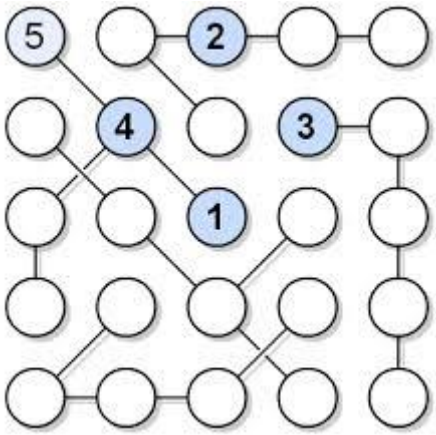
Departamento de Informática
Licenciatura em Engenharia Informática

Programação Declarativa



Strimko em prolog

Daniel Gonçalo Jesus Ramos, 29423
Bernardo Santos, 28958

Évora 2014

Conteúdo

ÍNDICE DE FIGURAS

INTRODUÇÃO

Neste trabalho pretende-se implementar em prolog um jogo de puzzles lógico chamado Strimko, que é uma variação do jogo Sudoku em que no Sudoku temos sub-quadrados e no Strimko temos streams.

Neste jogo existem três regras que tem que ser respeitadas para que a solução do puzzle seja considerada válida. A primeira regra indica que cada linha do puzzle tem que conter números diferentes. A segunda regra indica que cada coluna tem que conter números diferentes. Por último, além das duas regras anteriores, as streams do jogo, isto é, os nós (posições na matriz) que estão ligados entre si como se de um grafo tratasse, tem que conter números diferentes também, aumentando o grau de dificuldade.

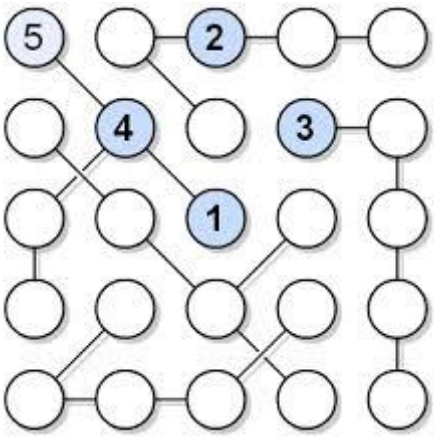


Figura 1: Exemplo da matriz

Além das regras implementadas, o tamanho do puzzle pode variar entre 2x2 e 9x9, sendo que o grau de dificuldade aumenta conforme o tamanho da matriz aumenta. Um exemplo de uma matriz pode ser visualizada na figura 1.

DESENVOLVIMENTO

Funcionamento geral do programa

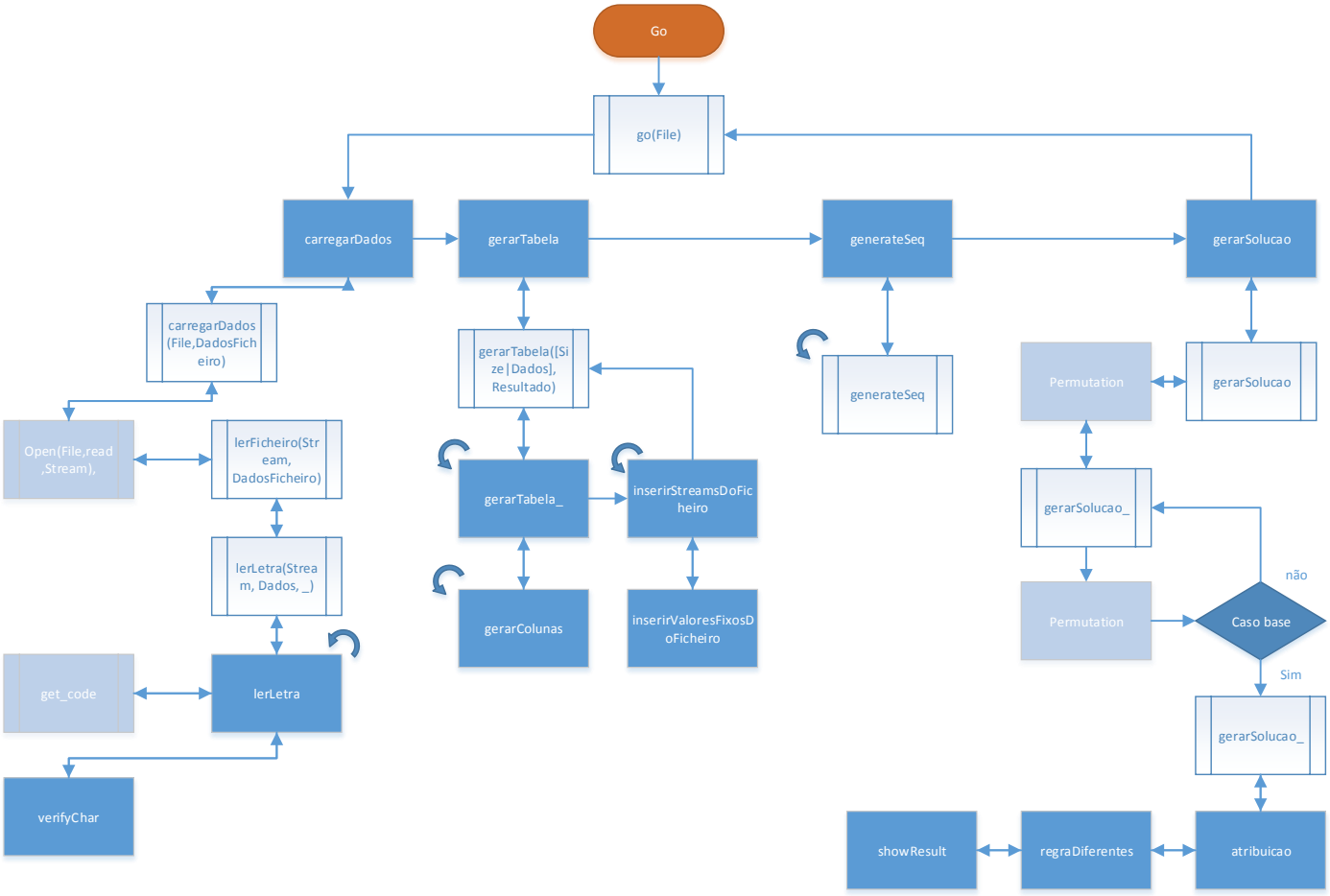


Figura 2: Fluxograma do programa

Na figura 1 está representado um fluxograma geral do programa implementado. De seguida segue uma explicação geral do fluxo do programa ao processar um puzzle. Para saber em detalhe o que cada goal faz, deve ser consultado os goals implementados mais abaixo. O fluxograma deve ser lido da esquerda para a direita.

Carregamento do ficheiro

Para iniciar o gerador de soluções, deve ser fornecido um nome de ficheiro ao goal go. O goal irá então processar os dados do ficheiro ao chamar o goal carregarDados. Para carregar os dados do ficheiro, o programa tenta abrir o ficheiro e chama o goal lerFicheiro. O goal lerFicheiro irá então chamar o goal lerLetra onde cada letra é lida de cada vez. Por cada letra lida do ficheiro, é feita uma verificação se a letra é válida ou não. Essa verificação é feita pelo goal verifyChar onde é analisado se o caracter lido contem códigos inválidos que têm que ser ignorados. Por cada letra válida, esta é adicionada a uma lista. O processamento de letras para quando não houver mais letras para processar, retornando a lista de dados lidos.

Geração do tabuleiro

Depois de ter sido lido os dados do ficheiro, é então gerada a tabela que contem os dados iniciais do tabuleiro, que é executada ao chamar o goal gerarTabela. Para gerar a tabela, é gerado N colunas linha a linha. É usado um functor com quatro argumentos em que a estrutura contem o seguinte formato:

C(Linha, Coluna, Stream, Value)

O resultado da geração de todas as colunas por linha é inserido numa lista temporária e retornada no fim do processo como resultado final.

Depois de ter sido gerado o tabuleiro com a estrutura montada, é então inserida as streams que foram lidas do ficheiro. Para tal é executado o goal inserirStreamsDoFicheiro em ao percorrer a lista que forma o tabuleiro é "extraído" a estrutura acima mencionada e é atribuído á variável na terceira posição o valor da stream correspondente. Este processo é feito até serem lidos n*n caracteres dos dados do ficheiro, em que n é o tamanho do tabuleiro.

Para finalizar a geração do tabuleiro, é posteriormente inserido os valores fixos no tabuleiro, bastando para isso usar o goal nth0 do prolog em que atribui automaticamente um X, Y, V a uma das estruturas dentro da lista se esta existir.

Com isto, fica finalizado a criação do tabuleiro, sendo que o passo seguinte será gerar as listas para gerar as permutações necessárias para encontrar uma solução se esta existir.

Geração de sequência

Apos a geração do tabuleiro é gerada uma lista que contêm uma sequência de um a N elementos, sendo N a dimensão da matriz. Esta lista é gerada e retornada no goal generateSeq.

Gerador de solução

Depois de gerada a sequência numérica é então gerada uma solução que respeite as constraints que estão presentes no ficheiro de input e inclua os números dados como pistas, no ficheiro.

O passo seguinte será então gerar P permutações da sequência ótima gerada, verificando a cada nova geração se existem elementos em comum com a permutação anterior. Caso existam elementos em comum, é gerada uma nova permutação, falhando a permutação atual, reduzindo assim o tempo de execução devido à exclusão de permutações que não respeitem as condições de valores distintos em cada coluna. Esta verificação é efetuada com o goal verificarPermutacoes.

Quando só restar uma linha do tabuleiro para ser permutada, é então criada uma nova tabela, sendo esta uma réplica da original mas com uma grande diferença, esta irá conter os valores das permutações. A criação da tabela contem os valores pré-definidos pelo ficheiro de input.

Depois de gerada a nova tabela com as permutações sem qualquer erro, é então submetida para verificação de linhas, colunas e streams. Se esta tabela passar todas as verificações, então é gerada uma solução com sucesso para o tabuleiro em questão e será feito o output da mesma.

Goals implementados

Para a realização deste trabalho foi necessário implementar vários goals. Abaixo segue uma lista desses goals com uma descrição do seu funcionamento.

Goal go

-Número de predicados: 1

-Cláusulas:

Clausula 1. Go/1 com **Termos:** (File).

-Descrição do funcionamento: Goal principal que dado um ficheiro de texto, carrega o seu conteúdo, gerando de seguida um tabuleiro com as condições predefinidas no ficheiro. É gerado uma lista com a sequência de números válida que podem ser inseridos no tabuleiro. As soluções são de seguida geradas se existirem.

Goal generateSeq

-Número de predicados: 2

-Cláusulas:

Clausula 1. generateSeq/3 com **Termos:**([], Size, Size) e como caso base.
Clausula 2. generateSeq/3 com **Termos**: ([Valor|Tail], Size, Pos)

-Descrição do funcionamento: Este goal gera uma sequência numérica de 1 a N inclusive, sendo o resultado uma lista de valores decrescente. Como caso base o termo Pos (posição) será igual à dimensão da matriz, e a lista estará vazia.

Goal carregarDados

-Número de predicados: 1

-Cláusulas:

Clausula 1. carregarDados/2 com **Termos:**(File, DadosFicheiro)

-Descrição do funcionamento: Retorna o resultado do carregamento dos dados de um ficheiro ao permitir a abertura desse ficheiro sendo de seguida chamado o goal lerFicheiro que é responsável pelo processamento do mesmo.

Goal lerFicheiro

-Número de predicados: 1

-Cláusulas:

Clausula 1. lerFicheiro/2 com **Termos:**(Stream, Dados)

-Descrição do funcionamento: Goal que retorna o resultado de ler todas as letras de um ficheiro como uma lista de caracteres.

Goal lerLetra

-Número de predicados: 3

-Cláusulas:

Clausula 1. lerLetra/3 com **Termos:** (Stream, [NewCharConverted|DadosTail), _)
Clausula 2. lerLetra/3 com **Termos:** (Stream, Dados, LastChar)
Clausula 3. lerLetra/3 com **Termos:** (_, [], _) como caso base

-Descrição do funcionamento: Goal que lê todos os caracteres válidos de um ficheiro. É lido um caracter a cada chamada recursiva e este é adicionado ao cabeçalho da lista, sendo que a leitura para quando o valor do último caracter é -1. É retornada uma lista com todos os caracteres lidos como resultado, sendo que a cada caracter válido, é subtraído por – 48 para converter para inteiro.

Goal verifyChar

-Número de predicados: 5

-Cláusulas:

Clausula 1. verifyChar/2 com **Termos:** (10,_)
Clausula 2. verifyChar/2 com **Termos:** (32,_)
Clausula 3. verifyChar/2 com **Termos:** (-1,_)
Clausula 4. verifyChar/2 com **Termos:** (end_of_file,_)
Clausula 5. verifyChar/2 com **Termos:** (_,_)

-Descrição do funcionamento: Goal que permite verificar se um caracter é válido. Os predicados com um cut e um fail são os predicados que permitem indicar que aquele caracter é inválido, fazendo com que o predicado retorne como fail em vez de retornar como true. O cut aqui é necessário para evitar que se verifique os predicados alternativos. Só o ultimo predicado deve retornar true para validar os restantes caracteres.

Goal inserirValoresFixosDoFicheiro

-Número de predicados: 2

-Cláusulas:

Clausula 1. inserirValoresFixosDoFicheiro/2 com **Termos:** ([], _) como caso base
Clausula 2. inserirValoresFixosDoFicheiro/2 com **Termos:** ([X,Y,V|Dados], Tabuleiro)

-Descrição do funcionamento: Insere os valores fixos nas posições adequadas. A cada chamada recursiva é instanciada três variáveis a partir da lista que contem os dados (X,Y,V) e é usado o goal nth0 do prolog para verificar se existe uma estrutura válida dentro do tabuleiro ao qual é possível atribuir o valor V à variável correspondente ao último termo dessa estrutura. O X e Y determinam as coordenadas do tabuleiro que tem que ser idênticas aos valores do primeiro e segundo termo dessa estrutura. É retornado o resultado assim que não houver mais elementos na lista de dados do ficheiro.

Goal inserirStreamsDoFicheiro

-Número de predicados: 2

-Cláusulas:

Clausula 1. inserirStreamsDoFicheiro/4 com **Termos:** (Dados,0, _, TabuleiroAux) como caso base
Clausula 2. inserirStreamsDoFicheiro/4 com **Termos:** ([HDados|Dados], Tamanho, [HTabuleiro|Tabuleiro], TabuleiroCompleto)

-Descrição do funcionamento: Insere os valores das streams que foram lidas do ficheiro no tabuleiro. O goal irá processar cada stream da lista de dados do ficheiro para um certo tamanho máximo que neste caso é n*n sendo n o tamanho do tabuleiro. A cada chamada recursiva o cabeçalho da lista do tabuleiro é unificado com a estrutura acima referida, mas que neste caso o termo correspondente à stream é instanciado baseado no valor da stream que está no cabeçalho da lista de dados do ficheiro. O processo é concluído quando o tamanho chega a zero.

Goal gerarTabela

-Número de predicados: 1

-Cláusulas:

Clausula 1. gerarTabela/2 com **Termos:** ([Size|Dados], Resultado)

-Descrição do funcionamento: Goal que permite gerar o tabuleiro, retornando o resultado.

Goal gerarTabela_

-Número de predicados: 2

-Cláusulas:

Clausula 1. gerarTabela_/4 com **Termos:** (0, _, L, L) como caso base
Clausula 2. gerarTabela_/4 com **Termos:**(Linhas, Colunas, RTemp, Resultado)

-Descrição do funcionamento: Goal que gera para cada linha do tabuleiro, todas as colunas correspondentes a essa linha. A cada chamada recursiva é gerado a próxima linha, parando quando o número de linhas chegar a zero, retornando o resultado.

Goal gerarColunas

-Número de predicados: 2

-Cláusulas:

Clausula 1. gerarColunas/4 com **Termos:** (0, _, L, L) como caso base
Clausula 2. gerarColunas/4 com **Termos:** (Linha, Coluna, RTemp, Resultado)

-Descrição do funcionamento: Goal que gera N colunas por cada linha do jogo. É aqui que é utilizada pela primeira vez a estrutura acima referida onde é usado o goal functor do prolog para gerar um functor c com quatro argumentos. Depois do functor gerado, é adicionado a posição referente à linha e à coluna no mesmo, sendo que a stream e o valor são variáveis para serem unificadas posteriormente. A chamada recursiva acaba quando não existirem mais colunas para gerar, isto é, quando a variável Coluna chega a zero.

Goal gerarSolucao

-Número de predicados: 1

-Cláusulas:

Clausula 1. gerarSolucao/3 com **Termos:**(Tabela,ListaSeq,Size)

-Descrição do funcionamento: Goal principal que gera as soluções ao gerar as permutações da primeira linha do jogo e chama o goal gerarSolucao_/5 para gerar as restantes permutações e linhas. As permutações geradas vão sendo adicionadas à variável Permutado sequencialmente.

Goal gerarSolucao_

-Número de predicados: 2

-Cláusulas:

Clausula 1. gerarSolucao /5 com **Termos:** (Tabela,ListaSeq,Permutado,N,Size)
Clausula 2. gerarSolucao/5 com **Termos:** (Tabela,_,Permutado,1,Size) como caso base

-Descrição do funcionamento: Gera as permutações das linhas restantes, e contem o caso base. O caso base não gera permutações mas é responsável por colocar os valores na estrutura respeitando os valores dados no ficheiro, falhando se não for possível. Esta cláusula também chama as cláusulas de verificação. Se esta cláusula for bem sucedida, implica que a solução seja válida.

Goal verificarPermutacoes

-Número de predicados: 2

-Cláusulas:

Clausula 1. verificarPermutacoes /3 com **Termos:** (_,Size,Size) como caso base
Clausula 2. verificarPermutacoes /3 com **Termos:**(Lista,Size,N)

-Descrição do funcionamento: Esta cláusula faz uma verificação rápida, para as permutações falharem o mais cedo possível.
Funciona verificando se existem pares de valores iguais entre a permutação atual e a permutação anterior. Caso existam, o goal falha e invalida a permutação atual.

Goal atribuicao

-Número de predicados: 3

-Cláusulas:

Clausula 1. atribuicao/3 com **Termos:** ([],[],[]) como caso base
Clausula 2. atribuicao /3 com **Termos:** ([Tabela|TTail],[Permutado|Tail],NovaTabela)
Clausula 3. atribuicao /3 com **Termos:** ([Tabela|TTail],[Permutado|Tail],NovaTabela)

-Descrição do funcionamento: Este goal é responsável por inserir os valores das permutações (lista) numa estrutura de dados que contem a posição e a stream de um valor. Se este valor já estiver definido, verifica se é igual à permutação. É saltado caso contrário o goal falhe.

A estrutura resultante vai ser igual à original em termos de posições e streams mas os valores estarão todos definidos. A estrutura original mantem-se, sendo esta gerada originalmente no goal gerarTabela.

Goal regraDiferentes

-Número de predicados: 2

-Cláusulas:

Clausula 1. regraDiferentes/2 com **Termos:** (Lista, N)
Clausula 2. regraDiferentes/2 com **Termos:** (_, 0)como caso base

-Descrição do funcionamento: Goal que verifica se o tabuleiro está em conformidade com as regras que são válidas para que uma solução seja gerada. Para cada N, onde N é o tamanho do tabuleiro, é verificado se a linha, coluna e stream N são válidas. Retorna true se o tabuleiro é válido.

Goal regraLinhaDiferente

-Número de predicados: 3

-Cláusulas:

Clausula 1. regraLinhaDiferente /2 com **Termos:** (Linha, [X | Lista])
Clausula 2. regraLinhaDiferente /2 com **Termos:** (_, []) como caso base
Clausula 3. regraLinhaDiferente /2 com **Termos:** (Linha, [X | Lista])

-Descrição do funcionamento: Goal que verifica se a linha contem números diferentes. A cada chamada recursiva o valor que está à cabeça da lista é unificado com a estrutura acima referida, contendo a linha e o valor como variáveis (c(Linha, _, _, V)). Caso não exista no resto da lista a mesma estrutura com a mesma linha e valor, quer dizer que não existe números repetidos nessa linha.

Goal regraColunaDiferente

-Número de predicados: 3

-Cláusulas:

Clausula 1. regraColunaDiferente /2 com **Termos:** (Coluna, [X | Lista])
Clausula 2. regraColunaDiferente /2 com **Termos:** (_, []) como caso base
Clausula 3. regraColunaDiferente /2 com **Termos:** (Coluna, [X | Lista])

-Descrição do funcionamento: Goal que verifica se a coluna contem números diferentes. A cada chamada recursiva o valor que está à cabeça da lista é unificado com a estrutura acima referida, contendo a coluna e o valor como variáveis (c(_, Coluna, _, V)). Caso não exista no resto da lista a mesma estrutura com a mesma coluna e valor, quer dizer que não existe números repetidos nessa coluna.

Goal regraStreamDiferente

-Número de predicados: 3

-Cláusulas:

Clausula 1. regraStreamDiferente /2 com **Termos:** (Stream, [X | Lista])
Clausula 2. regraStreamDiferente /2 com **Termos:** (_, []) como caso base
Clausula 3. regraStreamDiferente /2 com **Termos:** (Stream, [X | Lista])

-Descrição do funcionamento: Goal que verifica se a stream contem números diferentes. A cada chamada recursiva o valor que está à cabeça da lista é unificado com a estrutura acima referida, contendo a stream e o valor como variáveis (c(_, _, Stream, V)). Caso não exista no resto da lista a mesma estrutura com a mesma stream e valor, quer dizer que não existe números repetidos nessa stream.

Goal showResult

-Número de predicados: 3

-Cláusulas:

Clausula 1. showResult/4 com **Termos:** (Resto,0,_,Resto) como caso base
Clausula 2. showResult/4 com **Termos:** ([L|Tail],N,Dimensao,Rest)
Clausula 3. showResult/3 com **Termos:** (Lista, N, Dimensao)

-Descrição do funcionamento: Este goal é responsável por fazer o display do output da forma descrita no enunciado do trabalho. A cláusula showResult/3 é o ponto de entrada do goal, recebendo como termos a lista de objetos a serem mostrados, e a dimensão do tabuleiro (N e Dimensão).
A cláusula de entrada chama a cláusula showResult/4 para fazer print de uma linha da matriz de valores. Como o goal recebe como termo uma lista, faz print dos N primeiros elementos da lista e devolve o resto da lista por processar.

O goal é atingido quando o Número de colunas vistas for igual à dimensão da matriz.

CONCLUSÃO

Neste trabalho foi possível aumentar os conhecimentos da linguagem Prolog e adquirir uma maior compreensão sobre problemas do tipo np-Complete.
Estre trabalho estimulou o uso de tipos de dados mais complexos e o reconhecimento de padrões (Streams), a implementação de regras de verificação.

O método utilizado para encontrar a solução foi um método de força bruta. Sobre o resultado de todas as possibilidades foi aplicado um algoritmo de filtragem para eliminar possibilidades inválidas, restringindo os casos que são sujeitos a verificações. Caso a hipótese passasse esta verificação inicial, seria submetido a todas as verificações para garantir que a solução é realmente válida ou não.

Durante a fase de planeamento do trabalho foram pensadas várias estruturas para os dados do ficheiro e do tabuleiro. A estrutura atual foi selecionada devido a intuitividade de implementação. Entre as estruturas faladas estavam as matrizes devido ao acesso direto aos valores, e uma estrutura multidimensional em que os códigos de verificação seriam sempre os mesmos.

Tendo uma matriz multidimensional constituída por functors c(Posição X, Posição Y, Valor). No nível 1 encontrar-se-ia a matriz de valores (variáveis se não foram atribuídas), no segundo nível seria a matriz transposta para a verificação de colunas ser igual a verificação de linhas e no 3º nível estariam definidas as streams, cada stream numa linha da matriz. Os níveis estariam ligados pelas variáveis não atribuídas, de modo a que quando uma alteração fosse feita num nível as alterações se propagassem por todos os níveis.