MODELING LANGUAGE REFERENCE

＜日本語版＞

更新：2013 年 6 月 1 日

「誤訳、不適切な訳があるかもしれませんので、

英文と併読なさるようにしてください。」

目次

Introduction : Main principles of the modeler 2
Syntax and lexical analysis 2
Global structure : グローバル構造 2
Comments : コメント 2
Identifiers : 識別子 2
Keywords : キーワード(予約語) 3
Literals : リテラル 3
String literals : 文字リテラル 3
Escape sequences : エスケープ・シーケンス(エスケープ文字列、拡張表記) 3
Floats : 浮動小数点 4
A word on white spaces : スペース文字の使用 4
Values and types: 値と型 4
Nil 4
Integers : 整数型 4
Floats : 浮動小数点 4
Strings : 文字列 4
Maps : マップ 4
Files : ファイル 5
LS expressions : LS 表現式 5
Variables : 変数 5
Relational operators : 関係演算子 6
Arithmetic operators : 算術演算子 6
Logical operators : 論理演算子 7
Conditional (ternary) operator : 条件演算子(三項演算子) 7
Operator precedence & associativity : 演算子の優先順位と結合規則 7
Statements : ステートメント 7
The assignment statement : 代入文 7
The local statement : ローカル宣言ステートメント 8
If statements : If 文 8
For statements : For 文 8
Iterated assignment statements : 反復条件文 9
while/do-while statements : while/do-while 文 9
constraint/minimize/maximize statements : constraint/minimize/maximize ステートメント 9
Function definition : 関数定義 10
Function call : 関数の呼び出し 11
Variadic function call : 可変個引数の関数呼び出し 11
Execution model : 実行モデル 12
Command line : コマンドライン 12
Built-in functions and variables : 組み込み関数と変数の一覧 13
I/O Library 13
String library 14
Map library 15
Modeling library 16
Main error messages: 主なエラーメッセージ 17

Introduction：Main principles of the modeler

はじめに：モデラー基本概念

当マニュアルは、モデラーで提供されるシンタックス、意味論及びコア関数について記述しています。モデラーに関する詳細は、クイック・スタートガイドを参照ください。

LSP 言語は、LSP モデルの構文木(モデルの検証、解の検証を行うため)の生成およびその構文木解をパラメータ化します。LSP モデルは、演算子およびキーワード(minimize, maximize, constraint)で定義します。LSP 言語は、モデルの定義を簡潔にするため、命令型プログラミング及び構造型プログラミング(loops, conditions, variables...)で関数を提供します。LocalSolver のモデラーは、モデリング言語であり、命令型プログラミング言語でもあります。このインタプリタ言語は、最新のスクリプト記述機能を統合：強力かつ動的な変数の型付け(型は変数ではなく値で保持)および暗黙的な変数宣言を提供します。数多くの組み込み関数はモデリングおよびプログラミングの双方に使用可能です。

Syntax and lexical analysis

シンタックスと構文解析

Global structure ： グローバル構造

LSP は、強力な型付けかつブロック構造のプログラミング言語です。LSP プログラムは、LSP ファイル内の関数の連続で生成します。各関数は、表現式のリストまたは中括弧で囲う命令文からなるコード・ブロックを含みます。各コード・ブロックは内部のブロックに値を渡すことができます。モデリング言語は、大文字と小文字を区別します。各命令は';'で囲い、他の表現と区別させます。

符号化モデリング言語は、ASCII および拡張 ASCII 文字(8bit のみ)を使用頂けます。UTF-8、UTF-16 及びその他符号化形式は対応していません。非対応の符号化形式を使用された場合、エラーまたは、異常動作が生じます。

Comments ： コメント

コメントは構文解析では、無視されます。コメントはプログラムの処理時にないものとして扱われます。下記の3種類が使用頂けます。

・単一行コメント：単一行にコメントを入れる場合に使用します。コメントの先頭に'//'を入力します。'//'.から行末までがコメントとして扱われます。

・複数行コメント：複数行にまたがるコメントを入れる場合に使用します。コメントはコメントの先頭に'/*'を入力し、'*/'でコメントを終わらせます。複数行コメントは、'*/'でコメント終わりを宣言してください。複数行コメントのネスティング(入れ子)は禁止されています。従って、/*コメント 1/*コメント 2*/*/と入力できません。

・シバン行コメント：シバン行コメントは'#!'をコメントの先頭に入れます。シバン行コメントは、LSP プログラムの 1 行目で、特別なコメントのみに使用します。Unix 系では、プログラムの処理時(ここでは LocalSolver)にインタープリタのパスを指定するために使用します。 上記以外の行またはファイル内で、'#!'を使用した場合、エラーになります。

Identifiers ： 識別子

LSP ファイルの中で、変数や関数名を付ける場合に、識別子を使用します。識別子は下線と英数字で構成されたもののみ有効です。BNF 文[a-zA-Z_][a-zA-Z0-9_]*

- identifier：IdeNtiFier と異なる
- _ident：有効な識別子
- 0ident：無効な識別子(先頭が数字)
- for：無効な識別子(予約語は、下記を参照ください)

・Keywords：キーワード（予約語）

キーワードは、モデラーに対して特別な意味を持つ予約済みのワードです。キーワードは、識別子（変数名、関数名）に使用することはできません。文法規則として使用します。詳細はこのマニュアルの巻末を参照ください。いくつかのキーワードは、将来的使用を目的に定義されています。

特別な意味を持つ予約語；

- true, false, nil
- for, in, if, else, do, while
- minimize, maximize, constraint
- function, return, local, include

将来的な使用目的

- const, var, self, this
- continue, break, goto, switch, case
- throw, class, final, object,

Literals：リテラル

リテラルは、定数（固定データ値）を表現します

String literals：文字リテラル

文字列リテラルは、"'を先頭に入力し、"'で終わらせます。（ダブルコーテーションで囲む）
文字列は複数行に跨いで入力することができます。文字列の長さに制限はありません。ダブルコーテーションの間は、バックスラッシュやシングルクコーテーションを除く、全ての記号を使用できます。

```
"Multi line
                Literal"
```

は有効

```
"string literal \ "
```

は無効

Escape sequences：エスケープ・シーケンス（エスケープ文字列、拡張表記）

いくつかの文字は、エスケープ・シーケンスで記述する必要があります。対応する文字または ASCII の文字で記述し、＼(バックスラッシュ)を先頭に入力します。規定されたエスケープ・シーケンス：＼＼バックスラッシュ(＼)＼‘シングルコーテーション(互換性の問題があるため記述)＼``ダブルコーテーション＼t ASCII HT(水平タブ)＼r ASCII CR(復帰)＼n ASCII LF(改行)＼b ASCII BS(後退)＼f ASCII FF(改頁)

規定されていないエスケープ・シーケンスを導入した場合、パーサーにより（構文解析）エラーとなります。従って'＼c'は有効なエスケープ・シーケンスとして規定されていないため、"foo ＼c"はエラーです。

Integers：整数

LocalSolver は 64bit で符号付整数型を処理します。
10 進法のみ記述可能です。
数値は 0 から始まらない 0 から 9 の数字を使用する数字列です。

LSP ファイル内で記述した数値が、表現できる数値の範囲を超えた場合、パーサーの検証時、エラーになります。

- 01234 は、無効な数値（0 で始まっているため）
- 10000000000000000000 は無効な数値（表現可能な数値の範囲を超えているため）

Floats：浮動小数点

LocalSolver はポイント表記法（例えば 3.467）や、指数表記（例えば 8.57e-11）を使用し倍精度浮動小数点を処理します。
リテラル +Inf およびリテラル -Inf は正の無限大、負の無限大を表示します。

A word on white spaces：スペース文字の使用

モデラーにおいてスペース文字および、改行は特別な意味を持たず、ないものとして扱われますが、スペース文字は、2 つのキーワード、2 つの識別子、または 2 つのリテラルを分割するために必要です。

Values and types: 値と型

Nil

単一の値 Nil は値が（データが）存在しないという意味を持ちます。

Integers：整数型

整数型は、64bit の正と負の整数です。最小整数は、 9,223,372,036,854,775,808 で、最大整数は、9,223,372,036,854,775,807 です。リテラルの真と偽は、それぞれ 0 または 1 に等しい定数です。モデラーに対する明示的なブール型変換を行う必要はなく、複数の関数または、ステートメントが、整数が 0 または 1 と等しい、すなわち「ブール値である」と自動的に判断します。

例：下記のコードはエラーになります。

```
if (-38) print("foo"); // will throw the error    "Cannot cast 'int' to 'boolean'"
```

注：print(true)は 1 を表します。

Floats：浮動小数点

浮動小数点は IEEE754 規格のバイナリ浮動小数点数形式の倍精度浮動小数点数（64bit）です。

Strings：文字列

文字列は、ASCII 文字または拡張 ASCII 文字です。
注：非文字型が定義されているので、文字 a は文字長「1」です。
文字列の扱いに関する組み込み関数の詳細はセクション「String Library」を参照ください。

Maps：マップ[配列]

マップは、キーに関連付けて値を格納するデータ構造です。キーは文字列あるいは整数です。（必ずしも連続的ではない）
値は、任意の型で指定できます。キーに関連付けた値は指定するあるいは、括弧を使用し返されます。注；マップ a のキーk に値が存在しない場合、a[k]は nil 値を返します。既存のキーの値を変更するには、前に関連付けした値を上書きします。
マップは、連想表であり連想配列です。（非配列型が言語で定義されています。）マップは一番目の要素を代入する時、暗黙的に宣言されます。また空マップは、map()関数を使用するか、ブレース{}.を使用し定義します。ブレースを使用し、値の配列を代入することで、マップの定義も可能です。この場合、値は整数キー、0、1、2…に割り当てられます。

```
a[9] = "abc"              // assign value "abc" to key 9
a["abc"] = 9. // assign value "abc" to key 9

a = map("z", 9); // a[0] = "z", a[1] = 9
a = {"z", 9};             // a[0] = "z", a[1] = 9

a["a"] = "abc";           // a[0] = "z", a[1] = 9, a["a"] = abc
a[-3] = "xyz";            // a[0] = "z", a[1] = 9, a["a"] = abc, a[-3] = "xyz"
```

マップの扱いに関する組み込み関数の詳細は「Map library」に掲載しています。参照ください。

Files：ファイル

ファイル形式は built-in I/O ライブラリで開いたファイルを参照します。（セクション「I/O ライブラリ」を参照ください。）

LS expressions：LS 表現式

LS 表現式は、LSP 言語で定義された数理的モデルの変数および表現式です。関係演算子″<-″で導入します。

.

下記の例題″ x_0, x_1 と knapsackWeight が LS 表現式です

```
x_0 <- bool();
x_1 <- bool();
knapsackWeight <- 10*x_0 + 60*x_1
```

モデラーの観点から言うと、LS 表現形式しかありません。たとえ、以前定義した LS 表現式が演算子に依存していても、ソルバーは、違う意味として表現式を処理します。特に、ソルバーは制約として数値を宣言することを禁止し、ブール値のみに適用するために論理演算子が必要です。

Variables：変数

変数は値を参照します。変数は識別子により区別されます。注：変数は型を持ちません。（型は、値で保持され、変数は、単なるコンテナです。）例:下記のコードは有効です。

```
x = 2;
x = "a";
```

変数を事前に、宣言する必要はありません。いかなる変数もデフォルトで nil 値に関連付けられています。変数の参照範囲は、グローバルまたはローカルとして宣言することが可能です。グローバル変数は、全てのスコープから参照できますが、ローカル変数のスコープは、その変数が宣言されたブロック内およびその定義済みブロックで囲ったブロック内でのみ参照可能です。

[ローカル変数の例]

- 関数の引数
- ループ変数、つまり、for 文に記述された変数（あるいはループ条件文または可変個引数の関数呼び出し）
- 予約語「ローカル」で使用された変数

全ての変数の初期値はグローバルです。下記の初期値はローカル変数に設定されています。ローカル変数の使

用時、同名のローカル変数が既に定義されていて、現在のコードブロックで有効範囲になっている場合、エラーが出力されます。ある名前を持つグローバル変数が既に定義されていて、その名前と同名のローカル変数を使用した場合、その名前は、その有効範囲でローカル変数として関連付けられます。

例えば、下記のコードを実行した場合、"Variable 'i' already defined."とエラーになります。

```
function foo(i) {
    for [i in 1..10] print(i);
}
```

上記とは、反対に下記のコードは、i =2 をグローバル変数として使用したので、有効となり、単に、for 文ではマスクされたグローバル変数となります。

```
function foo() {
    i = 2;
    for [i in 1..10] print(i); // the output will be 12345678910
}
```

Expressions：表現式

Relational operators：関係演算子

LS 表現ではない型に値を代入する場合、比較が真の値を取るとき(その他の場合は 0)、演算子<, >, ==, >=, <=,および!=は整数値は 1 を返します。演算子<, >, >=および<=は nil 値、文字列、マップあるいはファイル形式に利用できません。つまり、数値および、LS 表現式のみに利用できます。演算子==および!=は nil 値などを比較する場合を除き、マップあるいはファイルで利用できません。演算子==や!=を含む 2 つの文字列を比較する時、2 つの文字列が、厳密に同一である場合のみ文字列は等しいと考えられます。

例：

```
println(8 < 9.2); // will print 1
a = {1,8};
b = {1,8};
println(a == b); // will throw an error since == is not defined on maps
println("abc" <= "abcde"); //will print 1
println("abc" == "aBc"); // will print 0 since comparison is case-sensitive
```

演算子<, >, ==, >=, <=,および!=で評価した場合、LS 表現となり、数理モデルに 2 つの項の間の評価を表している LS 表現を返します。
このトランスタイピング規則は全ての算術演算子に利用可能です。:全ての被演算子が数値の場合、結果も同種となり、被演算子の 1 つが LS 表現になり次第、LS 表現が返されます。

Arithmetic operators：算術演算子

演算子+, -, *, /, %は加算、減算、乗算、除算および剰余演算を表します。LS 表現ではない値を利用した場合、演算子は数値を返します。演算子+, -, *, /は被演算子を浮動小数に変換し、2 つの被演算子のうち 1 つが浮動小数の場合、浮動小数を返します。反対に、2 つの被演算子が整数の場合、整数が返されます。中でも、整数における除算は整数の除法です。最後に演算子%は浮動小数に使用できません、整数値で整数値を割る場合のみに使用できます。関係演算子に関して、被演算子の 1 つが LS 表現になり次第、算術演算子は LS 表現を返します。特別な演算子:'+'は複数の文字列を結合し 1 つの文字列にする時に使用する連結演算子としても使用できます。2 つの被演算子のうち 1 つの被演算子が文字列の場合、文字

列と数値を連結させると、文字列の方が強いので、数値は文字列に変換されます。例: abc"+12 は "abc12".を返します。

Logical operators：論理演算子

引数が 0 の場合、単項演算子！は 1 になり、他の場合は 0 になります。x && y の演算は最初に x を評価します。x が 0(偽)ならば、その値が返されます。他の場合は、y が評価され、得られた値が返されます。x ‖ y の演算は最初に x を評価します。x が 1(真)ならば、その値が返されます。その他は、y が評価され得られた値が返されます。
被演算子の 1 つが、LS 表現の場合、論理演算子は数理モデルの演算を表現している LS 表現を戻します。その後、2 つの被演算子を評価します。

Conditional (ternary) operator：条件演算子(三項演算子)

三項演算子'A ? B : C'は最初に式 A を評価します。A が 0 あるいは 1 と等しい整数ではないとき、エラーが出力されます。他の場合は、A が 1 ならば、B が返され、その他の場合、C が返されます。A、B または C が LS 表現ならば、数理モデルで条件演算子を表現している LS 表現を戻します。三項演算子を使った場合、全ての被演算子が評価されます。

Operator precedence & associativity：演算子の優先順位と結合規則

下記の表は、演算子の優先順位です。同一の項目内の演算子は優先順位が同格です。優先順位が高い順に記載しています。優先順位の高い演算子は、優先順位が低い演算子に先行して、評価されます。代入演算子を除き、全ての二項演算子は左から右に評価されます。:代入演算子は右から左に評価されます。

Operators	
Unary	- + !
multiplicative	* / %
additive	+ -
relational	< > <= >=
equality	== !=
and	&&
or	||
ternary	? :
assignment	= <-

Statements：ステートメント

The assignment statement：代入文

LSP プログラムは、2 種類の代入文形式が使用できます。

- 基本形式「変数＝値」の式は、変数定義がまだなされていない場合、指定変数はグローバル変数として定義されます。その後、前の値があっても、その式のその値は与えた値に変更されます。
- link 文は LS 表現あるいは数値に適用します。変数がまだ定義されていない場合、指定変数をグローバル変数として定義します。その後、前の値があっても、式の値は与えた値に変更され、この変数は数理モデルに追加されます。

例:

```
a = true;    // a = 1
b = 9;       // b = 9
c = a + b;  // c = 10
 c = a * b;  // c = 9
c = a == b; // c = 0
c = a < b;  // c = 1

x<- 1;       // x = 1 and is added to the mathematical model
<- bool(); // y is a new decision variable and is added to the mathematical model
z <- x+y;    // z is an ls expression and is added to the mathematical model
```

The local statement：local 宣言ステートメント

local 宣言ステートメントの「local 変数」あるいは「local 変数＝値」は与えた名前で新しい local 変数を導入し、その値を nil または指定値に設定します。既に同名の global 変数が定義されている場合、この local 変数名が参照範囲にある時、global 変数はマスクされます。同名の local 変数が既に定義されている場合は、エラーが出力されます。

```
local i;
local j = 2;
```

注:LS 表現式は local として定義できません。

If statements： If 文

if (C) S_true; else S_false; は条件式です。条件 C が真ならばステートメント S_true を実行し、(つまり、C が 1 ならば) C が偽ならばステートメント S_false を実行します。(つまり C が 0 ならば)注 else 条件分岐はオプションです。{}を使用し、ブロック・ステートメントを宣言することで、複数のステートメントを同時に実行できます。

- 表現式が 0 または 1 に等しい整数でない場合、エラーが出力されます。
- 条件付き三項演算子？:を使用するとコンパクトに記述することができます。

```
if (0) c = "ok";
if (true) c = "ok";
if (2) c = "error"; // ERROR: invalid condition
```

For statements： For 文

for [v in V] S;ステートメント S は V において全ての値を取る v で反復処理を行います。V はレンジあるいはマップの場合もあります。変数 v はこの for 文およびネスト化したブロック内で参照可能な local 変数として暗黙的に宣言されます。from..to 文で宣言されたレンジは、最大値と最小値を含みます。マップを得た場合、イタレーションはキーの増加率に従い、マップの値で実行します。マップの組は(キーと値)for [k,v in M].文を使用し、反復処理を行うことも可能です。

```
for [i in 0..2] a[i] = i + 1; // a[0] = 1, a[1] = 2, a[2] = 3
s = 0; for [v in a] s = s + v; // s = 6
s = 0; for [k,v in a] s = s + k + v; // s = 9
```

for [v in V : C]は、フィルタした反復処理です。変数 v は条件 C を満たしている V の値のみを取ります。注:ステートメントにフィルタ機能を使用することで、下記のように、簡潔にネスト構造を構築できます。

```
for[i in 0..9]
    for [j in i+1..9 : j % 2 == 0]
    for [k in j+2..9]
      a[i][j][k] = i + j + k;

for[i in 0..9][j in i+1..9 : j % 2 == 0][k in j+2..9] // compact
  a[i][j][k] = i + j + k;
```

全てのループ制御文に対し、{}を使用し、ブロック・ステートメントを宣言することで複数のステートメントを同時に実行できます。

```
for[i in 0..9][j in i+1..9][k in j+2..9] { a[i][j][k] = i + j + k;
b[i][j][k] = i * j * k;
}
```

Iterated assignment statements：反復条件文

a[v in V] = f(v);反復条件式を表し、for [v in V] a[v] = f(v);に相当し、同様に a[v in V] <- f(v)は、反復条件式を表し、for [v in V] a[v] <- f(v)に相当します。これらの反復条件式は、モデルを簡潔かつ読みやすく作成するときにとても便利です。前のセクションで解説した for 文と同様に、イタレーションをネスト構造として構築することやフィルタリングが可能です。

```
for[i in 0..9][j in i+1..9][k in j+2..9]
  a[i][j][k] = i + j + k;
a[i in 0..9][j in i+1..9][k in j+2..9] = i + j + k; // very compact!
```

while/do-while statements：while/do-while 文

while (C) S;条件 C が真(すなわち 1 と等しい)の間、スタートメント S の繰り返し処理を実行します。
do S; while (C)は、条件 C に関係なく、一回は必ずステートメント S を実行します。その後、条件を後判定して反復します。

constraint/minimize/maximize statements：constraint/minimize/maximize ステートメント

- constraint c; 数理モデルに制約条件として LS 表現 c を追加する。
  c の型がブール型 LS 表現(論理演算子あるいは関係演算子、制約を 0 または 1 で表現可能)でない場合、エラーが表示されます。
- minimize c; 数理モデルで最小化させるオブジェクトとして LS 表現 c を追加する。
- maximize c; 数理モデルで最大化させるオブジェクトとして LS 表現 c を追加する。

少なくても必ず 1 つ、オブジェクトを定義します。

Functions： 関数

LSP プログラムは、関数の連続で構成されます。関数の外側でステートメントおよび表現式の記述はできません。

Function definition： 関数定義

関数は、キーワード function を使用し宣言します。引数は、丸括弧内に与え、カンマで、区切り、そのコードを {} で囲います。引数は複数列記することが可能です。引数を与えない場合も、関数名()と表記します。

```
function isEven(v) {
  if (v % 2 == 0) return true;
  else return false;
}
```

多くの命令語と同様に、数(整数および浮動小数点数)はプリミティブ型です。文字列、マップ、ファイルおよび LS 表現は、参照型です。関数内の int あるいは float の引数に適用された変更は、呼び出し元の関数に渡された数値には反映されません。

```
function f(i) { i = 8;}
function g() {
  local i = 2;
  f(i);
  print(i); // will print 2
}
function h() {
 i = 2;
f(i);
print(i); // will print 2 because the global variable i was masked in f by the local parameter i
}
```

逆に、関数内で参照型の引数のコンテンツに適用された変更は、関数の外側で永続化します。

```
function f(a) {
a[2] = 8; // applies to the original map
a = {2,3,4}; // no impact on the map since it just assigns locally a new map to the name "a"
}
function g() {
 a[0] = 0;
f(a);
print(a); // will print [   0 => 0      2 => 8 ]
}
```

関数に変数を導入する場合、同名で他の場所で宣言されている変数に影響を与えないために、local の宣言子の使用を推奨します。

```
function computeSumOfEvenNumbers(a,b) {
  local total = 0;
  for [v in a..b : isEven(v)]
      total = total + v;
  return total;
}
```

Return statement： Return ステートメント

return value; 関数の実行を中止し、値を返すことができます。return;は return nil の省略です。注：return ステートメンを実行せずに、関数本体の最後に到達して終了した場合は、戻り値は nil 値になります。その結果、全ての関数が値（おそらく nil）を戻します。戻り値の型は自由に定義でき、全ての分岐を同様に定義する必要はありません。

下記のコードは有効です。

```
function f() {
if (a) return 2;
else return "zz";
}
```

Function call：関数の呼び出し

f(a,b,c)は引数 a、b、c で関数 f を呼び出します。引数は、宣言された順に評価されます。セクション 6.1 で説明したようにユーザー定義関数は、引数として定数を持ちますが、複数の組み込み関数では引数の数を可変個とすることができます。例えば、関数 println は引数の数を可変個とすることができ、下記のコードが有効となります。

```
println();
println("abc");
println("abc","de");
println("abc","de",180);
```

Variadic function call：可変個引数の関数呼び出し

セクション 5.3 で for 条件文に導入した反復処理形式を関数の引数に指定することができます。主に、可変個引数の関数に便利ですが、全ての関数にも有効です。例えば、println[i in 1..4](i,″ ″)は println(1,″ ″, 2,″ ″, 3,″ ″,4,″ ″)と等しい処理を行います。前章で導入した for 文と同様に、このイタレーションもネスト化およびフィルタリングが可能です。

```
println[k in 1..2][i in 1..9 : i % 2 == 0](i, " ");
```

この機能はモデリング表現や、数値計算に大変便利です。

例:

```
x <- sum[i in 1..10](w[i] * y[i]);   // declare x as a sum of 10 terms
v = min[j in 1..5][k in 1..9](m[j](k)); // retrieve the smallest element of a matrix
v = prod[v in a : v != 0](a[j]);   // compute the product of non-zero elements of an map a
```

Execution model：実行モデル

LSP は、メインプログラムがなく、input(), model(), param(), display(), output()の 5 つの基本的なファンクションから成ります。また全ての LSP プログラムに関数 model を実装する必要がありますが、他の関数は必要に応じて使用します。

```
// The smallest possible LSP program
function model() {
    minimize 0;
}
```

プログラムの実行の流れは下記の通りです。

- プログラムに関数 input()がある場合、function input() を呼び出す
- function model()を呼び出す
- プログラムに関数 param()がある場合、function param()を呼び出す
- 数理モデルで LocalSolver の探索を実行。この探索の間、プログラムに関数 adisplay()がある場合、この関数は、毎秒で呼び出される
- プログラムに関数 output()がある場合、function output()を呼び出す

5 つの関数は、下記の用途に適しています。

- input：ファイルからデータの宣言、あるいは読み込む
- model：最適化モデルの宣言
- param：実行前に local-search ソルバーのパラメータを確認する
- display：求解中に、コンソール、あるいはファイルに情報を表示する
- output：解を得たのち、コンソールに入出力、あるいはファイルに解を入出力する

Command line：コマンドライン

LSP プログラムはコンソールで下記のコマンドを使い実行されます。

```
localsolver file /<argument*/>
```

ファイルは有効な LSP プログラムを含むテキストファイルです。

各アーギュメントはグローバル変数に割り当てた値です。このプログラムの実行を開始する前に、これらの代入文が実行されます。

例：

```
localsolver  test.lsp  x=12  y=abc  z="a  string  with  whitespaces"  t=true
```

整数値 12 を可変 x に、可変 y に文字列 abc、文字列″a string with whitespaces″は可変 a に、値 1 は可変 t に割当てます。
マップに割り当てた値は、特別な構文で記述します。a=z,12 はキー0 に関連付けられた z でマップ@@a@を生成し、12 はキー1a=8:z,に関連付けられます。
akey：12 はキー8 に関連付けた z でマップ@@a@を生成し、12 がキー″akey″ に関連付けられます。注：shell 関数がこのコマンドを a=3 と解釈するため、a={2,3} の構文は無効です。

Built-in functions and variables：組み込み関数と変数の一覧

I/O Library

- f = openRead("data.in"); Opens file "data.in" in reading mode. The only argument of this function is the name of the file. It returns a value of type file or throws an error if the file cannot be opened.
- f = openWrite("data.out"); Opens file "data.out" in writing mode. The only argument of this function is the name of the file. It returns a value of type file or throws an error if the file cannot be opened.
- f = openAppend("data.out"); Opens file "data.out" in append mode. The only argument of this function is the name of the file. It returns a value of type file or throws an error if the file cannot be opened.
- close(f); Closes the file. The only argument of this function is a value of type file. Note that once no more variable refers to a file value, the file is automatically closed.
- eof(f) Returns true if the end of file is reached. The only argument of this function is a value of type file.
- i = readInt(); Prompt the user for an integer (in the console standard input) and returns this integer.
- i = readDouble(); Prompt the user for a floating-point number (in the console standard input) and returns this float.
- i = readInt(f); This function returns the next integer in the specified file, and position the cursor at the next non blank character following this integer, or at the end of the file if file contains only・ blanks after this integer. An error is thrown if the file contains no integer until the end of file. The only argument of this function is a value of type file. It returns a value of type int.
- i = readDouble(f); This function returns the next floating-point number in the specified file, and position the cursor at the next non blank character following this float, or at the end of the file if file contains only blanks after this float. An error is thrown if the file contains no float until the end of file. The only argument of this function is a value of type file. It returns a value of type float.
- s = readln(); Prompt the user for a line (in the console standard input).
- s = readln(f); This function returns the next line in the specified file, and position the cursor at he start of the following line, or at the end of the file if file contains no line after the next one. The only argument of this function is a value of type file. It returns a value of type string.
- s = readString(f); This function returns the next string in the specified file (that is to say the next sequence of non blank characters), and position the cursor at the next non blank character following this string, or at the end of the file if file contains only blanks after this string. An error is thrown if the file contains no string until the end of file. The only argument of this function is a value of type file. It returns a value of type string.
- print(s1,s2,...) Prints the string in console. This function is variadic that is to say that it accepts any number of parameters. Non-string parameters will be converted to strings. Example: print("s = " + s + "\n");

- print(f,s1,s2,...) Prints the string in file. Parameter f is of type file. Others parameters will be converted to

strings.Example:print(f, "s = " + s + "\n");

- println(s1,s2,...) Prints the string followed by a line feed in console. This function is variadic that is to say that it accepts any number of parameters. Non-string parameters will be converted to strings. Example: println("s = " + s);
- println(f,s1,s2,...) Prints the string followed by a line feed in file. Parameter f is of type file. Others parameters will be converted to strings.
- error(msg); Prints an error message and terminates the program. The only argument of this function is a value of type string representing the message to be written before terminating the program.

String library

- toInt(str); This function converts a string value into an integer value. If the string does not represent an integer, an error is thrown. The only argument of this function is a value of type string representing the value to convert. This function returns a value of type int. Example: i = toInt("123");.
- toDouble(str); This function converts a string value into a floating-point number value. If the string does notrepresent a float, an error is thrown. The only argument of this function is a value of type sring representing the value to convert. This function returns a value of type float. Example: i = toDouble("123.45");
- split(str,delim) This function splits the string given as first parameter on occurences of the pattern given as second parameter. Example: split("a::b::c::d", "::") splits string "a::b::c::d" into substrings (as a map) according to the separator "::".
- trim(str) This function removes white spaces at the beginning and at the end of the given string. The only argument of this function is a value of type string representing the value to trim. This function returns a value of type string. Example s = trim(" abcd "); will return "abcd".
- length(str) Returns the length of a string. The only argument of this function is a value of type string. This function returns a value of type int. Example: len = length("abcd"); will return 4.
- substring(str,from,to); Returns a new string that is a substring of this string.There are two versions of this function: The first one takes two arguments : the string, and the start index of the substring. The second one takes 3 arguments : the string, the start index and the length of the substring. Example s = substring("abcd",1,2); will return "bc".
- startsWith(str,prefix) Returns true if the first argument starts with the specified prefix given as a second argument.If the second argument is the empty string, returns true. Example startsWith("abcd","ab"); will return true.
- endsWith(str,postfix) Returns true if the first argument ends with the specified suffix given as a second argument.Example:endsWith("abcd","cd"); will return true.
- lowerCase(str) Returns a new string converted to lower case. The only argument of this function is a value of type string representing the value to convert to lowercase. This function returns a value of type string. Example: lowerCase("ABCD"); will return "abcd".

- upperCase(str); Returns a new string converted to upper case. The only argument of this function is a value oftype string representing the value to convert to uppercase. This function returns a value of type string. Example:upperCase("abcd"); will return "ABCD".
- replace(str,target,replacement); Replaces each substring of a string that matches the literal target string with the specified literal replacement string. The replacement proceeds from the beginning of the string to the end, for example, replacing "aa" with "b" in the string "aaaaa" will result in "bba" rather than "abb".This function takes 3 arguments :subject string, searched sequence and replace sequence. Example:
- s = replace("abcd","bc","x"); will return "axc".

Map library

- m = map(); m = {}; Creates an empty map.
- map(v1,v2,...) This function returns a value of type map containing all the values given as arguments. The integer keys associated by this function to the elements start from zero and are consecutive. This function takes a variable number of arguments. This function returns a value of type map containing all the values given as arguments. Example m = map(9, "abc"); m = {1, "abc"}; Creates a map containing values 9, "abc" at keys 0, 1 respectively.
- count(m) Counts the number of values in the map. The only argument of this function is a value of type map. It returns a value of type int representing the size of the map.
- values(m) Returns the values of the map as a map. The created map uses keys 0 to count(m)-1. Value associated to integer keys in the orginal map will appear first, in the increasing order of their keys. Values assigned to string keys will appear second, in no predefined order.
- keys(m) Returns the keys of the map as a map. The created map uses keys 0 to count(m)-1. Integer keys will appear first in increasing order. String keys will appear second, in no predefined order.
- add(m,v) This function adds the value at the end of the map (just after the current largest integer key).

マップが空の場合、値はキー0 に関連付けられます。この関数の一番目のアーギュメントは、追加された値を持つマップで表現しているマップ型の値です。この関数の二番目のアーギュメントは追加される値です。この関数は nill を返します。例：add(m, 123);最大整数キー　プラス 1 に等しいキーのマップに 123 を追加します。

Modeling library

Modeling functions：モデリング関数

全ての利用可能なモデリング関数は下記のように定義されています。言語記号に関連した演算子は既にセクション 4 で紹介しました。下記の表に記載されている通り、全ての中置演算子は複素数関数に相当します。例えば、a == b は eq(a,b)に等しいことを表します。引数の個数が″n>0″である関数は、可変個引数の関数です。つまり、任意の数の引数を与えられる関数です。非常に便利なこの関数型の呼び出しに関する詳細は、セクション 6.4 を参照ください。下記の関数は、引数として、LS 表現、整数あるいは浮動小数を取ります。全ての引数が数値である場合、(整数または浮動小数)関数は数値を戻します。数値でない場合、関数は LS 表現を戻します。

	Function	Description	Type	Arity	Symb
Decisional	bool	Boolean decision: decision variable with domain {0,1}.	bool	0	
Arithmetic	sum	Sum of all operands.	double*	n > 0	+
	prod	Product of all operands.	double*	n > 0	*
	min	Minimum of all operands.	double*	n > 0	
	max	Maximum of all operands.	double*	n > 0	
	div	Division of the first operand by the second one.	double*	2	/
	mod	Modulo: mod(*a*,*b*) = *r* such that *a* = *q***b* + *r* with *q*, *r* integers and |*r*| < *b*.	int	2	%
	abs	Asolute value: abs(*e*) = *e* if *e* >= 0, -*e* otherwise.	double*	1	
	dist	Distance: dist(*a*,*b*) = abs(*a*-*b*).	double*	2	
	sqrt	Square root.	double	1	
	cos	Cosine.	double	1	
	sin	Sine.	double	1	
	tan	Tangent.	double	1	
	log	Natural logarithm.	double	1	
	exp	Exponential function.	double	1	
	pow	Power: pow(*a*,*b*) is equal to the value of *a* raised to the power of *b*.	double	2	
	ceil	Ceil: round to the smallest following integer.	int	1	
	floor	Floor: round to the largest previous integer.	int	1	
	round	Round to the nearest integer: round(*x*) = floor(*x*+0.5).	int	1	
Logical	not	Not: not(*e*) = 1 - *e*.	bool	1	!
	and	And: equal to 1 if all operands are 1, and 0 otherwise.	bool	n > 0	&&
	or	Or: equal to 0 if all operands are 0, and 1 otherwise.	bool	n > 0	||
Relational	neq	Not equal to: neq(*a*,*b*) = 1 if a != b, and 0 otherwise.	bool	2	!=
	geq	Greater than or equal to: geq(*a*,*b*) = 1 if a >= b, and 0 otherwise.	bool	2	>=
	leq	Lower than or equal to: leq(*a*,*b*) = 1 if a <= b, and 0 otherwise.	bool	2	<=
	gt	Strictly greater than : gt(*a*,*b*) = 1 if a > b, and 0 otherwise.	bool	2	>
	lt	Strictly lower than: lt(*a*,*b*) = 1 if a < b, and 0 otherwise.	bool	2	<
Conditional	iif	Ternary conditional operator: iif(*a*,*b*,*c*) = *b* if a is equal to 1, and *c* otherwise.	int*	3	? :
	at	Access to an array: *T*[*i*] returns the *i*th value in array *T*.	int	2	[ ]

Appendix

BNF Syntax

TODO

Main error messages: 主なエラーメッセージ

コマンドラインで不可欠な引数は、lsp ファイル名です。lsp ファイルが利用出来ない場合、エラーになります。また、コマンド・ラインの他の全ての引数(パラメータ等)はフォーマット identifier=value.を持たなければなりません。

<f> doesn't exist or is not accessible. // LSP file

Invalid argument format for <arg>. Expected format : identifier=value.

LSP 言語は型を強く意識した言語であり、関数のパラメータには正しい型が必要です。

- Function <f> cannot handle argument of type <t>. Argument of type <t2> is expected.
- Function <f> takes <x> argument(s) but <y> were provided.
- Function <f> : <T> expression expected for argument <i>.

同様に、型チェックで不適切な型の場合には、エラーメッセージを出力します。

- Cannot apply <opName> operator on type <T>.
- Cannot apply <opName> operator between types <T1> and <T2>
- Cannot cast <T1> to <T2>.
- Cannot apply ternary operator '?:' on given operators : incorrect argument type.
- Cannot cast 'nil' to <T>. A variable or a map element may not be assigned.

いくつかの関数は引数を持ちます。引数が合わない場合、エラーを出力します。

- Function <f> takes at least <x> argument(s) but <y> were provided.
- Function <f> takes at most <x> argument(s) but <y> were provided.

関数を使う時、関数が未定義であれば、エラーメッセージを出力します。また既存の関数と同じ名前の関数を定義することもエラーとなります。

- Function <f> already defined.
- Function <f> undefined.
- Variable <name> already defined.

Input/output 関数は指定されたファイルの入出力チェックを行います。

- File <f> cannot be opened.
- Cannot read from file <f>.
- Cannot write to file <f>.

数字または文字列の入力時に、プログラムとデータが一致しない(データ数、タイプ数等が一致しない)場合、およびファイルの最後まで読んだ場合にはエラーを出力します。

- Cannot convert the current token to int.
- Cannot convert the current token to double.
- End of file: no more line to read from file <f>.
- End of file reached.

文字列の操作では、文字列が空でないこと及びインデックスが許容範囲内であることが必要です。

- The given index for substring is out of range. Min value: 0, Max value: <len>.
- Number of characters for substring must be greater than 0.
- Search string is empty.

マップの制限は 2 つあります。

- キーは整数または文字列であること
- イタレーション中(連続して探索計算している間)はマップを変更してはならない
- 'nil' provided as key for a map. The key variable may not be assigned.
- Only types 'string' and 'int' are allowed for keys in maps.
- Cannot iterate on a modified map.

LSP モデルに対して、パラメータで数値を指定する場合には、許容範囲の数値でなければなりません。

- The objective bound must be an integer, a double or a boolean for objective <objIndex>
- The objective bound must be an integer or a boolean for objective <objIndex>
- The number of threads cannot exceed 1024.
- The annealing level size must be an integer between 0 and 9.
- Advanced parameter <key> does not exist.

モデルで式または変数を表現する場合には<-で宣言する必要があります。局所変数を<-使用し宣言することはできません。
モデルには必ず目的関数がなければなりません。また目的関数は数式で表現する必要があります。マップ等は使用できません。
制約式はバイナリ表現でなければなりません。

- Cannot assign localsolver expressions to local variables.
- At least one objective is required in the model.
- Only boolean expressions can be constrained.
- Only expressions with a value can be added in the objectives list.

setValue 関数は意思決定変数(bool 変数)にのみ初期値を与えることができます。

- Only decisions can be set.
- The only allowed values are 0 or 1.

x <- a[y]の数式表現では、マップとしてゼロから連続した整数キーが必要です。またバリューとして、キーの数分だけ、数値データまたは LS 表現が必要です。

- All keys must be integers. Type found: <T>
- Values must be integers, booleans or expressions. Type found: <T>
- The first key must be 0. Key found: <key>
- Keys are not in a continuous range. Next key expected <key1>. Key found: <key2>.

探索の間に、変数値が制約を満足しない場合があります。
例えば、実行可能状態の時、実行可能解を探索中に発生するケースがあります。ゼロ割や配列オーバーフローが起きた時であり、割算の分母がゼロまたはインデックスが範囲外になったことを意味します。z <- x/y のような場合には、z<-x/max(1,y)と表現することが望ましいです。

- Division by zero
- Index out of bounds for 'at' operator (index: <indexId>, array size: <n>