OO jDREW Design Document

Indexing System for Mixed Positional/Slotted Terms

Marcel A. Ball <maball at gmail dot com>, Canada

Draft 0.7 - July 27, 2005

Introduction

In section 8, *Future Work,* of the report “OO jDREW: Design and Implementation of a Reasoning Engine for the Semantic Web”[1] the idea of developing an indexing system for OO jDREW based upon discrimination trees was briefly presented. This document serves to extend and further define the design of the envisioned indexing system. The proposed indexing system has not yet been implemented, therefor the exact level of performance change is not known, but the design is likely to lead to significant improvements in the usual cases, without significant overhead for the more unusual fringe cases.

The goal is to implement this design for an indexing system as part of the OO jDREW open source project[2] [3]. This should provide a significant performance increase to the OO jDREW engines (both bottom-up and top-down) when dealing with large knowledge bases.

---

[1] http://www.jdrew.org/oojdrew/docs/OOjDREW.pdf

[2] http://mail.jdrew.org/pipermail/jdrew-all/2005-June/000001.html

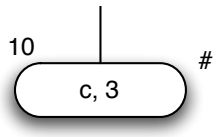
[3] http://www.jdrew.org/oojdrew

Notations and Conventions

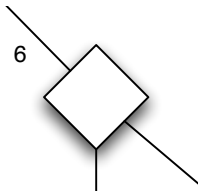
The following conventions are used in relation to the diagrams included in this document:

Regular nodes in the index tree are represented as ovals, as below:

- Numbers to the top-left of the node are for identification - it represents the order in which the nodes in that subtree were created.
- A symbol to the top-right of a node indicates that the clause indicated by that symbol would be stored at that node - for example in the following figure the clause indicated by # would be stored at that node.
- The two comma separated values in the node represent the role for the indexed term and the indexed symbol (role, symbol). If the role is ø that means there is no role (this is the case for atoms and for positional arguments).

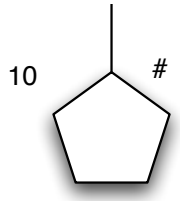


Positional rests (i.e. parameters with a role of IPREST) in the index tree are represented as diamonds, as below:



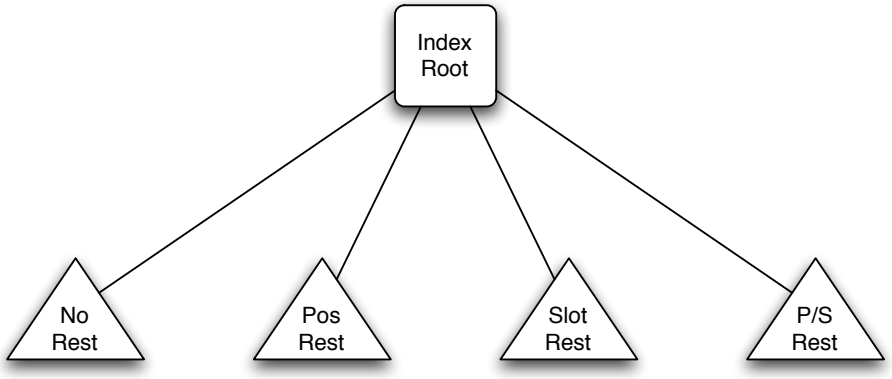
Slotted (non-positional) rests (i.e. parameters with a role of IREST) in the index tree are represented as pentagons, as below:

Because of the order that parameters are normalized in, a slotted rest parameter will always be a leaf node, and will always have at least one clause stored at it (other wise the node would not be needed and would not have been created).



General Structure

To make it possible to deal with the rest parameters that are specified in RuleML we need to know what type(s) of rest parameters a indexed atom has, before reaching that atom in the index. In order to make this possible the indexed terms are broken into four sections at the top, based upon the presence of rest parameters.



The proposed indexing structure assumes that the slots are sorted in some canonical order: positional parameters (indicated with a role of ø in this document), an optional positional rest parameter, slotted parameters, and finally the optional slotted rest parameter. OIDs should be ignored when indexing clauses and searching for clauses using this indexing method. Clauses are indexed by OID (possibly using a hash table) separately.

The root of the index will have four child nodes, one for atoms with no rest parameters, one for atoms with a positional rest parameter only, one for atoms with slotted rest parameters only, and one for atoms with both positional and slotted rest parameters.

Each node (except for the root) of the index will contain the following instance variables:

- children - A Hashtable that contains links to the child nodes of this node. The keys in the Hashtable are pairs of (role, symbol) and the value is the actual node object. In the case of a relation symbol or a positional argument the role value is ø (In the java code this value is SymbolTable.INOROLE).
- clauses - A Vector containing all of the clauses that are stored at that node in the index. Since clauses can be stored at internal nodes, and not just leaf nodes, this is present on all nodes.
- role - The role code for that node (This is ø for positional parameters and relations)
- symbol - The symbol code for that node

Inserting Clauses into Index

Inserting a clause into the index can be broken into three steps; selecting the correct sub-tree, processing the relation of the specified atom, and processing the parameters of the atom (only top level, the parameters for complex terms and plexes are not used in indexing).

The first step to insert a clause is to select the appropriate top-level subtree (i.e. the subtree for the appropriate rest parameters). The rest parameters that are present in an atom can be determined by the values of the **rest** and **prest** instance variables of the **Term** object that represents the atom. If **rest** has a value other than -1 then there is a slotted rest parameter in the atom; If **prest** has a value other than -1 then there is a positional rest parameter in the atom. By checking these two instance variable the top-level subtree that the clause should be stored in can be determined.

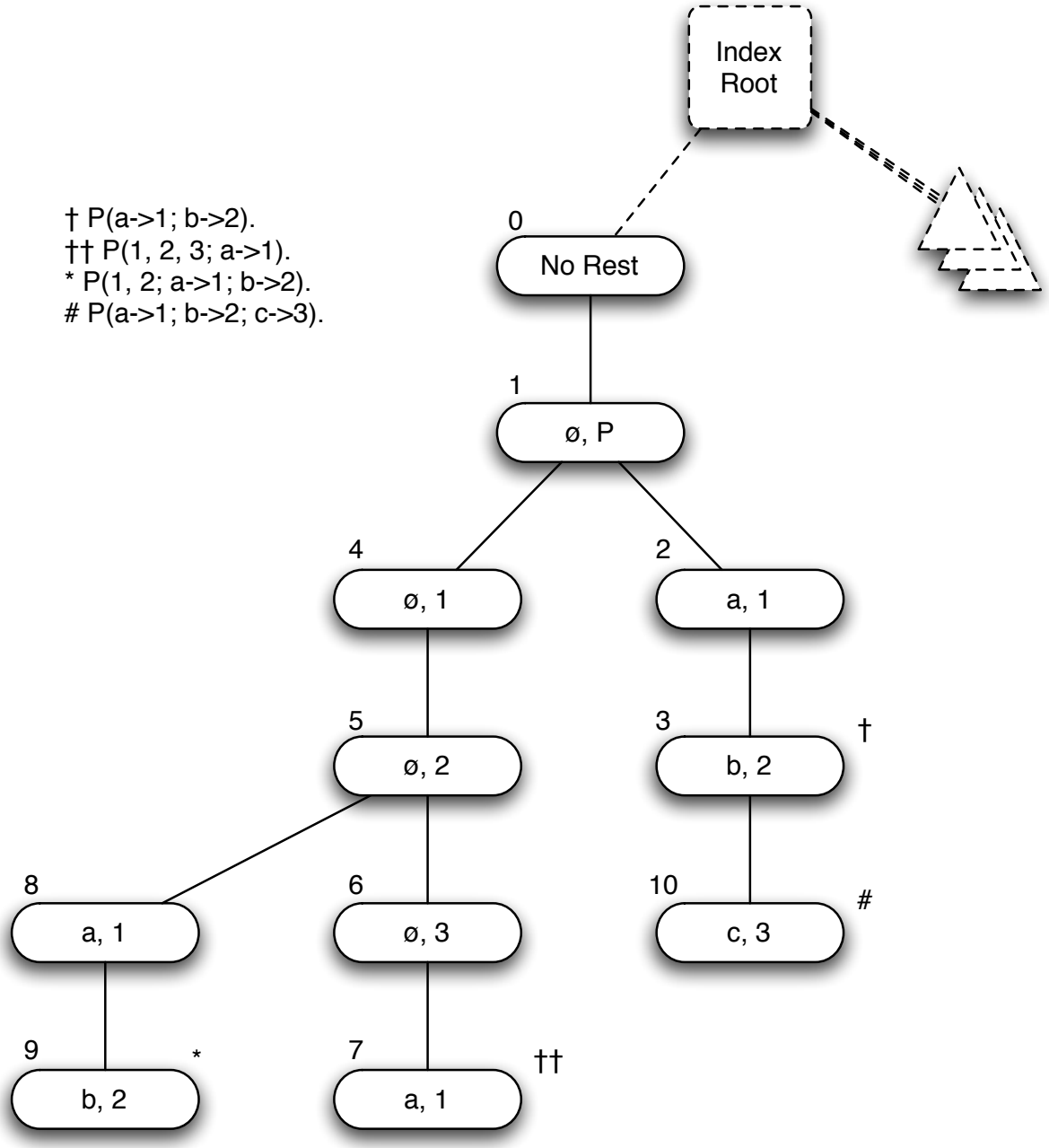
Once the top-level subtree has been selected (These are node 0 in the diagrams in the subsequent sections), a branch is selected based upon the predicate symbol of the clause to be indexed. If a branch for the predicate symbol (with a role of ø) exists then the clause is inserted under the existing branch based upon the parameters; otherwise a new branch for that predicate symbol (with a role of ø) is created, and the clause is inserted under the newly created node (based upon the parameters). In the diagrams in the subsequent sections the node with id 1 is the branch for atoms with a relation name of “P” (note that no other relation names are used in these examples).

The following steps are followed for each of the parameters for the atom (only the constructor name is used complex terms). Once all of the parameters have been used the clause is added to the clause list of the current node in the index.

- Select the child node which matches the current role and symbol (ø for the role if it is a positional argument and the constructor name for symbol if it is a complex term); If no child node exists with the appropriate role/symbol combination then create a new node and add it to the children of the current node.
- Add the clause to the selected child node, and move to the next parameter.

See the diagrams in subsequent sections to see examples of indexes with inserted clauses.

No Rest Parameters Subtree



Search Pattern

The case of no rest parameters creates the simplest search pattern when searching for clauses that are candidates for unification. We start processing by selecting the selecting the sub-tree that matches the relation of our search pattern (the atom that we are searching for unification matches for).

Once this is done we do the following steps for each of the parameters:

1. If the current parameter in the search pattern is a rest parameter (role of IREST or IPREST) move to the next parameter in the search pattern (or the end of the pattern if this is the last parameter) and a re-start at step 1

2. If the search is at the end of the search pattern:

    2.1. Add the clauses at the current node to the results list

    2.2. For each child node of the current node:

        2.2.1. If the search pattern has a positional rest parameter and the role of the child node is ø, visit the child node, remaining at the end of the search pattern

        2.2.2. If the search pattern has a slotted rest parameter and the role of the child node is greater than ø, visit the child node, remaining at the end of the search pattern

3. Otherwise, for each of the child nodes of the current:

    3.1. If the role of the child node is less than the role of the current parameter do the following:

        3.1.1. If the search pattern has a positional rest parameter and the child node's role is ø, visit the child node, staying at the current parameter in the search pattern

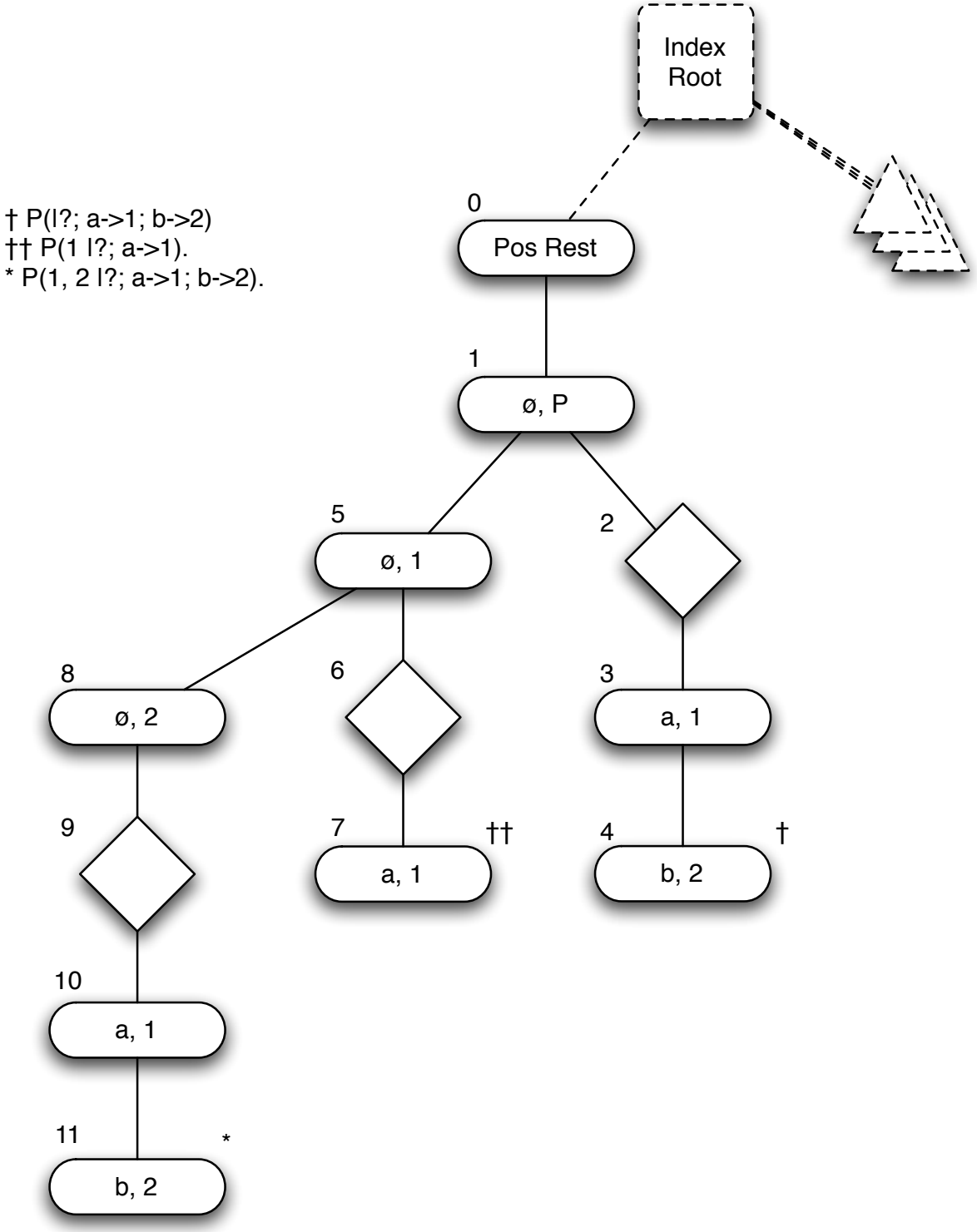
        3.1.2. If the search pattern has a slotted rest parameter and the child node's role is > ø visit the child node, staying at the current parameter in the search pattern

        3.1.3. If the situation does not meet either of these cases then ignore that child node.

    3.2. If the role of the child node is equal to the role of the current parameter of the search pattern, and the values are matching (i.e. same constant or one or both are variables) visit the child node, moving to the next parameter in the search pattern

    3.3. If the role of the child node is greater than the role of the current parameter of the search pattern then ignore that child node

Positional Rest Parameter Subtree



Search Pattern
We start processing by selecting the selecting the sub-tree that matches the relation of our search pattern (the atom that we are searching for unification matches for).

Once this is done we do the following steps for each of the parameters:

1. If the current parameter in the search pattern is a rest parameter (role of IREST or IPREST) move to the next parameter in the search pattern (or the end of the pattern if this is the last parameter) and a re-start at step 1

2. If the search is at the last (non-rest) parameter of the search pattern:

    2.1. Add the clauses at the current node to the results list

    2.2. For each child node of the current node:

        2.2.1. If the search pattern has a positional rest parameter and the role of the child node is ø, visit the child node, remaining at the end of the search pattern

        2.2.2. If the search pattern has a slotted rest parameter and the role of the child node is greater than ø, visit the child node, remaining at the end of the search pattern

        2.2.3. If the child node is a positional rest parameter (role of IPREST) visit the child node, remaining at the end of the search pattern.

3. Otherwise, for each of the child nodes of the current:

    3.1. If the role of the child node is less than the role of the current parameter do the following:

        3.1.1. If the search pattern has a positional rest parameter and the child node role is ø, visit the child node, staying at the current parameter in the search pattern

        3.1.2. If the search pattern has a slotted rest parameter and the child node role is > ø visit the child node, staying at the current parameter in the search pattern

        3.1.3. If the situation does not meet either of these cases then ignore that child node.

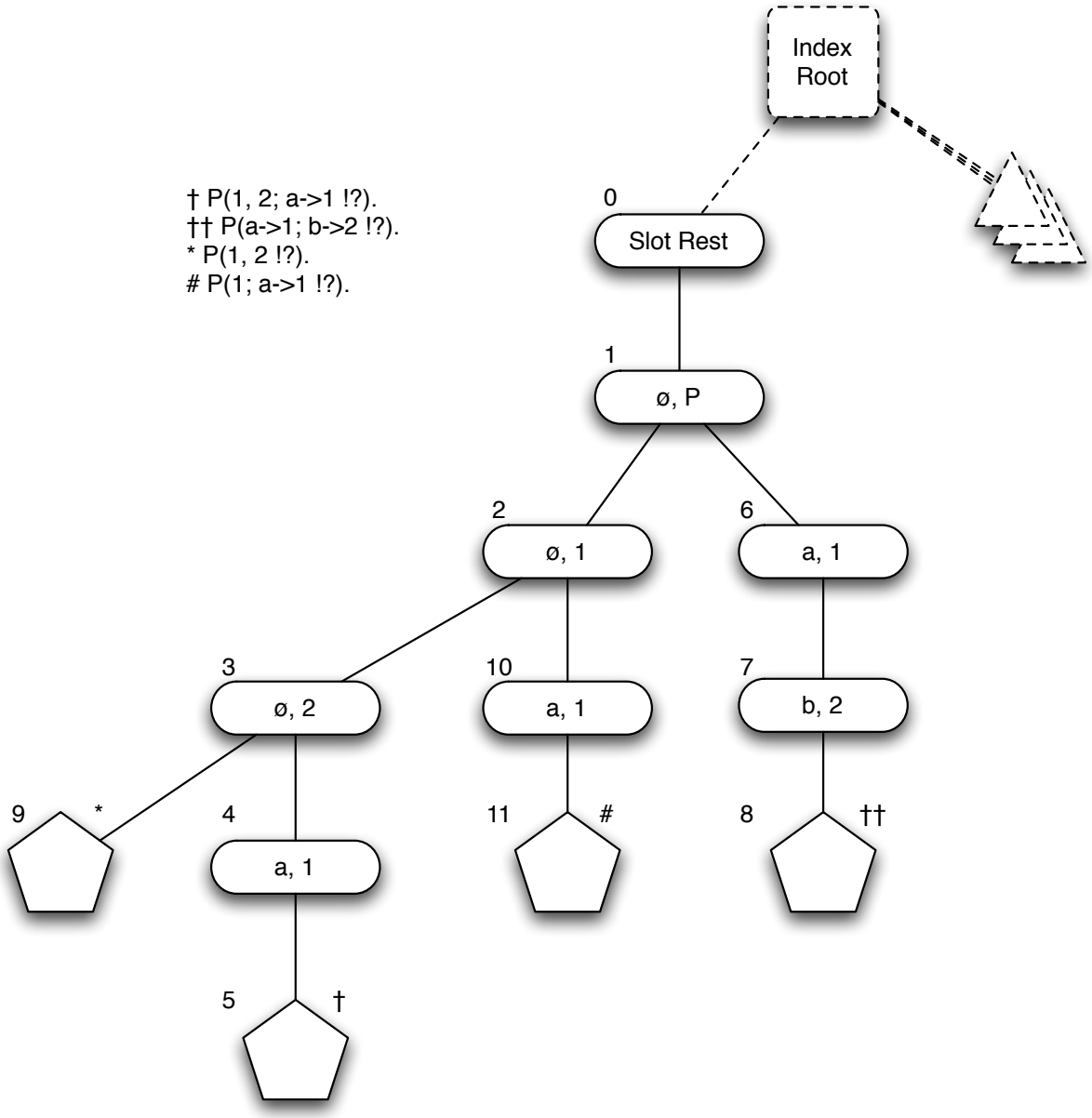
    3.2. If the role of the child node is equal to the role of the current parameter of the search pattern, and the values are matching (i.e. same constant or one or both are variables) visit the child node, moving to the next parameter in the search pattern

3.3. If the child node is a positional rest node (has a role of IPREST) and the current parameter of the search pattern has a role of ø or is the first non-positional parameter[4] visit the child node, moving to the first non-positional parameter (or to the end of the search pattern if there are no non-positional parameters) in the search pattern

3.4. If none of these other cases match and the role of the child node is greater than the role of the current parameter of the search pattern then ignore that child node

[4] This check may not need to be done - I do not think there will ever be a situation where the search would be at the second (or later) non-positional parameter where there would be a positional rest parameter encountered in the index.

Slotted Rest Parameter Subtree



Search Pattern
We start processing by selecting the selecting the sub-tree that matches the relation of our search pattern (the atom that we are searching for unification matches for).

Once this is done we do the following steps for each of the parameters:

1. If the current parameter in the search pattern is a rest parameter (role of IREST or IPREST) move to the next parameter in the search pattern (or the end of the pattern if this is the last parameter) and a re-start at step 1

2. If the search is at the last (non-rest) parameter of the search pattern:

    2.1. Add the clauses at the current node to the results list

    2.2. For each child node of the current node:

        2.2.1. If the search pattern has a positional rest parameter and the role of the child node is ø, visit the child node, remaining at the end of the search pattern

        2.2.2. If the search pattern has a slotted rest parameter and the role of the child node is greater than ø, visit the child node, remaining at the end of the search pattern

        2.2.3. If the child node is a slotted rest parameter (role of IREST) visit the child node remaining at the end of the search pattern

3. Otherwise, for each of the child nodes of the current:

    3.1. If the role of the child node is less than the role of the current parameter do the following:

        3.1.1. If the search pattern has a positional rest parameter and the child node role is ø, visit the child node, staying at the current parameter in the search pattern

        3.1.2. If the search pattern has a slotted rest parameter and the child node role is > ø visit the child node, staying at the current parameter in the search pattern

        3.1.3. If the situation does not meet either of these cases then ignore that child node.

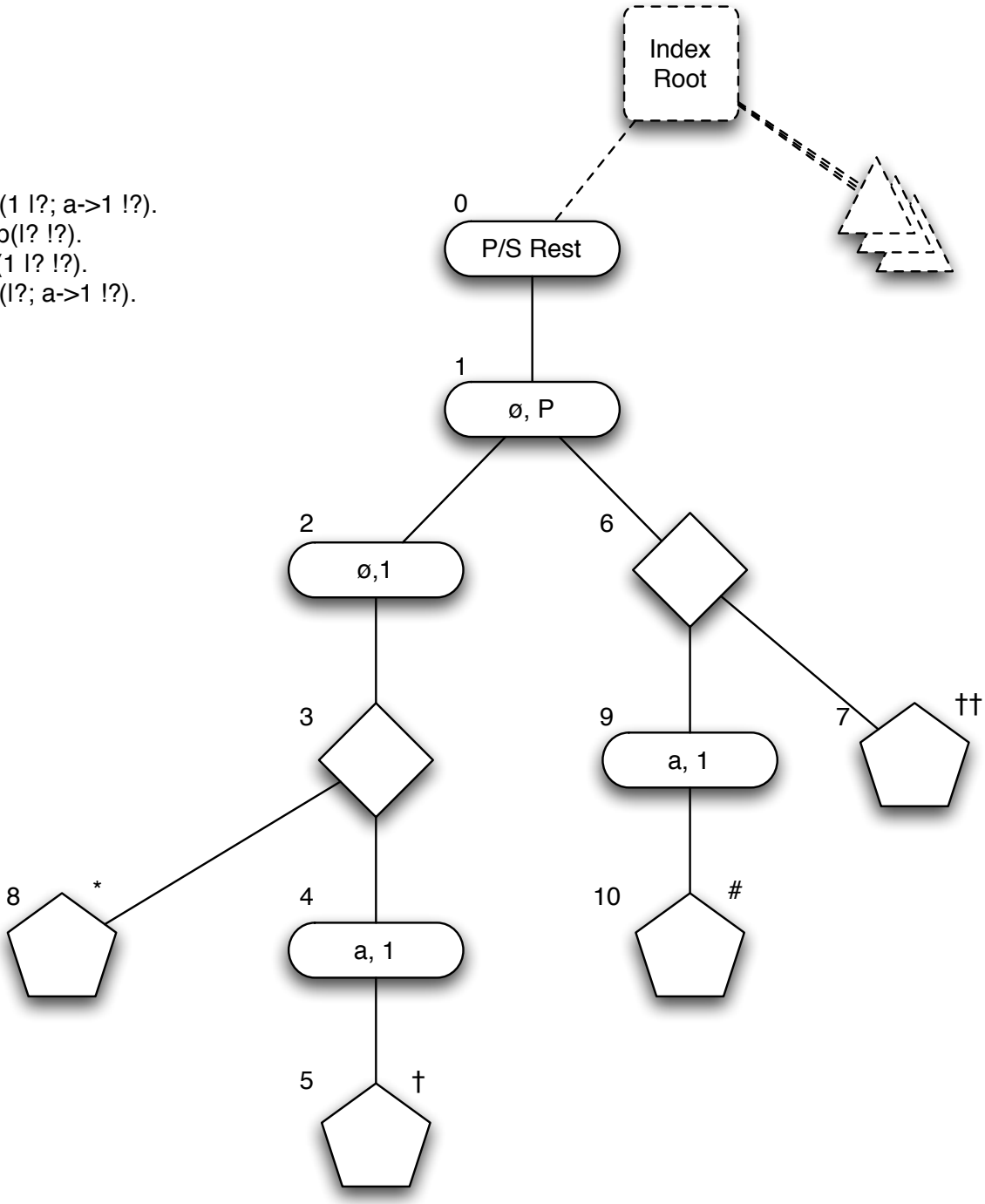
    3.2. If the role of the child node is equal to the role of the current parameter of the search pattern, and the values are matching (i.e. same constant or one or both are variables) visit the child node, moving to the next parameter in the search pattern

    3.3. If the role of the child node is greater than the role of the current parameter of the search pattern and the role of the current parameter is greater than IPREST then visit the child node, moving to the first parameter that has a role greater than the role of the child node, or to the end of the search pattern if no such parameter exists

3.4. If the child node is a slotted rest node (has a role of IREST) and the current parameter of the search pattern has a role greater-than IPREST (second lowest role value (i.e. the current parameter is a non-positional parameter (slot)) visit the child node, moving to the end of the search pattern

Positional and Slotted Rest Parameters Subtree

† P(1 l?; a->1 !?).
†† p(l? !?).
* P(1 l? !?).
# P(l?; a->1 !?).



Search Pattern
We start processing by selecting the selecting the sub-tree that matches the relation of our search pattern (the atom that we are searching for unification matches for).

Once this is done we do the following steps for each of the parameters:

1. If the current parameter in the search pattern is a rest parameter (role of IREST or IPREST) move to the next parameter in the search pattern (or the end of the pattern if this is the last parameter) and a re-start at step 1

2. If the search is at the last (non-rest) parameter of the search pattern:

    2.1. Add the clauses at the current node to the results list

    2.2. For each child node of the current node:

        2.2.1. If the search pattern has a positional rest parameter and the role of the child node is ø, visit the child node, remaining at the end of the search pattern

        2.2.2. If the search pattern has a slotted rest parameter and the role of the child node is greater than ø, visit the child node, remaining at the end of the search pattern

        2.2.3. If the child node is a slotted rest parameter (role of IREST) visit the child node remaining at the end of the search pattern

3. Otherwise, for each of the child nodes of the current:

    3.1. If the role of the child node is less than the role of the current parameter do the following:

        3.1.1. If the search pattern has a positional rest parameter and the child node role is ø, visit the child node, staying at the current parameter in the search pattern

        3.1.2. If the search pattern has a slotted rest parameter and the child node role is > ø visit the child node, staying at the current parameter in the search pattern

        3.1.3. If the situation does not meet either of these cases then ignore that child node.

    3.2. If the role of the child node is equal to the role of the current parameter of the search pattern, and the values are matching (i.e. same constant or one or both are variables) visit the child node, moving to the next parameter in the search pattern

    3.3. If the child node is a positional rest node (has a role of IPREST) and the current parameter of the search pattern has a role of ø or is the first non-

positional parameter[5] visit the child node, moving to the first non-positional parameter (or to the end of the search pattern if there are no non-positional parameters) in the search pattern

3.4. If the child node is a slotted rest node (has a role of IREST) and the current parameter of the search pattern has a role greater-than IPREST (second lowest role value (i.e. the current parameter is a non-positional parameter (slot)) visit the child node, moving to the end of the search pattern

3.5. If the role of the child node is greater than the role of the current parameter of the search pattern and the role of the current parameter is greater than IPREST then visit the child node, moving to the first parameter that has a role greater than the role of the child node, or to the end of the search pattern if no such parameter exists

---

[5] This check may not need to be done - I do not think there will ever be a situation where the search would be at the second (or later) non-positional parameter where there would be a positional rest parameter encountered in the index.