Programming Languages and Translators - Proposal

Daniel Perlmutter (dap2163), Joaquin Ruales (jar 2262), Yasser Aboudkhil (ya2282)

P.L.A.T.O. - Programing Language for Abstract Transformation Operators

PLATO is designed for succinctly expressing mathematical programs. It is inspired by Matlab, Python, Prolog, Java and Ocaml

PLATO is

- Strongly typed
- Imperative with Higher Order functions
- Not Object Oriented

primitive types -

- BOOLEAN is True or False
- NUMBER is an exact number or NaN, +INF, -INF
  - NUMBER is OVER one of the following (which must contain a finite number of elements or be built in) - defaults to complex numbers
  - GROUP: requires defining add(x,y) and allows + and -
  - RING: require mult(x,y) and allows *
  - FIELD (by default uses the complex field): allows /
- INTEGER (like NUMBER but allows %)
- SYMBOL (String) - formal symbol, for use with user defined group/rings/fields

derived types -

- SET - Cannot contain duplicates, all elements are of the same type
- MATRIX - All elements are NUMBER
  - VECTOR is syntactic sugar for 1d matrix
  - Vectorization - Operating MATRIX as if it were a NUMBER will implicitly perform elementwise computation
  - Slice Syntax - M[v1,v2. ..vn] returns a rectangular slice of an n dimensional matrix
  - Logical indexing - M[BM] where BM is a matrix of 0/1s the same size as M will return a vector of all values in M specified by the positions where BM = 1
- TUPLE - Can contain mixed types

keywords-

AND (boolean and, set intersection), OR (boolean or, set union), NOT (boolean not, set complement when used with IN), AS (typecasting), IF, ELSE (conditionals), SOME, ALL, SATISFIES (quantifier syntax), IN (domains), TO, BY (iteration), OVER (fields), PRINT (print)

operators-

+ (plus, union), - (minus, set difference), * (times, cartesian product), / (divided by), % (modulo), ** (matrix times), := (assignment), ^ (exponentiation)

conditionals-

< (less than), > (greater than), <= (less than or equal to) , >= (greater than or equal to), = (equal to), != (not equal to)

block constructions-

{ (open block), } (close block), ; (end statement), /* (open comment), */ (close comment)

The following are examples of PLATO code. Each example illustrates 1 or more features of PLATO. Examples marked as CORE are key to the language, those as STRETCH will be included if we have time.

examples:

CORE - Manipulate Primitives and Printing

```
main() {
  NUMBER x := 1;
  NUMBER y := 2;
  BOOLEAN z := x >= y;
  PRINT z /* prints false */
}
```

CORE - Manipulate vectors and matrices

```
main() {
  VECTOR v1 := [2 TO 9 BY 3]; /* [2 5] */
  VECTOR v2 := [2 TO 6];      /* [2 3 4 5 6] */
  VECTOR v3 = v2[2 TO 3];     /* [3 4] */
  MATRIX m1 := [v1; v3]       /* [2 5
                                    3 4] */
  MATRIX m2 := m1 > 3         /* [0 1
                                    0 1] */
  VECTOR v4 = m1[m2]          /* [5 4] */
  MATRIX m3 := m1 ** m1;      /* [19 30
                                    18 31] */
}
```

CORE - Manipulating sets and type casting -

```
main() {
  SET<NUMBER> s1 := {1, 4};
  SET<NUMBER> s2 := [1 TO 3] AS SET<NUMBER>; /* {1, 2, 3} */
  SET<NUMBER> s3 := s1 + s2 /* {1,2,3,4} */
  SET<NUMBER> s4 := s1 - s2; /* {4} */
  SET<SET<NUMBER>> s5 = s1 * s2; /* {{1,1}, {1,2}, {1,3}, {4,1}, {4,2}, {4,3}}
}
```

CORE - Manipulating tuples

```
main() {
  TUPLE t1 = (1,2,3);
  TUPLE t2 = t1 + [4,5]; /* (1, 2, 3, [4 5]) */
```

```
    TUPLE t3 = t2 - 2; /* (1, 3, [4 5]) */
    INTEGER x = t3(2) AS INTEGER; /* 3 */
    TUPLE t4 = t3([2 TO 3]) /* (3,  [4 5]) */
  }

CORE - User defined finite domains
  main() {
    NUMBER x OVER z2 := 1;
    NUMBER y OVER z2 := 1;
    PRINT x + y /* prints 0 */
  }
  FIELD z2 {
    SET<INTEGER> elements := {0, 1};
    INTEGER add(INTEGER n1, INTEGER n2) /*
      (n1 + n2) % 2;
     }
     INTEGER multiply(INTEGER n1, INTEGER n2) {
       (n1 * n2) % 2;
     }
   }

CORE - Quantifier syntax for sets and implicit vectorization
  main() {
    PRINT prime([1 TO 10]) /* prints [2, 3, 5, 7];
  }
  INTEGER prime(INTEGER n) {
    NOT (SOME d IN ([2 TO n - 1]) SATISFIES (n % d = 0);
  }

STRETCH - Constructors for fields
  main() {
    NUMBER x OVER z(2) := 1;
    NUMBER y OVER z(2) := 1;
    PRINT x + y /* prints 0 */
  }
  FIELD z(n) {
    SET<INTEGER> elements := [0 TO n-1] AS SET<INTEGER>;
    INTEGER add(INTEGER n1, INTEGER n2) /*
      (n1 + n2) % n;
     }
     INTEGER multiply(INTEGER n1, INTEGER n2) {
       (n1 * n2) % n;
     }
```

```
  }

STRETCH - Exact computation on fractions
 main() {
   NUMBER x := 1 / 2;
   NUMBER y := 1 / 3;
   NUMBER z := x + y;
   PRINT z := 5 / 6 /* prints true */
 }

STRETCH - Manipulate symbols
  main() {
    SYMBOL x := “s”;
    PRINT x; /* prints s */
    SYMBOL y OVER c2 = “b”;
    PRINT y + y; /* prints a */
  }
  GROUP<SYMBOL> c2 {
    SET<INTEGER> elements := {“a”, “b”};
    SYMBOL add(SYMBOL s1, SYMBOL s2) /*
    if (s1 = s2) {
       “a”;
     } else {
       “b”;
     }
   }
```

Here are some examples of classical algorithms written in PLATO -

```
/* Euclid’s gcd */
INTEGER gcd(INTEGER a, INTEGER b) {
 if (a = 0) {
   a;
 } else {
   gcd(b, a % b);
 }
}

/* Sieve of Eratosthenes */
SET<INTEGER> primes(INTEGER n) {
  sieve([2 TO n], n) AS SET<INTEGER>;
}
```

```
VECTOR<INTEGER> sieve(VECTOR<INTEGER> v, INTEGER n) {
  d := v[1];
  if (d * d <= n) {
    [d, sieve(v[NOT (v % d = 0)], n)];
  } else {
    v;
  }
```