattern is compared with the image of the node reached

(3) If the image *matches* the pattern, then *familiarisation* occurs

(4) If the image *mismatches* the pattern, or the node is the rootnode, then *discrimination* occurs

Discrimination Process

- Discrimination is how *new nodes* get added to the network
- We take the part of the input pattern not used so far in sorting (the *difference pattern*), and attempt to sort it through the network

  1. If we stay at the root, then learn a new primitive
  2. Get the image of the retrieved node:
     1. If it is empty, familiarise with the difference pattern
     2. Else, make a new test link:

        the test is the retrieved image (without any end marker)

        the image of the new node is the set of tests needed

        to reach it

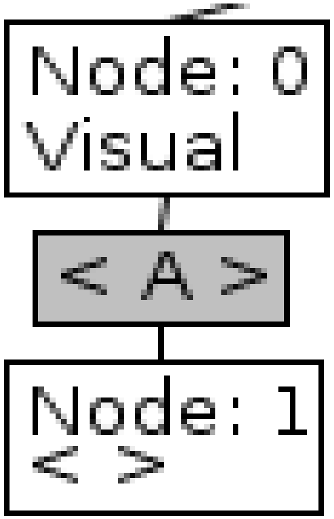
Familiarisation Process

- Familiarisation is how *new information* is added to the chunks in the network
- We take the difference between the image of the retrieved node and the input pattern and sort it through the network
- Three things may occur:

1. We get the root node back, then we have to learn a new primitive, using discrimination
2. If retrieved image is empty or larger than difference, we just add first item from difference to (first) image
3. Otherwise, add the first item from retrieved image to the first image

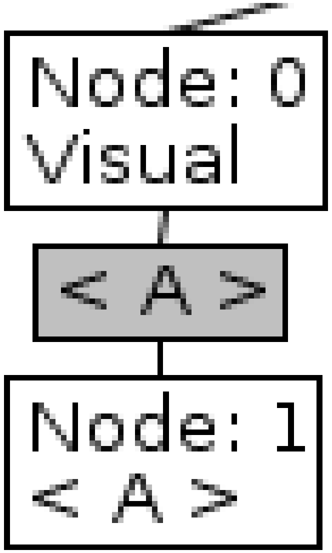
Learning Example (1)

- First presentation of <a b c $>



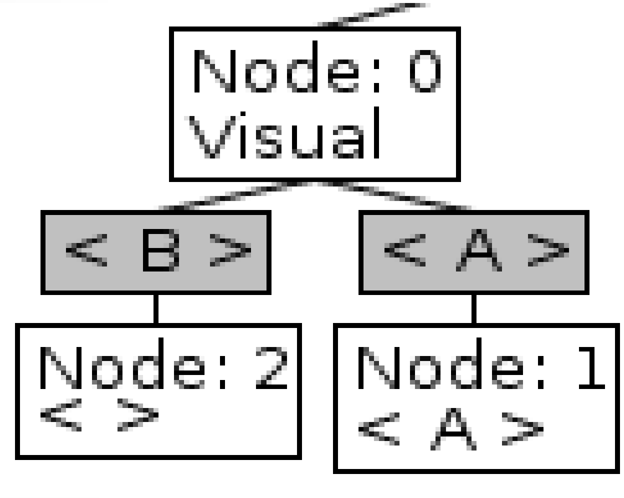
Learning Example (2)

- Second presentation of <a b c $>



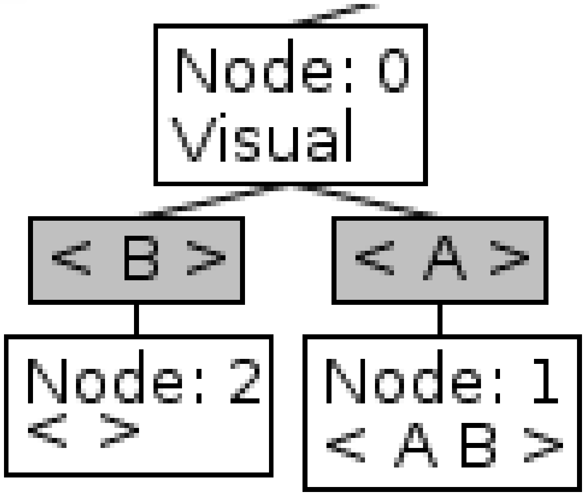
Learning Example (3)

- Third presentation of <a b c $>



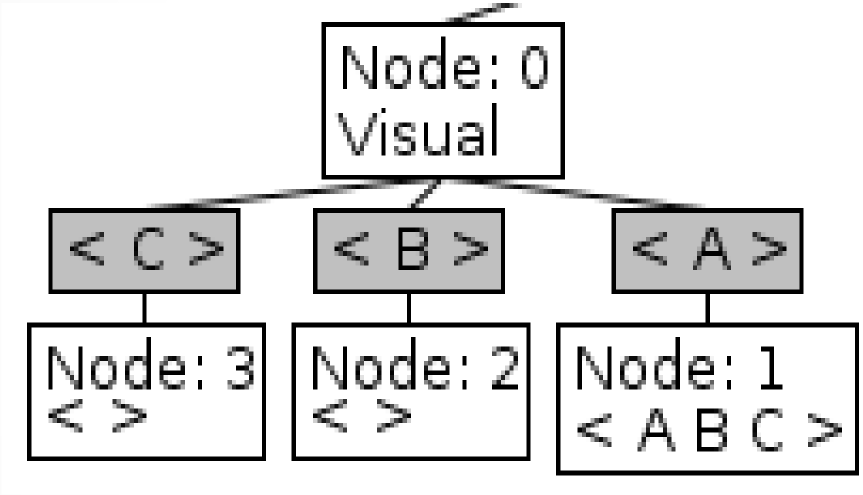
Learning Example (4)

- Fourth presentation of <a b c $>



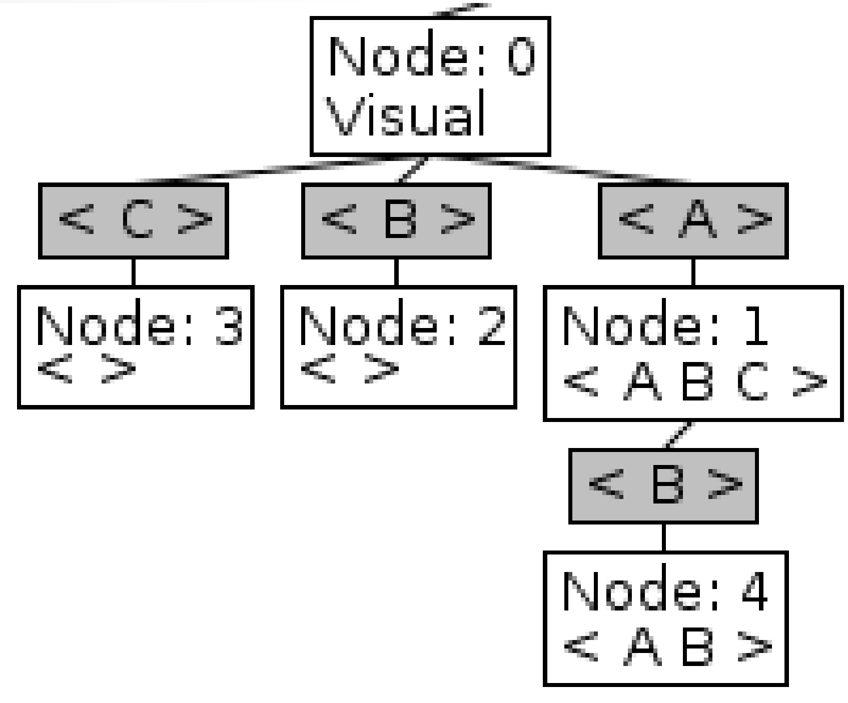
Learning Example (5)

- ... Sixth presentation of <a b c $>



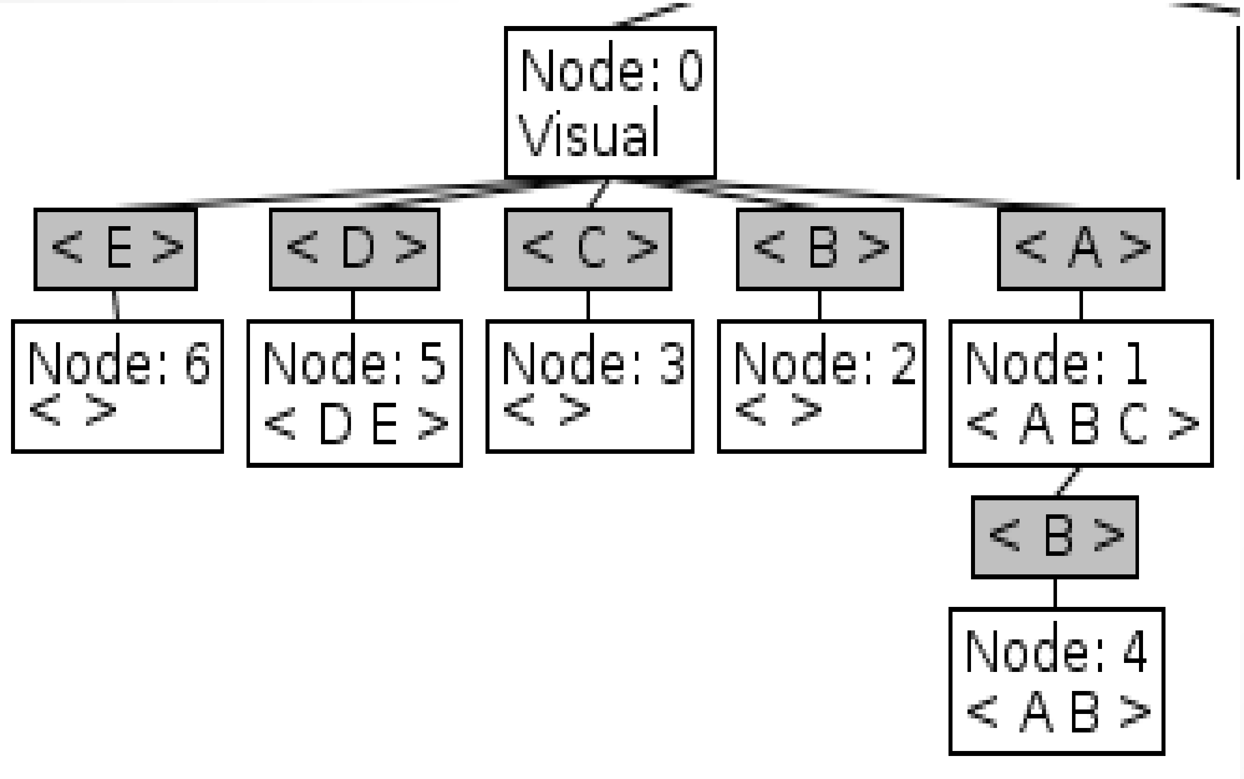
Learning Example (6)

- Presentation of <a b $>



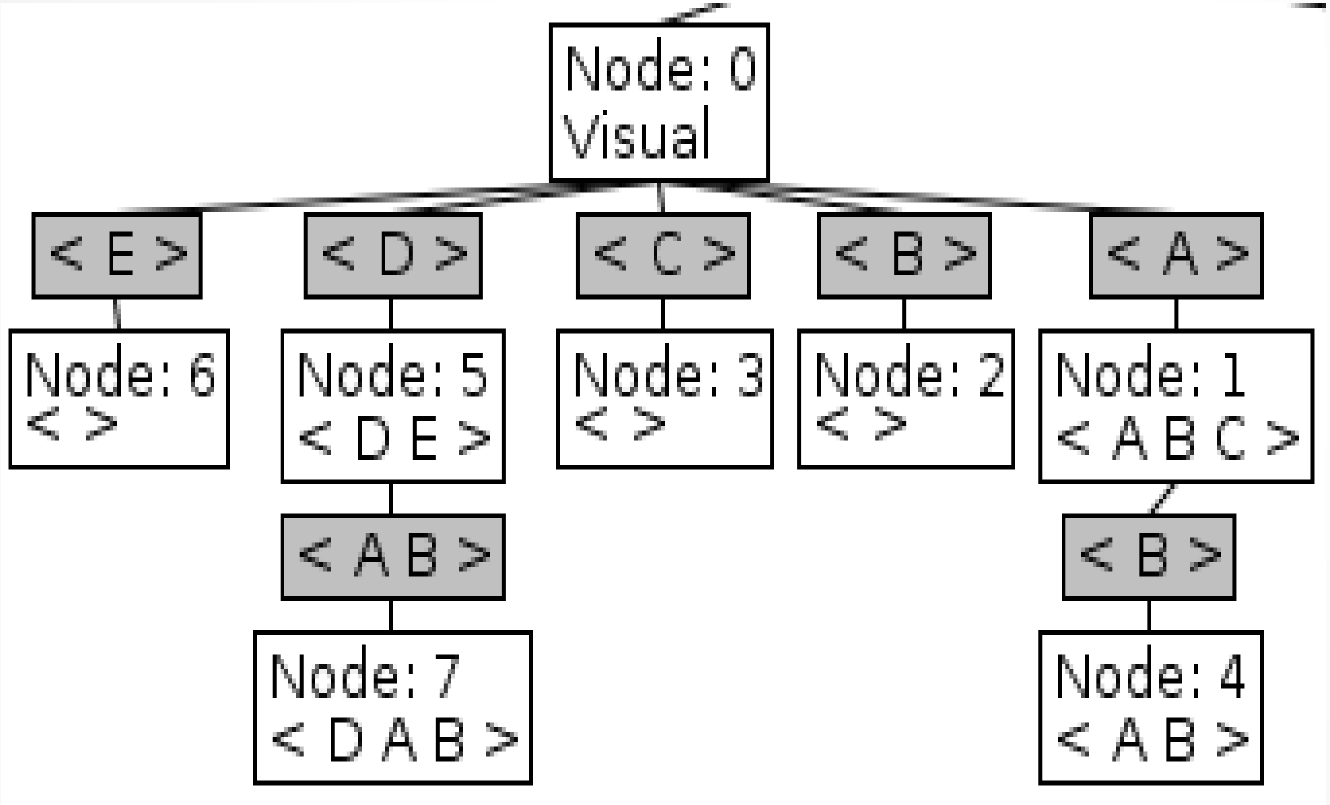
Learning Example (7)

- Presentation of <d e f $> four times



Learning Example (8)

- Presentation of <d a b $>



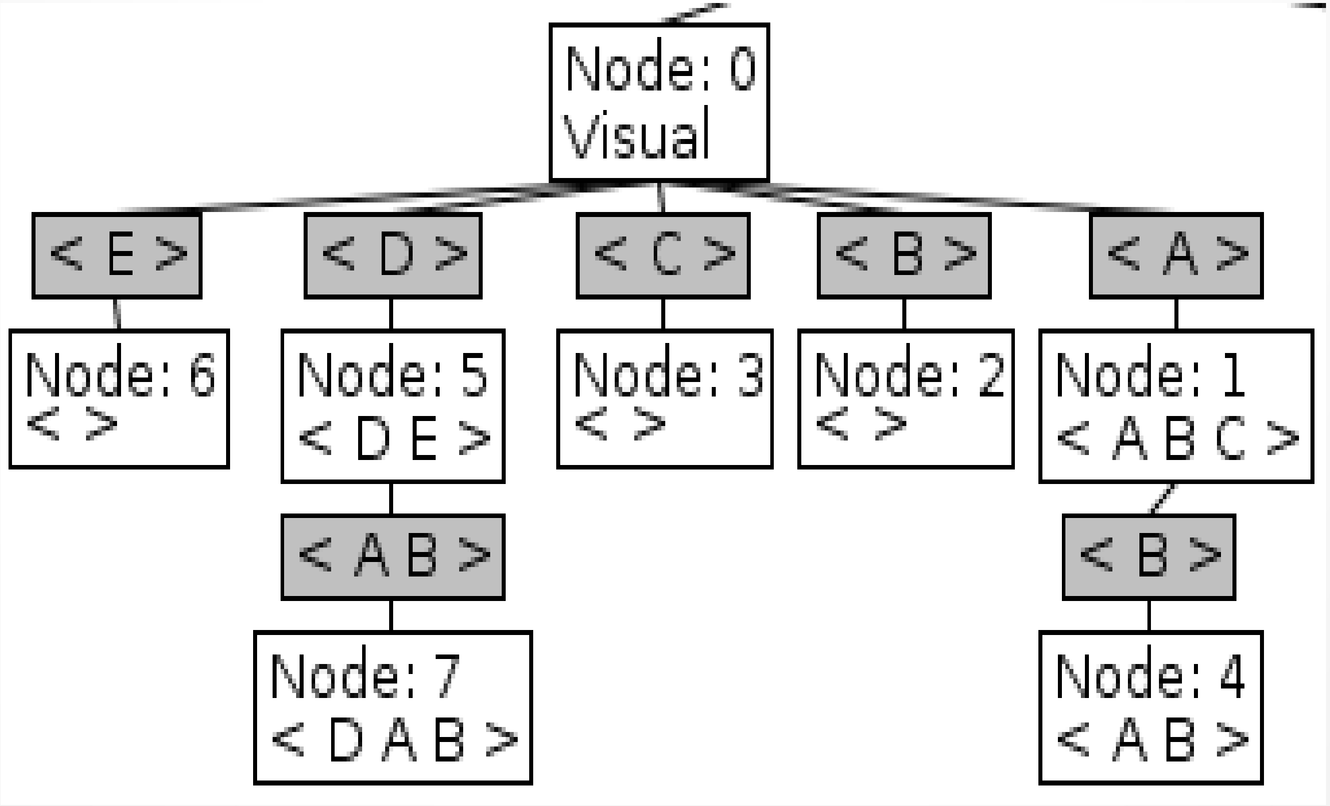
Time Parameters

- CHREST assumes (like EPAM) that every process we have described takes a physical amount of time
  - discrimination, 8-10 seconds
  - familiarisation, 1-2 seconds
  - testing a pattern, 50 milli-seconds

Discrimination 'Tree'

- So far, what we have studied are discrimination trees

  recognition starts from the top, following branches, until reaching a leaf node



Discrimination Network

- A 'network' is a structure where nodes are interconnected by *links*
  - A 'tree' is a hierarchical network where each node has a unique parent.
  - More generally, nodes may be reachable from more than one path
- Note that CHREST networks have a unique root node, and are also *directed*, as recognition proceeds down the tree
- Now we look at how nodes may be linked to other nodes in ways other than *test links*

Types of Lateral Link

- *Similarity* link: formed between two nodes which are sufficiently 'similar'
- *Sequence* link: formed when the pattern in one node typically follows that in the first
- *Cross-modal* link: formed when the pattern in one node occurs at the same time as another
  - *Naming* link: between a visual and verbal pattern
  - *Production* link: between a pattern and action
- *Generative* link: formed when the preceding and succeeding tests at two nodes overlap

Formation is triggered by the short-term memory

Short-Term Memory

- The short-term memory (STM) is a capacity-limited storage space
  - It is *temporary*, as later information will remove earlier information
  - It holds *pointers* to nodes in LTM
  - The agent may use a *strategy* to, for example, maintain the most informative nodes in STM
- Information from nodes in STM can be unpacked into a 'mind's eye', and used for further learning or driving attention
- Different STMs exist for each *modality*

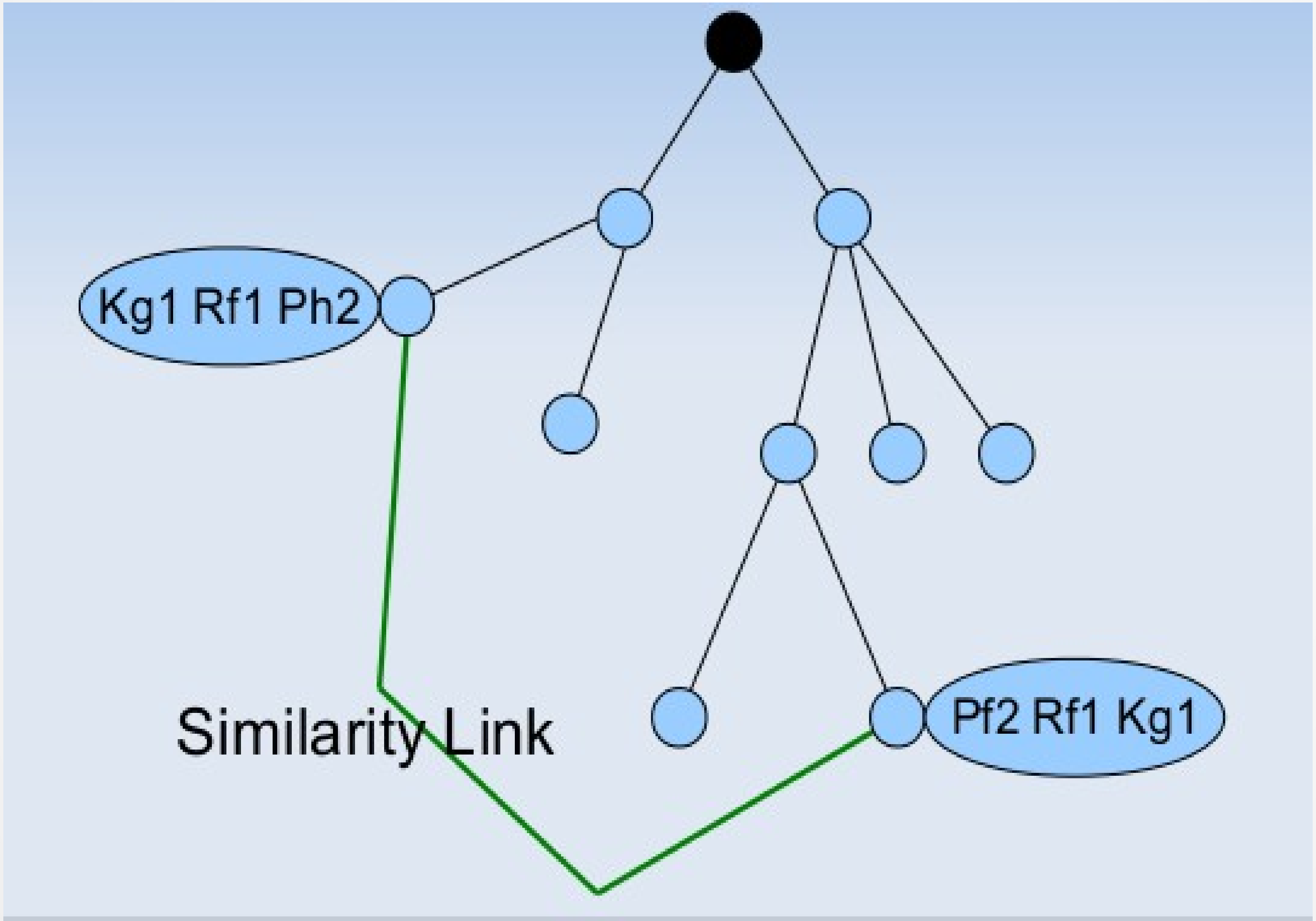
Modality

- Modality refers to the type of pattern (from the word mode, a particular form or variety)
- CHREST supports three types of modality:
    - Visual: what is *seen*
    - Verbal: what is *heard*
    - Action: what is *done*

Similarity Links

- A *similarity link* is used to indicate that two nodes are more-or-less the same, e.g.
    - The lists <a b c d e> and <a b g d e>
    - Or, <a b c d e> and <e d c b a>
- The learning process considers the image of the two nodes, and compares the items within the two
    - A similarity link is formed when a certain percentage of overlap is achieved
- During retrieval, a descendant link from any of the connected nodes can be taken

Example of a Similarity Link



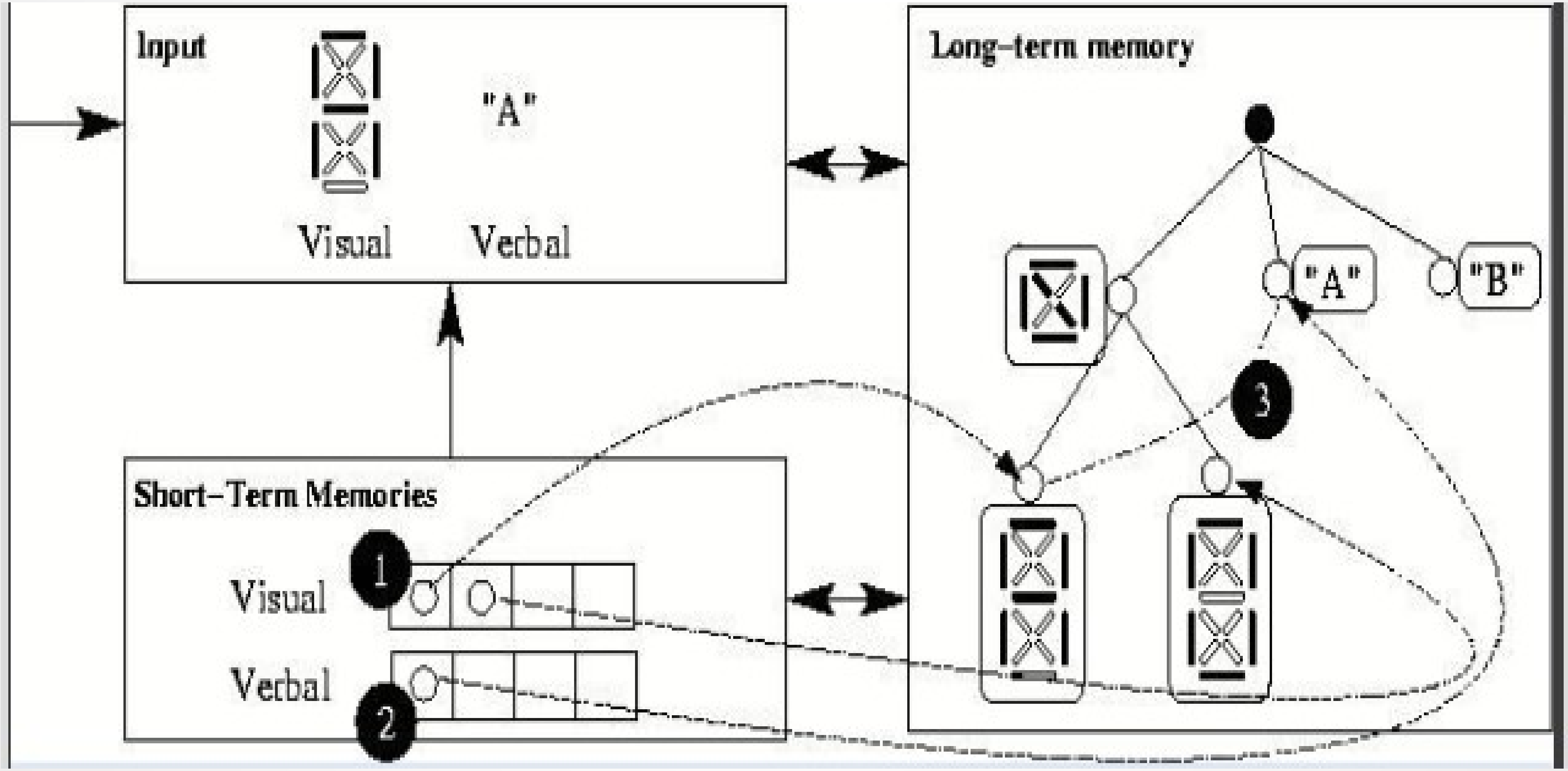
Sequence Links

- A *sequence link* is used to indicate that two nodes have been observed to follow each other in time, e.g.
  - When learning a list of words
- The learning process
  - Considers the top two nodes of any short-term memory
  - Links the top node as having followed the second node
- In recognition, a sequence link may be followed to *predict* which pattern(s) may come next

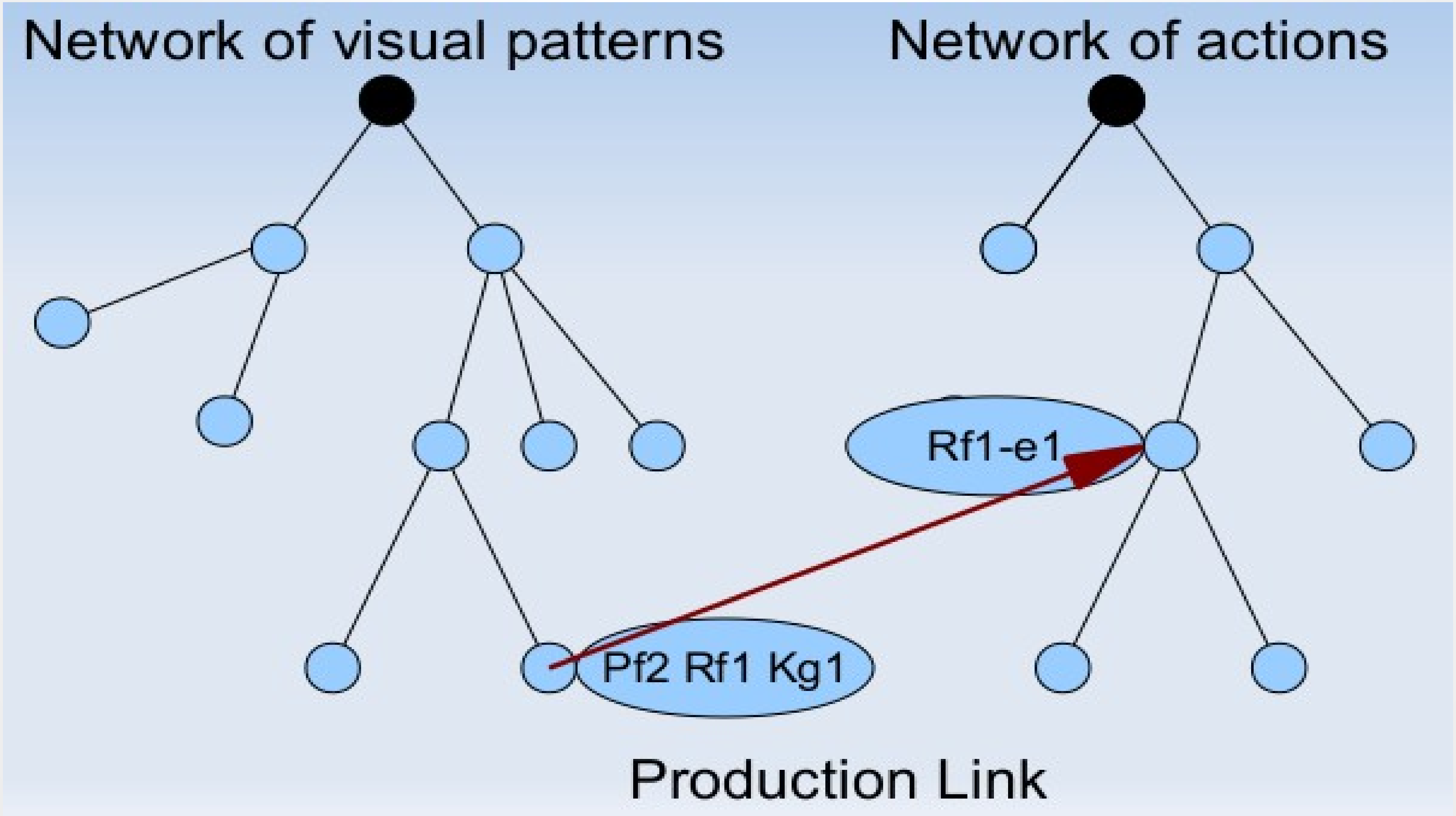
Cross-Modal Links

- Cross-modal links are links between two nodes of *different* modality
- They are formed when two nodes co-occur on different STMs
  - Links between visual and verbal nodes are typically called *naming links*
  - Links between visual or verbal nodes and action nodes are typically called *production links*
- During retrieval, the links may be used to name an object, or propose an action, as appropriate

Example of a Naming Link



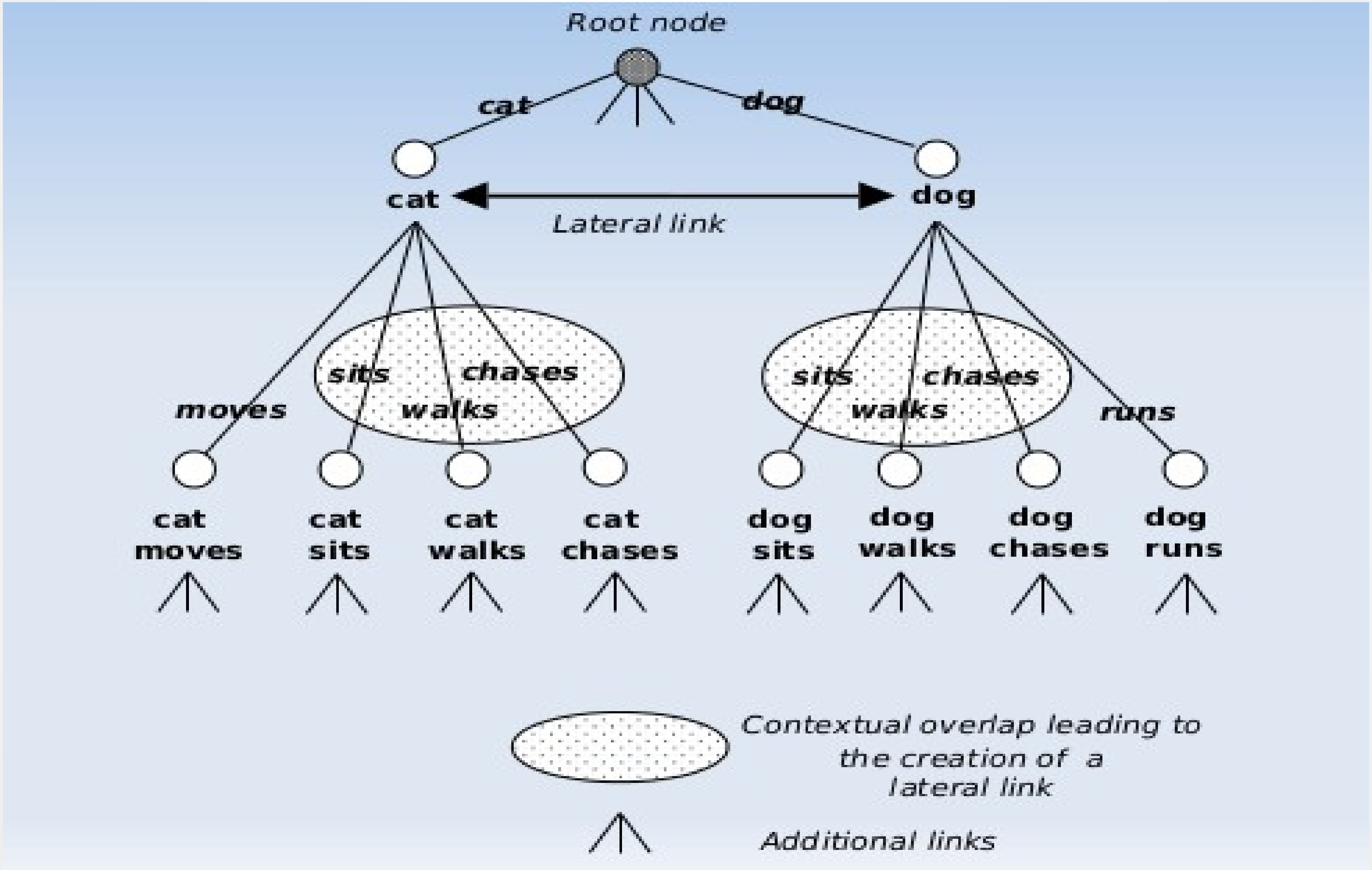
Example of a Production Link



Generative Links

- A generative link is used to generalise across different lists, e.g.
  - From ”The cat sat on the mat”
  - And ”The dog sat on the rug”
  - We could say: ”The cat sat on the rug” ...
- The learning process considers the set of nodes before and after the given nodes, and forms a link if 10% of the nodes are the same
- The links are used to generate output, by tracing down through the network

Example of a Generative Link



Core and Changes

- So far, what CHREST does is capture *what stays the same*
  - Which chunks have I seen before also apply in this situation?
- However, nothing is ever simply composed of chunks of previous events
- There may be large similarities, but there will also be small differences

Programming Expertise

For example, a programmer will not have a problem understanding either of:

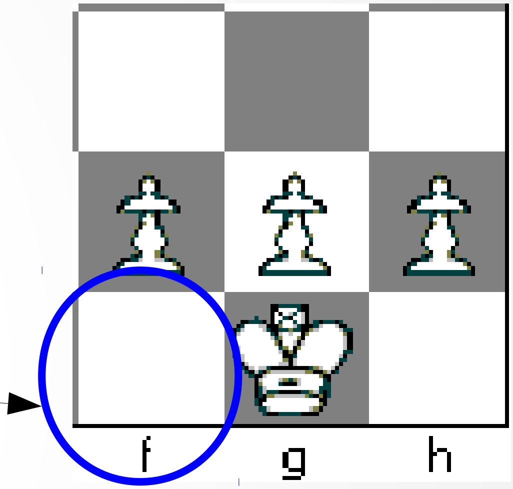
```
for (int i = 0, max = 100; i < max; ++i) {
// do some stuff
}


for (int q = 0, max = 100; q < max; ++q) {
// do some stuff with non-standard variable name
}
```

Templates

- To capture the highest level of memory, CHREST includes an additional knowledge representation structure which captures this idea of *mostly the same* but *some permitted changes*
- We call this a *template*:
  - core elements are the same
  - slots hold variable elements (either item or location)
  - slots can be filled quickly with perceived information, without having to learn the contents separately

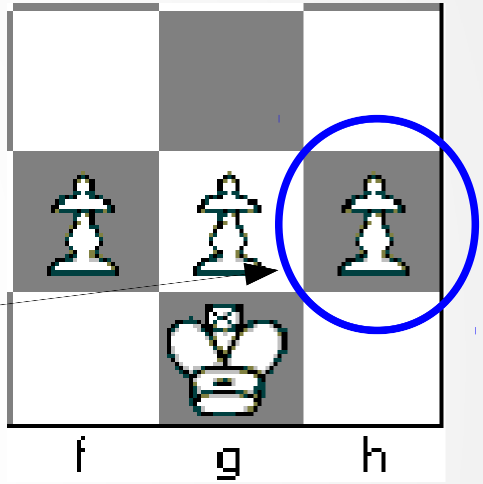
What is a Template?

- Template definition:
  - pieces in diagram are core
  - slots could be
    - square f1, could contain R or B or nothing
    - Ph2 could be on h2 or h3
    - square f3 could contain N or nothing
  - related semantic information
    - ”typical castled position”
    - ”secure pawn structure”
    - ”Kg1-f1” typical move in endgame



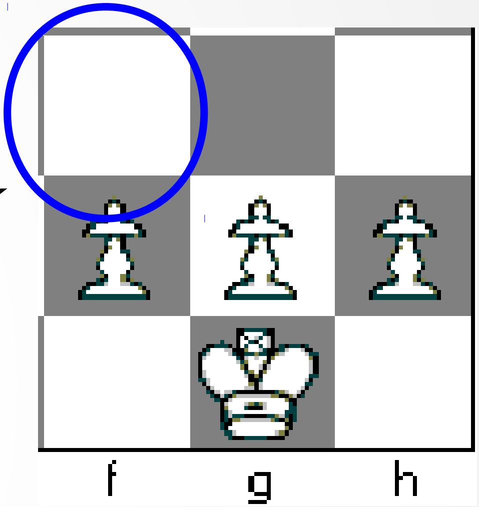
What is a Template?

- Template definition:
  - pieces in diagram are core
  - slots could be
    - square f1, could contain R or B or nothing
    - Ph2 could be on h2 or h3
    - square f3 could contain N or nothing
  - related semantic information
    - ”typical castled position”
    - ”secure pawn structure”
    - ”Kg1-f1” typical move in endgame



What is a Template?

- Template definition:
  - pieces in diagram are core
  - slots could be
    - square f1, could contain R or B or nothing
    - Ph2 could be on h2 or h3
    - square f3 could contain N or nothing
  - related semantic information
    - ”typical castled position”
    - ”secure pawn structure”
    - ”Kg1-f1” typical move in endgame



How to Use a Template?

- A template is stored in LTM and accessed using the discrimination network, just like any other chunk
- When the template is accessed in STM, then information from the perceived scene can quickly be loaded into the slots (~250 ms)
- This information is preserved whilst the template is 'live' (usually, within STM)
- Hence, templates allow rapid memorisation of the novel/variable features of a situation

How to Learn a Template?

- The idea of a template is rather like that of a generative link: it generalises chunks across context
- The context for a template is the linked set of chunks from a given node
  - child test links
  - similar nodes
- Form a template on current node (in STM) if:
  - at least n pieces are in common across all linked nodes
  - the current node becomes the core
  - use the varying information to create *slots*

Summary

This section has covered the main learning mechanisms within CHREST, including

- Creating a discrimination network
- Lateral links
- Templates

Extended descriptions can be found in the papers.