Parameter Space Language:

Symbols and strings that appear as a part of the final parameter specification are double-quoted and are in bold-face. Curly parentheses (when not quoted) are used for better readability of the language.

Symbols '?', '*' and '+' have the following meaning:

?: Symbol (or group of symbols in curly parenthesis) is optional (can appear 0 or 1 time).

*: Symbol (or group of symbols in curly parenthesis) can appear zero or multiple times.

+: Symbol (or group of symbols in curly parenthesis) have to appear at least once and can appear multiple times.

1. <*parameter space*> ::= "**parameter space**" <parameter space name>
   "**{**"
   {<code region declaration>}*
   {<region set declaration>}*
   {<parameter declaration>}+
   {<constraint declaration>}*
   {<constraint specification>}*
   {<ordering info>}*
   "**}**"
2. <*parameter space name*> ::= <identifier>

Code Region and Region Set Declaration Part:

1. <*code region declaration*> ::= "**code_region**" <region name> "**;**"
2. <*region name*> ::= <identifier>
3. <*region set declaration*> ::= "**region_set**" "**{**" <region set name> <region name> "**}**" "**;**"
   | "**region_set**" "**{**" <region set name> <region name> <region name list> "**}**" "**;**"
4. <*region set name*> ::= <identifier>
5. <*region name list*> ::= <region name> { "**,**" <region name> }*

Parameter Declaration Part:

1. <*parameter declaration*> ::= "**parameter**" <parameter name> <type> "**{**"
   <domain restrictions>
   {<default>}?
   {<code region>}*
   "**}**"
2. <*parameter name*> ::= <identifier>
3. <*type*> ::= "**int**"
   | "**float**"
   | "**enum**"
   | "**bool**"
4. <*domain restrictions*> ::= <int restrictions>
   | <float restrictions>
   | <enumeration>
5. <*int restrictions*> ::= "**range [**" <min num>"**:**"<step num>"**:**"<max num> "**];**"
   | "**range [**" <min num> "**:**" <max num> "**];**"
   | "**range [**" <expression> "**];**"
6. <*float restrictions*> ::= "**range [**" <min num>"**:**"<step num>"**:**"<max num> "**];**"
   | "**range [**" <expression> "**];**"

Integer and float restriction are now in matlab format.

7. <*enumeration*> ::= "**enumeration [**" <element> "**];**"
   | "**enumeration [** " <comma separated elements> "**];**"
8. <*element*> ::= <identifier>
9. <*comma separated elements*> ::= <element> {"**,**" <element>}*
10. <*min num*> ::= <integer>
    | <float>

11. <*step num*> ::= <integer>
| <float>
| "**^**" <expression>

For example, [2:^2:256] specification will only look at powers of 2.

12. <*max num*> ::= <integer>
| <float>
13. <*default*> ::= "**default** " <integer> "**;**"
| "**default** " <float> "**;**"
| "**default** " <element> "**;**"
| "**default** " <bool default> "**;**"
14. <*code region*> ::= "**region** " <region name> "**;**"
| "**region** " <region set name> "**;**"
15. <*bool default*> ::= "**T**"
| "**F**"
16. <*integer*> ::= <digit>
| <integer> <digit>
17. <*float*> ::= <integer>"**.**"<integer>
18. <*digit*> ::= [0-9]
19. <*identifier*> ::= <letter>
| <identifier> <letter>
| <identifier> <digit>
20. <*letter*> ::= [a-z]
| [A-Z]
| "**_**"

Constraint Declaration Part:

1. <*constraint declaration*> ::= "**constraint**" <constraint name> "**{**"
<expression> "**;**"
"**}**"
2. <*constraint name*> ::= <identifier>
3. <*expression*> ::= <simple expression>
| <simple expression> <relational op> <simple expression>
| <simple expression> <logical op> <simple expression>
| <if expression>
4. <*simple expression*> ::= <term>
| <term> <additive op> <term>
5. <*term*> ::= <factor>
| <factor> <multiplicative op> <factor>
6. <*factor*> ::= <enum ref expression>
| <region set parameter ref>
| <region parameter ref>
| <parameter name>
| <integer>
| <float>
| <identifier>
| "**(**" <expression> "**)**"
7. <*additive op*> ::= "**+**"
| "**-**"
8. <*multiplicative op*> ::= "*****"
| "**/**"
| "**%**"
9. <*relational op*> ::= "**==**"
| "**<**"
| "**>**"
| "**≤**"
| "**≥**"
| "**!=**"
10. <*logical op*> ::= "**AND**"

| "**OR**"
| "**IMPLIES**"

11. <*enum ref part*> ::= <parameter name>"**.value**"
12. <*region set parameter ref*> ::= <region set name>"**.**"<paramter name>
13. <*region parameter ref*> ::= <region name>"**.**"<paramter name>
14. <*if expression*> ::= <expression> "**?**" <then part>
    | <expression> "**?**" <then part> "**:**" <else part>
15. <*then part*> ::= <identifier> "**=**" <expression>
16. <*else part*> :: = <identifier> "=" <expression>

Conjunction or Disjunction Declaration Part:

1. <constraint specification> ::= "**specification**" "**{**"
   <specification> "**;**"
   "**}**"
2. <specification> ::= <constraint name>
   | <constraint name> <logical op> <constraint name>
   | "**(**" <specification> "**)**" <logical op> "**(**" <specification> "**)**"

Ordering Declaration Part:

1. <ordering declaration> ::= "**ordering**" "**{**"
   <parameter list> "**;**"
   "**}**"
2. <parameter list> ::= <parameter name>
   | <parameter name> {"**,**" <parameter name>}*

We now provide a few examples. We have used '#' for comments.

Example #1:

```
parameter space simple_example
  {

        # In this example, there are no definitions for code regions or sets. Code region and
        # set definitions are optional in the language. Note that at least one parameter
        # declaration must be provided. Parameters declared in this example are integers and
        # are given in [min:step:max] format (matlab style).

        parameter x int {
           range [1:1:3];
           default 3;
        }
        parameter y int {
           range [1:1:3];
           default 2;
        }
        parameter z int {
           range [1:1:3];
           default 1;
        }

        # And then the constraints.
        constraint c1 {
            x≥z;
        }

        constraint c2 {
            y>z;
        }
```

```
        # Constraint specification.
         specification  {
             c1 AND c2;
         }

        # Ordering information is optional.
 }

One possible Solution: x=2,y=3,z=2
```

Example #2:

```
parameter space tiling {
        code_region loopI;
        code_region loopJ;
        region_set loop [loopI, loopJ];

        # declare tile_size parameter and associate this to a region set. Default tile_size of 32.
        parameter tile_size int {
             range [2:2:256]
             default 32;
             region loop;
        }

        # Arbitrary constraint (loosely based on Rivera paper on tiling for 3D stencil loops)
        # for the purposes of this example.
        constraint c1 {
             (loopI.tile_size * loopJ.tile_size * 3 * 4) ≤ 2048;
        }

        # lets assume we like rectangular tiles better.
        constraint c2 {
             loopI.tile_size > loopJ.tile_size;
        }

        constraint c3 {
             loopJ.tile_size > loopI.tile_size;
        }

        specification {
            (c1 AND c2) OR (c1 AND c3);
        }

}
```

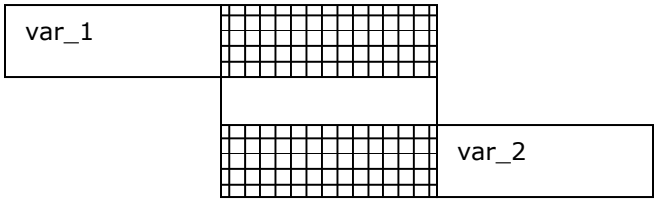
Example #3:

```
# Application input parameters for PSTSWM benchmark:

parameter space pstswm {
        parameter p int {
             range [1:1:16];
             default 4;
        }
        parameter q int {
             range [1:1:16];
             default 4;
        }
```

```
      # FTopt determines what FFT algorithm to use.
       parameter FTopt enum {
            enumeration [distributed, single_transpose, double_transpose];
            default distributed;
       }

      # LTopt detemines which LT algorithm to use.
       parameter LTopt enum {
            enumeration {distributed, transpose_based};
            default distributed;
       }

       constraint pq {
            (p*q) == 16;
       }

       # When FTopt is 'double_transpose', LTopt has to be 'transpose_based'
       constraint ftLT {
            (FTopt.value=double_transpose) IMPLIES (LTopt.value=distributed);
       }

       specification {
            pq AND ftLT;
       }
}
```

Some observations:

1. Should we introduce constant symbols to the language and have a notion of "include" in our grammar? Constants can be declared and subsequently reused later for L1Size, L2Size, CacheLineSize (and other platform specific attributes).
2. Consider two variables (type enum) with the following dependency relation:
   If var_1 assumes a value within the hashed region, var_2 has to take a value in the hashed region as well and if var_1 assumes a value in the plain region, var_2 has to take a value in the plain region as well.
   To express the constraint, we need to introduce operators like "subscript" or "range" on enumerated types.



3. We need more intuitive ways to express constraints on enumerated types.

Questions for the people at USC:
Is this language sufficient for expressing the types of constraints you deal with in your work? There is an example on using Domain knowledge to guide the search process in the document that introduced search and code transformation work at ISI. Could you please use this language to put together a parameter specification for that example?

Question for the broader PERI-search group:
Are there interesting constrained optimization problems that can be expressed intuitively using this language?