# Lecture 6 - Prolog Arithmetic

- executing goals: DON'T change answers, only terminations
- Unification ≠ computation ← here, returns X = 1+(2*3)
  - X (is) 1+(2*3) (is/2)
    returned: "X = 7."
    ↓
    relies ground with. exp to evaluation
- "not sufficiently instantiated args" — not everything was a valid num. expression
  - when goal seen, RHS must be computable
  - X < 10, X = 1. → error: args not instantiated
    X = 1, X < 10. → ok :)
    - logical/declarative properties lost :(
- Solution to that problem: reasoning with constraints
  - constraint system can automatically tell if solution set is finite
  - concrete (not set) substitutions need variable enumeration
- Domain relations:
  - "Var in lower..Upper": X ∈ {Upp...low}
    ↑ numbers or int/sup.

- magic squares [A1, A2, ...]
  - don't care about A2

- Zs :: 1..9 ← in domain $1 \leq Z \leq 9$

- CLP(FD) arithmetic program: general structure
  1. Define list Zs of FINITE domain variables for future labeling
  2. set variables' domain
  3. Add constraints on core predicates
  4. Label Zs

- why label ONLY in end?
  - new constraints facilitates PROPAGATION
    - accumulation but also SIMPLIFYING on the fly

- Label, because:
  - Constraints not guaranteed to be satisfiable
    - (e.g. X#>Y, Y#>X)
      - constraints put on both of them but no EXAMPLES given
  - labeling Guarantees Satisfiability

- Labeling strategies:
  - Variable selection
  - Value order
  - Branching strategy (enum = BFS type strategy)

- Improving performance:
  - reduce labeling strategies & number of solutions
    - if sum in magic squares known beforehand, can reduce # of solutions and make program much faster
    - MS: add symmetry breaking constraints
      - don't care about 'mirrored' solutions
      - speed dramatically increases
- NON-logical predicates:
  - Input/output
  - Cut
  - If-then-else
  - Negation as failure

  } all don't have declarative characteristics usually seen in Prolog