Creating General Purpose AI

M. Sebastian
df84sadw@gmail.com

December 31, 2016

Chapter 1

Introduction

In this work we present an approach through which higher functions of the human brain such as learning, deduction, abstraction, intuition and reasoning can be emulated through a family of algorithms. We make use of layered model similar to that of a set of CNNs that allows for correlation between the outputs of the intermediate layers belonging to distinct CNNs. The CNNs are modified so that patterns have life spans depending on their use in inference and degree of utility. The learning process is gradual, starting with an initialization stage (initial blank state) used to deploy patterns and perform training, and learning stage in which, unlike in classical CNN, new CNN and patterns are added to the existing CNN set. The learning process is somewhat similar to that of a new born.

Input is generated by a set of sensors, each sensor perceiving a distinct type of data (e.g. visual, auditive). For simplicity, we only consider a single sensor for each type of input data and only two types of data, visual and auditive, in this work. In the remainder of this section we will give an intuitive overview of how input is processed and modelled by our algorithm .

Modelling Sensory Input

Visual Input

In order to deal with complexity, we make use of a predefined set of shapes each defining a category in which the observed object can be classified into. Each object is perceivable if its edges can be fitted into these shapes or a combination of them; geometric transformations are also applied over the shapes such as scalling, rotating, etc. in order to provide a better matching over the input data.

In essence, a matching algorithm that fits the observed object from the sensor with an optimal number of shapes is needed. Optimal is a function

of the available computational/storage resources. In other words, our machine[1] can see into greater detail if it can process more shapes pertinent to the observed object.

At the time of this writing, the state of the art in visual pattern recognition is achieved by GoogleNet as described in [1]. Our model aims to split the CNN and retrieve outputs from hidden layers and apply correlation functions over the respective outputs and a set of distinct outputs retrieved from other CNNs. The result of the correlation can be the input to some CNN depending on the type of classification it performs as well as its performance. The idea is to always use the CNN that has retrieved optimal classification results for any stage of the output of the hidden layers.

<<look-up pattern recognition algorithms and pick a set of algorithms that have been proven to work well in practice. Look at approaches both in the geometric and the frequency domain>>

Auditive Input

Sound is processed as a contiguous signal. A correlation between two sets of sounds is made w.r.t. the variance between each other. For example, if our robot "hears" multiple combustion engines that produce "similar" sounds it will group them into a set due to the lower variance between them compared to other distinct sources (e.g. a thud, brief sounds, etc.).

A subsequently perceived set of sounds that vary from the first in that they are, for example, too sharp or have significant [2] frequency/amplitude differences from engine sounds will be placed in a category of their own or a preexisting category if similar sounds have been input before.

Up to this point we have two unrelated categories of inputs provided by the two sensors (future implementations will make use of more than one sensor per input type). Next, we propose an algorithm that creates mappings between visual and auditive input sets and further uses these mappings to create definitions that are used in problem (task) solving. Definitions are detailed in Section 2.3.1 and essentially represent the internal representation of the observed object.

This mapping is determined by a set of constraints related to:

- the time when sensory data is being input
- the relevance of the mapping within a definition set (how suitable are the respective definitions in task solving)

Time Constraints

We propose an approach that creates a mapping (correlation) between data as a function of their input time. For example, if our robot sees and hears a

[1] We use the terms machine, robot, algorithm interchangeably throughout this work. Relevant differences are context defined.

[2] Throughout our work we use subjective terms due to the fact that at the time of this writing our work is empirical; the choice of algorithm families and data structures are still subject of discussion.

car at the same time a correlation will be made between the set of the shape images (pattern) matching the car and the corresponding pattern of the sound it produces. Such a correlation allows to infer the visual data from the auditive data. Note that our robot can hear different sounds than that of the engine (the most pronounced input), such as tires on the ground, environment sounds, etc., the most dominant sound (not the loudest but the longest in duration while the car is being observed) will be associated to cars (we will discuss abstraction in later sections; the robot can generalize a category of inputs to a broader set of data depending on the number of observations). It is also possible for our robot to observe only an engine that runs and produces similar sounds to a car which creates "confusion" (conflicting mapping with a previous one between a running car). An equivalence relationship is created stating that a car is an engine given the auditive input although visually they are distinct. As we will describe later, task solving involves creating mappings between patterns across input data regardless the sensor it has been perceived from.

Definition Relevance Constraints

The core principle of task solving relies on being able to create a definition for a task. A definition, as we will detail later, is a list of nodes each holding a pattern with an associated weight that shows its relevance within the definition. A task is comprised of a set of standalone data entities each having a definition or are at least definable at the time the task is input. For example, a request written in English words "What is your name?" where each word has a preexisting definition. Given the definitions of all the terms in the task, the algorithm builds a definition for the task.

Validity of Mappings

In the above paragraph on time constraints mappings can be created between objects if they have similar features (in the example a mapping between a running car and an engine). Such mappings can create equivalence relationships between the observed objects (in that a car is an engine) that are not valid (a car has distinct visual features from an engine) in task solving. In such cases the visual correlation between the car and engine is discarded but the auditive kept.

Our robot at this point knows that a started engine and car produce approximately the same sets of sound. It still cannot establish any relationship between the car and the engine such as if a car has an engine, the engine has a car, etc. There is no mapping between a car that is not running and an engine. Such a relation can be created indirectly by knowing that a car with a stopped engine looks the same (visually) as a car with a stopped engine (a correlation between the auditive input and visual input can be made).

Two approaches are possible:

1. Explicitly create the mapping and define a relationship between a car with a running engine and an engine (a car that sounds like an engine has an

engine)

2. The robot can look at the engine (by receiving visual input data from a sensor) and then at a car with a visible engine (the robot can then identify "enough" of the engine so that it reaches the conclusion that a car has an engine). The conclusion is reached on the basis that a car is more than an engine at least visually. This leads to another relational issue; suppose, for example, that there is a garage that has a running engine inside (producing sound) and the engine is also visible (more visible than when inside a car) how can the robot distinguish between the garage and the car? To its perspective a garage is more of a car than a car with respect to the "having an engine" relationship (since the engine is assumed to be more visible). By observational count; if our robot sees and hears more cars than engines in garages by a threshold, then it reaches the conclusion that garages are different than cars. The idea is that unused definitions (mappings) are discarded after a time period.

Any observable objects input from any type of sensor may have a set of relational connectors between them (there is no distinction between data types w.r.t. mappings). The connectors can denote positional reference such as over, under, in between, etc., identity such as is, is not (for example, an observed car is visually distinct from a person and the robot can conclude that visually a car is different from a person) and to what extent an object is part of another (this topic will be detailed in the next chapters).

In essence, the algorithm expands a set of patterns depending on the number of observed objects depending on a criteria of similarity, which are used in task solving in later runs.

For visual pattern matching, an approach as described in [2] and [1]is used for individual object classification with a variation that, outputs from hidden layers in an RCNN used to classify a pattern can be used as inputs to layers of a RCNN used to classify a distinct pattern. This is necessary in order to establish correlations between patterns which are the basis for constructing definition paths. The conceptual basis of our work is that observed objects can be defined (observed) if they can be categorized into a previously learned set of templates (regardless of sensor types (visual, auditive, etc.). Similarly, tasks can be solved if they can be defined where a definition is a function over a set of sub-definitions (whose granularity expands to atomic definitions which are singular patterns).

Pattern creation can either be provided or learned through repeated observation. By observing a number of different objects with different shapes, a pattern is created from applying an averaging function over the most common extracted features.

The pattern creation algorithm must, at its highest level, follow the steps:

1. Run a feature extracting function over each of the input data, from each sensors (the number of features is a function of available computational and storage resources)

2. Combine the features s.t. shapes matching the patterns are created. If the pattern set is empty use the first set of extracted features as an initialization value

3. The pattern is "completed"/"strengthened" by each feature set extracted from the future observed objects

In the next chapters we will analize how correlations between patterns can be made and expanded across different types of input data retrieved from independent sensors. We show a means of structuring information in sets of circular lists which we've coined as knowledge spheres and how reasoning processes such as intuition, deduction, learning and defining and fulfilling goals can be formalized.

Chapter 2

Data Organization

We propose using a knowledge sphere organized in layered circular nested list sets to store patterns and matching data input from sensors. Each list will be referred to as a sphere; the knowledge sphere is represented by a number of such nested spheres.

2.1 Core Ideas

Our algorithm is based upon the core ideas that:

- any observed object can be defined based on previous observations or throughout extended observation (the latter being the case when patterns are not provided but created by repeated observations by extracting common features from observed objects); for visual input the robot can see an object as long as it can construct it as a set of shapes (features)
- correlations lie at the basis of deduction and induction
- correlations are used to model intuition
- each entity within the system have a weight value
- each component (node, definition, correlation which are detailed later) has a life span
- the system state is determined by a progress in time measured in clock cycles
- the end goal of the system is to solve tasks; each task being observable by at least an input sensor
- new solutions to future tasks are predicted by probabilistically combining previous task definitions; this is the basis of creativity and free will

- tasks can be prioritized; the higher the priority the more resources are allocated to it. This, in principle, is similar to the concept of needs
- all elements within the system are stored in circular, nested lists in a knowledge sphere

Deduction is based on the fact that observation does not be extensive (include all observed features/attributes) in order for a definition or correlation to be created. For visual data, an attribute can represent a set of shapes of which the observed object is comprised of while for auditive sensors, a feature is a signal in a frequency range. A weight value is used to represent the strength given by similarities between the last extracted features from an input and the features extracted up to that point.

The weight is a function of four variables:

- The number of observed inputs containing similar features
- The degree of similarity between features
- The duration in time objects with the same similarities in features have been observed
- The relevance of the observed input in task solving

If the observed feature is significantly different from features observed so far, it is assigned a default, low weight value, and the previous feature weights are decreased with a value depending on the new feature's relevance within the set of tasks (old and recent). If more objects are observed that have significantly different intensities from the previous, after a period of time, become a majority their weights are increased while the previous feature weights are reduced. Features are dependent on sensor type, previous correlations and task relevance each of which will be discussed <<later>>

Up to this point, we have a set of observed objects and a set of attributes and their respective weights for each. Weights can also be looked at as a belief placed by our algorithm into the observed data.

Understanding comes from the ability to give meaning to data relevant to a set of contexts. A definition groups concrete observed objects based on a common criteria matching a number of patterns.

Structurally, definitions are lists of patterns matching features in an increasing amount of depth; each pattern offering a decreased level of abstraction than the previous.

2.2 The Knowledge Sphere

In this section we propose a means of structuring definitions into sets that allow data exchange through a querying mechanism. To this end, we define a database which we will be referring to as a knowledge sphere that allows storing, adding, removing, balancing (optimizing) and organizing definition paths as shown in

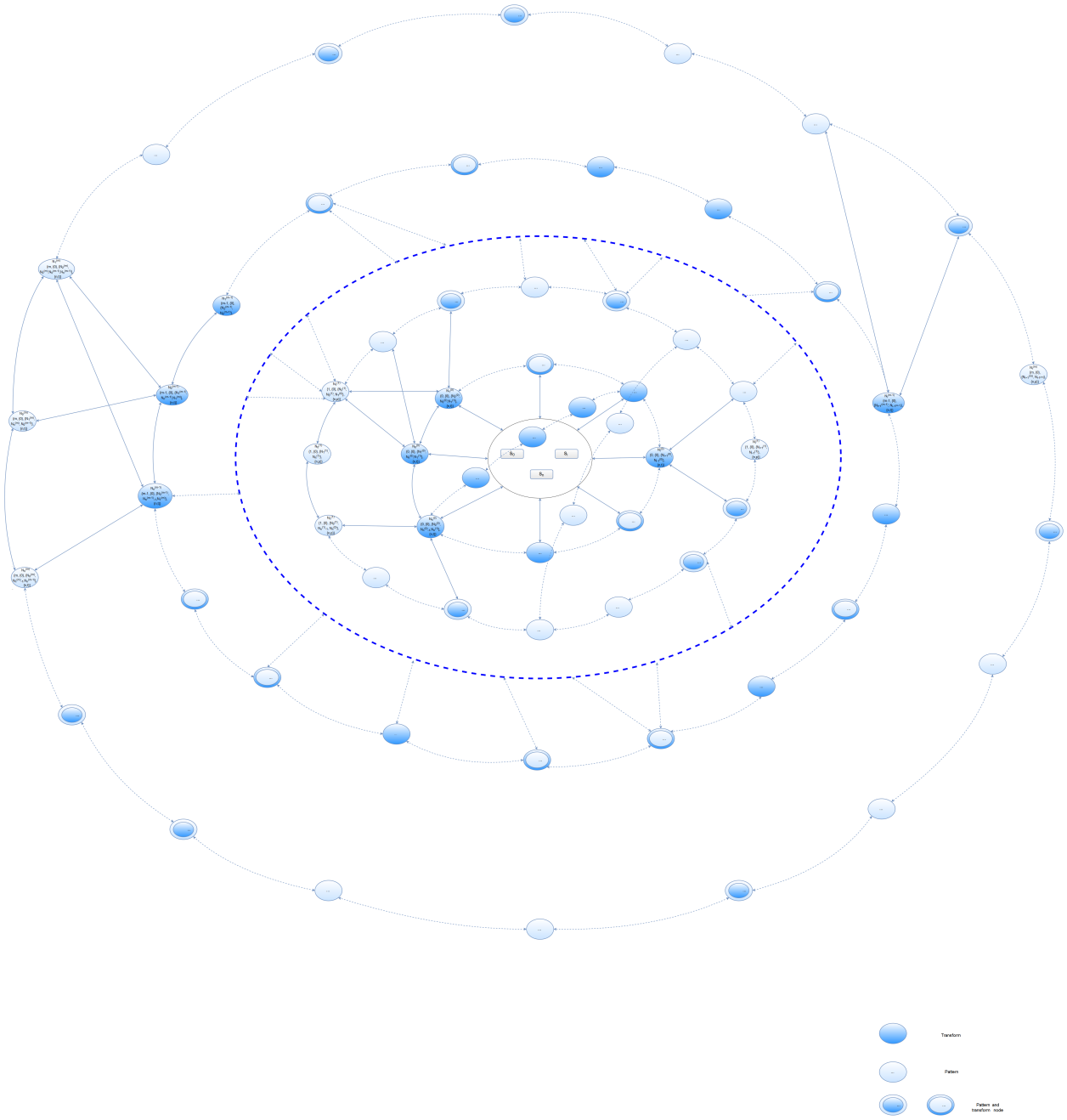


Figure 2.1: Single sensor input type knowledge sphere. Dark blue nodes are transform nodes that perform feature extraction, apply a transform function over the extracted features and compute a cost function against the patterns stored in the nodes of the next layer marked as light blue

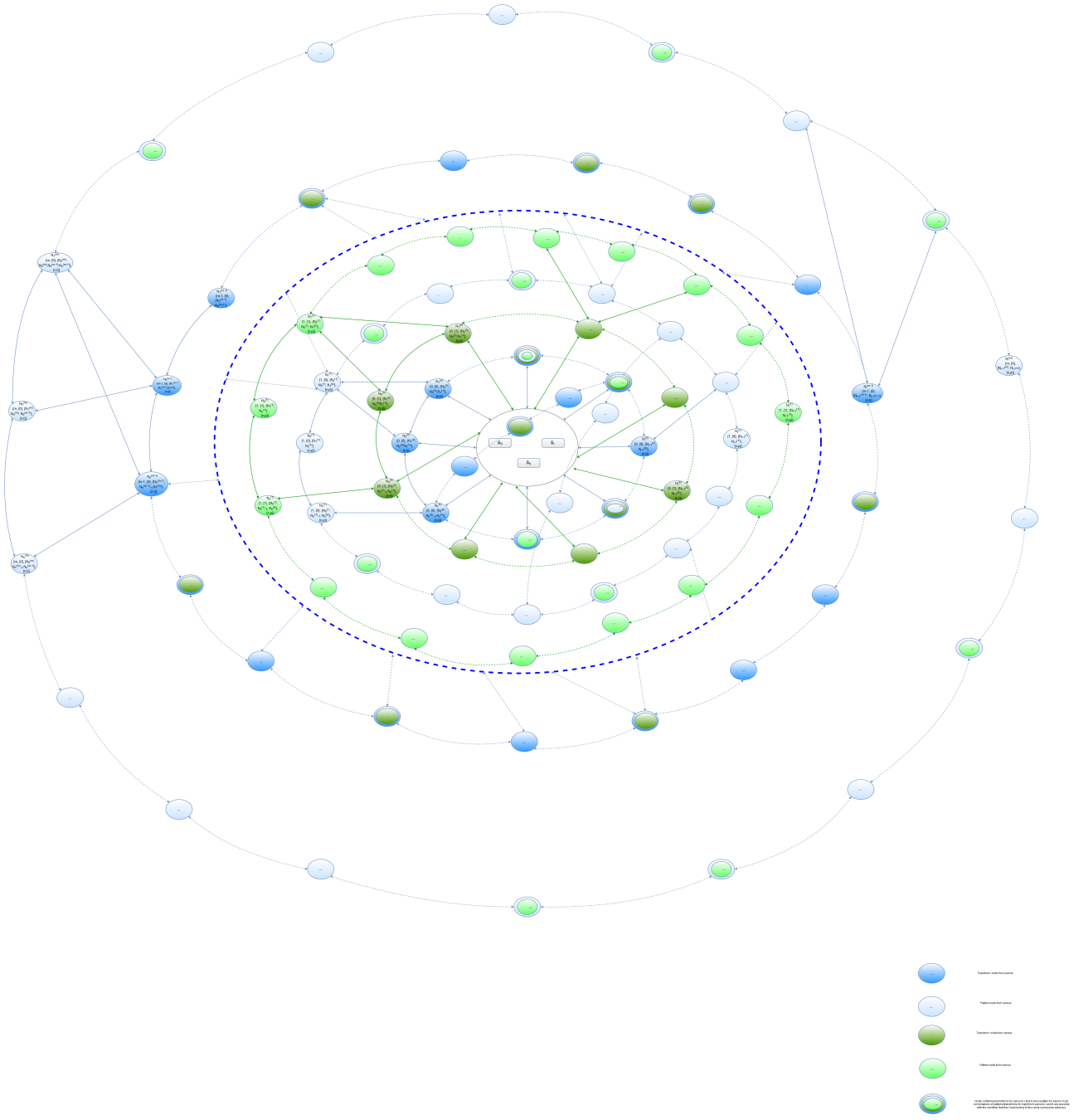


Figure 2.2: Double input type knowledge sphere. This sphere has two distinct types of sensors each having its own layers of transform and pattern nodes. The two types are marked in the figure by dark blue and light blue and dark green and light green nodes respectively. The dark nodes are transform while the light ones are pattern nodes. For the green nodes, the external layers have been omitted due to clarity considerations. An important remark is that each node can contain patterns matching data received from both sensor types; this consideration is made to make the network flexible in terms of node functionality w.r.t. computation and storage

Figure 2.1. The sphere is modelled only for one type of input sensor (e.g. visual); a sphere that is able to process data from two distinct types of sensors is shown in Figure 2.2. A fully generalized model with an arbitrary number of types of input sensors has been omitted for brevity. We will not be focusing on the actual data (patterns, transform and cost functions) stored by each node in this section..

Structurally, the sphere is comprised of nodes stored in nested circular lists organized in layers called spheres at the center of which there are input sensors (denoted by $S_0, ..., S_n$ in the figure) perceiving data from an external environment. The layer depth determines pattern level of concreteness; nodes on the peripheral layers have the highest level of concreteness, the level decreases towards the center of the sphere where the input sensors are located. The reasoning is that patterns close to the input sensors should match every possible observable object regardless of its features. Upon observing an object, features are extracted from its raw data as provided by sensors and a transform operation (e.g. geometric transforms for images and frequency transforms for auditive input) set applied over them. Next, a set of patterns that best matches the output of the transform functions (up to a threshold based on the cost between the features and the pattern) is chosen as the first node in the definition path (the most abstract pattern that fits the extracted features). Each layer (sphere) is interleaved with nodes that apply transform functions over features matching patterns in the previous layer and in turn match their outputs with patterns in the next layer.

Subsequently, pattern matching algorithm's steps are:

1. Similarly, next layers perform more granular transform operations over the extracted features and choose a number of patterns that match the output up to a threshold value from the next, more concrete, layer. Thus, a concrete pattern definition path is created for the observed object. Note that only structural definitions follow this step as behavioral, extra and meta definitions build over structural definitions as a set of correlations between them (correlations are discussed in Section 2.3.3)

2. Features match a pattern depending on a weight value. This weight is a function of the output of a cost function applied over the features and the pattern ("measuring how much features have in common with the pattern"). The highest weights indicate the patterns to be probabilistically chosen in the pattern definition path. We don't choose a single maximal value of the best matching pattern but allow some margin of error by selecting the best matching patterns up to a threshold value. This allows for creating multiple definition paths for a single object and discarding paths that are invalid in task solving. There are multiple sets of definition paths "branching" out at each layer of abstraction (each layer within the sphere).

3. Definition pattern paths have a lifespan dependent on the utility of the definition towards a task and its frequency of use. A path becomes more

relevant ("heavier") the more it is being looked up and positively contributes to solving a task. Paths that are redundant are pruned (the algorithm uses an internal clock that checks the state of the knowledge sphere and updates an internal time related value. The more time a definition path has been inactive the more likely it is going to be pruned)

The input looses degrees of abstraction (becomes more concrete) as definition paths are created from the center of the sphere towards its "surface". A definition can either be:

- created when the observed object does not "exactly" match any of the existing definition. By exact match we refer that the cost between the transformed features and existing definition paths are smaller than a constant (the constant has to be small w.r.t. to previous existing definitions). A new node is created, comprised of transformed features that matched a pattern from the previous pattern layer (sphere). It is also a pattern node in that if a new object is observed and has a definition path that falls on the same layer as it, and has a cost smaller than a threshold, then a pattern will be created from the two concrete nodes and the nodes will be pushed on the next layer, "outwards". The pattern is a result of an averaging function applied over the feature set. We will elaborate on how the topological distance between two nodes located on the same layer is relevant for abstracting their definitions into a pattern. This distance is also dependent on the cost between the patterns of the respective nodes (nodes with smaller costs will be located topologically closer)

- matching, when there exists a set of pattern definitions for previously observed objects that match the definition of the current input up to a given cost

A pattern definition path for an observed object is created (or matched to an existing definition) by adding pattern matching nodes; classifying features of an input object while progressing through the layers of the knowledge sphere. Correlations (detailed in Section 2.3.3) are created by applying a cost function over pattern matching values located in circular lists on the same layer (sphere) of the knowledge sphere. This approach allows for correlating data received from different types of sensors (in our model we'll use an identifier to specify the type and sensor from where the features were input). This is due to the fact that pattern matching applies only to data received from a single type of sensor (we cannot apply a visual pattern to an auditive input in that our robot "cannot see sound") and are independent across sensor types.

Each node has an identifier within the sphere it belongs to. The respective sphere has an identifier that is unique for the entire knowledge sphere scope. Thus, each node (and every component) of the sphere is uniquely identifiable and reachable starting from layer 0 (with the highest degree of abstraction). As an optimization mechanism, we propose for the definition look-up to use further layers as starting points and attempt to find a more concrete definition earlier.

The definition path would then be completed with sets of pattern nodes starting from the "starting nodes" to nodes in layer 0. This is the premise of formalizing intuition on which we will elaborate on in later sections <<add section>>.

Links are created between nodes belonging to the same sphere and/or distinct spheres based on their use in definitions. Structurally, nodes can have only links to nodes in the current sphere and links to nodes in previous and next spheres. We refer to nodes having links to nodes in other spheres as nexus nodes; the nexus linkage is represented by a flag n in the figure (nodes that are not nexus nodes do not have this flag set).

Definitions are lists of nodes comprising patterns and transform functions that may address one or more input data types by their input sensor. Based on their role, nodes are:

1. transform nodes, that compute a cost function over features received from nodes belonging to previous (more abstract ("inner")) spheres
2. pattern nodes storing patterns against which transform nodes compare the cost function's output. Topologically, they are always located in a sphere after the sphere containing the transform nodes
3. nodes containing both patterns and transform functions

In Figure 2.2 node coordinates and types are indicated through the following structure:

$$\{N_i^{(j)}, \{j, \{t_{i'}\}, \{N_k^{(j)}\}, \{n, p, t\}\}\}$$

where:

- $(j), 0 \leq j \leq n_s$ where n_s is the number of spheres in the knowledge sphere. j is the identifier of the sphere the node belongs to
- $i, 0 \leq i \leq n_n$ where n_n is the number of nodes in the current sphere (referenced by j). i is a node identifier with current sphere scope
- $t_{i'}, 0 \leq i' \leq n_t$ where n_t is the number of types of input sensors. i' is an identifier for the type of input data (e.g. visual, auditive) received from a sensor
- $N_i^{(j)}$ is a node within the current sphere
- $N_k^{(j)}$ represents the set of nodes, node $N_i^{(j)}$ is currently connected to (this set of nodes can change as nodes and links to them are added or pruned from the knowledge sphere)
- $\{n, p, t\}$ represent flags indicative of the type of the node, n is the nexus flag, p is the pattern flag (suggesting whether this node contains patterns) and t the transform flag (set if the node performs feature transformation and pattern matching)

Considerations

The following hold for every knowledge sphere:

- each node can fulfill multiple roles (pattern/transform) and have distinct types of data (e.g. visual/auditive) pattern and transform functions simultaneously as shown in Figure 2.2. This is specified by setting the type flags n, p, t accordingly to the nodes role
- each node can have patterns and apply, transform and cost functions over features belonging to different types (e.g. visual and auditive) of data
- the number of spheres (layers) for one type of data can differ from that of another
- each node can itself be an entry point (fulfill the same role as the sensors in the center of the knowledge sphere) for an arbitrary number of knowledge spheres. This is done so that the structure has a maximum degree of freedom in terms of layout configuration and altering

The next section describes how definitions are created from input as well as correlations between definitions.

2.3 Definitions and Correlations

2.3.1 Definitions

Definitions constitute a mapping between the features of observed objects and data structures within the sphere which are represented by sets of lists (which we are going to refer to as paths) with starting nodes, on the first, center, most abstract, pattern layers of the knowledge sphere branching outwards, in increasing level of concreteness as shown in Figure 2.3. A set of lists is used over a single list due to the fact that multiple objects can match the same definition but have different features making their definitions distinct on the concrete layers.

The process of definition path creation follows the steps:

- an object is observed by a set of sensors
- the sensors forward the raw data to the first sphere of transform nodes which perform a feature extraction operation and a cost function over the results and the node patterns in the next, consecutive, sphere. A set of nodes from the next layer is then chosen probabilistically depending on the cost function's result. Links are created from the transform nodes to the respective pattern nodes forming a path outwards through the knowledge sphere
- the process is repeated in the next (more concrete) layers until a set of sufficiently concrete definitions is found in order to solve the current or

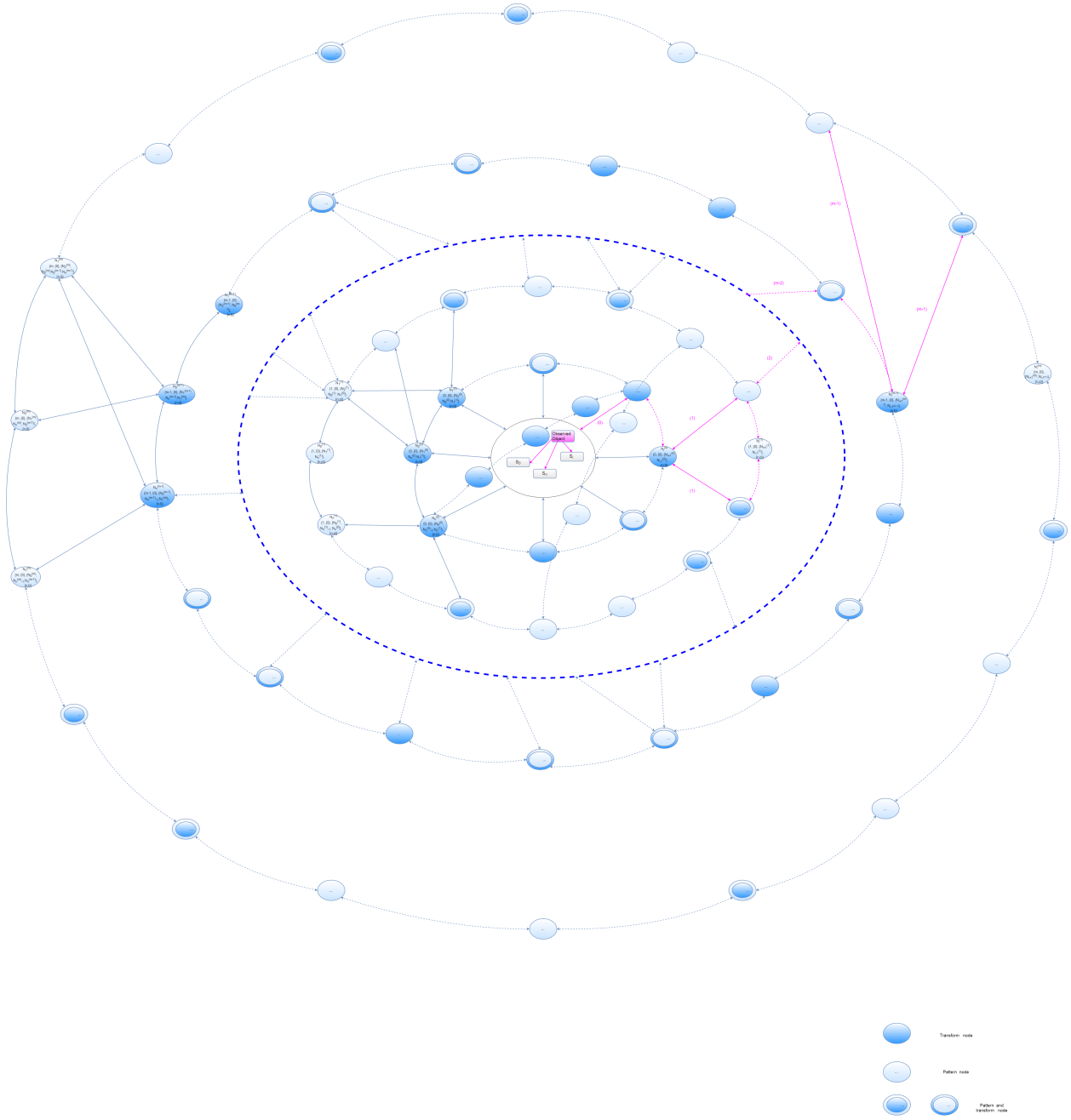


Figure 2.3: shows a definition set containing two definition paths (highlighted in purple) for an observed object. This sphere uses only one input sensor type (marked by dark and light blue nodes) for simplicity. In an elaborate model an observed object's definition would be multidimensional having a set of definition paths for each type of input sensors

previously input tasks (we will detail the topic of tasks in chapter <<add chapter>>)

- these definition paths are stored for a number of clock cycles depending on their relevance in task solving

In Figure 2.3 the link numbers between the definition paths' nodes represent the order in which patterns are compared and matched against features extract from the raw input data provided by sensors; definitions are created/looked up starting from nodes on the most abstract layers and concluding with nodes on layers of a lower level of abstraction.

Depending on the task, full definition paths may not be necessary for obtaining (sufficiently) valid results.

For example, if a task is to determine whether a car is observable and moving, it does not need a full definition path of the exact observable details (size, color, auditive definition) only a subset of the path sufficient to capture the observed object as being a car and that behaviorally it's changing position during observation (the behavioral and observational (structural) components are discussed later in this section).

Initially, the whole path is used as there is no task definition (the task is solved for the first time). On subsequent iterations, the path is shortened by a number of nodes from the outer layers (the definition path thus becoming more abstract). The amount of nodes that are discarded is dependent on the definition's path relevance w.r.t. the task; we propose a mechanism that decreases or increases the number of nodes exponentially (as a power of 2, <<similar to TCP window scaling>>).

A definition must contain the following information w.r.t. the object it defines:

- the (pattern and transform) nodes in the definition path
- an identifier set for each definition of subobjects comprising the currently observed object; the set if empty if the definition is atomic or there are no previously observed objects that can be observed in the current object
- time of object observation
- duration of observation
- a reference to the tasks it is/can be used in
- relevance value for each task

Figure 2.3 shows two definition paths created for an observed object. During task solving, definition paths are chosen probabilistically based on the cumulative costs of their pattern nodes; an overall weight is given to the definition as a function of these scores and relevance in task solving.

As an example, consider that the observed object is a car and the two definition paths are two concrete cars that have been previously been observed and have the best cost function result matching the currently observed car.

A definition is thus a path through a set of patterns that each match the observed object with increasing levels of precision (detail).

W.r.t. pattern nodes, matching can be:

1. Of the same origin, where definition paths from possibly distinct observed objects share definition paths each beginning with the same most abstract pattern (in the center of the knowledge sphere). For example, two cars observed from the same angle.

2. Of different origin, where definition paths do not match the same first, most abstract pattern (matching is done on the subsequent, more concrete layers). For example, two cars observed from different angles

Figure 2.4 shows an example illustrative of both cases. Objects I and II (marked with purple and green) share an abstract pattern node in the first layer while objects III and IV (marked with orange and light blue) have common nodes in subsequent layers.

By matching, we mean that there is a set of transformations that are applied over the extracted features from the observed object so that their output fits the pattern (the difference between the result and the pattern is below a threshold). The transformation can be either in the geometric or frequency domain depending on the sensor/type of features needed from a sensor.

The threshold value starts with a small subunitary value and is gradually increased as more objects match the definition (meaning that the definition becomes more "specific", similar to the bias in neural networks) and are valid for a set of tasks. Definition creation can be automated by employing a classifier that performs feature extraction and creates a pattern as the output of an averaging function performing cost minimization across all observed objects (in an early implementation patterns will be provided, not created through observation).

For example, if the robot observes three images: one of a coupe, one of a family and one of a van it extracts features from all images and compares them against its existing set of patterns. The pattern with the highest matching value (cost function between extracted features and the matching pattern is below a threshold) is chosen as the starting pattern of the definition path. Suppose that the coupe was the first car to be added to the definition path and similarly the family car and the van. These objects are very likely to share multiple nodes of the definition path (they are merged into a single path up until features become distinct enough to create separate paths containing more concrete patterns matching each car); when another, distinct car, is observed by our robot the process of feature extraction, transform functions applied over the previous result and matching is repeated and it is likely that the car is added to the definition path (up until some point) of the previously created definition paths of the cars observed until now. All observed cars will thus share common nodes of patterns within their definition paths.

Considerations:

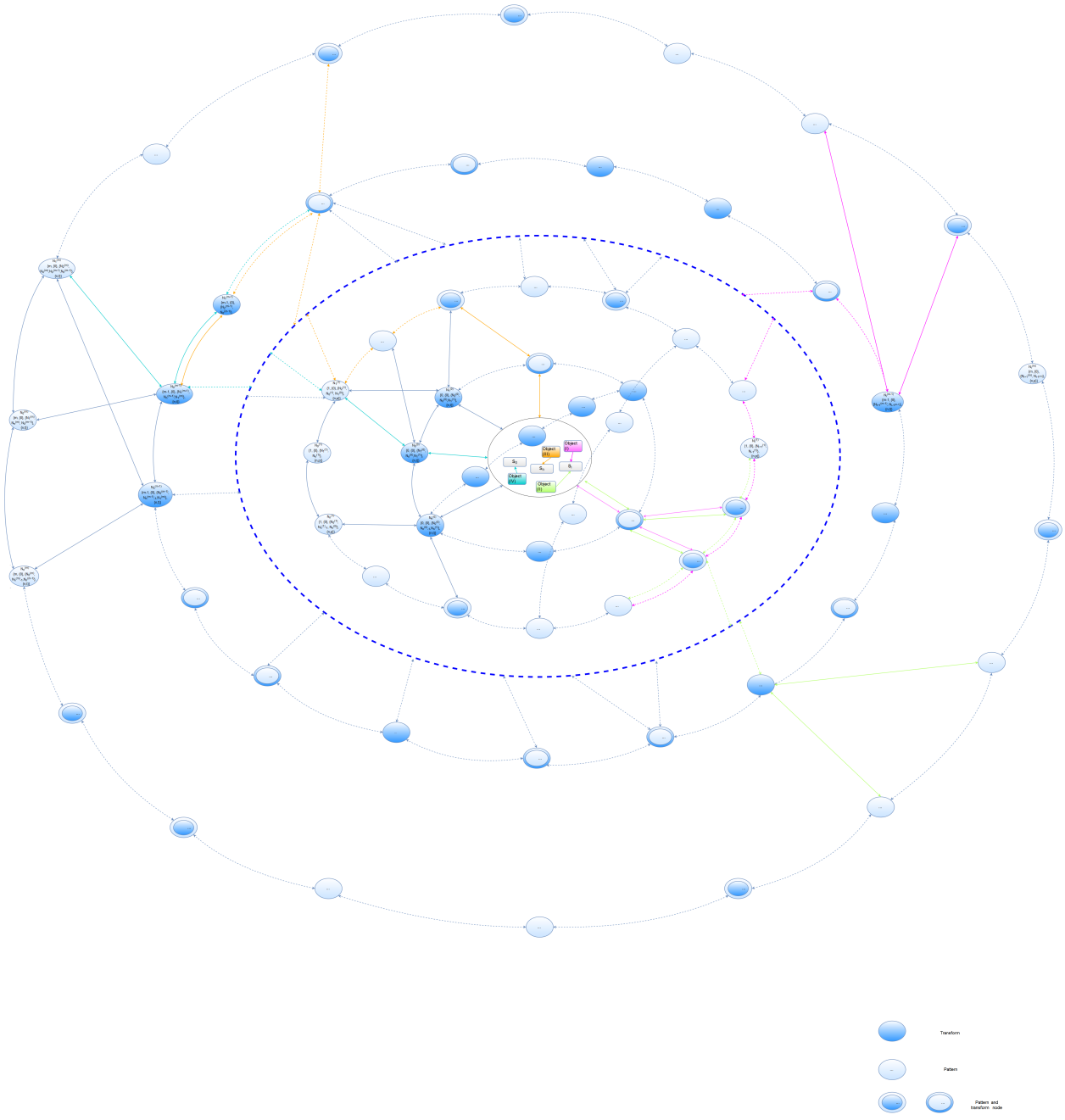


Figure 2.4: shows definition matchings between two pairs of objects; one starting from the same, first, abstract node and the second from distinct nodes

- definition matching (look-up) lies at the basis of correlations which will be described in Section 2.3.3
- intuitively, the more nodes two definitions of observed objects share, the more similar the objects they correspond to are
- definitions are layered: at the first level (node containing a pattern), the most abstract pattern, is comprised of a very permisive pattern matching a broad feature set. The subsequent levels (nodes containing patterns within definition paths) are narrowing down the feature set in order to define the observed object concretely (uniquely). As an example, the definition of the term vehicle encompasses a wide range of objects: trains, family cars, sedans, airplanes, etc. with the main correlation between them being that they are all able to move from one point to another. The definition of a car matches objects such as family car, sedan, sports car, etc.. The definition of fast car may match only sports cars. Thus all these objects have the same underlying definition path at the most abstract level (they are all vehicles that can move from one point to another)
- definitions are grouped (correlated) regardless of the input sensor they originate from
- the length of the definition represents the level of detail the observed object is currently known and varies depending on observation
- definition path length is object and sensor type independent (two objects do not have to share the same definition length regardless of definition and data type; similarly, the definition path length of an object can differ across data (e.g. visual, auditive) and definition (structural, behavioral, extra and meta) types
- there is always a best matching pattern that fits any observed objects (all objects are split into a predefined existing set of shapes that can make up any object) meaning that definition creation is always possible as long as the a set of abstract patterns that fit it to some degree exist. A new definition (that has no other exact matching pattern paths) is added to its own definition path

Depending on their pattern nodes, definition paths can be:

1. **Atomic** given that each node of the respective path does not have a distinct definition path in the knowledge sphere (all nodes are self contained in that they do not have a definition path of their own)
2. Or **Complex** where pattern nodes are not required to be atomic; they can have a definition path

A definition is comprised of a set of four multidimensional components that essentially state what the observed object “is”, “can do”, “done”, “does”, “will (is likely) to do” and “how”:

1. **Structural component** that defines static characteristics of the object's features. Visually, such features can be the size, scale, etc. or auditively such as the pitch of the sound an object produces

2. **Behavioral component** that defines a set of transitions an observed object goes through as well as predict future states. For example, if a car can move from point A to point B, the robot infers that it can also move to a distinct point from A and B, namely C

3. **Extra component** is a subset of the behavioral component that defines corner case situations, that are far less likely to happen than the expected behavior. For example, a car, while moving from one location to another, may collide with another car or break down. These events, however, are less likely to occur than the expected behavior of the car reaching a destination

4. **Meta component** which does not use the extracted features directly but rather a set of properties over them

Next, we explain each component in more detail.

Structural Component

All definitions are organized into hierarchical sets of patterns and are characterized by inclusion operations in that, a definition path may be a reunion of definition paths. Each node in the path has a definition path of its own unless the definition is atomic. An atomic definition's path is comprised of nodes that do not have a definition path and are part only of the respective atomic definition's path.

For example, generally, all cars have wheels, all wheels have tires, all cars have doors, an engine and a frame. The definition of a car is made up of its components' definitions. Thus a correlation exists between the components' definitions and the car definition. Correlations pertaining to the patterns of definitions are referred to as structural correlations and are covered in Section 2.3.3. This allows our robot to deduce definitions of observed objects even if only a subset of their constituting parts are observable. Deduction works by observing the object and checking if there exists a pattern matching at least one of its component definitions to some degree. If such a pattern exists, a set of definition paths is probabilistically looked up for objects containing the respective component's definition. If not, the observed features are compared against constituting definitions of the best matching patterns and one is picked probabilistically based on its cost.

For example, if the observed object is an image of a car behind a wall so that only part of the car is visible, definitions for visible parts (e.g. wheels, car door, frame, etc.) are looked up (assuming that a whole image of a car has been observed previously). If a number of features are identified to fit pattern definitions for objects (components) comprising the observed object then the

robot infers that the observed objects fits the respective definition (the robot can infer that the observed object is a car even if it only sees a portion of it).

Figure 2.5 shows the deduction process applied to an observed object (marked with orange) which yields two definition paths (marked with purple and green) of previously observed objects that have features of the observed object in their definition paths.

Behavioral Component

Behavior is characterized by a number of transitional states the observed object goes through in a time interval. In these states:

- new features may be extracted that have not been observed earlier and added to definition paths of previous objects
- less or no new features are observed

Behavior implies structural (at least observational) change and is defined as a variation function over structural definition pattern nodes. Behavioral definition paths are relational and capture state differences across observed objects. These differences are defined as outputs of a set of relationship functions over patterns from transitional definition paths each corresponding to a state of the observed object.

State change is observable only w.r.t. a reference point which can be:

- an object observed simultaneously with the object whose behavior changes from one state to another
- the object itself, compared with its versions from future states
- a combination of the above

For example, structurally, both a family car and a van are cars, road vehicles and vehicles, the hierarchical relationship being that cars are road vehicles which are vehicles. A train and a cable car are rail vehicles thus both the car and the train fit the pattern definition for a vehicle at a higher level in their definition paths. The underlying definition of a vehicle is that of an object that can move from one point to another, the transition of moving constituting the behavioral component. But this definition also matches a person moving from one point to another which would classify the person as a vehicle implying that, at least behaviorally, both a person and a vehicle match an abstract pattern. This matching is valid as the differences (both structural and behavioral) between a person and a vehicle become distinguishable in further, more concrete patterns nodes, in their definition paths.

Extra Component

This component encompasses corner case behavioral components, scenarios that are observed infrequently w.r.t. a threshold value. Concretely, if there is a dominant set of structural and behavioral definitions matching a set of similar objects

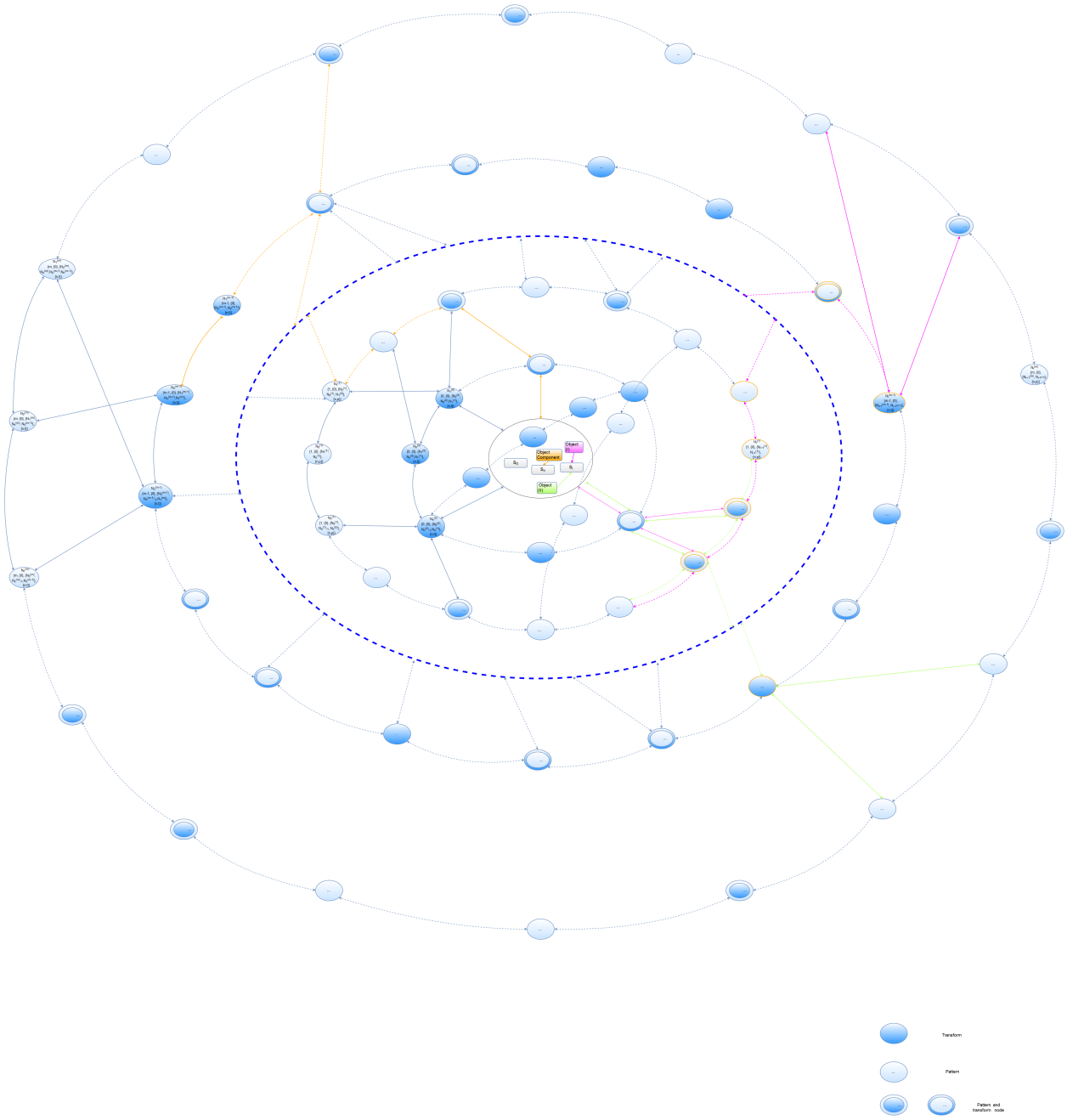


Figure 2.5: The currently observed object is marked with orange and is a structural component of previously observed objects marked with purple and green. During inference one of the latter objects is picked probabilistically depending on a task relevance value dependent on the task the object is used in.

and a new object is observed that in some state no longer matches the dominant definition set (it is sufficient that a single mismatch occurs regardless of definition (structural, behavioral, extra and meta) or data (e.g. visual, auditive) type) an extra definition is created for that object.

Figure 2.6 shows two objects (marked with purple and green) that are part of a dominant definition set (assuming more similar objects have been observed earlier) and a third object (marked with orange) that differs from the definition set and triggers the creation of an extra definition.

As an example, consider cars that are observed regularly moving from one point to another. In the knowledge sphere a dominant definition path matching cars is created. When a car crashes its structural and behavioral definitions change and an extra definition path is created for it.

It should be noted that extra definitions depend on observation count; once extra definitions become dominant (occur more frequently that the current definition paths for a set of objects), the previouly dominant definitions become extra definitions.

Meta Component

Meta components lie at the core of the algorithm, working with definition path attributes rather than path nodes. These attributes are:

- time and duration of observation that groups a definition with a set of definitions that vary in temporal observation up to a threshold
- number of definition look-ups that groups definitions based on their frequency of use in task solving
- task relevance grouping definitions as a function of the validity of the output they produce
- (behavioral and structural) correlation creation and pruning that associates definitions based on the comprising structural and behavioral sub-definitions
- any set of category identifiers that are not structural nor behavioral

Definition categories are created from the above attributes and used to query the knowledge sphere on input tasks.

Remarks

In conclusion, an observed object's definition is a set of definitions (structural, behavioral, extra and meta) obtained from data received at each sensor of distinct type (e.g. visual, auditive) mapped together based on a set of variables:

- time of observation: if the robot sees and hears a moving at a given time difference apart car then it creates two types of definition sets for both visual and auditive sensors

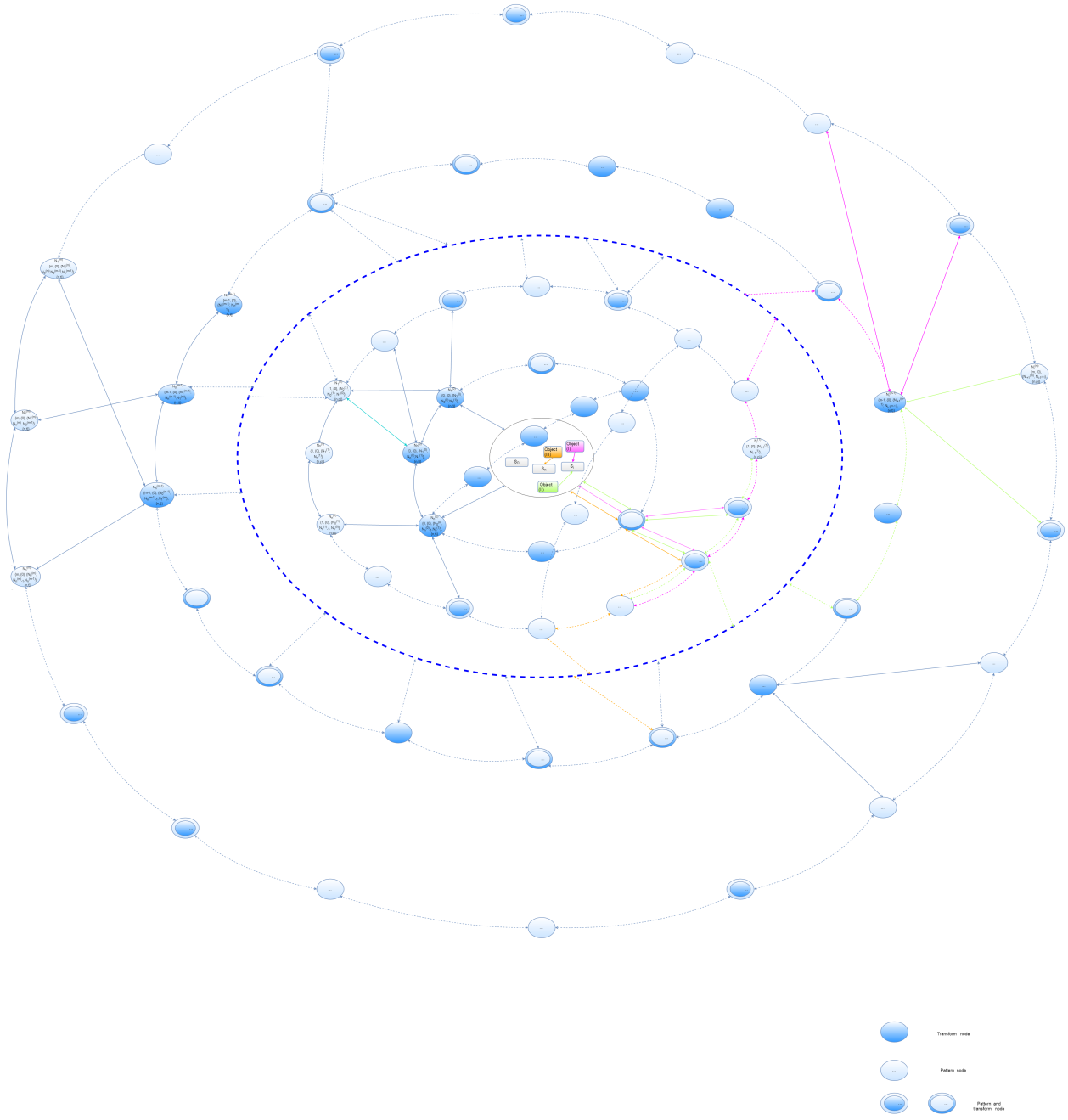


Figure 2.6: Objects I and II match a dominant definition while III doesn't. An extra definition paths is created for the mismatching object.

- the definition's utility in that how much is a definition needed for further processing. The utility has an associated weight which might increase or decrease depending on the number of times the definition is used and the results obtained by using it. Definitions for a single sensor are cross referenced across their types (structural, behavioral, extra and meta) in order to find a better matching chance for an object. By cross referencing we mean that it is possible to infer a definition from another. For example, if the robot sees a stationary car it can infer that the car can move (given it has seen moving cars before that match the same abstract pattern that the observed car matches). Cross referencing is not limited to a single sensor type being possible across different definition types across different types of sensors. For example, if the robot hears a car it also creates a visual pattern definition of the car (expecting to see it).

2.3.2 Using Definitions

In this section we show how definitions can be used to solve a basic task; the task does not depend on previous tasks (the setting assumes that it is the only task being solved) and all terms already have previously created valid definition paths (valid w.r.t. the output of the task). Suppose the task input is:

- What is your name?

The question can be approached by considering that, since the input is a string (perceived by a visual sensor) is split into atomic, distinguishable patterns and can thus be "read" by our robot. In order for the robot to be able to understand the task it must previously have knowledge of the English alphabet and language (at least to the extent of understanding each word). Words are comprised of letters where each letter is comprised of a set of shapes (edges) that match patterns, each forming a definition path. A word is thus a set of matching patterns for each of its constituent letters. On its own, the definition of a word is meaningless unless attached to pattern definitions of previously observed objects; a word is an identifier (label) for a definition (intuitively, two words in distinct languages can point to the same definition). For example, a human reader can stumble across a word not read before. Its meaning can be looked up in a dictionary and attached to a set of words that the reader is familiar with.

Knowledge of the English alphabet and a set of words (the robot must understand each word of the assignment and, optionally, in order to have meaningful behavior, be able to solve the task satisfactorily) is obtained training a classifier that matches an input image (of the letter) with a pattern with a given probability. The pattern itself must be built into the algorithm initially. This process is similar to that of learning the alphabet in school; the patterns (letters) are shown to pupils after which they are able to recognize them in texts even though they are distinct (yet similar) to the ones they've been shown.

Perceived (written, heard) words are identifiers of definition paths within the knowledge sphere. Words are multidimensional in a sense that they can refer two multiple observed objects and have different meanings in different contexts.

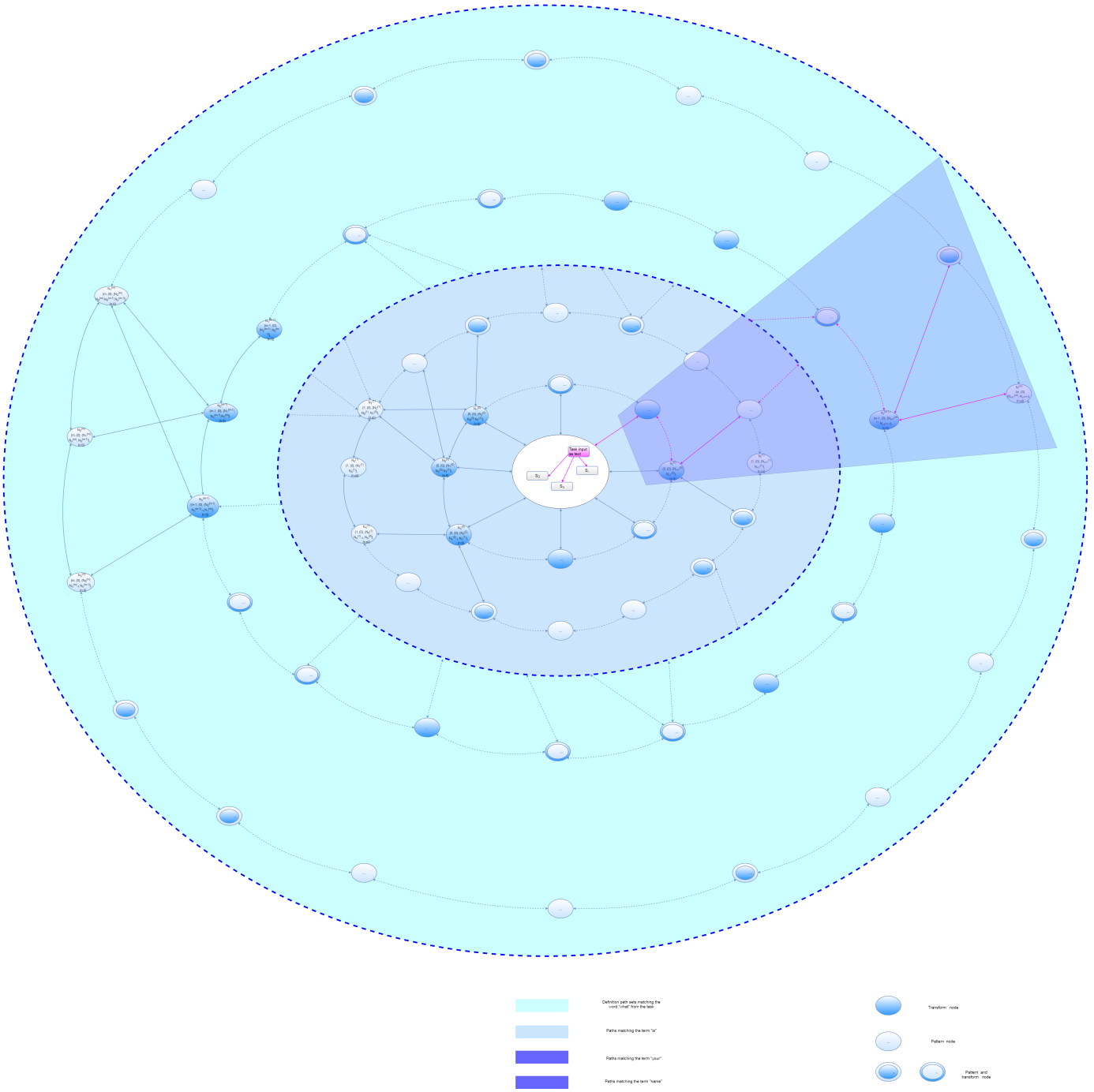


Figure 2.7: The task is split into its terms (the image does not consider term as well as atomic character definition look-ups) where each term matches a definition set (marked in the figure as a region with a background color) in the knowledge sphere. The term "what" denoted by a light blue background has a general scope encompasing each definition path. The term "is" narrows down the task's definition scope w.r.t. a temporal reference and is marked with a circular dark blue background. The term "your" further reduces the scope and is marked with a transparent dark blue background. Finally, the term "name" concludes the task which now has a matching definition in the sphere. Note that the question mark "?" is ommitted from the look-up as it is assumed that all tasks are questions which the robot has to provide a best matching definition (we do not consider the option not to answer).

The word "what" in the first questions refers to a category of definitions that, from a human perspective, can refer to physical objects, a number of phenomena, etc. This definition is not useful to a great extent as it groups a large number of definitions that are unrelated to each other and more importantly to the task.

As the second term, "is", is read the robot can infer a temporal (present) and identity reference of the task. Similarly to the term "what", "is" is a pointer to a set of definition paths that share features w.r.t. these references to an extent (defined by a threshold). When "is" is used as a temporal reference, definitions with a more recent creation date will have higher weights than older ones making them more likely to be probabilistically picked in task solving.

It can be seen that the task's definition becomes clearer as more words are read; the underlying principle of the algorithm is that "everything can be defined", any observed object has an atomic definition or one created from combining existing definitions from previously observed objects that at their core are sets of atomic definitions.

When the terms "your" and "name" are input, the number of definition paths is further narrowed down. The term "your" indicates possession and points to a region in the sphere containing a meta attribute characterizing object ownership while "name" points to definitions that hold all input names (a name is essentially an identifier within a context). Since there are no more terms, the algorithm probabilistically outputs an intersection of all definition inferred to by the terms in the assignment. Figure 2.7 shows, intuitively, how the task's definition is looked up in the knowledge sphere.

The approach is flexible in that the robot can have multiple names or none, depending on the knowledge accumulated to the point the task is input. Definitions may change depending on their degree of use and usefulness in task solving.

For example, the meaning of letter A can be assigned to letter B if there are more tasks where B's definition leads to valid results over a period of time. Result validity in this example is assumed to be externally defined; once a result is output an input shows whether it's valid or not (this input uses a side channel not linked to the sphere's sensors. If the output value is not valid another is picked probabilistically from the remaining matching definition set).

To this end, we define patterns as "facts" in that they are never changed or discarded; the algorithm cannot "forget" their meaning (the definition path nodes are not pruned) thus definition creation and look-up are always possible w.r.t. these patterns even if previous definition paths that build over them have been discarded (pruned).

It's interesting to note that our robot can learn through observation and abstraction of features implying that ("spoken" or "written") words are not required for task solving. As mentioned earlier, words are only used as pointers to other, existing definitions, within the knowledge sphere. A task is solvable if it's outcome can be defined by a set of definition paths. To solve the question "What is your name?" the robot must understand all constituent parts (letters, words and the definitions they map to, as well as the task's definitions the words point to in the knowledge sphere. Random words such as "quick", "sky", "building"

can all have definitions in the knowledge sphere but are very unlikely to define a task; the algorithm will probabilistically output an object that matches an intersection of each definition if present or the maximal intersection set between each term's definition). The word "name" points to a region in the knowledge sphere where each definition offers uniqueness to an object in a context (names are used as identifiers to distinguish notions, people, facts, etc.). Tasks **** follow the same definition creation/look-up rules as any observed objects; once created they are stored for an amount of time that is a function of their degree of use, validity as well as usefulness in subsequent task solving. The degree of use represents the number of times a definition has been looked up/created within an interval of time and validity rate, the number of times a definition was successfully used in solving a task. Definition merging applies to tasks in that, similar tasks will have a larger set of commonly shared nodes than distinct ("unrelated") tasks. For example, the task "What is your name?" is more similar to "What is his name?" than "How many tomatoes are in the basket?".

A task definition path creation/look-up follows the steps:

1. Initially, if there are no definition paths related to the current task, definition path look-up is exhaustive, starting from nodes (patterns) in the first, most abstract, circular lists and progressing towards the outer lists containing more concrete patterns

2. If the sphere contains definitions of previously solved tasks, they are checked for common nodes with the current task (a cost function is applied over the extracted and transformed features of the current task and the patterns of the previously solved tasks. A set of best matching patterns is chosen probabilistically). If suitable definitions are found, their paths are expanded with nodes containing objects required in the current task. For example, suppose we had a task "Who was Enrico Fermi?" that was solved successfully and followed by the task "Who was Michael Faraday?", the look-up process will identify common patterns ("who was") within the first task in the second and begin the look-up process from the region in the sphere where the task subset denoted by the terms "who was" points to. The weight of the common subset of patterns ("who was") is also increased since it has been looked up twice (in this context) and (we assume) provided valid outputs in both cases

Simple tasks as the ones mentioned above, can be correlated by their terms. Definition look-up is less costly in that it can use as starting nodes, nodes from layers pointed to by the common terms ("who was") and move towards the outer layers without initializing the look-up from the first, most abstract, layers.

For more elaborate tasks we propose the use of starting nodes in deeper layers of the sphere. This approach approximates intuition, it tries to make a reasonable "guess" based on the terms (words) of the task. In principle, each word points to region containing definition paths. The algorithm groups sequences of words (the key elements are features; words are relevant for the example task in this section but may be replaced with different inputs in the context of another

task; for example, the robot wants to cross a street and checks ("hears" and "sees") whether there are cars passing by) and probabilistically selects starting nodes in the regions defined by the correlations (correlations are covered in Section 2.3.3) of the definitions they map to. The number of previously solved tasks are factored in if they have been defined with terms that are found in the current task <<more on this later>>.

The starting nodes are, topologically, located between the first layer and last layers of the sphere. Once they are labeled as starting nodes, definition creation from these nodes is two-way:

- there is a definition path created from the starting nodes towards the center of the sphere, containing the more abstract layers of patterns
- and a definition path comprised of nodes beyond the starting nodes, towards the surface of the sphere containing concrete patterns of observed objects

Starting node definitions only complement normal definition look-up/creation which starts from the input sensors and progresses to the concrete, outer, layers of the sphere. The idea is that each layer has an identifier and definition paths can keep track of the layers their nodes are located in. This offers a topological reference as definition paths can "know" when they have common nodes and when they have passed each other on different layers of the sphere as shown in Figure 2.8. Tasks have identifiers bound to definition paths and their constituting nodes.

If two definition path look-ups "pass" each other during creation/look-up (in Figure 2.8 the path modelled by intuition does not have common nodes with the conventional path), a topological distance is computed in between the nodes located on the layer where the definition paths first intersected. A region can be defined in between these two nodes that can link the two distinct definitions (we assume they have no common nodes). Nodes are chosen from this region and a definition path created between these nodes and the intersecting definition paths. The conventional definition path look-up operation denoted by purple edges in Figure 2.8 is ongoing while the intuitive model is in progress; if a valid definition is reached before the intuitive model finishes the process is stopped.

If there are no definitions with common nodes more starting nodes are picked and the definition set extended; the conventional definition look-up path that started from the input sensors is preserved and continued definitions generated from the new starting nodes. The starting nodes are picked based on a topological distance function that computes the length of a path (in edges) between two nodes in the sphere and defines a region bordered by the nodes from which the future starting nodes are picked with a probability.

Similarly to the previous definitions, if the result of the task is valid (the intuition that created the definitions was correct) the paths are stored for an amount of time depending also on the number of future uses.

If the looked up/created definition path is not valid w.r.t. the task three approaches are possible:

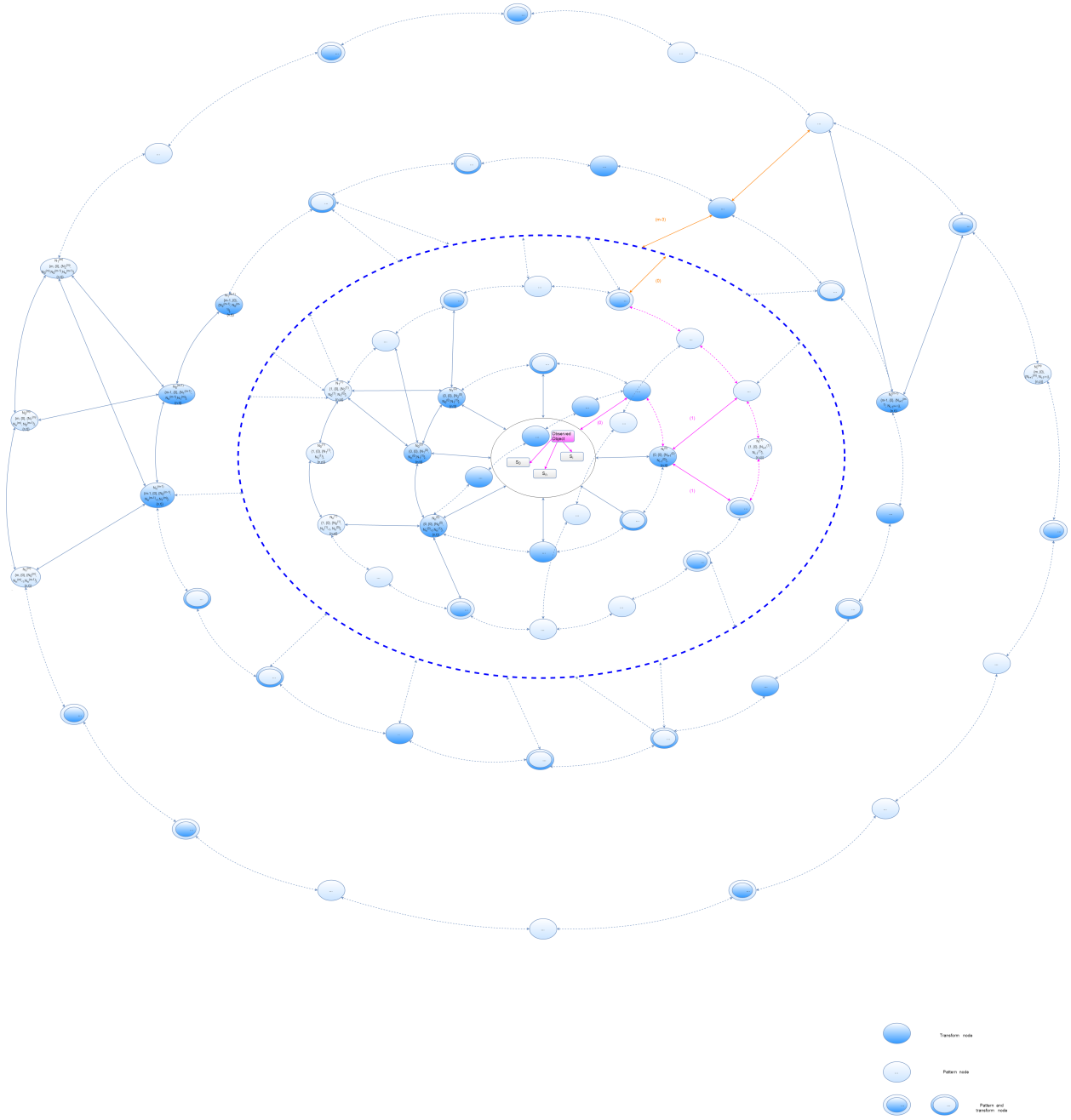


Figure 2.8: The figure shows two definition look-ups starting from different layers; the purple definition path is lookeup using the conventional approach of pattern matching starting from the inner layers while the orange definition path relies on an intuitive model; the starting node's position is chosen as a function of previous task definition paths and their validity. The number on the edges represent the order in which they are created; the intuition model as well as the conventional look-up are made simultaneously.

1. The conventional look-up is used
2. The starting node set is extended with more nodes from within the region as mentioned previously
3. If there are no valid definitions for the task in a suitable amount of time, the existing starting nodes and the nodes added in step 1 are discarded and new ones, from different regions of the sphere are chosen and the process follows a similar path as described earlier. The term suitable is defined as an average value over the set of similar tasks solved by the robot prior to the current task. The degree of similarity is represented by:

- a cost function <<define cost function later>> over the features (definition paths) of previous tasks, in this case the terms (words) used to describe the task
- the amount of computational effort allotted to the task (we use this bound to prevent the algorithm from being stuck on a single task)
- the time stamp since the task has been last used in a definition look-up (older tasks have less relevance)
- the validity of the task. Both valid and invalid tasks are considered since both are relevant to the current task but invalid tasks have lesser weight

Figure 2.9 shows how intuition is modelled in our approach. An intuition region is selected based on the input task and the previously solved tasks with which it shares a set of common nodes (e.g. contains similar terms from the task (we assume that both previous and current task are described in written words)).

This is a simplified example using mono-dimensional definition. Real world examples include definitions from all scopes (structural, behavioral, extra and meta) correlated across all input sensor types (visual, auditive).

2.3.3 Correlations

Structural Correlations

Capture similarities between object definition paths as a function of pattern node raw data. By raw data we mean data that has been perceived from an external environment and is not a relation over existing definitions. A correlation is created between two definition paths of two distinct objects if on at least one layer (sphere) the output value of a cost function applied over a subset of their pattern nodes belonging to that layer is less than a threshold value. Note that, if the definitions share nodes, then the cost between the respective nodes is 0 and a structural correlation is created.

The meta component keeps track of each atomic and complex definitions (described in Section 2.3.1) of an object . Each layer, contains a meta scheduler

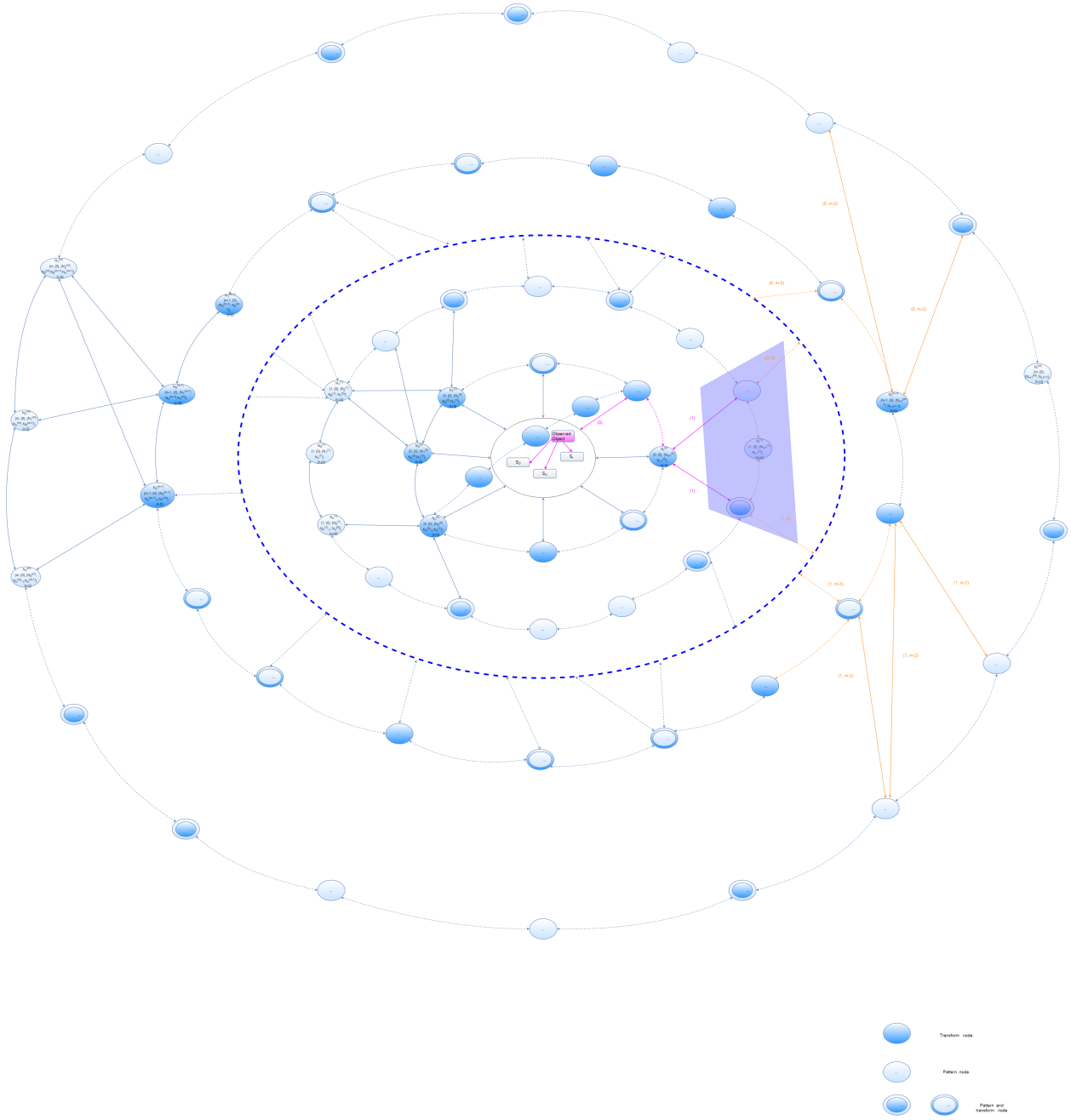


Figure 2.9: The purple path represents the definition path created by conventional look-up while the orange paths represent paths that continue the conventional path. The region marked with dark blue contains nodes that are probabilistically chosen in definition paths (they model the intuitive process). Both paths are created simultaneously and only when conventional look-up reaches nodes belonging to the denoted region is the second definition path appended to it. If the path is not valid for the task, another path starting from the region is appended (in the figure, paths starting from the region have edges marked with tuples (x, y) where x represents the order in which the path is appended to the conventional path and y the order in which edges from the region are created).

<<more emphasis on this later>> that applies transform functions over pattern nodes on the same layer and compares the result with a threshold value, a structural correlation being created based on the comparison's result.

The knowledge sphere contains a meta scheduler that has global scope (can process nodes from the entire knowledge sphere) and creates cross-layer structural correlations from correlated pattern nodes located in succesive layers based on their cost computed by the local (layer) meta scheduler.

Similar to definitions, correlations have a weight value, initially defined as a function of the cost difference value for nodes located on the same layer and consecutive layers. The weight varies on degree of use <<expand on this later>> and correlation task relevance <<more info is needed>>.

Figure 2.10 shows a correlation between the definition of an observed object (whose path (links between nodes) is shown in purple) and one of its components that already has a definition created in previous observations (orange).

Considerations:

- in Figure 2.10, the correlation between the current definition (purple) and the previous (orange) is represented by a pattern node located on the last layer of the knowledge sphere. There are, however, no conditions on the number of component node definitions, their roles (pattern or transform) or their topological position inside the knowledge sphere and definition paths
- following the statement above, the definition path of the previously observed object (marked as orange in the figure) does not necessarily have to end on the last (most outward) layer of the knowledge sphere
- structural correlations allow for an object's definition look-up only from observing a subset of the respective object's features (comprising objects) implying that the object, as a whole, has been previously observed and already has a definition path
- structural correlations are not bound to a single sensor type; nodes can hold patterns for distinct types of data (e.g. visual, auditive); correlations thus being possible regardless of input type. Consequently, a definition path look-up for an object can be made from a subset of the respective object's features regardless of their type

An example of a correlation similar to the one in the figure would be a car being observed (the purple definition) containing the definition of car door (the yellow definition).

Behavioral Correlations

Behavior is defined by a set of transitional states the object goes through during observation. State transition can only be observed relative to a reference point (state) that has a structural component as a function of time (states change

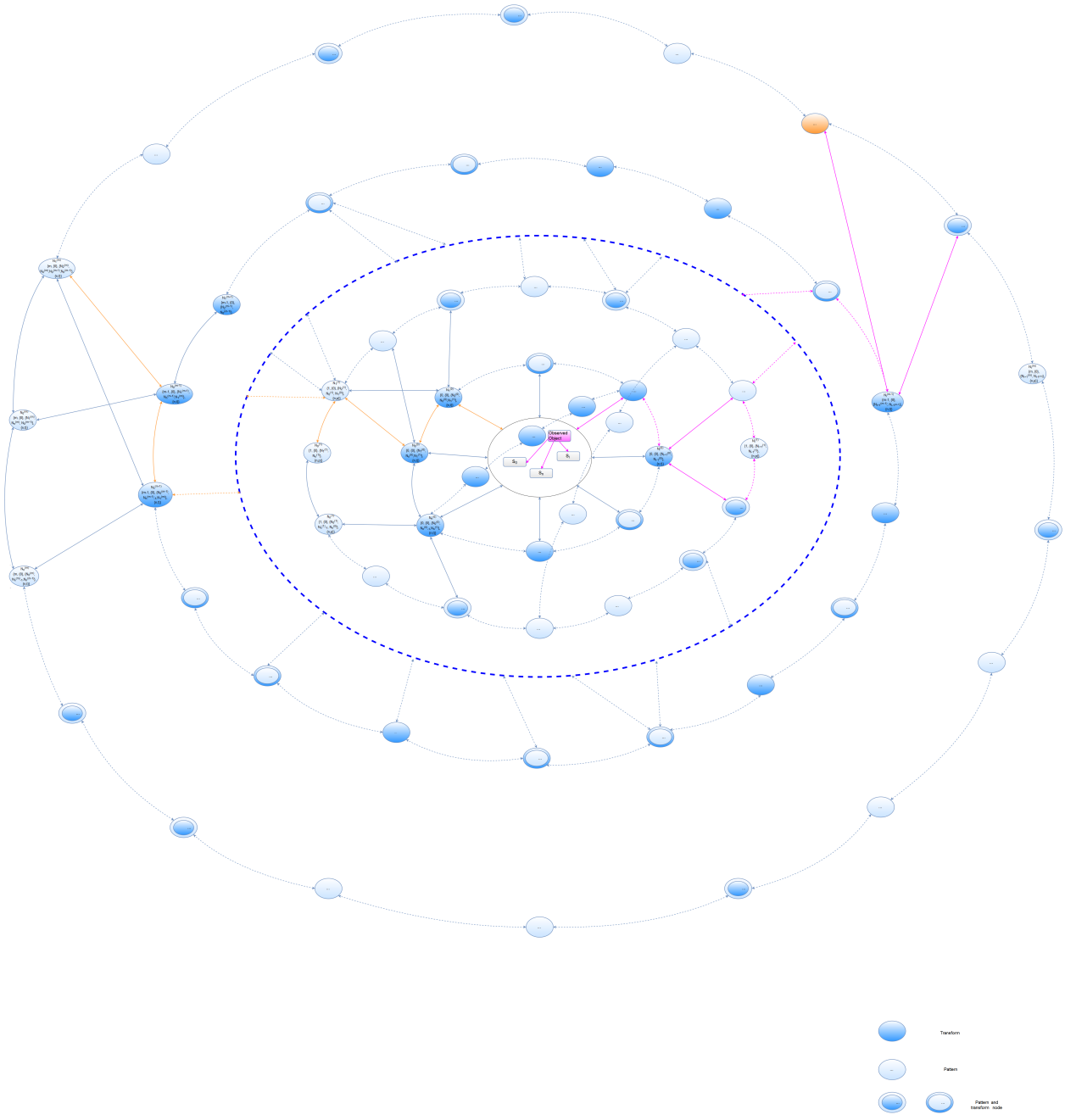


Figure 2.10: The observed object (purple) contains a pattern node (orange) whose definition has been created from previous observations

in time). Transitioning across states implies change of at least one definition component (structural, behavioral, extra and meta).

Behavior from a structural standpoint implies observing at least two objects where one is the reference point and the other, the object for which state change is captured relative to the reference object. Note that the object can be its own reference point from a previous state.

From a temporal standpoint, each state has an associated time of observation and duration. There exists a time interval, in which the object is expected to change states, that is a median function of previous time intervals between state transitions. This value is a function of state frequency change.

In order to capture structural transitions, cost functions are applied over pattern nodes from the observed object's and pattern nodes from the reference's definition paths. From the results, the following can be inferred:

- features have been added or removed from either the reference or the observed objects
- features are altered (from either objects)

The above can reflect state transitions such as positional or structural (e.g. visual aspect); the object is moving from one position to another or it changed it's "appearance" to the input sensors.

Choosing Reference Objects

By default, reference objects are picked as the objects that have the minimal amount of (observable) state change (lowest value of cost functions applied over reference objects in distinct states). References, in task solving, are chosen w.r.t. the relevance of objects involved n the task; any object can be a reference object for a number if states in which the task is solved (the status of reference is dependent on the object's relevance in task solving).

As an example for behavioral state change, consider a car that is moving on a road from one point to another. If a task implies observing the car , then the road is chosen as a reference point by default as it has lesser state change than the car and the car's features (relative to position) are compared (have a cost function computed over features) with the road in order to capture positional change. Structurally (e.g. visually), the car's feature vary depending on factors such as angle of observation, lighting, etc. and are captured by cost functions.

For simplicity, in the above example, we consider that only one car is moving on the road and is perceived only visually; any other environmental details are discarded.

If the task and setting were changed so that there are two cars on the road moving, for example, in the same direction and the task would be observing one car then both cars are more likely to get picked as reference points to each other as state change is lesser between the cars than it is compared to the road.

In the examples up to now, each reference point was distinct from the object being observed. An object in its current state can be a reference point for past

and future states of itself. As an example, consider the shape of a triangle moving on a homogenous white background. Each state shows the triangle in a different position (assuming that observation time intervals are granular enough) than the previous. The current position is a reference point for previous and future observations. Conversely, past and future states are (will be) reference points for the current state.

Behavior captures feature changes relative to states and is represented as its own set of definition paths in the knowledge sphere. State changes have descending degrees of abstraction from the middle of the sphere towards peripheral layers which are more concrete in describing state variation, focusing on the highest weighted features of state variation.

As an example, a behavioral correlation can be made between a person and a car (assuming they have been observed moving in an earlier state) illustrating that both can move from one location to another (reflecting positional change). This correlation has a high level of abstraction since it applies to any object observed as moving and capable of positional change. Lower levels of abstraction focus on feature details that lie at the basis of the higher abstraction layers.

In the example, a person is likely to use bipedal movement while a car, the rolling movement of its wheels that contribute to its movement.

Correlations are weighted depending on criteria such as cost functions over nodes located in the same or topologically close layers, their task relevance, number, moment and duration of observation.

As an example, the correlation between two moving cars has a higher weight since the cost of the more concrete pattern nodes of their behavioral definition is less than that of a moving car and a person which differ substantially in their behavior (at least visually).

As structural correlations, behavioral correlations are cross-sensor; state change perceived one sensor type can infer variations of another.

For example, two sensors "seeing" and "hearing" a car; after a number of observations when both the structural and behavioral definitions are created, the auditive behavioral definition can be inferred from the visual and vice-versa.

As with structural correlations, behavioral correlations are managed (created, used and discarded) by meta schedulers both at layer and knowledge sphere scopes.

Figure 2.11 shows a behavioral definition (illustrated by dashed red vertexes) between the object observed in a first state and a second state (the state change implies that the observed object's features are distinct (at least from one type of features) from state I to II). Behavioral definitions need a reference state (in the figure, this is the first state the object is observed in) relative to which they are created as a cost function between at least two definition components (structural, behavioral, extra or meta) <<add further details as to how>> between patterns located on the same layers of the knowledge sphere. This approach is intuitive as it is expected that the state structural definitions are in, are topologically close (e.g. a car that is moving on a road has a number of state transitions that all map to the structural definition of the car).

Considerations:

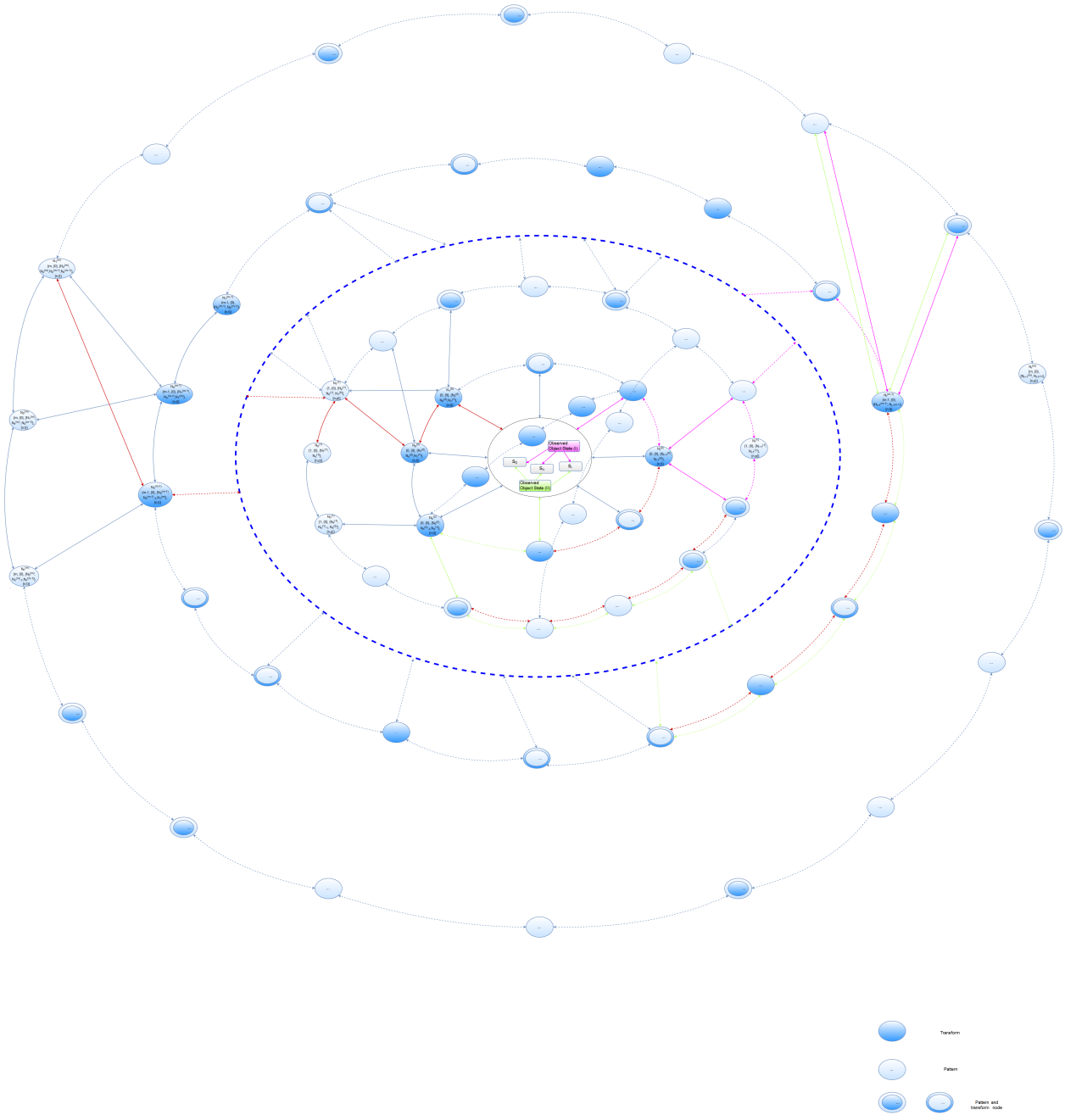


Figure 2.11: The two distinct states are marked with purple and green. The behavioral definition is denoted by the dark red dashed line

- behavioral definitions are time bound; an observed object's definition may prune after a number of clock cycles it is not used or its low or invalid weight in task solving. Each behavioral definition needs a reference state of an object with an existing structural definition
- state change observation is bounded by input sensor capture frequency (state change cannot be observed if it occurs at a faster rate than sensors are capable of observing; an observed object may move faster than the frame rate of the visual input sensor)
- in Figure 2.11 the definition path (comprised of nodes and vertexes) of the behavioral definition is marked in dark red. The premise of our approach is that any entity or property can be represented as a definition. A behavioral definition's role is to capture state change; the property characterizing the change and the amount of change (e.g. in the case of a moving car the most dominant state change would be its varying position (there is a set of weaker definitions such as spinning wheels, angles if the car makes a turn, etc.).

Meta Correlations

Use the meta definition component of observed objects to manage structural and behavioral correlation definitions. As mentioned in Section 2.3.1, meta definition components work with node attributes instead of their pattern values such as moment and duration of observation, task definition look-up/creation, network topological definition approximation for tasks (how close w.r.t. a cost function is the definition of the current task compared to the previous input tasks) and task relevance.

Meta correlations make use of meta schedulers defined at layer and global (knowledge sphere) scopes. Figure 2.12 shows two correlations (related to moment and duration of observation) marked with dashed green and red edges between three objects. The color intensity shows the correlation's weight; darker colors representing higher weights (object I is more strongly correlated with object II w.r.t. moment of observation and less correlated with object III while object II and III are more correlated w.r.t. their duration.

Meta components control definition creation look-up and creation by adding or removing (transform/pattern) nodes from the network and, altering node and (structural and behavioral) definition weights depending on task relevance. They are the "consciousness" of the algorithm. Meta components also control definition node lifespan by pruning (discarding) nodes belonging to definition paths that are not used (looked up) within a time period defined as a number of clock cycles. This time period is a function of definition path degree of use and task relevance, and is node granular in that definitions are pruned gradually, from the most concrete nodes, in the outer layers, to more abstract, inner nodes.

Next we are introducing the concept of needs that allow the knowledge sphere to expand in terms of computational and storage resources thus allowing for more elaborate task solving. Needs are represented as tasks with a variable

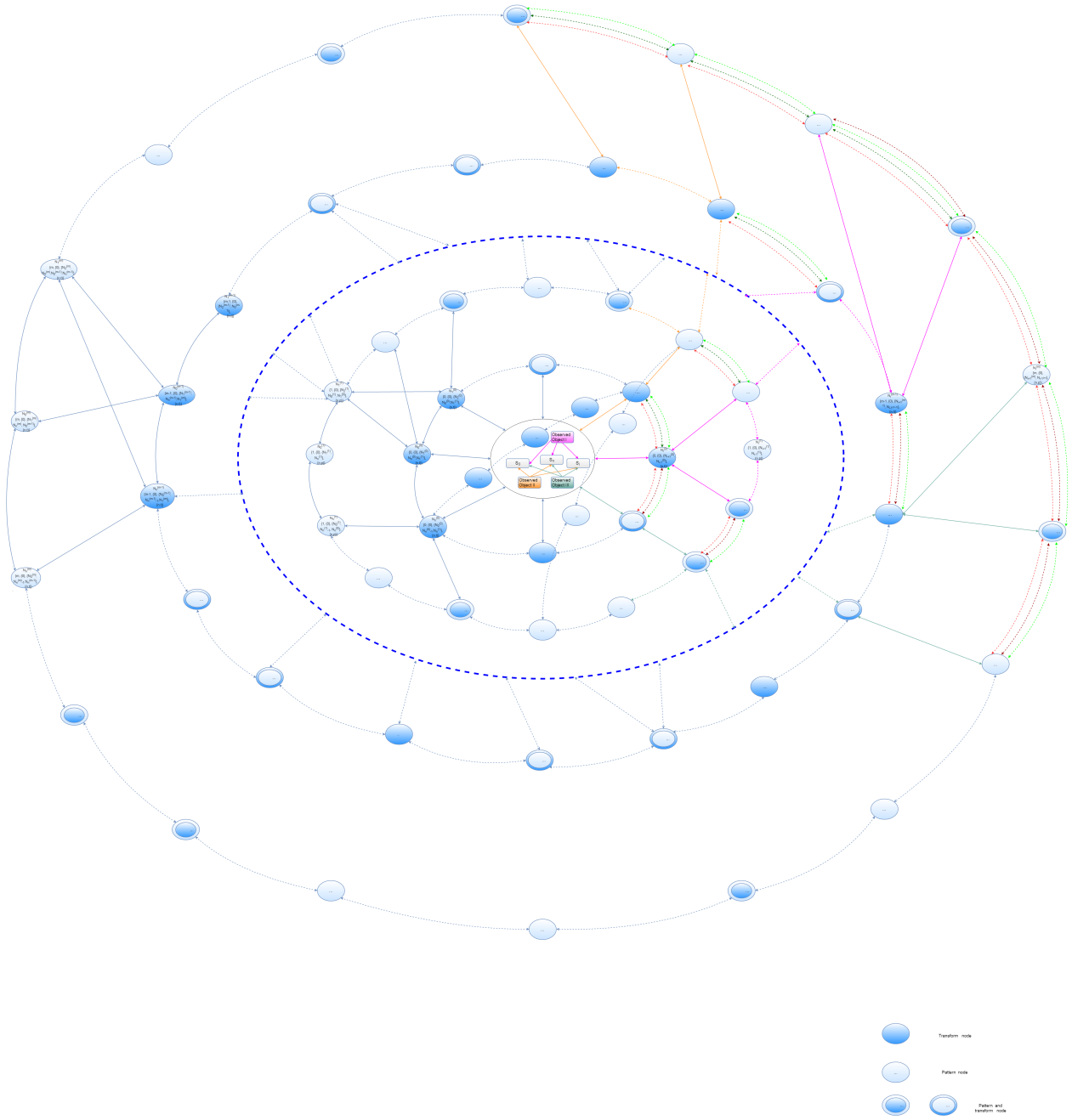


Figure 2.12: Correlations are shown as dashed edges between nodes on the same layer (sphere). Green edges represent the moment of observation while red edges its duration. Color intensity indicates correlation weight; dark edges have a higher weight than light ones. Each object is correlated to the remaining objects; the correlation weight w.r.t. the moment of observation is greater between Object I and II while Objects II and III have a higher weight w.r.t. observation duration. For clarity, correlation edges have been ommitted from the second layer.

priority attribute. Note that all tasks have the priority attribute but are solved only when input while needs are similar to a background process that are always allotted computational resources for solving. There exists a core set of needs that have a higher priority than possibly the current input task. These needs, if not met, affect the physical structure of the knowledge sphere (e.g. being destroyed by a process not controllable by the sphere; an external factor). This represents the need for "survival". Another need represents the goal of the knowledge sphere of expanding; permanently acquiring new computational and storage resource for the newly observed object definitions and tasks.

2.4 Definition Merging

The data set for observed objects can grow to an impractical size as objects are being observed and definitions created for each. To this end, we propose to merge nodes that have a similarity degree higher than a threshold value. The resulting node has a weight computed using an averaging function over the weights of features it has been merged from. Concerning definition paths, merging takes place between pattern nodes, resulting paths containing nodes with averaged patterns. Figure 2.13 shows two definition paths (highlighted with purple and green) that have a number of nodes that are close to each other w.r.t. a cost function over their features (nodes that share similar features are close topologically and are more likely to be merged on future runs of the algorithm (definition look-ups/creation)). Figure 2.14 shows the nodes on the first layers being merged into a single node and a new definition path that both definitions from Figure 2.13 share up to a level of abstraction (the final nodes of the path containing the concrete instances of the observed objects remain distinct).

Considerations:

- merging reduces the number of nodes by replacing an existing set of nodes that are similar feature-wise with a single node encompassing an averaged pattern (reducing the number of "duplicate" nodes in the network)
- merging is more likely to occur on outwards (concrete) layers since the feature differences decrease with the increase level of concreteness
- both pattern and transform nodes are merged <<detail intuition behind transform node merging>>
- nodes from which a merge node is created from are discarded (pruned) and the new node used in definition look-ups/creation

As an example, if a car is being observed from multiple angles, the robot can conclude that each image corresponds to the same definition of the car and the observation definition paths are merged into its definition.

If there are no similar or identical features for an observed object, the best matching set of current definitions is chosen for the object (definition path

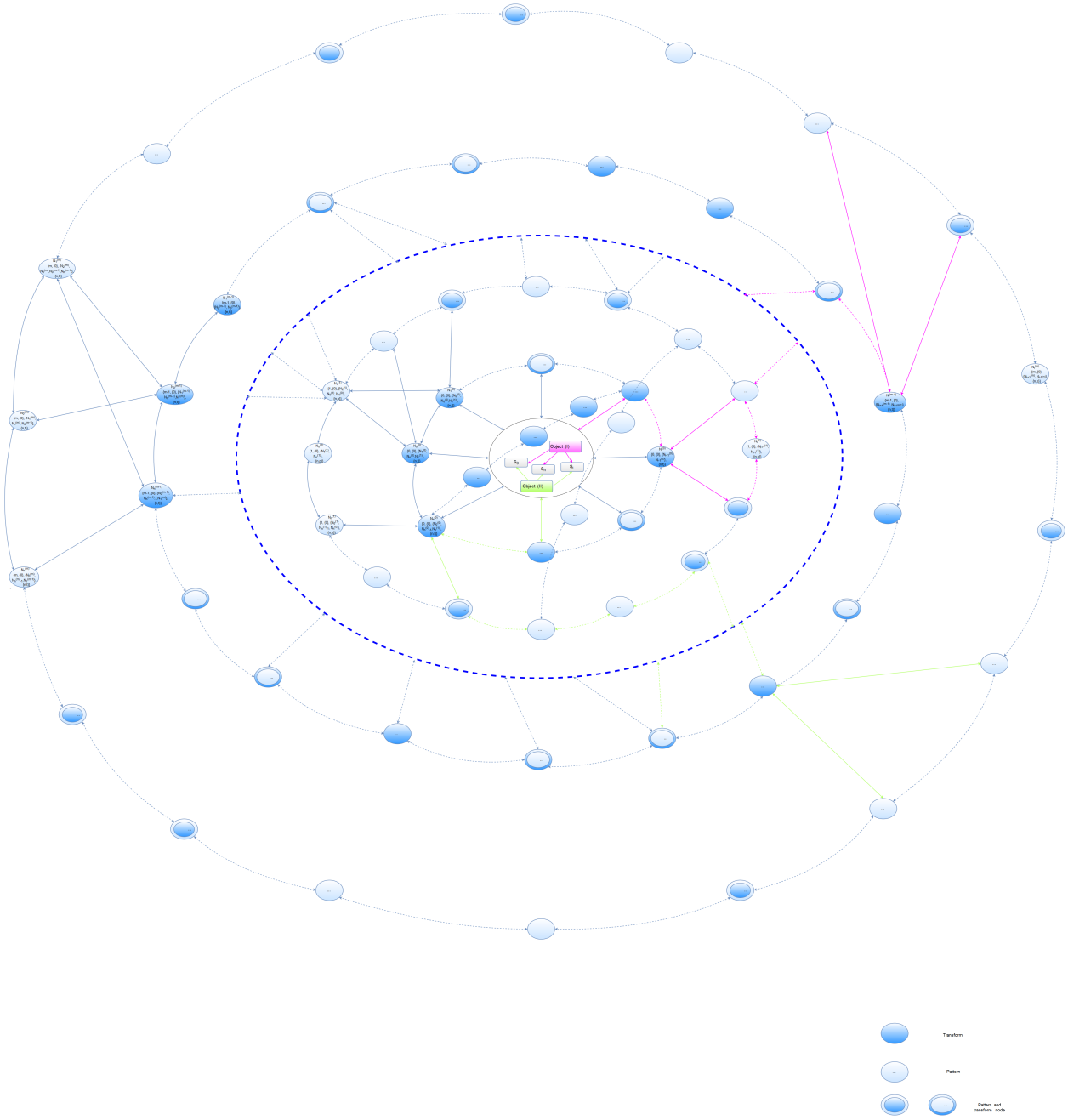


Figure 2.13: shows two definition paths for distinct objects (marked with purple and green vertexes) prior to their merging. Their pattern nodes having a high degree of similarity are closer topologically within the spheres

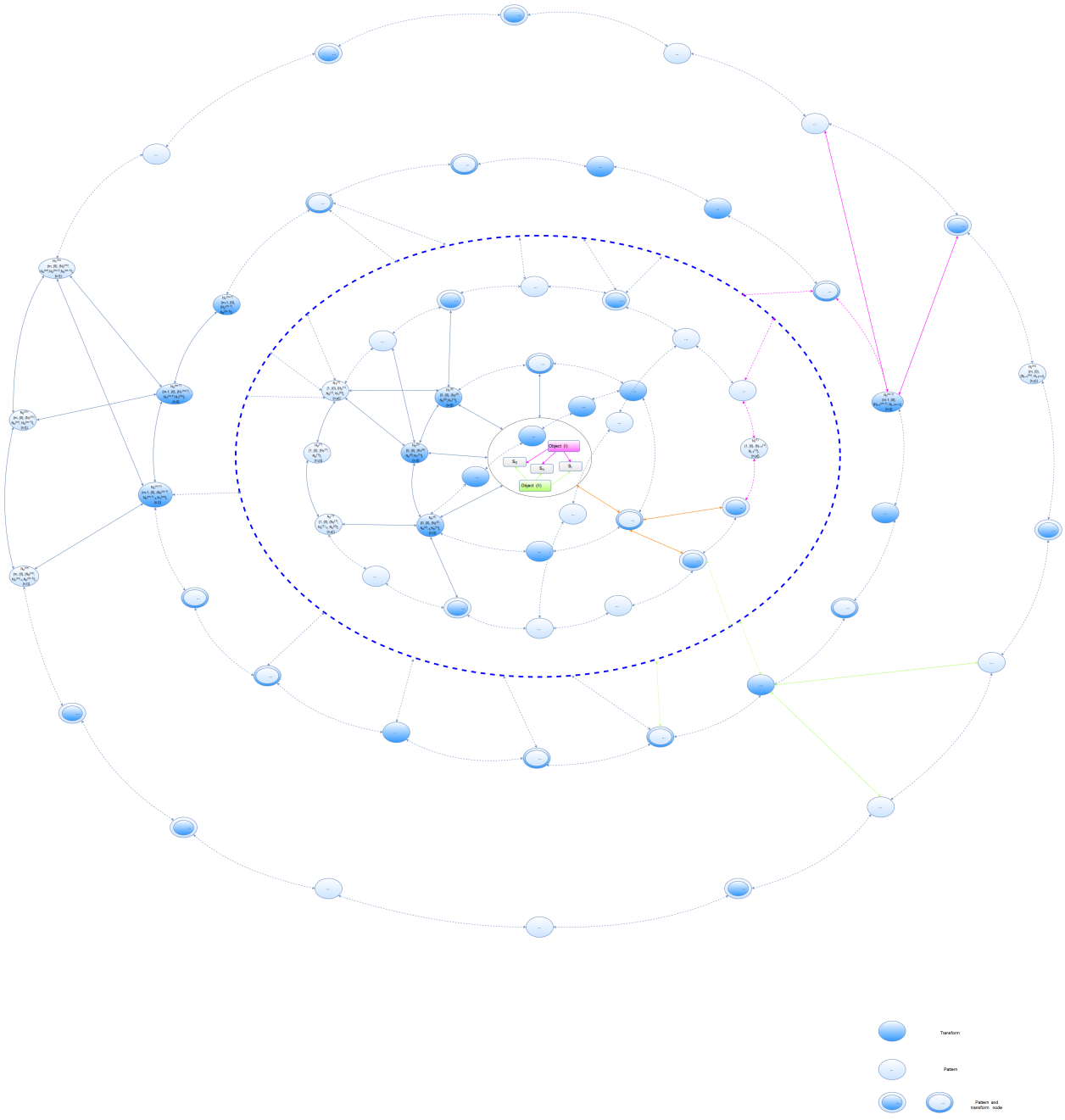


Figure 2.14: shows the definition paths after a set of nodes in their definition paths have been merged.

creation is done as described in Section 2.3.1). These definition paths will have sets of nodes that are going to be merged as more objects with similar extracted features are observed over time.

At a conceptual level, a definition essentially groups a set of objects on the basis of some criteria (set of correlated features) and is cross-sensor in that the feature set used for binding is received from more than one type of sensor as shown in Figure 2.2 where nodes containing multiple patterns (on the figure they are marked as two or more overlapping elipses of different colors).

Cross sensor definition considerations:

- when the object is perceived initially, a definition path is looked up or created for data input from each type of sensor (e.g. both visual or auditive). Definition nodes contain patterns specific to a data type (initially there are no nodes containing patterns matching two or more data types)
- nodes are merged (correlations are created) as a function of:

1. Observation duration: if definitions are created or looked up for different data types in a time interval smaller than a threshold, a correlation is created between their nodes (the nodes are merged probabilistically and will contain patterns matching the respective data types). As an example, if a car is both "seen" and "heard" nodes within the visual and auditive definition paths are likely to be merged on the observation frequency; for how long does the robot perceive cars that share a degree of similarity both visually and by the sounds they produce in a bounded time interval (the closer the time difference different types of data are received, the higher the likelihood of a correlation to be created)
2. Observation frequency: increases the likelihood of merging as a function of the number of observations of objects; if similar objects are perceived by sensors of different types in a bounded time interval , for a number of times the likelihood is increased
3. Pattern similarity: structurally, if two nodes hold patterns that are lesser than a threshold given by a cost function, they are merged into a single node

Initialization

As mentioned previously, patterns are created by applying an averaging function over features extracted from observed objects that have a cost difference less than a threshold value. However, in the incipient phase, there are no patterns defined (the robot has not yet observed any objects). They can be either created by:

- observing multiple objects and using a feature extractor set to extract features in varying levels of details and define a pattern as the result

of an averaging function applied over them. This stage creates multidimensional definitions (structural, behavioral, etc. correlated across input sensor types) through abstracting the feature sets across input data types

- or defining an existing pattern set that has a high enough level of abstraction that it can match any observable objects (by splitting the observed data into subsets that each have a matching pattern and then creating the definition by mapping each of the split data definitions)

In our approach we propose the use of an existing pattern set as it is computationally less expensive than creating patterns through observation.

Chapter 3

Task Solving and I/O

3.1 Introduction

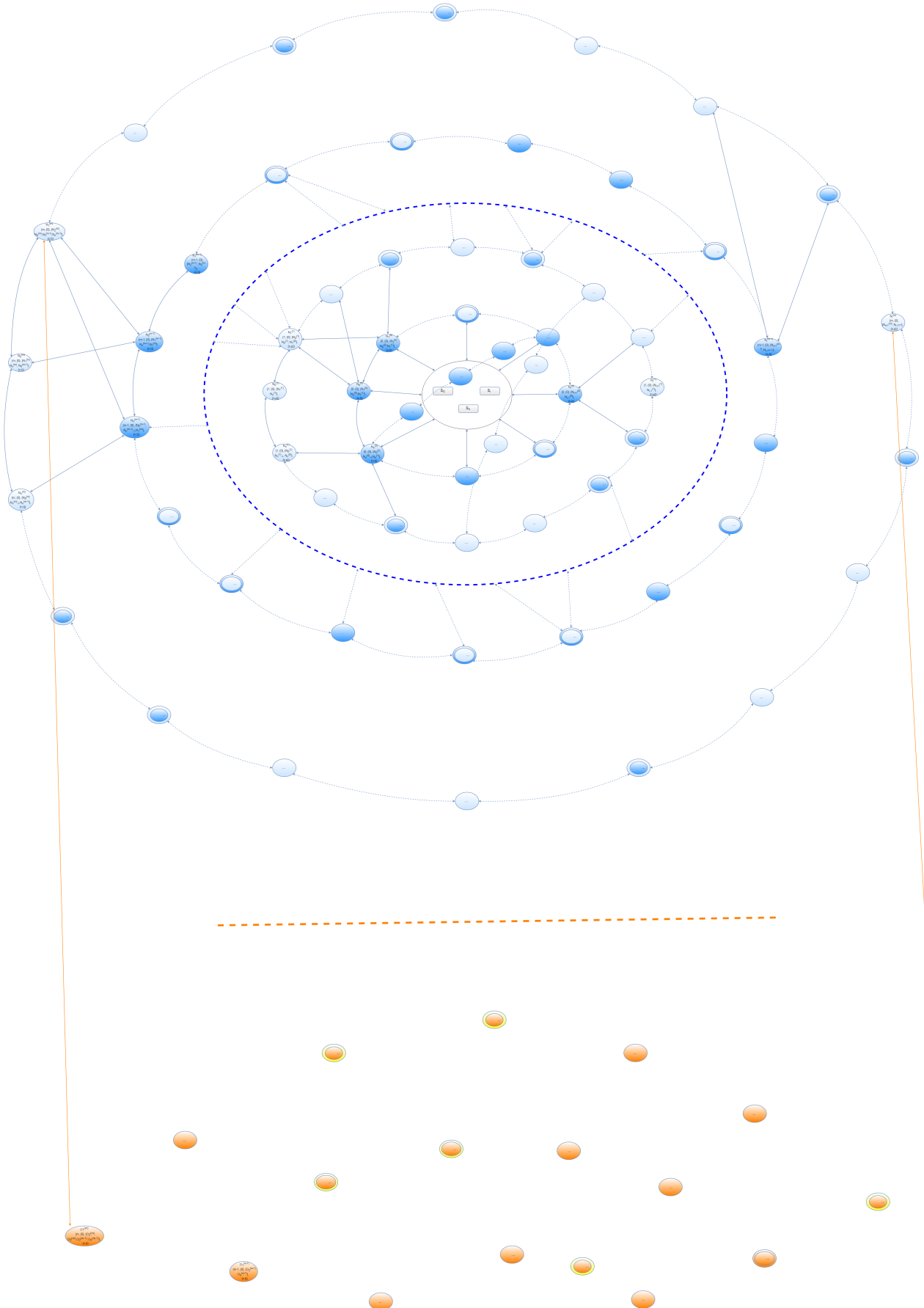
In Section 2.3.2 an example of task solving is shown for an individual task given in written form.

Task solving follows the steps:

1. Break the task into, individually definable, sub-tasks
2. look-up the definition of each sub-task
3. If found, wait a number of clock cycles for the remaining sub-tasks to finish
4. If not found, break the sub-task into a set of comprising sub-tasks
5. Repeat steps 2 and 3 until all sub-tasks are at most atomic
6. Combine all sub-task definitions to create a task definition
7. Return a set of best matching definition paths from the knowledge sphere

Up to this point there has been no specific description of how data is conveyed from the knowledge sphere to an external environment. Outputs can be retrieved (read) from nodes regardless of their topology; each node is able to output data matching a pattern regardless of its type (a node can, for example, output both visual and auditive data). Figure 3.1 shows generalized output model for a knowledge sphere. A fully generalized model implies that each node can be connected to each output as well as outputs being organized in clusters that output values for an aggregate of nodes instead of a single node.

A key note is that, upon retrieving nodes from definition paths, the most concrete (furthest) nodes are returned first. Depending on the task, abstraction level verbosity can be controlled by either specifying the desired level of abstraction specifically, through the task, or enable a default behavior in the algorithm



returning values only from a specified length of definition paths (starting from a specified node).

There are two cases in which steps 1 to 7 do not output a valid task result:

1. The current definitions in the knowledge sphere are not valid w.r.t. task; the information stored in nodes does not produce a valid output

2. One or more sub-tasks do not have a previous definition in the knowledge sphere (the task isn't "understood" entirely)

As an example, for the first case, a task "What are the last blue cars observed?" input to a knowledge sphere that has no pattern matching blue cars would lead to a set of best matching definitions to be output for that task. In the example's context these would be all observed objects fitting the structural, behavioral and meta definition of a car up to a threshold value.

A similar approach is taken for each sub-task for which there exist no, or not valid definitions for it.

If a definition for a task is not valid after completing step 7 two approaches can be taken:

1. The robot probabilistically returns the best matching definition paths for the task

2. The robot proceeds to learn the respective notion (unrecognized observed object) required for task solving starting from current definitions

We mentioned that a set of definition paths can be invalid for a task but did not focus on the notion of validity. A task is essentially a query to retrieve definition path subset of nodes from the sphere. Initially, a training data set is used to create definition paths as well as an input set of tasks to setup correlations between the definition paths (the tasks are valid w.r.t. the definitions).

A task is considered invalid if at least one of the following conditions is met:

1. The definition is incomplete in that the current level of concreteness is not sufficient to solve the task satisfactorily

2. The desired object required by the task is not present in the sphere (the sphere always returns the best matching result for a task. In this case the result is not relevant w.r.t. the task consequently being invalid)

3. The definition path does not solve the task w.r.t. its context, a context being defined by constraints enforced through correlations (for example, temporal constraints enforced by meta correlations)

In the first case, the definition is extended with a more concrete set of paths with nodes belonging to an external knowledge sphere that is likely to provide a valid definition for the task input to the first sphere as shown in Figure 3.2; the last most concrete pattern nodes from the task definition in the first sphere are used as an input to the second sphere. Nodes are input to the second sphere starting

from the most concrete to the most abstract in order to achieve a gradually increasing definition path set (using the inputs directly from the first sphere can produce definition paths that are too detailed for the scope of the current task and impact performance negatively).

If conditions 2 and 3 are not met, the task is converted to a "need" and assigned a priority.

Since tasks are subsets of sub-tasks, each sub-task must be valid in order for the task comprising it to be valid. For example, a sub-task can be recognizing letters in the body of a task (assuming it was input as written text). While definition paths for written words can be pruned (the meaning of a word "forgotten") the letters are more likely to match low granularity definitions due to the higher frequency of use (tasks provided as written input text require a look-up through the definition paths of each letter present in each word describing the task. Consequently, the time till prune value of the nodes defining letter definition paths is increased).

The approach, that of learning (observing) an object required by a task assumes that, currently, there are no definitions matching the task as a whole (there exists a non-empty set of sub-tasks that have no matching definition paths (in written tasks, the words are not defined in the knowledge sphere; are either input for the first time or "forgotten" due to pruning)). Meta correlations (making use of features such as time and place of object observation as well as scheduling structural and behavioral correlations) are used to find objects that are most likely to match the definition of the unknown sub-task.

As an example, if the input task is given as written text and a number of words are unknown/not "understood" (do not map to definitions), the algorithm "might" choose to use a dictionary to look-up their meaning and then create a mapping between the word and its definition path. Finding definition paths for unknown words represents solving a sub-task that is constituent of the main task. We have used the term "might" for the option of using a dictionary as it is only one option of finding the word's meaning (and in turn its respective definition path in the sphere).

A more trivial approach is to display the unknown word from the task at an output and inputting its written definition. When input:

1. A set of definition paths is created for the new word (the word's meaning is learned) and a mapping created between its text representation and the set of paths

2. The definitions in step 1 are used in (sub-)task solving. If a word from the text definition is not found (the initial term was defined using terms that are in turn undefined (do not have a mapping between a written word and a set of definition paths)), step 1 is applied for the respective word and its definition (provided as text input) used in the previous definition. This step is repeated recursively as long as there are unknown words used in defining terms used in (sub-)tasks

In a previous example, we mentioned that the algorithm "might" use a dictionary

for looking up text definitions of written words present in sub-tasks. The reason to this is that a dictionary is not unique w.r.t. providing definitions for words in tasks; an approach of using an external source such as requesting the definition of an unknown term of a (sub-)task becomes more likely depending on:

1. The (sub-)task relevance of the provided definition (meaning)

2. Time constraints; how much time does a dictionary look-up take compared to inputting the task's definition (obviously, a look-up is significantly faster than an input from a human operator).

The latter point is relevant to the case where a knowledge sphere queries another for a (sub-)task's definition. As mentioned in Section **??**, a knowledge sphere can reference, and be referenced by a set of distinct (external), knowledge spheres. This is accomplished by creating vertexes between nodes belonging to different spheres (implying there is a physical channel connecting the spheres) allowing for sphere specialization in definitions of object categories.

A (sub-)task definition look-up in this context is done by:

1. Looking-up the definition in the current sphere

2. Assuming that the definition is not found or the set of returned definition paths is not valid for the task w.r.t. the constituting sub-tasks, subsequent look-ups are made in specialization spheres. Figure 3.2 shows two knowledge spheres, where a sub-task definition of a task input to the first (marked by I in the figure) is looked up in the second (as the first does not offer a valid definition for the task it's part off). Each node in the path has an edge to the entry point in the second sphere from where the look-up is continued. Upon completion, the resulting definition path spans both spheres. Definition path node duplication is thus, avoided. Another option would be to copy the definition path's nodes from the specialization sphere to the initial sphere the algorithm which would more closely resemble the human learning process (or the static linking of a library to a binary). The meta definition component controls whether the path is copied into the first to improve look-up speeds based on task validity and frequency of use.

Coming back to the previous example, a dictionary can be defined as a specialization knowledge sphere that receives as input terms that do not have a valid definition path w.r.t. tasks in a (possibly) distinct set of knowledge spheres (as an analogy more people can use the same dictionary as well as multiple binaries can link to a library). Note that a dictionary/specialization sphere can be a separate instance of the running algorithm that has received different inputs (observed different objects) than the current instance (as an analogy, a person can ask another about the meaning of a term).

Assuming that the dictionary is a physical book, when observed initially visual features are extracted and a set of definition paths looked up/created

probabilistically from the best matching patterns of similar previously observed objects (e.g.parallelepipedic objects, other books, etc.).

Considering that the dictionary lies on a surface and there is no perceivable state change (its behavioral definition component is limited; there are no empty definition components. The lack of observable state change is a state of its own), the algorithm will attempt to make correlations with similar object definitions observed up until that point. If a behavioral correlation is found such as opening a book (whose structural feature set is similar to that of a dictionary), turning its pages and reading their content, the algorithm will apply the same behavioral steps for the dictionary.

In this example, for the sake of simplicity, we do not consider details such as the angle the dictionary is opened and viewed from, the position it's held in, the number of pages that are turned and the frequency at which data is read.

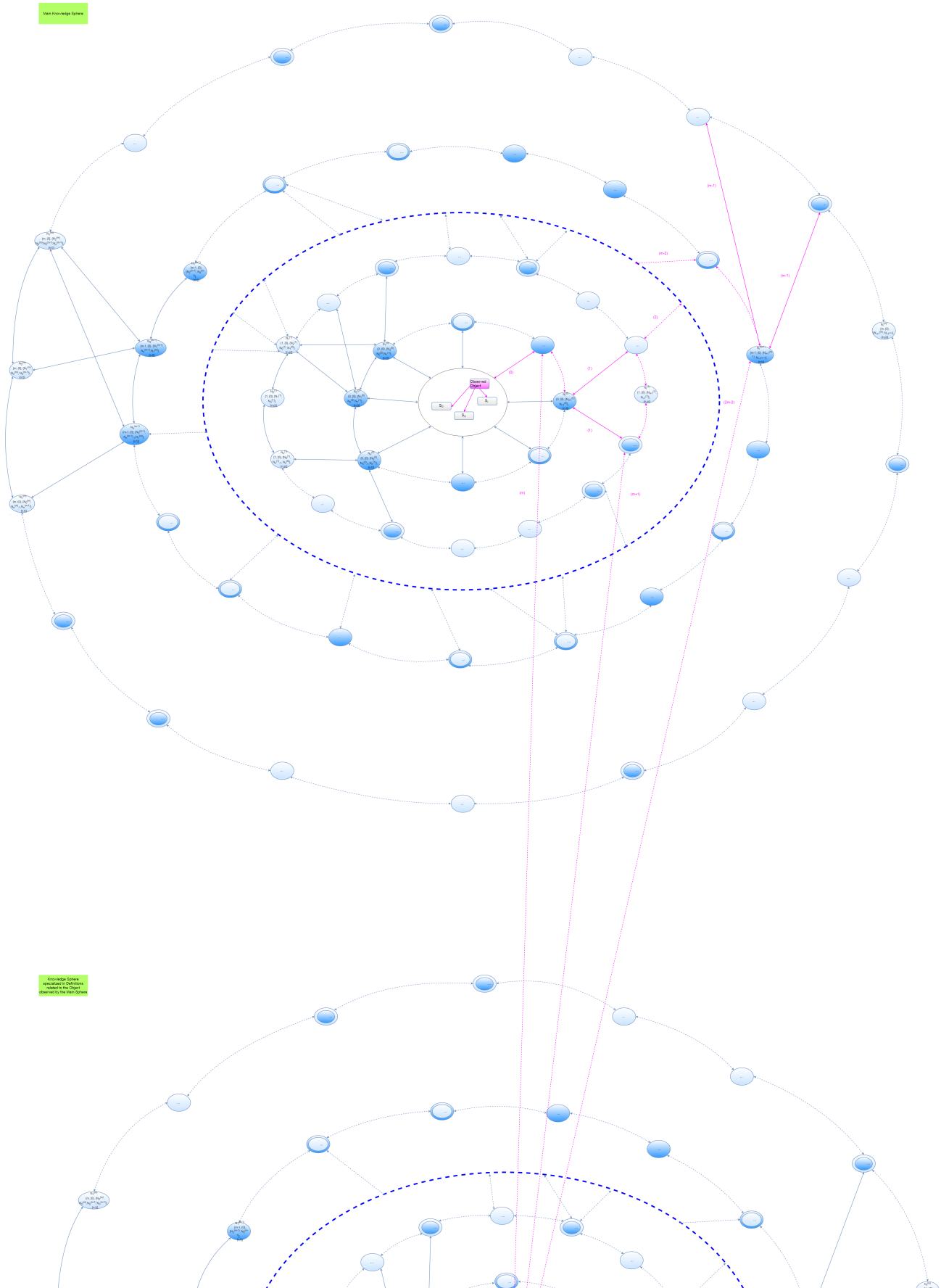
A dictionary contains mappings between words and their meanings which are described using a more common set of terms. The robot is likely to see new words that have not yet been stated in tasks. Regardless, definition paths are created for each word. Their life span is determined by their degree of use in future tasks; if there are no tasks containing them they are pruned ("forgotten" from the knowledge sphere). The initial life span value is low for each object as long as it is not required in tasks.

The meta correlation created w.r.t. the dictionary is that it is likely to contain words that do not have a valid definition for a task (are "unknown"). The attribute of "unknown" is a meta attribute applicable to all objects for which there is a lower usage than a threshold or nodes that have been already pruned and the definition paths they were part of are no longer sufficient for task solving. However, if multiple tasks require a specific object definition currently not present in the sphere then a look-up is made into a set of accessible specialization knowledge spheres or the robot tries to learn based on observation <<This is explained in the next section>>.

In the previous example, if the task and its comprising sub-tasks contain words that the robot does not understand then it proceeds to follow the steps of looking up the unknown term present in the task, in the dictionary.

A dictionary is only one source of data from where definition paths can be created for unknown terms. Another can be through the input sensors, by requesting their definition using an output. Similar to nodes and definitions, data sources are weighted w.r.t. the validity of the definitions they provide in task solving (actually each source has its own definition in the sphere). If definitions provided by a source have a higher task relevance than others from a distinct source, the weight value is increased making it more likely to be used in future tasks.

The next section discuses specialization knowledge sphere creation and usage.


Main Knowledge Sphere
Observed Object
Knowledge Sphere specialized in Definitions related to the Object observed by the Main Sphere

3.2 Specialization Knowledge Spheres (Libraries)

In the previous section we mentioned how the algorithm can delegate definition look-ups to a distinct knowledge sphere containing more specialized (containing longer, more comprehensive definition paths) in objects of the same type (that have higher correlations between them than similar objects from the "calling" sphere). Specialization essentially indicates that there is a higher likelihood that the definition looked up in that sphere is valid for tasks of a particular type.

For example, looking for car definitions in a sphere created by observing cars is more likely to retrieve a valid result than one specializing in face recognition.

Library creation can be done by:

1. using a sphere trained in a specialization of objects that in turn will have definitions looked-up from distinct spheres
2. extracting definitions matching a specialization from the current sphere and creating a separate instance to which definition look-ups are delegated to from the current as well as distinct spheres. Once the library sphere is created, the original definition paths are removed from the initial sphere (nodes with high degrees of abstraction located near the input sensors cannot be pruned as there are definition paths that depend on them and are not related to the newly created specialization sphere).

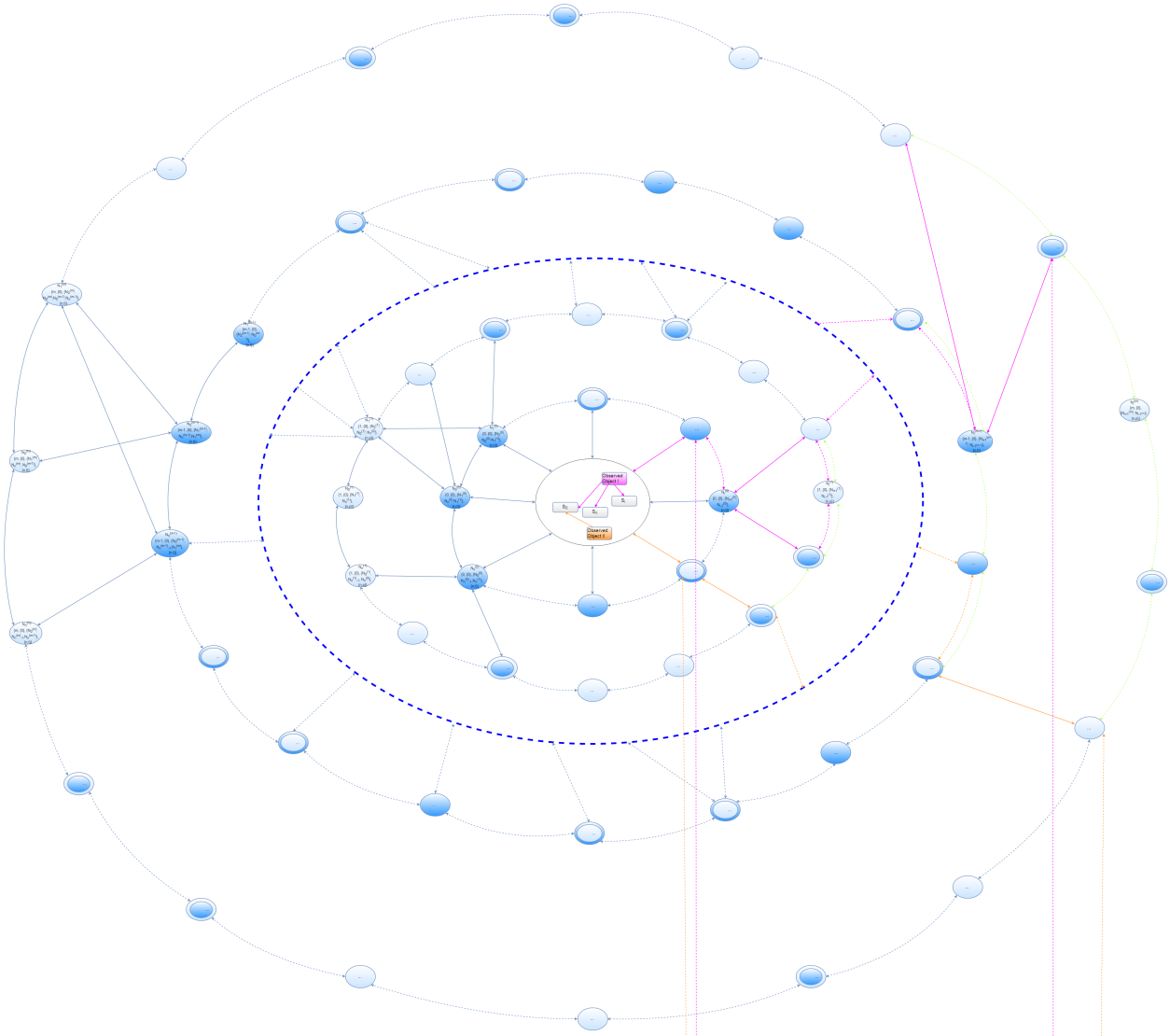
We haven't talked about how specializations are represented; they are sets of meta correlations comprised of structural and behavioral correlations. Objects of a specialization must have correlations of a weight greater than a threshold on at least a definition component over which a specialization is built.

Figure 3.3 shows a specialization sphere created for two correlated objects. After the specialization sphere is initialized the concrete nodes of the object definition paths in the originating sphere are removed (the resulting sphere is shown at the right). The most abstract nodes are kept as they lie at the basis of distinct object definition paths that would become unreachable (definition look-up would become impossible for future observed objects).

3.3 Tasks and Needs

Needs are, essentially, meta scheduler policies that govern the knowledge sphere's structure and behavior (the rules according to which definitions are created and pruned (by adding or removing nodes from their paths)). We define a default set of policies that aim to expand the sphere w.r.t. definition paths and their respective lengths (the algorithm is "always learning" and improving its knowledge base). The policies instruct the sphere to:

1. Observe objects (look-up/create definition paths)
2. Execute a task of acquiring computational/storage resources when there is insufficient space/processing power to store/look-up/create definition paths


Observed Object I
Observed Object II

3. Repeat steps 1 and 2. When resources are scarce increase the weight (relevance) of the task definition sets that were relevant in acquiring them (we do not consider the initial sphere that is always assumed to be available)

3.3.1 Expansion Step

This step characterizes default behavior of observing objects. If no new objects are observed over a period of time the robot will use locomotory functions to expand the perception area of sensors. New data is learned with a higher weight on objects related to current and past unsolved tasks.

3.3.2 Resource Acquisition

When computing/storage resources become insufficient, an acquisition task is executed and, if found, the resources are used to allocate and instantiate a new knowledge sphere to which future definition path creation/look-up will be delegated to.

The acquisition task is input similarly to conventional tasks (e.g. written form) with the exception that it runs in the background (the robot is permanently looking for resources) and it's priority increased when resources are below a threshold. Unlike conventional tasks, it is stored in the meta scheduler having both layer and sphere scope both in the originating knowledge sphere as well as instances created from newly acquired resources.

The process works by defining resources (structurally, behaviorally, etc.). The robot will try and identify resources from observed objects as well as finding them on its own. Figure 3.4 shows an example of outputs from a knowledge sphere. Note that outputs are not topologically bound in a sphere; the figure focusing on the case of a text terminal where both inputs and outputs are done using the same device. A generalized model is shown in Figure 3.5.

The outputs are connected to locomotory devices that allow for modifying sensor position which in turn allows expanding the observation field (more objects are observed; more features extracted). Higher feature extraction implies more accurate pattern definition matching as well as smaller look-up times (e.g. in the case of recognizing objects based on their distance from the sensors; close objects being more feature rich are more quickly identifiable than distant objects that can match higher level of abstractions and thus lead to invalid definitions) as less features mean more definition paths the algorithm has to create/look-up in task solving. Input sensors as well as locomotory devices classify as resources and are, consequently, part of the acquisition process. This is due to the fact that using a small set of inputs and outputs becomes a bottle neck for task solving. Increasing input/output and storage/computation capacity ensures that the robot's (all of its knowledge sphere) lifespan is extended and that task solving is improved as the available definition path set are extended.

Topologically, outputs are created so that definition look-up speeds are improved; outputs from one sphere may be created and linked to the inputs of a specialization sphere the first sphere requires in task solving.

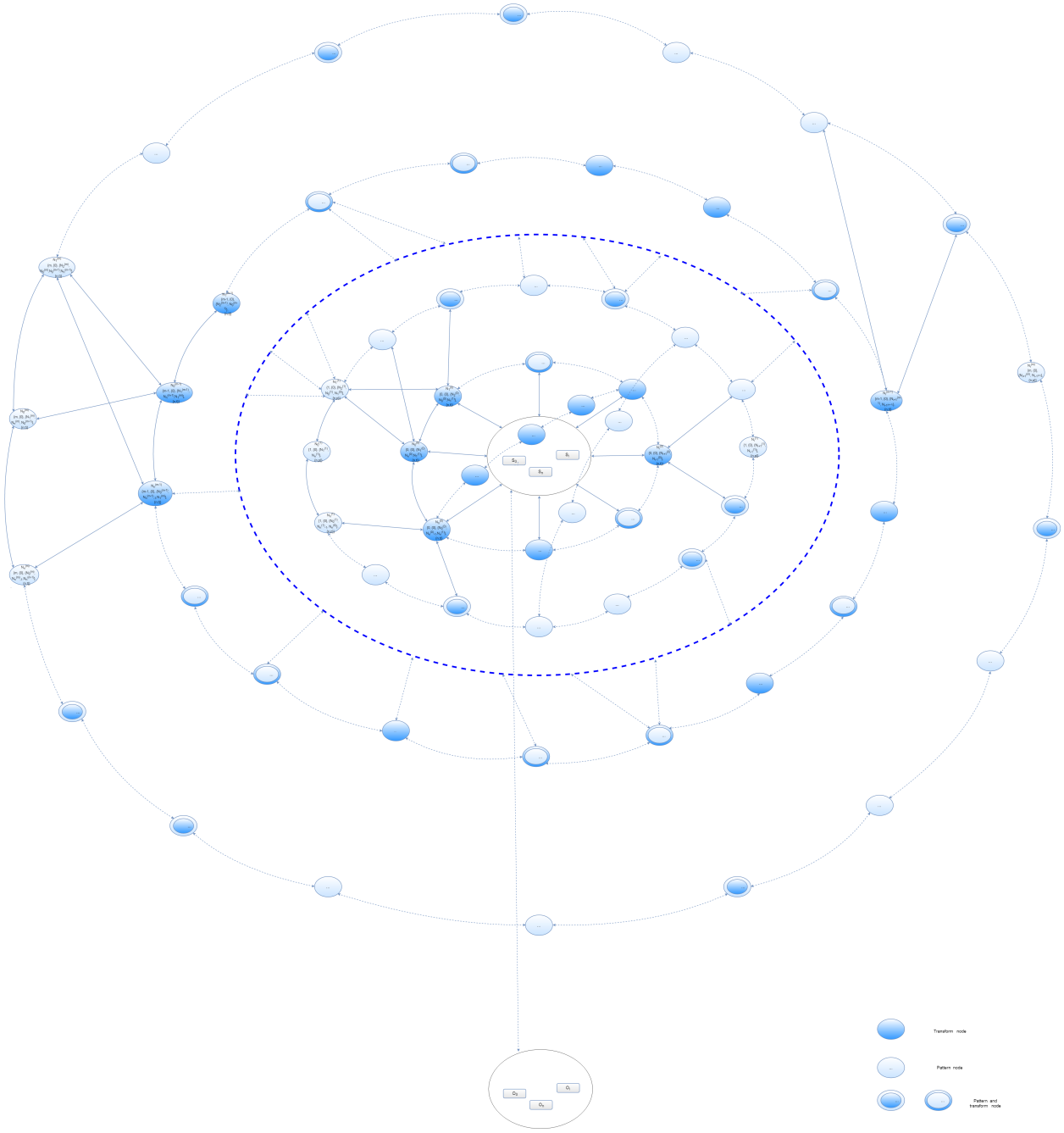


Figure 3.4: shows a particular case where there is one output set for a sphere. The edge from the input sensor to the outputs only show that the respective outputs belong to that sphere.

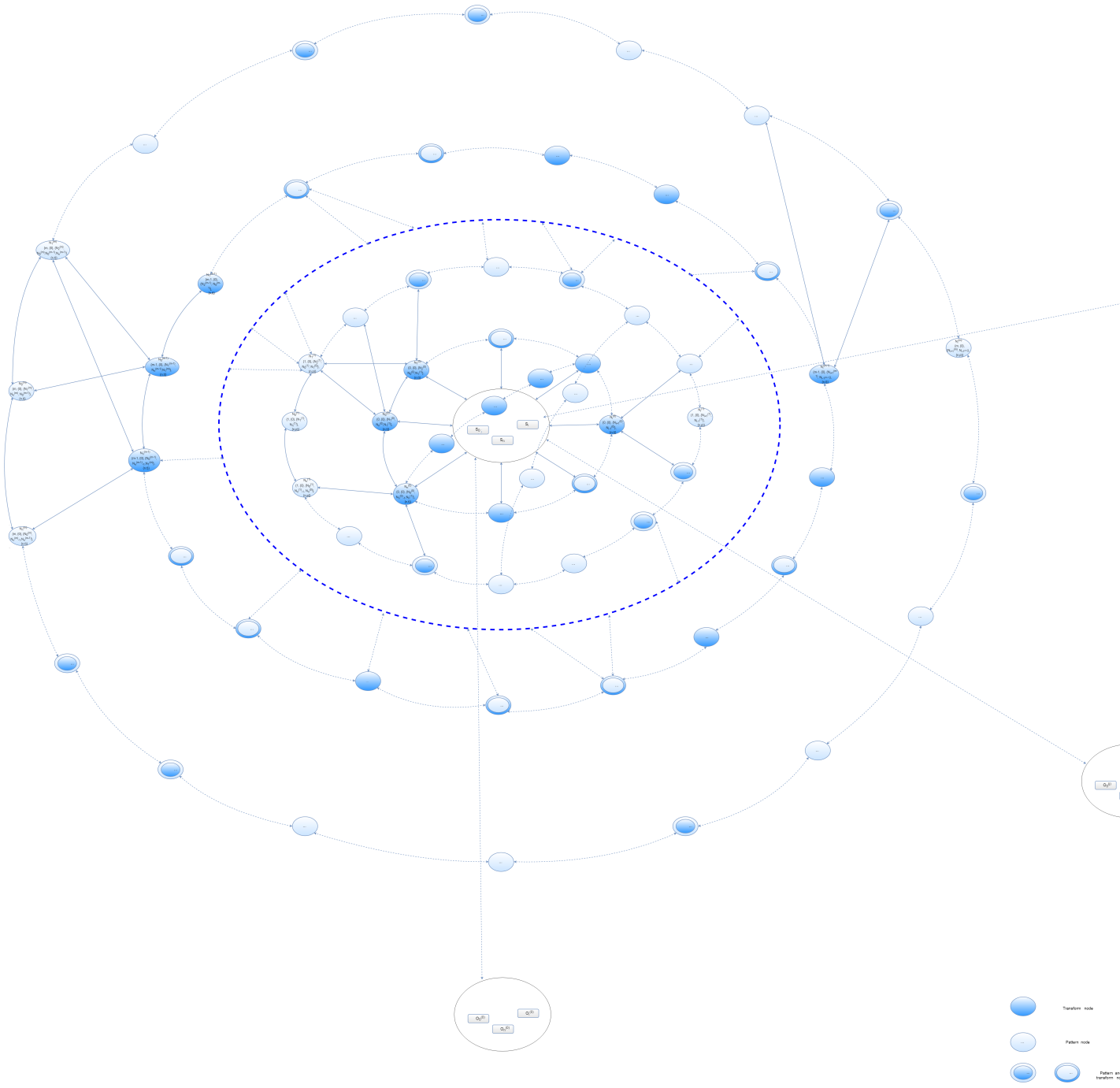


Figure 3.5: shows a generalized model with multiple output sets.

This approach also ensures robustness in that, even if a subset of spheres is removed from the system, as long as there exists at least one sphere containing the most abstract patterns, the learning process can always be restarted.

Previously, we mentioned that the robot is able to perform locomotory functions for resource acquisition. This task is complex; the algorithm must be able to:

1. Change input sensor position (e.g. "look around") to increase the observed object set and thus the likelihood of finding resources (which, probabilistically, are a subset of the observed object set)

2. Recognize resources; the robot must make a distinction between resources and objects that cannot be used as resources w.r.t. in/outputs and storage/computing capacity. It should be noted that even though an object might not directly fit the definition of a resource it can be processed by a set of functions so that the end result is a suitable resource

As an example for the second point, suppose that the robot requires computing resources in the form of a family of chipsets. The means of acquisition can be the following (or a combination):

1. Structure the whole manufacturing process of the respective chipsets into a task comprised of sub-tasks each having a valid exisitng definition. The sub-tasks would include steps such as retrieving semiconductor material (Si), using or creating integrated circuits for the respective chip, etc.. This approach is prohibitive in terms of cost and computational complexity

2. Use readily available resource acquisition mechanisms so that, once resources are required, a signal is sent to an output (the task of requesting resources through the output is learned beforehand). For example, outputting a console value and in turn, an operator attaches a new resource to which the requesting sphere can link to. Sensors can be used to monitor a shared medium (e.g. bus architecture) in order to detect the newly attached resources. Once detected, meta schedulers of the most abstract layers are notified which in turn broadcast a signal to with lengthy/dense definition paths that can be stored in the new sphere. Note that the first nodes to be copied into the newly acquired sphere are the most abstract nodes from the originating knowledge spheres. Definitions are then copied/created in the new sphere in order to improve robustness

3. In more advanced stages of runtime, the robot is able to formulate more elaborate steps of acquiring resources based on a comprehensive set of correlations that define each object constituent of a resource w.r.t. a set of correlated objects from which it can be acquired. A scenario would consist in, for example, the robot performing a transaction (buying) with an operator of the needed resource. At this stage, the robot is knowledgeable enough to integrate new, purchased, chips with its architecture

Step 3 implies that a resource is defined inside the sphere as well as its acquisition and integration steps (modelled as tasks).

Tasks are split into sub-tasks, each sub-task having an assigned priority managed through a function <<detail later>> present in the meta scheduler. The priority attribute is part of the meta component definition paths of the objects relevant to the sub-task.

The function represents a means of structuring over how sub-tasks are completed and grouped into the main task (which can either be a conventional task input to sensors or a need that is a background running task with an assigned priority).

Figure 3.6 shows a solution to a task comprised of two sub-tasks (marked with purple and green) each having its own definition path.

As mentioned earlier in this section, task solving is performed by splitting the task into sub-tasks until each sub-task becomes atomic, looking up their respective definitions in the current sphere, a library (specialization sphere) or an external (predefined) source (e.g. terminal input). The solution to the task is an intersection set of best matching definitions. In Figure 3.6, two look-ups are made for sub-tasks I and II and definitions looked up for each (for the sake of simplicity we assume they're atomic and cannot be split into further sub-tasks. Their definitions are also present in the sphere and mustn't be learned). Once both definitions are looked-up, the task can be solved as a query of the object definitions they point to (the sub-tasks are merged into the initial task).

As an example, such a task can represent a request to the robot of moving two objects, one square and the other round shaped, structurally and behaviorally independent (the goal is to reduce correlations as much as possible).

While, in the above example, the order in which sub-tasks are solved is not relevant for task validity, ordering becomes necessary in more elaborate tasks where sub-tasks depend on each other w.r.t. a set of attributes such as the time and order in which they are complete, etc..

Such tasks require correlations between their sub-tasks which can be:

1. Structural, if the task needs both sub-tasks to define objects that only if structurally correlated can solve the main task

2. Behavioral, similar to structural with the difference that state change is relevant to solving the task

3. Meta, which are more elaborate as they capture remaining attributes:

- temporal order if a sub-task needs to be completed before, after or simultaneously with a set of others. Temporal references capture features related to the moment of observation and duration, thus being relevant in capturing state change in the behavioral definition component

- task relevance and frequency (the number of times the task has to be solved in a time period)

- manage structural and behavioral correlations

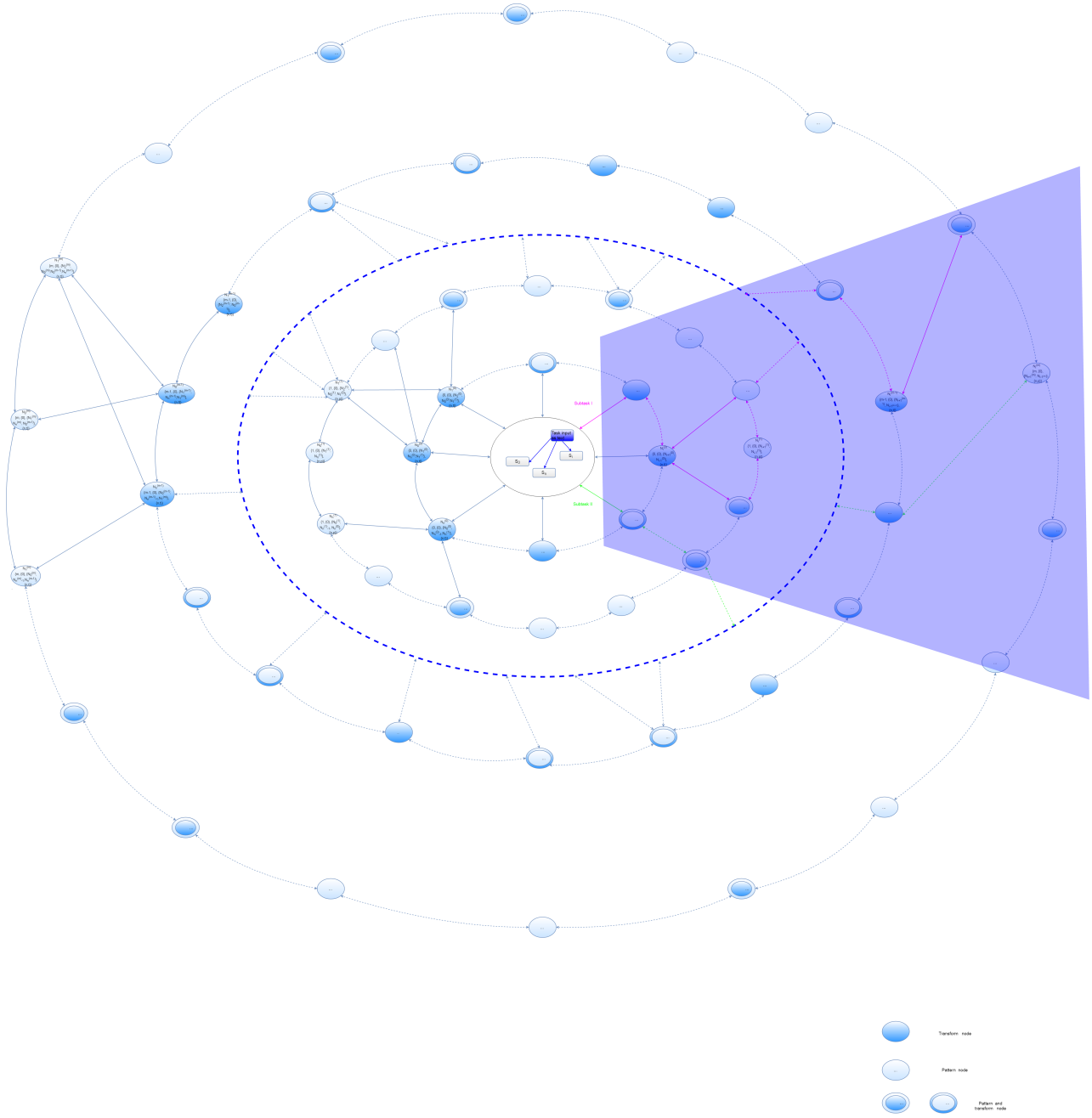


Figure 3.6: shows a task that is split into two sub-tasks whose definition paths are marked with purple and green. Correlations have been omitted for clarity; the objects have at least a meta correlation in that they appear in the same task. The task region marked with a dark blue background shows a topologic region of nodes related to the task (and consequently its sub-tasks). Note that the figure shows a simplified layout; sub-tasks may span distinct regions in the sphere. For simplicity, it is assumed that the sub-tasks are independent (have no correlations apart from the meta correlations of belonging to the same task) and atomic.

- definition path creation/look-up
- knowledge sphere specialization, specialization granularity (a knowledge sphere can have more than one specialization)
- likelihood of (sub-)task optimization which allows task solving using a balance between time and available resources

3.4 Task Optimization

Referrs to solving a set of sub-tasks an input task is split into with fewer resources as an initial solution (that is used as a reference point for subsequent task optimization). Resources are quantified by:

- time measured in clock cycles [1] needed to complete the task
- time measured as date-time that measures the time of task completion at higher level than c.cs.
- definition paths
- number of definition path look-ups
- amount of external resources used to complete the task

Definition Paths, Look-up Time

Time is two dimensional and measured in:

1. Clock cycles that are used in definition path/look-up/creation and are maintained by the meta component; selecting a node in a definition path takes one c.c. which can be a nexus node linking to the next sphere. More than one node can be picked during the same c.c.. C.c. time is isomorphic aross definition components (each component has an associated c.c. value. Note that values can differ across components in that components do not necessarily have to have the same creation time c.c. even though they define the same object)
2. Date time that is part of the meta component definition storing a time related value using an external temporal reference. The term external referrs to the fact that the temporal value doesn't play a role in pruning nodes as the c.c. value does

W.r.t. c.c. shorter definition paths take less time to look-up and can thus be used in optimization. Definition path shortening follows the steps:

[1] We use a shorthand notation c.c. for clock cycles for the remainder of this work

1. Initially, the definition is looked up conventionally; each sub-task is applied a transform function, the result is compared by the meta scheduler with pattern nodes located in the next layer and a set of best matching pattern nodes probabilistically picked. These steps are repeated for nodes in subsequent layers

2. The previous sub-task definition is merged with the remaining sub-task definitions periodically and checked against the input task's validity. The time period is defined by the number of nodes added to the definition path of each sub-task. An approach similar to TCP window scalling can be used; whenever the definition path length of a sub-task is a power of 2 the sub-task is synched with the remaining sub-task definitions and tested for validity in the main task. If the definition path length is less than a power of 2 the full length path is used

3. Path shortening is done by discarding nodes from the start and the end of the path and checking that the new path still leads to a valid task. Intuitively, more nodes can be removed from the start than the end as further nodes (from the input sensors) are more concrete and are likely to be more relevant to the task. Removing nodes from the start can be done using a divide-et-impera approach in that, the number of nodes that are removed are a power of 2 until the task becomes invalid. Afterwards, nodes are added (we assume a soft deletion approach that allows for node recovery and hard deletion for permanent removal) with an increasing step in 2s exponent. in the interval defined by the number of nodes from the current and previous deletion operations. Nodes from the of the path are removed incrementally (one at a time). Note that an algorithm removing nodes as a logarithmic function might be preferable

Due to the difficulty of tracking sub-task relevance, definition paths containing the most number of nodes are shortened with a higher probability than shorter paths.

Number of Definition Paths

Optimization consists in not using definition paths of sub-tasks that have proven to be invalid within the context of the main task up to a threshold. Such definitions are more likely to be pruned as the system states progress.

External Resources

Section 3.3.2 describes the process of acquiring external resources that are not directly related to the system's state (they first need to be converted or linked to the current system in the form of nodes or vacant knowledge spheres). Task optimization in this context referrs to reducing the time and amount of resources used in the acquisition process that is in itself defined as a background task that resources should always be allotted to.

As a task, it is comprised of a set of sub-tasks each having a definition for a set of objects and a corresponding functionality set (denoted by their definition components) relevant in resource acquisition. As these objects have structural and behavioral correlations to different objects optimization can be done by probabilistically picking the definitions of these objects and inserting them into the sub-task and consequently into the main (resource acquisition) task. Highly correlated objects have a greater likelyhood of replacing existing ones.

As an example, if one such task would be to build a mobile platform able to move from one point to another, the robot could change its constituting materials so that the movement time is minimized (e.g. replacing plastic wheels with rubber ones) or discard the platform for a more resource/time saving alternative.

Such optimization techniques do not account for creating new task solving solutions; creativity is discussed in Section 3.5

3.5 Creativity

The term creativity constitutes the ability of creating definition paths starting from a (sub-)set of available definitions created from previous observations and applying probabilistic multi-dimensional (structural, behavioral, extra and meta) changes. Creativity can be:

- task-oriented in that changes are applied to existing definitions based on correlations in order to create new definitions that are used in task solving
- pure that is not driven by task solving; definitions are created from pseudo-randomly picked nodes that are extended with nodes obtained by applying pseudo-random changes to patterns

3.5.1 Task-Oriented Creativity

Optimization techniques described in Section 3.4 consider shortening (sub-)task definition paths thus only contributing to a reduction of c.c. in definition creation/look-up steps (we do not take into account replacing definitions with correlated alternatives). This approach has limitations in that, only existing paths are used that may not necessarilly be optimal.

As an example, if our robot is a self-contained system capable of moving and a task requires that it moves from one point to another, the robot will solve the task as is (no definition path shortening is made on the first run). If the same task is input a number of times (assuming the definitions created/looked-up for the previous solutions haven't been pruned) a reduction of number of nodes is made w.r.t. starting and ending nodes as long as resulting paths are valid in the task's context. If there is a platform that moves between the same points (or at least two that are each closer to the start/end points than the distance between the robot must move in between to solve the task), has a higher speed than our robot and can be used by it to move between the task implied points faster the robot has an incentive of using the platform to improve the tasks's

solution. Upon observation, the robot creates a definition path set for the platform as well as correlations with previously observed objects. A behavioral correlation captures similarities between the robot's and platform's ability to move. Another correlation is made with the task that implies movement between two points that are the same or close to the points in between which the platform is moving.

Having the definition path and correlation set, the robot can use the platform to solve the task. The goal of optimization is to reduce the amount of resources used in task solving.

Creativity implies that our robot is able to build platforms instead of just using existing ones. More generally, the robot must be able to create new objects starting from definition components of previously observed objects that allow for task solving with lesser amounts of resources. This is the premise of creativity.

Such new objects do not have definition paths since they have not been observed in earlier runs and are created as part of a background task when:

- it is useful in task solving (including background tasks related to resource acquisition)
- it optimizes a task

Prior to building the object, the robot must ensure that the resources (internal/external and the amount of time) justify the object's creation. This is done as a function of task frequency (background resource acquisition tasks have a high priority and amount of resources allotted to them) and relevance.

Definition paths for new objects are created proabilistically based on correlations and definitions of existing paths of previously observed objects and their comprising object definition set.

For example, when creating the platform, the robot might make its wheels that were observed previously on a moving car whose behavioral definition component must also be fulfilled by the platform. The robot must also make the correlation that spinning wheels contribute to motion which is created by observing a number of moving cars or use a definition of a wheel that states its behavioral role.

Existing definitions are picked and probabilistically merged into a new object definition, as a function of correlations and likelihood of task relevance.

Given the structural and behavioral correlations between the definition path of a new object and existing objects, structural and behavioral changes are made so that the resulting object provides a valid solution for the task. Subsequently, its definition path is optimized as described in section 3.4.

For example, such changes are reflected in the engine type, transmission and overall structure of the platform.

Figure 3.7 shows how a copy of an object's (car's) component definition (for a wheel) is used as a structural and behavioral node, in the definition of a new object (platform). Correlation does not imply identity in that, the component used in a definition needs not be the same as used in another. For example,

the robot does not have to use the same wheels as observed on the car when building the platform; changes that allow for faster or more efficient movement of the platform are considered when building a new wheel to be used by the platform starting from the car wheel's definition.

3.5.2 Predictive Creativity

Does not aim to create definition paths for current or past tasks; definition creation starts from probabilistically choosing and copying nodes of paths or extending paths with new nodes, that are frequently used in task solving and applying transformations over their nodes so that the resulting definition path now matches an object likely to be observed during future task solving. The respective transform operations must be consistent with the needs of the system; the resulting solution for future tasks must be at least as efficient as previous ones.

Upon creation, the new definition is not used in task solving but correlated with the definitions it was derived from, stored for an amount of time that is a function of a mean over the task releveance and use of the originating paths. Once a new task is input, it is defined and checked against correlations with previous definitions of similar tasks as well as the newly created/extended paths; a set of paths is chosen probabilistically based on correlations and relevance.

If the newly created path offers a valid output for the task, it's lifespan in c.c. is extended as well as its relevance.

Predictive creativity is analogous to research while task-oriented creativity handles the external environment directly. Creativity is priority driven; background tasks related to the system's integrity and expansion (through resource acquisition) have the highest priority (and consequently the highest amount of alloted resources). Newly input tasks and predictive creativity processes have a lower priority. Drawing an analogy with reasoning, a person can conduct research with lower needs than those needed to ensure the person's well being.

3.5.3 Artifacts

Are definitions created from random or probabilistically chosen nodes based on loose criteria such as moment of definition creation that are not necessarily intended for task solving. They are similar to random thoughts or actions that are not strictly motivated by needs. Such paths are stored temporarily and may be used in task solving if sufficiently strong correlations exist between artefacts and paths used in task solving. Artefacts are probabilistically output to the extent to which they do not degrade task solving.

As an analogy, artefacts are similar to art.

They are assigned a lower priority that task-oriented and predictive creativity related processes.

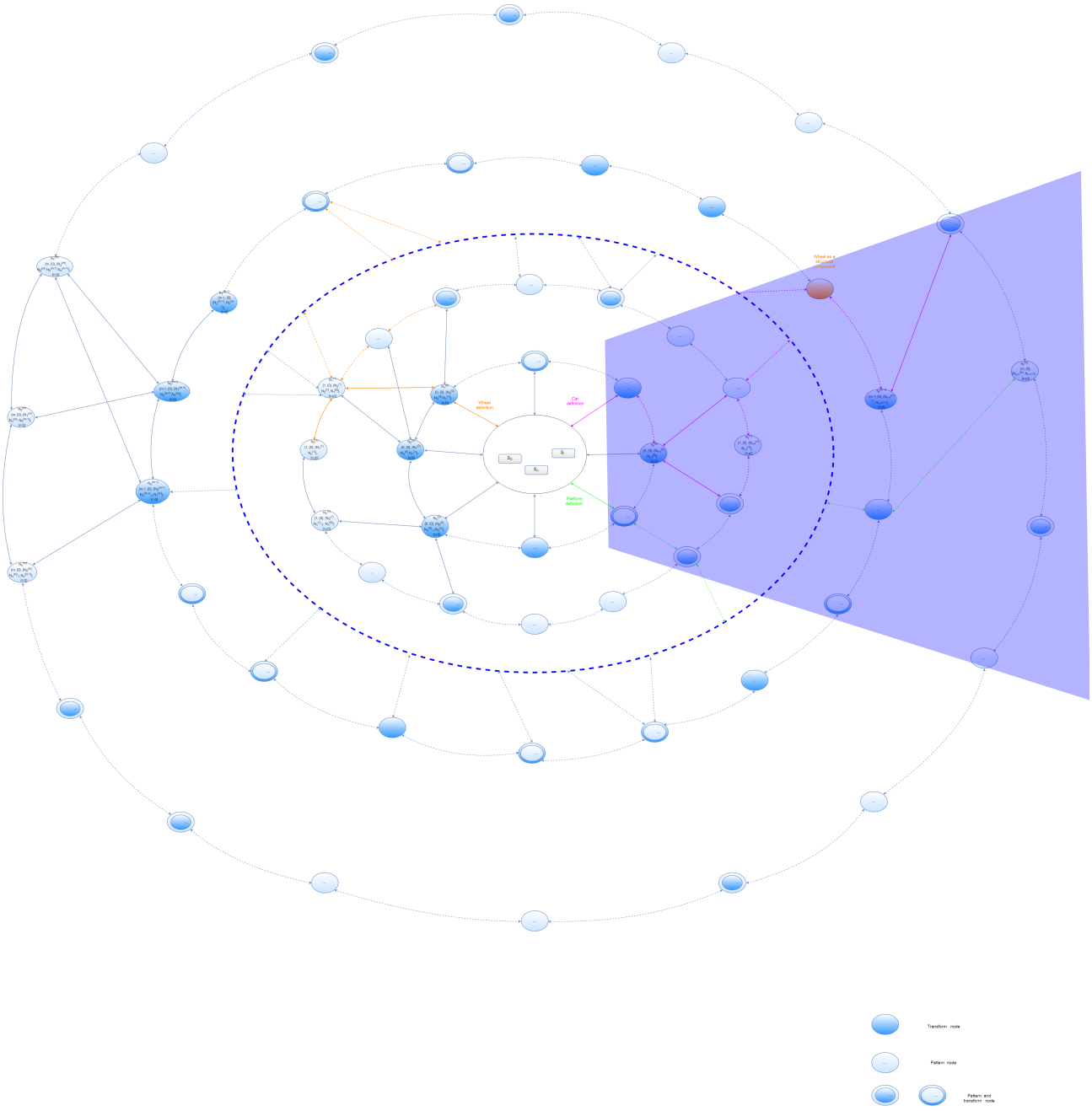


Figure 3.7: The sub-definition associated to a wheel is marked with orange and is a structural component of both the car and the platform. The car's definition path is marked with purple edges, the wheel's definition with orange and the platform's definition with green. The blue shaded area indicates that the car and the platform are correlated (both structurally (each uses wheels for moving) and behaviorally (each is capable of moving). Correlations exist between the wheel's definition and the two definitions but have been omitted for clarity.

3.6 Knowledge Sphere Structure

In this section we will be focusing on the structural elements and layout of the knowledge spheres such as nodes, circular lists (nested spheres), patterns as well as their attributes. As seen in earlier chapters, a knowledge sphere is comprised of circular lists of pattern nodes structured in spheres, which are nested with an ascending degree of concreteness starting from the center towards the surface of the sphere. Interleaved with the pattern spheres, are spheres comprised of nodes performing feature extraction and cost computation over the data received from the previous pattern sphere; the first input being the raw data perceived by sensors.

Each pattern node is a container holding:

- a list of patterns
- a weight for each pattern
- a node identifier
- coordinates within the knowledge sphere (node index and layer number)
- meta data attributes, weight and threshold values for each attribute
- life span
- use value
- use rate
- a set containing the definition identifiers of whose definition path the node is part of
- degree of abstraction <<maybe remove or emphasize this later>>
- links to nodes located on the same layer
- links to nodes from previous and next layer (sphere)
- links to transform, cost computation and feature extraction nodes

There are three types of nodes:

1. Pattern nodes that are only compared against a set of features and chosen in a definition path if a resulting cost value is less than a threshold
2. Nexus nodes that link consecutive nested spheres to each other. These nodes are the only node that allow for definition path creation across spheres
3. Transform nodes that have the role of extracting features from data received from nodes on the previous sphere and applying a series of transform operations over them.

The type of a node can change during runs of the algorithm. Initially, there is a set of high level abstraction nodes of all three types located in the first layers of the knowledge sphere that have indefinite life span (cannot be destroyed). This approach ensures that the sphere will always be able to create definitions and tasks without starting from scratch. In a worst case scenario, the algorithm has to create the abstraction layers on its own by observing objects and computing patterns from a common feature set between them. This process is time consuming as numerous observations must be made in order to create spheres capable of task solving.

The type of a node is essentially defined by the role and use of the node in definitions and, consequently, in task solving. As an example, a pattern node can become a nexus node if the definitions created through it and the nodes from previous and/or next spheres are optimal w.r.t. solving a task. Conversely, a nexus node may loose its status if there are no or invalid definitions that include it over a period of time.

Transform nodes handle feature extraction and cost computation of the difference between the extracted features and patterns. They are linked to the nexus nodes in between two successive spheres.

Depending on the data type it holds, each node is multidimensional; it contains patterns matching input sensor data from all available input sensors; for example the same node can have patterns matching auditive and visual data. Similarly, one node can be part of a set of distinct definitions belonging to different scopes (structural, behavioral, extra and meta).

<<Write on abstraction; an pattern can be created from two nodes that are similar to a degree. The definition paths will contain the new pattern and each node.>>

<<Move this paragraph to a more appropriate section>>

Figure <<draw figure>> shows a newly created abstract node which binds (merges) a set of definition paths. Node C is the new abstracted node created from applying a feature averaging function over the attributes of nodes P,Q,R. Future definition paths will use C when referring to the definition paths of objects A and B.

A critical structural element of the knowledge sphere are the links between nodes. Links are a means of establishing a correlation between two nodes and are multidimensional:

- pertinent to the definition scopes (structural, behavioral, extra and meta)
- definition paths (a path is represented by nodes chained together by links)
- and time constraints (each link has a life span depending on their use in solving tasks, number and frequency of use of definition paths containing nodes joined by them)

A link is created when there is a degree of similarity between two nodes defined by a cost less than a threshold value that can be part of a definition path used to solve a task. Conversely it is destroyed if the definition path is not looked up

in an interval of time or the definition paths created through it do not provide valid results w.r.t. a task.

Bibliography

[1] S Christian, L Wei, ad Pierre S Yangqing, J, R Scott, A Dragomir, E Dumitru, V. Vincent, and R Andrew. Going deeper with convolutions. 2014.

[2] Ross. G, D Jeff, and Jitendra M Trevor, D. Rich feature hierarchies for accurate object detection and semantic segmentation. 2014.

** Throughout our work we use subjective terms due to the fact that at the time of this writting our work is empirical; the choice of algorithm families and data structures are still subject of discussion.

*** We will define what the term "closer" refers to in <<add chapter number>>

**** The terms goal, tasks, question are synonyms unless otherwise implied by context.