Assignment 1
Substructural Inference

15-836: Substructural Logics
Frank Pfenning

Due Thursday, September 7, 2023
60 points

In this assignment we explore structural, linear, and ordered inference. You will write some inference rules and can test them on your own machines using the provided code. For testing purposes, you should use Standard ML of New Jersey (SML/NJ). Eventually, you should hand them in via Gradescope which also has its own tests and will assign a score. In addition we may review your code. You may hand in as many times as you like.

The file you should hand in must be called `hw1.sml`. A template is in `starter/hw1.sml`. The autograder will assign 44 points; the remaining 16 points will be at my discretion. The tasks to describe your code is optional, but maybe useful in case you don't have the full autograder score and I have to read your rules.

Problem 4 is entirely optional and is not included in the autograder. If you decide to hand this in, it should be as a separate file on Gradescope with some brief instructions on how to run it.

The handout has the following structure:

- `src/`, the implementation of structural, linear, and ordered inference
- `starter/`, the starter code for this assignment, including a template for `hw1.sml`
- `hw1.pdf`, this assignment spec
- `*.{tex,sty}`, LaTeX source, macros, and style files

1 Deductive Parsing with Ordered Inference

Any context-free grammar can be put into normal form where only the start symbol S may have a production with an empty right-hand side, $S \to \epsilon$.

We represent an input word in the form of an *ordered state*

$$\$\ a_1\ \ldots\ a_n\ \$$$

where all of the a_i are propositions representing the letters of the alphabet for a particular grammar and $\$$ are propositions functioning as end markers.

The task is to translate a given collection of grammar productions into a collection of inference rules such that

$$\begin{array}{c} \$\ a_1\ \ldots\ a_n\ \$ \\ \vdots \\ S \end{array}$$

can be derived by ordered inference if and only if $S \longrightarrow a_1 \ \ldots \ a_n$ according to the grammar. (This should not be confused with notation of $\Omega \longrightarrow \Omega'$ we introduced in lecture where the arrow points exactly the other way, unfortunately.)

Because of the nondeterminism that may be inherent in grammar productions, the corresponding inference rules should generate a finite set of all reachable states. Success (= the initial word is generated by the grammar) should mean that one of these states is a single S.

Task 1 *(optional) Describe, in a comment, how grammar productions are translated into inference rules in general.*

Task 2 *A grammar generating the language $a^n \ b^n$ for $n \geq 0$*

$$\begin{array}{lcl} S & \to & \epsilon \\ S & \to & T \\ T & \to & a \ T \ b \\ T & \to & a \ b \end{array}$$

Task 3 *A grammar generating the language of nonempty even-length palindromes $w \ w^R$ over the alphabet $\{a, b\}$.*

$$\begin{array}{lcl} S & \to & a \ S \ a \\ S & \to & a \ a \\ S & \to & b \ S \ b \\ S & \to & b \ b \end{array}$$

Task 4 *A grammar generating the language of matching parentheses*

$$\begin{array}{lcl} S & \to & \epsilon \\ S & \to & T \\ T & \to & T \ T \\ T & \to & '(' \ T \ ')' \\ T & \to & '(' \ ')' \end{array}$$

2 Deductive Parsing with Structural Inference

We have seen deductive parsing from the perspective of ordered inference in Problem 1. In this problem we implement parsing using *structural inference*. The technique we use is at the core of a general method for mapping ordered inference to structural inference.

We use a unary representation of the natural numbers as $0, \mathsf{s}(0), \mathsf{s}(\mathsf{s}(0)), \ldots$. A word

$$a_1 \ \ldots \ a_n$$

is represented as the propositions

$$\mathsf{ext}(0, a_1, 1), \mathsf{ext}(1, a_2, 2), \ldots, \mathsf{ext}(n-1, a_n, n), \mathsf{num}(n)$$

Your task is to translate a grammar into a collection of inference rules such that starting from the state above you can deduce

$$\mathsf{ext}(0, S, n)$$

where S is the starting symbol if and only if $S \longrightarrow a_1 \ \ldots \ a_n$. As a corner case, the empty string is represented just by $\mathsf{num}(0)$ and if it is generated by the grammar the proposition $\mathsf{ext}(0, S, 0)$ should be in the saturated state.

You may make the same assumptions about grammars as in Problem 1. For any given grammar, your set of rules should saturate for any valid input as described above.

Task 5 *(optional) Describe in a comment how the grammar productions are translated into inference rules in general.*

Task 6 *The grammar in Task 2.*

Task 7 *The grammar in Task 3.*

Task 8 *The grammar in Task 4.*

3 Two-Counter Minsky Machines

A two-counter Minksy Machine consists of two register r1 and r2, a program counter p, and a finite program consisting of commands $i : \mathsf{INC}(r, j)$ and $i : \mathsf{JZDEC}(r, j_z, j_{nz})$. The registers as well as the program counter can hold arbitrary natural numbers encoded in unary form: 0, $\mathsf{s}(0)$, $\mathsf{s}(\mathsf{s}(0))$, etc.

If the program counter is i and $i : \mathsf{INC}(r, j)$ then the machine increments register r and jumps to instruction j.

If the program counter is i and $i : \mathsf{JZDEC}(r, j_z, j_{nz})$ then the machine first tests register r. If its current value is 0, it jumps to address j_z. If its current value is greater than 0, the machine will decrement register r and jump to address j_{nz}.

We assume all addresses i occurring in a program are "in bounds" in the sense that there is a unique instruction $i : I$ except for $i = 0$. We start execution at instruction 1 and halt when we jump to 0 (because there is no corresponding instruction).

We encode the current execution state of a Minsky machine in three linear propositions $\mathsf{reg}(\mathsf{r1}, m)$, $\mathsf{reg}(\mathsf{r2}, n)$, and $\mathsf{pc}(i)$. Your task is to encode the program as a set of linear inference rules such that the program transitions from one state to the next if and only if the representation of the state changes correspondingly. Because a Minsky machine is deterministic, your encoding should be deterministic as well. And because the halting problem for two-counter Minsky machines is undecidable, your encoding cannot guarantee quiescence.

Task 9 *(optional) Describe in a comment how a program is translated into a set of inference rules.*

Task 10 *Give an encoding of the program*

$$\begin{array}{lcl} 0 & : & \\ 1 & : & \mathsf{INC}(\mathsf{r1}, 2) \\ 2 & : & \mathsf{INC}(\mathsf{r2}, 0) \end{array}$$

Task 11 *Give an encoding of the program*

$$\begin{array}{lcl} 0 & : & \\ 1 & : & \mathsf{JZDEC}(\mathsf{r2}, 0, 2) \\ 2 & : & \mathsf{INC}(\mathsf{r1}, 1) \end{array}$$

Task 12 *Give an encoding of the program*

$$\begin{array}{lcl} 0 & : & \\ 1 & : & \mathsf{JZDEC}(\mathsf{r2}, 2, 1) \\ 2 & : & \mathsf{JZDEC}(\mathsf{r1}, 0, 3) \\ 3 & : & \mathsf{JZDEC}(\mathsf{r1}, 0, 4) \\ 4 & : & \mathsf{INC}(\mathsf{r2}, 2) \end{array}$$

4 Post Correspondence Problem (Optional)

In this problem you are asked to encode instances of Post Correspondence Problem (PCP) as rules for ordered inference. We like the formulation of the problem as being given a set of dominos labeled with two words (top and bottom) over a finite alphabet. A solution to and instance of PCP is a list of these dominos such that the concatenation of the top words and the concatenation of the bottom words are the same string. See, for example, the Wikipedia article on Post Correspondence Problem for additional explanation and examples.

Your task is to translate a set of dominos into ordered inference rules such that

$$\begin{array}{c} \mathsf{init} \\ \vdots \\ \mathsf{success} \end{array}$$

if and only if the problem has a solution. We are not explicitly asking this, but one should be able to read off the list of dominos from the proof of success from init.

The translation algorithm for an arbitrary set of dominos should be described in a comment in your solution file.

Because PCP is undecidable, you will not be able to give an encoding that always terminates with a finite number of reachable states. Instead we will give the inference engine a reasonable bound so it can fail when no solution

Task 13 *Describe in a comment how a set of dominos is represented as a collection of inference rule.*

Task 14 *Give an encoding of the dominos*

$$\left[\begin{array}{c} b\,c \\ c\,a \end{array}\right], \left[\begin{array}{c} a \\ a\,b \end{array}\right], \left[\begin{array}{c} c\,a \\ a \end{array}\right], \left[\begin{array}{c} a\,b\,c \\ c \end{array}\right]$$

Task 15 *Give an encoding of the dominos*

$$\left[\begin{array}{c} a\,b\,c \\ a\,b \end{array}\right], \left[\begin{array}{c} c\,a \\ a \end{array}\right], \left[\begin{array}{c} a\,c\,c \\ b\,a \end{array}\right]$$

Task 16 *Give an encoding of the dominos*

$$\left[\begin{array}{c} a \\ b\,a\,a \end{array}\right], \left[\begin{array}{c} a\,b \\ a\,a \end{array}\right], \left[\begin{array}{c} b\,b\,a \\ b\,b \end{array}\right]$$