UNICAMP
Universidade Estadual de Campinas

FEEC
Faculdade de Engenharia Elétrica e de Computação

Aluno: Mateus Neves Barreto
R.A.: 142358
Disciplina: IA006
Professor: Ricardo R. Gudwin

Relatório – Aula 2

1 Atividade 1: Instalando o SOAR

A realização desta atividade foi bem simples, realizar o download e instalação do SOAR.

2 Atividade 2: Construindo um Agente SOAR Simples usando Regras

2.1 Criando um Agente SOAR e Utilizando o SOAR Debugger

Nesta etapa, foi apenas o entendimento e execução da aplicação "SOAR Debugger" que é chamada pela linha de comando "./SoarJavaDebugger.sh".

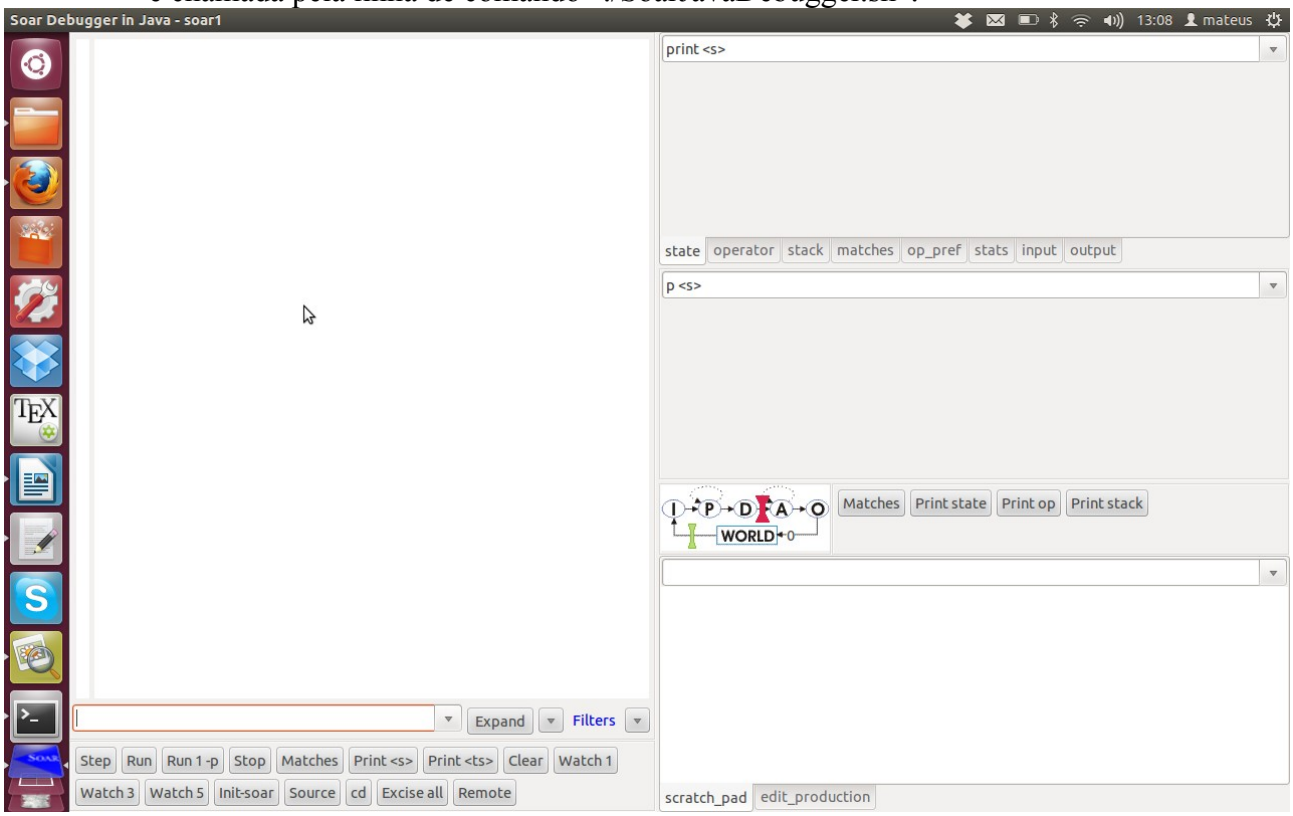


Figura 1 – Entendendo SOAR Debugger.

2.2 Executando o Hello World do SOAR

Para a criação do Hello World, foi utilizado o seguinte código simples:

```
sp {hello-world
   (state <s> ^type state)
-->
   (write |Hello World|)
   (halt)
}
```

A saída da execução deste código pode ser observada na Figura 2:

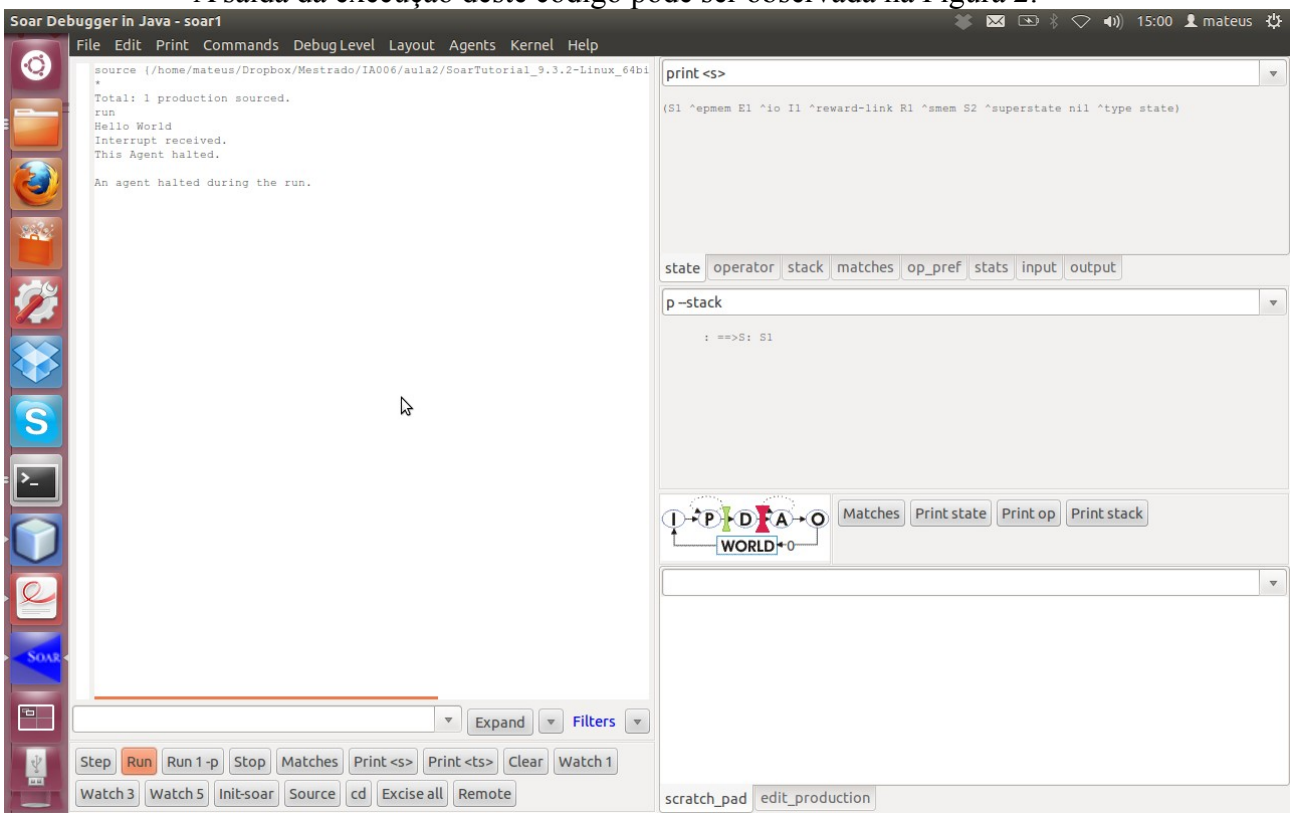


Figura 2 – Executando um Hello World no SOAR Debugger.

2.3 Utilizando a Memória de Trabalho

A memória de trabalho possui todas as informações dinâmicas de um agente SOAR. Toda a memória SOAR é organizada em forma de grafo, onde cada nó é representado por um estado possível.

O mecanismo de entrada e saída de dados do SOAR funciona através 3 atributos ligados em hierarquia como na figura a seguir. Para entrada de dados, existe o "input-link" ligado ao estado do grafo I2, onde as informações dos agentes estão disponíveis na memória de trabalho. Para a saída (ou mudança de estado), os comandos de ação dos agentes são direcionados para o "output-link". Deste modo, utilizando o "output-link" e "input-link", é possível modificar estados e criar soluções para problemas. Na Figura 3, pode-se observar a estrutura básica de papéis e operadores do SOAR, encontrando os estados I1, I2 e I3, que representam e realizando as tarefas de entrada e saída no SOAR.

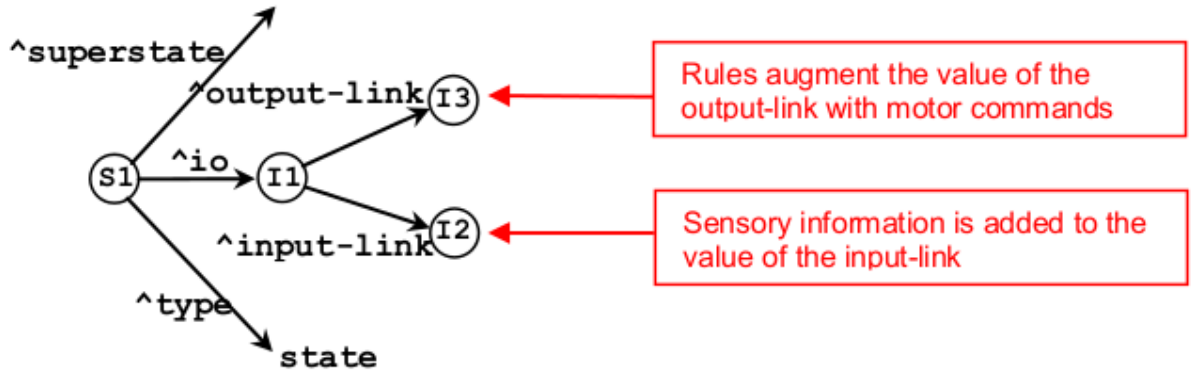


Figura 3 – Estrutura básica do SOAR.

3 Atividade 3: Agentes Simples utilizando Operadores

3.1 Executando o Hello World com Operadores

Para a criação do Hello World utilizando agentes e operadores, o seguinte código foi utilizado:

```
sp {propose*helloWorldAgente
  (state <s> ^type state)
-->
  (<s> ^operator <o> +)
  (<o> ^name helloWorldAgente)
}
sp {apply*helloWorldAgente
  (state <s> ^operator <op>)
  (<op> ^name helloWorldAgente)
-->
  (write |Hello World!|)
  (halt)
}
```

As imagens a seguir (Figura 4 e 5) ilustram as etapas seguidas para o desenvolvimento, execução e análise da tarefa descrita.

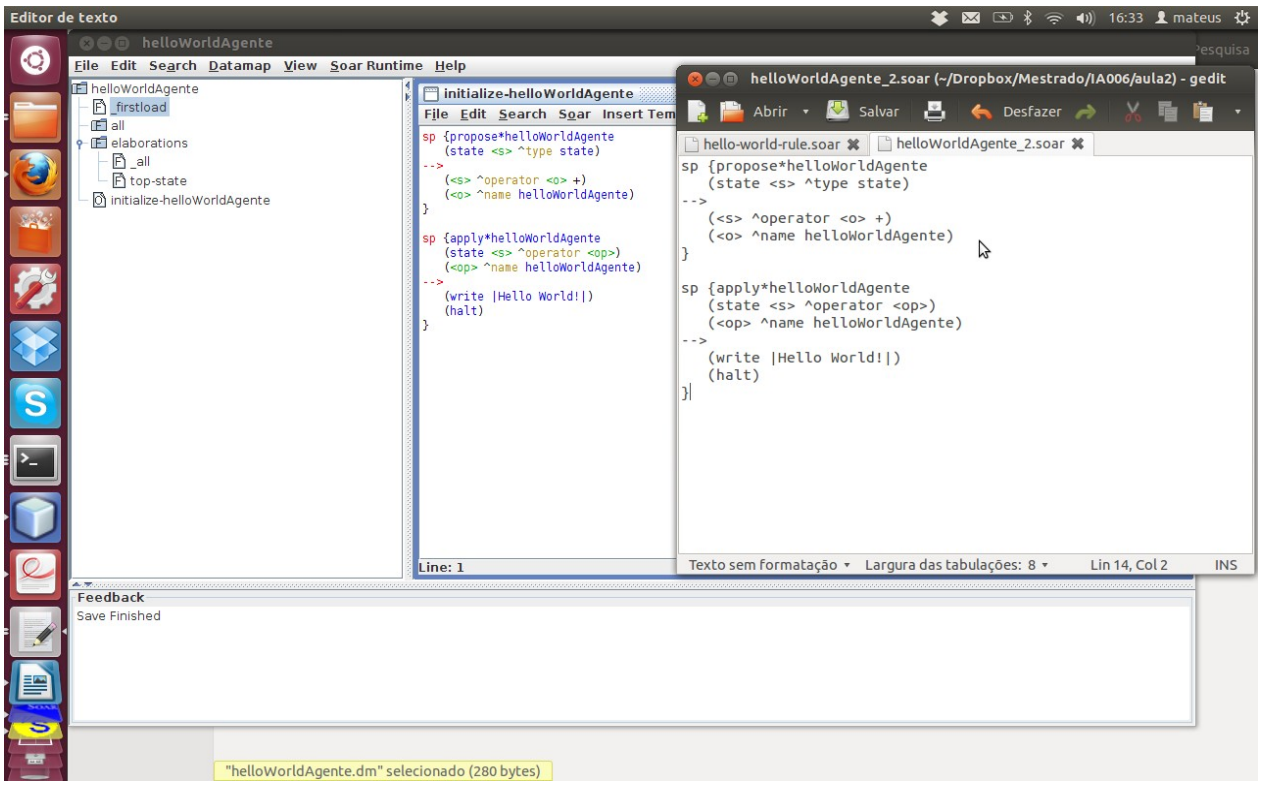


Figura 4 – Criando um Hello World com auxílio do VisualSoar.

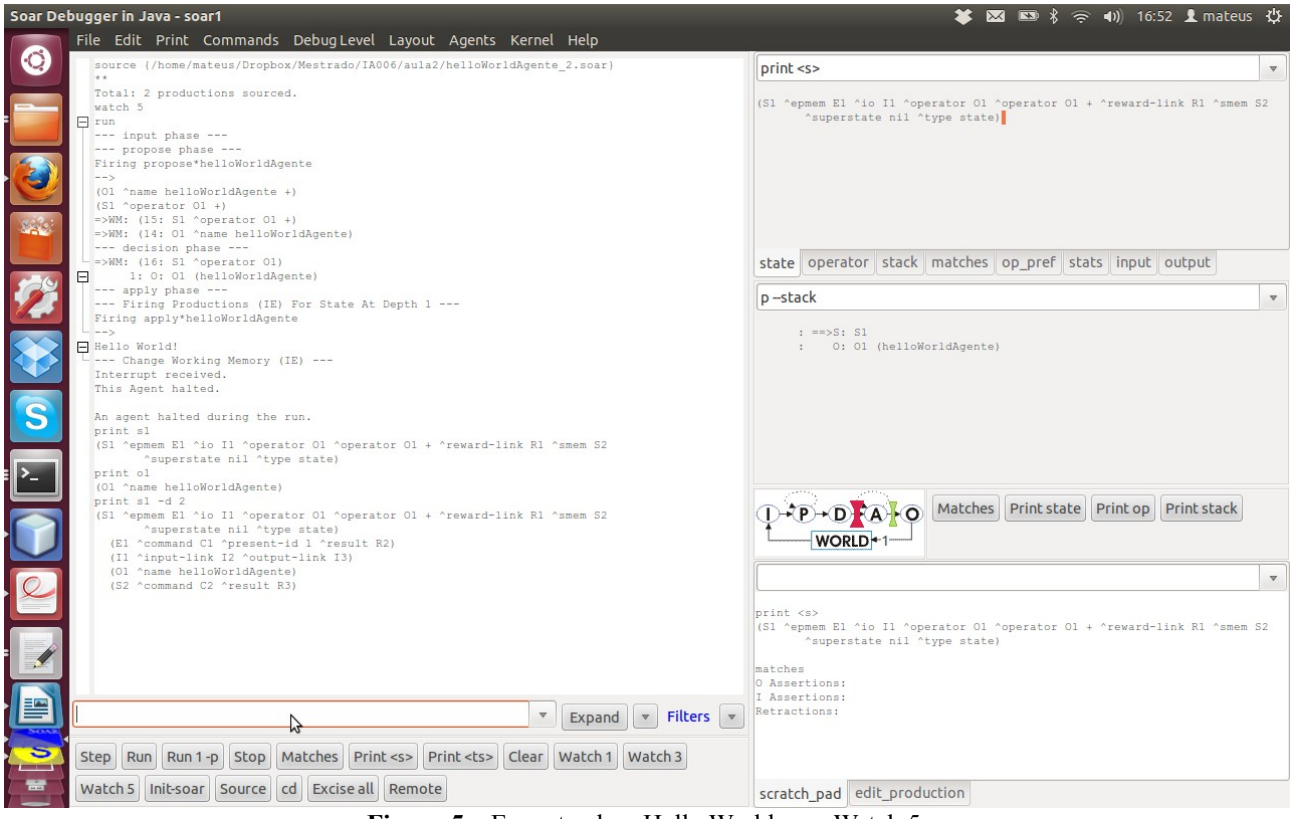


Figura 5 – Executando o Hello World com Watch 5.

3.2 Examinando a Memória de Trabalho

Os operadores são criados a partir de proposições e armazenados na memória de trabalho. Em seguida as regras que executam as propostas, regras "apply", procura na memória de trabalho a existência de uma condição que coincida com suas restrições. No caso específico deste subtópico, uma regra de "propose" criou um operador com o nome "helloWorldAgente" na memória e a regra "apply" procurou por um operador com o mesmo nome na memória de trabalho. Como a existência

da condição foi verdadeira, a regra “apply” pode executar sua ação, ou seja, pode escrever na saída a expressão “Hello World!”. A Figura 6 pode auxiliar a visualisação desta análise.

Figura 6 – Analisando a memória de trabalho.

3.3 Utilizando o Visual SOAR

A ferramenta VisualSoar é uma ferramenta de auxílio a programação da linguagem SOAR. A utilização da mesma, pelo usuário programador, trás vantagens como:

- Organização do código;
- Advertências ao erros;
- Estruturação do código;
- Ferramentas de pesquisa;
- Criação de pré código, templates e outros.

A Figura 7 apresenta a ferramenta em execução.

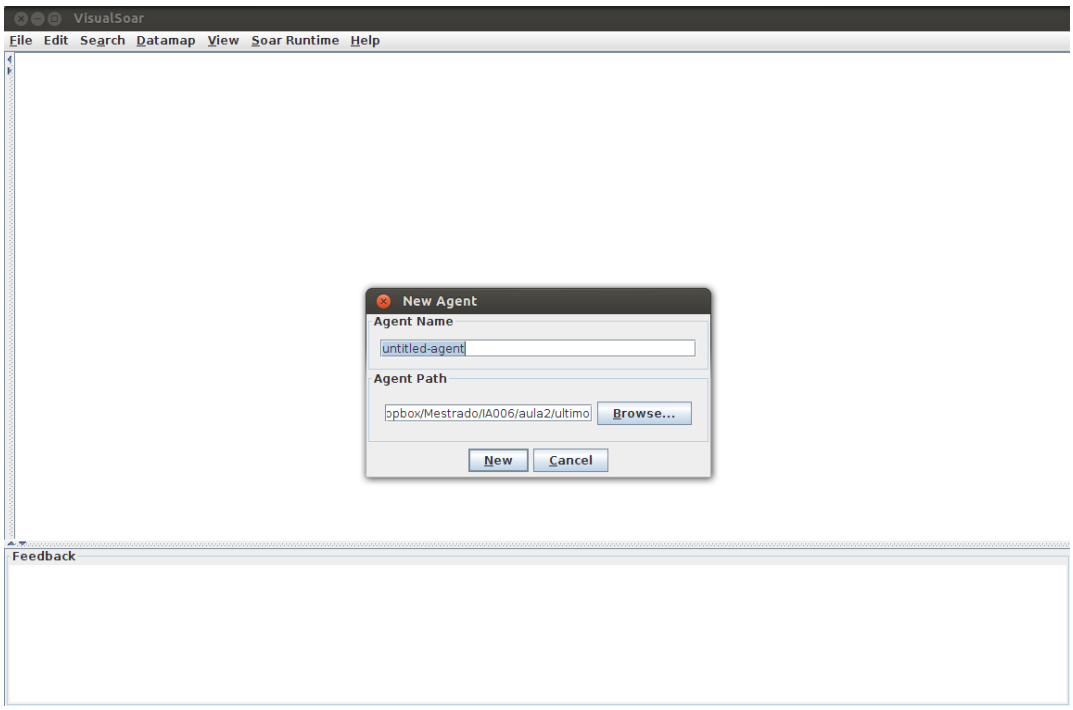


Figura 7 – Criando novo projeto pelo VisualSoar.

4 Atividade 4: Criando um Agente para o Water Jug Problem

A criação do projeto water-jug foi realizada via VisualSoar. A figura 8 apresenta a criação do projeto desta atividade.

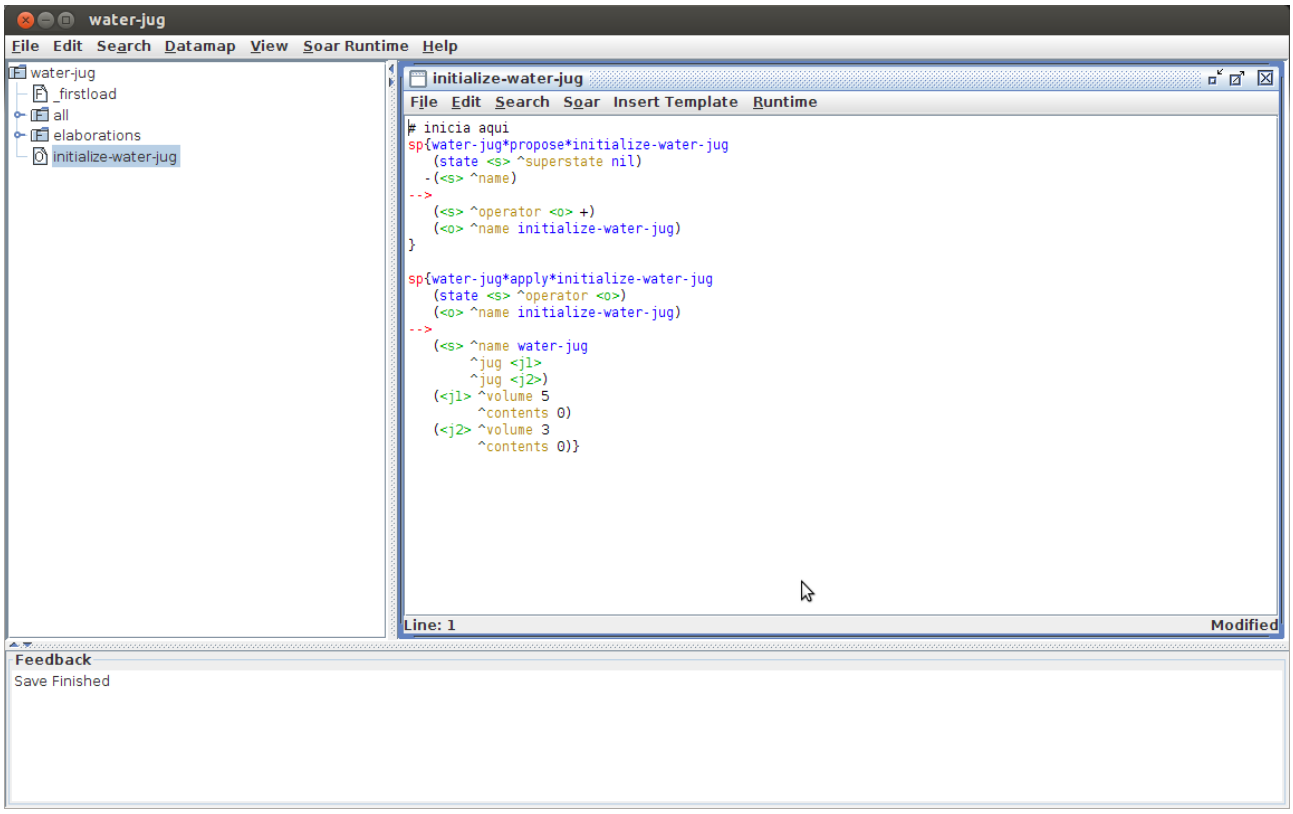


Figura 8 – Criando o projeto water-jug.

4.1 Elaboração de Estado

A estratégia de elaboração de estado, consiste em calcular a capacidade de volume que o jarro ainda pode receber. Esse calculo é realizado subtraindo a capacidade de volume que o jarro

suporta pelo conteúdo que o jarro possui. Podendo ser expressa pela Equação (1).

$$empty = volume - contents \quad (1)$$

Onde *volume* é a capacidade do jarro e *contents* é o volume de água que o jarro esta portando no momento do cálculo. A aplicação do código de elaboração pode ser notada na Figura 9.

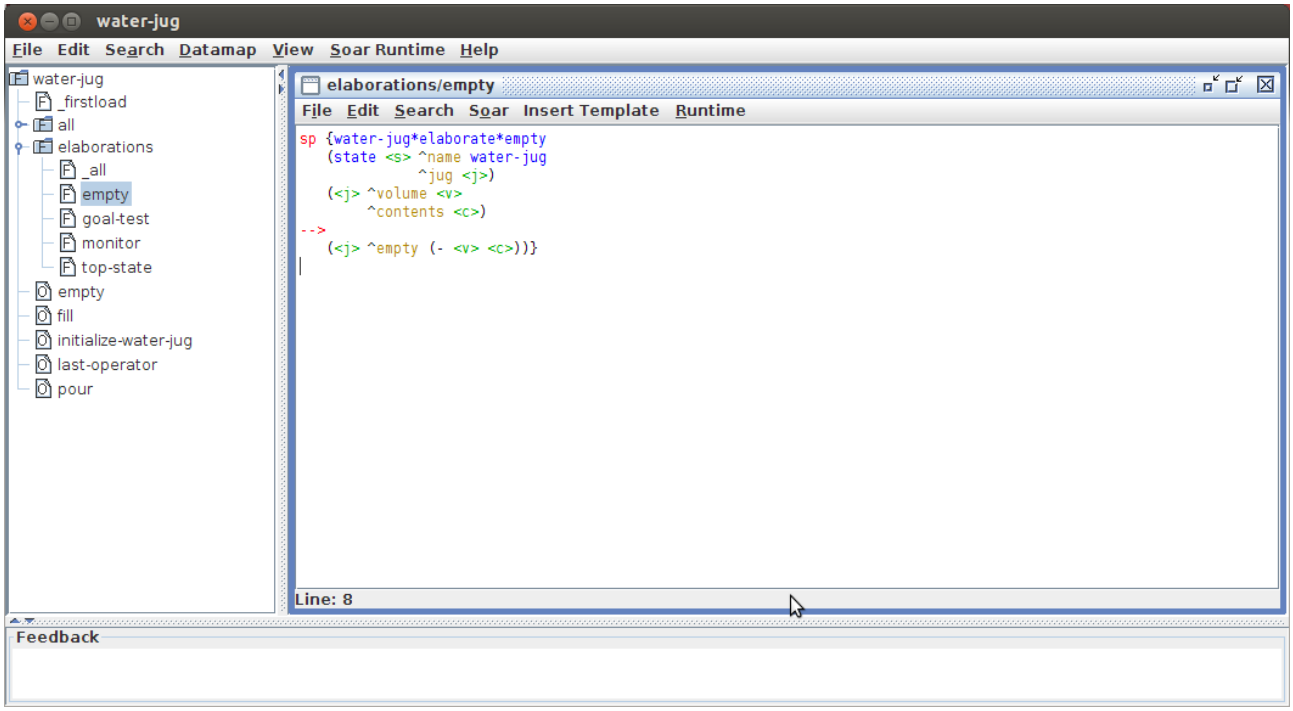
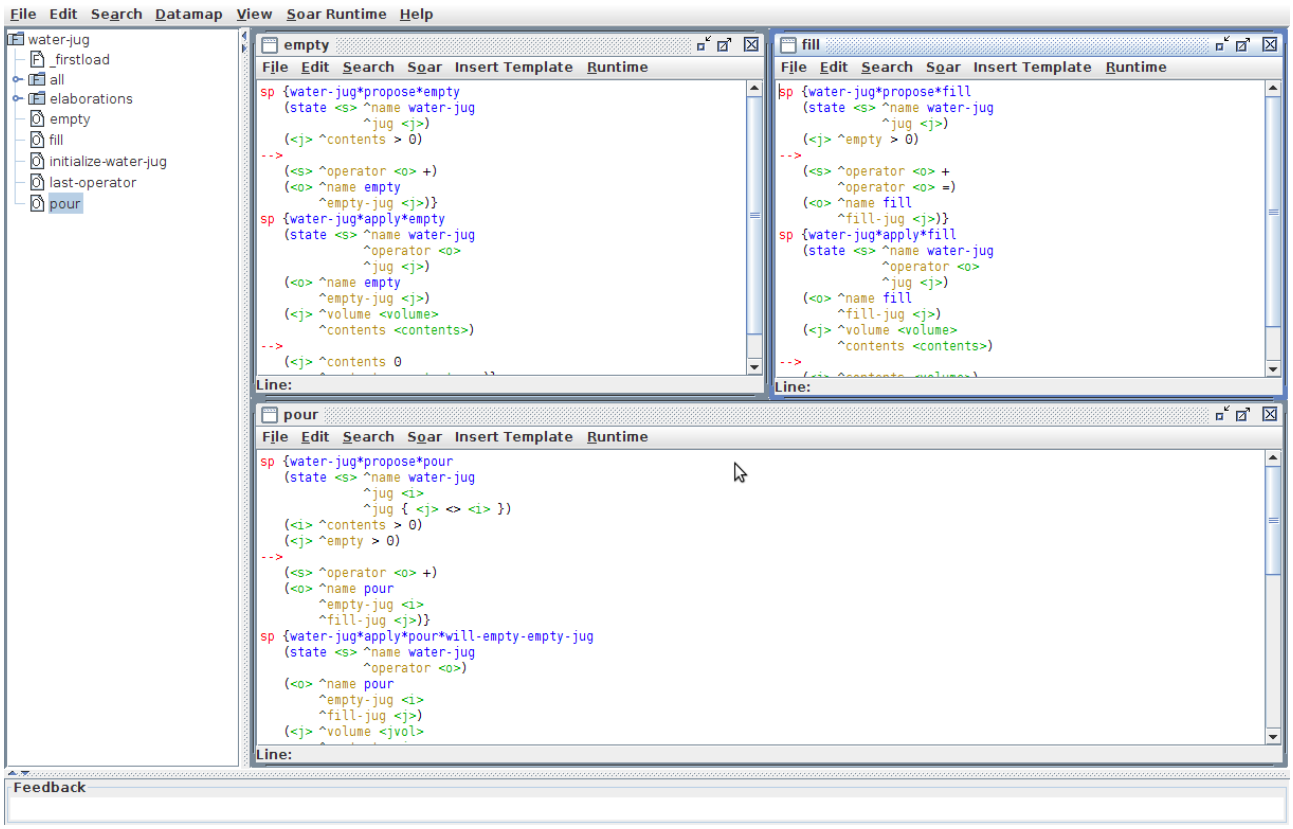


Figura 9 – Código elaborate/empty.

4.2 Proposição de Operadores



4.3 Aplicação de Operadores

A aplicação de operadores pode ser notada na Figura 10.

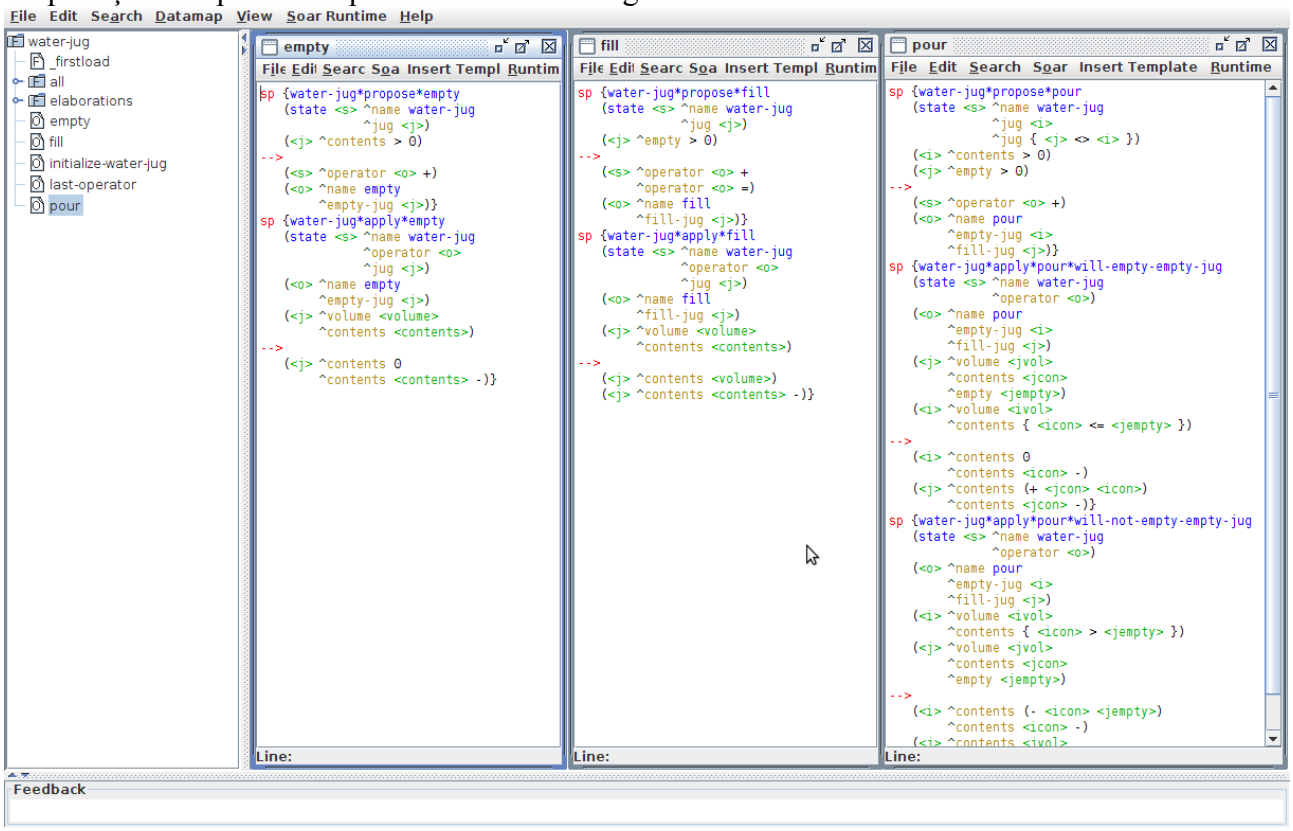


Figura 10 – Código aplicação de operadores.

4.4 Monitoramento de Estados e Operadores

O código de monitoramento de estados e operadores pode ser observado na Figura 11.

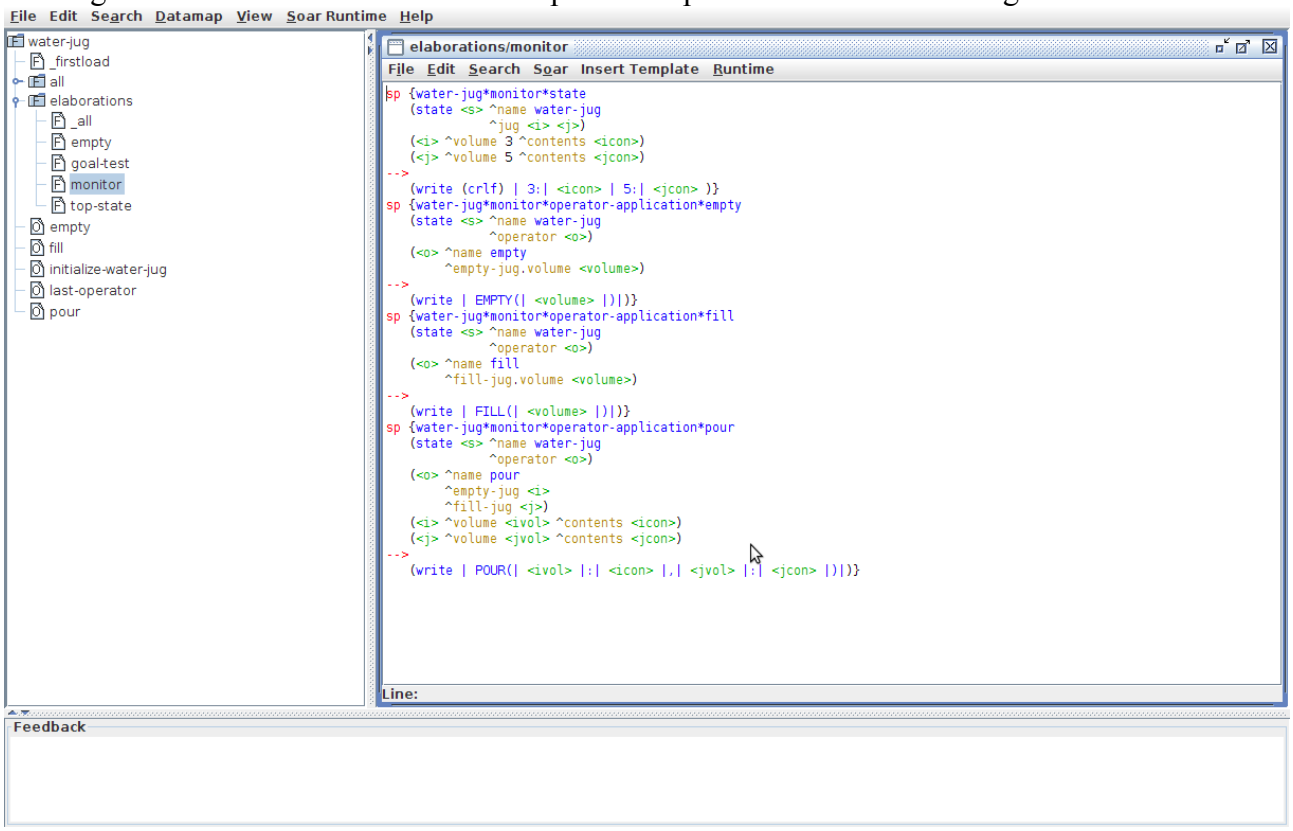


Figura 11 – Código monitoramento.

4.5 Reconhecimento do Estado Desejado

O código de reconhecimento de estado pode ser encontrado na Figura 12.

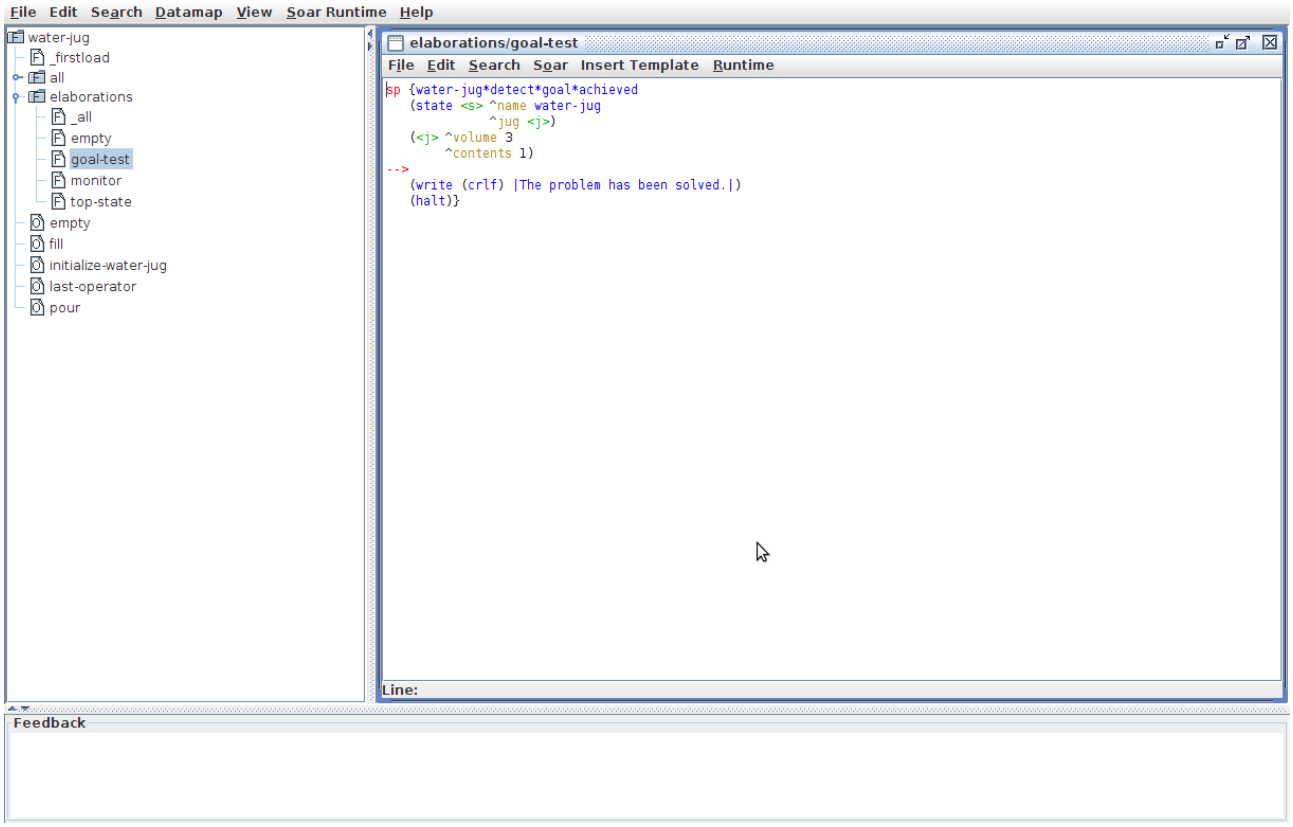


Figura 12 – Código de reconhecimento de estado.

4.6 Controle de Busca

Para realizar uma busca de modo mais eficiente, a alternativa tomada no tutorial foi a gravação da última operação realizada. Com esse mecanismo, foi possível evitar loops como: encher o vazo 3 e na próxima execução esvaziar o vazo 3. Essa gravação possibilita que o programa SOAR lembre de sua última decisão. A altenativa para a realização desta gravação foi a criação de um tipo de operador na memória de trabalho nomeado como last-operator. Do mesmo modo que foi armazenado o último, se fosse necessário poderia ser criado mais regras para o armazenamento do penúltimo e assim sucessivamente, onde cada um teria de ser identificado como um nome diferente. A realização desta tarefa pode ser observada na Figura 13.

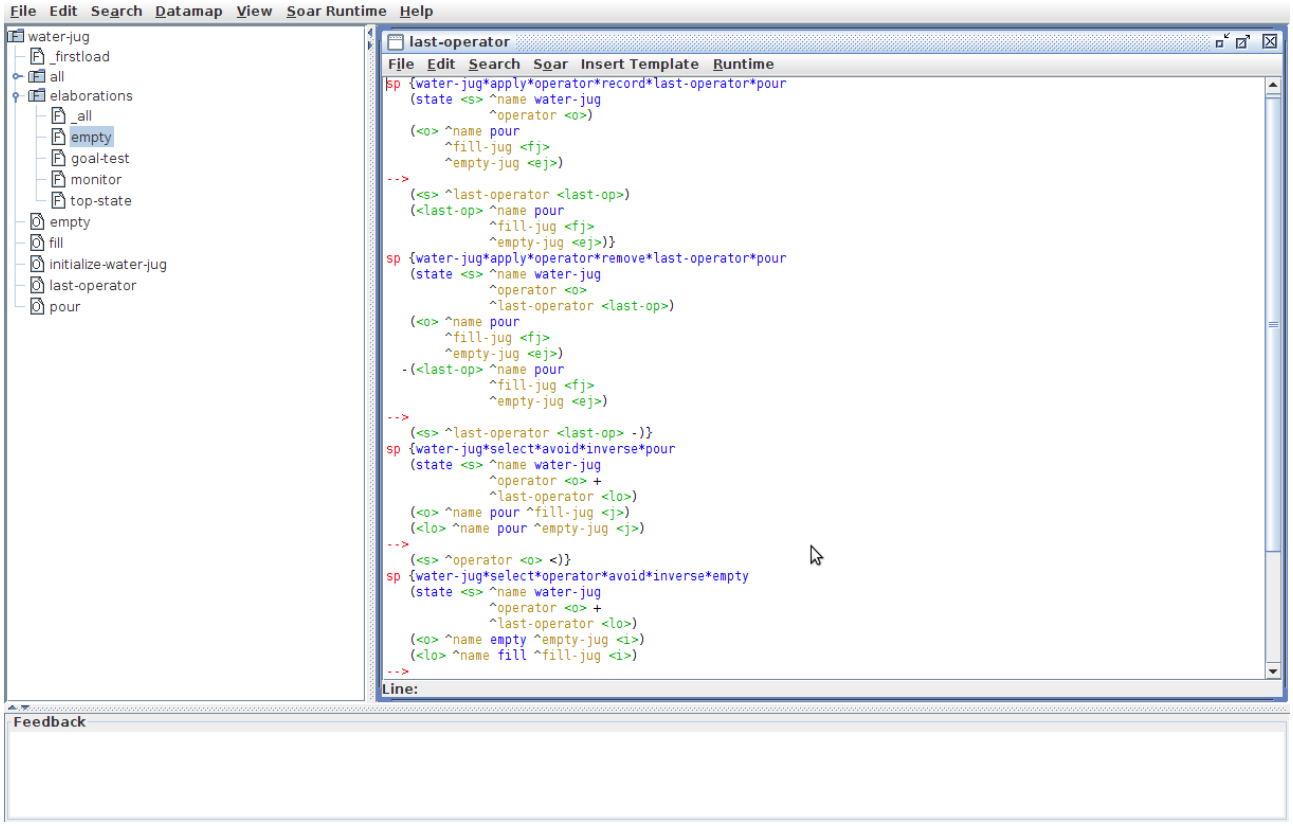


Figura 13 – Código do controle de busca.

5 Conclusão

A linguagem SOAR tem a vantagem sobre as outras de ser mais elaborada em relação a uma linguagem como prolog. A vantagem em relação a um sistema especialista, é que para adicionar novas regras não se encontra muita dificuldade depois que a aplicação estiver em execução.