Projeto PCS2427
Autômato de Pilha Estruturado

Professor: João José Neto

Henrique Rodrigues – 7628927

Data: 29/06/2015

Autômatos de pilha estruturada

Autômatos de pilha estruturados são definidos como uma ênupla:

M = (Q, A, Σ, Γ, P, Z0, q0, F), na qual:

- Q é o conjunto de estados de M
- A é a coleção de submáquinas (similares aos autômatos finitos) que implementam M
- Cada submáquina Ai, i = 1, …, n, é da forma Ai = (Qi, Σ, Ri, q0i, Fi) , sendo Qi um subconjunto de Q, Ri um subconjunto de R, q0i o estado inicial de Ai e Fi, subconjunto de Qi, o conjunto de estados de retorno de Ai.
- Σ é o alfabeto de entrada contendo os símbolos que são consumidos pelas submáquinas Ai.
- Γ é o alfabeto de pilha contendo os estados finais Fi das diversas submáquinas Ai.
- P é o conjunto de regras de transição de autômatos.
- Z0 é o indicador de pilha vazia e pertence a Γ.
- q0 é o estado inicial de M, e pertence a Q
- F é subconjunto de Q e representa os estados finais de M

Portanto o autômato de pilha estruturada é um conjunto de submáquinas, cada qual responsável por reconhecer uma determinada classe de subcadeias que poderá existir dentro da cadeia de entrada a ser analisada.

Ao ser executada uma chamada de submáquina, o estado para onde deverá ocorrer o retorno é salvo na pilha, e o próximo estado a ser atingido é o estado inicial da submáquina chamada. Ao ocorrer retorno de submáquina, desempilha-se o estado de retorno que foi empilhado na ocasião de chamada. Desse modo, transições internas à submáquinas não causam alteração na pilha, somente transições entre máquinas. Da mesma forma, apenas transições internas das submáquinas consomem símbolos, e transições de chamada e retorno (finalização) de submáquina são executadas em vazio.

Portanto caracteriza-se a situação do autômato como sendo o conteúdo da pilha, o estado da submáquina corrente e a parte da cadeia de entrada que ainda não foi analisada naquele instante de processamento.

A aceitação de uma cadeia por um autômato de pilha estruturado ocorre quando uma submáquina (no caso, a submáquina mais superior, de início do reconhecimento) retorna uma cadeia vazia e a pilha encontra-se vazia.

Reforça-se o fato de transições entre submáquinas ocorrem em vazio, seja pelo comando de STACK ou pelo comando de RETURN. Porém isto não gera um não-determinismo pois permite-se apenas uma única transição para estados que possuem estes tipos de transição. Não podem ocorrer, dentro de um mesmo estado, transições do tipo 1 (transição dentro da mesma submáquina) e dos tipos 2 e 3

(transição entre submáquinas).

Produções em uma submáquina são definidos da seguinte forma:

Pi,m → produção de número m na submáquina i

$$Pi,m: \lambda s \rho \rightarrow \lambda' s' \rho'$$

$\lambda \in \Gamma^*$: situação da pilha antes da transição

$s \in Q$: estado antes da transição

$\rho \in \Sigma^*$: cadeia de entrada antes da transição

$\lambda' \in \Gamma^*$: situação da pilha após transição

$s' \in Q$: estado após transição

$\rho' \in \Sigma^*$: cadeia de entrada após transição

Regras a serem lidas para definirem a máquina de estados finita

Para definição de uma máquina de estados, as seguintes características são lidas do arquivo txt de entrada, sendo cada elemento em uma linha do arquivo:

- Estados
- Submáquinas
- Símbolos de entrada
- Símbolos da pilha
- Estado inicial
- Estados finais
- Produções

Elemento	**Formato**
Conjunto de estados Contém todos os conjuntos da máquina	Sequência de conjunto de números, separados por vírgula, indicando a submáquina e o estado dentro da submáquina. Ex.: 1,1 1,2 2,1 2,2 → Significa que o autômato possui os estados 1 e 2 na submáquina 1, e os estados 1 e 2 na submáquina 2.
Conjunto de submáquinas Indica quais submáquinas o sistema contém	Indicado por sequência de números separados por espaço, cada número sendo a identificação da submáquina Ex.: 1 2 3 → Sistema contém as submáquinas 1, 2 e 3.
Conjunto de símbolos de entrada Grupo de símbolos a serem lidos pelo autômato	Sequência de caracteres separados por espaço. Ex.: a b x log
Conjunto de símbolos de pilha	Sequência de conjunto de números, separados por vírgula, indicando a submáquina e o estado dentro da submáquina.

Contém os estados para os quais há retorno após passagem por uma submáquina chamada pela submáquina de retorno.	Além do símbolo que indica finalização de reconhecimento (aceitação). Ex.: 1,4 2,3 Z0 → Significa que é possível voltar ao estado 4 da submáquina 1, e ao estado 3 da submáquina 2, após passagem por outra submáquina chamada.
Estado inicial Por onde se inicia o processo de análise da cadeia de entrada	Um par de números, separados por vírgula, indicando a submáquina inicial e o estado inicia dela.
Conjunto de estados finais Estados nos quais, se a cadeia de entrada já foi completamente lida, representa a aceitação da cadeia pela linguagem definida pela máquina de estados	Sequência de conjunto de números, separados por vírgula, indicando a submáquina e o estado dentro da submáquina. Ex.: 1,2 2,1 → Significa que o autômato possui como estado de aceitação os estados 1 na submáquina 2, e 2 na submáquina 1.
Produções Regras de transição de estados que fornecem, antes e depois da aplicação da produção, o conteúdo da pilha, o estado e a cadeia de entrada.	Cada produção está em uma linha do arquivo de entrada, e está escrito na forma de notação simplificada. Então há quatro tipos de produção: **Tipo 1)** submáquina_I,estado_A símboloDeEntrada submáquina_I,estado_B OU **Tipo 2)** submáquina,estado RETURN OU **Tipo 3)** submáquina_I,estado_Asubmáquina_J STACK submáquina_I,estado_B Ex.: • 1,2 a 1,4 → vai para 1,4 a partir de 1,2 ao receber “a” • 2,3 RETORNA → retorna para estado no topo da pilha quando em 2,3 • 1,5 2 EMPILHA 1,6 → ao chegar no estado 1,5 ocorre a transição em vazio para o estado inicial da submáquina 2, e o estado 1,6 é adicionado na pilha

Objetos representados por classes no sistema

- (Automaton) Autômato finito
- (Submachine) Submáquinas do autômato finito

- (State) Estados do autômato finito
- (Production) Produções
- (Symbol) Cadeia de entrada
- (Stack) Pilha auxiliar do autômato

Atributos de cada objeto

Autômato finito: Automaton

- (lista de State) Estados: conjunto de estados do sistema (eles contém também o número da submáquina)
- (lista de Symbol) Símbolos: alfabeto que compõe a lingagem descrita pelo autômato de pilha estruturado.
- (Stack) Pilha: elemento de estrutura de dados que armazena os estados de retorno ao chamar submáquinas.
- (Boolean) Aceito: setado em true quando estiver em um estado de aceitação, e em false quando não (valor default)
- (Boolean) Erro: setado em false por default, e setado em true quando não há transição possível a partir de um estado; a simulação é finalizada indicando que a cadeia não pertence à linguagem definida pelo autômato finito.
- (Input) Cadeia de entrada: conjunto de caracteres a ser analisado pelo autômato finito, e símbolo atual lido pelo cabeçote de leitura da máquina
- (State) Estado atual: onde se encontra a máquina de estados, incluindo submáquina e estado.
- (Boolean) Mostrar rastreamento: indica se é desejável ver o rastro da cadeia pelo autômato ou não; caso não deseja-se, apenas o resultado de aceitação é mostrado

Submáquinas: Submachine

- (State) Estado inicial: por onde se inicia o reconhecimento nesta submáquina
- (Integer) Submachine ID: identificador da submáquina
- (Submachine) Next submachine: próxima submáquina da lista ligada

Estado da Submáquina: State

Para representar cada estado, cria-se esta classe contendo os seguintes atributos de estado. Para representar todos os estados, adiciona-se esta classe como célula de uma lista ligada.

- (Integer) State ID: indica o número do estado
- (Integer) Submachine ID: indica o número da submáquina que contém este estado
- (State) Next State: ponteiro para o estado seguinte da lista

Produção: Production

Para representar cada transição, cria-se esta classe contendo os seguintes atributos de transição. Para representar todos os estados, adiciona-se esta classe como célula de uma lista ligada.

- (State) Estado atual: presente nos quatro tipos definidos
- (Symbol) Símbolo(s) de entrada: presente na produção de tipo 1
- (State) Estado seguinte: presente na produção de tipos 1, indicando o estado para o qual se encaminha após recebimento do símbolo, e de tipo 3 indicando o estado que deverá ser empilhado para ser retornado após execução da submáquina sendo chamada.
- (String) Comando “EMPILHA” ou “RETORNA”: presente na produção de tipos 2, ao chamar uma submáquina e tipo 3 ao retornar de uma submáquina.
- (Submachine) Submáquina seguinte: encaminha-se para o estado inicial desta submáquina ao receber comando EMPILHA
- (Production) Produção seguinte: link para produção seguinte da lista

Cadeia de entrada: Symbol

- (String) Cadeia de entrada: string com a sequência de símbolos de entrada que ao coincidirem com o símbolo atual da cadeia de entrada executam a produção.
- (Symbol) Next Symbol: símbolo seguinte da lista ligada contendo todo o alfabeto do sistema.

Pilha: Stack

- (State[]) Returning States: trata-se da pilha em si, pois é o vetor de estados e contém os objetos da classe State empilhados sempre que uma outra submáquina é chamada para execução.
- (Integer) Stack Pointer: elemento da estrutura de dados da pilha, armazena a posição da pilha atualmente. Vale -1 caso a pilha esteja vazia.
- (Integer) Stack size: tamanho total da pilha, utilizado na inicialização do vetor que representa a pilha para estabelecer o tamanho deste vetor.

Listas ligadas: Linked Lists

Todas as classes que possuem mais de um objeto formando listas são implementados em listas ligadas. São eles:

- ProductionLinkedList
- StateLinkedList
- SubmachineLinkedList

- SymbolLinkedList

Todas estas listas são semelhantes e possuem os mesmos atributos de listas ligadas usuais. São compostas pela forma genérica da estrutura de dados lista ligadas, de modo que possuem como atributo somente o primeiro nó da lista, sendo ele um objeto da classe para a qual a lista foi criada.

Métodos das classes do programa

Acima foram descritos os atributos que caracterizam cada classe presente no sistema para a construção dos autômatos. A seguir, especifica-se os métodos das classes.

Autômato: Automaton

- Automaton(): construtor da classe, inicializa todas as listas ligadas do sistema, e inicia a execução com accepted=false e error=false.
- void createMachineStructure(): lê os dados do arquivo txt e cria os estados do sistema (StateID e SubmachineID). Por default, o primeiro estado lido no arquivo é o estado inicial do autômato, e o primeiro estado lido de uma nova submáquina é o estado inicial desta submáquina. Lê-se também do arquivo se se deseja ver o processo de leitura da cadeia pela máquina (mostra por quais estados está se passando, qual o símbolo lido em cada estado e qual o estado seguinte, efetuada a transição, ou se houve um comando de RETURN ou de STACK). Em seguida, lê-se do arquivo os símbolos do alfabeto que descrevem a linguagem descrita pelo autômato. Por fim, lê-se as produções até que o arquivo finalize, sendo estas produções do formato descrito acima, dos tipos1, 2 ou 3.
- void createStateAndSubmachine(): este método recebe um vetor de strings lido do arquivo txt que contém o estado e a submáquina deste estado separados por vírgula. Cria-se o estado com o “new” da linguagem Java e adiciona-se o estado a lista de estados StateLinkedList do autômato. O igual processo é feito para submáquinas, ou seja, cria-se um novo objeto da classe Submachine e adiciona-o no objeto da classe SubmachneLinkedList para conter a lista de todas as submáquinas do sistema.
- void get_productions(): lê-se as produções passadas pelo arquivo txt. Elas devem ser do formato descrito acima, respeitando o formato dos tipos 1, 2 ou 3.
- void displayStatus(): display do status atual do sistema, ou seja, estado atual; cadeia sendo analizada no momento (a partir do símbolo apontado pelo cursor atualmente e todos os símbolos à sua direita); próximo estado, mostrando o estado e a submáquina atuais; comando, indicando se trata-se apenas de uma transição interna (comando = “---”), chamada de submáquina (comando = “STACK”) ou retorno de submáquina e finalização de execução na submáquina em questão (comando = “RETURN”).
- void analyzeSymbol(String input): principal método do programa. Caso deseja-se ver as passagens pelo autômato finito (show_track = 'y' no arquivo de descrição da máquina) o método anterior displayStatus() é constantemente chamado a cada transição efetuada. Também neste método, lê-se o simbolo embaixo do cabeçote e compara-o com as transições

possíveis para aquele estado. Caso ele não possua transição possível, ou seja, não há transição para aquele dado estado e símbolo de entrada, toma-se como novo símbolo a ser analisado o atual mais o seguinte, e assim por diante. Caso não se encontre transição possível para o estado atual e o símbolo de entrada = toda a word a direita do cabeçote, finaliza-se a análise e envia-se sinal de erro “Cadeia não aceita pelo autômato”. Caso haja transição possível e ela seja do tipo 1, torna-se o símbolo atual como igual ao estado seguinte dado pela produção, move-se o cabeçote de leitura, ou seja, incrementa-se a posição de leitura da string inpu. Caso seja do tipo 2, armazena-se o estado passado pela produção na Stack, e seta-se o estado seguinte como igual ao estado inicial da submáquina chamada por este comando, e como a transição ocorre em vazio, o cabeçote permanece na mesma posição de leitura. Caso seja do tipo 3, retorna-se ao estado presente no topo da pilha, setando-o como estado atual; caso o topo da pilha seja vazio, verifica-se se a word a ser lida é a cadeia vazia, se for finaliza-se o processo aceitando a cadeia de entrada, e se não for envia-se mensagem de erro pois o autômato foi finalizado mas a cadeia de entrada não, sem mais transições possíveis para ela; o cabeçote se mantém também na mesma posição, pois transições entre submáquinas são feitas como transição em vazio.

Submáquinas: submachines

- Submachine(int initial_state, int submacine_id): construtor que inicializa estado inicial e id da submáquina.
- gets e sets dos atributos privados.

Estados da submáquina: State

- State(int state_id, int submacine_id): construtor que inicializa id do estado e id da submáquina. Além do next_state, para compor a lista ligada, em null.
- gets e sets dos atributos privados.

Produção: Production

- public Production(State state, Symbol input, State next_state, String command, Submachine next_submachine): inicializa estado em que ocorre produção, símbolo de input, estado seguinte caso a transiçã ocorra, comando caso seja do tipo 2 ou 3, e próxima submáquina caso seja do tipo 2.
- gets e sets dos atributos

Símbolos: Symbol

- Symbol(String symbol): inicializa com a string que é este elemento do alfabeto da linguagem descrita pelo autômato finito.
- gets e sets dos atributos

Pilha: Stack

- Stack(int size): inicializa o vetor da pilha com o size, além do tamanho da piha e do stack pointer em -1.

- void push(State state): se pilha não atingiu tamanho máximo, incremetar stack pointer e igualar este setor do vetor com state; caso contrário, mostrar mensagem de erro.
- State pop(): se stack_pointer >= 0 (ainda há elementos na pilha), retornar ele e decrementar stack pointer; caso contário, retornar mensagem de não mais estados na pilha.
- void displayStatus(): mostra os IDs dos estados atualmente na pilha em sua ordem de cima para baixo.

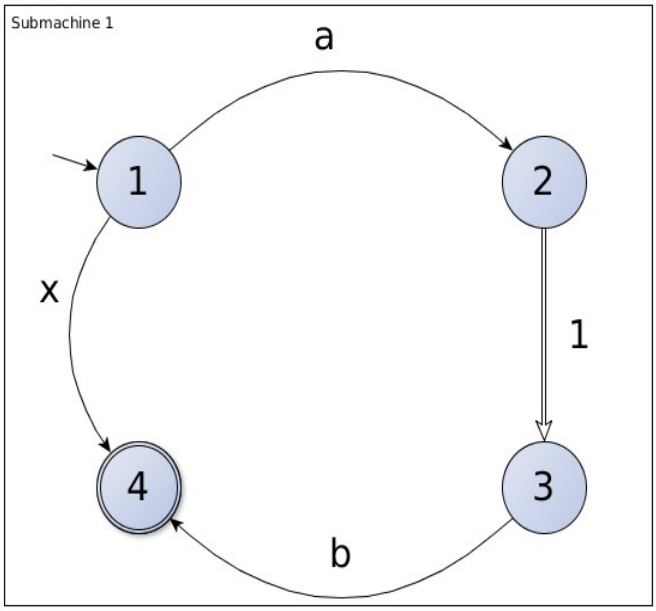
Listas ligadas: Linked Lists

As listas ligadas foram criadas de forma particular cada uma, porém elas sempre seguem o padrão da estrutura de dados Lista Ligada. Portanto contém, de forma geral, os seguintes métodos:

- void insert(Node node): insere como primeiro elemento da fila
- Node search(int node_id): busca um node na lista comparando seu ID.
- void display(): mostra no console todos os elementos da lista, provavelmente printando seu ID.

Testes Realizados

Primeiro teste



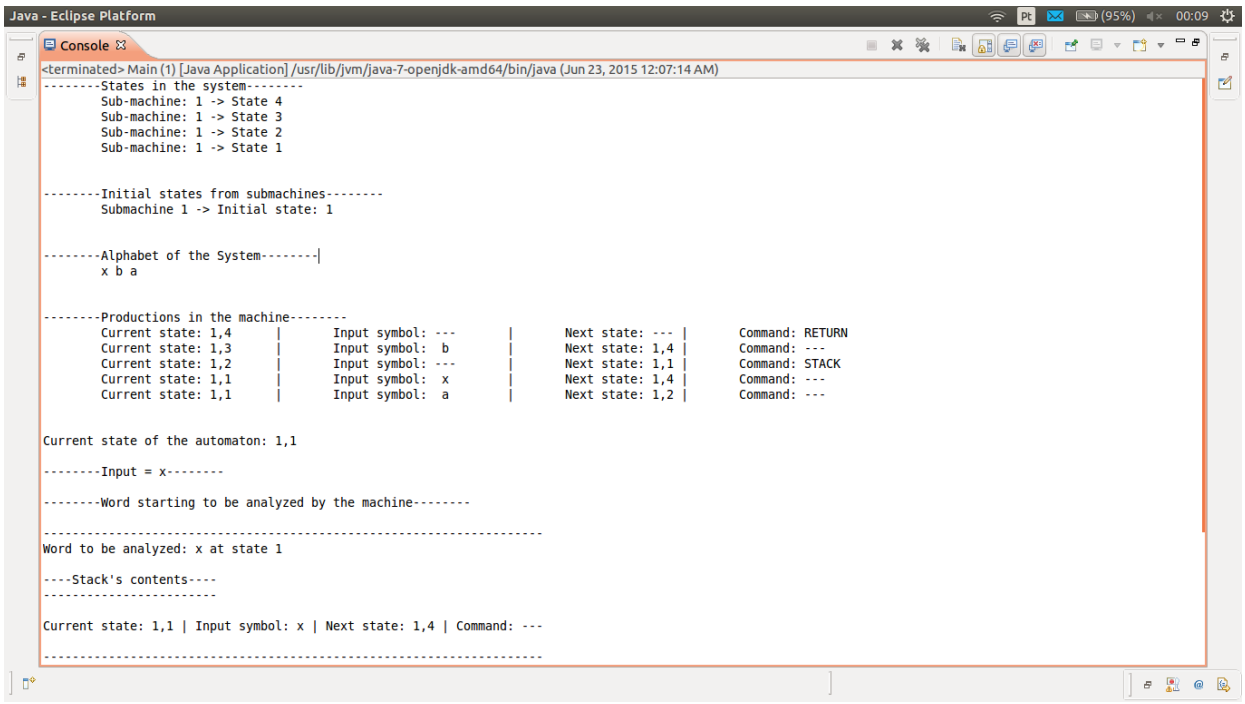
Este autômato aceita cadeias do tipo x, axb, aaxbb, etc. Ou seja, palíndromos delimitados pelo x no centro. A seguir está o arquivo de descrição do autômato:

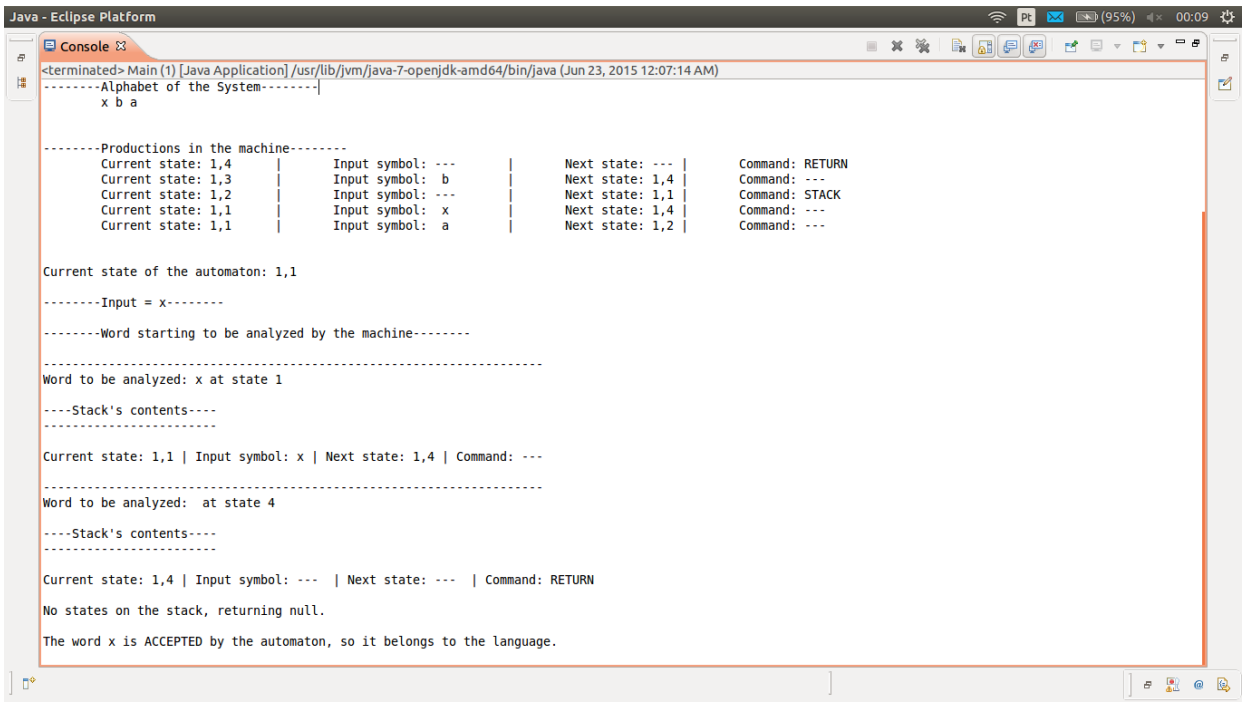
word.txt × machine.txt ×

```
1,1 1,2 1,3 1,4
y
a b x
1,1 a 1,2
1,1 x 1,4
1,2 1 STACK 1,3
1,3 b 1,4
1,4 RETURN
```

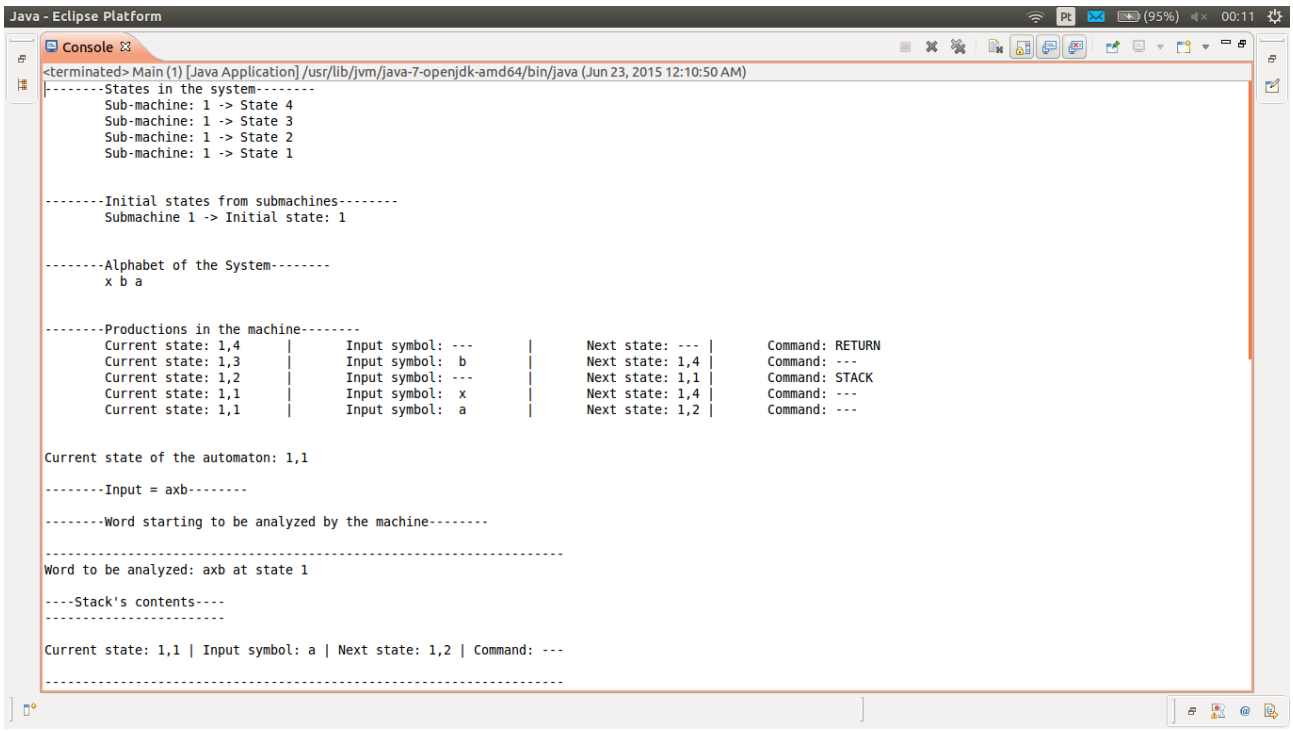
As palavras testadas foram as seguintes:

1. x: Aceito

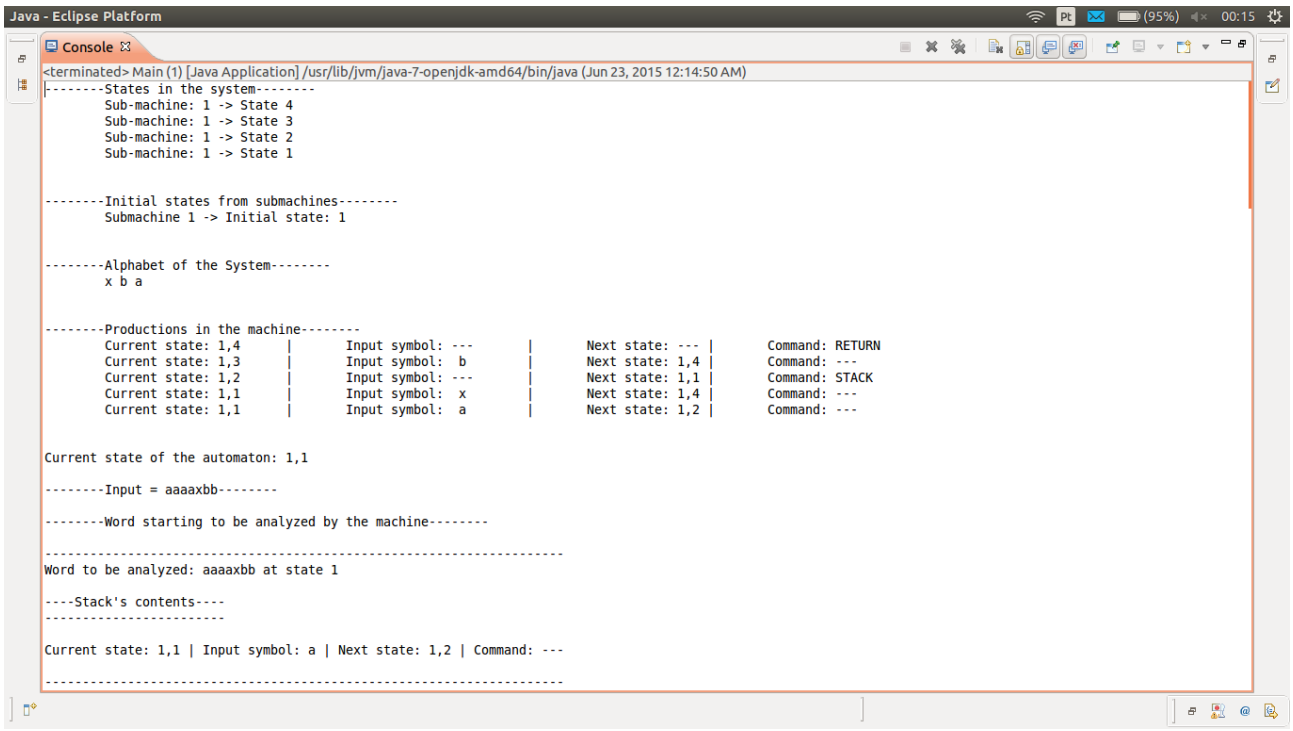

Java - Eclipse Platform
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 12:07:14 AM)
--------States in the system--------
Sub-machine: 1 -> State 4
Sub-machine: 1 -> State 3
Sub-machine: 1 -> State 2
Sub-machine: 1 -> State 1
--------Initial states from submachines--------
Submachine 1 -> Initial state: 1
--------Alphabet of the System--------
x b a
--------Productions in the machine--------
Current state: 1,4 | Input symbol: --- | Next state: --- | Command: RETURN
Current state: 1,3 | Input symbol: b | Next state: 1,4 | Command: ---
Current state: 1,2 | Input symbol: --- | Next state: 1,1 | Command: STACK
Current state: 1,1 | Input symbol: x | Next state: 1,4 | Command: ---
Current state: 1,1 | Input symbol: a | Next state: 1,2 | Command: ---
Current state of the automaton: 1,1
--------Input = x--------
--------Word starting to be analyzed by the machine--------
Word to be analyzed: x at state 1
----Stack's contents----
Current state: 1,1 | Input symbol: x | Next state: 1,4 | Command: ---

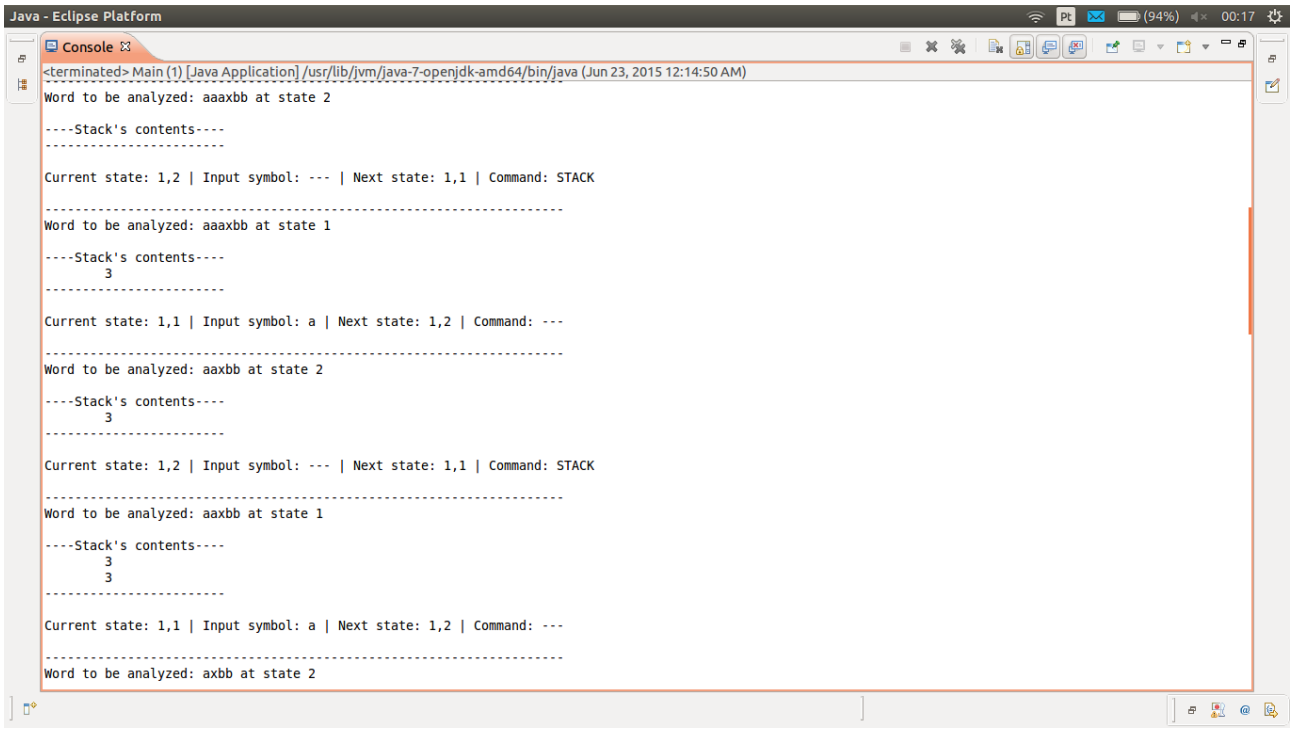

Java - Eclipse Platform
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 12:07:14 AM)
--------Alphabet of the System--------
x b a
--------Productions in the machine--------
Current state: 1,4 | Input symbol: --- | Next state: --- | Command: RETURN
Current state: 1,3 | Input symbol: b | Next state: 1,4 | Command: ---
Current state: 1,2 | Input symbol: --- | Next state: 1,1 | Command: STACK
Current state: 1,1 | Input symbol: x | Next state: 1,4 | Command: ---
Current state: 1,1 | Input symbol: a | Next state: 1,2 | Command: ---
Current state of the automaton: 1,1
--------Input = x--------
--------Word starting to be analyzed by the machine--------
Word to be analyzed: x at state 1
----Stack's contents----
Current state: 1,1 | Input symbol: x | Next state: 1,4 | Command: ---
Word to be analyzed: at state 4
----Stack's contents----
Current state: 1,4 | Input symbol: --- | Next state: --- | Command: RETURN
No states on the stack, returning null.
The word x is ACCEPTED by the automaton, so it belongs to the language.

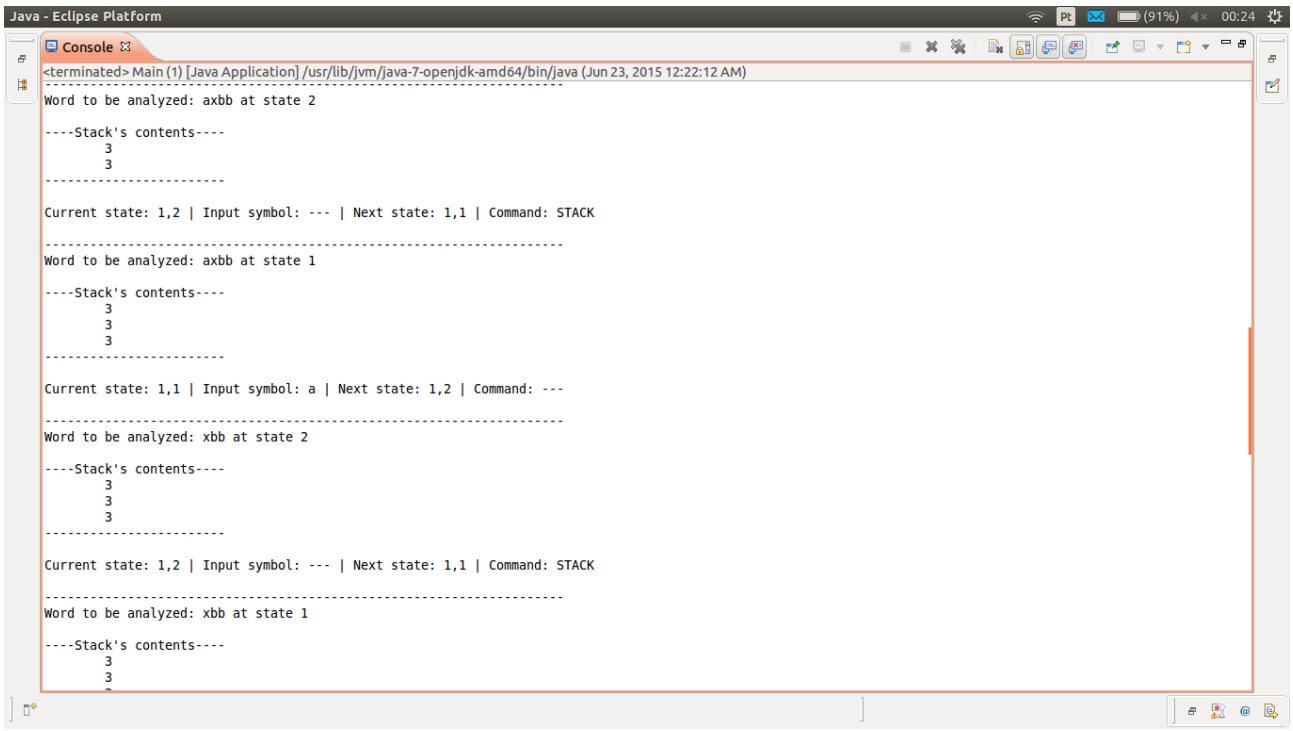
2. axb: Aceito

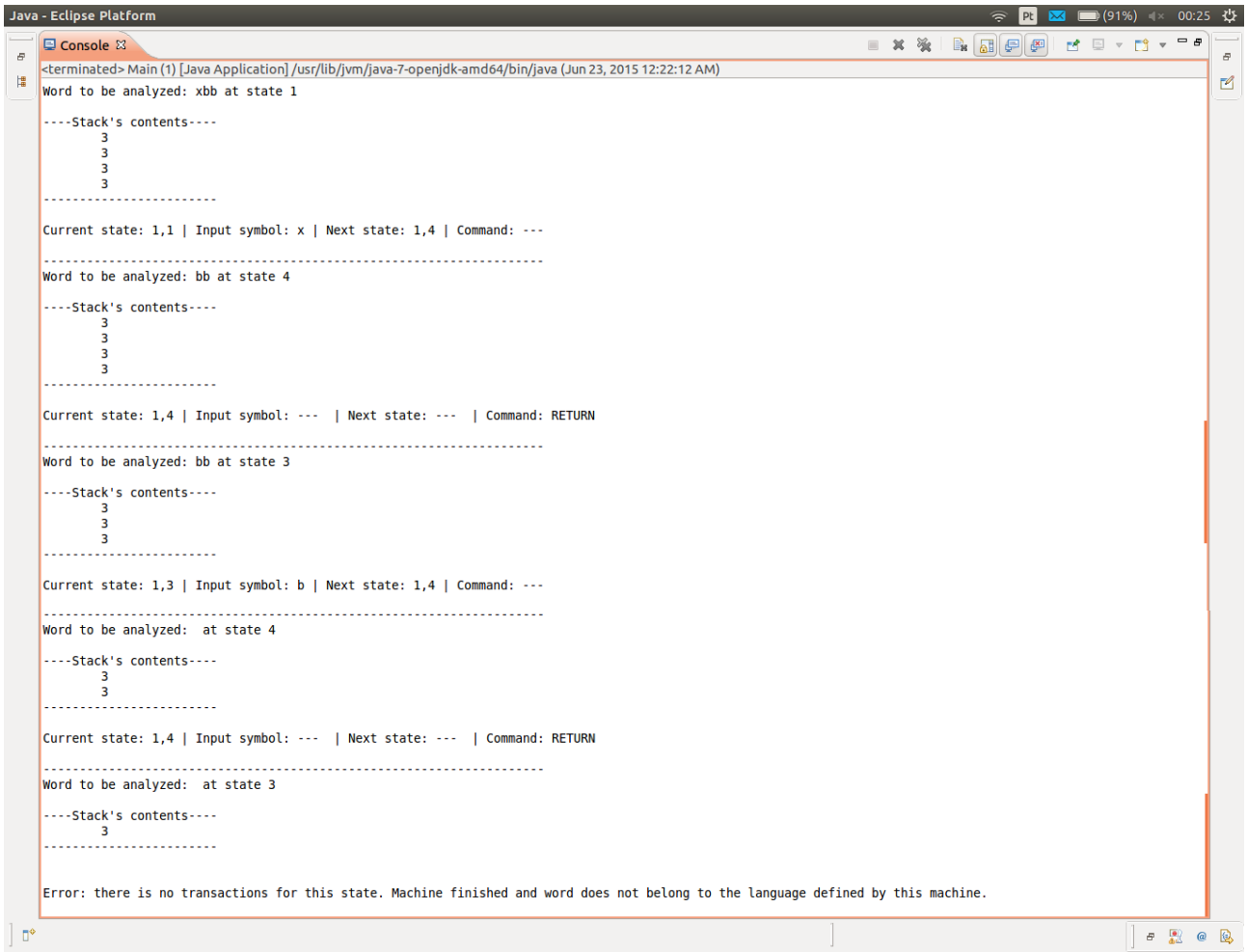


3. aaaaxbb: Não aceito

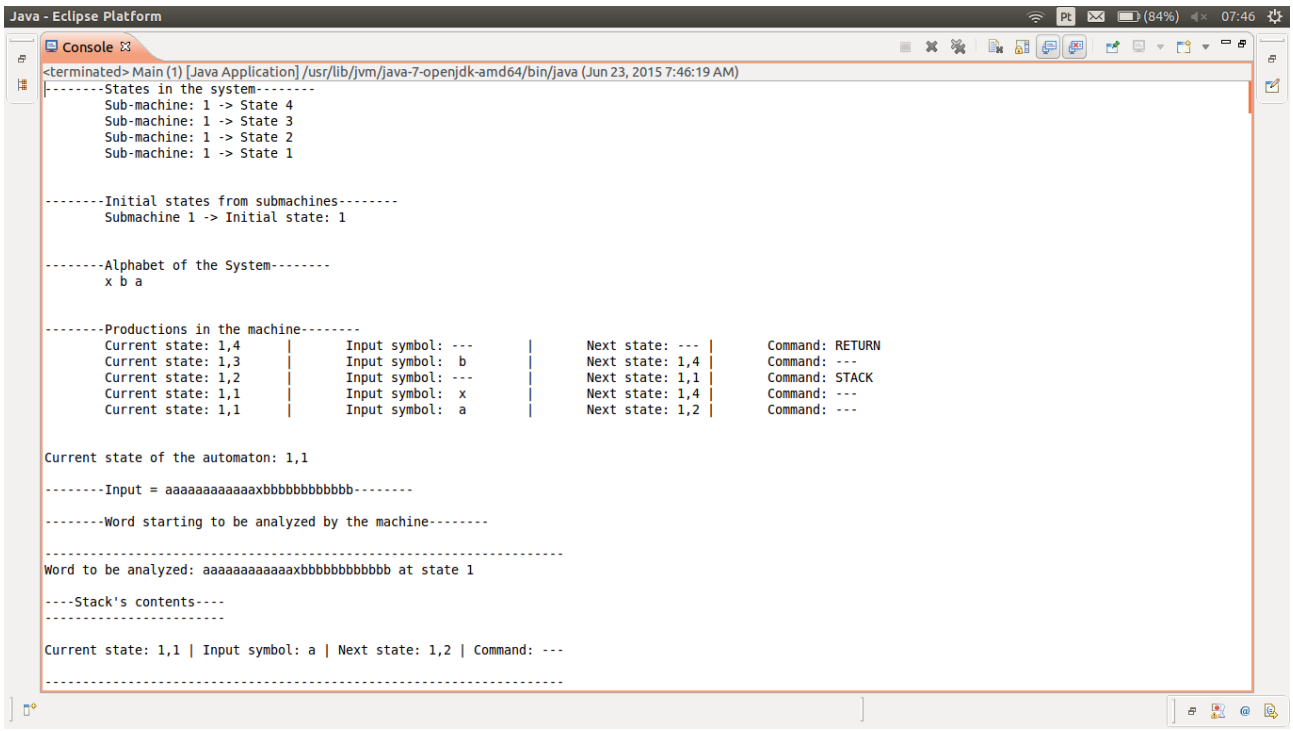

Java - Eclipse Platform
(95%) 00:15
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 12:14:50 AM)
--------States in the system--------
Sub-machine: 1 -> State 4
Sub-machine: 1 -> State 3
Sub-machine: 1 -> State 2
Sub-machine: 1 -> State 1
--------Initial states from submachines--------
Submachine 1 -> Initial state: 1
--------Alphabet of the System--------
x b a
--------Productions in the machine--------
Current state: 1,4 | Input symbol: --- | Next state: --- | Command: RETURN
Current state: 1,3 | Input symbol: b | Next state: 1,4 | Command: ---
Current state: 1,2 | Input symbol: --- | Next state: 1,1 | Command: STACK
Current state: 1,1 | Input symbol: x | Next state: 1,4 | Command: ---
Current state: 1,1 | Input symbol: a | Next state: 1,2 | Command: ---
Current state of the automaton: 1,1
--------Input = aaaaxbb--------
--------Word starting to be analyzed by the machine--------
Word to be analyzed: aaaaxbb at state 1
----Stack's contents----
Current state: 1,1 | Input symbol: a | Next state: 1,2 | Command: ---

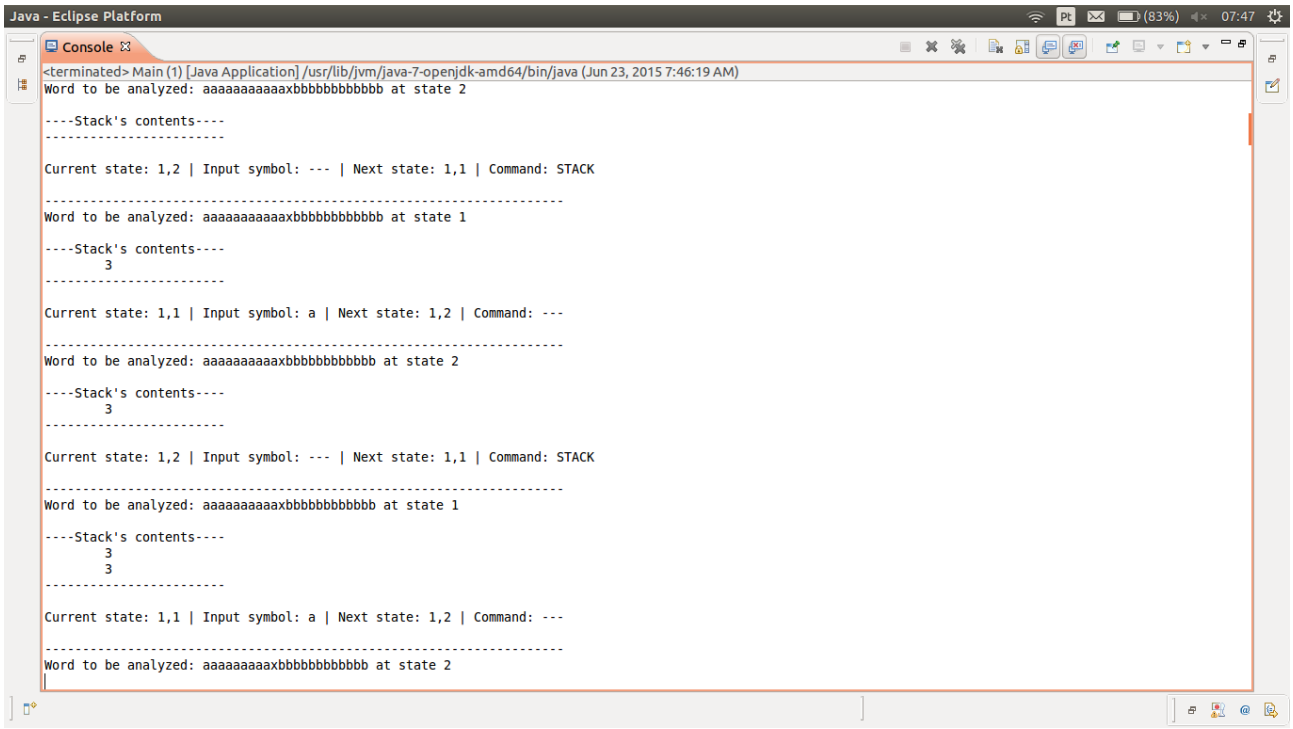

Java - Eclipse Platform
(94%) 00:17
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 12:14:50 AM)
Word to be analyzed: aaaxbb at state 2
----Stack's contents----
Current state: 1,2 | Input symbol: --- | Next state: 1,1 | Command: STACK
Word to be analyzed: aaaxbb at state 1
----Stack's contents----
3
Current state: 1,1 | Input symbol: a | Next state: 1,2 | Command: ---
Word to be analyzed: aaxbb at state 2
----Stack's contents----
3
Current state: 1,2 | Input symbol: --- | Next state: 1,1 | Command: STACK
Word to be analyzed: aaxbb at state 1
----Stack's contents----
3
3
Current state: 1,1 | Input symbol: a | Next state: 1,2 | Command: ---
Word to be analyzed: axbb at state 2

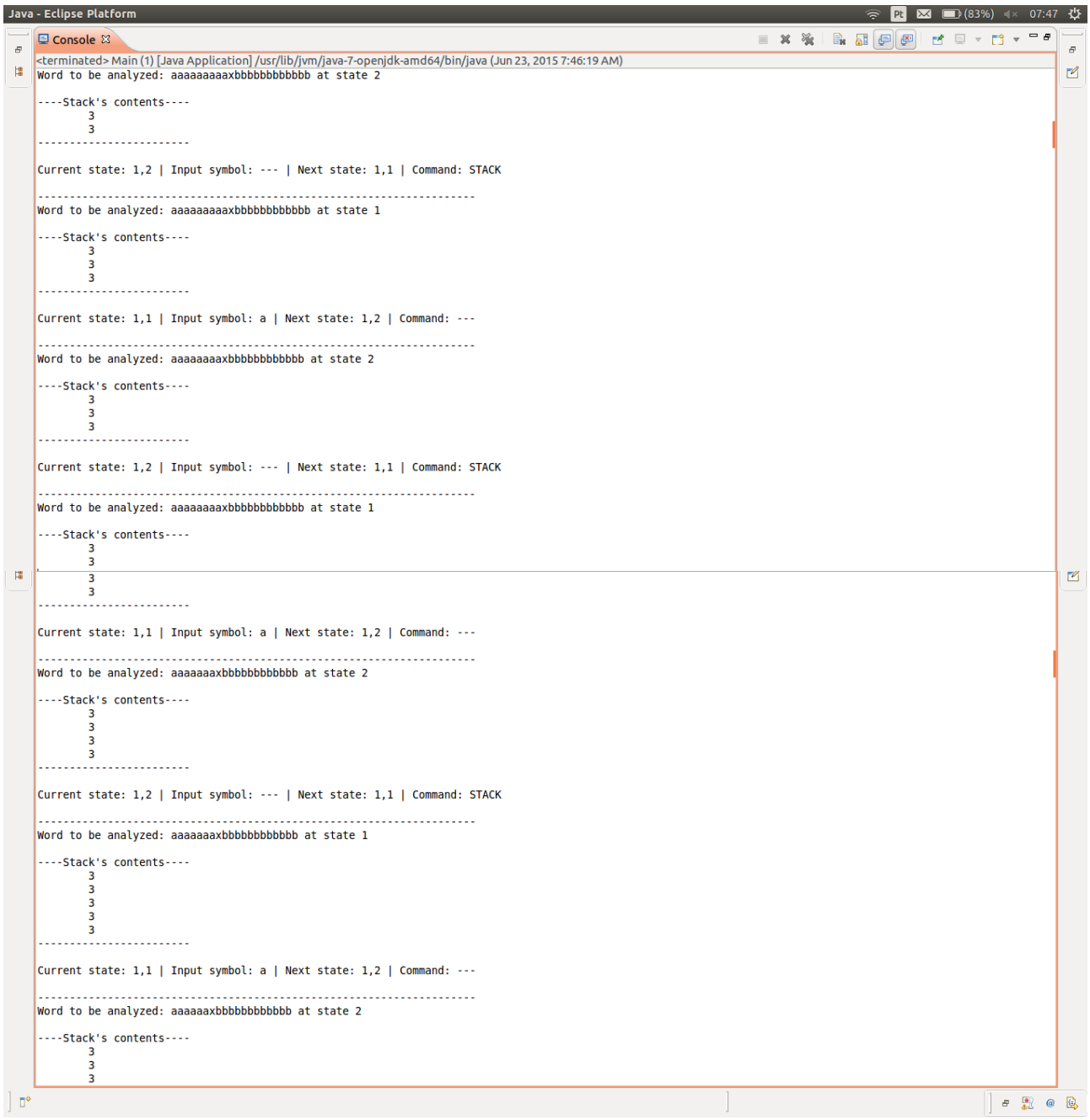


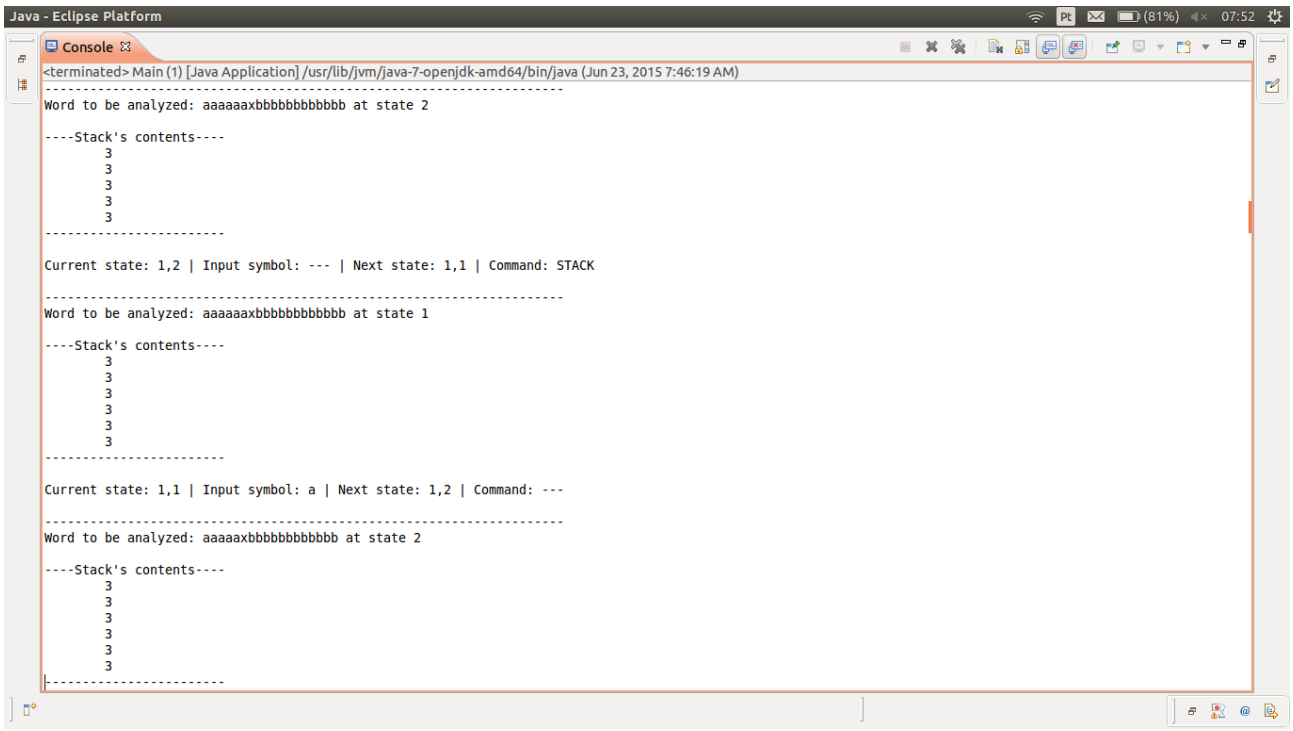


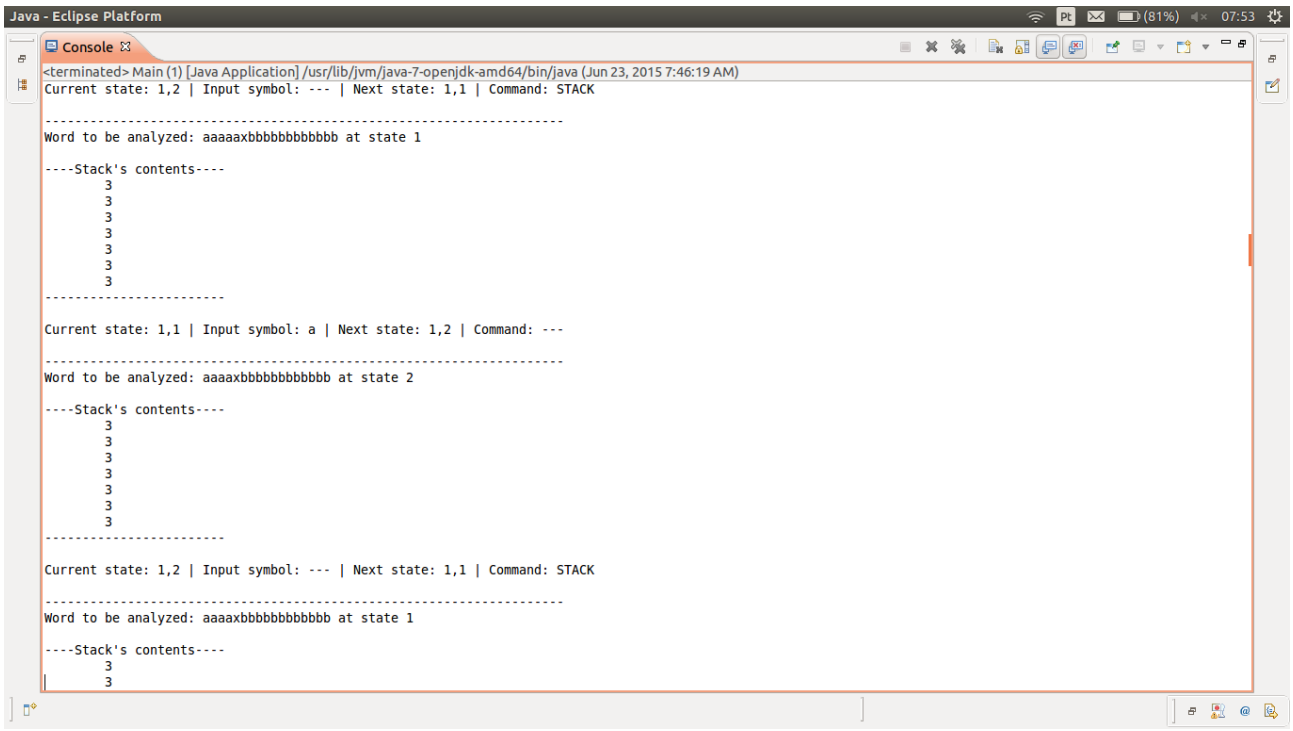
4. aaaaaaaaaaaaaaaaaxbbbbbbbbbbbbbbbbb

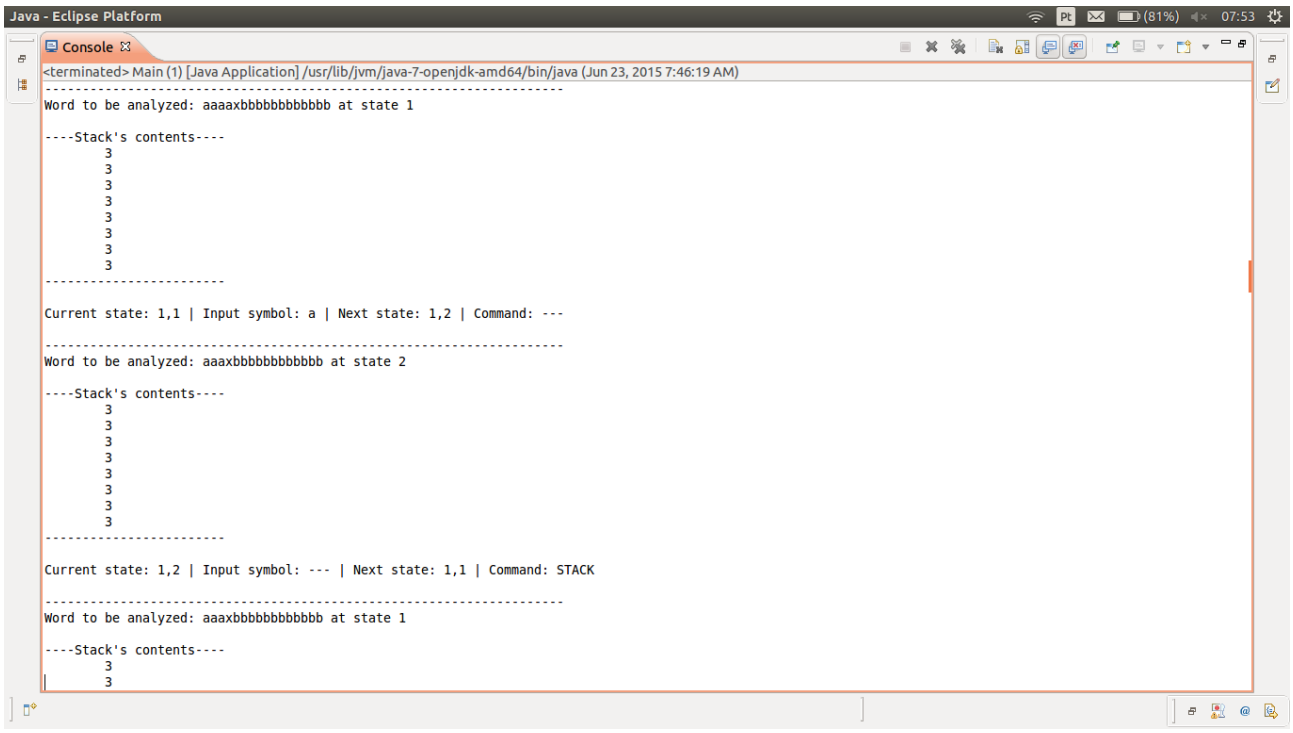

Java - Eclipse Platform
(84%) 07:46
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 7:46:19 AM)
--------States in the system--------
Sub-machine: 1 -> State 4
Sub-machine: 1 -> State 3
Sub-machine: 1 -> State 2
Sub-machine: 1 -> State 1
--------Initial states from submachines--------
Submachine 1 -> Initial state: 1
--------Alphabet of the System--------
x b a
--------Productions in the machine--------
Current state: 1,4 | Input symbol: --- | Next state: --- | Command: RETURN
Current state: 1,3 | Input symbol: b | Next state: 1,4 | Command: ---
Current state: 1,2 | Input symbol: --- | Next state: 1,1 | Command: STACK
Current state: 1,1 | Input symbol: x | Next state: 1,4 | Command: ---
Current state: 1,1 | Input symbol: a | Next state: 1,2 | Command: ---
Current state of the automaton: 1,1
--------Input = aaaaaaaaaaaaxbbbbbbbbbbbb--------
--------Word starting to be analyzed by the machine--------
Word to be analyzed: aaaaaaaaaaaaxbbbbbbbbbbbb at state 1
----Stack's contents----
Current state: 1,1 | Input symbol: a | Next state: 1,2 | Command: ---

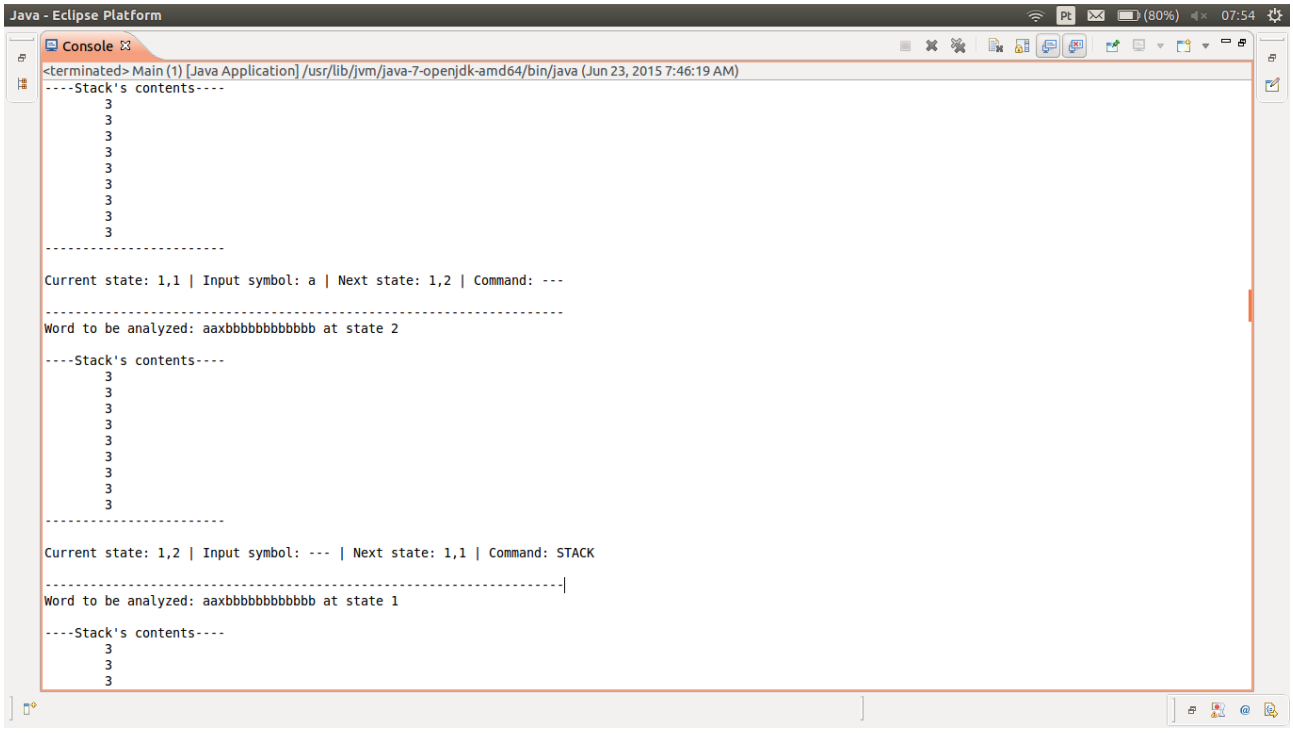

Java - Eclipse Platform
(83%) 07:47
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 7:46:19 AM)
Word to be analyzed: aaaaaaaaaaaxbbbbbbbbbbb at state 2
----Stack's contents----
Current state: 1,2 | Input symbol: --- | Next state: 1,1 | Command: STACK
Word to be analyzed: aaaaaaaaaaaxbbbbbbbbbbb at state 1
----Stack's contents----
3
Current state: 1,1 | Input symbol: a | Next state: 1,2 | Command: ---
Word to be analyzed: aaaaaaaaaaxbbbbbbbbbbb at state 2
----Stack's contents----
3
Current state: 1,2 | Input symbol: --- | Next state: 1,1 | Command: STACK
Word to be analyzed: aaaaaaaaaaxbbbbbbbbbbb at state 1
----Stack's contents----
3
3
Current state: 1,1 | Input symbol: a | Next state: 1,2 | Command: ---
Word to be analyzed: aaaaaaaaaxbbbbbbbbbbb at state 2

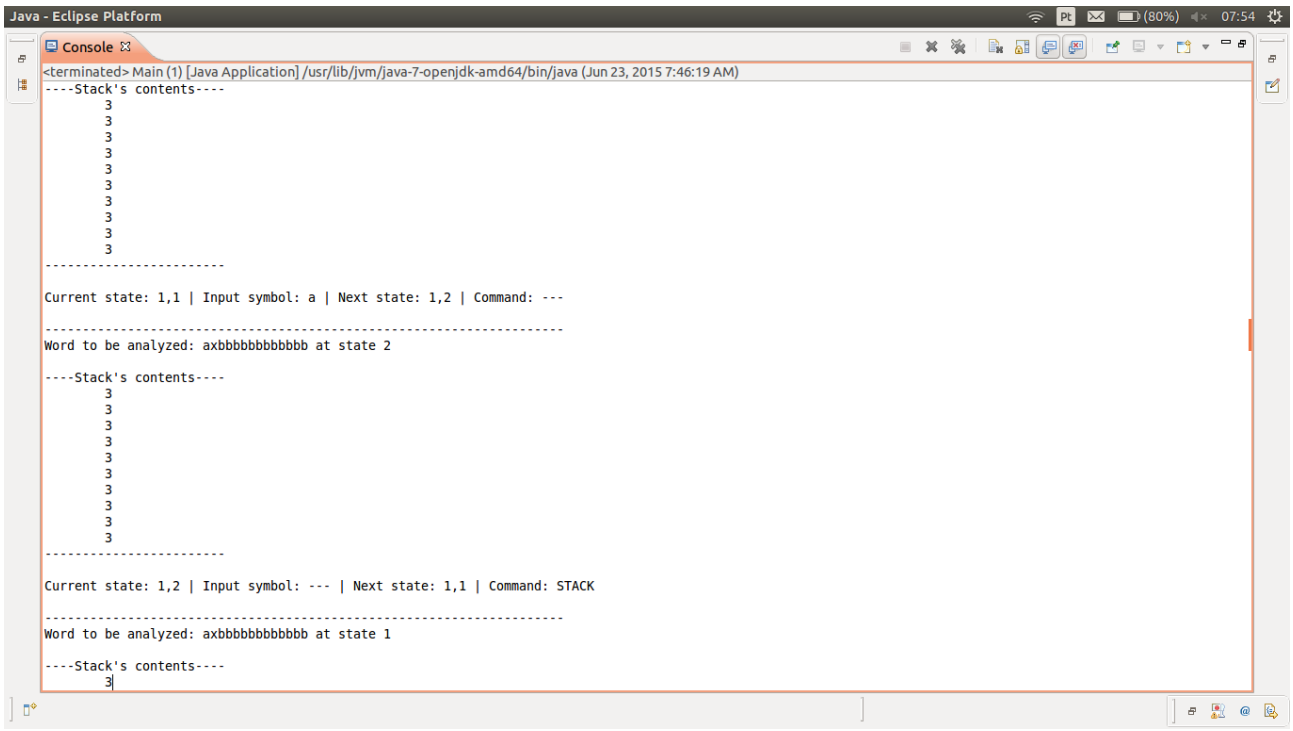

Java - Eclipse Platform
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 7:46:19 AM)
Word to be analyzed: aaaaaaaaxbbbbbbbbbbb at state 2
----Stack's contents----
3
3
Current state: 1,2 | Input symbol: --- | Next state: 1,1 | Command: STACK
Word to be analyzed: aaaaaaaaxbbbbbbbbbbb at state 1
----Stack's contents----
3
3
3
Current state: 1,1 | Input symbol: a | Next state: 1,2 | Command: ---
Word to be analyzed: aaaaaaaxbbbbbbbbbbb at state 2
----Stack's contents----
3
3
3
Current state: 1,2 | Input symbol: --- | Next state: 1,1 | Command: STACK
Word to be analyzed: aaaaaaaxbbbbbbbbbbb at state 1
----Stack's contents----
3
3
3
3
Current state: 1,1 | Input symbol: a | Next state: 1,2 | Command: ---
Word to be analyzed: aaaaaaxbbbbbbbbbbb at state 2
----Stack's contents----
3
3
3
3
Current state: 1,2 | Input symbol: --- | Next state: 1,1 | Command: STACK
Word to be analyzed: aaaaaaxbbbbbbbbbbb at state 1
----Stack's contents----
3
3
3
3
3
Current state: 1,1 | Input symbol: a | Next state: 1,2 | Command: ---
Word to be analyzed: aaaaaaxbbbbbbbbbbb at state 2
----Stack's contents----
3
3
3


Java - Eclipse Platform
(81%) 07:52
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 7:46:19 AM)
Word to be analyzed: aaaaaaxbbbbbbbbbbb at state 2
----Stack's contents----
3
3
3
3
3
Current state: 1,2 | Input symbol: --- | Next state: 1,1 | Command: STACK
Word to be analyzed: aaaaaaxbbbbbbbbbbb at state 1
----Stack's contents----
3
3
3
3
3
3
Current state: 1,1 | Input symbol: a | Next state: 1,2 | Command: ---
Word to be analyzed: aaaaaxbbbbbbbbbbb at state 2
----Stack's contents----
3
3
3
3
3
3


Java - Eclipse Platform
(81%) 07:53
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 7:46:19 AM)
Current state: 1,2 | Input symbol: --- | Next state: 1,1 | Command: STACK
Word to be analyzed: aaaaaxbbbbbbbbbbb at state 1
----Stack's contents----
3
3
3
3
3
3
3
Current state: 1,1 | Input symbol: a | Next state: 1,2 | Command: ---
Word to be analyzed: aaaaxbbbbbbbbbbb at state 2
----Stack's contents----
3
3
3
3
3
3
3
Current state: 1,2 | Input symbol: --- | Next state: 1,1 | Command: STACK
Word to be analyzed: aaaaxbbbbbbbbbbb at state 1
----Stack's contents----
3
3

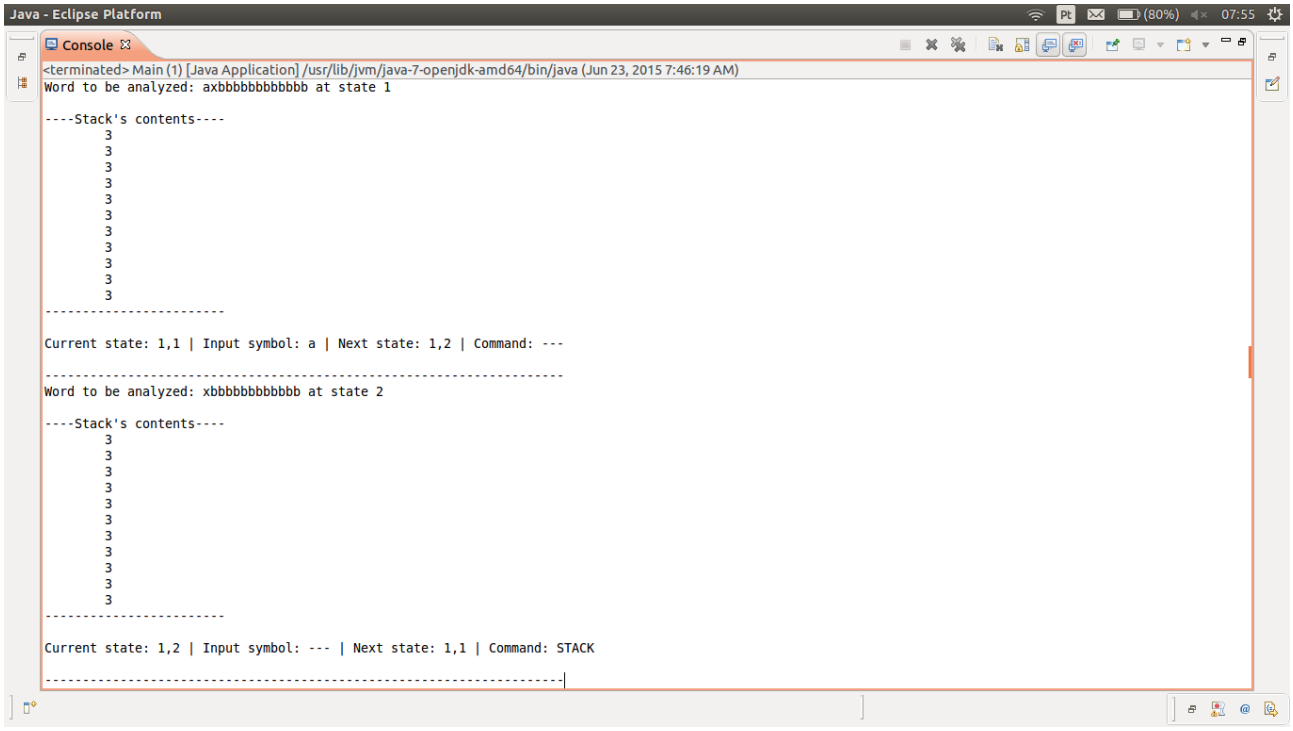

Java - Eclipse Platform
(81%) 07:53
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 7:46:19 AM)
Word to be analyzed: aaaaxbbbbbbbbbbb at state 1
----Stack's contents----
3
3
3
3
3
3
3
3
Current state: 1,1 | Input symbol: a | Next state: 1,2 | Command: ---
Word to be analyzed: aaaxbbbbbbbbbbb at state 2
----Stack's contents----
3
3
3
3
3
3
3
3
Current state: 1,2 | Input symbol: --- | Next state: 1,1 | Command: STACK
Word to be analyzed: aaaxbbbbbbbbbbb at state 1
----Stack's contents----
3
3


Java - Eclipse Platform
(80%) 07:54
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 7:46:19 AM)
----Stack's contents----
3
3
3
3
3
3
3
3
3
Current state: 1,1 | Input symbol: a | Next state: 1,2 | Command: ---
Word to be analyzed: aaxbbbbbbbbbbb at state 2
----Stack's contents----
3
3
3
3
3
3
3
3
3
Current state: 1,2 | Input symbol: --- | Next state: 1,1 | Command: STACK
Word to be analyzed: aaxbbbbbbbbbbb at state 1
----Stack's contents----
3
3
3


Java - Eclipse Platform
07:54
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 7:46:19 AM)
----Stack's contents----
3
3
3
3
3
3
3
3
3
3
3

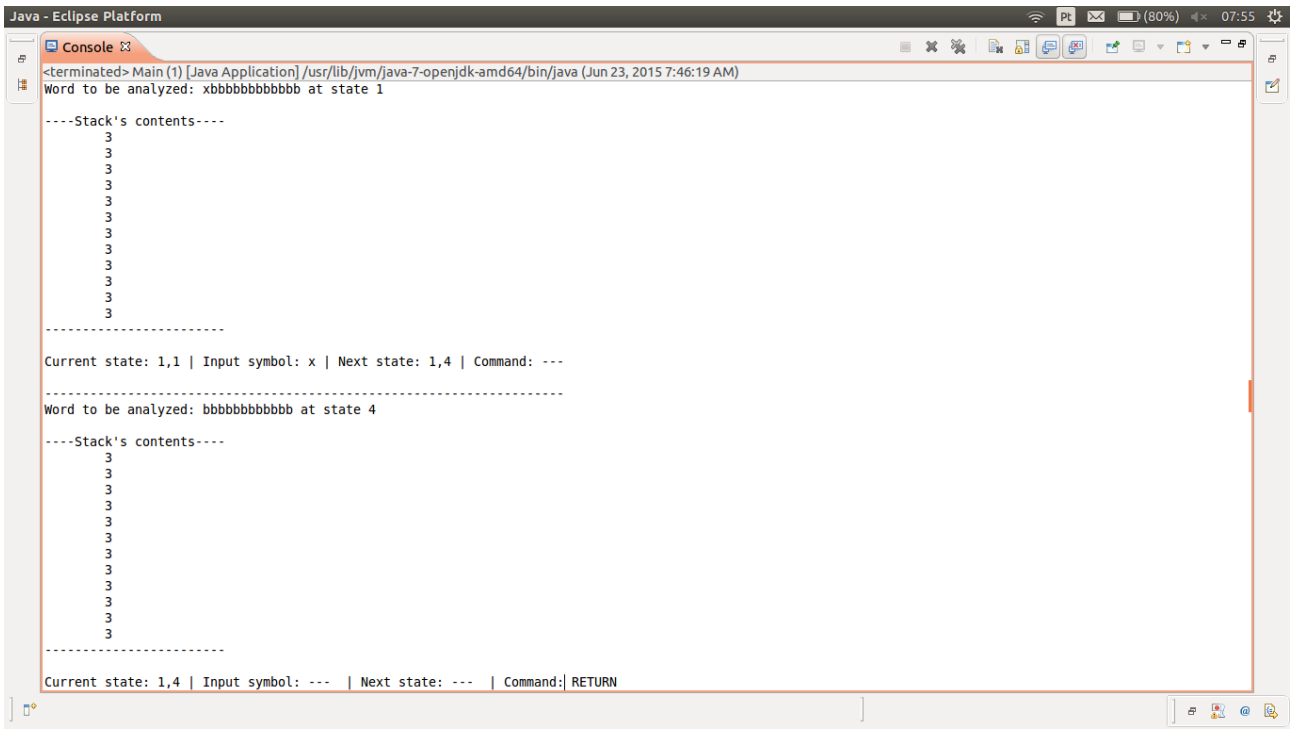
Current state: 1,1 | Input symbol: a | Next state: 1,2 | Command: ---
Word to be analyzed: axbbbbbbbbbbb at state 2
----Stack's contents----
3
3
3
3
3
3
3
3
3
3
3

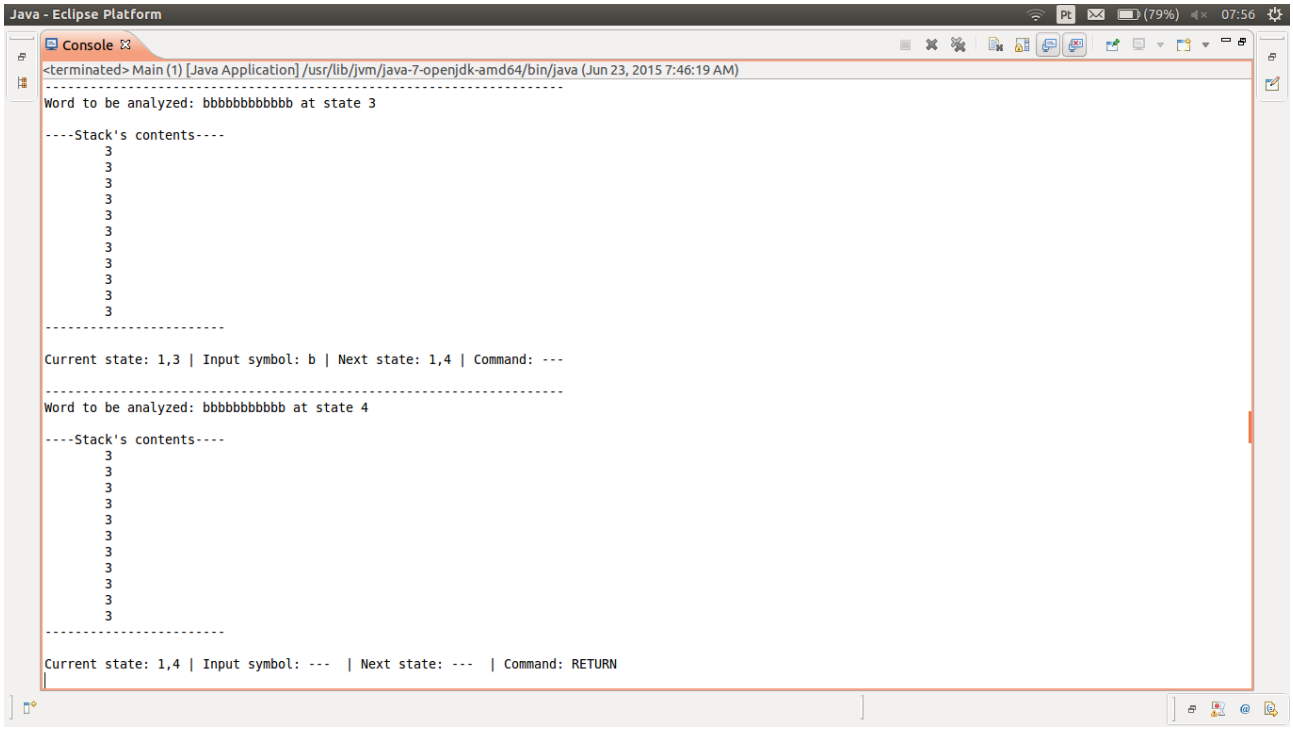
Current state: 1,2 | Input symbol: --- | Next state: 1,1 | Command: STACK
Word to be analyzed: axbbbbbbbbbbb at state 1
----Stack's contents----
3


Java - Eclipse Platform
07:55
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 7:46:19 AM)
Word to be analyzed: axbbbbbbbbbbb at state 1
----Stack's contents----
3
3
3
3
3
3
3
3
3
3
3

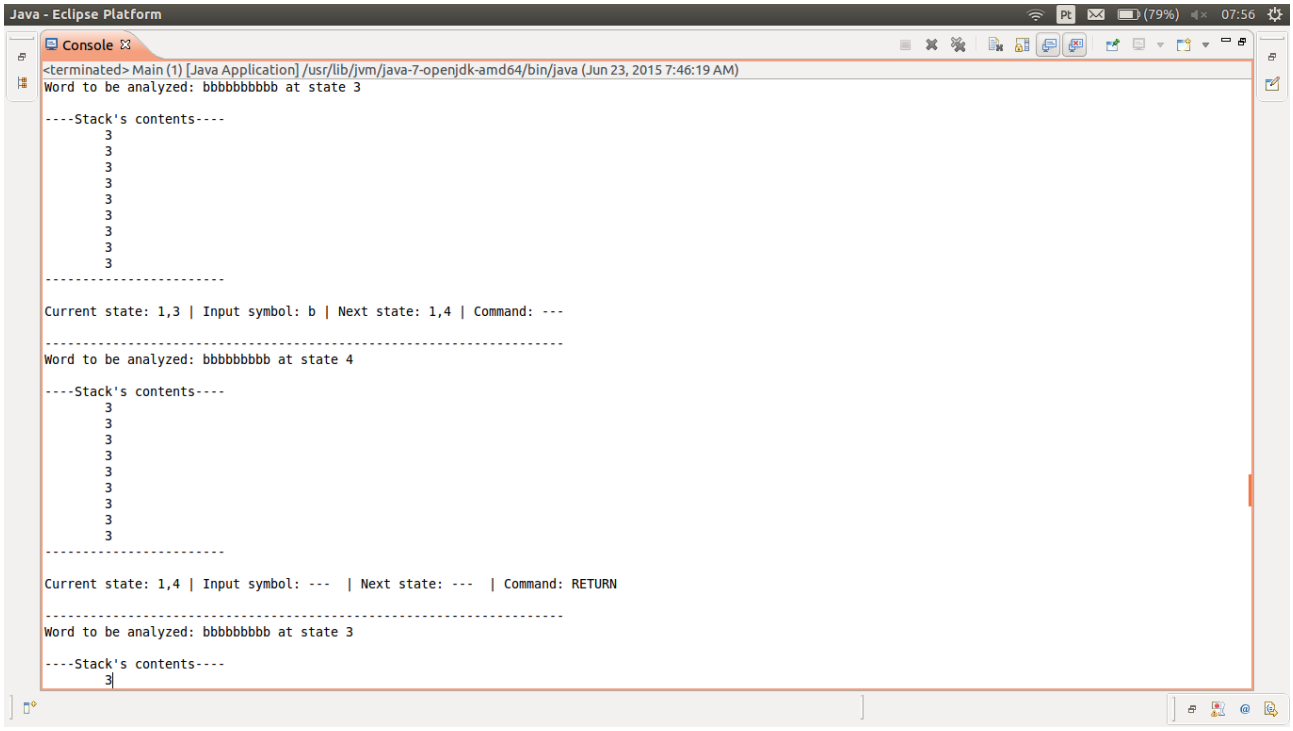
Current state: 1,1 | Input symbol: a | Next state: 1,2 | Command: ---
Word to be analyzed: xbbbbbbbbbbb at state 2
----Stack's contents----
3
3
3
3
3
3
3
3
3
3
3

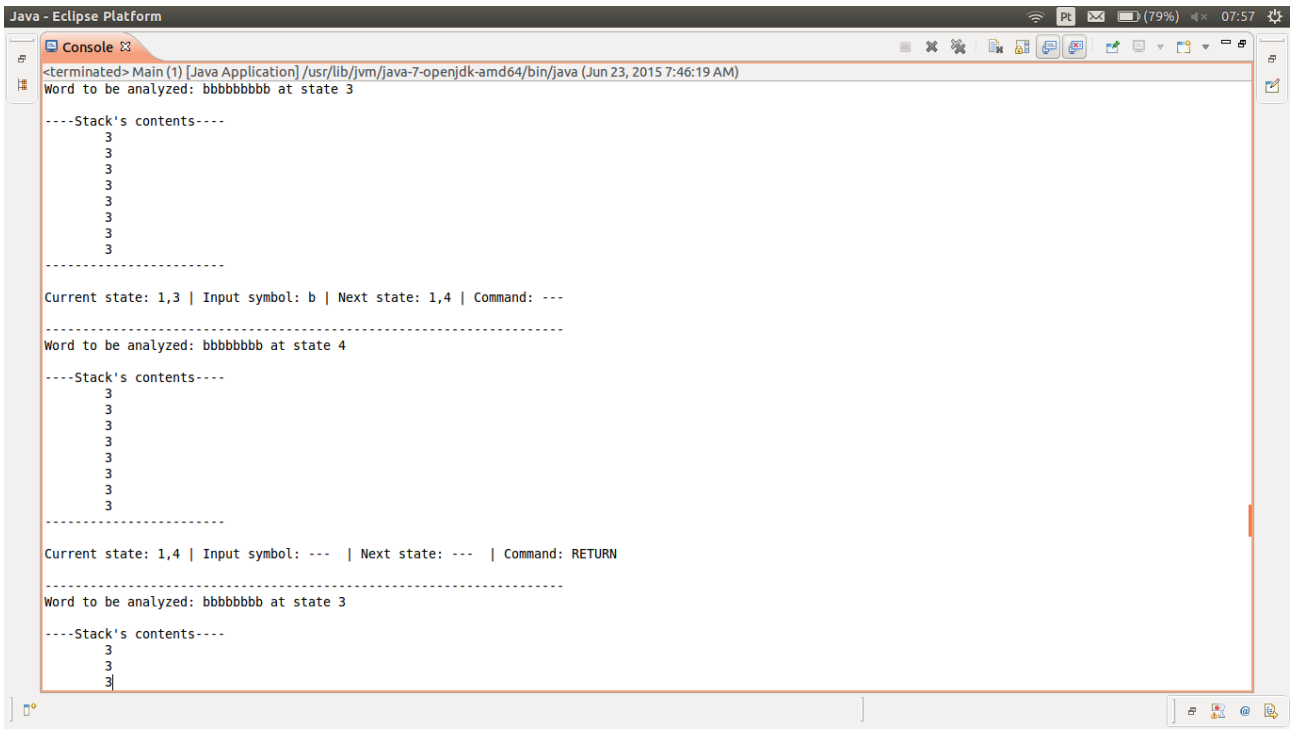
Current state: 1,2 | Input symbol: --- | Next state: 1,1 | Command: STACK

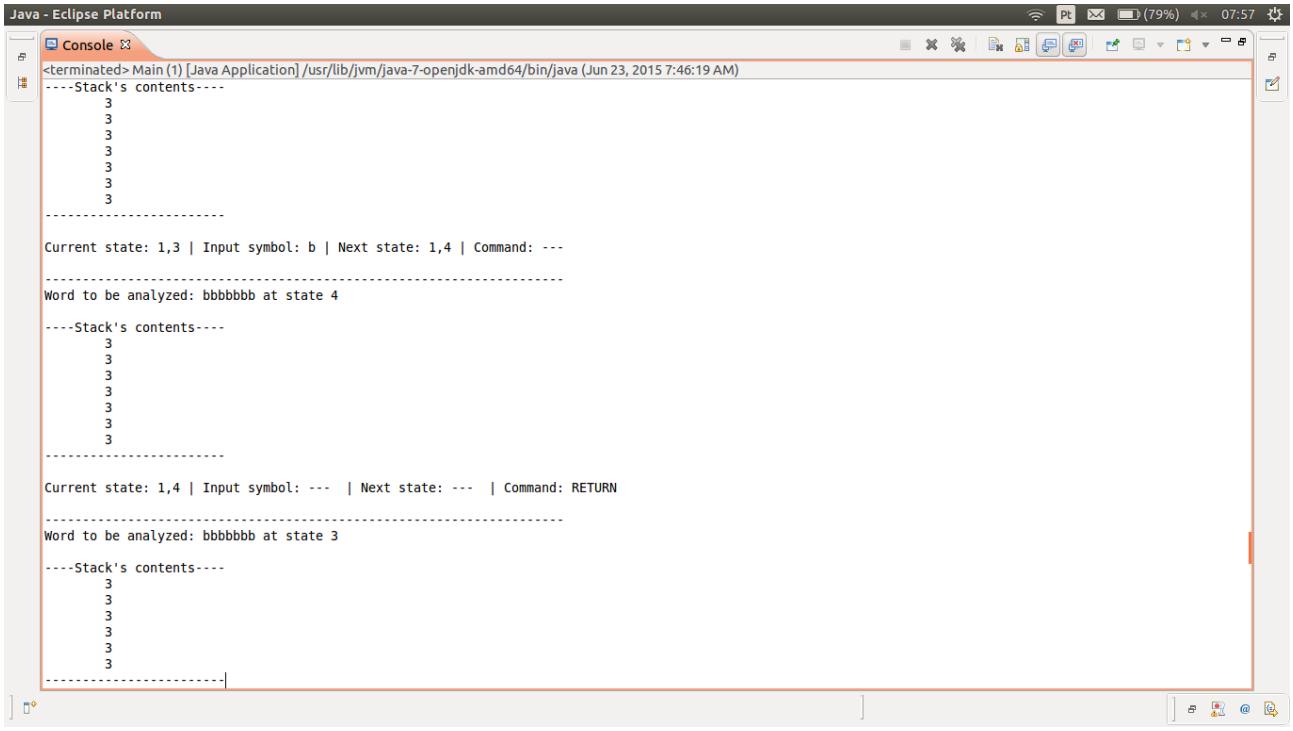

Java - Eclipse Platform
(80%) 07:55
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 7:46:19 AM)
Word to be analyzed: xbbbbbbbbbbb at state 1
----Stack's contents----
3
3
3
3
3
3
3
3
3
3
3
3
Current state: 1,1 | Input symbol: x | Next state: 1,4 | Command: ---
Word to be analyzed: bbbbbbbbbbb at state 4
----Stack's contents----
3
3
3
3
3
3
3
3
3
3
3
3
Current state: 1,4 | Input symbol: --- | Next state: --- | Command: RETURN

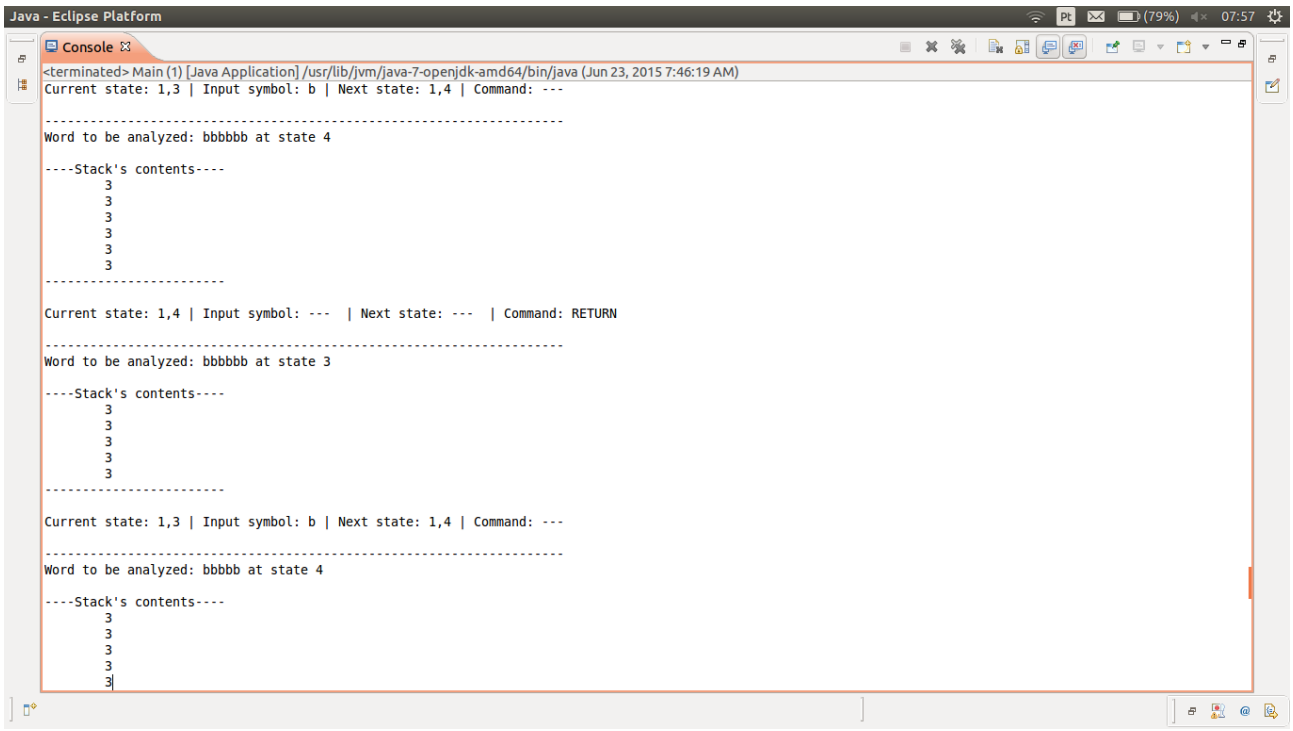

Java - Eclipse Platform
(79%) 07:56
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 7:46:19 AM)
Word to be analyzed: bbbbbbbbbbb at state 3
----Stack's contents----
3
3
3
3
3
3
3
3
3
3
3
3
Current state: 1,3 | Input symbol: b | Next state: 1,4 | Command: ---
Word to be analyzed: bbbbbbbbbb at state 4
----Stack's contents----
3
3
3
3
3
3
3
3
3
3
3
3
Current state: 1,4 | Input symbol: --- | Next state: --- | Command: RETURN


Java - Eclipse Platform
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 7:46:19 AM)
Word to be analyzed: bbbbbbbbbb at state 3
----Stack's contents----
3
3
3
3
3
3
3
3
3
3
Current state: 1,3 | Input symbol: b | Next state: 1,4 | Command: ---
Word to be analyzed: bbbbbbbbb at state 4
----Stack's contents----
3
3
3
3
3
3
3
3
3
3
Current state: 1,4 | Input symbol: --- | Next state: --- | Command: RETURN
Word to be analyzed: bbbbbbbbb at state 3

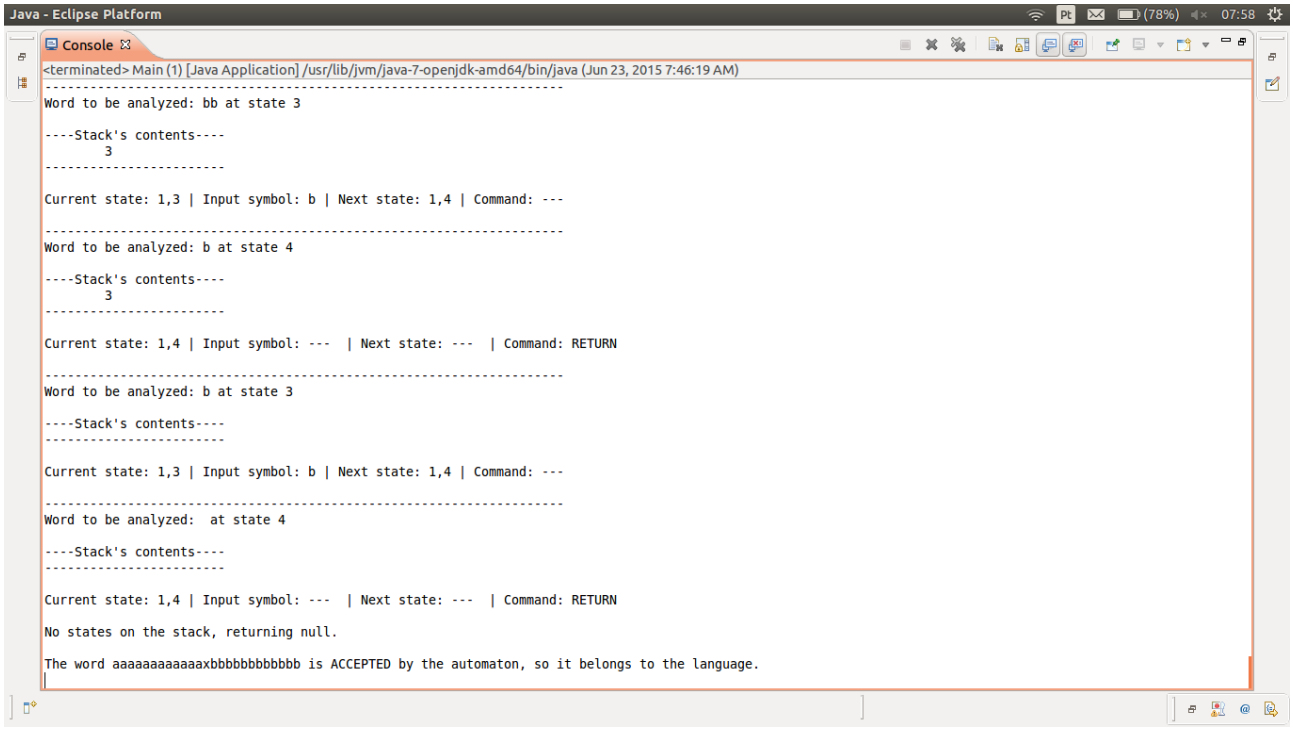

Java - Eclipse Platform
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 7:46:19 AM)
Word to be analyzed: bbbbbbbbb at state 3
----Stack's contents----
3
3
3
3
3
3
3
3
3
3
Current state: 1,3 | Input symbol: b | Next state: 1,4 | Command: ---
Word to be analyzed: bbbbbbbbb at state 4
----Stack's contents----
3
3
3
3
3
3
3
3
3
3
Current state: 1,4 | Input symbol: --- | Next state: --- | Command: RETURN
Word to be analyzed: bbbbbbbbb at state 3
----Stack's contents----
3


Java - Eclipse Platform
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 7:46:19 AM)
Word to be analyzed: bbbbbbbb at state 3
----Stack's contents----
3
3
3
3
3
3
3
3
Current state: 1,3 | Input symbol: b | Next state: 1,4 | Command: ---
Word to be analyzed: bbbbbbbb at state 4
----Stack's contents----
3
3
3
3
3
3
3
3
Current state: 1,4 | Input symbol: --- | Next state: --- | Command: RETURN
Word to be analyzed: bbbbbbbb at state 3
----Stack's contents----
3
3
3


Java - Eclipse Platform
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 7:46:19 AM)
----Stack's contents----
3
3
3
3
3
3
3
Current state: 1,3 | Input symbol: b | Next state: 1,4 | Command: ---
Word to be analyzed: bbbbbb at state 4
----Stack's contents----
3
3
3
3
3
3
3
Current state: 1,4 | Input symbol: --- | Next state: --- | Command: RETURN
Word to be analyzed: bbbbbb at state 3
----Stack's contents----
3
3
3
3
3
3

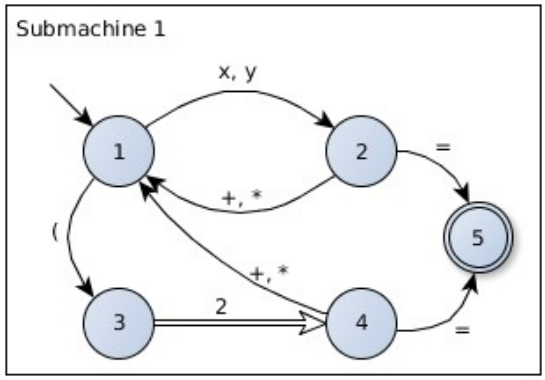

Java - Eclipse Platform
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 7:46:19 AM)
Current state: 1,3 | Input symbol: b | Next state: 1,4 | Command: ---
Word to be analyzed: bbbbbb at state 4
----Stack's contents----
3
3
3
3
3
3
Current state: 1,4 | Input symbol: --- | Next state: --- | Command: RETURN
Word to be analyzed: bbbbbb at state 3
----Stack's contents----
3
3
3
3
3
Current state: 1,3 | Input symbol: b | Next state: 1,4 | Command: ---
Word to be analyzed: bbbbb at state 4
----Stack's contents----
3
3
3
3
3

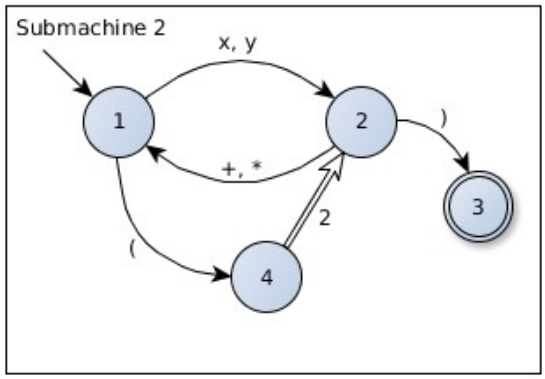

Java - Eclipse Platform
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 7:46:19 AM)
Current state: 1,4 | Input symbol: --- | Next state: --- | Command: RETURN
Word to be analyzed: bbbbb at state 3
----Stack's contents----
3
3
3
3
Current state: 1,3 | Input symbol: b | Next state: 1,4 | Command: ---
Word to be analyzed: bbbb at state 4
----Stack's contents----
3
3
3
3
Current state: 1,4 | Input symbol: --- | Next state: --- | Command: RETURN
Word to be analyzed: bbbb at state 3
----Stack's contents----
3
3
3
Current state: 1,3 | Input symbol: b | Next state: 1,4 | Command: ---



Segundo teste

O seguinte autômato tem por objetivo determinar se a expressão matemática lida está correta ou possui erros. Exemplos de erros seriam parênteses abertos e não fechados, multiplicações ou somas sem o segundo operando, não finalizar a expressão com um sinal de =. O alfabeto desta linguagem é: {x, y, *, +, (,), =}.





O arquivo de entrada de descrição da máquina:

word.txt × machine.txt ×

```
1,1 1,2 1,3 1,4 1,5 2,1 2,2 2,3 2,4
n
x y + * ( ) =
1,1 x 1,2
1,1 y 1,2
1,1 ( 1,3
1,2 = 1,5
1,2 + 1,1
1,2 * 1,1
1,3 2 STACK 1,4
1,4 = 1,5
1,4 * 1,1
1,4 + 1,1
1,5 RETURN
2,1 x 2,2
2,1 y 2,2
2,1 ( 2,4
2,4 2 STACK 2,2
2,2 + 2,1
2,2 * 2,1
2,2 ) 2,3
2,3 RETURN
```

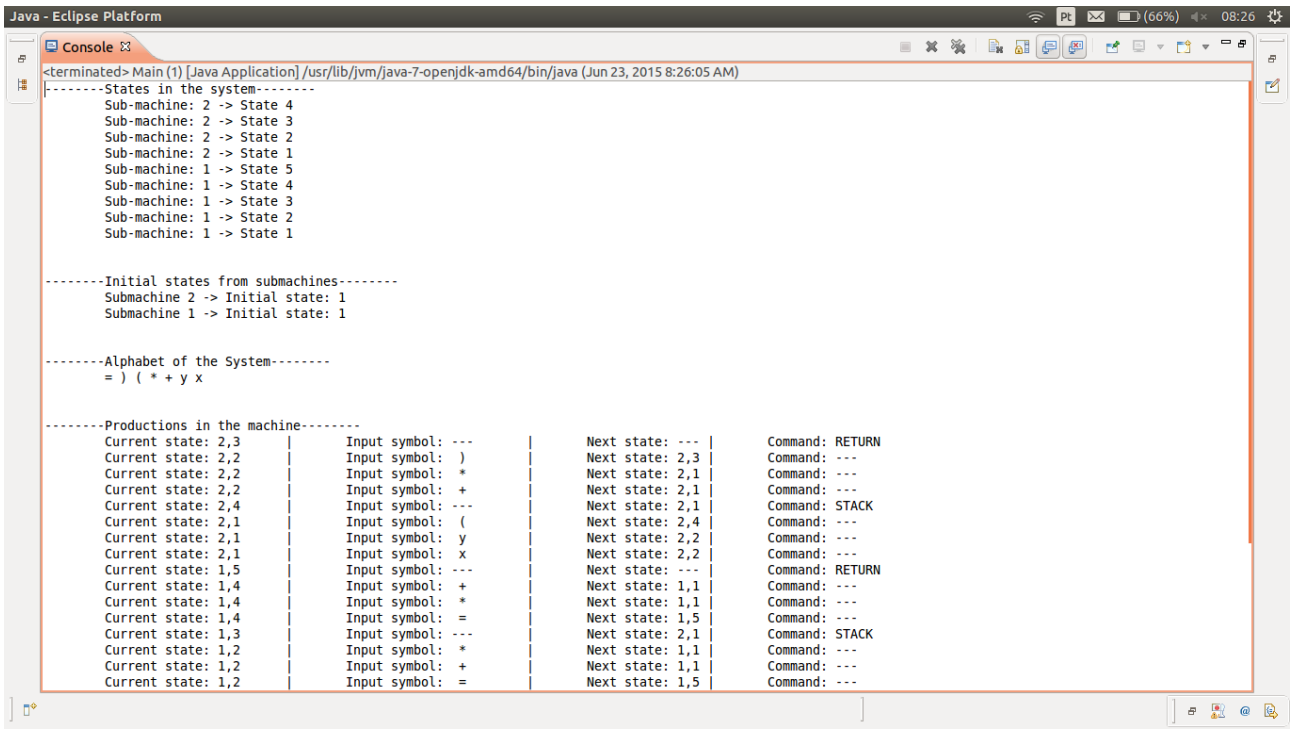
Primeira expressão testada:

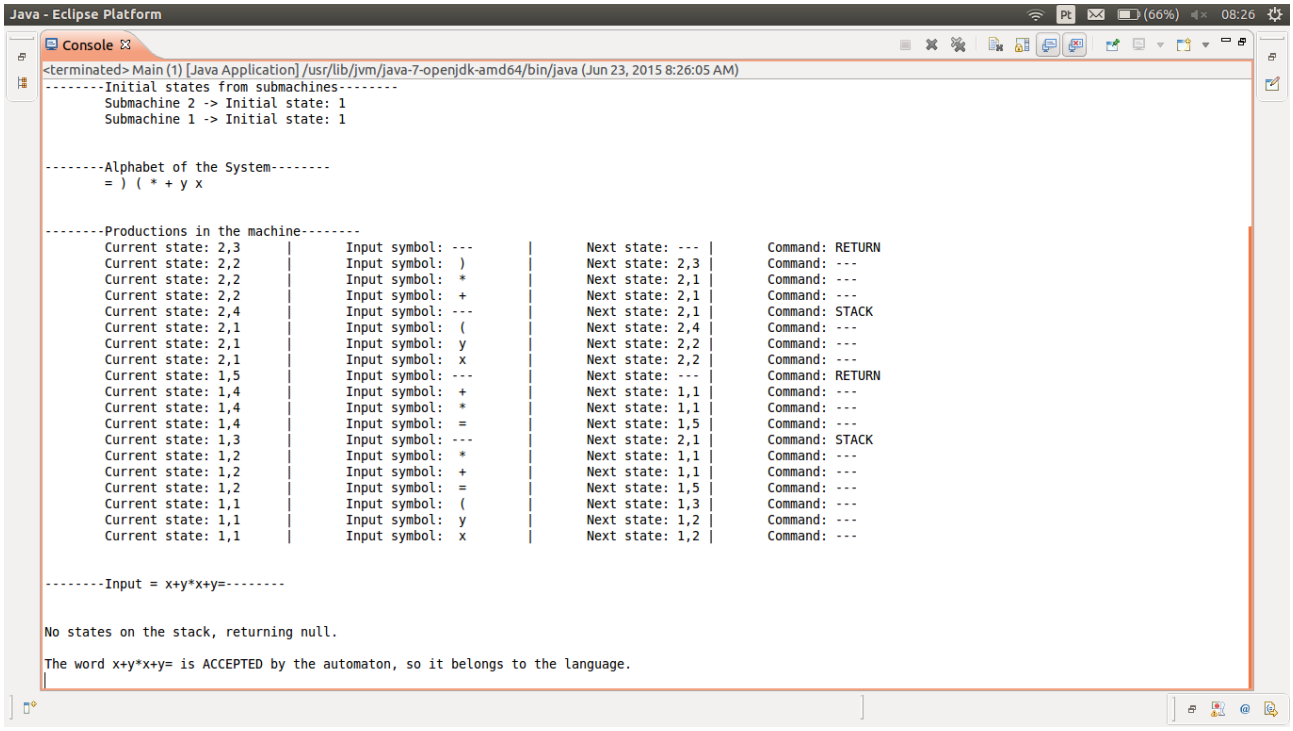
word.txt × machine.txt ×

```
x+y*x+y=
```

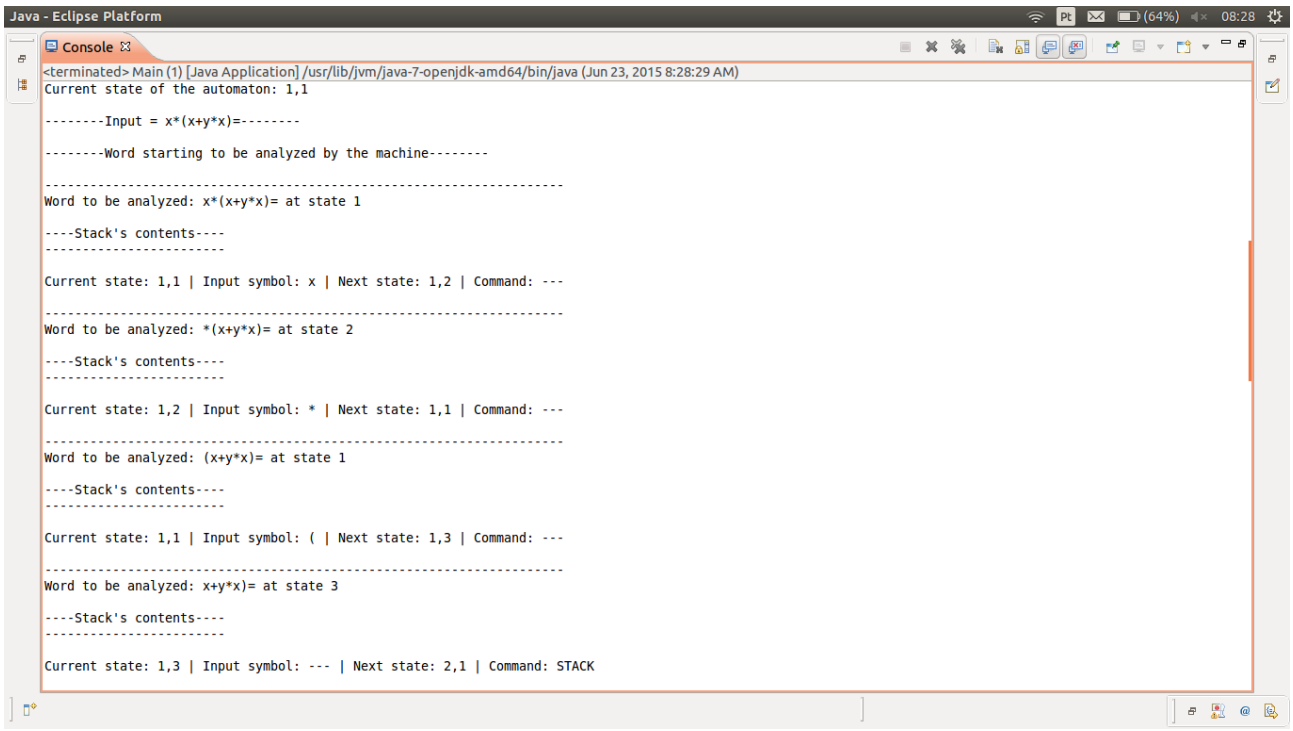
Observar que no caso acima, seleciona-se a opção show_track = 'n' (segunda linha). Portanto, na primeira expressão testada não será mostrado as transições no interior do autômato.

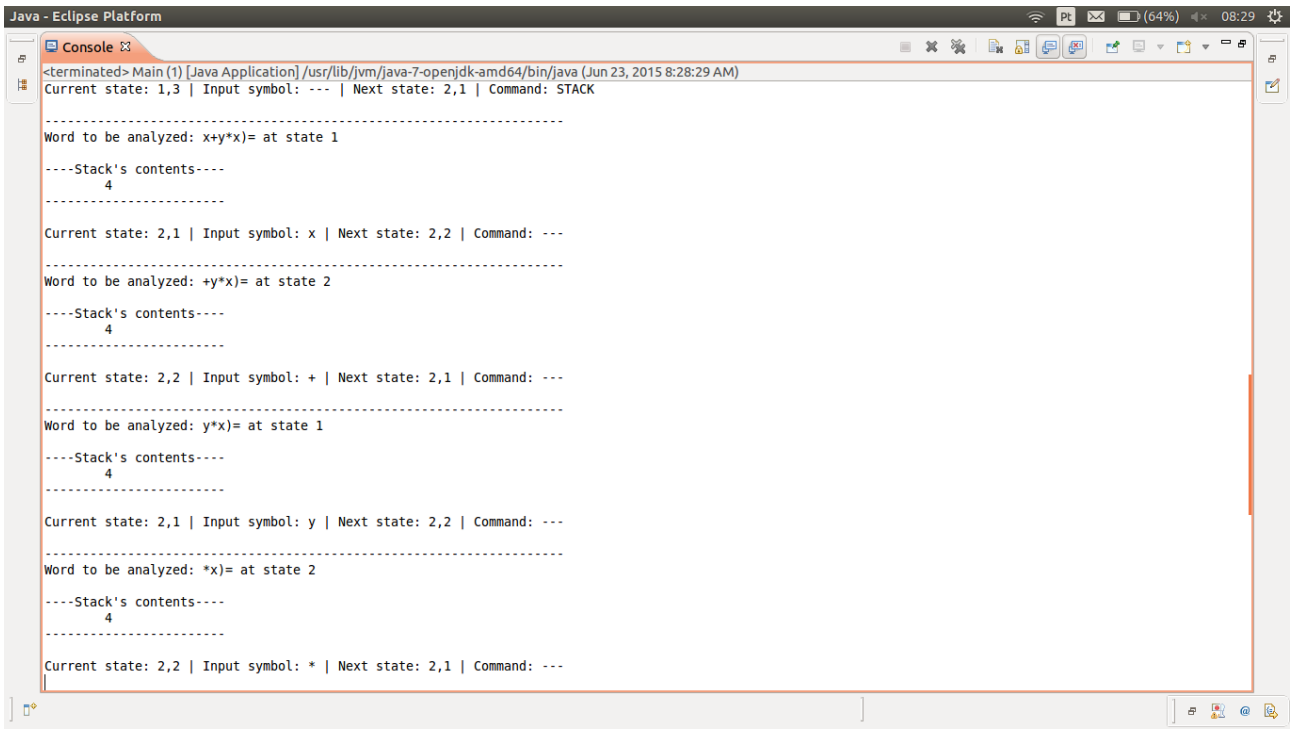
1. x+y*x+y=

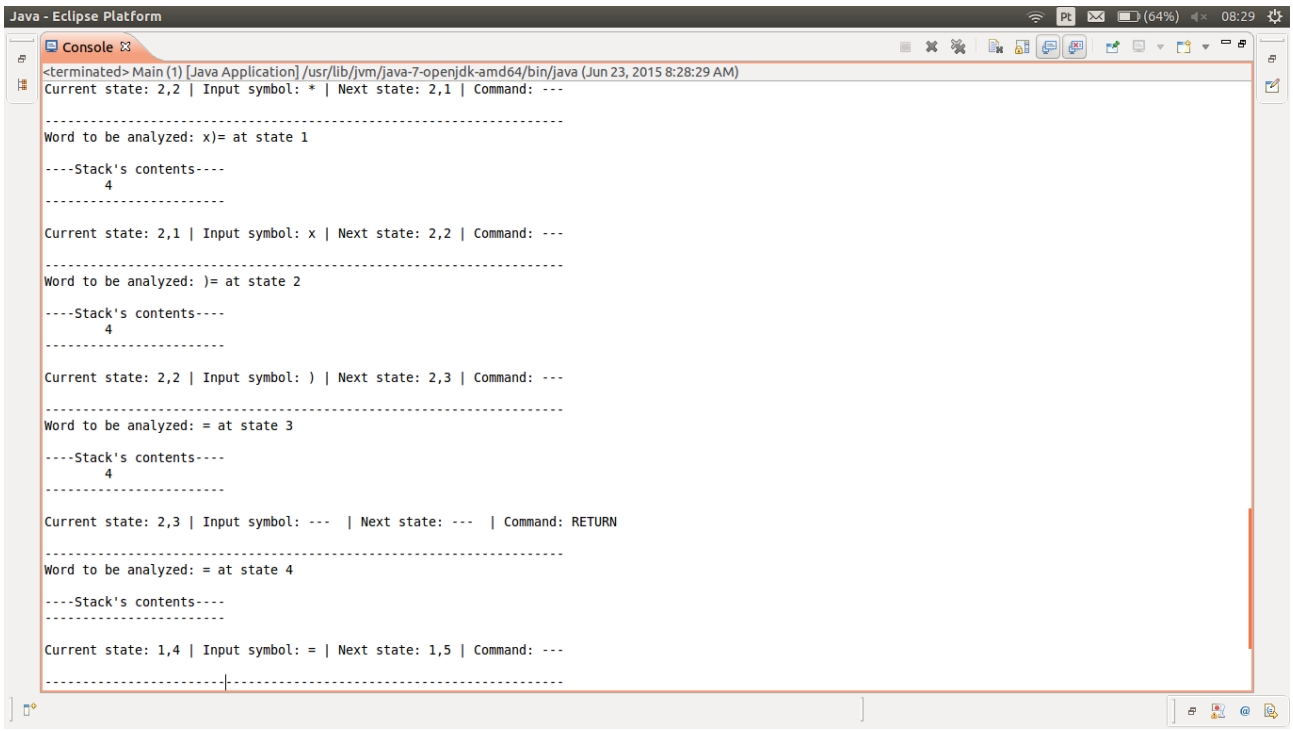


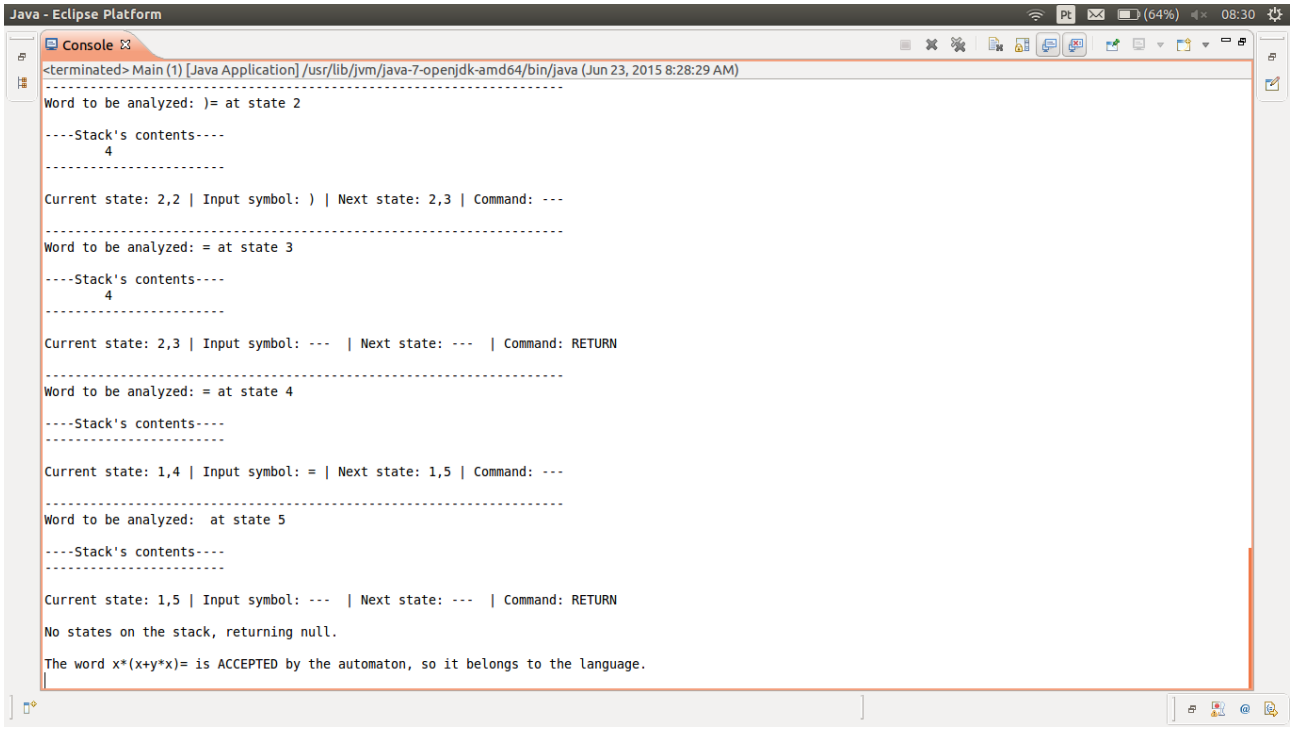


2. x*(x+y*x)=: Agora escolhendo a opção show_track='y'

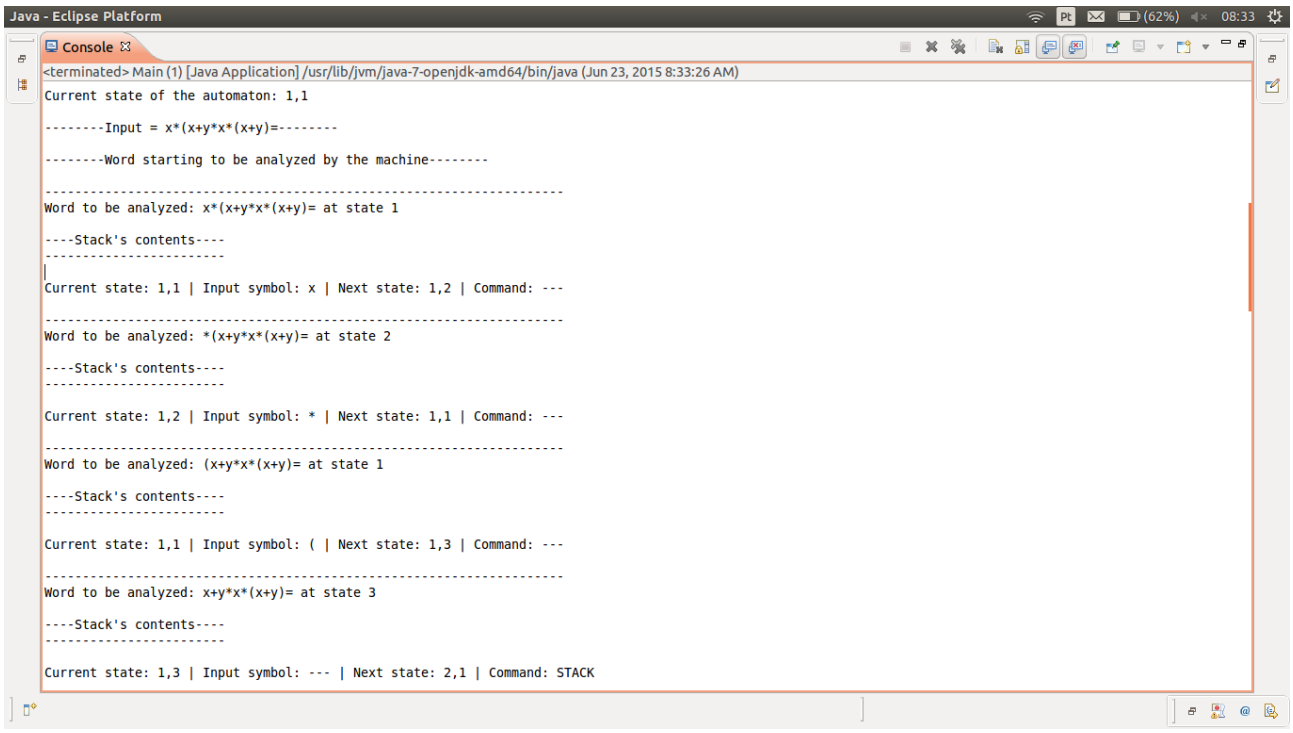

Java - Eclipse Platform
(64%) 08:28
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 8:28:29 AM)
Current state of the automaton: 1,1
--------Input = x*(x+y*x)=--------
--------Word starting to be analyzed by the machine--------
Word to be analyzed: x*(x+y*x)= at state 1
----Stack's contents----
Current state: 1,1 | Input symbol: x | Next state: 1,2 | Command: ---
Word to be analyzed: *(x+y*x)= at state 2
----Stack's contents----
Current state: 1,2 | Input symbol: * | Next state: 1,1 | Command: ---
Word to be analyzed: (x+y*x)= at state 1
----Stack's contents----
Current state: 1,1 | Input symbol: (| Next state: 1,3 | Command: ---
Word to be analyzed: x+y*x)= at state 3
----Stack's contents----
Current state: 1,3 | Input symbol: --- | Next state: 2,1 | Command: STACK

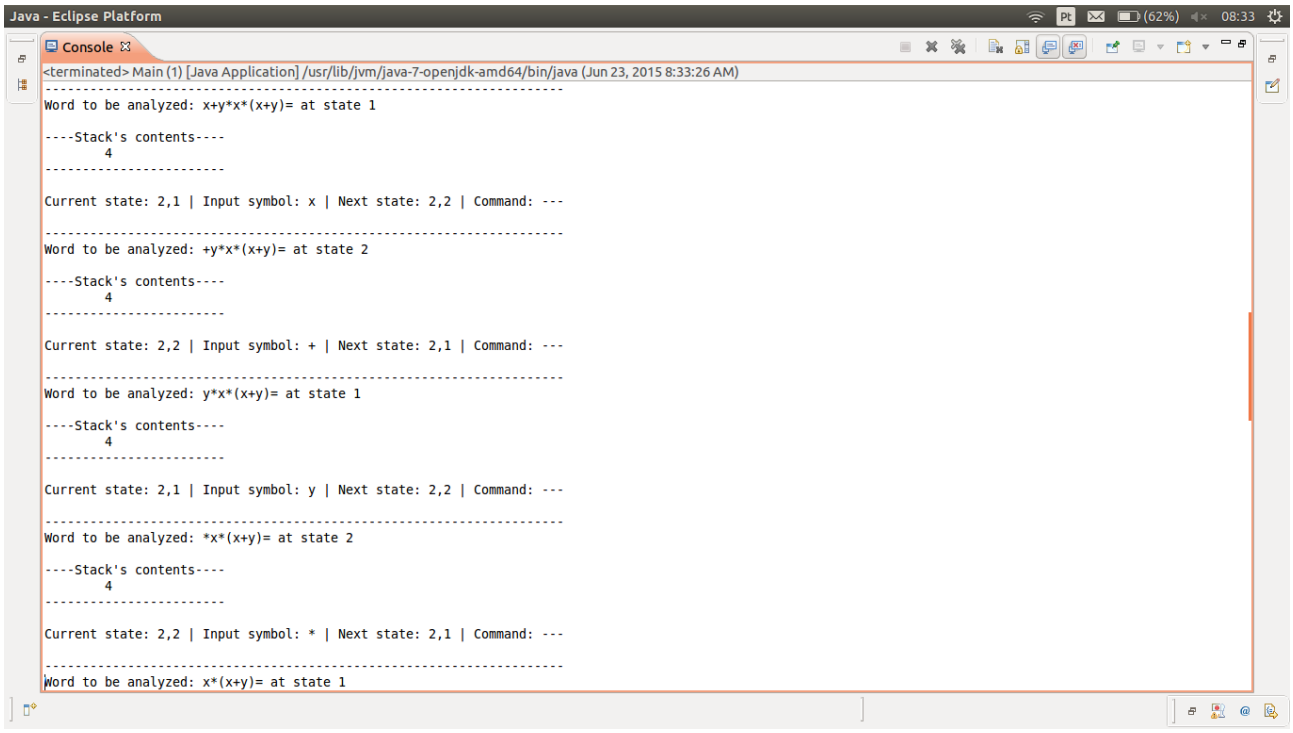

Java - Eclipse Platform
(64%) 08:29
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 8:28:29 AM)
Current state: 1,3 | Input symbol: --- | Next state: 2,1 | Command: STACK
Word to be analyzed: x+y*x)= at state 1
----Stack's contents----
4
Current state: 2,1 | Input symbol: x | Next state: 2,2 | Command: ---
Word to be analyzed: +y*x)= at state 2
----Stack's contents----
4
Current state: 2,2 | Input symbol: + | Next state: 2,1 | Command: ---
Word to be analyzed: y*x)= at state 1
----Stack's contents----
4
Current state: 2,1 | Input symbol: y | Next state: 2,2 | Command: ---
Word to be analyzed: *x)= at state 2
----Stack's contents----
4
Current state: 2,2 | Input symbol: * | Next state: 2,1 | Command: ---

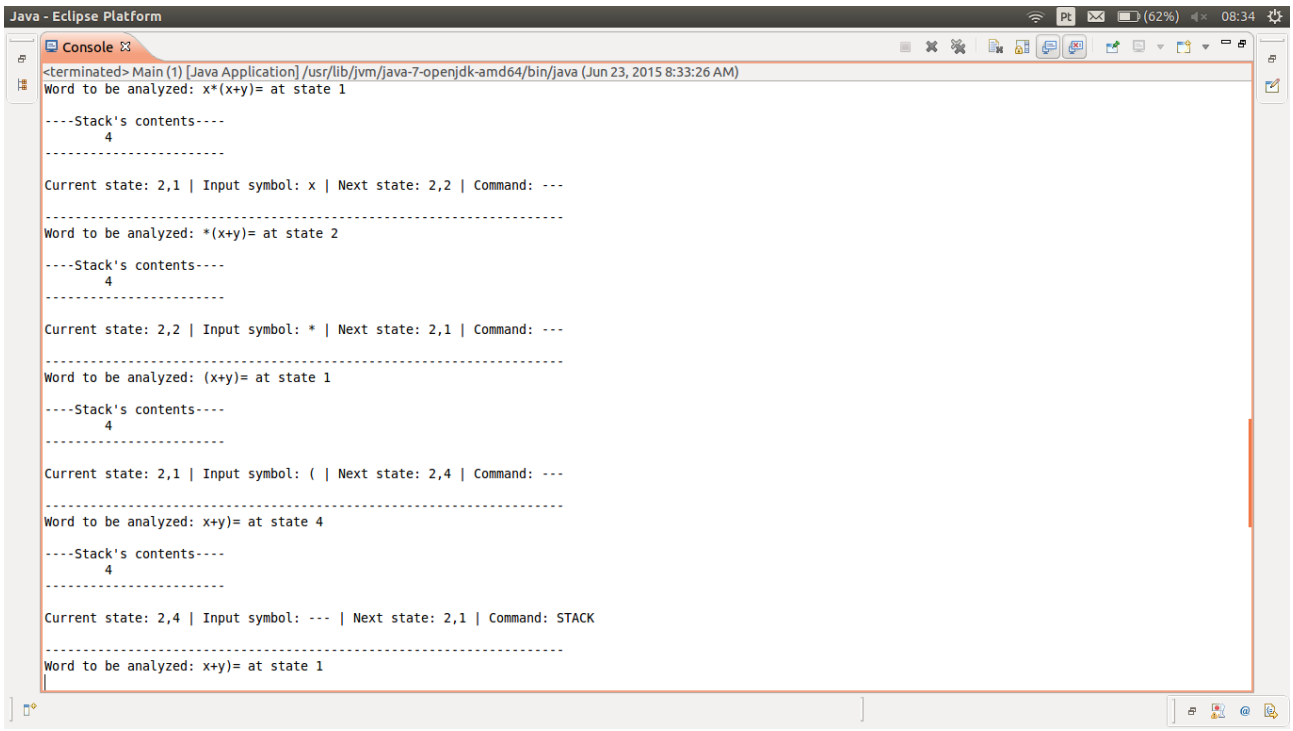


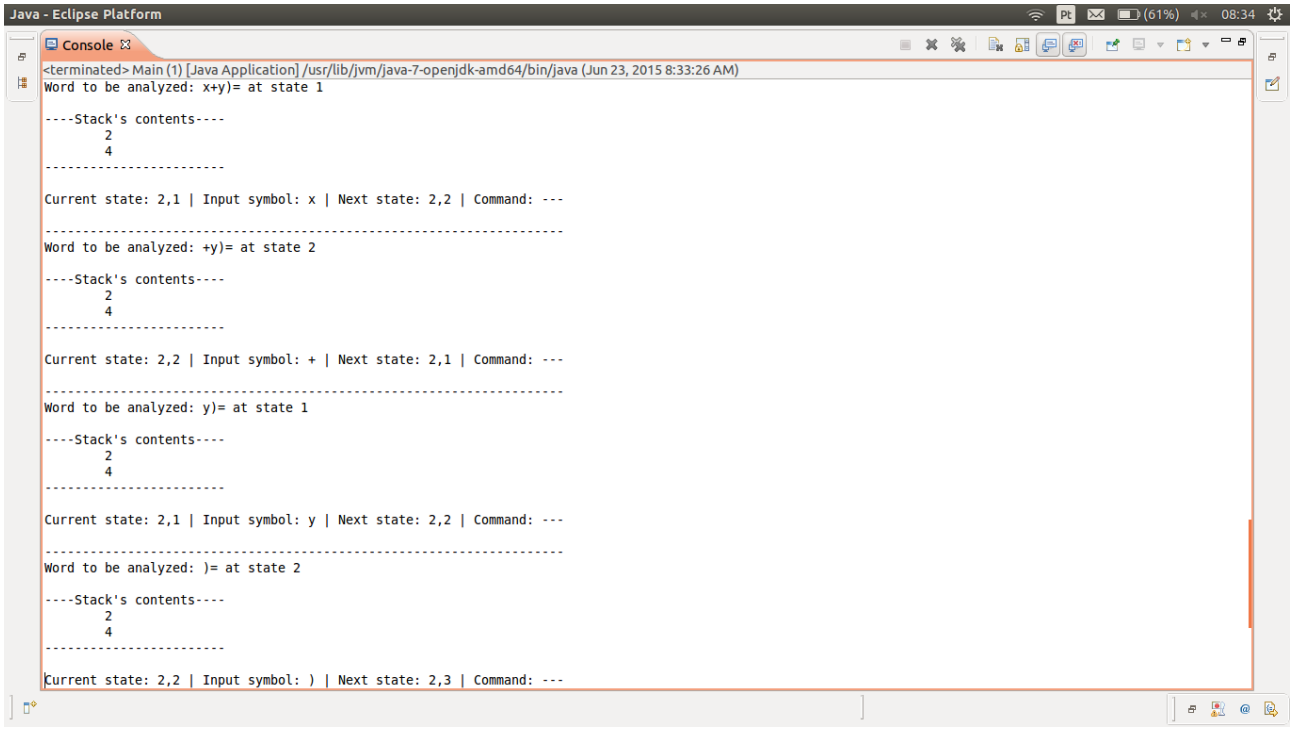


3. x*(x+y*x*(x+y)=: Não aceito, pois o primeiro parênteses não fecha

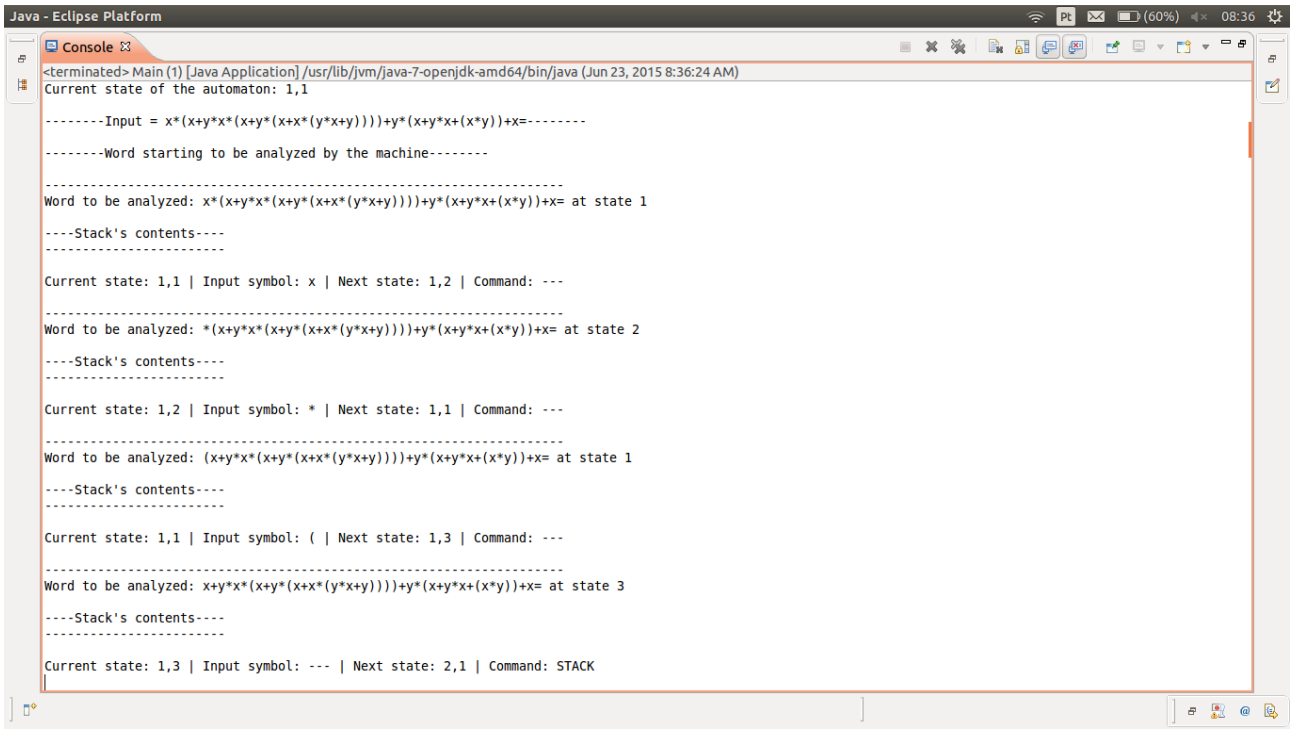

Java - Eclipse Platform
08:33
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 8:33:26 AM)
Current state of the automaton: 1,1
--------Input = x*(x+y*x*(x+y)=--------
--------Word starting to be analyzed by the machine--------
Word to be analyzed: x*(x+y*x*(x+y)= at state 1
----Stack's contents----
Current state: 1,1 | Input symbol: x | Next state: 1,2 | Command: ---
Word to be analyzed: *(x+y*x*(x+y)= at state 2
----Stack's contents----
Current state: 1,2 | Input symbol: * | Next state: 1,1 | Command: ---
Word to be analyzed: (x+y*x*(x+y)= at state 1
----Stack's contents----
Current state: 1,1 | Input symbol: (| Next state: 1,3 | Command: ---
Word to be analyzed: x+y*x*(x+y)= at state 3
----Stack's contents----
Current state: 1,3 | Input symbol: --- | Next state: 2,1 | Command: STACK


Java - Eclipse Platform
08:33
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 8:33:26 AM)
Word to be analyzed: x+y*x*(x+y)= at state 1
----Stack's contents----
4
Current state: 2,1 | Input symbol: x | Next state: 2,2 | Command: ---
Word to be analyzed: +y*x*(x+y)= at state 2
----Stack's contents----
4
Current state: 2,2 | Input symbol: + | Next state: 2,1 | Command: ---
Word to be analyzed: y*x*(x+y)= at state 1
----Stack's contents----
4
Current state: 2,1 | Input symbol: y | Next state: 2,2 | Command: ---
Word to be analyzed: *x*(x+y)= at state 2
----Stack's contents----
4
Current state: 2,2 | Input symbol: * | Next state: 2,1 | Command: ---
Word to be analyzed: x*(x+y)= at state 1


Java - Eclipse Platform
(62%) 08:34
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 8:33:26 AM)
Word to be analyzed: x*(x+y)= at state 1
----Stack's contents----
4
Current state: 2,1 | Input symbol: x | Next state: 2,2 | Command: ---
Word to be analyzed: *(x+y)= at state 2
----Stack's contents----
4
Current state: 2,2 | Input symbol: * | Next state: 2,1 | Command: ---
Word to be analyzed: (x+y)= at state 1
----Stack's contents----
4
Current state: 2,1 | Input symbol: (| Next state: 2,4 | Command: ---
Word to be analyzed: x+y)= at state 4
----Stack's contents----
4
Current state: 2,4 | Input symbol: --- | Next state: 2,1 | Command: STACK
Word to be analyzed: x+y)= at state 1

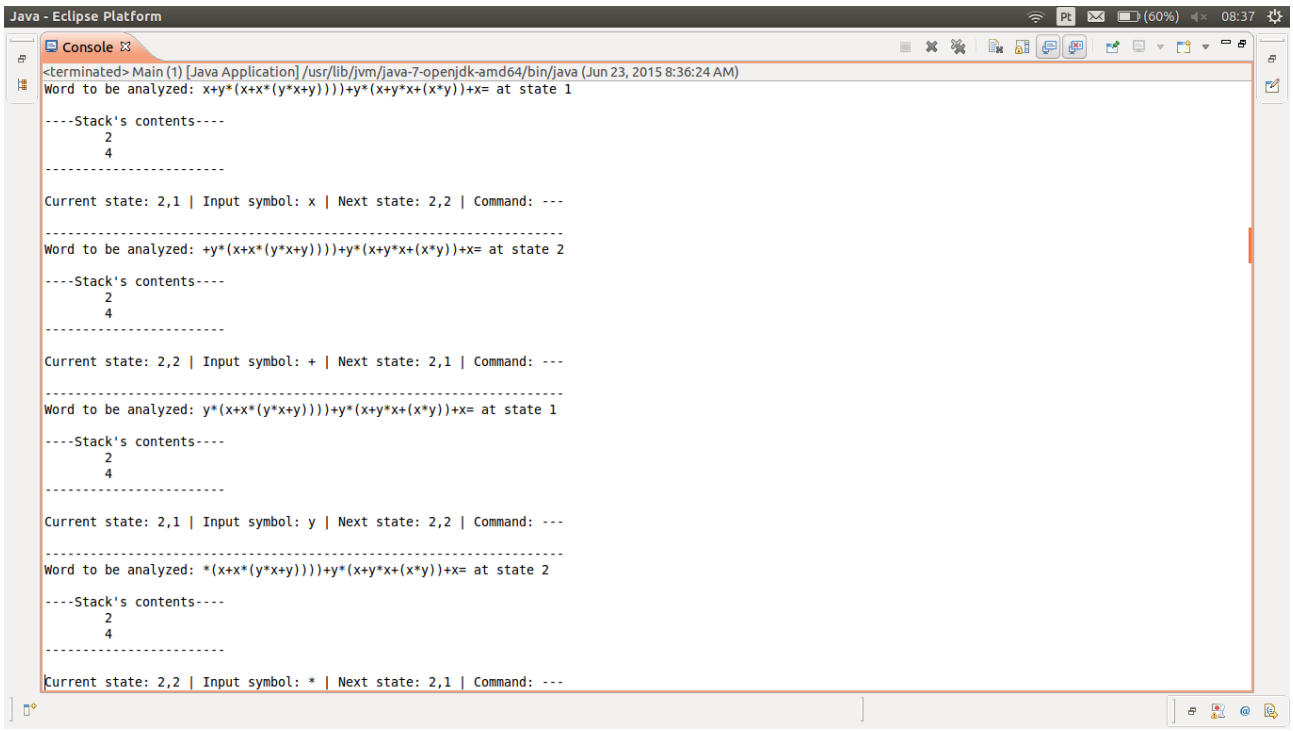

Java - Eclipse Platform
(61%) 08:34
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 8:33:26 AM)
Word to be analyzed: x+y)= at state 1
----Stack's contents----
2
4
Current state: 2,1 | Input symbol: x | Next state: 2,2 | Command: ---
Word to be analyzed: +y)= at state 2
----Stack's contents----
2
4
Current state: 2,2 | Input symbol: + | Next state: 2,1 | Command: ---
Word to be analyzed: y)= at state 1
----Stack's contents----
2
4
Current state: 2,1 | Input symbol: y | Next state: 2,2 | Command: ---
Word to be analyzed:)= at state 2
----Stack's contents----
2
4
Current state: 2,2 | Input symbol:) | Next state: 2,3 | Command: ---

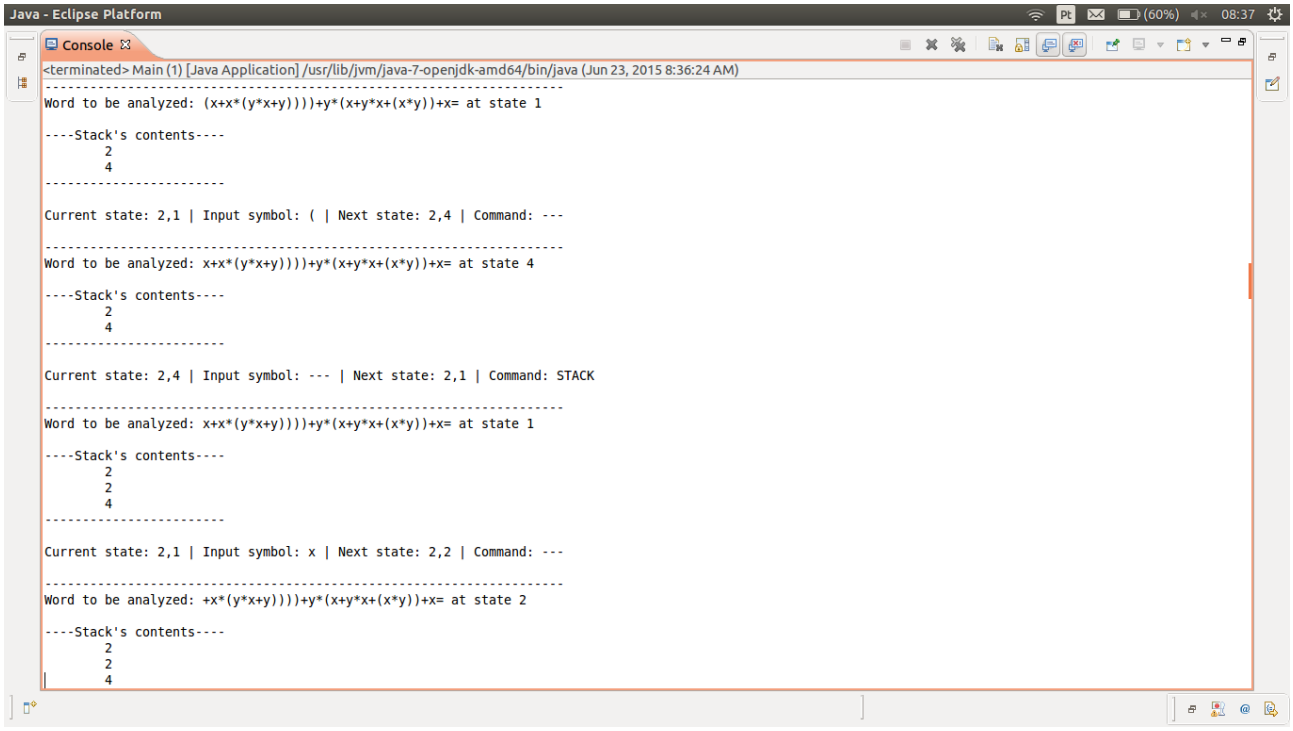
4. x*(x+y*x*(x+y*(x+x*(y*x+y)))+y*(x+y*x+(x*y))+x=: Aceito

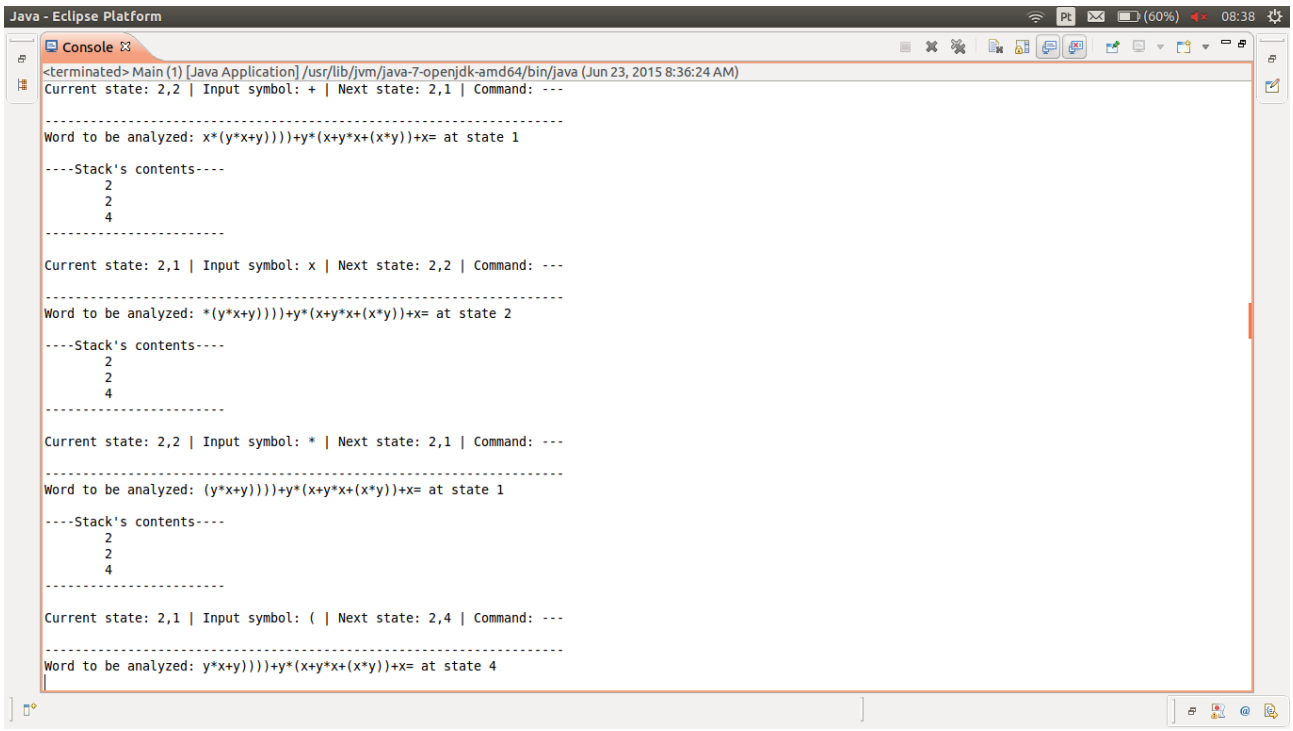


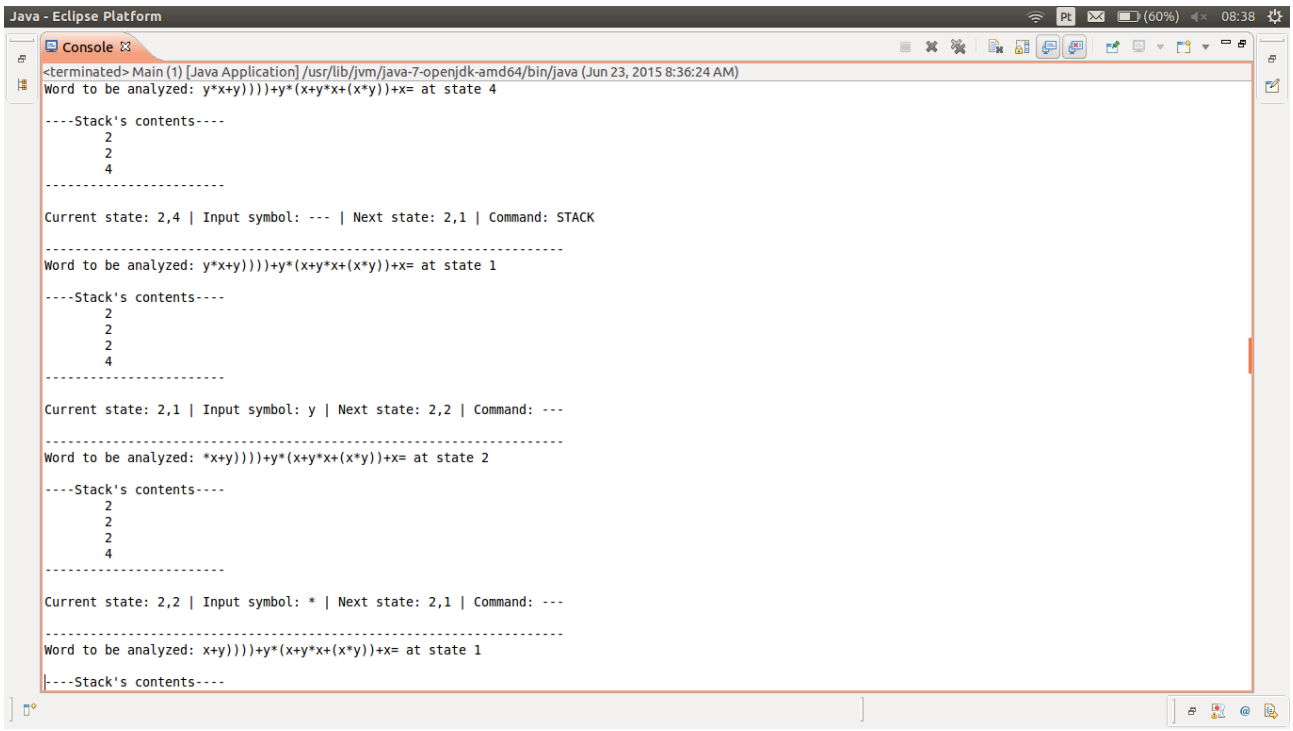

Java - Eclipse Platform
(60%) 08:37
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 8:36:24 AM)
Word to be analyzed: x+y*x*(x+y*(x+x*(y*x+y))))+y*(x+y*x+(x*y))+x= at state 1
----Stack's contents----
4
Current state: 2,1 | Input symbol: x | Next state: 2,2 | Command: ---
Word to be analyzed: +y*x*(x+y*(x+x*(y*x+y))))+y*(x+y*x+(x*y))+x= at state 2
----Stack's contents----
4
Current state: 2,2 | Input symbol: + | Next state: 2,1 | Command: ---
Word to be analyzed: y*x*(x+y*(x+x*(y*x+y))))+y*(x+y*x+(x*y))+x= at state 1
----Stack's contents----
4
Current state: 2,1 | Input symbol: y | Next state: 2,2 | Command: ---
Word to be analyzed: *x*(x+y*(x+x*(y*x+y))))+y*(x+y*x+(x*y))+x= at state 2
----Stack's contents----
4
Current state: 2,2 | Input symbol: * | Next state: 2,1 | Command: ---
Word to be analyzed: x*(x+y*(x+x*(y*x+y))))+y*(x+y*x+(x*y))+x= at state 1

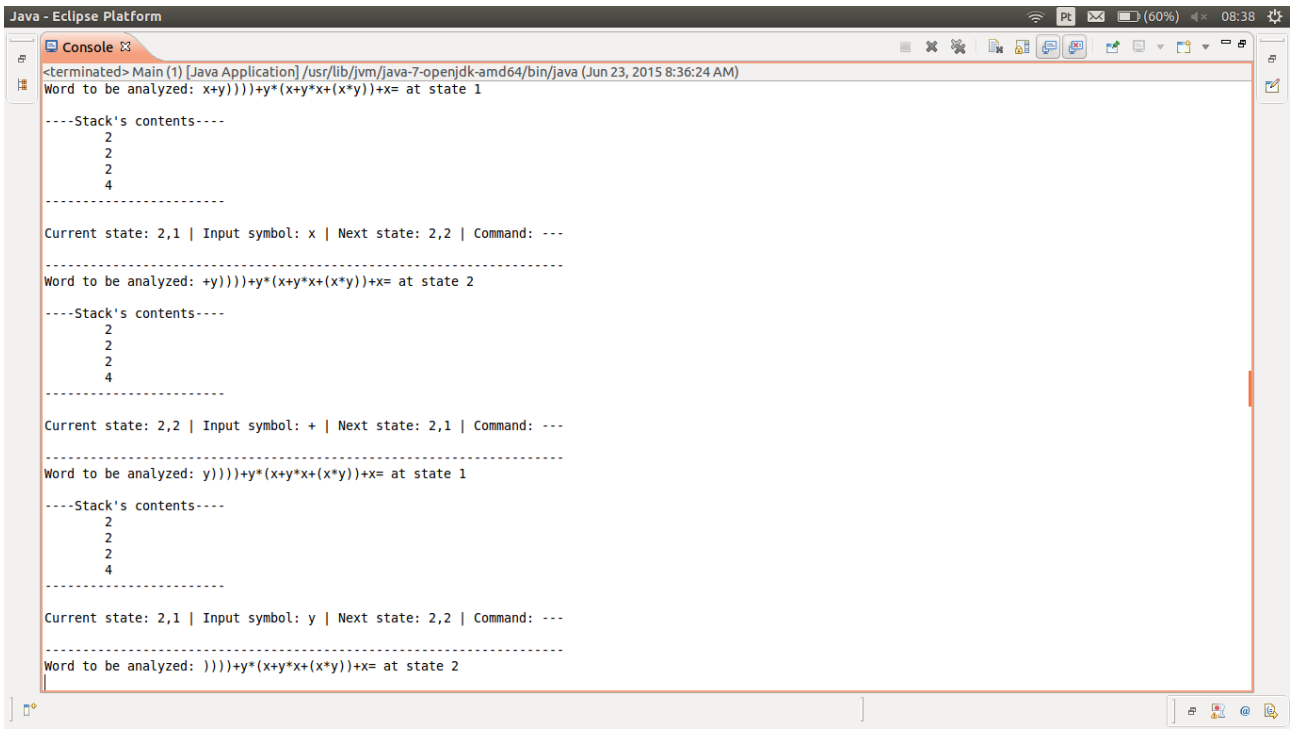

Java - Eclipse Platform
(60%) 08:37
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 8:36:24 AM)
Word to be analyzed: x*(x+y*(x+x*(y*x+y))))+y*(x+y*x+(x*y))+x= at state 1
----Stack's contents----
4
Current state: 2,1 | Input symbol: x | Next state: 2,2 | Command: ---
Word to be analyzed: *(x+y*(x+x*(y*x+y))))+y*(x+y*x+(x*y))+x= at state 2
----Stack's contents----
4
Current state: 2,2 | Input symbol: * | Next state: 2,1 | Command: ---
Word to be analyzed: (x+y*(x+x*(y*x+y))))+y*(x+y*x+(x*y))+x= at state 1
----Stack's contents----
4
Current state: 2,1 | Input symbol: (| Next state: 2,4 | Command: ---
Word to be analyzed: x+y*(x+x*(y*x+y))))+y*(x+y*x+(x*y))+x= at state 4
----Stack's contents----
4
Current state: 2,4 | Input symbol: --- | Next state: 2,1 | Command: STACK
Word to be analyzed: x+y*(x+x*(y*x+y))))+y*(x+y*x+(x*y))+x= at state 1

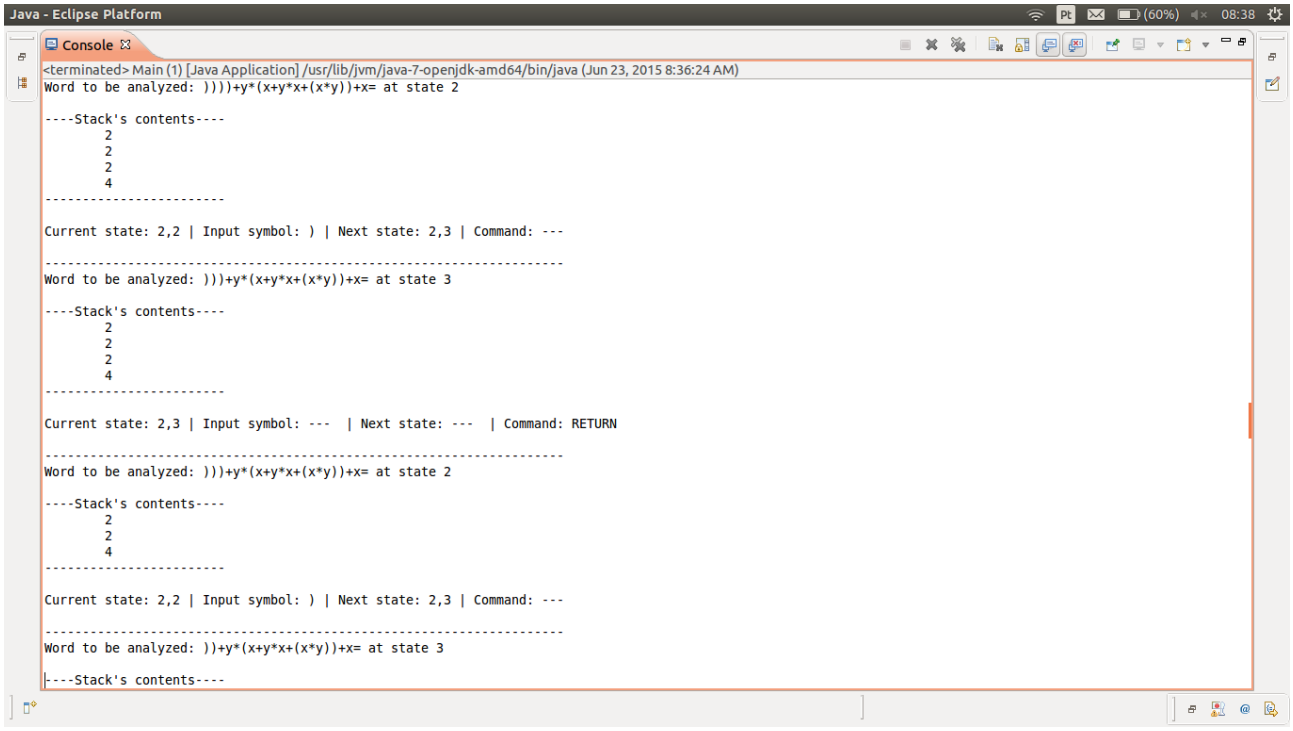

Java - Eclipse Platform
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 8:36:24 AM)
Word to be analyzed: x+y*(x+x*(y*x+y))))+y*(x+y*x+(x*y))+x= at state 1
----Stack's contents----
2
4
Current state: 2,1 | Input symbol: x | Next state: 2,2 | Command: ---
Word to be analyzed: +y*(x+x*(y*x+y))))+y*(x+y*x+(x*y))+x= at state 2
----Stack's contents----
2
4
Current state: 2,2 | Input symbol: + | Next state: 2,1 | Command: ---
Word to be analyzed: y*(x+x*(y*x+y))))+y*(x+y*x+(x*y))+x= at state 1
----Stack's contents----
2
4
Current state: 2,1 | Input symbol: y | Next state: 2,2 | Command: ---
Word to be analyzed: *(x+x*(y*x+y))))+y*(x+y*x+(x*y))+x= at state 2
----Stack's contents----
2
4
Current state: 2,2 | Input symbol: * | Next state: 2,1 | Command: ---

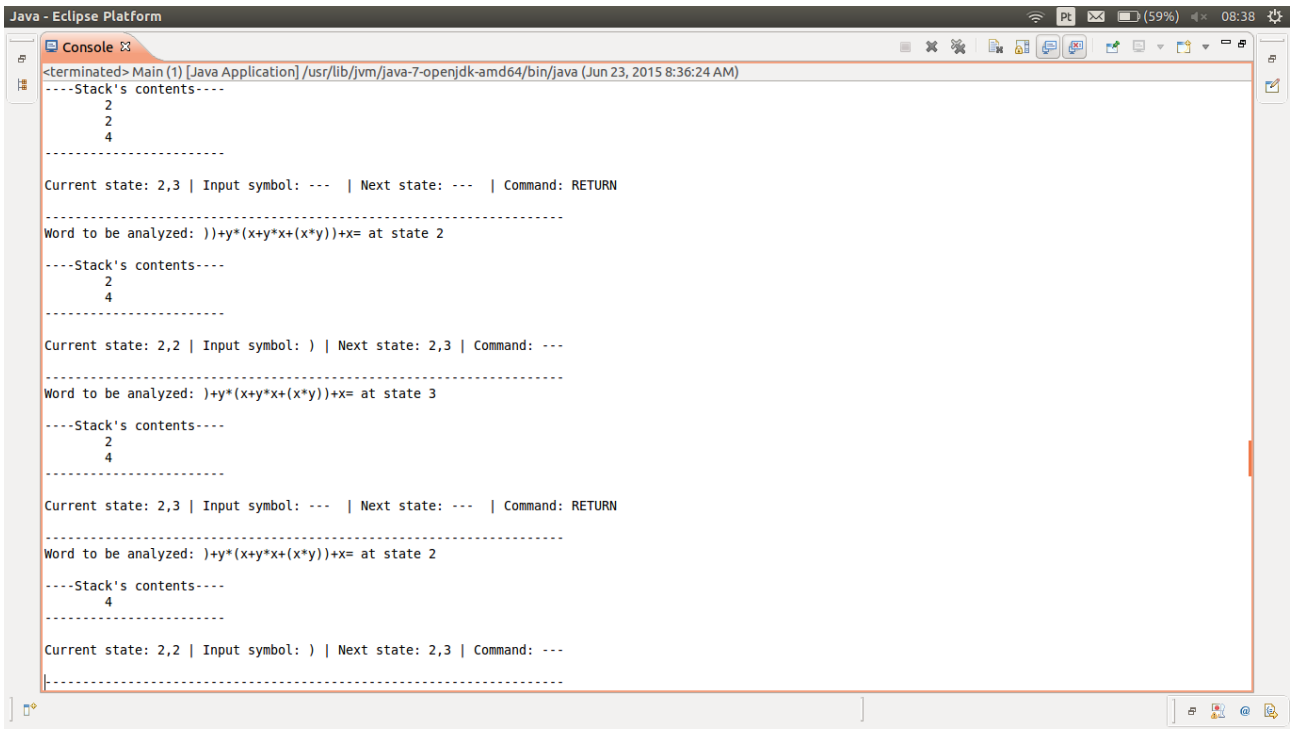

Java - Eclipse Platform
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 8:36:24 AM)
Word to be analyzed: (x+x*(y*x+y))))+y*(x+y*x+(x*y))+x= at state 1
----Stack's contents----
2
4
Current state: 2,1 | Input symbol: (| Next state: 2,4 | Command: ---
Word to be analyzed: x+x*(y*x+y))))+y*(x+y*x+(x*y))+x= at state 4
----Stack's contents----
2
4
Current state: 2,4 | Input symbol: --- | Next state: 2,1 | Command: STACK
Word to be analyzed: x+x*(y*x+y))))+y*(x+y*x+(x*y))+x= at state 1
----Stack's contents----
2
2
4
Current state: 2,1 | Input symbol: x | Next state: 2,2 | Command: ---
Word to be analyzed: +x*(y*x+y))))+y*(x+y*x+(x*y))+x= at state 2
----Stack's contents----
2
2
4

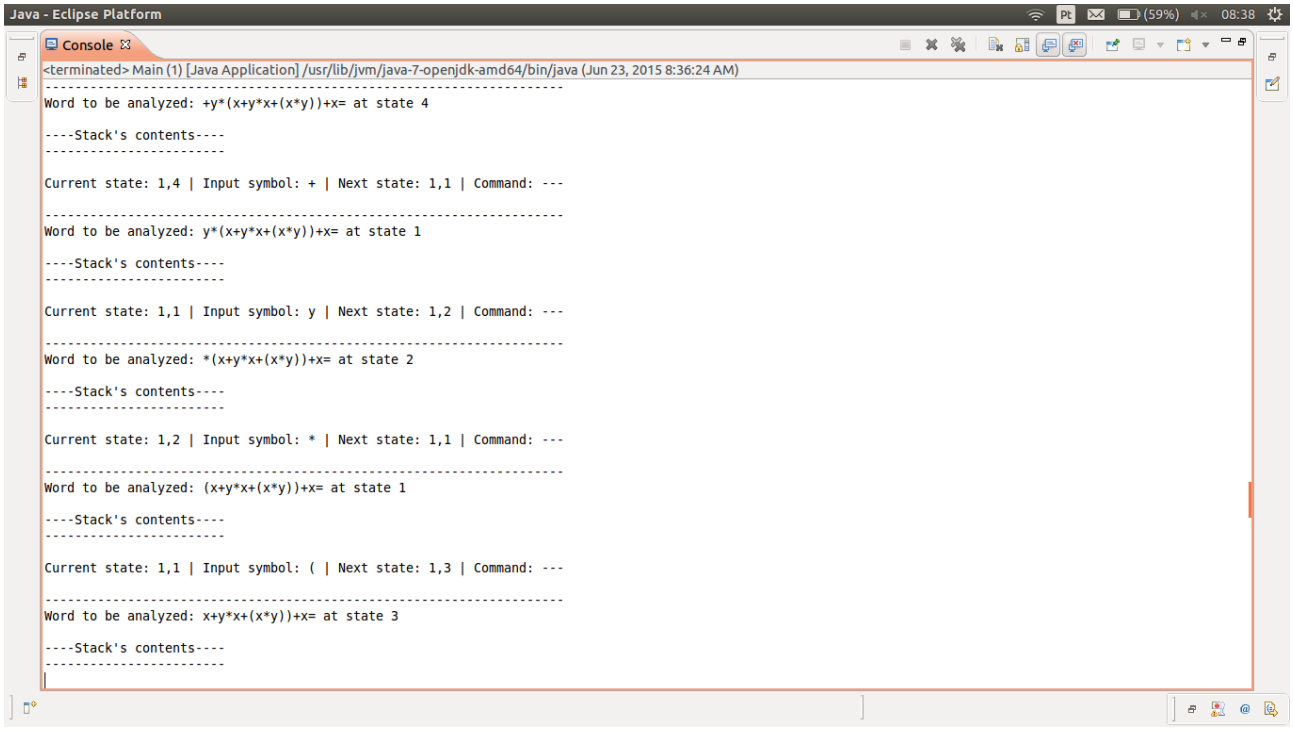

Java - Eclipse Platform
(60%) 08:38
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 8:36:24 AM)
Current state: 2,2 | Input symbol: + | Next state: 2,1 | Command: ---
Word to be analyzed: x*(y*x+y))))+y*(x+y*x+(x*y))+x= at state 1
----Stack's contents----
2
2
4
Current state: 2,1 | Input symbol: x | Next state: 2,2 | Command: ---
Word to be analyzed: *(y*x+y))))+y*(x+y*x+(x*y))+x= at state 2
----Stack's contents----
2
2
4
Current state: 2,2 | Input symbol: * | Next state: 2,1 | Command: ---
Word to be analyzed: (y*x+y))))+y*(x+y*x+(x*y))+x= at state 1
----Stack's contents----
2
2
4
Current state: 2,1 | Input symbol: (| Next state: 2,4 | Command: ---
Word to be analyzed: y*x+y))))+y*(x+y*x+(x*y))+x= at state 4

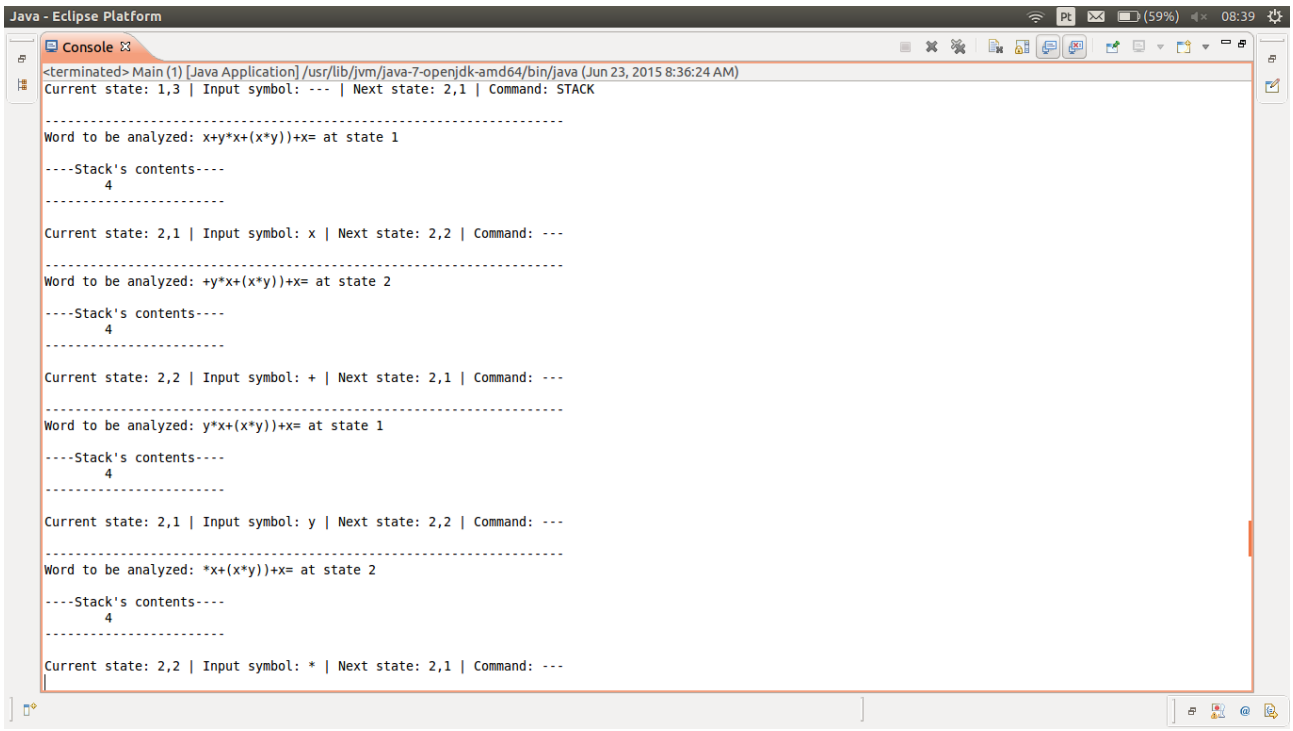

Java - Eclipse Platform
(60%) 08:38
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 8:36:24 AM)
Word to be analyzed: y*x+y))))+y*(x+y*x+(x*y))+x= at state 4
----Stack's contents----
2
2
4
Current state: 2,4 | Input symbol: --- | Next state: 2,1 | Command: STACK
Word to be analyzed: y*x+y))))+y*(x+y*x+(x*y))+x= at state 1
----Stack's contents----
2
2
2
4
Current state: 2,1 | Input symbol: y | Next state: 2,2 | Command: ---
Word to be analyzed: *x+y))))+y*(x+y*x+(x*y))+x= at state 2
----Stack's contents----
2
2
2
4
Current state: 2,2 | Input symbol: * | Next state: 2,1 | Command: ---
Word to be analyzed: x+y))))+y*(x+y*x+(x*y))+x= at state 1
----Stack's contents----


Java - Eclipse Platform
(60%) 08:38
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 8:36:24 AM)
Word to be analyzed: x+y))))+y*(x+y*x+(x*y))+x= at state 1
----Stack's contents----
2
2
2
4
Current state: 2,1 | Input symbol: x | Next state: 2,2 | Command: ---
Word to be analyzed: +y))))+y*(x+y*x+(x*y))+x= at state 2
----Stack's contents----
2
2
2
4
Current state: 2,2 | Input symbol: + | Next state: 2,1 | Command: ---
Word to be analyzed: y))))+y*(x+y*x+(x*y))+x= at state 1
----Stack's contents----
2
2
2
4
Current state: 2,1 | Input symbol: y | Next state: 2,2 | Command: ---
Word to be analyzed:))))+y*(x+y*x+(x*y))+x= at state 2


Java - Eclipse Platform
(60%) 08:38
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 8:36:24 AM)
Word to be analyzed:))))+y*(x+y*x+(x*y))+x= at state 2
----Stack's contents----
2
2
2
4
Current state: 2,2 | Input symbol:) | Next state: 2,3 | Command: ---
Word to be analyzed:)))+y*(x+y*x+(x*y))+x= at state 3
----Stack's contents----
2
2
2
4
Current state: 2,3 | Input symbol: --- | Next state: --- | Command: RETURN
Word to be analyzed:)))+y*(x+y*x+(x*y))+x= at state 2
----Stack's contents----
2
2
4
Current state: 2,2 | Input symbol:) | Next state: 2,3 | Command: ---
Word to be analyzed:))+y*(x+y*x+(x*y))+x= at state 3
----Stack's contents----


Java - Eclipse Platform
(59%) 08:38
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 8:36:24 AM)
----Stack's contents----
2
2
4
Current state: 2,3 | Input symbol: --- | Next state: --- | Command: RETURN
Word to be analyzed:))+y*(x+y*x+(x*y))+x= at state 2
----Stack's contents----
2
4
Current state: 2,2 | Input symbol:) | Next state: 2,3 | Command: ---
Word to be analyzed:)+y*(x+y*x+(x*y))+x= at state 3
----Stack's contents----
2
4
Current state: 2,3 | Input symbol: --- | Next state: --- | Command: RETURN
Word to be analyzed:)+y*(x+y*x+(x*y))+x= at state 2
----Stack's contents----
4
Current state: 2,2 | Input symbol:) | Next state: 2,3 | Command: ---


Java - Eclipse Platform
(59%) 08:38
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 8:36:24 AM)
Word to be analyzed: +y*(x+y*x+(x*y))+x= at state 4
----Stack's contents----
Current state: 1,4 | Input symbol: + | Next state: 1,1 | Command: ---
Word to be analyzed: y*(x+y*x+(x*y))+x= at state 1
----Stack's contents----
Current state: 1,1 | Input symbol: y | Next state: 1,2 | Command: ---
Word to be analyzed: *(x+y*x+(x*y))+x= at state 2
----Stack's contents----
Current state: 1,2 | Input symbol: * | Next state: 1,1 | Command: ---
Word to be analyzed: (x+y*x+(x*y))+x= at state 1
----Stack's contents----
Current state: 1,1 | Input symbol: (| Next state: 1,3 | Command: ---
Word to be analyzed: x+y*x+(x*y))+x= at state 3
----Stack's contents----


Java - Eclipse Platform
(59%) 08:39
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 8:36:24 AM)
Current state: 1,3 | Input symbol: --- | Next state: 2,1 | Command: STACK

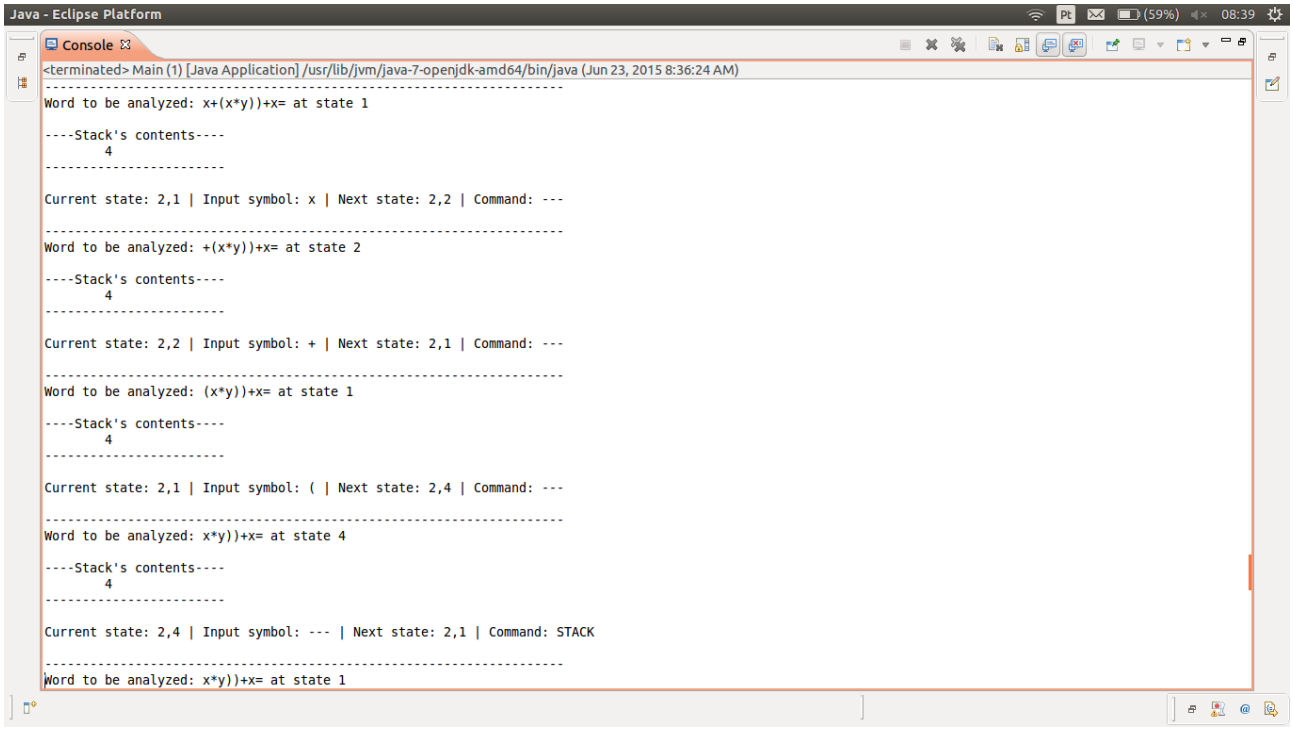
Word to be analyzed: x+y*x+(x*y))+x= at state 1
----Stack's contents----
4

Current state: 2,1 | Input symbol: x | Next state: 2,2 | Command: ---
---
Word to be analyzed: +y*x+(x*y))+x= at state 2
----Stack's contents----
4

Current state: 2,2 | Input symbol: + | Next state: 2,1 | Command: ---
---
Word to be analyzed: y*x+(x*y))+x= at state 1
----Stack's contents----
4

Current state: 2,1 | Input symbol: y | Next state: 2,2 | Command: ---
---
Word to be analyzed: *x+(x*y))+x= at state 2
----Stack's contents----
4

Current state: 2,2 | Input symbol: * | Next state: 2,1 | Command: ---


Java - Eclipse Platform
(59%) 08:39
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 8:36:24 AM)

Word to be analyzed: x+(x*y))+x= at state 1
----Stack's contents----
4

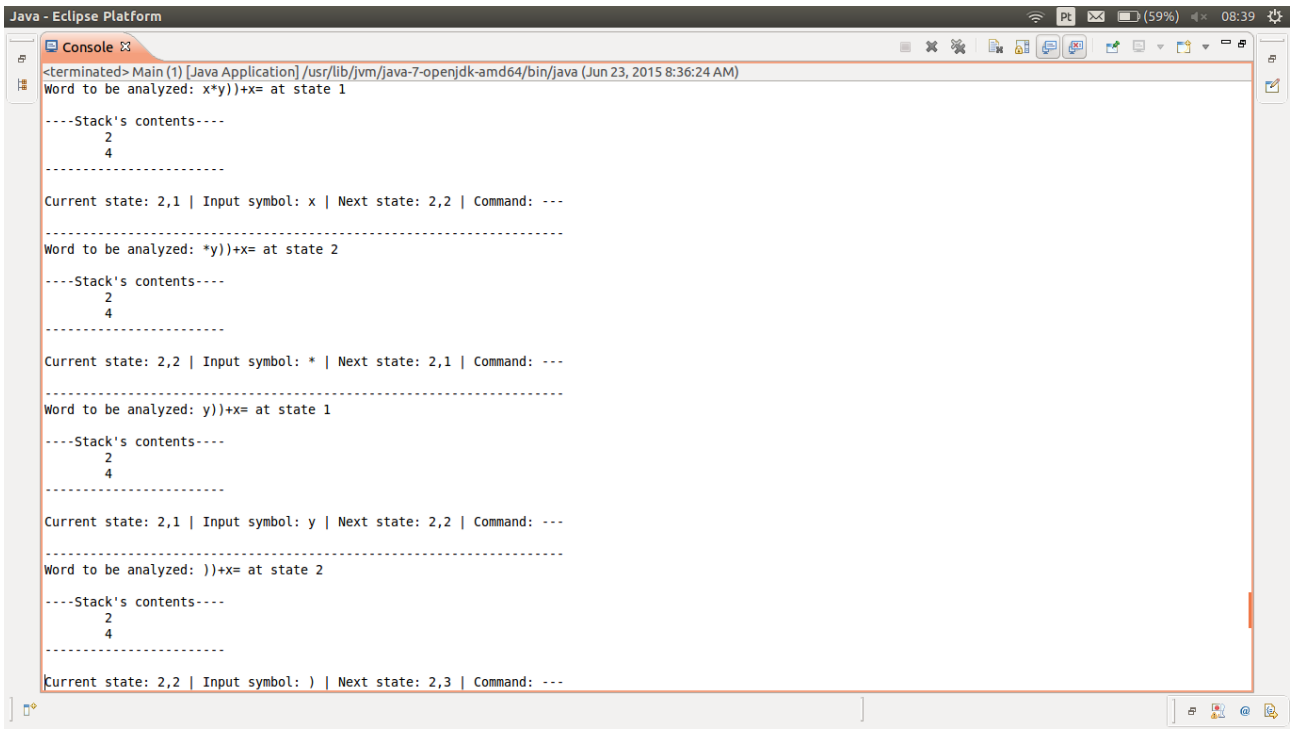
Current state: 2,1 | Input symbol: x | Next state: 2,2 | Command: ---
---
Word to be analyzed: +(x*y))+x= at state 2
----Stack's contents----
4

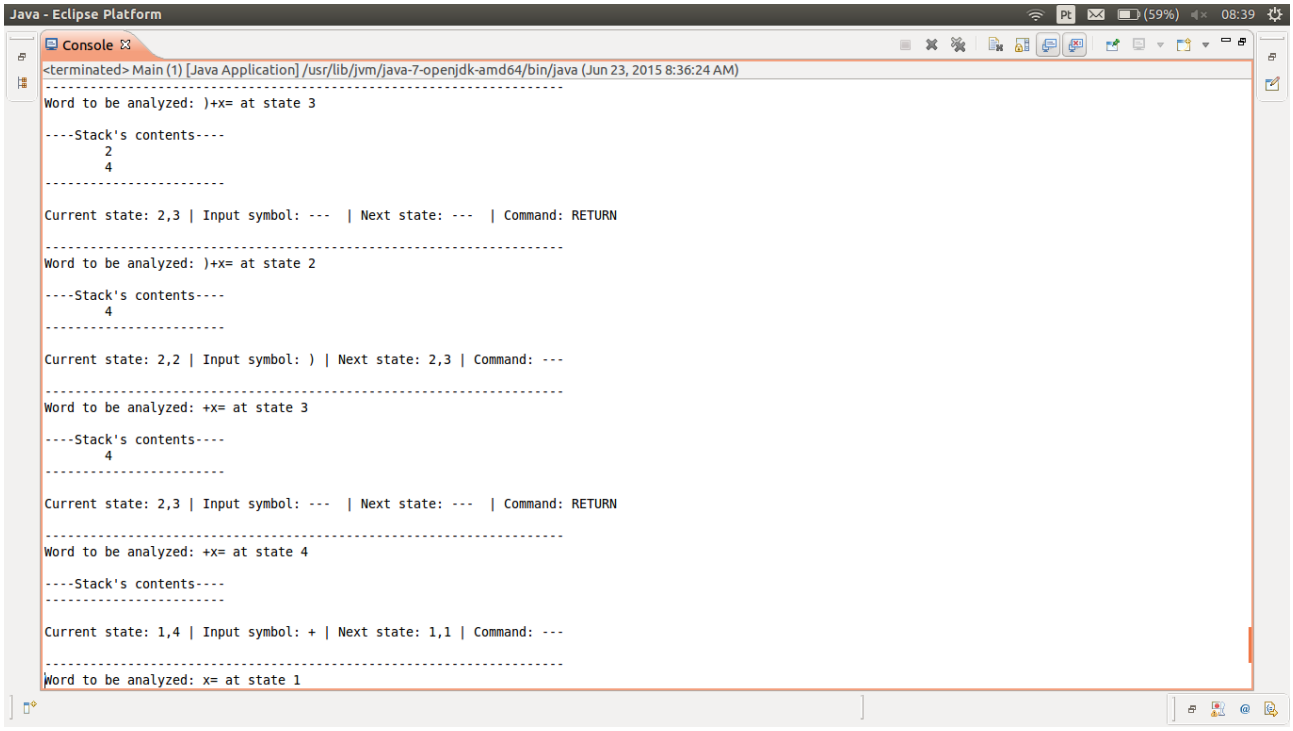
Current state: 2,2 | Input symbol: + | Next state: 2,1 | Command: ---
---
Word to be analyzed: (x*y))+x= at state 1
----Stack's contents----
4

Current state: 2,1 | Input symbol: (| Next state: 2,4 | Command: ---
---
Word to be analyzed: x*y))+x= at state 4
----Stack's contents----
4

Current state: 2,4 | Input symbol: --- | Next state: 2,1 | Command: STACK

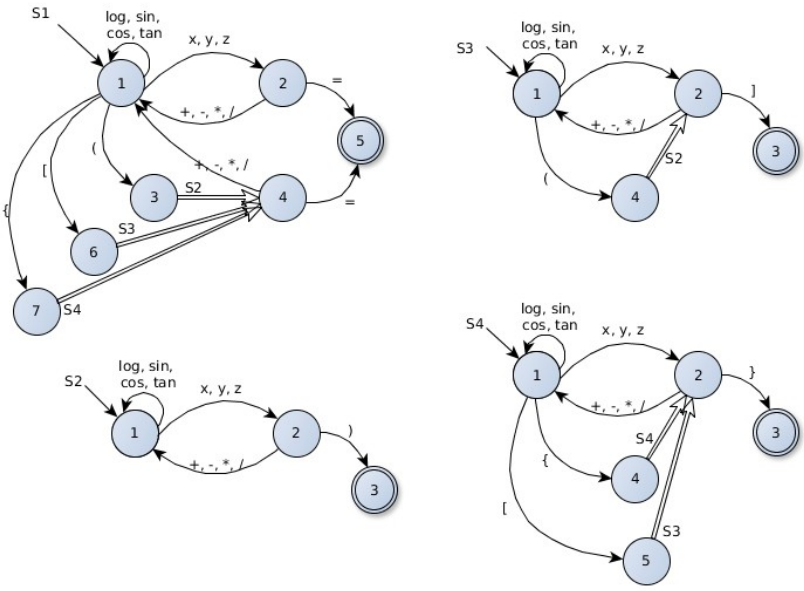
Word to be analyzed: x*y))+x= at state 1


Java - Eclipse Platform
(59%) 08:39
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 8:36:24 AM)
Word to be analyzed: x*y))+x= at state 1
----Stack's contents----
2
4
Current state: 2,1 | Input symbol: x | Next state: 2,2 | Command: ---
Word to be analyzed: *y))+x= at state 2
----Stack's contents----
2
4
Current state: 2,2 | Input symbol: * | Next state: 2,1 | Command: ---
Word to be analyzed: y))+x= at state 1
----Stack's contents----
2
4
Current state: 2,1 | Input symbol: y | Next state: 2,2 | Command: ---
Word to be analyzed:))+x= at state 2
----Stack's contents----
2
4
Current state: 2,2 | Input symbol:) | Next state: 2,3 | Command: ---

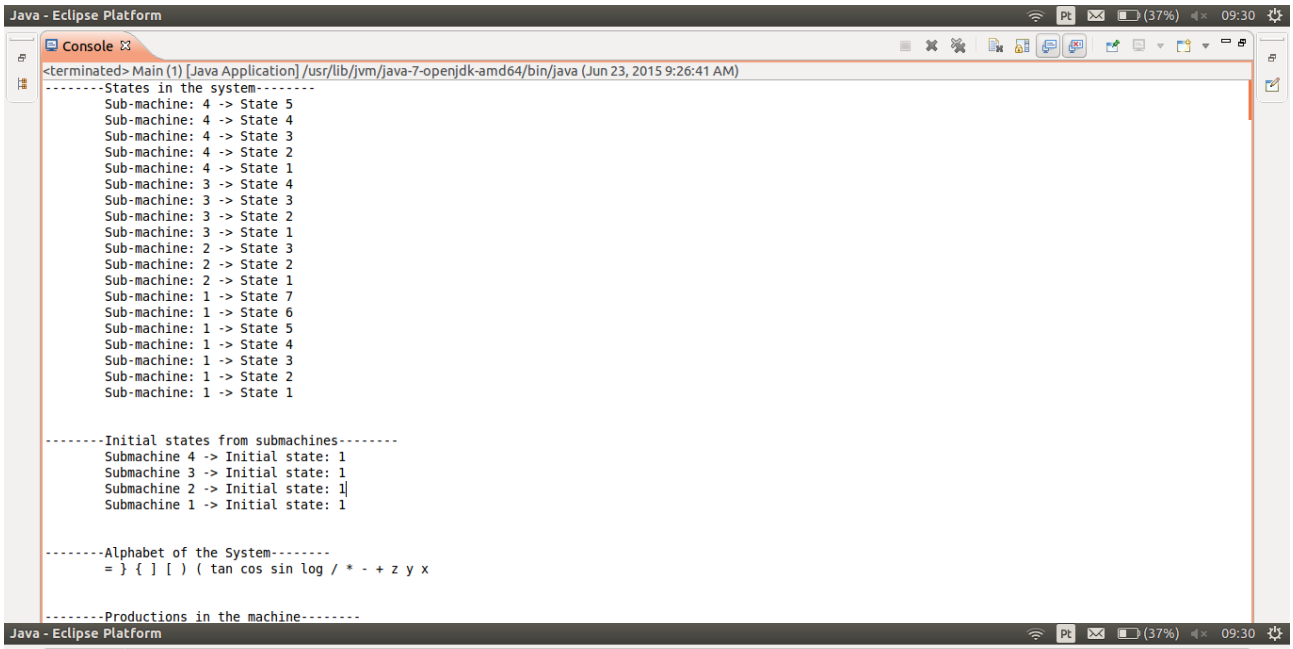

Java - Eclipse Platform
(59%) 08:39
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 8:36:24 AM)
Word to be analyzed:)+x= at state 3
----Stack's contents----
2
4
Current state: 2,3 | Input symbol: --- | Next state: --- | Command: RETURN
Word to be analyzed:)+x= at state 2
----Stack's contents----
4
Current state: 2,2 | Input symbol:) | Next state: 2,3 | Command: ---
Word to be analyzed: +x= at state 3
----Stack's contents----
4
Current state: 2,3 | Input symbol: --- | Next state: --- | Command: RETURN
Word to be analyzed: +x= at state 4
----Stack's contents----
Current state: 1,4 | Input symbol: + | Next state: 1,1 | Command: ---
Word to be analyzed: x= at state 1

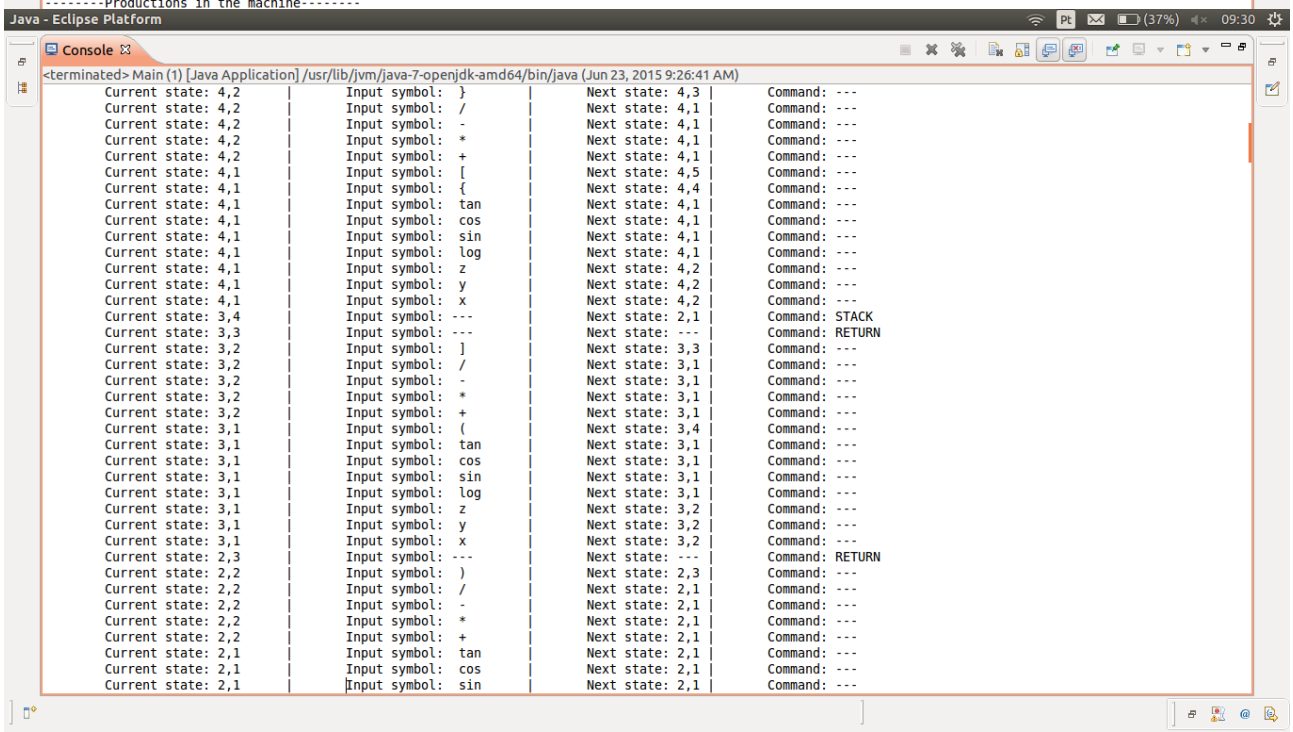
Teste 3

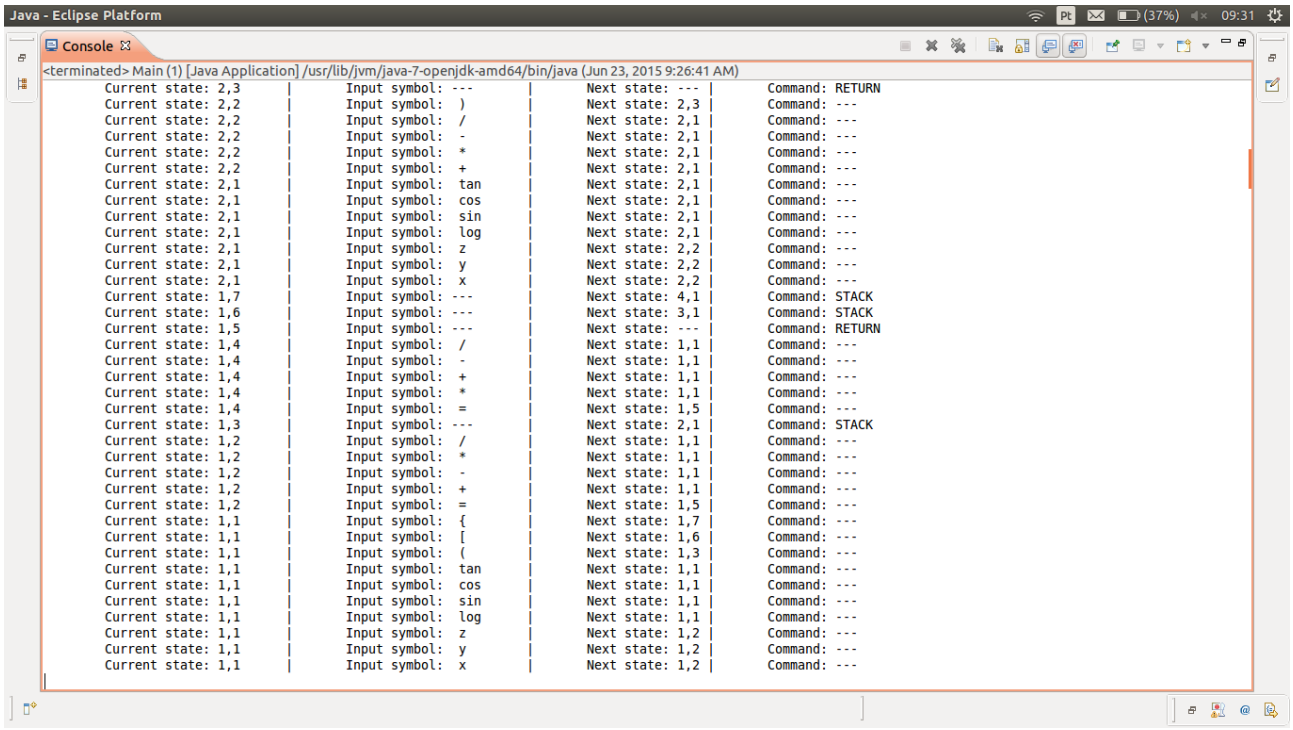
O autômato seguinte analisa expressões matemáticas mais complexas. Ele permite o uso de chaves e colchetes além dos parênteses, segundo a ordem estabelecida pelas regras matemáticas: parênteses são os mais internos, depois colchetes e chaves os mais externos e quantas vezes forem necessárias. Além disso, a expressão pode conter operações logarítmicas e trigonométricas. O alfabeto é {x, y, z, +, -, *, /, log, sin, cos, tan, (,), [,], {, }}.

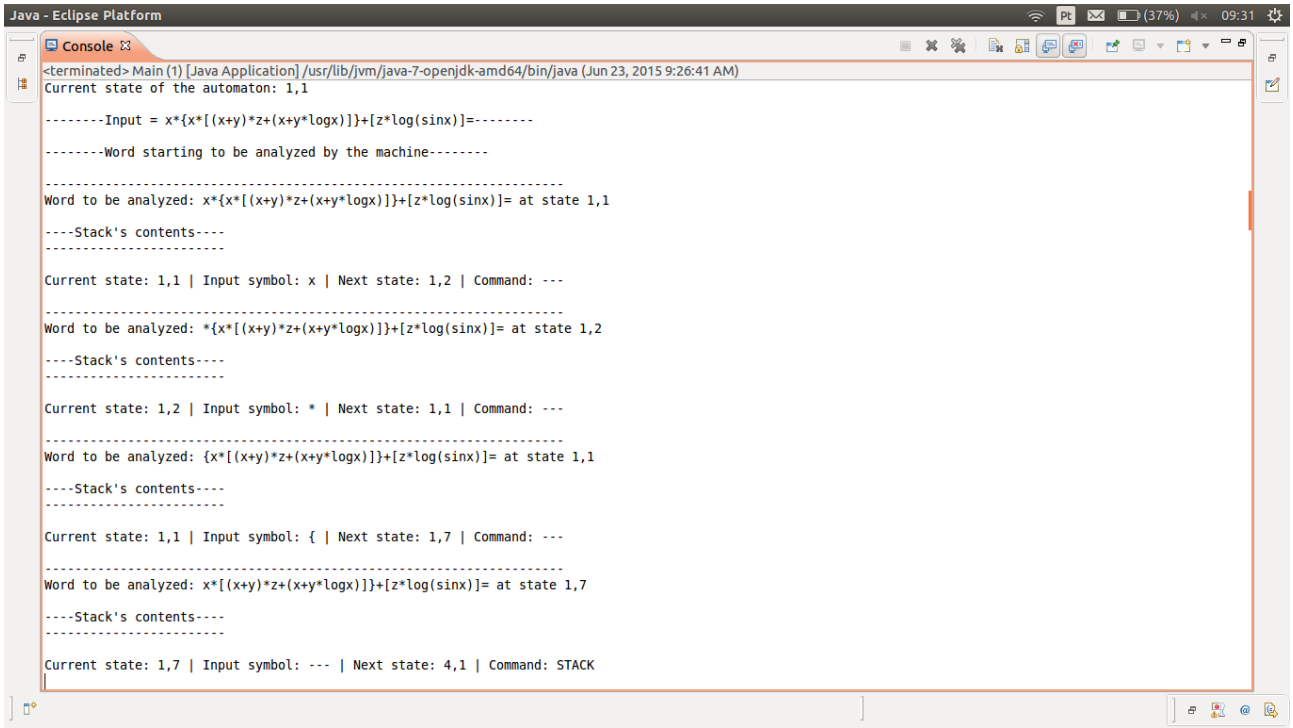


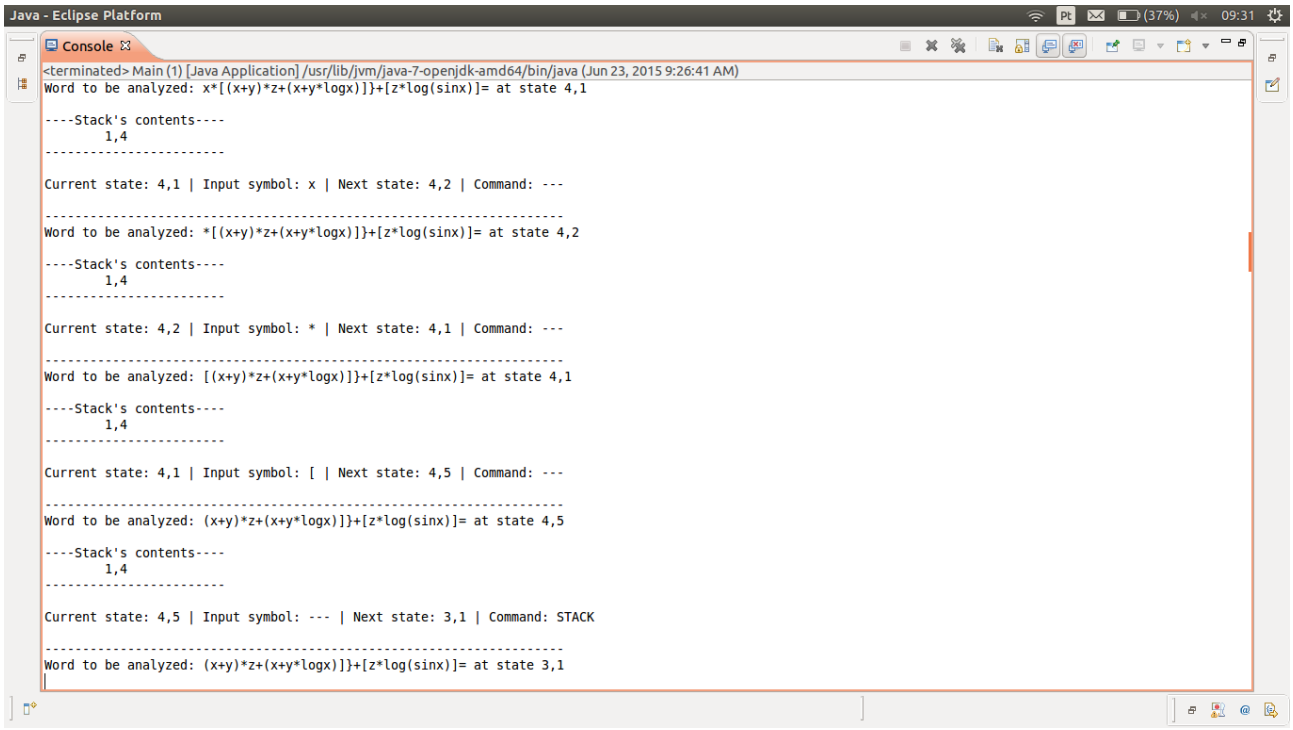
1. x*{x*[(x+y)*z+(x+y*logx)]}+[z*log(sinx)]=: Aceito



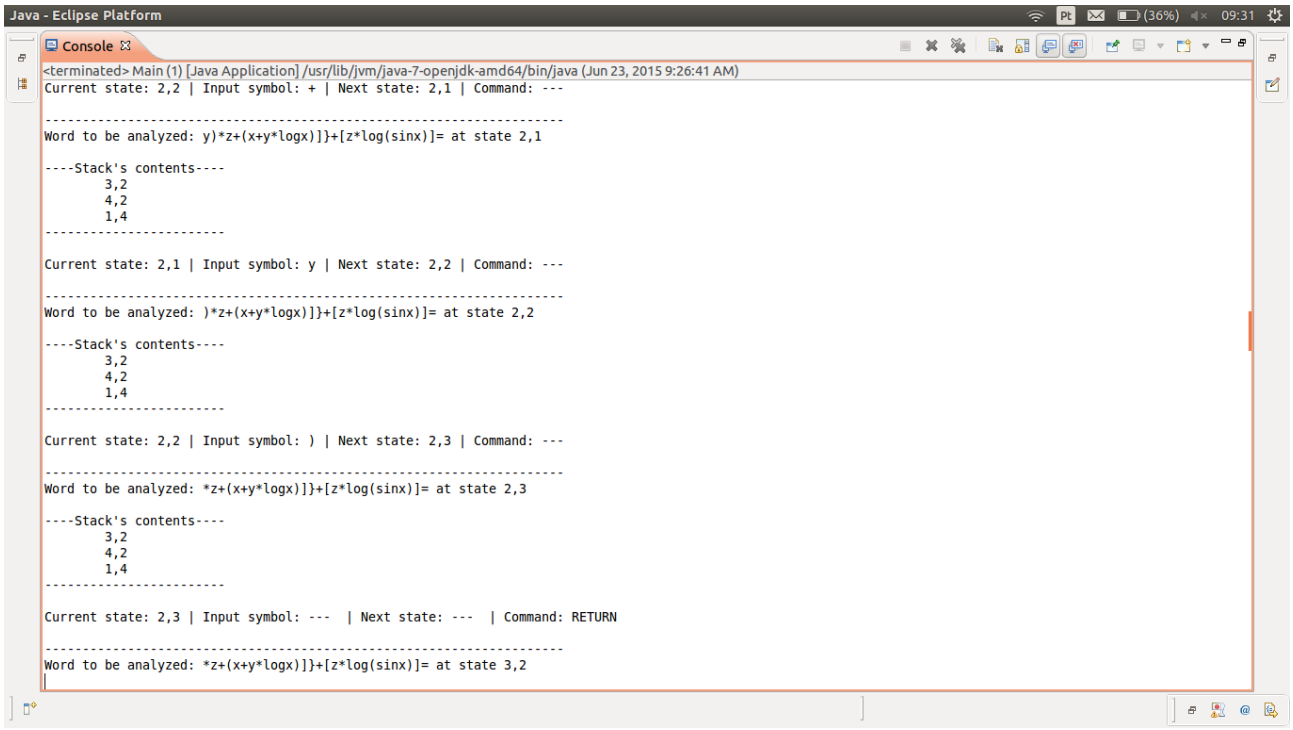


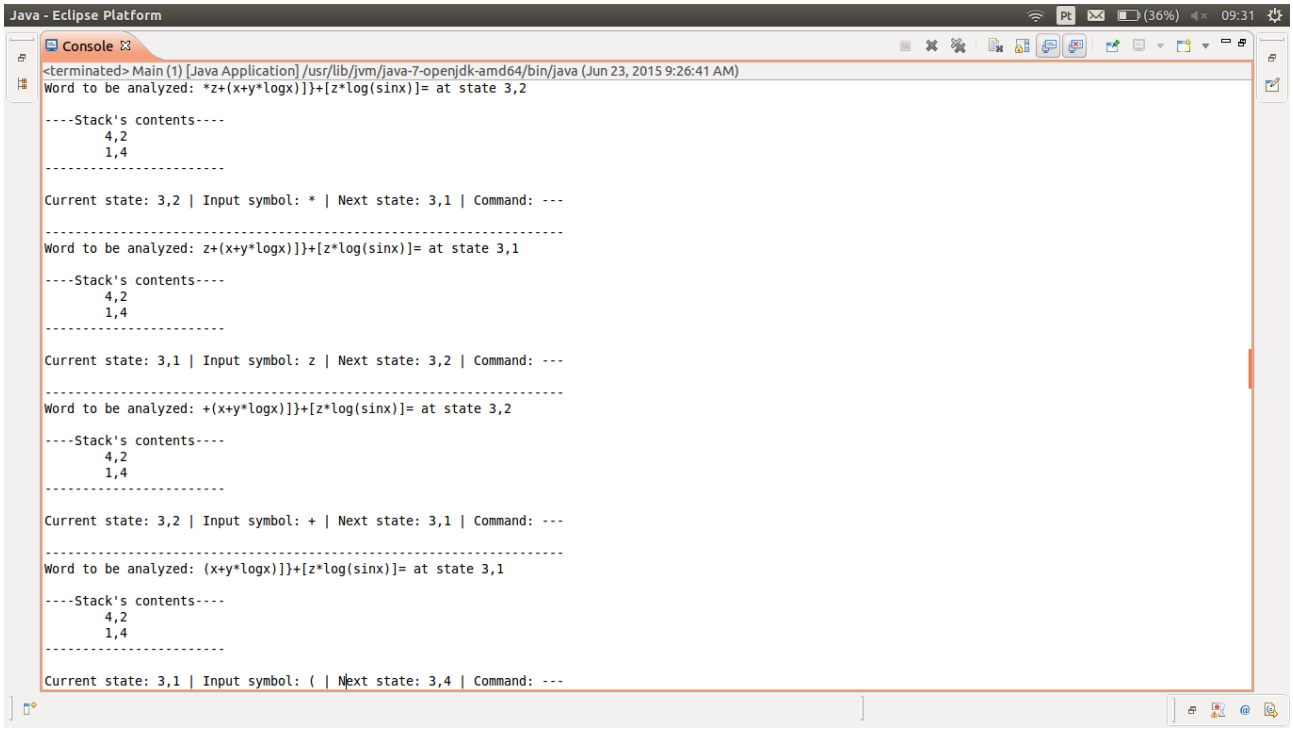

Java - Eclipse Platform
09:31
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 9:26:41 AM)
Current state: 2,3 | Input symbol: --- | Next state: --- | Command: RETURN
Current state: 2,2 | Input symbol:) | Next state: 2,3 | Command: ---
Current state: 2,2 | Input symbol: / | Next state: 2,1 | Command: ---
Current state: 2,2 | Input symbol: - | Next state: 2,1 | Command: ---
Current state: 2,2 | Input symbol: * | Next state: 2,1 | Command: ---
Current state: 2,2 | Input symbol: + | Next state: 2,1 | Command: ---
Current state: 2,1 | Input symbol: tan | Next state: 2,1 | Command: ---
Current state: 2,1 | Input symbol: cos | Next state: 2,1 | Command: ---
Current state: 2,1 | Input symbol: sin | Next state: 2,1 | Command: ---
Current state: 2,1 | Input symbol: log | Next state: 2,1 | Command: ---
Current state: 2,1 | Input symbol: z | Next state: 2,2 | Command: ---
Current state: 2,1 | Input symbol: y | Next state: 2,2 | Command: ---
Current state: 2,1 | Input symbol: x | Next state: 2,2 | Command: ---
Current state: 1,7 | Input symbol: --- | Next state: 4,1 | Command: STACK
Current state: 1,6 | Input symbol: --- | Next state: 3,1 | Command: STACK
Current state: 1,5 | Input symbol: --- | Next state: --- | Command: RETURN
Current state: 1,4 | Input symbol: / | Next state: 1,1 | Command: ---
Current state: 1,4 | Input symbol: - | Next state: 1,1 | Command: ---
Current state: 1,4 | Input symbol: + | Next state: 1,1 | Command: ---
Current state: 1,4 | Input symbol: * | Next state: 1,1 | Command: ---
Current state: 1,4 | Input symbol: = | Next state: 1,5 | Command: ---
Current state: 1,3 | Input symbol: --- | Next state: 2,1 | Command: STACK
Current state: 1,2 | Input symbol: / | Next state: 1,1 | Command: ---
Current state: 1,2 | Input symbol: * | Next state: 1,1 | Command: ---
Current state: 1,2 | Input symbol: - | Next state: 1,1 | Command: ---
Current state: 1,2 | Input symbol: + | Next state: 1,1 | Command: ---
Current state: 1,2 | Input symbol: = | Next state: 1,5 | Command: ---
Current state: 1,1 | Input symbol: { | Next state: 1,7 | Command: ---
Current state: 1,1 | Input symbol: [| Next state: 1,6 | Command: ---
Current state: 1,1 | Input symbol: (| Next state: 1,3 | Command: ---
Current state: 1,1 | Input symbol: tan | Next state: 1,1 | Command: ---
Current state: 1,1 | Input symbol: cos | Next state: 1,1 | Command: ---
Current state: 1,1 | Input symbol: sin | Next state: 1,1 | Command: ---
Current state: 1,1 | Input symbol: log | Next state: 1,1 | Command: ---
Current state: 1,1 | Input symbol: z | Next state: 1,2 | Command: ---
Current state: 1,1 | Input symbol: y | Next state: 1,2 | Command: ---
Current state: 1,1 | Input symbol: x | Next state: 1,2 | Command: ---

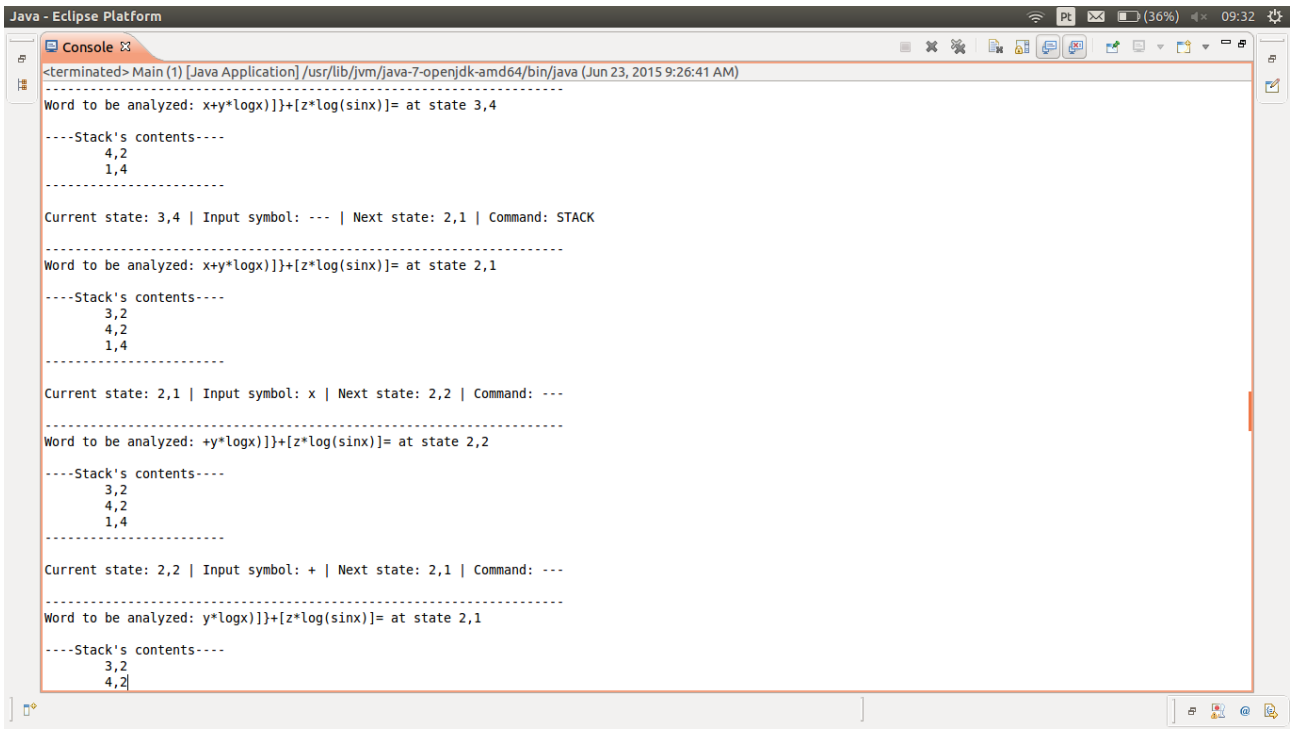

Java - Eclipse Platform
09:31
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 9:26:41 AM)
Current state of the automaton: 1,1
--------Input = x*{x*[(x+y)*z+(x+y*logx)]}+[z*log(sinx)]=--------
--------Word starting to be analyzed by the machine--------
Word to be analyzed: x*{x*[(x+y)*z+(x+y*logx)]}+[z*log(sinx)]= at state 1,1
----Stack's contents----
Current state: 1,1 | Input symbol: x | Next state: 1,2 | Command: ---
Word to be analyzed: *{x*[(x+y)*z+(x+y*logx)]}+[z*log(sinx)]= at state 1,2
----Stack's contents----
Current state: 1,2 | Input symbol: * | Next state: 1,1 | Command: ---
Word to be analyzed: {x*[(x+y)*z+(x+y*logx)]}+[z*log(sinx)]= at state 1,1
----Stack's contents----
Current state: 1,1 | Input symbol: { | Next state: 1,7 | Command: ---
Word to be analyzed: x*[(x+y)*z+(x+y*logx)]}+[z*log(sinx)]= at state 1,7
----Stack's contents----
Current state: 1,7 | Input symbol: --- | Next state: 4,1 | Command: STACK

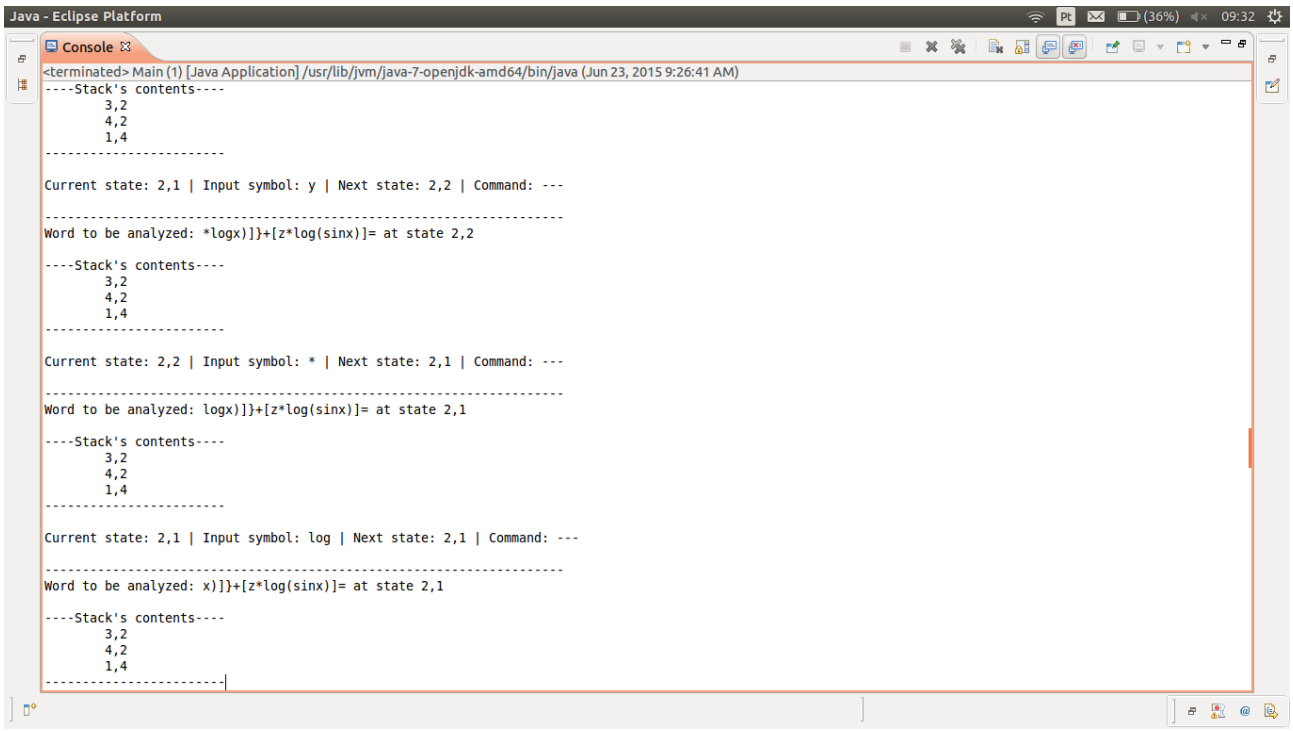

Java - Eclipse Platform
09:31
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 9:26:41 AM)
Word to be analyzed: x*[(x+y)*z+(x+y*logx)]}+[z*log(sinx)]= at state 4,1
----Stack's contents----
1,4
Current state: 4,1 | Input symbol: x | Next state: 4,2 | Command: ---
Word to be analyzed: *[(x+y)*z+(x+y*logx)]}+[z*log(sinx)]= at state 4,2
----Stack's contents----
1,4
Current state: 4,2 | Input symbol: * | Next state: 4,1 | Command: ---
Word to be analyzed: [(x+y)*z+(x+y*logx)]}+[z*log(sinx)]= at state 4,1
----Stack's contents----
1,4
Current state: 4,1 | Input symbol: [| Next state: 4,5 | Command: ---
Word to be analyzed: (x+y)*z+(x+y*logx)]}+[z*log(sinx)]= at state 4,5
----Stack's contents----
1,4
Current state: 4,5 | Input symbol: --- | Next state: 3,1 | Command: STACK
Word to be analyzed: (x+y)*z+(x+y*logx)]}+[z*log(sinx)]= at state 3,1

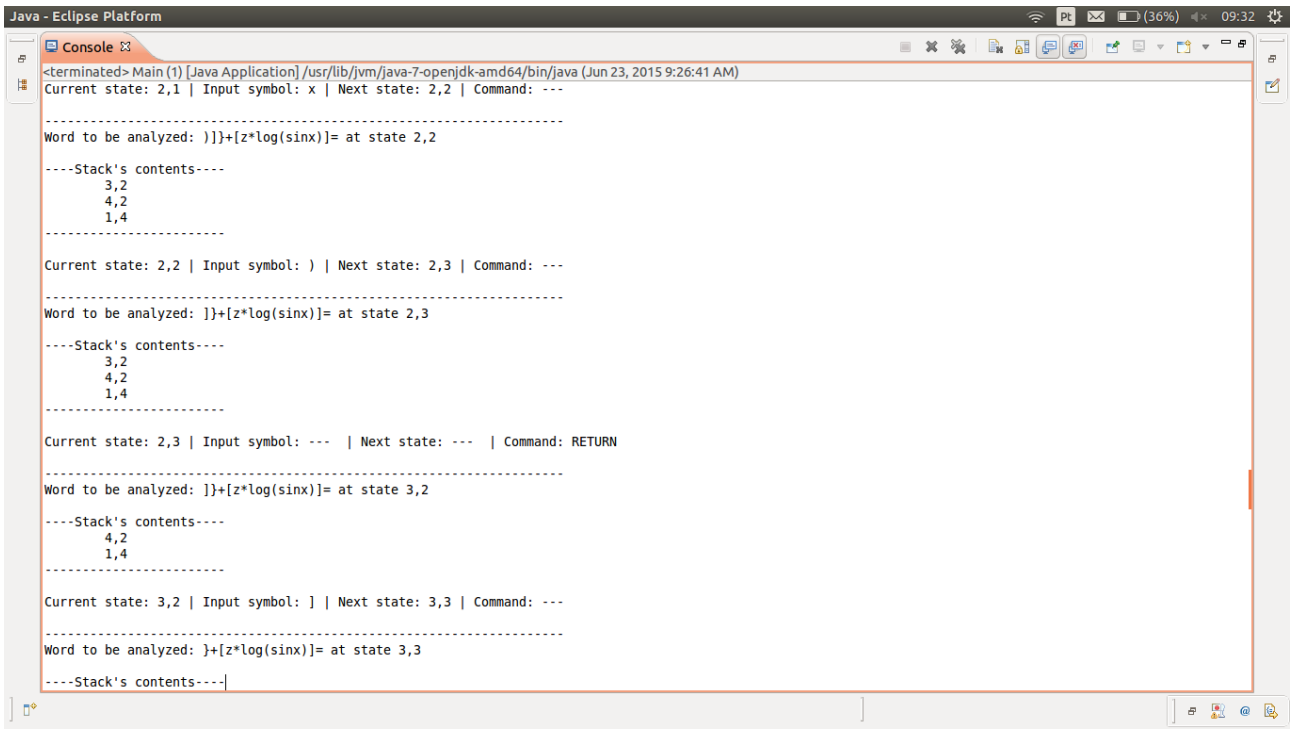

Java - Eclipse Platform
09:31
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 9:26:41 AM)
Word to be analyzed: (x+y)*z+(x+y*logx)]}+[z*log(sinx)]= at state 3,1
----Stack's contents----
4,2
1,4
Current state: 3,1 | Input symbol: (| Next state: 3,4 | Command: ---
Word to be analyzed: x+y)*z+(x+y*logx)]}+[z*log(sinx)]= at state 3,4
----Stack's contents----
4,2
1,4
Current state: 3,4 | Input symbol: --- | Next state: 2,1 | Command: STACK
Word to be analyzed: x+y)*z+(x+y*logx)]}+[z*log(sinx)]= at state 2,1
----Stack's contents----
3,2
4,2
1,4
Current state: 2,1 | Input symbol: x | Next state: 2,2 | Command: ---
Word to be analyzed: +y)*z+(x+y*logx)]}+[z*log(sinx)]= at state 2,2
----Stack's contents----
3,2
4,2
1,4


Java - Eclipse Platform
09:31
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 9:26:41 AM)
Current state: 2,2 | Input symbol: + | Next state: 2,1 | Command: ---
Word to be analyzed: y)*z+(x+y*logx)]}+[z*log(sinx)]= at state 2,1
----Stack's contents----
3,2
4,2
1,4
Current state: 2,1 | Input symbol: y | Next state: 2,2 | Command: ---
Word to be analyzed:)*z+(x+y*logx)]}+[z*log(sinx)]= at state 2,2
----Stack's contents----
3,2
4,2
1,4
Current state: 2,2 | Input symbol:) | Next state: 2,3 | Command: ---
Word to be analyzed: *z+(x+y*logx)]}+[z*log(sinx)]= at state 2,3
----Stack's contents----
3,2
4,2
1,4
Current state: 2,3 | Input symbol: --- | Next state: --- | Command: RETURN
Word to be analyzed: *z+(x+y*logx)]}+[z*log(sinx)]= at state 3,2


Java - Eclipse Platform
09:31
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 9:26:41 AM)
Word to be analyzed: *z+(x+y*logx)]}+[z*log(sinx)]= at state 3,2
----Stack's contents----
4,2
1,4
Current state: 3,2 | Input symbol: * | Next state: 3,1 | Command: ---
Word to be analyzed: z+(x+y*logx)]}+[z*log(sinx)]= at state 3,1
----Stack's contents----
4,2
1,4
Current state: 3,1 | Input symbol: z | Next state: 3,2 | Command: ---
Word to be analyzed: +(x+y*logx)]}+[z*log(sinx)]= at state 3,2
----Stack's contents----
4,2
1,4
Current state: 3,2 | Input symbol: + | Next state: 3,1 | Command: ---
Word to be analyzed: (x+y*logx)]}+[z*log(sinx)]= at state 3,1
----Stack's contents----
4,2
1,4
Current state: 3,1 | Input symbol: (| Next state: 3,4 | Command: ---


Java - Eclipse Platform
(36%) 09:32
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 9:26:41 AM)
Word to be analyzed: x+y*logx)]}+[z*log(sinx)]= at state 3,4
----Stack's contents----
4,2
1,4
Current state: 3,4 | Input symbol: --- | Next state: 2,1 | Command: STACK
Word to be analyzed: x+y*logx)]}+[z*log(sinx)]= at state 2,1
----Stack's contents----
3,2
4,2
1,4
Current state: 2,1 | Input symbol: x | Next state: 2,2 | Command: ---
Word to be analyzed: +y*logx)]}+[z*log(sinx)]= at state 2,2
----Stack's contents----
3,2
4,2
1,4
Current state: 2,2 | Input symbol: + | Next state: 2,1 | Command: ---
Word to be analyzed: y*logx)]}+[z*log(sinx)]= at state 2,1
----Stack's contents----
3,2
4,2


Java - Eclipse Platform
(36%) 09:32
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 9:26:41 AM)
----Stack's contents----
3,2
4,2
1,4
Current state: 2,1 | Input symbol: y | Next state: 2,2 | Command: ---
Word to be analyzed: *logx)]}+[z*log(sinx)]= at state 2,2
----Stack's contents----
3,2
4,2
1,4
Current state: 2,2 | Input symbol: * | Next state: 2,1 | Command: ---
Word to be analyzed: logx)]}+[z*log(sinx)]= at state 2,1
----Stack's contents----
3,2
4,2
1,4
Current state: 2,1 | Input symbol: log | Next state: 2,1 | Command: ---
Word to be analyzed: x)]}+[z*log(sinx)]= at state 2,1
----Stack's contents----
3,2
4,2
1,4


Java - Eclipse Platform
09:32
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 9:26:41 AM)
Current state: 2,1 | Input symbol: x | Next state: 2,2 | Command: ---
---
Word to be analyzed:)]}+[z*log(sinx)]= at state 2,2
----Stack's contents----
3,2
4,2
1,4

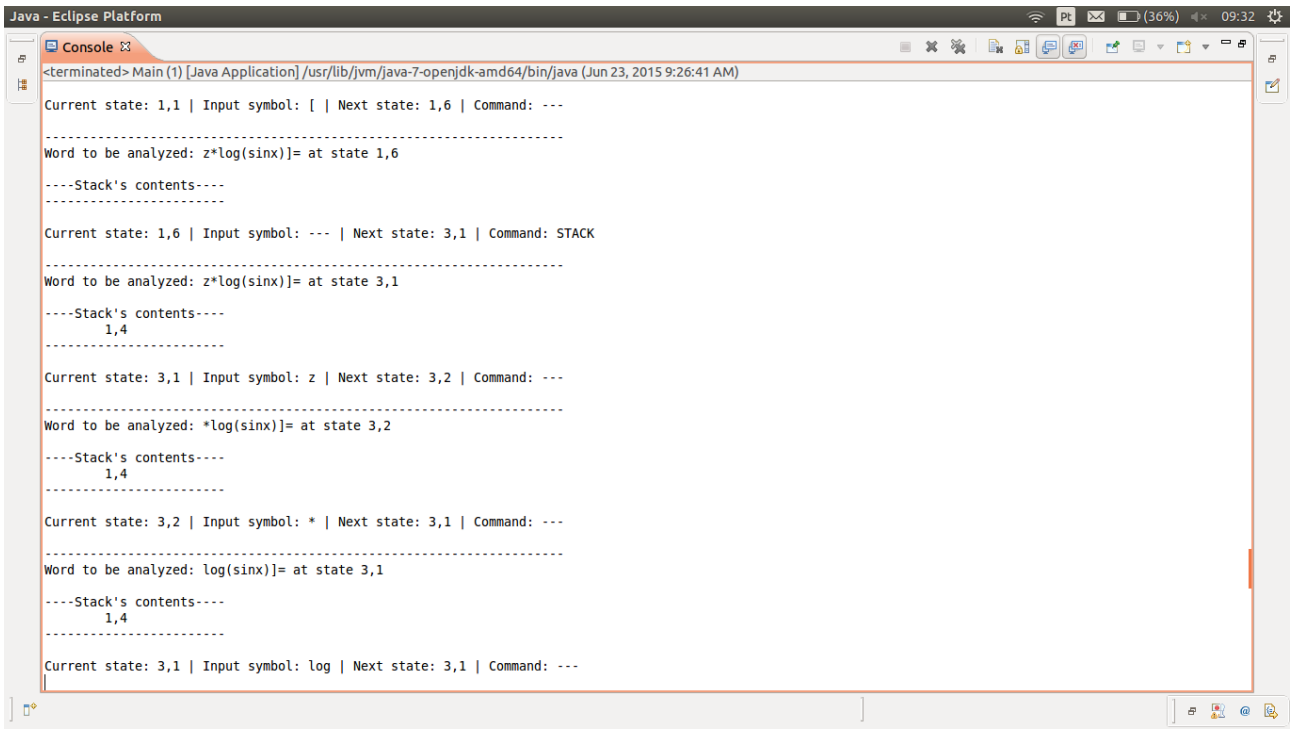
Current state: 2,2 | Input symbol:) | Next state: 2,3 | Command: ---
---
Word to be analyzed:]}+[z*log(sinx)]= at state 2,3
----Stack's contents----
3,2
4,2
1,4

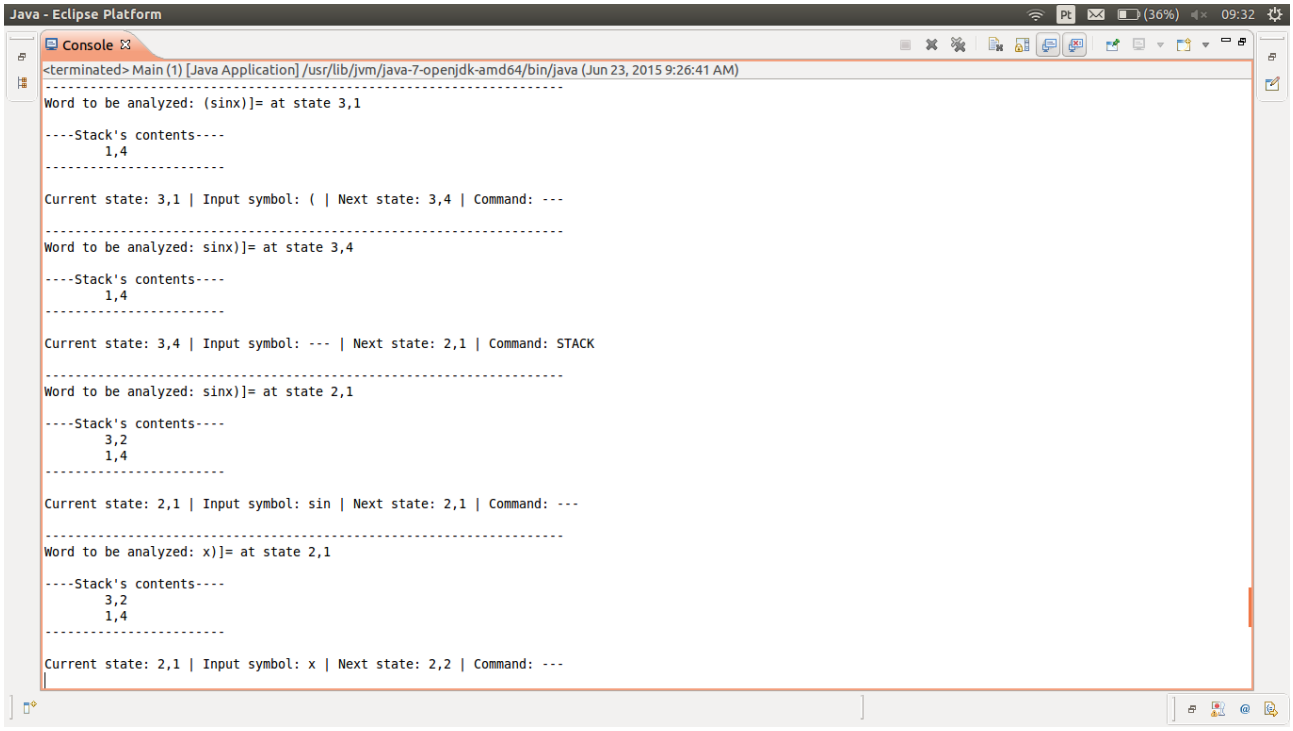
Current state: 2,3 | Input symbol: --- | Next state: --- | Command: RETURN

Word to be analyzed:]}+[z*log(sinx)]= at state 3,2
----Stack's contents----
4,2
1,4

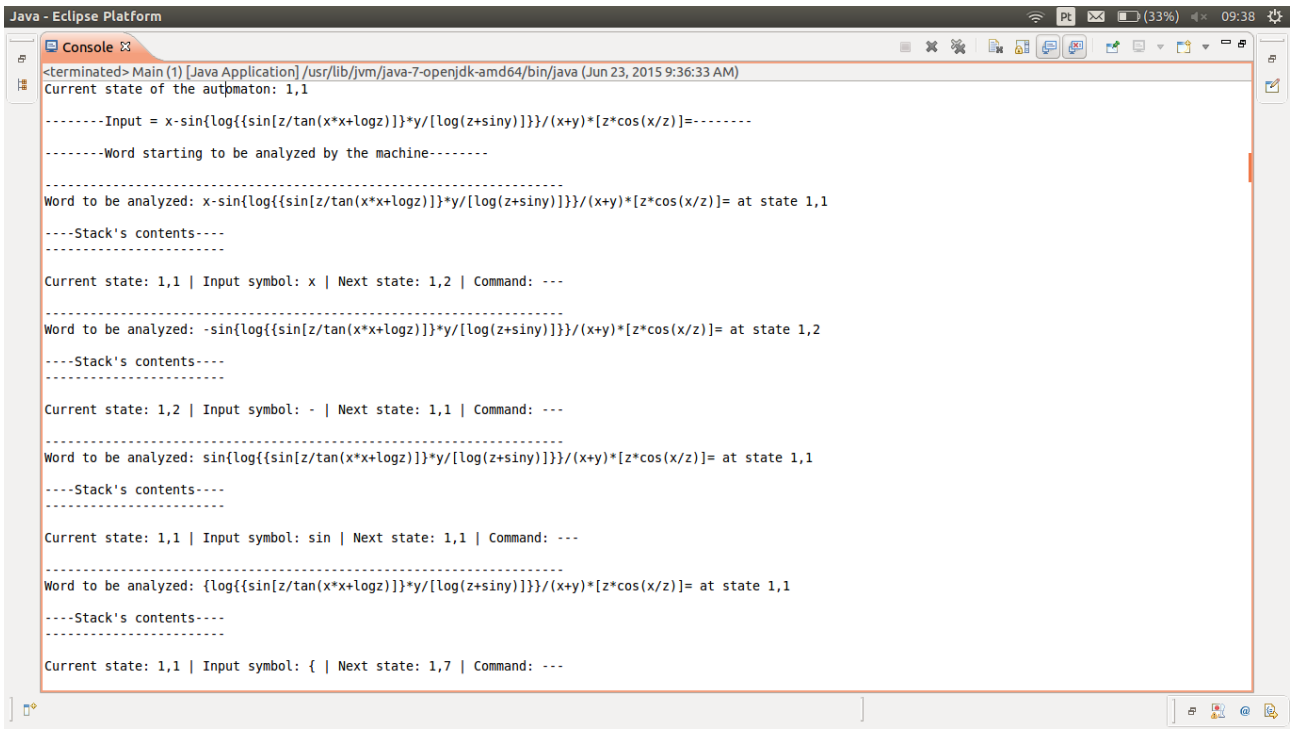
Current state: 3,2 | Input symbol:] | Next state: 3,3 | Command: ---
---
Word to be analyzed: }+[z*log(sinx)]= at state 3,3
----Stack's contents----

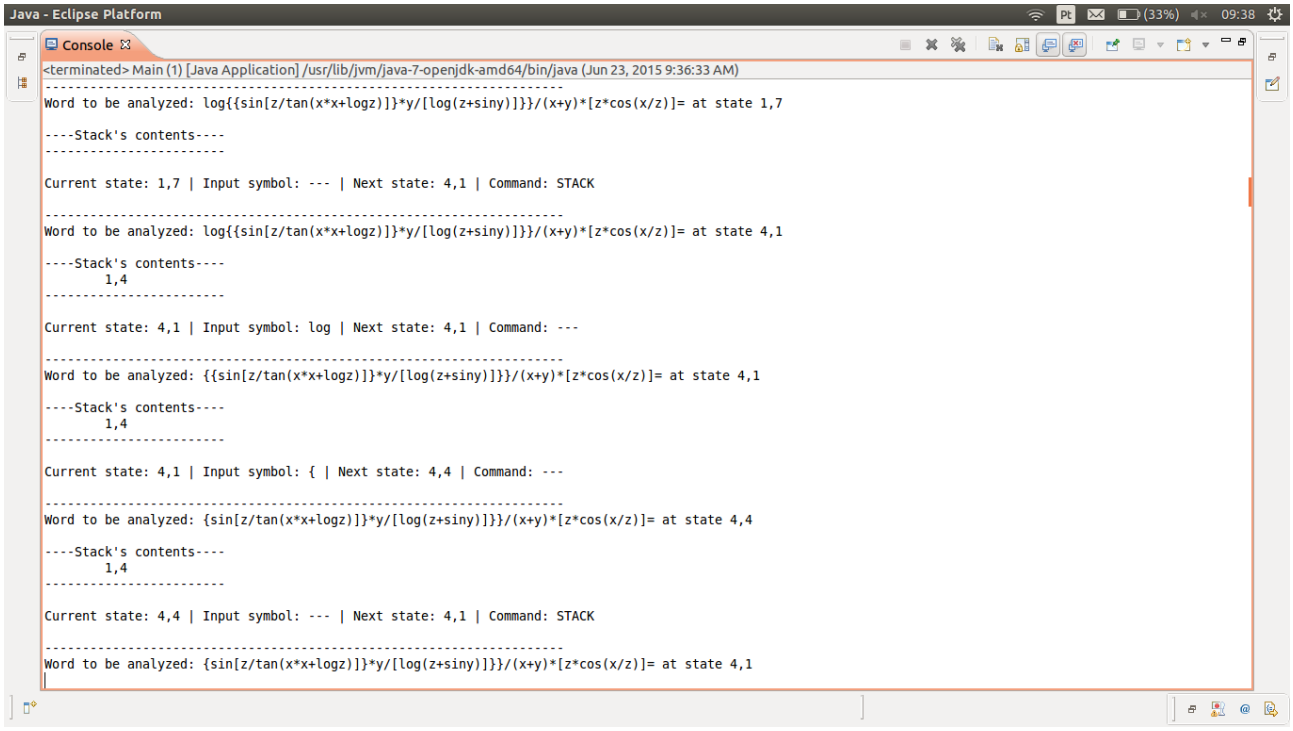

Java - Eclipse Platform
09:32
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 9:26:41 AM)
----Stack's contents----
4,2
1,4
Current state: 3,3 | Input symbol: --- | Next state: --- | Command: RETURN
Word to be analyzed: }+[z*log(sinx)]= at state 4,2
----Stack's contents----
1,4
Current state: 4,2 | Input symbol: } | Next state: 4,3 | Command: ---
Word to be analyzed: +[z*log(sinx)]= at state 4,3
----Stack's contents----
1,4
Current state: 4,3 | Input symbol: --- | Next state: --- | Command: RETURN
Word to be analyzed: +[z*log(sinx)]= at state 1,4
----Stack's contents----
Current state: 1,4 | Input symbol: + | Next state: 1,1 | Command: ---
Word to be analyzed: [z*log(sinx)]= at state 1,1
----Stack's contents----

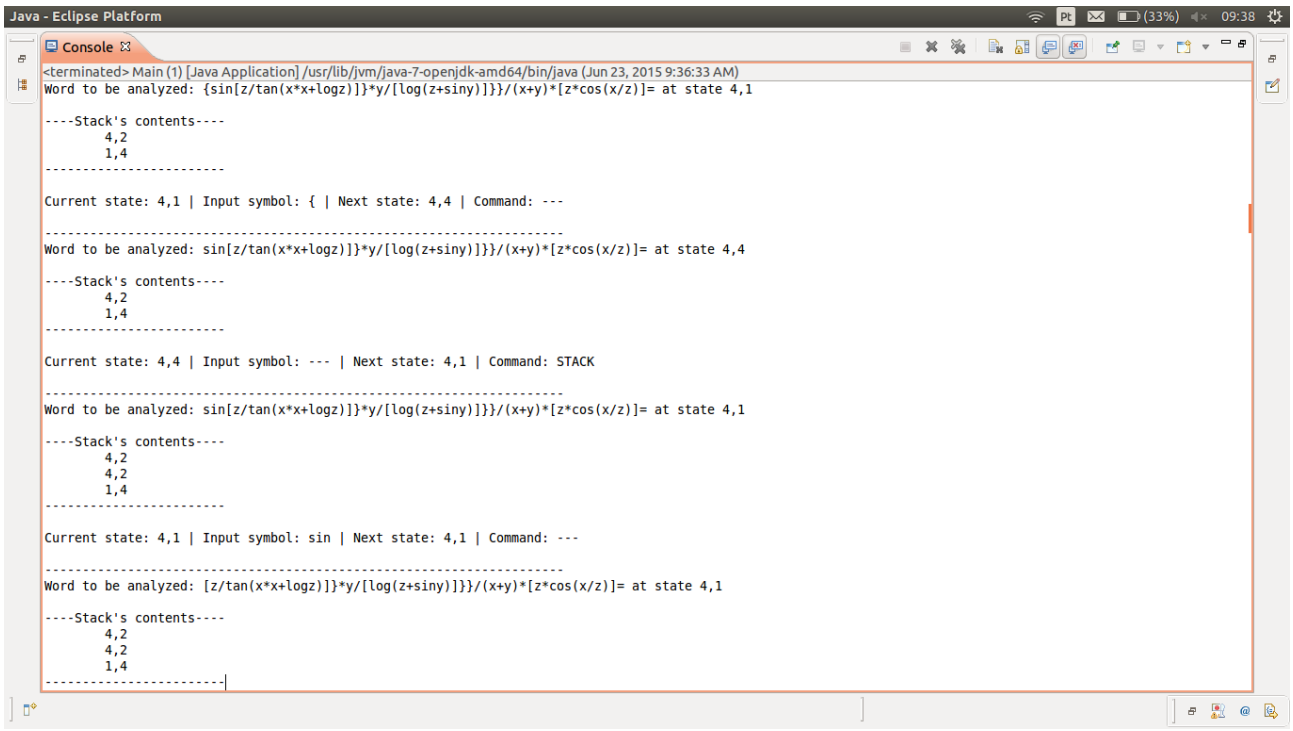

Java - Eclipse Platform
(36%) 09:32
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 9:26:41 AM)
Current state: 1,1 | Input symbol: [| Next state: 1,6 | Command: ---
Word to be analyzed: z*log(sinx)]= at state 1,6
----Stack's contents----
Current state: 1,6 | Input symbol: --- | Next state: 3,1 | Command: STACK
Word to be analyzed: z*log(sinx)]= at state 3,1
----Stack's contents----
1,4
Current state: 3,1 | Input symbol: z | Next state: 3,2 | Command: ---
Word to be analyzed: *log(sinx)]= at state 3,2
----Stack's contents----
1,4
Current state: 3,2 | Input symbol: * | Next state: 3,1 | Command: ---
Word to be analyzed: log(sinx)]= at state 3,1
----Stack's contents----
1,4
Current state: 3,1 | Input symbol: log | Next state: 3,1 | Command: ---

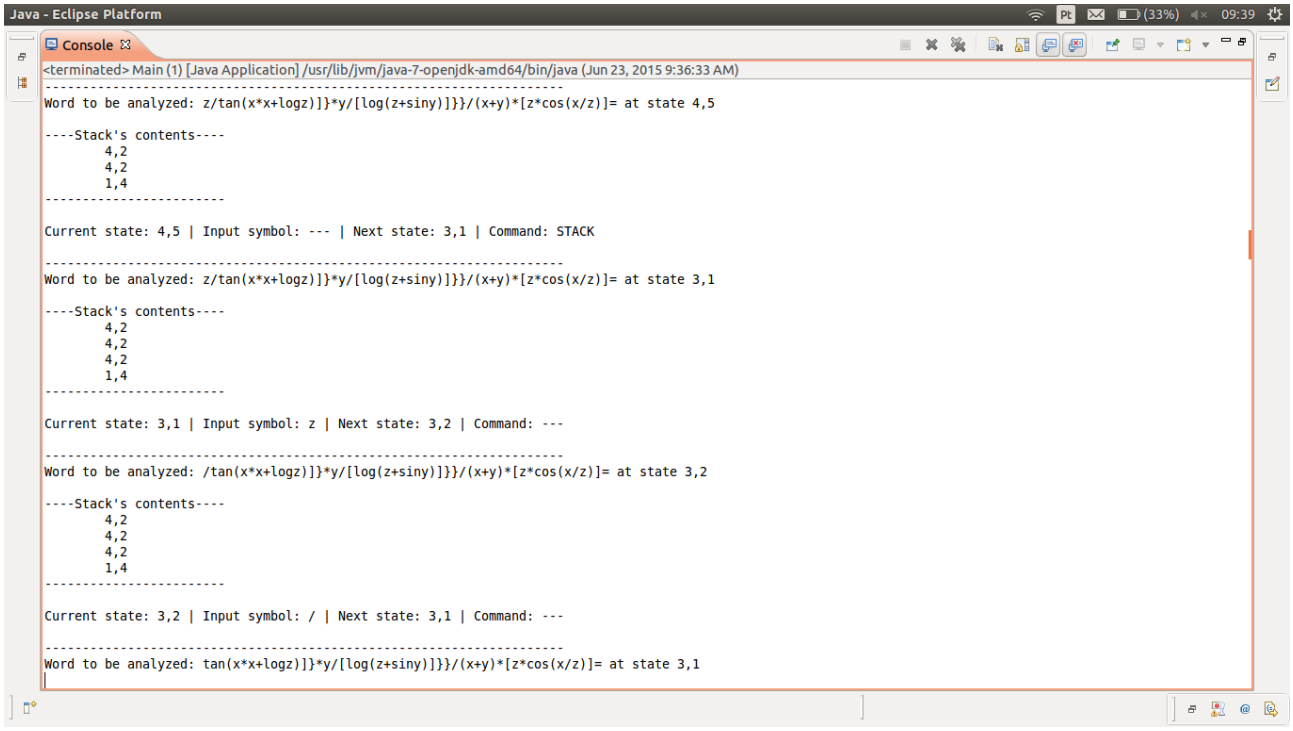

Java - Eclipse Platform
(36%) 09:32
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 9:26:41 AM)
Word to be analyzed: (sinx)]= at state 3,1
----Stack's contents----
1,4
Current state: 3,1 | Input symbol: (| Next state: 3,4 | Command: ---
Word to be analyzed: sinx)]= at state 3,4
----Stack's contents----
1,4
Current state: 3,4 | Input symbol: --- | Next state: 2,1 | Command: STACK
Word to be analyzed: sinx)]= at state 2,1
----Stack's contents----
3,2
1,4
Current state: 2,1 | Input symbol: sin | Next state: 2,1 | Command: ---
Word to be analyzed: x)]= at state 2,1
----Stack's contents----
3,2
1,4
Current state: 2,1 | Input symbol: x | Next state: 2,2 | Command: ---

2. x-sin{log{{sin[z/tan(x*x+logz)]}*y/[log(z+siny)]}}/(x+y)*[z*cos(x/z)]=: Aceito

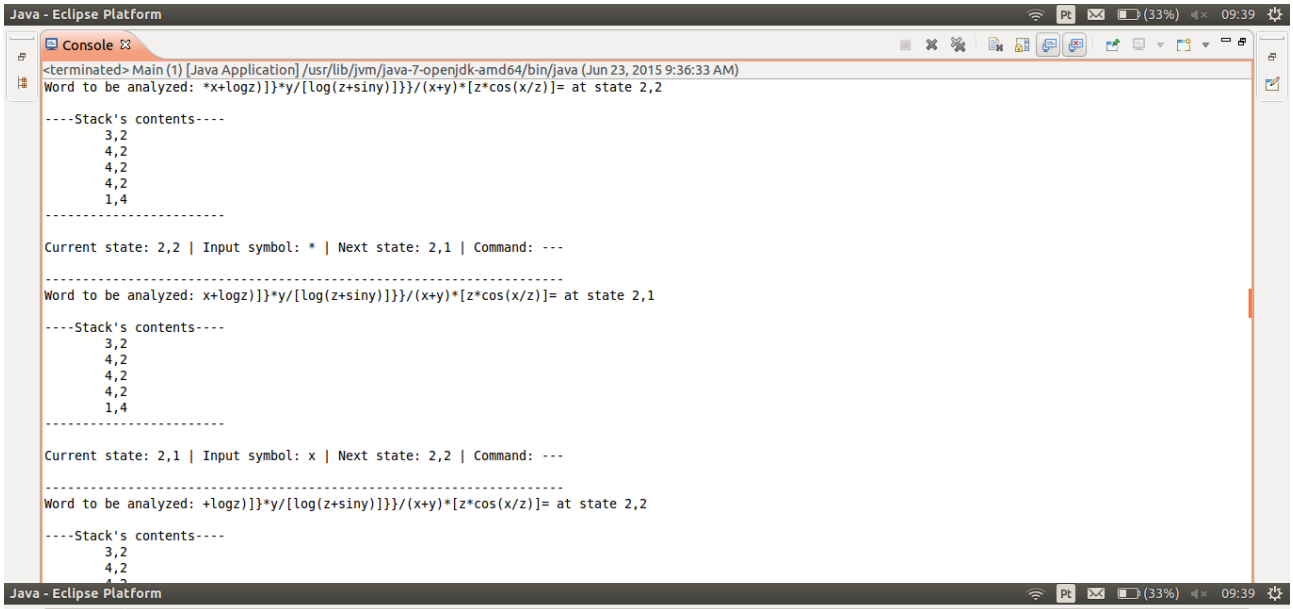

Java - Eclipse Platform
(33%) 09:38
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 9:36:33 AM)
Current state of the automaton: 1,1
--------Input = x-sin{log{{sin[z/tan(x*x+logz)]}*y/[log(z+siny)]}}/(x+y)*[z*cos(x/z)]=--------
--------Word starting to be analyzed by the machine--------
Word to be analyzed: x-sin{log{{sin[z/tan(x*x+logz)]}*y/[log(z+siny)]}}/(x+y)*[z*cos(x/z)]= at state 1,1
----Stack's contents----
Current state: 1,1 | Input symbol: x | Next state: 1,2 | Command: ---
Word to be analyzed: -sin{log{{sin[z/tan(x*x+logz)]}*y/[log(z+siny)]}}/(x+y)*[z*cos(x/z)]= at state 1,2
----Stack's contents----
Current state: 1,2 | Input symbol: - | Next state: 1,1 | Command: ---
Word to be analyzed: sin{log{{sin[z/tan(x*x+logz)]}*y/[log(z+siny)]}}/(x+y)*[z*cos(x/z)]= at state 1,1
----Stack's contents----
Current state: 1,1 | Input symbol: sin | Next state: 1,1 | Command: ---
Word to be analyzed: {log{{sin[z/tan(x*x+logz)]}*y/[log(z+siny)]}}/(x+y)*[z*cos(x/z)]= at state 1,1
----Stack's contents----
Current state: 1,1 | Input symbol: { | Next state: 1,7 | Command: ---

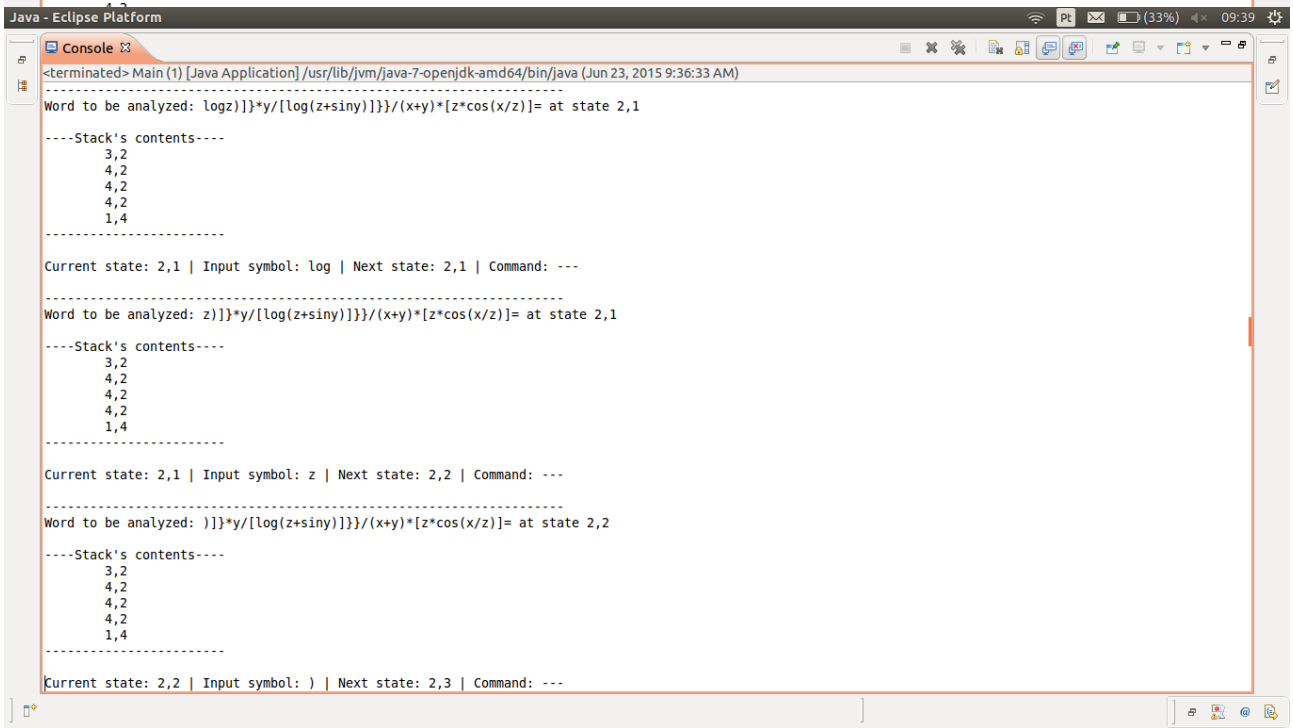

Java - Eclipse Platform
(33%) 09:38
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 9:36:33 AM)
Word to be analyzed: log{{sin[z/tan(x*x+logz)]}*y/[log(z+siny)]}}/(x+y)*[z*cos(x/z)]= at state 1,7
----Stack's contents----
Current state: 1,7 | Input symbol: --- | Next state: 4,1 | Command: STACK
Word to be analyzed: log{{sin[z/tan(x*x+logz)]}*y/[log(z+siny)]}}/(x+y)*[z*cos(x/z)]= at state 4,1
----Stack's contents----
1,4
Current state: 4,1 | Input symbol: log | Next state: 4,1 | Command: ---
Word to be analyzed: {{sin[z/tan(x*x+logz)]}*y/[log(z+siny)]}}/(x+y)*[z*cos(x/z)]= at state 4,1
----Stack's contents----
1,4
Current state: 4,1 | Input symbol: { | Next state: 4,4 | Command: ---
Word to be analyzed: {sin[z/tan(x*x+logz)]}*y/[log(z+siny)]}}/(x+y)*[z*cos(x/z)]= at state 4,4
----Stack's contents----
1,4
Current state: 4,4 | Input symbol: --- | Next state: 4,1 | Command: STACK
Word to be analyzed: {sin[z/tan(x*x+logz)]}*y/[log(z+siny)]}}/(x+y)*[z*cos(x/z)]= at state 4,1


Java - Eclipse Platform
09:38
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 9:36:33 AM)
Word to be analyzed: {sin[z/tan(x*x+logz)]}*y/[log(z+siny)]}}/(x+y)*[z*cos(x/z)]= at state 4,1
----Stack's contents----
4,2
1,4
Current state: 4,1 | Input symbol: { | Next state: 4,4 | Command: ---
Word to be analyzed: sin[z/tan(x*x+logz)]}*y/[log(z+siny)]}}/(x+y)*[z*cos(x/z)]= at state 4,4
----Stack's contents----
4,2
1,4
Current state: 4,4 | Input symbol: --- | Next state: 4,1 | Command: STACK
Word to be analyzed: sin[z/tan(x*x+logz)]}*y/[log(z+siny)]}}/(x+y)*[z*cos(x/z)]= at state 4,1
----Stack's contents----
4,2
4,2
1,4
Current state: 4,1 | Input symbol: sin | Next state: 4,1 | Command: ---
Word to be analyzed: [z/tan(x*x+logz)]}*y/[log(z+siny)]}}/(x+y)*[z*cos(x/z)]= at state 4,1
----Stack's contents----
4,2
4,2
1,4

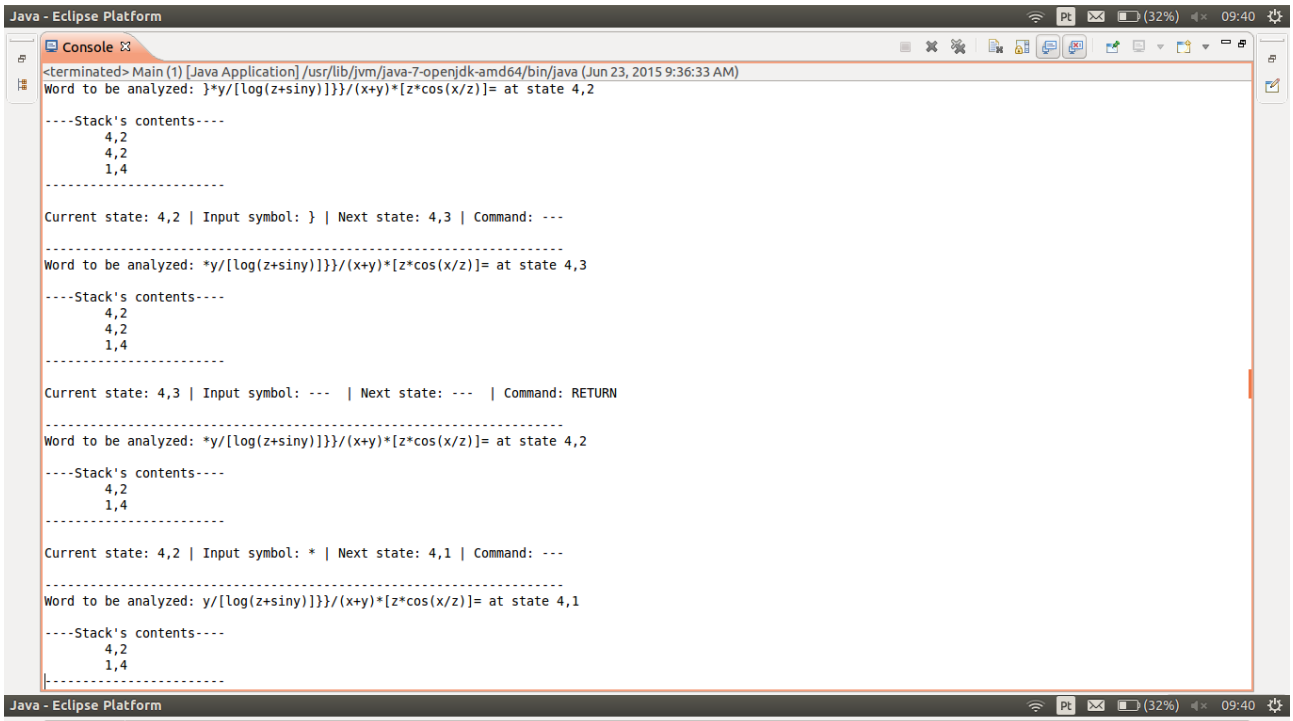

Java - Eclipse Platform
09:39
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 9:36:33 AM)
Word to be analyzed: z/tan(x*x+logz)]}*y/[log(z+siny)]}}/(x+y)*[z*cos(x/z)]= at state 4,5
----Stack's contents----
4,2
4,2
1,4
Current state: 4,5 | Input symbol: --- | Next state: 3,1 | Command: STACK
Word to be analyzed: z/tan(x*x+logz)]}*y/[log(z+siny)]}}/(x+y)*[z*cos(x/z)]= at state 3,1
----Stack's contents----
4,2
4,2
4,2
1,4
Current state: 3,1 | Input symbol: z | Next state: 3,2 | Command: ---
Word to be analyzed: /tan(x*x+logz)]}*y/[log(z+siny)]}}/(x+y)*[z*cos(x/z)]= at state 3,2
----Stack's contents----
4,2
4,2
4,2
1,4
Current state: 3,2 | Input symbol: / | Next state: 3,1 | Command: ---
Word to be analyzed: tan(x*x+logz)]}*y/[log(z+siny)]}}/(x+y)*[z*cos(x/z)]= at state 3,1

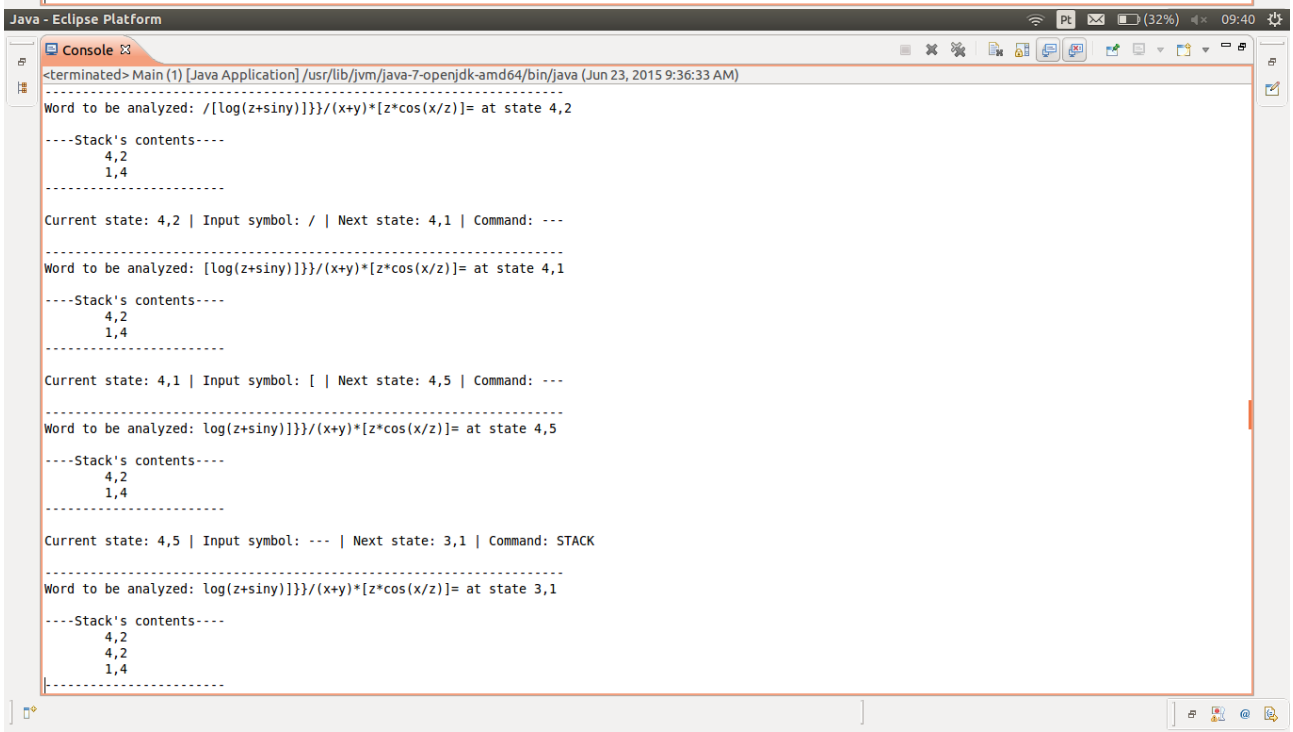

Java - Eclipse Platform
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 9:36:33 AM)
Word to be analyzed: (x*x+logz)]}*y/[log(z+siny)]}}/(x+y)*[z*cos(x/z)]= at state 3,1
----Stack's contents----
4,2
4,2
4,2
1,4
Current state: 3,1 | Input symbol: (| Next state: 3,4 | Command: ---
Word to be analyzed: x*x+logz)]}*y/[log(z+siny)]}}/(x+y)*[z*cos(x/z)]= at state 3,4
----Stack's contents----
4,2
4,2
4,2
1,4
Current state: 3,4 | Input symbol: --- | Next state: 2,1 | Command: STACK
Word to be analyzed: x*x+logz)]}*y/[log(z+siny)]}}/(x+y)*[z*cos(x/z)]= at state 2,1
----Stack's contents----
3,2
4,2
4,2
4,2
1,4
Current state: 2,1 | Input symbol: x | Next state: 2,2 | Command: ---

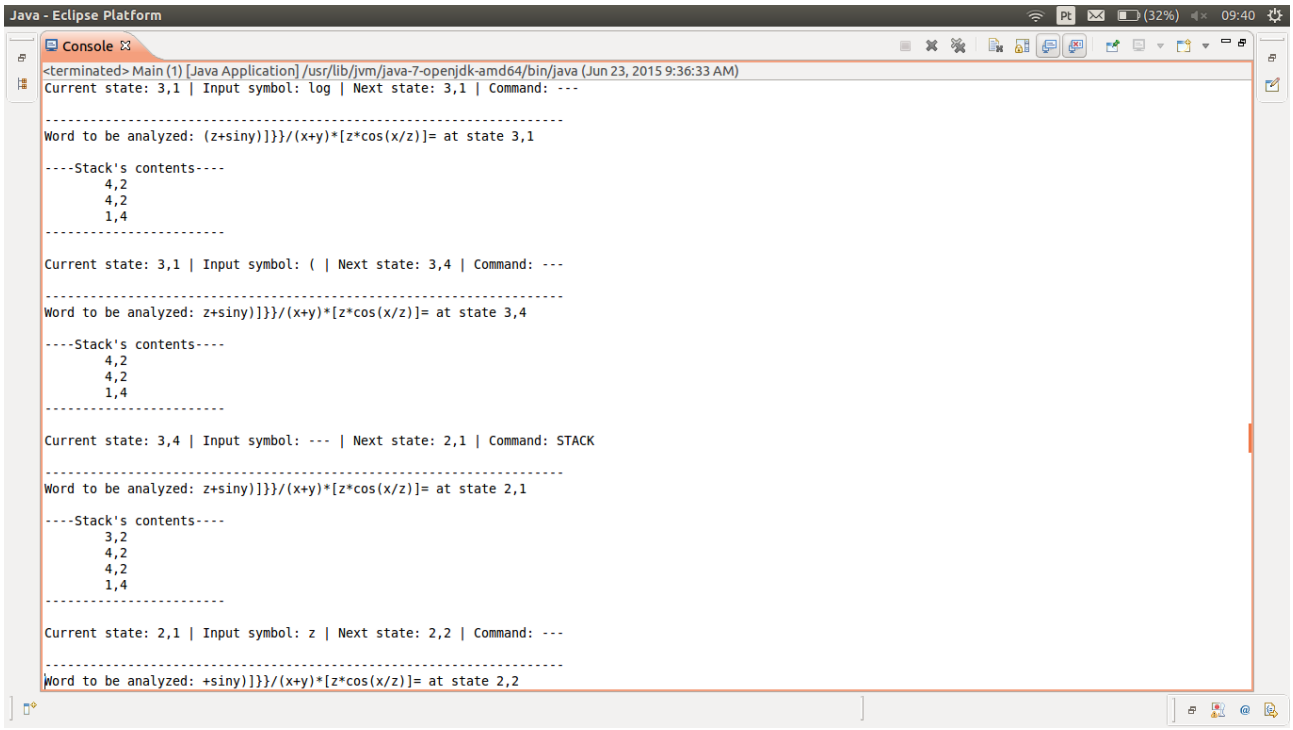

Java - Eclipse Platform
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 9:36:33 AM)
Word to be analyzed: *x+logz)]}*y/[log(z+siny)]}}/(x+y)*[z*cos(x/z)]= at state 2,2
----Stack's contents----
3,2
4,2
4,2
4,2
1,4
Current state: 2,2 | Input symbol: * | Next state: 2,1 | Command: ---
Word to be analyzed: x+logz)]}*y/[log(z+siny)]}}/(x+y)*[z*cos(x/z)]= at state 2,1
----Stack's contents----
3,2
4,2
4,2
4,2
1,4
Current state: 2,1 | Input symbol: x | Next state: 2,2 | Command: ---
Word to be analyzed: +logz)]}*y/[log(z+siny)]}}/(x+y)*[z*cos(x/z)]= at state 2,2
----Stack's contents----
3,2
4,2

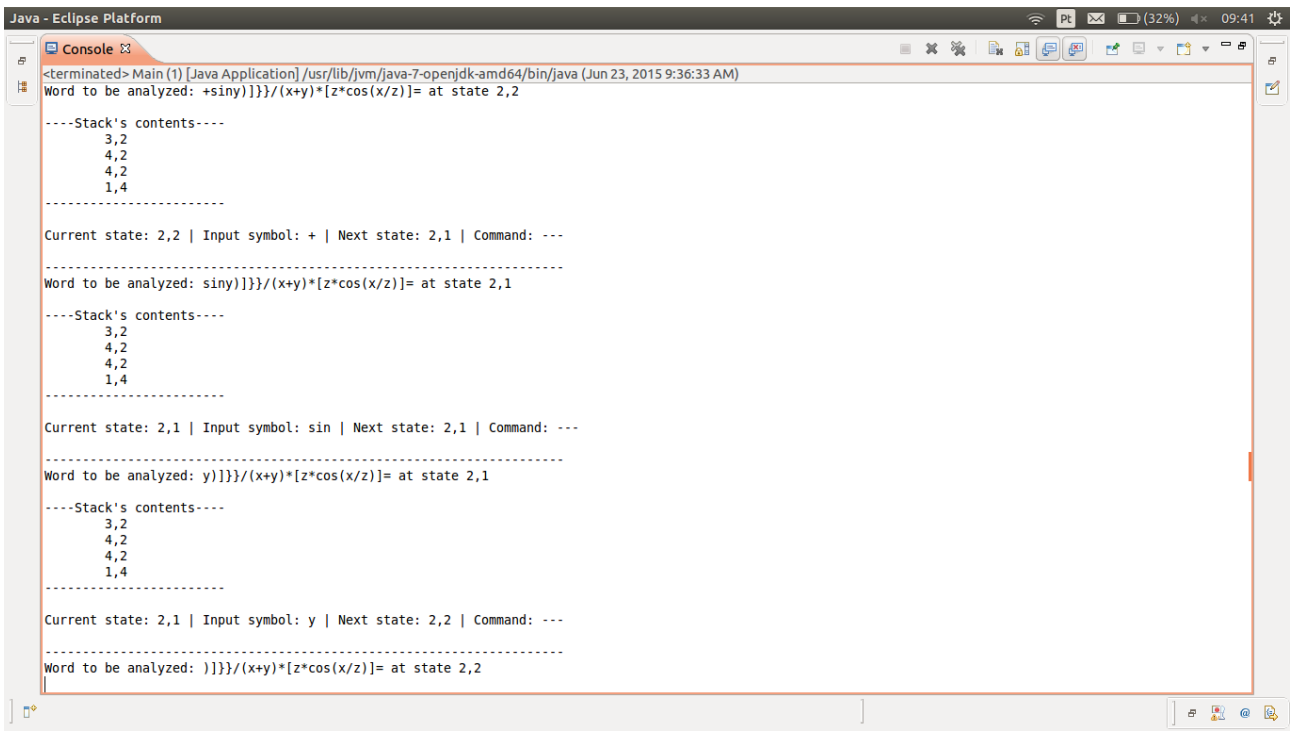

Java - Eclipse Platform
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 9:36:33 AM)
Word to be analyzed: logz)]}*y/[log(z+siny)]}}/(x+y)*[z*cos(x/z)]= at state 2,1
----Stack's contents----
3,2
4,2
4,2
4,2
1,4
Current state: 2,1 | Input symbol: log | Next state: 2,1 | Command: ---
Word to be analyzed: z)]}*y/[log(z+siny)]}}/(x+y)*[z*cos(x/z)]= at state 2,1
----Stack's contents----
3,2
4,2
4,2
4,2
1,4
Current state: 2,1 | Input symbol: z | Next state: 2,2 | Command: ---
Word to be analyzed:)]}*y/[log(z+siny)]}}/(x+y)*[z*cos(x/z)]= at state 2,2
----Stack's contents----
3,2
4,2
4,2
4,2
1,4
Current state: 2,2 | Input symbol:) | Next state: 2,3 | Command: ---

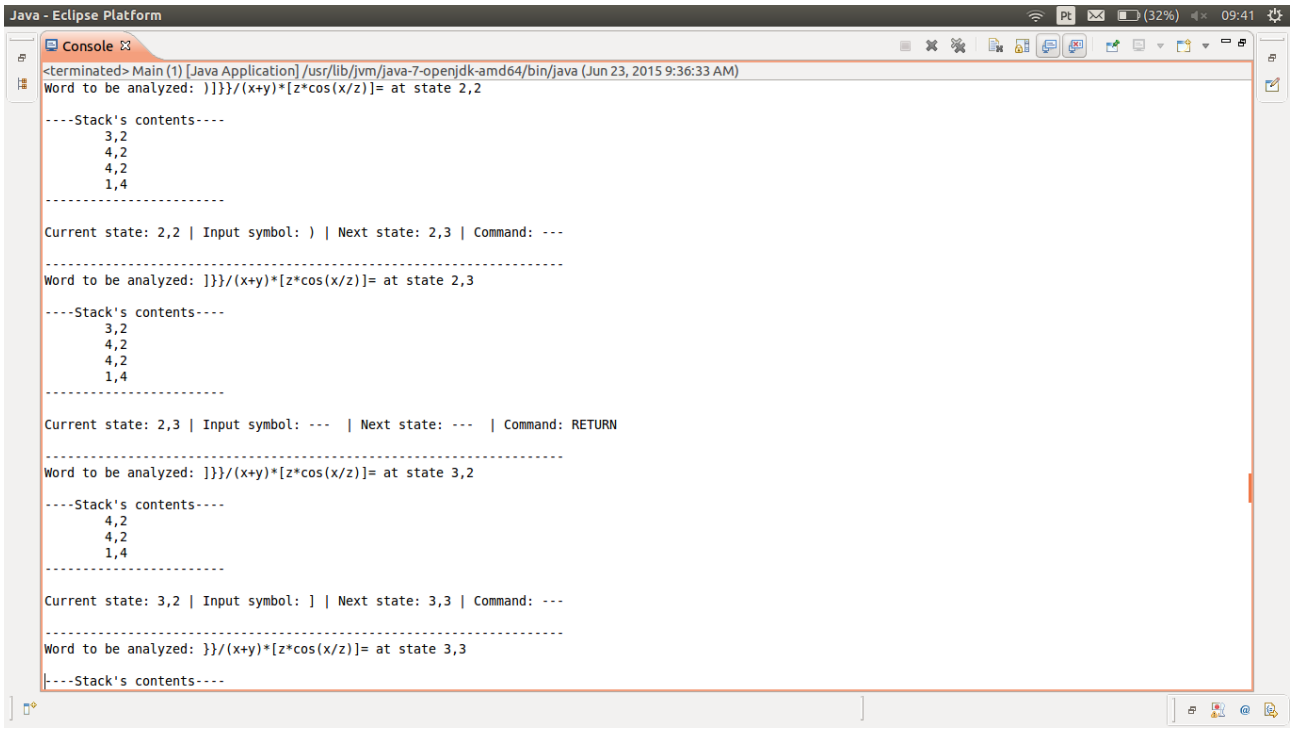

Java - Eclipse Platform
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 9:36:33 AM)
Word to be analyzed:]}*y/[log(z+siny)]}}/(x+y)*[z*cos(x/z)]= at state 2,3
----Stack's contents----
3,2
4,2
4,2
4,2
1,4
Current state: 2,3 | Input symbol: --- | Next state: --- | Command: RETURN
Word to be analyzed:]}*y/[log(z+siny)]}}/(x+y)*[z*cos(x/z)]= at state 3,2
----Stack's contents----
4,2
4,2
4,2
1,4
Current state: 3,2 | Input symbol:] | Next state: 3,3 | Command: ---
Word to be analyzed: }*y/[log(z+siny)]}}/(x+y)*[z*cos(x/z)]= at state 3,3
----Stack's contents----
4,2
4,2
4,2
1,4
Current state: 3,3 | Input symbol: --- | Next state: --- | Command: RETURN
Word to be analyzed: }*y/[log(z+siny)]}}/(x+y)*[z*cos(x/z)]= at state 4,2

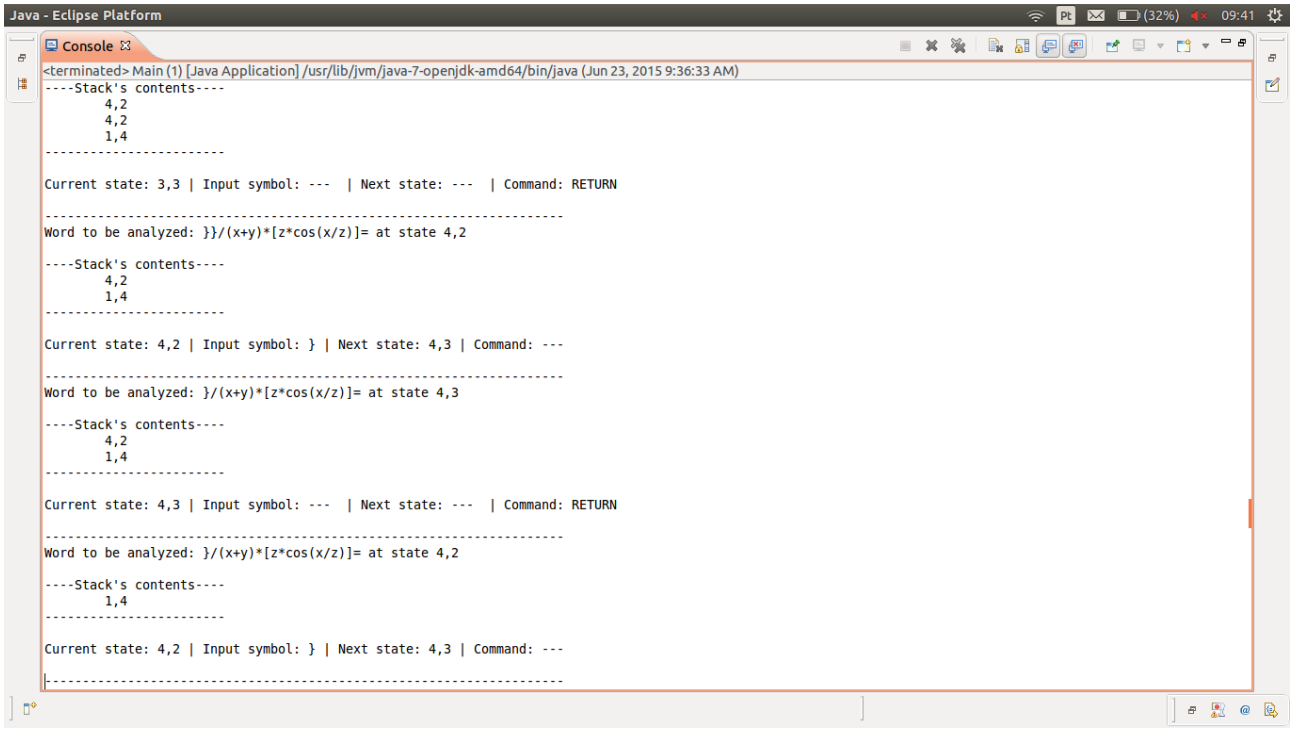

Java - Eclipse Platform
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 9:36:33 AM)
Word to be analyzed: }*y/[log(z+siny)]}}/(x+y)*[z*cos(x/z)]= at state 4,2
----Stack's contents----
4,2
4,2
1,4
Current state: 4,2 | Input symbol: } | Next state: 4,3 | Command: ---
Word to be analyzed: *y/[log(z+siny)]}}/(x+y)*[z*cos(x/z)]= at state 4,3
----Stack's contents----
4,2
4,2
1,4
Current state: 4,3 | Input symbol: --- | Next state: --- | Command: RETURN
Word to be analyzed: *y/[log(z+siny)]}}/(x+y)*[z*cos(x/z)]= at state 4,2
----Stack's contents----
4,2
1,4
Current state: 4,2 | Input symbol: * | Next state: 4,1 | Command: ---
Word to be analyzed: y/[log(z+siny)]}}/(x+y)*[z*cos(x/z)]= at state 4,1
----Stack's contents----
4,2
1,4

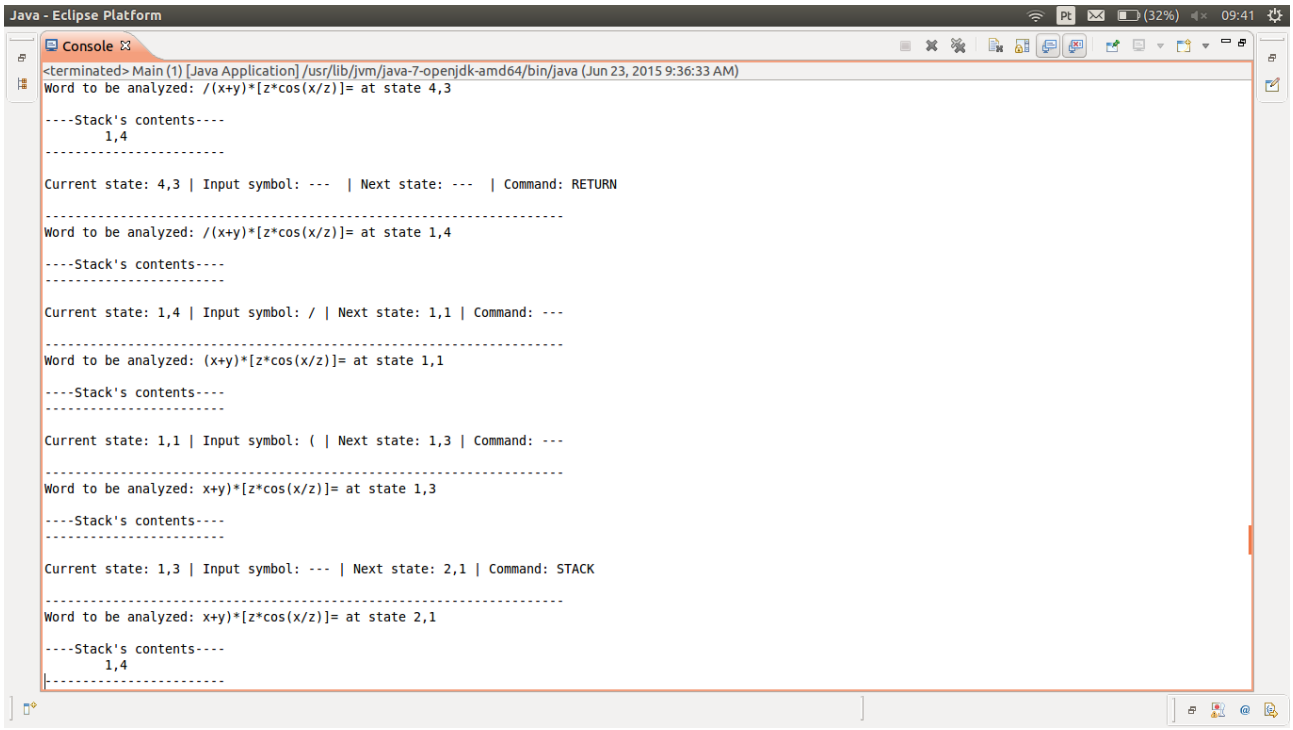

Java - Eclipse Platform
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 9:36:33 AM)
Word to be analyzed: /[log(z+siny)]}}/(x+y)*[z*cos(x/z)]= at state 4,2
----Stack's contents----
4,2
1,4
Current state: 4,2 | Input symbol: / | Next state: 4,1 | Command: ---
Word to be analyzed: [log(z+siny)]}}/(x+y)*[z*cos(x/z)]= at state 4,1
----Stack's contents----
4,2
1,4
Current state: 4,1 | Input symbol: [| Next state: 4,5 | Command: ---
Word to be analyzed: log(z+siny)]}}/(x+y)*[z*cos(x/z)]= at state 4,5
----Stack's contents----
4,2
1,4
Current state: 4,5 | Input symbol: --- | Next state: 3,1 | Command: STACK
Word to be analyzed: log(z+siny)]}}/(x+y)*[z*cos(x/z)]= at state 3,1
----Stack's contents----
4,2
4,2
1,4

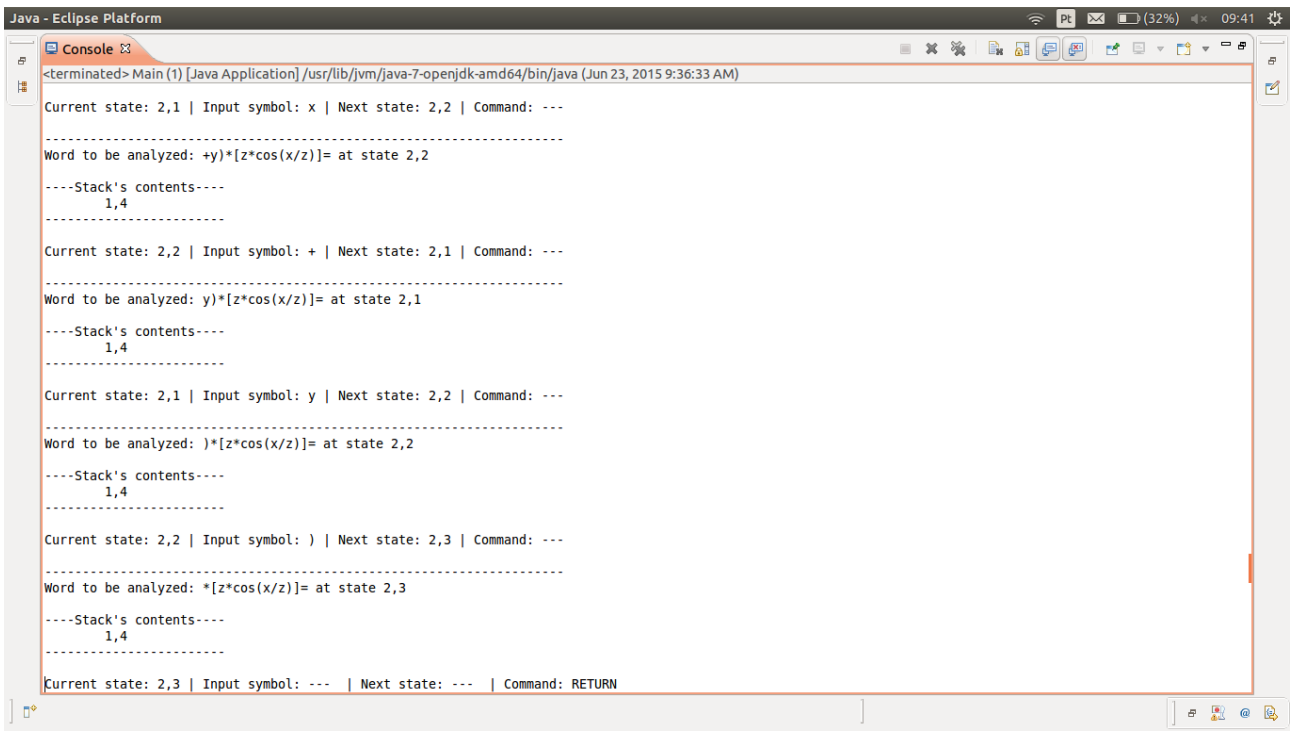

Java - Eclipse Platform
09:40
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 9:36:33 AM)
Current state: 3,1 | Input symbol: log | Next state: 3,1 | Command: ---
Word to be analyzed: (z+siny)]}}/(x+y)*[z*cos(x/z)]= at state 3,1
----Stack's contents----
4,2
4,2
1,4
Current state: 3,1 | Input symbol: (| Next state: 3,4 | Command: ---
Word to be analyzed: z+siny)]}}/(x+y)*[z*cos(x/z)]= at state 3,4
----Stack's contents----
4,2
4,2
1,4
Current state: 3,4 | Input symbol: --- | Next state: 2,1 | Command: STACK
Word to be analyzed: z+siny)]}}/(x+y)*[z*cos(x/z)]= at state 2,1
----Stack's contents----
3,2
4,2
4,2
1,4
Current state: 2,1 | Input symbol: z | Next state: 2,2 | Command: ---
Word to be analyzed: +siny)]}}/(x+y)*[z*cos(x/z)]= at state 2,2

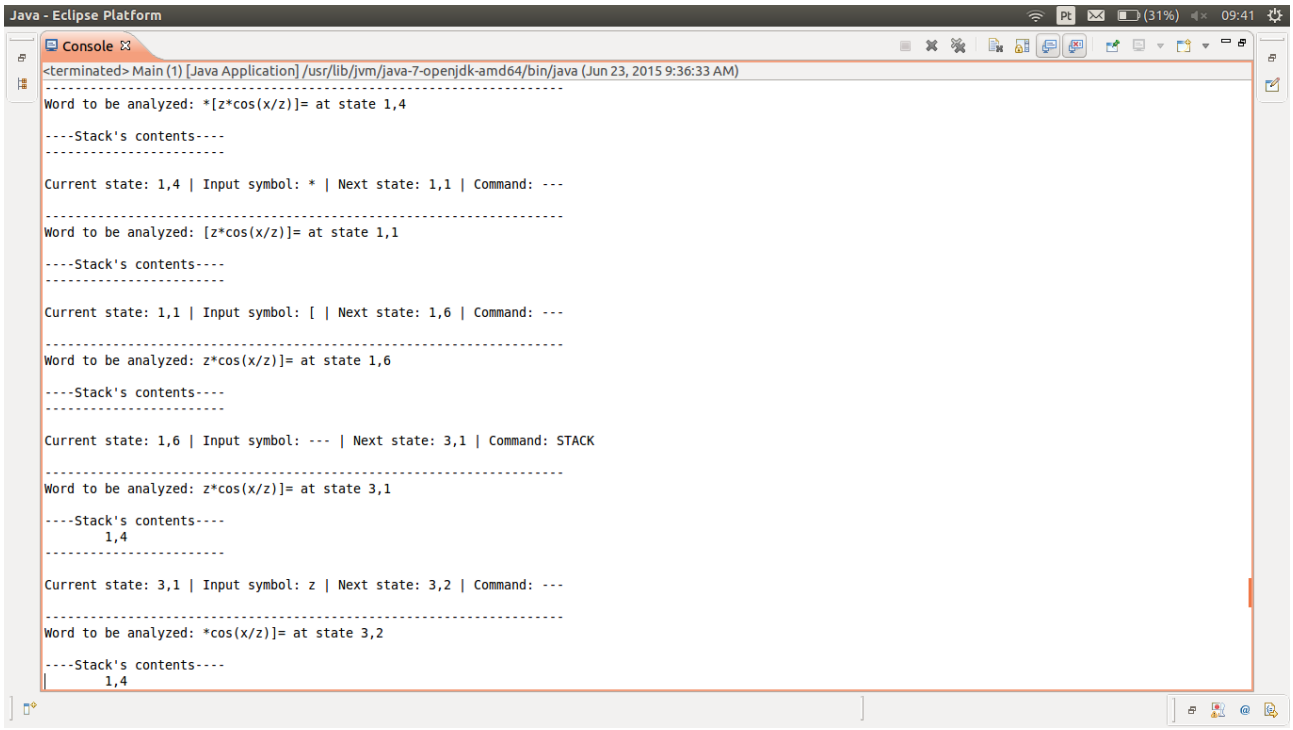

Java - Eclipse Platform
09:41
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 9:36:33 AM)
Word to be analyzed: +siny)]}}/(x+y)*[z*cos(x/z)]= at state 2,2
----Stack's contents----
3,2
4,2
4,2
1,4
Current state: 2,2 | Input symbol: + | Next state: 2,1 | Command: ---
Word to be analyzed: siny)]}}/(x+y)*[z*cos(x/z)]= at state 2,1
----Stack's contents----
3,2
4,2
4,2
1,4
Current state: 2,1 | Input symbol: sin | Next state: 2,1 | Command: ---
Word to be analyzed: y)]}}/(x+y)*[z*cos(x/z)]= at state 2,1
----Stack's contents----
3,2
4,2
4,2
1,4
Current state: 2,1 | Input symbol: y | Next state: 2,2 | Command: ---
Word to be analyzed:)]}}/(x+y)*[z*cos(x/z)]= at state 2,2

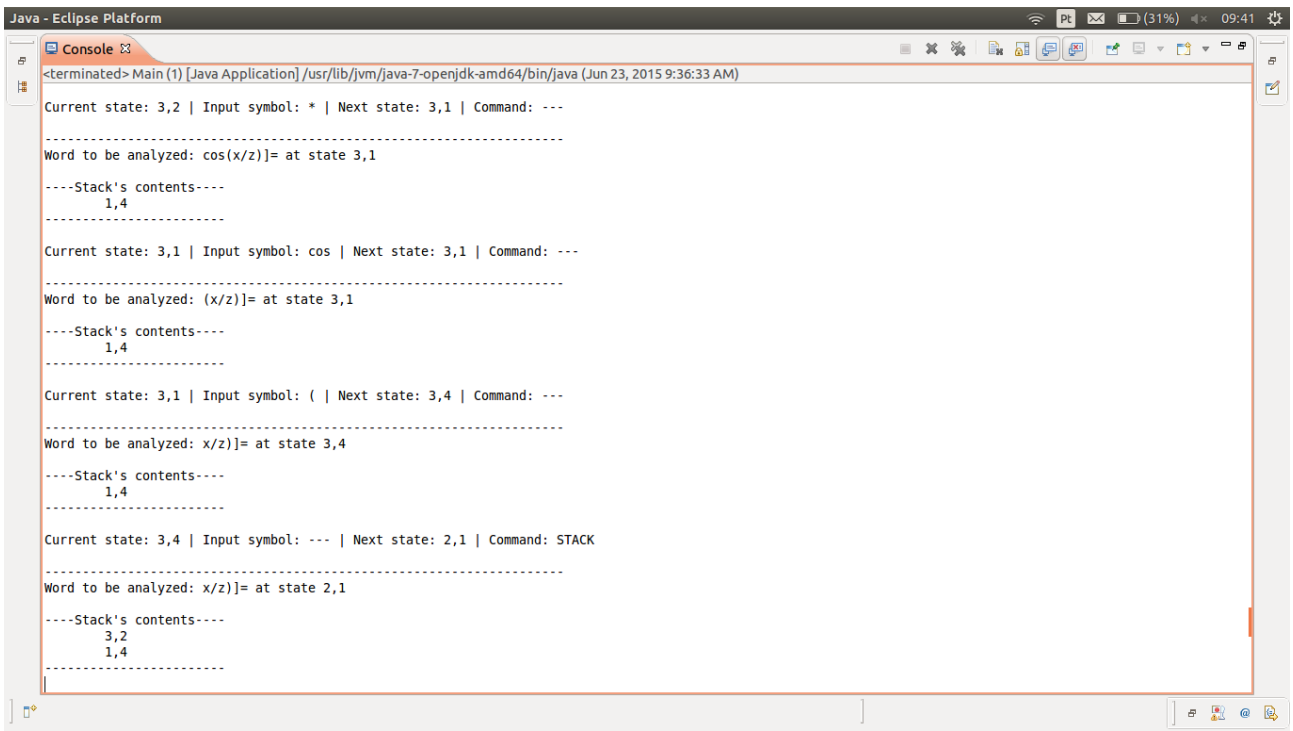

Java - Eclipse Platform
09:41
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 9:36:33 AM)
Word to be analyzed:)]}}/(x+y)*[z*cos(x/z)]= at state 2,2
----Stack's contents----
3,2
4,2
4,2
1,4
Current state: 2,2 | Input symbol:) | Next state: 2,3 | Command: ---
Word to be analyzed:]}}/(x+y)*[z*cos(x/z)]= at state 2,3
----Stack's contents----
3,2
4,2
4,2
1,4
Current state: 2,3 | Input symbol: --- | Next state: --- | Command: RETURN
Word to be analyzed:]}}/(x+y)*[z*cos(x/z)]= at state 3,2
----Stack's contents----
4,2
4,2
1,4
Current state: 3,2 | Input symbol:] | Next state: 3,3 | Command: ---
Word to be analyzed: }}/(x+y)*[z*cos(x/z)]= at state 3,3
----Stack's contents----

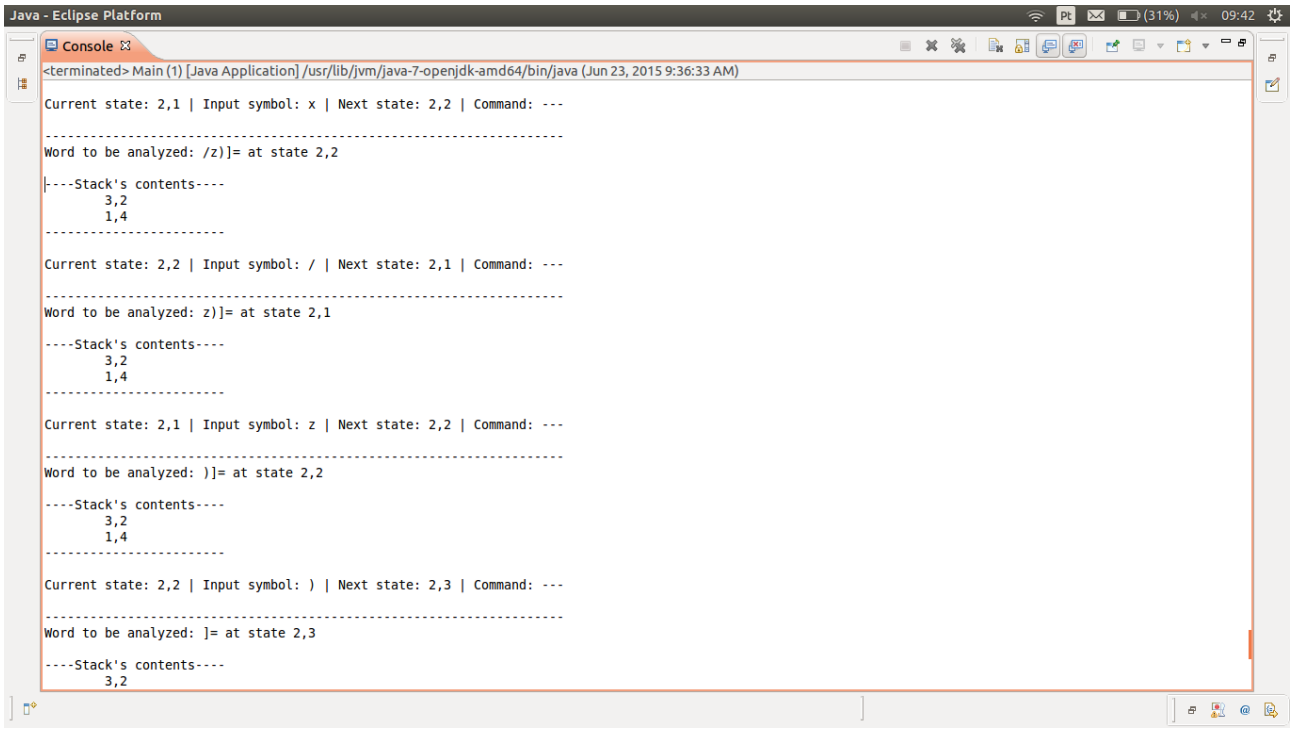

Java - Eclipse Platform
09:41
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 9:36:33 AM)
----Stack's contents----
4,2
4,2
1,4
Current state: 3,3 | Input symbol: --- | Next state: --- | Command: RETURN
Word to be analyzed: }}/(x+y)*[z*cos(x/z)]= at state 4,2
----Stack's contents----
4,2
1,4
Current state: 4,2 | Input symbol: } | Next state: 4,3 | Command: ---
Word to be analyzed: }/(x+y)*[z*cos(x/z)]= at state 4,3
----Stack's contents----
4,2
1,4
Current state: 4,3 | Input symbol: --- | Next state: --- | Command: RETURN
Word to be analyzed: }/(x+y)*[z*cos(x/z)]= at state 4,2
----Stack's contents----
1,4
Current state: 4,2 | Input symbol: } | Next state: 4,3 | Command: ---

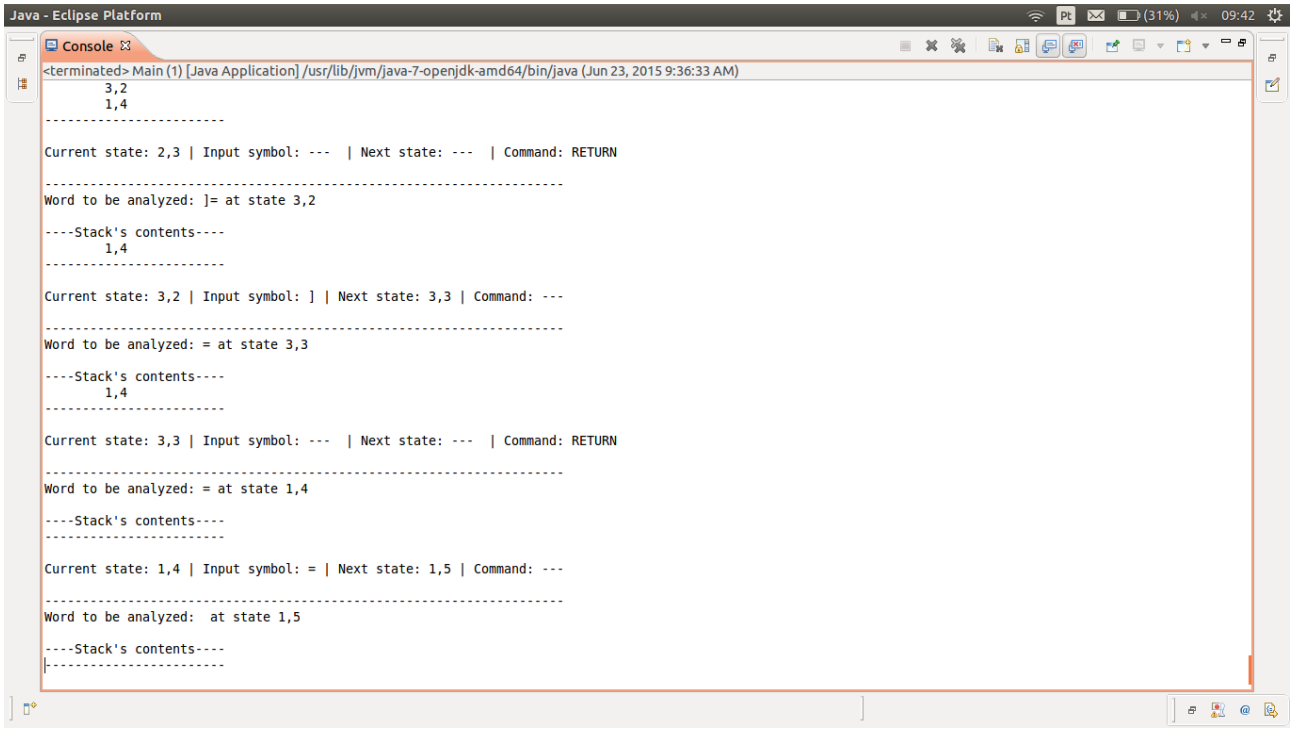

Java - Eclipse Platform
09:41
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 9:36:33 AM)
Word to be analyzed: /(x+y)*[z*cos(x/z)]= at state 4,3
----Stack's contents----
1,4
Current state: 4,3 | Input symbol: --- | Next state: --- | Command: RETURN
Word to be analyzed: /(x+y)*[z*cos(x/z)]= at state 1,4
----Stack's contents----
Current state: 1,4 | Input symbol: / | Next state: 1,1 | Command: ---
Word to be analyzed: (x+y)*[z*cos(x/z)]= at state 1,1
----Stack's contents----
Current state: 1,1 | Input symbol: (| Next state: 1,3 | Command: ---
Word to be analyzed: x+y)*[z*cos(x/z)]= at state 1,3
----Stack's contents----
Current state: 1,3 | Input symbol: --- | Next state: 2,1 | Command: STACK
Word to be analyzed: x+y)*[z*cos(x/z)]= at state 2,1
----Stack's contents----
1,4

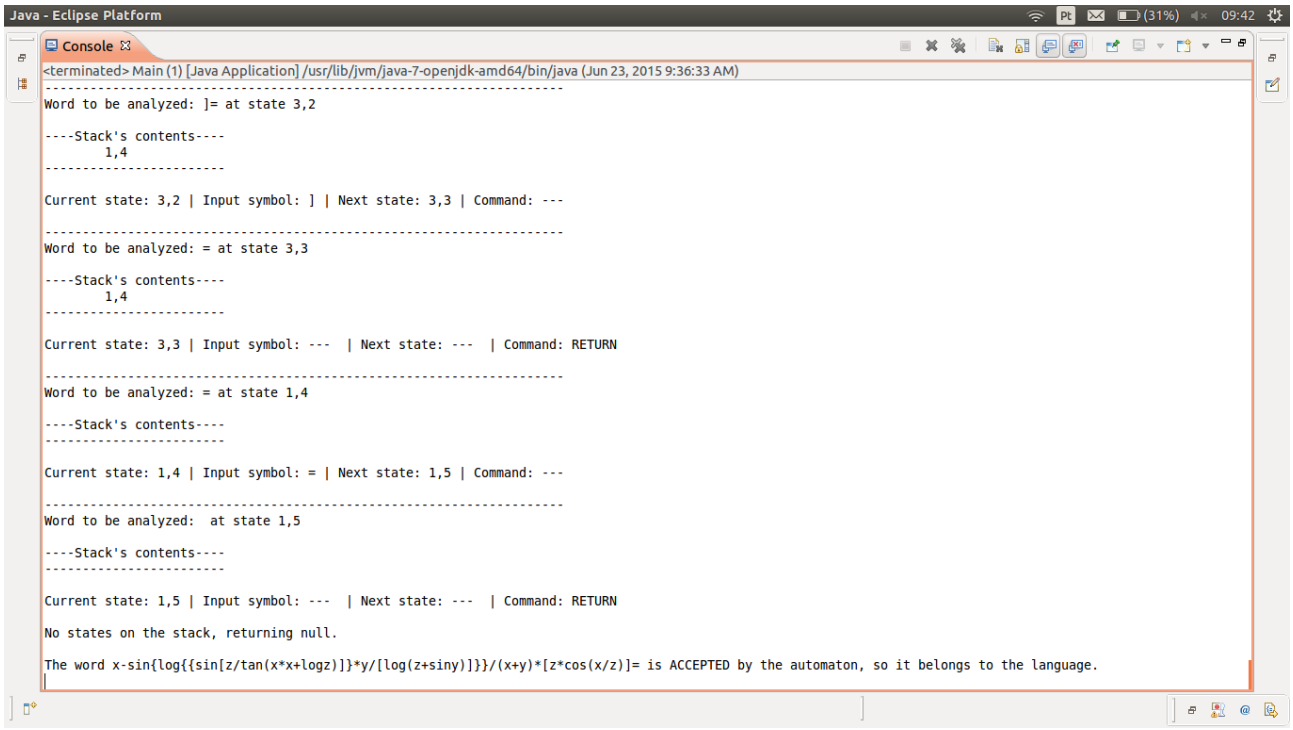

Java - Eclipse Platform
09:41
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 9:36:33 AM)
Current state: 2,1 | Input symbol: x | Next state: 2,2 | Command: ---
Word to be analyzed: +y)*[z*cos(x/z)]= at state 2,2
----Stack's contents----
1,4
Current state: 2,2 | Input symbol: + | Next state: 2,1 | Command: ---
Word to be analyzed: y)*[z*cos(x/z)]= at state 2,1
----Stack's contents----
1,4
Current state: 2,1 | Input symbol: y | Next state: 2,2 | Command: ---
Word to be analyzed:)*[z*cos(x/z)]= at state 2,2
----Stack's contents----
1,4
Current state: 2,2 | Input symbol:) | Next state: 2,3 | Command: ---
Word to be analyzed: *[z*cos(x/z)]= at state 2,3
----Stack's contents----
1,4
Current state: 2,3 | Input symbol: --- | Next state: --- | Command: RETURN


Java - Eclipse Platform
09:41
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 9:36:33 AM)
Word to be analyzed: *[z*cos(x/z)]= at state 1,4
----Stack's contents----
Current state: 1,4 | Input symbol: * | Next state: 1,1 | Command: ---
Word to be analyzed: [z*cos(x/z)]= at state 1,1
----Stack's contents----
Current state: 1,1 | Input symbol: [| Next state: 1,6 | Command: ---
Word to be analyzed: z*cos(x/z)]= at state 1,6
----Stack's contents----
Current state: 1,6 | Input symbol: --- | Next state: 3,1 | Command: STACK
Word to be analyzed: z*cos(x/z)]= at state 3,1
----Stack's contents----
1,4
Current state: 3,1 | Input symbol: z | Next state: 3,2 | Command: ---
Word to be analyzed: *cos(x/z)]= at state 3,2
----Stack's contents----
1,4

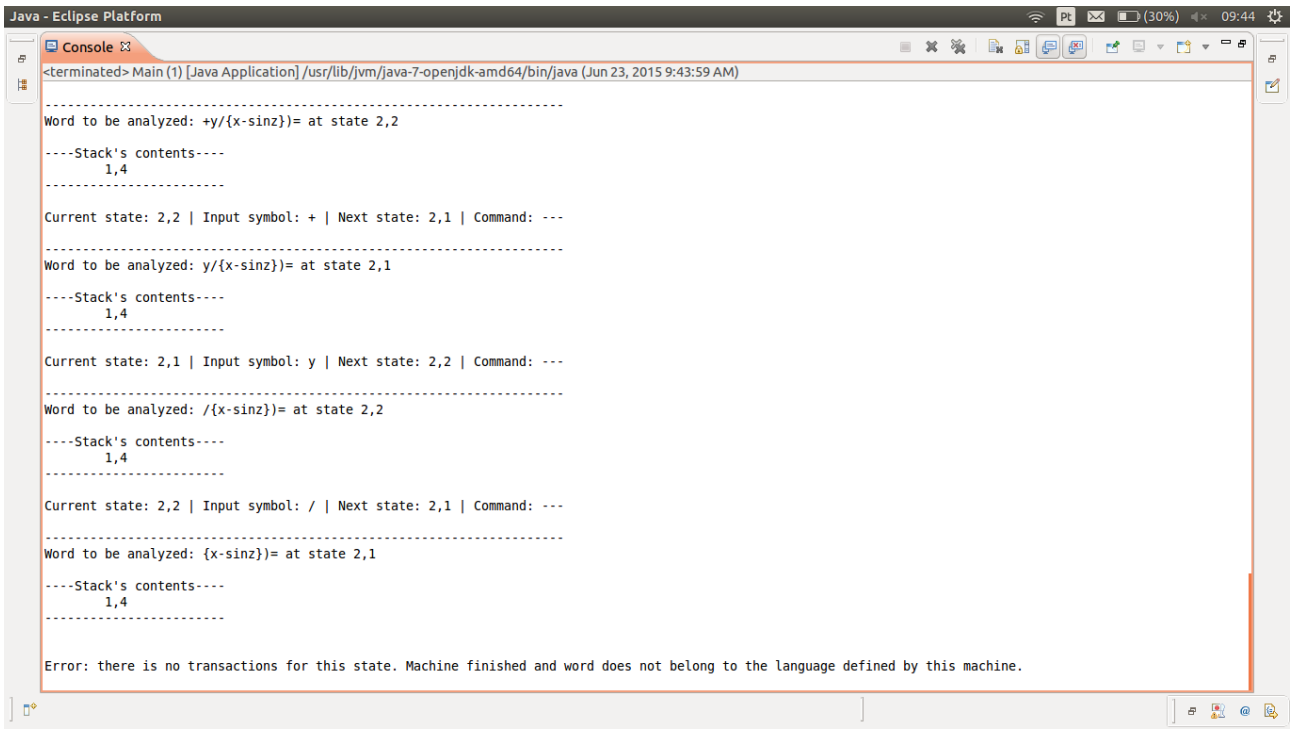

Java - Eclipse Platform
09:41
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 9:36:33 AM)
Current state: 3,2 | Input symbol: * | Next state: 3,1 | Command: ---
Word to be analyzed: cos(x/z)]= at state 3,1
----Stack's contents----
1,4
Current state: 3,1 | Input symbol: cos | Next state: 3,1 | Command: ---
Word to be analyzed: (x/z)]= at state 3,1
----Stack's contents----
1,4
Current state: 3,1 | Input symbol: (| Next state: 3,4 | Command: ---
Word to be analyzed: x/z)]= at state 3,4
----Stack's contents----
1,4
Current state: 3,4 | Input symbol: --- | Next state: 2,1 | Command: STACK
Word to be analyzed: x/z)]= at state 2,1
----Stack's contents----
3,2
1,4


Java - Eclipse Platform
09:42
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 9:36:33 AM)
Current state: 2,1 | Input symbol: x | Next state: 2,2 | Command: ---
Word to be analyzed: /z)]= at state 2,2
----Stack's contents----
3,2
1,4
Current state: 2,2 | Input symbol: / | Next state: 2,1 | Command: ---
Word to be analyzed: z)]= at state 2,1
----Stack's contents----
3,2
1,4
Current state: 2,1 | Input symbol: z | Next state: 2,2 | Command: ---
Word to be analyzed:)]= at state 2,2
----Stack's contents----
3,2
1,4
Current state: 2,2 | Input symbol:) | Next state: 2,3 | Command: ---
Word to be analyzed:]= at state 2,3
----Stack's contents----
3,2


Java - Eclipse Platform
09:42
Console
<terminated> Main (1) [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Jun 23, 2015 9:36:33 AM)
3,2
1,4
Current state: 2,3 | Input symbol: --- | Next state: --- | Command: RETURN
Word to be analyzed:]= at state 3,2
----Stack's contents----
1,4
Current state: 3,2 | Input symbol:] | Next state: 3,3 | Command: ---
Word to be analyzed: = at state 3,3
----Stack's contents----
1,4
Current state: 3,3 | Input symbol: --- | Next state: --- | Command: RETURN
Word to be analyzed: = at state 1,4
----Stack's contents----
Current state: 1,4 | Input symbol: = | Next state: 1,5 | Command: ---
Word to be analyzed: at state 1,5
----Stack's contents----



3. x*sin(x+y/{x-sinz})=: não aceito, não respeita a ordem dos agrupamentos; a seguir mostra-se apenas o instante em que ocorre erro.



4. tan[z/(x+y]=: não aceito, o parênteses não é fechado