SADの文法(2)

2011/06/08

森田 昭夫

FFS Level(1)

- 参考資料
  - SADHelp
  - 字句解析　　src/getwrd.f
  - 構文解析 & 評価器　　src/tffsa.f
- 基本動作
  - 入力列がSADScriptとして解釈可能な場合、その評価結果を表示する(tfprint())
  - 入力列がSADScriptとして解釈不能な場合、空白区切りの字句列に分解し、一語一語逐次解釈する(FFSコマンド)
    - FFSコマンドと同名の大域スコープSymbolをSADScriptで定義すると SADScript側の定義が優先するので要注意
      - Block[]構文などで回避は可能
    - エレメント名に SADScript上で意味を持つメタ文字(演算子)が含まれる場合、SADScriptとして解釈される恐れがある
      - ""などでクォートする等の対処が必要

FFS Level(2)

- FFSコマンドラインの解析
  - 字句列の先頭語で、後に続く動作が決定される
  - 入力字句は Upper Caseへ変換の上でマッチングされる
    - ビームラインのエレメント名も同様の扱い
    - 小文字を含むエレメント名にはマッチしない！
      - PRSVCASE, CONVCASEフラグで動作を変えられる
  - 予約語は、src/tffsa.f内のifの塊で判定している文字列
    - コマンド名の一部は、語幹がマッチすれば省略を認めている
    - abbrev()で分かち書きになっている後半部分
  - コマンドとなれるのは…
    - 予約語(CALCULATE, GO, …)
    - エレメント名(マッチング変数操作)
    - フラグ名(MAIN Levelフラグ操作)
  - エレメント名のマッチングではワイルドカードが使える
    - 「SAD/Tkinterの使い方」12.5.1(p.117)参照

SADScript入門(1)

コンピューター言語としての特徴

- 文法と意味論は、Mathematicaがモデルになっている
- リストによる汎用性の高いデータ表現
  - プログラム自身もある種のリストである
  - 汎用性は、2番目に古い高級言語 Lispで立証済み
- 項書き換え(Term Rewriting)による式評価
  - パラメータ付きパターンマッチングによる置換規則の適用
  - 引数へのパターンマッチによる演算の多重定義が可能
    - 引数依存の実質的な演算子オーバーロードが可能
- 簡素な文法と豊富な構文糖(Syntax Sugar)
  - 最小限の構文は、LispのS式に類似
  - ほとんどの演算子表現は、構文糖である
    - Mathematicaと異なり一部の演算子は、文法要素である

SADScript入門(2)

SAD上のデータ構造と表現形

- アトム(atom)
  - それ以上分解出来ないデータ型
  - 長さが0
  - Symbol型、Real型、String型、Pattern型
- リスト(list)
  - 広義: アトムでない全てのデータ表現
    - 頭部と呼ばれる要素と有限個の要素の順序列
    - リストの長さは、要素列に並んだ要素の数(非負整数)
    - 頭部の後に','演算子で区切られた要素列を[]で囲って表現する
      - 頭部[要素1, 要素2, …, 要素n]
  - 狭義: 特にListシンボルを頭部とするリスト
    - List[要素1, 要素2, …, 要素n]
    - {}演算子を用いた構文糖表現 {要素1, 要素2, …, 要素n}がある
  - プログラム自身もある種のリストである

SADScript入門(3)

- 参考資料
  - SAD/Tkinterの使い方 6～13章
  - 字句解析　　src/tfetok.f
  - 構文解析 & 評価器　　src/tfeval.f
- Symbol
  - 識別子
    - [$a-zA-Z][$a-zA-Z0-9]*
- Real
  - 実数　　実態は、IEEE754倍精度浮動小数点型
    - 有理数の部分集合に非数(NaN)と無限大(INF)元を加えたもの
    - 0[xX]([0-9a-fA-F]+(.[0-9a-fA-F]*)?|.[0-9a-fA-F]+)([pP][+-]?[0-9]+)
    - ([0-9]+(.[0-9]*)?|.[0-9]+)([dDeE][+-]?[0-9]+)
    - 例外則として、小数点'.'に'.'が継続する場合、最初の'.'をReal型の一部としない
      - 2..　　→ Repeated[2]

SADScript入門(3)

- String
  - 文字列　　長さのあるunibyte-character列
    - Single-Quote'"'もしくはDouble-Quote'"'で囲まれたunibyte-character列で表現する
    - Quote内の'\'はメタ文字

\n	改行文字(LF)	\r	キャリッジリターン(CR)
\f	改ページ(FF)	\t	タブ文字(H_TAB)
\e	エスケープ	\\	\文字
\'	'文字	\"	"文字
\[0-7]+	8進数によるASCIIコード		
その他	\エスケープした文字自身		

    - 行末の'\'は、行の継続を意味する

SADScript入門(4)

- 項書換え(式評価)の基礎(1)
  - 評価対象の式(リスト)の全体もしくは部分式を、項書換え規則(環境)とパターンマッチし、マッチしたものが有れば置き換える
    - 環境も同時に書き換える組み込みの項書換え規則(Unset,Set,SetDelayed,UpSet,UpSetDelayed)が存在する
    - 複数マッチした場合は、適合度、定義順で選ばれる
    - 重複する規則定義は、後から定義したものが上書きする
  - 適用可能な規則が無くなるまで置き換えを行う
    - 無限再帰な規則を定義すると、実行時エラーになる
  - 項書換え規則の集合は、Symbolによって紐付けされた連想記憶として扱われる
    - 一般には部分式の頭部Symbol
    - 部分式の要素に現れたSymbolにも紐付けできる
      - 上方値の割り当て(UpSet)

SADScript入門(5)

- 項書換え(式評価)の基礎(2)
  - 変数代入式 foo = bar(Set[foo, bar])は、Symbol fooを値 barに置き換える規則を環境に追加している(Lisp的には、Symbolの束縛(bind)という)
- 評価のタイミング
  - 部分式の要素を何時評価するかは、当該部分式の頭部の属性(Attribute)に依存する
  - 指定無しの場合は、先行評価(Eager evaluation)を適用
    - 部分式の評価が発生した段階で、部分式の要素の評価を行う
  - Hold属性を持つ場合、遅延評価(Lazy evaluation)を適用
    - 遅延評価では、要素の評価前の式自身が評価結果となる
    - 8.5.1 SetAttributes(p.82)を参照
    - 例) SetDelayed, UpSetDelayedの第2要素は Hold属性を持つ

SADScript入門(6)

- 関数(1)
  - 関数は、第一級オブジェクト(First-class object)！
    - 引数に出来る(式の要素になれる)
    - 変数に代入できる
  - 式頭部に現れたときに、式の要素を引数として値を返すオブジェクトのこと
  - 定義 f[x_] := x^2のfのこと…では有りません！
    - fに束縛された値は関数ではない！(コピー出来ていない)
    - 例では、gは Symbol fに束縛されているだけ
    - f[x_]なる式を、値 x^2へ置換する規則定義
      - x_を Patternアトムと呼ぶ
      - xは、項書換え時の仮名
  - 本物の関数は、純関数(Pure-function)
    - 関数言語的な意味では純粋ではない
      - 環境への副作用を持てる

```
  In[1]:= f[x_] := x^2
Out[1]:= (Class`s1$^2)
  In[2]:= f[1]
Out[2]:= 1
  In[3]:= f[2]
Out[3]:= 4
  In[4]:= g = f
Out[4]:= f
  In[5]:= g[2]
Out[5]:= 4
  In[6]:= Clear[f]
  In[6]:= g[2]
Out[6]:= f[2]
```

SADScript入門(7)

- 関数(2)
  - 純関数
    - 例えば、引数を2乗する関数オブジェクト:　(#1^2)&
    - &は、関数を生成する単項後置演算子
    - #は右結合なSlot演算子で、#nは純関数のn番目の引数を指す
      - %(Out)演算子と異なり、一般の実数と結合するので… #1.#2&は、Dot[#1, #2]&ではなく Times[#1, #2]&と解釈されます
    - もちろん、コピーも代入も引数渡しもOK
    - 実は、純関数だけで全ての計算が可能だったりする
      - 興味のある人は、λ計算騎士団の人に聞いてみること！

SADScript入門(8)

- 上方値
  - 式の頭部Symbolではなく、要素内Symbolに束縛された値
  - 実質的な組み込み演算子のオーバーロードを実現できる
  - 例) BeamLine[___]への演算規則は、組み込みTimes演算子への遅延上方値束縛(UpSetDelayed)で実装されている
    - 組み込み演算子自身のオーバーロードは出来ない
    - 上方値束縛で、頭部依存した演算のオーバーロードが可能
      - 汎用演算のオーバーロードは、Class内なら可能
      - DynamicLink[] + dlfunalloc()によるオーバーライドは可能(再定義)

```
-BeamLine[x__] ^:= BeamLine@@(-Reverse[{x}]);
(n_?(#>=0&))*BeamLine[x__] ^:= BeamLine@@Flatten[Table[{x}, {n}]];
n_*BeamLine[x__] ^:= Abs[n]*BeamLine@@(-Reverse[{x}]);
```

(Packages/beamline.nより)

    - BeamLine[___]をユーザー定義型と解釈すれば、ユーザー定義型へ対する演算子オーバーロードと見做せる
  - Set系演算子や Clear関数のオーバーロードは、慎重に！
    - 間違えると、再定義や定義の消去が出来なくなります

SADScript入門(9)

- Symbolスコープ
  - Symbol毎に識別子としての有効範囲(スコープ)がある
    - 大域スコープ
    - クラススコープ
    - 局所スコープ
  - Symbol参照解決では、より内側のスコープが優先する
  - スコープを出る時、Symbolは破壊(UnSet)される
  - スコープ操作
    - 局所スコープの生成　　Module, With
    - 大域スコープの再定義　　Block

SADScript入門(10)

- 予約Symbol
  - SADScriptインタープリタ上で予約されているSymbol
    - 組み込み演算子、組み込み関数
    - 定数(True, False, Pi, ...)
    - インタープリタ制御変数
      - NPARA, $FORM, $WildCard, $SortMethod, MatchingResidual, ...
    - Package関数
    - 他のレベルへのアクセスSymbol
      - 例) fooが MAINレベルのフラグ名の時、?fooはフラグの真偽値を返す
  - 組み込み関数、定数Symbolは src/tfinitn.fを参照
    - 一部は、src/*.cで dlfunalloc()経由で定義されている
  - Package関数 Symbolは、Packages/init.nと Autoloadされる Packages/*.nを参照
  - インタープリタ制御変数は、一般的なスコープ規則から外れるので要注意
    - Block[]等で変更できない

SADScript入門(11)

例外的な構文・評価規則

- 暗黙の積(Times)演算子
  - 式と式の間に演算子が存在しない場合、積演算子があるものと見做す
  - ただし、加算・減算2項演算子はこれに優先する
    - でないと 'a + b'は、Times[a, +b]と解釈されるよ！
- 暗黙のNull
  - リストの要素が必要な場所に空白文字のみがある場合、Nullがあるものと見做す
    - 例) f[a, b, c,] → f[a, b, c, Null]
- 暗黙のPart演算子
  - 文字列ないしは狭義のリストに束縛されたSymbol fooに対して、暗黙の項書き換え規則 foo[argv___] :> Part[foo, argv]が存在すると見做す

May the source be with you.