UNIVERSIDADE TÉCNICA DE LISBOA

INSTITUTO SUPERIOR TÉCNICO

LICENCIATURA DE ENGENHARIA INFORMÁTICA E DE COMPUTADORES

PROJECTO DE FUNDAMENTOS DE PROGRAMAÇÃO 2005 / 2006

DOCUMENTAÇÃO DE UTILIZAÇÃO

APL – A Programming Language

21 de Dezembro de 2005

André Matias nº 56907

Pedro Pinto nº 57006

Índice

O Que o Programa Faz

O programa desenvolvido neste projecto é, basicamente, uma implementação na **linguagem Scheme** de uma **variante da linguagem APL** (*A Programming Language*). O programa permite a utilização de um conjunto relativamente vasto de operações características da linguagem APL, seguindo sempre a sintaxe e a semântica da linguagem Scheme.

Os tipos de dados elementares que o programa manipula são apenas **números** e **booleanos**. Contudo, estes tipos de dados podem estar estruturados em **tensores** (ou *arrays*). Um tensor é, por sua vez, um aglomerado de valores dispostos numa estrutura rectangular, podendo a mesma possuir qualquer número de dimensões.

Como casos particulares dos referidos tensores temos os escalares, os vectores e as matrizes, dependendo directamente das dimensões do tensor, da seguinte maneira:

- Se o tensor tem **zero dimensões**, corresponde a um **escalar**.
- Se o tensor tem **uma dimensão**, corresponde a um **vector**.
- Se o tensor tem **duas dimensões**, então dizemos que é uma **matriz**.
- No **caso geral**, dizemos apenas que temos um **tensor com um dado número de dimensões**.

Uma das operações mais básicas que o programa disponibiliza é a que permite construir um vector a partir de um número arbitrário de elementos:

- **v :** Operação básica denominada *v* que, tal como referido, recebe um número arbitrário de argumentos e constrói um vector com esses mesmos elementos colocados pela ordem na qual foram inseridos.

A importância desta operação deve-se ao facto de ela estar na base da construção dos tensores de dimensão maior que zero. Existem igualmente algumas funções que recebem necessariamente um vector para determinados fins, sendo esta operação necessária para o efeito (por exemplo, a função *reshape* recebe necessariamente como primeiro argumento um vector com as dimensões do tensor, como se verá mais adiante).

Foram implementadas igualmente algumas funções que manipulam os referidos tensores. Estas funções dividem-se em dois tipos:

- **Monádicas**, funções que recebem apenas um argumento.
- **Diádicas**, funções que recebem dois argumentos.

As **funções monádicas** implementadas no programa são as seguintes:

- **display-tensor :** Recebe um tensor como argumento e escreve o conteúdo do tensor no ecrã. A escrita referida obedece a quatro regras:
    1. Se o tensor é um escalar, escreve simplesmente o escalar.
    2. Se o tensor é um vector, escreve os elementos do vector todos na mesma linha e separados por um espaço em branco.
    3. Se o tensor é uma matriz, escreve as linhas da matriz como se estivesse a escrever vectores e muda de linha entre cada linha escrita.
    4. Se o tensor não é nenhum dos casos anteriormente referidos, então para cada sub-tensor da primeira dimensão, escreve o sub-tensor separado por um número de mudanças de linha igual ao número de dimensões do tensor menos um.
- **symmetrical :** Produz um tensor cujos elementos são o simétrico dos elementos correspondentes do tensor argumento.
- **inverse :** Produz um tensor cujos elementos são o inverso dos elementos correspondentes do tensor argumento.
- **! :** Produz um tensor cujos elementos são o factorial dos elementos correspondentes do tensor argumento.
- **sqrt :** Produz um tensor cujos elementos são a raiz quadrada dos elementos correspondentes do tensor argumento.
- **sin :** Produz um tensor cujos elementos são o seno dos elementos correspondentes do tensor argumento.
- **cos :** Produz um tensor cujos elementos são o cosseno dos elementos correspondentes do tensor argumento.
- **~ :** Produz um tensor cujos elementos são a negação dos elementos correspondentes do tensor argumento. O tensor resultante terá, como elementos, os inteiros 0 ou 1 consoante o elemento correspondente do tensor argumento for diferente de zero ou igual a zero, respectivamente.
- **shape :** Produz um vector cujos elementos correspondem ao comprimento de cada dimensão do tensor argumento.
- **interval :** Produz um vector cujos elementos correspondem a uma enumeração dos inteiros desde 1 até ao escalar argumento.

No que diz respeito às **funções diádicas**, as funções implementadas no programa são as seguintes:

- **+ :** Produz um tensor com a soma dos elementos dos tensores argumentos. Se os argumentos forem tensores com o mesmo tamanho e forma, o tensor resultante tem o mesmo tamanho e forma dos tensores argumentos e tem, como elementos, a soma dos elementos correspondentes dos tensores argumentos. Se um dos argumentos for um escalar, o tensor resultado tem o mesmo tamanho e forma que o outro argumento e tem, como elementos, a soma do argumento escalar com todos os elementos do outro argumento. Se o caso no qual se aplica a soma não é nenhum dos anteriormente referidos, a soma é um erro.
- **- :** Apresenta o mesmo comportamento que a função anterior, mas aplicando a subtracção.
- *** :** Apresenta o mesmo comportamento que a função anterior, mas aplicando a multiplicação.
- **/ :** Apresenta o mesmo comportamento que a função anterior, mas aplicando a divisão.
- **quotient :** Apresenta o mesmo comportamento que a função anterior, mas aplicando a divisão inteira.
- **remainder :** Apresenta o mesmo comportamento que a função anterior, mas empregando o resto da divisão inteira.
- **< :** Apresenta o mesmo comportamento que a função anterior, mas aplicando a relação "menor que". O tensor produzido terá, como elementos, os inteiros 0 ou 1 (booleanos) consoante a relação é falsa ou verdadeira, respectivamente.
- **> :** Apresenta o mesmo comportamento que a função anterior, mas aplicando a relação "maior que".
- **<= :** Apresenta o mesmo comportamento que a função anterior, mas aplicando a relação "menor ou igual que".
- **>= :** Apresenta o mesmo comportamento que a função anterior, mas aplicando a relação "maior ou igual que".
- **= :** Apresenta o mesmo comportamento que a função anterior, mas aplicando a relação "igual a".

- **|| :** Apresenta o mesmo comportamento que a função anterior, mas aplicando a disjunção lógica.
- **&& :** Apresenta o mesmo comportamento que a função anterior, mas aplicando a conjunção lógica.
- **drop :** Recebe um escalar n_1 ou vector (de elementos n_i) e um tensor não escalar e devolve um tensor onde foram removidos n elementos no início (se $n > 0$) ou no fim (se $n < 0$) da dimensão i do tensor.
- **reshape :** Produz um tensor com as dimensões referidas no vector primeiro argumento e cujos elementos são obtidos através da enumeração dos elementos do tensor segundo argumento, repetindo essa enumeração as vezes que forem necessárias para preencher o tensor resultado.
- **catenate :** No caso de os dois argumentos serem escalares, a função produz um vector contendo esses argumentos. No caso de os dois argumentos serem tensores, produz um novo tensor que junta os outros dois ao longo da sua última dimensão.
- **member :** Produz um tensor de booleanos com a mesma forma e dimensão do primeiro argumento contendo um booleano 1 para cada elemento na posição correspondente do primeiro tensor que ocorra algures no segundo tensor e 0 em caso contrário.
- **select :** Recebe como argumentos um vector de booleanos e um tensor e devolve um tensor contendo apenas os elementos da última dimensão do tensor argumento que possuem o valor 1 na posição correspondente do primeiro vector.

Além das funções "comuns" referidas, existe, em APL, uma categoria especial de funções denominadas **operadores**. Um **operador** é uma função que recebe outras operações como argumentos (operações de ordem superior) e devolve funções como resultados.

Os operadores, tal como as funções anteriormente descritas, dividem-se em operadores **monádicos** e **diádicos**, consoante recebam um ou dois argumentos, respectivamente.

Apresentam-se de seguida os **operadores monádicos** implementados no programa:

- **fold :** O operador de redução *fold* recebe uma função como argumento e devolve outra função que, dado um vector, avalia o resultado de inserir a função entre cada dois elementos dos vector.
- **scan :** O operador *scan* funciona de modo semelhante ao operador *fold* mas empregando conjuntos sucessivamente maiores de elementos, desde um conjunto com apenas o primeiro elemento até ao conjunto com todos os elementos.
- **outer-product :** Recebe uma função como argumento e devolve uma função que, dados dois tensores, produz um novo tensor com o resultado de aplicar a função argumento a cada uma das combinações de valores do primeiro e segundo tensores.

Relativamente aos **operadores diádicos**, implementámos um único operador, cujo nome e função são os seguintes:

- **inner-product :** Recebe duas funções como argumentos e devolve uma função que, dados dois tensores, produz um novo tensor de acordo com a regra do produto interno algébrico mas substituindo a soma e a multiplicação pela primeira e segunda funções, respectivamente.

Processo de Utilização do Programa

Neste ponto pretende-se explicar ao utilizador como proceder para utilizar o programa em questão, tendo o utilizador que seguir um determinado número de passos relativamente simples para o efeito, explicados em seguida.

Primeiramente, o utilizador deverá iniciar uma sessão com o **Interpretador do Scheme**. O local no disco rígido dependerá do caminho que o utilizador escolheu inicialmente ao instalar o Scheme.

Ao iniciar uma sessão com o Scheme, o utilizador visualizará algo semelhante ao apresentado na figura seguinte:

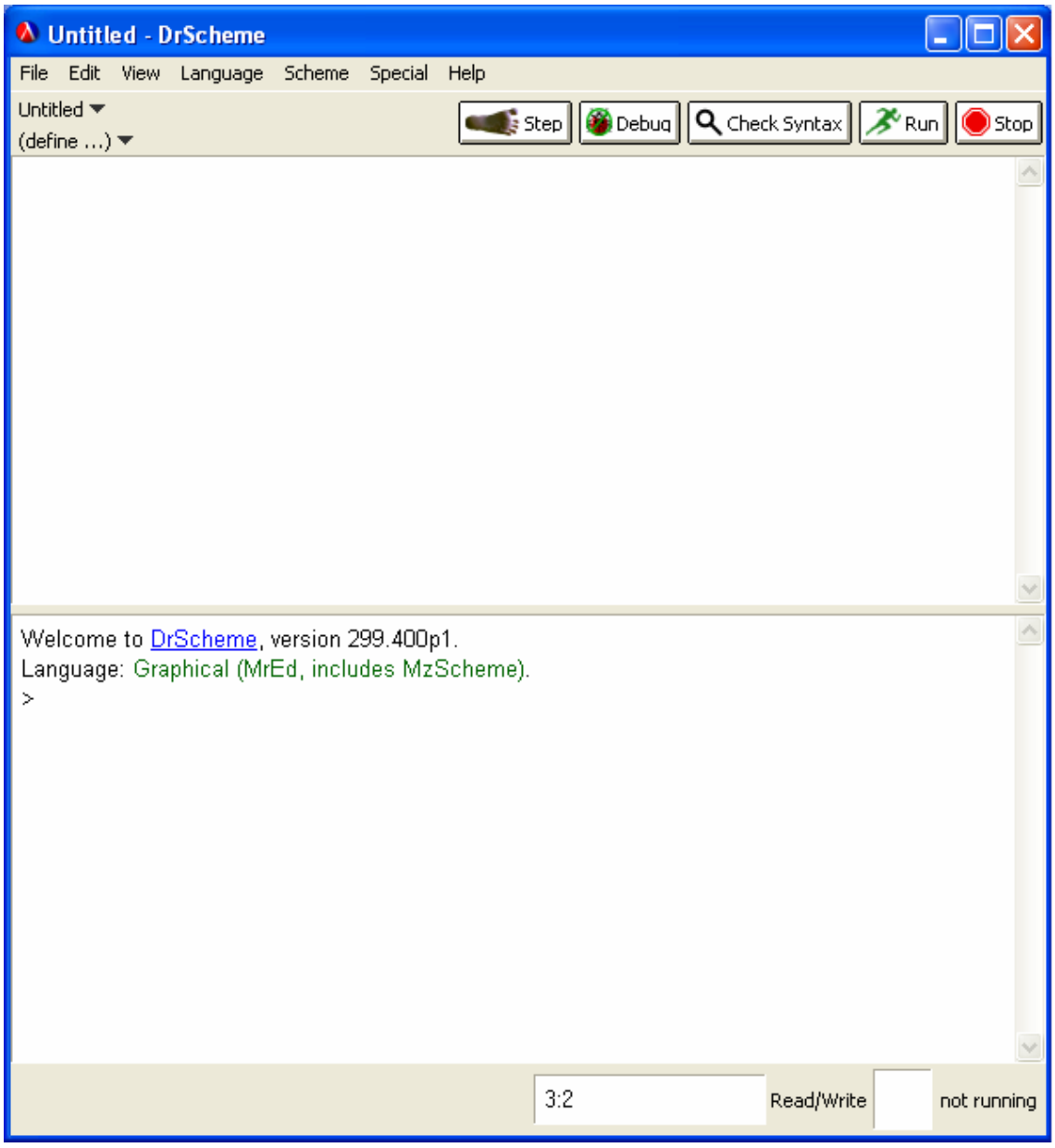

Figura 1: Janela que se obtém ao iniciar uma sessão em Scheme.

Repare-se que a janela possui uma divisão central, separando duas janelas de menores dimensões:

- **Janela de Interacções:** Janela inferior, na qual se comunica com o avaliador do Scheme. É aqui que se vai dar uso ao programa em questão.

- **Janela de Definições:** Janela superior, na qual se coloca o código dos programas, sendo este depois armazenado em ficheiros.

O passo seguinte consiste em o utilizador se certificar de que o Scheme se encontra em funcionamento na linguagem correcta. Neste caso, a linguagem a utilizar será a **Standard (R5RS)**. A linguagem em uso poderá ser verificada pela observação da janela de interacções ao se iniciar uma nova sessão com o interpretador do Scheme (ver figura 1 – "*Language: Graphical (MrEd, includes MzScheme).*").

Para mudar a linguagem corrente utiliza-se a entrada *Language* presente na barra de menus (no topo da janela total do Scheme), dependendo a apresentação desta barra do sistema operativo em uso. De seguida escolhe-se a opção *Choose Language…*, obtendo-se a janela presente na seguinte figura:

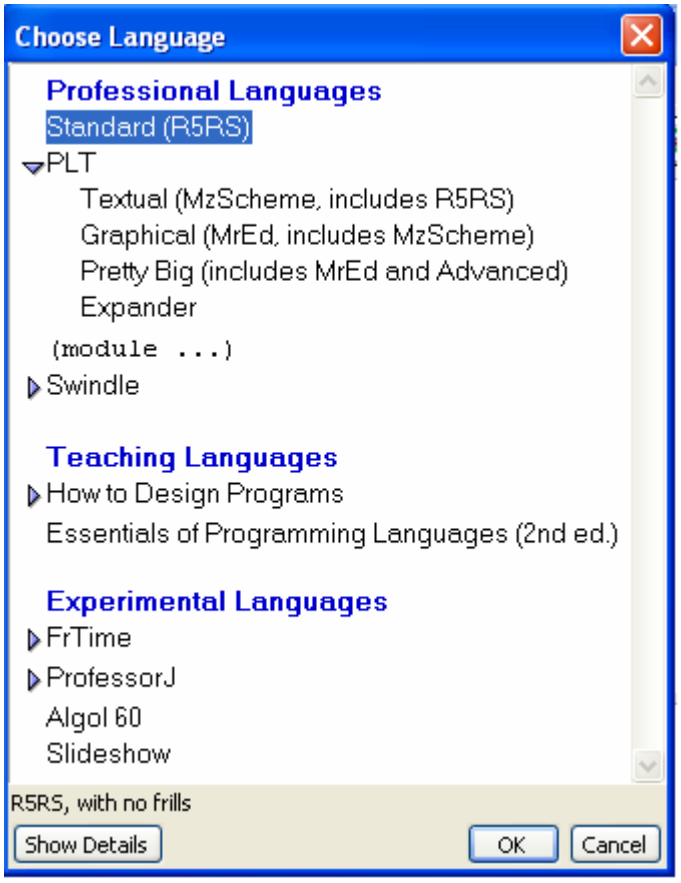

Figura 2: Janela de escolha de linguagem.

Seguidamente, após a escolha da linguagem correcta, neste caso, a linguagem Standard (R5RS), o **código do programa** deverá ser colocado na janela de definições.

Por último, há que utilizar o botão ***Run*** (

) para que a linguagem previamente escolhida e o código do programa inserido surtam efeito. Só depois deste passo, o programa estará pronto a ser usado, apresentando-se a janela do Scheme como demonstrado na seguinte figura:

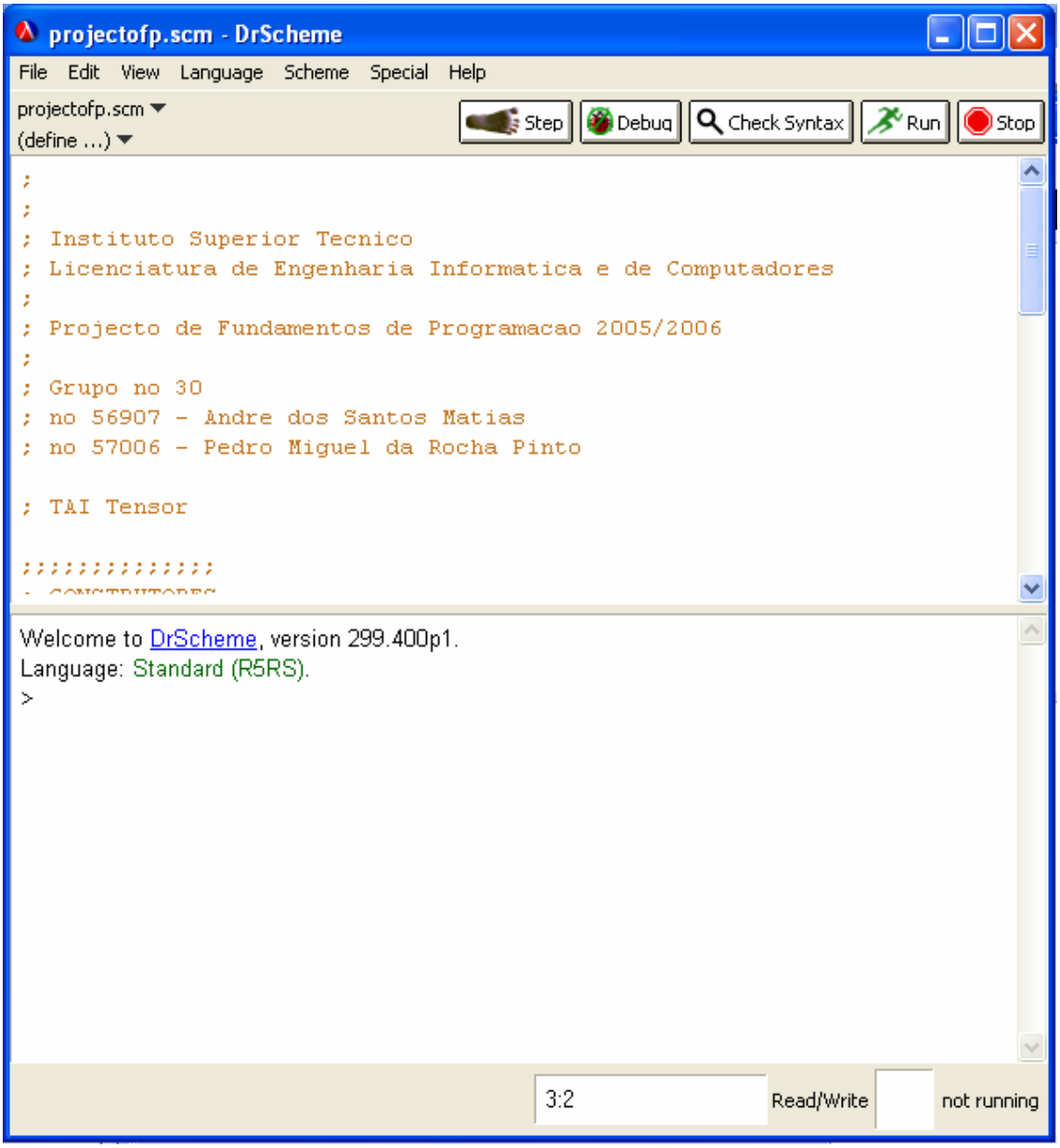

Figura 3: Estado da janela do Scheme após inserção do código e escolher *Run*.

Se o utilizador seguir os passos descritos pela ordem em que foram colocados, o programa encontra-se agora pronto para ser utilizado, bastando que, para isso, o utilizador insira na janela de interacções as funções, operadores, etc. que pretenda usar, respeitando sempre a sintaxe característica do Scheme, explicada mais na secção seguinte deste manual.

O utilizador poderá eventualmente, após realização correcta dos passos anteriormente descritos, utilizar a entrada *File* na barra de menus, escolhendo de seguida a opção *Save Definitions as…*, de modo a guardar o código do programa numa unidade de memória com um caminho e nome de ficheiro arbitrários, à escolha do utilizador. Para aceder ao ficheiro anteriormente guardado, utiliza-se na mesma a entrada *File* na barra de menus, escolhendo a opção *Open…*, indicando de segudia o caminho correcto no qual se situa esse mesmo ficheiro.

Informação Necessário ao Bom Funcionamento do Programa

A este ponto do manual o utilizador já deverá saber qual a utilidade do programa em questão, quais as funções existentes e respectivos nomes e tornar o programa pronto a ser usado, ou seja, pronto a receber expressões por parte do utilizador e avaliá-las correctamente, segundo os objectivos deste mesmo programa.

Seguir-se-á então a explicação de como o utilizador deve proceder para aplicar os procedimentos disponíveis (e respectivas funções/operadores que lhes estão inerentes) aos argumentos que o utilizador preferir.

Para esse fim, é necessário que o utilizador conheça e domine minimamente a **sintaxe usada pelo programa**, que corresponde à **sintaxe do próprio Scheme**.

Primeiramente, há que relembrar que qualquer procedimento que o utilizador aplicar, terá que a escrever necessariamente na janela de interacções do Scheme.

A sintaxe de uma linguagem é o conjunto de regras que definem quais as relações válidas entre os componentes dessa mesma linguagem, tal como acontece com as palavras e as frases. No caso relevante, é a sintaxe da linguagem Scheme que teremos que respeitar para assegurar um bom funcionamento do programa.

Em linguagem corrente usamos notação prefixa para indicar operações, ou seja, o operador surge entre os operandos (argumentos do operador). Em linguagem Scheme, as operações são escritas utilizando **notação prefixa**, ou seja, o operador aparece antes dos operandos e o operador e os seus operandos são escritos dentro de parênteses.

Recomendamos a compreensão da seguinte interacção, que exemplifica a correcta utilização da função + do programa em questão.

Exemplo:

```
> (display-tensor (+ (v 1 2 3)
                     (v 1 1 1)))
2 3 4
```

Nota: A identação (ou alinhamento vertical) dos argumentos da função + apresentada no exemplo não é obrigatória, podendo o utilizador inserir na janela de interacções do Scheme a expressão do exemplo sem respeitar a identação, ou seja, sem

recorrer a qualquer parágrafo. Tal forma de apresentação será eventualmente usada neste manual para facilitar ao utilizador a compreensão dos exemplos mostrados.

Repare-se que, no exemplo anterior, foram usadas várias operações, sendo apenas uma delas a função +. As restantes operações igualmente usadas foram as seguintes: *display-tensor* e *v*. Recorde-se que a operação *v* recebe um número arbitrário de argumentos e constrói um vector a partir deles. Neste caso, portanto, somámos o vector (1 2 3) ao vector (1 1 1), tendo obtido o vector (2 3 4). A função monádica *display-tensor*, como referido anteriormente, é responsável pela escrita no ecrã do conteúdo do tensor resultante, neste caso, da soma entre os dois vectores. Mais detalhes sobre o *output* ou informação produzida pelo programa durante as interacções serão explicadas na secção seguinte deste manual.

Em geral, todas as **funções** recebem um ou dois tensores argumentos, consoante sejam monádicas ou diádicas, respectivamente. O conteúdo dos tensores terá que ser, por sua vez, composto por um tipo de dados aceite, nomeadamente **inteiros** ou **booleanos**. Estas funções devolvem um tensor, exceptuando os casos de erro, que serão explicados mais à frente.

Existe, no entanto, um conjunto de funções especiais que não segue o referido anteriormente. Estas funções especiais designam-se, como já foi referido atrás, por **operadores**. Estes operadores, à semelhança das restantes funções, recebem um ou dois argumentos, consoante sejam monádicas ou diádicas. Todavia, os argumentos que estes operadores recebem não são tensores mas sim **outras operações**, devolvendo, aquando da sua aplicação, uma outra função, exceptuando os casos de erro, que serão descritos mais à frente.

Segue-se um exemplo da aplicação de uma operação, *fold*.

Exemplo:

> (display-tensor ((fold +) (v 1 2 3 4 5)))

15

A expressão *(fold +)* quando é avaliada devolve uma função que recebe um vector como argumento. Deste modo *(v 1 2 3 4 5)* não funciona como argumento da operação *fold*, pois esta apenas recebe o argumento +, funciona sim como argumento da função gerada pela aplicação da operação *fold* ao seu argumento +.

Descrição da Informação Produzida

Segue-se a descrição da **informação produzida pelo programa**, que inclui o *output* produzido e formato de *output*, aquando da aplicação de funções e/ou operações aos respectivos argumentos. Entenda-se por *output* produzido a representação externa que o programa atribui aos tensores. Serão igualmente explicados os **casos de aplicação dos procedimentos implementados que resultam em erro**, sendo explicado o porquê do respectivo erro.

A representação externa, salvo casos de erro explicados mais à frente, está a cargo da função monádica *display-tensor*. Relembremos o seu funcionamento, recorrendo a exemplos:

display-tensor: Recebe um tensor como argumento e escreve o conteúdo do tensor no ecrã. A escrita referida obedece a quatro regras:

1 Se o tensor é um escalar, escreve simplesmente o escalar.

Exemplo:

> (display-tensor 3)

3

2 Se o tensor é um vector, escreve os elementos do vector todos na mesma linha e separados por um espaço em branco.

Exemplo:

> (display-tensor (v 1 2 3 4))

1 2 3 4

3 Se o tensor é uma matriz, escreve as linhas da matriz como se estivesse a escrever vectores e muda de linha entre cada linha escrita.

Exemplo:

> (display-tensor (reshape (v 2 2) (v 1 2 3 4)))

1 2

3 4

4 Se o tensor não é nenhum dos casos anteriormente referidos, então para cada sub-tensor da primeira dimensão, escreve o sub-tensor separado por um número de mudanças de linha igual ao número de dimensões do tensor menos um.

Exemplos:

> (display-tensor (reshape (v 2 2 2) (v 1 2 3 4)))
1 2
3 4

1 2
3 4
> (display-tensor (reshape (v 2 2 2 2) (v 1 2 3 4)))
1 2
3 4

1 2
3 4

1 2
3 4

1 2
3 4

Relativamente ao *output* produzido como resultado de erro, será devolvida uma mensagem de erro sempre que se aplicar um determinado procedimento a argumentos inválidos ou em número errado, para esse mesmo procedimento.

Primeiramente, há que referir novamente que o programa foi concebido unicamente para manipular números e booleanos (0 e 1), podendo a utilização de outro tipo de dados, qualquer que seja o procedimento aplicado, resultar em erro.

Passaremos a descrever os possíveis erros e o procedimento ao qual se aplica, caso sejam erros específicos para um dado procedimento.

Funções Monádicas:

Destas funções não resulta qualquer erro de relevante importância para o utilizador. No entanto, recorde-se de que o anteriormente dito deve ser respeitado, ou seja, estas funções recebem apenas um argumento, necessariamente um tensor, cujo conteúdo é composto por números ou booleanos e o número de argumentos deve ser respeitado.

Funções Diádicas:

As funções +, -, *, /, *quotient*, *remainder*, <, >, <=, >=, =, || e && são aplicadas correctamente quando ambos os argumentos são tensores com o mesmo tamanho e forma ou quando um dos argumentos é um escalar e o outro for um qualquer tensor, resultando em erro em caso contrário.

Exemplo:

> (display-tensor (+ (reshape (v 2 2) (v 1 2 3 4))
(v 1 23 4)))
erro: funcao-tensor: os argumentos nao tem a mesma forma

Há que relembrar que as funções || e && são para aplicação apenas em tensores cujo conteúdo são booleanos. Caso tal não seja respeitado, obter-se-á um resultado não previsto.

Eis outro conjunto de funções diádicas, cada uma com um erro que lhe é específico:

drop : Caso, aquando da sua aplicação, o primeiro argumento não seja um vector ou escalar e o segundo argumento não for um tensor não escalar, o programa devolve um erro.

Exemplo:

> (display-tensor (drop (v 1 2 3 4) 3))
Erro: drop: o segundo argumento nao pode ser um escalar

Outro caso de erro com este mesmo procedimento, é a sua aplicação com um primeiro argumento que indique a eliminação de uma dada dimensão (linha, coluna, subtensor) não existente no segundo argumento.

Exemplo:

> (display-tensor (drop (v 3 3)
(reshape (v 2 2) (v 1 2 3 4))))
Erro: remove-tensor: posicao fora do tensor

reshape : Este procedimento recebe obrigatoriamente um vector como primeiro argumento, resultando em erro em caso contrário.

Exemplo:

> (display-tensor (reshape 3 (v 1 2 3)))
Erro: cria-tensor: o primeiro argumento nao e um vector

catenate : Se o procedimento for chamado com tensores com um número diferente de dimensões ou número igual de dimensões em que a forma de uma determinada dimensão, que não seja a última, é diferente em cada tensor, o resultado será um erro.

Exemplos:

> (display-tensor (catenate (reshape (v 2 2 2) (v 12 3 4))
(reshape (v 2 3 3) (v 1 2 3 4))))
Erro: junta-tensor: os argumentos devem ter o mesmo tamanho

select : Se o procedimento for aplicado com um primeiro argumento que não seja um vector de booleanos então o resultado não está definido.

Note-se que o vector primeiro argumento tem de ter o mesmo tamanho que a ultima dimensão do tensor segundo argumento, resultando em erro em caso contrário.

Operadores Monádicos e Diádicos:

As funções incluídas nestes grupos denominam-se operadores, que são por sua vez, funções que recebem outras operações como argumentos e devolvem funções como resultados. Desta forma, aplicar um operador com um argumento que não seja uma outra operação, resultará em erro.

Exemplos:

> (display-tensor ((fold 3) (v 1 2 3 4)))
Erro: procedure application: expected procedure, given: 3; arguments were: 3 4
> (display-tensor ((outer-product 3 5) (v 1 2 3 4)))
Erro: procedure outer-product: expects 1 argument, given 2: 3 5

A função *inner-product* apresenta um caso particular de erro, que ocorre quando a função resultante devolvida pela sua aplicação recebe pelo menos um tensor com mais de duas dimensões. Caso se aplique o procedimento erradamente desta forma, o resultado devolvido é indefinido, não sendo previsto.

Convém relembrar que os operadores monádicos recebem apenas um argumento e os operadores diádicos recebem dois argumentos, resultando em erro a aplicação de um destes operadores, caso tal não seja respeitado.

Qualquer outra aplicação de um dos procedimento implementados no programa que não respeite as suas regras de utilização descritas no primeiro ponto deste manual do utilizador (“O que o Programa Faz”) e que não tenha sido referida neste ponto corrente do manual (“Descrição da Informação Produzida”), resultará num valor indefinido.

Limitações do Programa

Por último, nesta secção, resta referir algumas das limitações que o programa oferece ao utilizador.

O utilizador deve estar sempre ciente de que este programa é apenas uma implementação, em Scheme, de um dado número de funções existentes em APL (ver secção “O que o Programa Faz”). Não se espera, obviamente, que este programa resolva todo e qualquer problema possível de ser resolvido em APL e com a mesma eficiência, pois este programa “simula” apenas em parte essa linguagem.

Um utilizador mais experiente em APL poderá reparar nisso mesmo, sentindo-se limitado ao trabalhar neste programa. Aconselha-se, portanto, que este manual seja lido integralmente e pela ordem em que é apresentado, para que o utilizador, independentemente da sua experiência em APL e/ou Scheme, possa usufruir ao máximo do programa que disponibilizamos.