Introduction to DML

R. Sekar

What is DML

- Discrete Math Language: a programming language custom-designed for teaching Discrete Math.
  - Developed at Stony Brook by the instructor and some of the TAs

What is DML

- Discrete Math Language: a programming language custom-designed for teaching Discrete Math.
  - Developed at Stony Brook by the instructor and some of the TAs
- Goals:
  - Provide a simple syntax that closely matches that of math.
  - Help students understand math concepts better by *executing* them.
  - Provide a gateway to programming:
    - Refine your specification to go from “what” to “how”
    - Maintain consistency in syntax with established programming languages

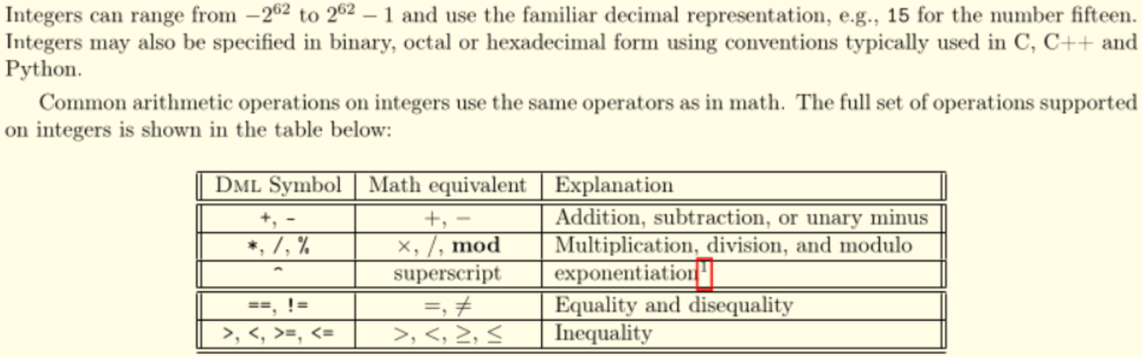
Key differences from Math

- In math, data types are rarely mentioned.
  - In DML, all values have a type, and the DML interpreter shows it.
- In math, we often use symbols that aren't on your keyboard.
  - In DML, we map these symbols to those on the keyboard
- In math, we often work with infinite sets
  - DML can only represent finite sets
- In math, we rely on proofs to establish the truth or falsehood of propostions
  - DML uses computation and search, which works only on some propositions

Elements of DML

- Values and Types
  - Base types: integers, reals, booleans, strings
  - Aggregate types: Sets, lists, tuples, dictionaries
- Expressions
- Variables
- Functions

Integers



Integers can range from -2^{62} to $2^{62}-1$ and use the familiar decimal representation, e.g., `15` for the number fifteen. Integers may also be specified in binary, octal or hexadecimal form using conventions typically used in C, C++ and Python.

Common arithmetic operations on integers use the same operators as in math. The full set of operations supported on integers is shown in the table below:

DML Symbol	Math equivalent	Explanation
`+`, `-`	$+, -$	Addition, subtraction, or unary minus
`*`, `/`, `%`	$\times, /, \mathbf{mod}$	Multiplication, division, and modulo
`^`	superscript	exponentiation[1]
`==`, `!=`	$=, \neq$	Equality and disequality
`>`, `<`, `>=`, `<=`	$>, <, \geq, \leq$	Inequality

Reals

- Can be written in:
  - fixed point, e.g., `-1.53`, or
  - scientific format, e.g., `1.53e-27` for 1.53×10^{-27}.
- Support the same set of operations as integers, except for **mod**
- Conversion to integers
  - `round` rounds its argument, e.g., `round(1.49)` is 1, while `round(1.5)` is 2.
  - `ceil` computes the closest integer greater than or equal to the argument, e.g., `ceil(1.49)` is 2.
  - `floor` computes the closest integer less than or equal to the argument, e.g., `floor(1.99)` is 1.

Booleans

- Distinct from integers, and can't be intermixed
- Arise mainly from comparisons
  - but can also define variables and constants of boolean types
- Boolean operators

DML Symbol	Math equivalent	Explanation
`<->`	$\leftrightarrow$	Logical equivalence ("if and only if")
`->`	$\rightarrow$	Logical implication
`||`	$\vee$	Disjunction ("or" operator)
`&&`	$\wedge$	Conjunction ("and" operator)
`!`	$\neg$	Logical negation

Strings

String literals can be enclosed in single or double quotes, e.g., `"This is a string"` and `'This is another string'`. Common escape sequences for special characters are supported in a manner compatible with languages such as C and Python. For example, `'a\nb'` is a string that includes a newline character, denoted `\n`. DML will use these escape sequences when it shows the values of string variables and constants, but then such a string is provided as input to the `print` function, it is printed as a newline. Here is a DML session illustrating this.

```
dml> x="a\nb"
x:string = "a\nb"
dml> print(x)
a
b
dml>
```

Strings can be concatenated using the operator `++`, and can be compared with each other using any of the comparison operators discussed above.

```
dml> "Hello" ++ "World"
"HelloWorld"
dml>
```

Sets

- Set construction
  - Sets of consecutive integers: `7..100`
  - Sets with enumerated elements:
    - `{1, 7, 100, 33}`
    - `{"Alex", "Dana", "John", "Jennifer"}`
- DML directly supports most set operations

DML Symbol	Math equivalent	Explanation
`in`	$\in$	Membership check
`union`	$\cup$	Set union
`inter`	$\cap$	Set intersection
`subseteq`	$\subseteq$	Subset operators
`-`	$-$	Set difference
`*`	$\times$	Cartesian product
`pow`	$\wp$	Power Set

Set Builder Notation in Math and DML

- Set of squares of odd numbers ≤ 100
  - Math: $E ::= \{n^2 \mid n \in \mathbb{N} \wedge (n < 100) \wedge (n \textbf{ mod } 2 = 1)\}$
  - DML: `E = {n^2 for n in 0..99 if (n % 2 == 1)}`

Set Builder Notation in Math and DML

- Set of squares of odd numbers ≤ 100
  - Math: $E ::= \{n^2 \mid n \in \mathbb{N} \wedge (n < 100) \wedge (n \textbf{ mod } 2 = 1)\}$
  - DML: `E = {n^2 for n in 0..99 if (n % 2 == 1)}`
- Set of numbers that satisfy a given condition
  - Math: $E ::= \{n \in \mathbb{N} \mid (n \leq 100) \wedge (n^2 - 41n - 40 > 0)\}$
  - DML: `E = {n for n in 0..100 if (n^2 - 41*n - 40 > 0)}`

Set Builder Notation in Math and DML

- Set of squares of odd numbers ≤ 100
  - Math: $E ::= \{n^2 \mid n \in \mathbb{N} \wedge (n < 100) \wedge (n \bmod 2 = 1)\}$
  - DML: `E = {n^2 for n in 0..99 if (n % 2 == 1)}`
- Set of numbers that satisfy a given condition
  - Math: $E ::= \{n \in \mathbb{N} \mid (n \leq 100) \wedge (n^2 - 41n - 40 > 0)\}$
  - DML: `E = {n for n in 0..100 if (n^2 - 41*n - 40 > 0)}`
- Set of Pythagorean triples ≤ 10
  - Math: $E ::= \{(x, y, z) \in \mathbb{N} \times \mathbb{N} \times \mathbb{N} \mid x^2 + y^2 = z^2\}$
  - DML:
    ```
    {(x,y,z) for x in 1..10 for y in 1..10 for z in 1..10
             if x^2 + y^2 == z^2}
    ```

DML Lists

- Lists are enclosed in square brackets, and are *ordered*
  - `[1,2]` is different from `[2,1]`
- Use built-in function `range` to construct integer lists:
  - `range(7, 11) = [7, 8, 9, 10]`
    - Unlike sets, the largest element is one less than the higher limit
  - An optional third parameter specifies the step:
    - `range(7, 17, 5) = [7, 12]`
    - `range(7, 18, 5) = [7, 12, 17]`

DML Lists

- Lists are enclosed in square brackets, and are *ordered*
  - `[1,2]` is different from `[2,1]`
- Use built-in function `range` to construct integer lists:
  - `range(7, 11) = [7, 8, 9, 10]`
    - Unlike sets, the largest element is one less than the higher limit
  - An optional third parameter specifies the step:
    - `range(7, 17, 5) = [7, 12]`
    - `range(7, 18, 5) = [7, 12, 17]`
- List builder notation is similar to set builder notation:

```
[ x for x in {1,2,3,2,1} ]  =  [1,2,3]
```

DML Lists

- Lists are enclosed in square brackets, and are *ordered*
  - `[1,2]` is different from `[2,1]`
- Use built-in function `range` to construct integer lists:
  - `range(7, 11) = [7, 8, 9, 10]`
    - Unlike sets, the largest element is one less than the higher limit
  - An optional third parameter specifies the step:
    - `range(7, 17, 5) = [7, 12]`
    - `range(7, 18, 5) = [7, 12, 17]`
- List builder notation is similar to set builder notation:

```
       [ x for x in {1,2,3,2,1} ]  =  [1,2,3]
[ y for y in range(1,50,2) if y % 7 == 0]  =  [7,14,21,28,35,42,49]
```

DML Lists

- Lists are enclosed in square brackets, and are *ordered*
  - `[1,2]` is different from `[2,1]`
- Use built-in function `range` to construct integer lists:
  - `range(7, 11) = [7, 8, 9, 10]`
    - Unlike sets, the largest element is one less than the higher limit
  - An optional third parameter specifies the step:
    - `range(7, 17, 5) = [7, 12]`
    - `range(7, 18, 5) = [7, 12, 17]`
- List builder notation is similar to set builder notation:

```
                 [ x for x in {1,2,3,2,1} ]  =  [1,2,3]
[ y for y in range(1,50,2) if y % 7 == 0]  =  [7,14,21,28,35,42,49]
                     [5*z+1 for z in 1..6]  =  [6, 11, 16, 21, 26, 31]
```

Tuples and Records

- Tuples in math (and DML) correspond to cartesian products of sets:

```
dml> x = (1, 'w', 2.0)
x:(int, string, real) = (1, "w", 2.0)
```

- Ordering matters in cartesian product (and in DML tuples)
  - Elements of a tuple can be accessed using a index

    ```
    dml> x[0]
    1
    ```

  - index starts from 0, can't be a variable or an expression
- *Records* are tuple variants that have field names

```
dml> y = {ival=1, sval='w', rval=2.0}
y:{ival:int, rval:real, sval:string} = {ival=1, rval=2.0, sval="w"}
```

Dictionaries

Dictionaries, also called *associative lists* or *hash tables* in some languages, are a very versatile and efficient data structure frequently used in programs. A dictionary can be viewed as a set of pairs of the form $(key, value)$. Its advantage is that the value associated with a given key can be located efficiently using the index operator. An example dictionary is as follows:

```
x = {'V':'violet', 'I':'indigo', 'B':'blue', 'G':'green',
      'Y':'yellow', 'O':'orange', 'R':'red'}
```

Note that the elements are listed within braces. Within each element, the key precedes the colon operator, while the value follows it. They can both be of arbitrary types. The value associated with a key can be looked up using the index operator, e.g., `x['B']` will return `'blue'`.

While the above notation is convenient for most uses, there can be situations where we want to generate a dictionary from a set of key-value pairs. A predefined function `dict` can do this. For instance, with

```
y = {('V','violet'), ('I','indigo'), ('B','blue'), ('G','green'),
        ('Y','yellow'), ('O','orange'), ('R','red')}
```

`dict(y)` will yield the exact same dictionary as `x`.

Function Definitions and Let Statements

- New functions are introduced using the `fun` keyword:
  - `fun square(x) = x*x`
- `let` statements enable a function definition to be broken up into simpler steps:

  -
    ```
    fun mypoly(x, y) =
        let  t1 = x*x
             t2 = y*y
         in t1 + t2
    ```

Commonly Used DML Functions

- `abs`: Returns absolute value of a number
- `avg`, `sum`: Returns the average or sum of a set or list of numbers
- `concat`: Takes a list of lists or strings, concatenates them.
    - Note: ++ is the binary concatenation operator on lists/strings
- `len`: Returns the length of a set/list/dict/string
- `insert`: Insert new element into a set/list/string
- `min`, `max`: Returns the min or max element in a set or list.
- `print`: Prints all arguments with spaces between them
- `printr`: Raw print, does not implicitly print spaces or newlines

More Commonly Used DML Functions

- `rand`: Returns a pseudorandom number. Changes on each call.
- `remove`: Remove specified set element or dictionary key. Expensive.
- `removeat`: Remove list element at specified index. Expensive.
- `sortaz`, `sortza`: Input is a set or list, returns a list in sorted order.
- See the DML manual for the full list and additional explanation.