Daniel Zambrano

Extensão da gramática de Imp

- Uma lista de inteiros pode ser declarada da forma [1, 4, 3..] e com isso foi definido **list_int**:

```
list_int = "[" e:exp {','e:exp}* "]" ;
```

- Para acessar um elemento de uma lista L no índice n utiliza-se L[n], e com isso foi definido **list_index**:

```
list_index = idn:identifier"["e:exp"]" ;
```

- Para atualizar um elemento de uma lista L no índice n utiliza-se L[n] := valor, e com isso foi definido **list_assign**:

```
list_assign = idn:identifier"["idx:exp"]" op:":=" e:exp ;
```

- Para realizar uma operação de "append" de uma lista M a uma lista L utiliza-se L ++ M, e para essa operação foi definido **list_append**:

```
list_append = l1:exp "++" l2:exp           |
              l1:list_int "++" l2:list_int |
              l1:exp "++" l2:list_int      |
              l1:list_int "++" l2:exp      ;
```

- Para realizar uma operação de concatenação de um número n ao final de uma lista L se utiliza L << n, e para isso foram estendidas as definições de **binop** e **bin_exp:**

```
binop = "<<" | "and" | "or" | …

bin_exp = l:list_int op:binop e:exp | ...
```

- Para realizar a operação que retorna o tamanho de uma lista L é utilizado :: L, e para isso foi estendida a definição de **un_exp:**

```
bin_exp = op:"::" e:exp | op:"not" e:exp ;
```

Para comportar essas novas operações, também foram estendidas as definições de **atom_cmd e exp:**

```
exp = list_index | list_append | paren_exp |
      bin_exp | un_exp | list_int | @:atom ;

atom_cmd = cond | loop | list_assign |
           assign | print | call | skip ;
```

Pi denotações de Imp à Pi IR para a extensão da gramática

Foram estendidas duas Pi denotações e criadas quatro novas:

bin_exp

Adicionada verificação do operador "<<" para chamar a nova classe de pi "*Concat*" de operação de concatenação, passando as duas expressões de *ast*.

un_exp

Adicionada verificação do operador "::" para chamar a nova classe de pi ListSize de operação de tamanho, passando a única expressão de *ast*.

obs.: Foram escolhidos os operadores "::" e "<<" por conta de uma dificuldade em implementar as operações com "+" e '#', respectivamente, mencionados na definição da avaliação.

Novas denotações:

list_int

Essa denotação diz respeito a chamada direta de uma lista [a, b, c], e possui como atributo *ast.e* uma lista de objetos Num, e é a representação primária da lista de inteiros. Essa denotação chama a nova classe "*ListInt"* que categoriza as novas equações do Pi Framework referente a visualização de uma lista.

O condicional dentro dessa denotação é para tratar os casos de uma chamada com uma lista de expressões ou apenas um *"Num"*, no caso de uma lista de um elemento só.

list_index

Denotação referente a operação de projeção de leitura de um elemento da lista e chama a nova classe de pi "*ListIndex"* que recebe como parâmetros um identificador representando a variável da lista e uma expressão representando o índice que deve ser acessado.

list_append

Essa denotação faz referência a operação de append entre duas listas, e chama a nova classe de pi "*ListIndex"* com as duas listas como parâmetro.

list_assign

Essa denotação cuida da projeção de escrita de um elemento de uma lista, e possui os operandos *idn* para identificar a variável, *idx* para representar o índice que deve ser acessado para escrita e por fim *e* contendo a expressão com o valor que deve ser escrito. Ela chama a nova classe de pi *ListAssign.*

Novas equações do Pi Framework

Foram adicionadas as classes *ListInt(Statement), ListSize(Exp), ListIndex(Exp), Concat(Exp) ListAppend(Exp) e ListAssign(Cmd)* para tratar das denotações definidas acima. Em algumas classes já existentes foi estendida a condição de inicialização para dar suporte a essas classes novas, como em *Assign,* onde o atributo *ast.e* agora pode ser tanto *Exp* quanto *ListInt*.

Para todas as classes, com exceção de ListInt, foram criados dois métodos de *evaluation* dentro da classe de expressões do Pi Autômato, uma para colocar o resultado da operação na pilha de valores e outra para escrever na pilha de controle a chamada de operação. Todos os métodos foram incluidos na lista de condicional do autômato para que pudessem ser chamadas.

Para o *ListInt* foi criado apenas o da pilha de valores, pois ele é tratado de maneira análoga ao *Num* nesse aspecto, ou seja, sem necessidade do uso na pilha de controle.

Abaixo está a lista das equações que dão semântica às novas construções em Pi IR para lista de inteiros:

__evalListInt

e contém uma lista de inteiros
empilha a lista na pilha de valores

__evalListAppend

e contém duas listas
empilha #APPEND na pilha de controle e em seguida a primeira e segunda lista

__evalListAppendKW

desempilha duas listas da pilha de valores
empilha uma nova lista contendo a junção delas na pilha de valores

__evalConcat

e contém uma lista e uma expressão
empilha #CONCAT na pilha de controle e em seguida a lista e a expressão

__evalConcatKW

retira uma lista e um inteiro da pilha de valores
empilha na pilha de valores uma nova lista com o inteiro concatenado no final dela

__evalListIndex

e contém um identificador indicando a variável e uma expressão indicando o índice na lista
empilha #IDX na pilha de controle e em seguida o identificador e o índice

__evalListIndexKW

desempilha uma lista e um número n pilha de valores
coloca na pilha de valores o enésimo elemento da lista

__evalListSize

e contém uma lista
empilha #SIZE na pilha de controle seguida da lista

__evalListSizeKW

desempilha uma lista da pilha de valores
empilha na pilha um número com o tamanho da lista

__evalListAssign

e contém um identificador para uma variável, uma expressão para índice e uma expressão para o valor a ser escrito

empilha a string do identificador na pilha de valores seguido do índice e empilha na pilha de controle #LASG seguido do valor a ser escrito

__evalListAssignKW

desempilha da pilha de valores um valor a ser escrito, seguido de uma expressão representando um índice e um identificador de variável

busca o valor bindable que está atribuído ao identificador no ambiente e busca no *store* a lista associada ao identificador para que tenha o valor no índice atualizado. Caso o índice seja um identificador, faz o mesmo processo de busca no storage para descobrir seu valor. Nada é empilhado, mas o *store* é atualizado na localização do identificador com a lista atualizada.

Arquivo de testes

O arquivo de testes criado está em *examples/list-comprehension.imp2* e aborda as operações especificadas acima. ~~Não foi criado um arquivo contendo o BubbleSort.~~ Foi criado um arquivo com o BubbleSort após o prazo de entrega.