Production systems and shift-reduce parsing assignment

A second purpose is to expose you to programming in a production system language on a problem that takes advantage of its special capabilities. Production systems are a class of rule-based formalisms that, like logic programs, rely centrally on relational pattern matching, but that differ by mimicking the cyclic operation of human cognition over time. You have already used CLIPS for the first exercise. The production system framework you will use for this assignment, PROPS, has a different syntax, similar to that in Prolog, the language in which it is implemented.

This provides it with more powerful pattern-matching abilities that will prove useful for the task. As explained in the lecture on production systems, a program in this framework utilizes two stores: a working memory that changes dynamically and a production rule memory that is (typically) static. You will find an introduction to the language's syntax and operation at http://www. CSS. Auckland. AC. NZ/courses/socioeconomics/assignments/ex./props . HTML Briefly, each working memory element should take the form of a Prolog fact, and thus should have a predicate and one or more arguments.

Each production rule should have a left-hand side comprising a set of conditions and a right-hand side that specifies one or more actions, the most important being actions that add elements to working memory or that remove them. You can download the PROPS interpreter at rename it to be props. Pl, and load the file into Prolog to run your production system program. You may not modify the PROPS interpreter for this assignment; you may only create production rules that it runs.

A third aim of the assignment is to give you experience with parsing natural language. Transforming sentences into parse trees is a basic problem in artificial intelligence that often serves as the first step in linguistic processing. This exercise focuses on a wait-and-see' or shift-reduce' approach to parsing. Such a parser operates on two main structures: a buffer that initially contains the input sentence and a stack that holds partial parse trees as they are constructed.

This approach applies three types of operators: creating a new (initially empty) phrase structure in the stack, dropping a phrase structure from the stack into the buffer, and attaching a phrase structure in the buffer to one on the stack. These occur in different guises that depend on the types of phrase structures, parts of speech, and conditions involved. The mechanism uses these operators to construct a parse tree incrementally, applying one of them on each recognize-act cycle, until it has processed the sentence completely. Part A This exercise has two components.

For Part A, you should translate the paraphrased production rules that are available at http://www. CSS. Auckland. AC. NZ/courses/socioeconomics/assignments/ex./shift onto the syntax outlined earlier. You should also create an initial working memory with information about the parts of speech for various words, an empty stack, and an empty buffer. Once you have translated the paraphrased rules, you should show that your program processes the following sentences (in italics) correctly in that it generates the parse trees indicated for each one: [john, can, dance]. [the, cat, ran]. The, big, black, cat, chased, a, small, mouse]. [some, hungry, mice, plan, a, heist]. [up, [verb, Demonstrate that the program works correctly on these sentences by typing hem into the system when it requests a sentence, tracing the rule firings, and displaying the final parse tree. The final stack should contain only the parse tree associated with each of the test sentences above. Part B For Part B of the exercise, you should extend the grammar to handle sentences that include prepositional phrases. The system should be able to attach such phrases either to noun phrases or to verb phrases.

We recommend that you add this functionality by introducing eight rules with the suggestive names create up for NP, attach_P to_up, NP in UP for D, NP in UP for N, attach UP to up, UP to NP, and although you are welcome to organize things differently. Once you have added the new rules, you should show that the extended program processes the following sentences (in italics) correctly in that it generates the parse trees indicated for each one: [a, cat, pounces, on, a, ball]. [the, black, cat, was, on, the, big, table]. [a, black, cat, chased, the, ball, under, the, table]. The, eager, frog, under, the, table, may, meditate, on, the, problem, throughout, the, night, despite, the, poisonous, fumes]. [up,[sax, may],[verb, meditate], NP,[diet, the],[noun, night], [up,[prep, despite associated with each of the test sentences above. Sample Rules and Traces To make the assignment tractable, we have provided PROPS versions of the final four (miscellaneous) paraphrased rules. These should clarify how you can encode the buffer and stack as working memory elements and how PROPS rules can manipulate them.

We have also provided traces of a system that runs correctly on the eight test sentences. This shows the name of the rule that fires on each cycle, along with the buffer and stack that result from its application. Comparing our program's behavior to these traces should help during the debugging process. Marking Guidelines for Assignment (10 total marks) We will grade the answers that your provide for this assignment in terms of four factors: Ability to parse the sentences in Part A correctly (4 marks).

You will receive one mark for each sentence on which your code produces the desired parse tree. If a parse tree contains any errors, you will receive nothing for that sentence. Readable documentation of your production rules for Part A (1 mark). The marker should be able to understand the function intended for each rule. The easiest way to achieve this is to include an English paraphrase as a comment in your file just before each rule. You are welcome to use the paraphrases provided for this purpose.

Ability to parse the sentences in Part B correctly (4 marks). You will receive one mark for each sentence on which your code produces the desired parse tree. If a parse tree contains any errors, you will receive nothing for that sentence. Readable documentation of the new production rules introduced for Part B (1 mark). The marker should be able to understand the function intended for each new rule. The easiest way to achieve his is to include an English paraphrase as a comment in your file just before each rule.

You will need to generate your own paraphrases for this purpose. Make sure your program runs as described above. Submit your final program in a single file that clearly marks the initial working memory, the production rules for Part A, and the additional rules for Part B. You should also submit a separate file that contains a trace of the program's behavior on the eight test sentences in the order given above. Deposit your programs and traces in the Computer Science Assignment Dropper before the assignment deadline.