Computational Logic
Introduction to Constraint Logic Programming
Comparison of CLP with LP (I)

• Example (Prolog):  

$q(X, Y, Z) :- Z = f(X, Y)$.

| ?- q(3, 4, Z).  
Z = f(3,4) |

| ?- q(X, Y, f(3,4)).  
X = 3, Y = 4 |

| ?- q(X, Y, Z).  
Z = f(X,Y) |

• Example (Prolog):  

$p(X, Y, Z) :- Z \text{ is } X + Y$.

| ?- p(3, 4, Z).  
Z = 7 |

| ?- p(X, 4, 7).  
{ERROR: illegal arithmetic expression} |
Example (CLP(\(Q\))): \(p(X, Y, Z) :- Z =. X + Y\).

?- \(p(3, 4, Z)\).
Z = 7 ?
yes

?- \(p(X, 4, 7)\).
X = 3 ?
yes

?- \(p(X, Y, 7)\).
X =. 7 - Y ?
yes
Prolog: Built-in Arithmetic

- Practicality: interface to the underlying CPU arithmetic capabilities.
- These arithmetic operations are not as general as their logical counterparts.
- Interface: evaluator of arithmetic terms.
- The type of arithmetic terms:
  - a number is an arithmetic term,
  - if \( f \) is an \( n \)-ary arithmetic functor and \( X_1, \ldots, X_n \) are arithmetic terms then \( f(X_1, \ldots, X_n) \) is an arithmetic term.
- Arithmetic functors: +, −, *, / (float quotient), // (integer quotient), mod, and more.

Examples:
- \((3*X+Y)/Z\), correct if when evaluated \( X, Y \) and \( Z \) are arithmetic terms, otherwise it will raise an error.
- \( a+3*X \), incorrect: “\( a \)” is not an arithmetic term.
● Built-in arithmetic predicates:

◊ the usual `<`, `>`, `=<`, `>=`, `:=` (arithmetic equal), `=\=` (arithmetic not equal), ...

◊ Z is X
  X (which must be an arithmetic term) is evaluated and result is unified with Z.

● Examples: let X and Y be bound to 3 and 4, respectively, and Z be a free variable:

◊ Y < X+1, X is Y+1, X =:= Y. fail (the system will backtrack).

◊ Y < a+1, X is Z+1, X =:= f(a). error (abort).
• Features in CLP:
  ◦ Domain of computation (reals, integers, booleans, etc). Have to meet some conditions.
  ◦ Type of constraints allowed for each domain: e.g. arithmetic constraints (+, *, =, ≤, ≥, <, >)
  ◦ Constraint solving algorithms: simplex, gauss, etc.

• LP can be viewed as a constraint logic language over Herbrand terms with a single constraint predicate symbol: “=”
Comparison of CLP with LP (IV)

• Advantages:
  ◦ Helps making programs expressive and flexible.
  ◦ May save much coding.
  ◦ In some cases, more efficient than traditional LP programs due to solvers typically being very efficiently implemented.
  ◦ Also, efficiency due to search space reduction:
    * LP: generate-and-test.
    * CLP: constrain-and-generate.

• Disadvantages:
  ◦ Complexity of solver algorithms (simplex, gauss, etc) can affect performance.

• Solutions:
  ◦ better algorithms
  ◦ compile-time optimizations (program transformation, global analysis, etc)
  ◦ parallelism
Making Programs More Expressive and Flexible: Example

- Fibonacci numbers: (The good old) Prolog version:

```prolog
fib(0, 0).
fib(1, 1).
fib(N, F) :-
    N > 1,
    N1 is N - 1,
    N2 is N - 2,
    fib(N1, F1),
    fib(N2, F2),
    F is F1 + F2.
```

- Can only be used with the first argument instantiated to a number:

```prolog
?- fib(4, F).
F = 3
?- fib(N, 3).
{ERROR: illegal arithmetic expression}
```
• Fibonacci (CLP(\(Q\)) version):
  
  \[
  \text{fib}(N,N) :- N =. 0.
  \]
  
  \[
  \text{fib}(N,N) :- N =. 1.
  \]
  
  \[
  \text{fib}(N,R) :- N >. 1, F1 >=. 0, F2 >=. 0,
  \]
  
  \[
  \text{fib}(N1,F1), \text{fib}(N2,F2),
  \]
  
  \[
  R =. F1+F2.
  \]

• Semantics greatly enhanced! E.g.

```prolog
?- fib(4, F).
F = 3
?- fib(N, 3).
N = 4
?- fib(N, F).
F = 0, N = 0 ;
F = 1, N = 1 ;
F = 1, N = 2 ;
F = 2, N = 3
```
Example of Search Space Reduction

- Prolog (generate–and–test):
  ```prolog
  solution(X, Y, Z) :-
    p(X), p(Y), p(Z),
    test(X, Y, Z).
  ```

  ```prolog
  ```

  ```prolog
  test(X, Y, Z) :- Y is X + 1, Z is Y + 1.
  ```

- Query:
  ```prolog
  ?- solution(X, Y, Z).
  X = 14,
  Y = 15,
  Z = 16 ? ;
  no
  ```

- 458 steps (all solutions: 465 steps).
Example of Search Space Reduction

- **CLP(\(Q\)) (using generate–and–test):**
  
  ```prolog
  solution(X, Y, Z) :-
  p(X), p(Y), p(Z),
  test(X, Y, Z).
  
  
  test(X, Y, Z) :- Y =. X + 1, Z =. Y + 1.
  ```

- **Query:**
  
  ```prolog
  ?- solution(X, Y, Z).
  X = 14,
  Y = 15,
  Z = 16 ? ;
  no
  ```

- 458 steps (all solutions: 465 steps).
Example of Search Space Reduction

- **Move** `test(X, Y, Z)` at the beginning (constrain–and–generate):
  ```prolog
  solution(X, Y, Z) :-
      test(X, Y, Z),
      p(X), p(Y), p(Z).
  ```

- **Prolog**: `test(X, Y, Z) :- Y is X + 1, Z is Y + 1.`
  ```prolog
  | ?- solution(X, Y, Z).{ERROR: illegal arithmetic expression}
  ```

- **CLP(\(Q\))**: `test(X, Y, Z) :- Y .=. X + 1, Z .=. Y + 1.`
  ```prolog
  ?- solution(X, Y, Z).
  X = 14,
  Y = 15,
  Z = 16 ? ;
  no
  ```

- **11 steps (all solutions: 11 steps).**