1　予備知識

Maxima には式の一部を取り出すための関数として part や inpart などが用意されています。

```
(%i1) h: a + b*sin(c+d) + e*log(f + g);
(%o1)                  e log(g + f) + b sin(d + c) + a

(%i2) part(h, 2);
(%o2)                           b sin(d + c)

(%i3) part(h, 2, 2);
(%o3)                            sin(d + c)

(%i4) part(h, 2, 2, 1);
(%o4)                               d + c
```

関数 part（や inpart）は汎用性が高く大変重宝しますが、式が複雑になると、番号による指定が困難になることがあります。例えば、

$$a\left(b+c\left(d+e\int f(x)\,dx\right)\right)$$

この程度の式でさえ、積分 $\int f(x)\,dx$ を取り出すことは容易ではないでしょう。非積分関数 $f(x)$ が単純な式なら、直接入力した方が速いかもしれません[*1]。

2　対象物が 1 つのみの場合

式の一部を抽出する関数を定義してみます。抽出対象となる部分式が全体式の中に 1 しか存在しないことが分かっている場合は、次の通りです。

– f_pickup.mac –

```
1  f_pickup(F, V) := block([partswitch: true],
2      if not(freeof(V, F)) then (
3          for i: 1 while (true) do (
4              G: part(F, i),
5              if not(freeof(V, G)) then return(f_pickup(G,
                   V))
6              elseif G = end then return(F)
```

[*1] 関数 part 等で取り出すことが困難な場合の対応策として、1 次元表示（display2d: false）や wxMaxima を利用してのマウスによるコピー・ペーストが考えられます。

```
7             )
8         )
9 );
```

関数 part は範囲外の幅や深さを指定すると、標準ではエラーを返しますが、変数 partswitch を true に設定すると（デフォルトは false）エラーの代わりに end を返します（1 行目）。Maxima には「式の幅や深さ」を取得する関数が用意されていないため、end になるまで処理を繰り返すことで、式全体を走査（traverse）します。

「式」の中に「対象物」が含まれているか否かを調べるには、関数 freeof を利用します。全体式（親）F の各部分式（子）G に対して、対象物 V が含まれているか否かを調べて、含まれていれば、再帰呼び出しにより、「部分式 G の部分式」を調べ（5 行目）、最後まで調べてどこにも対象物含まれていなかった場合には、親を返します（6 行目）。

実行例は次の通りです。

```
(%i1) load("f_pickup.mac");
(%o1)                          f_pickup.mac

(%i2) a*(b + c*(d + e*integrate(f(x), x)));
                                  /
                                  [
(%o2)                    a (c (e I f(x) dx + d) + b)
                                  ]
                                  /

(%i3) f_pickup(%, integrate);
                                   /
                                   [
(%o3)                              I f(x) dx
                                   ]
                                   /
```

6 行目の処理から分かる通り、対象物（上の例ではインテグラル）が入れ子になっている場合は、一番内側の部分式が出力されます。

③ 対象物が複数有り得る場合

前節の関数 f_pickup では、式の中に対象物が複数存在する場合、最初に見つかったものしか抽出できません。対象物のすべてを抽出するには少々工夫が必要です。

– pickup.mac –

```
1 pickup(f, v) := block([f_pickup, L: []],
2     f_pickup(F, V) := block([partswitch: true, d],
3         if not(freeof(V, F)) then (
```

```
4                   d: 0,
5                   for i: 1 while (true) do (
6                       G: part(F, i),
7                       if not(freeof(V, G)) then (
8                           d: 1,
9                           f_pickup(G, V)
10                      ) elseif G = end then (
11                          if (d = 0) then L: cons(F, L),
12                          return()
13                      )
14                  )
15              )
16      ),
17      f_pickup(f, v),
18      L
19  );
```

前節の関数 f_pickup をサブルーチン化し、対象物が見つかる毎に、リスト L に蓄積していきます（11 行目）。

前節の関数との最大の違いは、部分式 G の中に対象物 V が見つかった場合の処理です（7 行目～ 9 行目）。return する代わりに、変数 d に 1（0 以外の数）を代入し、再帰呼び出しを実行します[*2]。抽出すべき式は、「その部分式に対象物を含まない式」ですから、同じ深さの部分式全てを調べて、変数 d が初期値（0）のままの場合にリスト L に蓄積し、ループを抜けます（10 行目～ 12 行目）。

実行例は次ページの通りです。

[*2] return すると、以降の処理が実行されずにループを抜けます。

```
(%i1) load("pickup.mac");
(%o1)                           pickup.mac

(%i2) a*(b + c*integrate(f(x), x))/(d + integrate(g(x), x));
                                  /
                                  [
                            a (c I f(x) dx + b)
                                  ]
                                  /
(%o2)                             ------------------
                                 /
                                 [
                                 I g(x) dx + d
                                 ]
                                 /

(%i3) pickup(%, integrate);
                              /          /
                              [          [
(%o3)                        [I g(x) dx, I f(x) dx]
                              ]          ]
                              /          /
```

ここで定義した関数 `f_pickup` や `pickup` は、（もちろん）インテグラル $\int$ だけでなく、和 $\sum$ や三角関数等にも利用できますが、前節の最後にも注意したように、入れ子になっている場合は、一番内側の対象物のみが抽出されます。

```
(%i4) cos(cos(x + 1) + 2) + cos(sin(x + 3) + 4);
(%o4)              cos(sin(x + 3) + 4) + cos(cos(x + 1) + 2)

(%i5) pickup(%, cos);
(%o5)                   [cos(x + 1), cos(sin(x + 3) + 4)]
```

作成日：平成 21 年 10 月 12 日
ソフトウェア：Maxima 5.19post & ECL 9.10.2