Parallel evaluation of logic programs

Intelligent Backtracking in AND/OR process model

KAIST, 전산학과

최광무

2004. 2. 12.

Parallel Evaluation of Logic Programs

- Logic program
- AND/OR proof tree
- AND/OR process model
- OR parallelism and AND parallelism
- Intelligent backtracking(AND parallelism)
- Implementation
- Research Topics

Logic Programs

- Logic programs(Horn clauses)
  Disjunction(OR) of **predicates**(literals)
  with **at most one** un-negated literal
- $\neg p_1 \vee \ldots \vee \neg p_n \vee p$
  $\equiv \neg(p_1 \wedge \ldots \wedge p_n) \vee p$
  $\equiv (p_1 \wedge \ldots \wedge p_n) \Rightarrow p$ **implication**
- p is true, if p_1 and p_2 and … and p_n are true,
  p is either true or false, otherwise.
- Logic says **nothing** when hypothesis is **false**.

Logic program(Prolog)

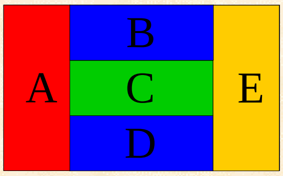
- $h \Leftarrow b_1 \wedge \ldots \wedge b_n$.
  h: **head** literal
  $b_1, \ldots, b_n$: **body** literals
- Four cases (clause)
  1. $\Leftarrow b_1 \wedge \ldots \wedge b_n$? **query** clause
  2. $h \Leftarrow b_1 \wedge \ldots \wedge b_n$. **rule** clause
  3. h . **fact** clause
  4. . tautology(X)

Logic program
A **query** and a set of **rules** and **facts**
Relational database

Syntax of Logic program(Prolog)

- Clause
  - query clause ⇐ body literals ?
  - rule clause head literal ⇐ body literals .
  - fact clause head literal .
- Literal predicate (terms)
- Term
  - constant
  - variable
  - **literal**

Example(map coloring problem)



⇐ color(A, B, C, D, E) ?

color(A, B, C, D, E) ⇐ diff(A, B), diff(A, C), diff(A, D),diff(B, C),
diff(B, E), diff(C, D), diff(C, E), diff(D, E)

diff(X, Y) ⇐diff1(X, Y) | diff1(Y, X)

diff1(red, blue). diff1(red, green) . diff1(red, yellow) .
diff1(blue, green). diff1(blue, yellow) . diff1(green, yellow) .

Top down evaluation of logic programs

- query clause(AND)
    body literals(OR)
        rule clauses(AND)
            body literals(OR) **recursion**
            rule clauses(AND) **recursion**
            …
        fact clauses(AND) **basis**

  Q – L – R – L – …– R – L – F.

  Q – L – $(R – L)^*$ – F.

AND/OR proof tree

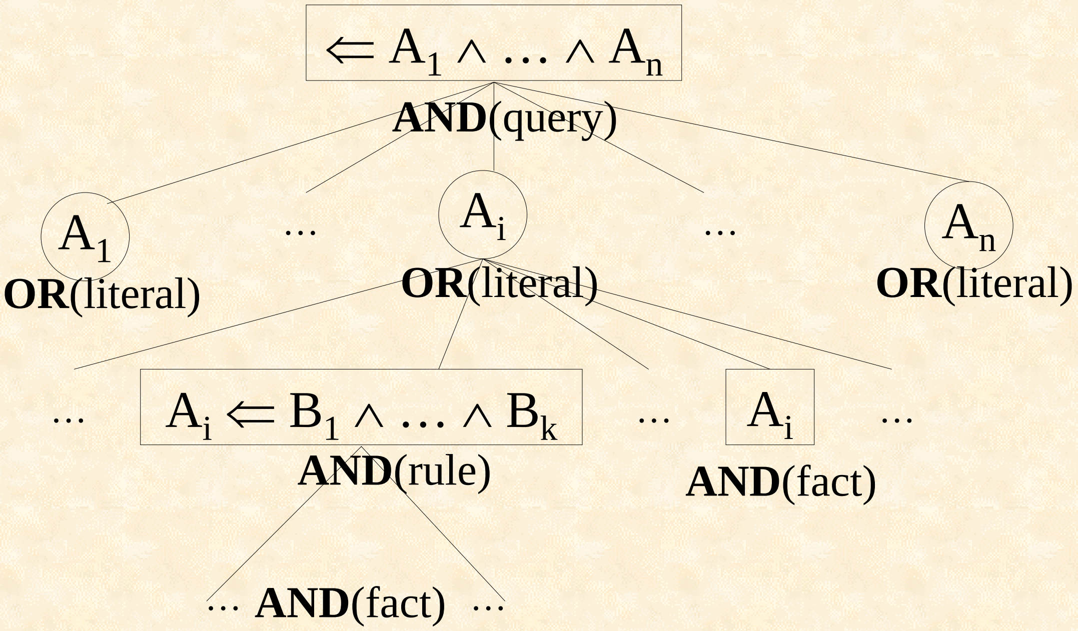
- AND/OR proof tree
    clause and literals
    interlaced graph
- Clause(AND)
    AND logic among children body literals, if any

  Literal(OR)
    OR logic among children clauses with **same head**
- AND node Clause(query, rule, fact)

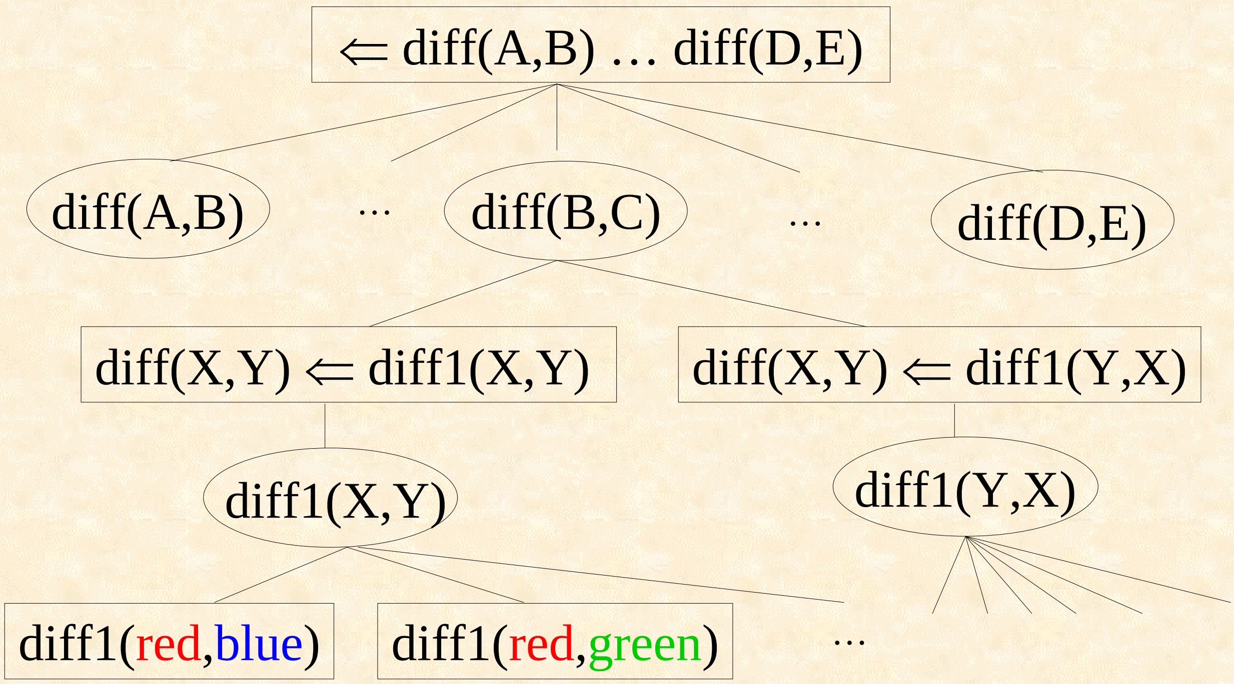
  OR node Literal
- Q – L – $(R – L)^*$ – F.

  A – O – $(A – O)^*$ – A.

AND/OR proof tree(2)



Example(AND/OR proof tree)



Parallel evaluation of programs

- **Parallelizing compiler**
  Find the parallelism in the programs
  Difficult, parallelisms may be lost!
  Array index(integer linear programming)
- **Writing parallel programs**
  concurrent, parallel algorithm
  mathematically well defined problems
    Fourier transformation
- **Inherited parallelism**
  Logic(natural parallelism)

Parallel evaluation of logic programs

- J. C. Conery, 1981 U. C. Irvine
  AND/OR proof tree   AND/OR process tree
    AND node(clause)   AND process
    OR node(literal)   OR process
- AND **process**(clause)
  parallel AND among **children OR processes**
- OR **process**(literal)
  parallel OR among **children AND processes**

Messages between AND/OR processes

1. **create** parent process → child process
   - **Start** evaluation and give me a **solution.**
2. **success** θ child process → parent process
   - **Yes,** θ is the **solution.**
3. **fail** child process → parent process
   - **No,** I do **not** have **solution** any more.
4. **cancel** parent process → child process
   - **Stop** evaluation.
5. **redo** parent process → child process
   - Give me the **next** solution.
6. **reset** parent AND proc. → child OR proc.
   - Give me the **first** solution. (???)

OR parallelism(1)

1. **create** message from parent AND process
   - **Create** all of its children AND processes.
   - Stay in wait mode.
2. **success** θ message from child AND process
   - Store θ in the solution list.
   - Send **redo** to the child AND process.
   - If in wait mode, send **success** θ to the parent,
     mark the **first** solution as θ, and
     change to gathering mode.

OR parallelism(2)

3. **fail** message from child AND process
   - **Cancel** the child AND process.
   - If **no more** children AND processes,
     empty solution list, and in wait mode,
     report **fail** to its parent AND process.
4. **cancel** message from parent AND process
   - **Cancel all** of its children AND processes.

OR parallelism(3)

5. **redo** message from parent AND process
   (in gathering mode only)
   - If there is a **next** solution θ in the solution list,
     send **success** θ to the parent.
   - Else change to wait mode, and
     if no more children, report **fail** to parent.
6. **reset** message from parent AND process
   - Restore solution list.
   - If the first solution is θ,
     send **success** θ to the parent.

AND parallelism

- **Shared variables** among body literals
  - Variable **binding**
- **Join** scheme
  - Generate **all** of the tuples, and **check** binding
  - Fully **parallel** but **inefficient**
- **Forward** scheme
  - **One** literal **generates**(binds) a constant

    the **other literals consumes** the constant
  - **Generator and consumers model**
  - Parallel in **sequence** but **efficient**

Data dependency graph

- Selection of the among **shared** literals

  nondeterministic

  parallel
- Data dependency graph

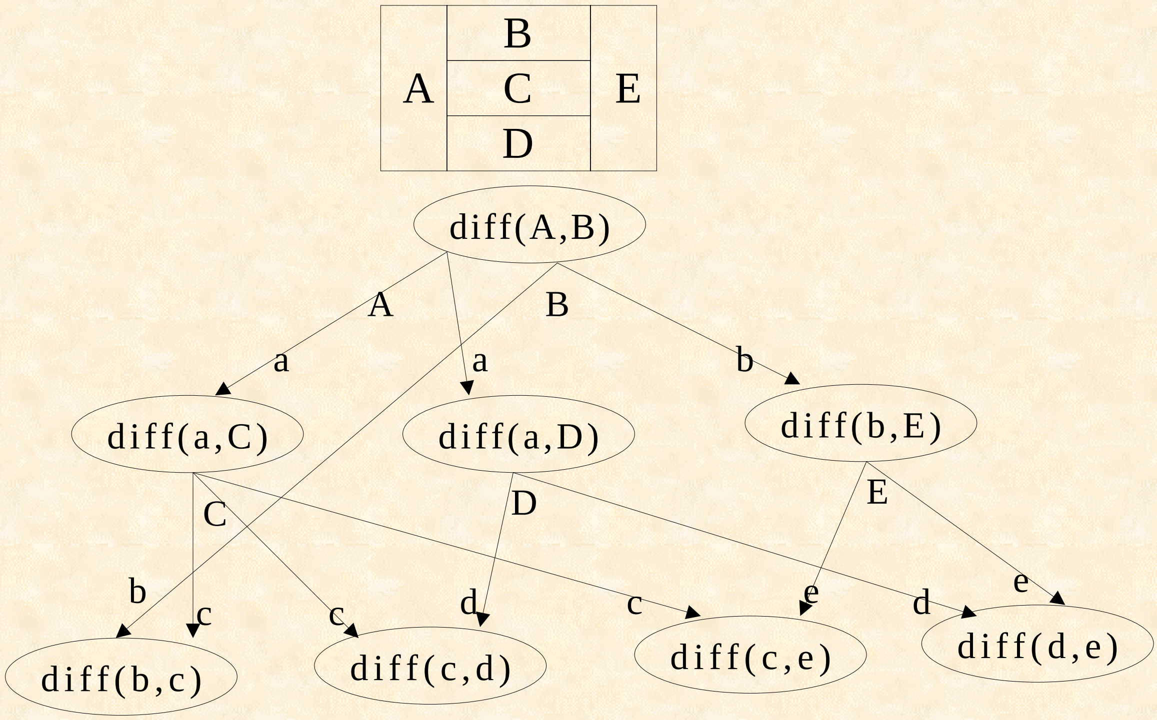
  generator and **consumers** relationships

  for each **shared** variable

  single generator and multiple consumers

  the graph may be **dynamic**

Data dependency graph(example)



Forward and backward execution

- **Forward** execution

  create OR processes parallel **in sequence**

  diff(A, B) – (diff(a, C), diff(a, D), diff(b, E)) – ???

  All of the **children** OR processes **success**,

  report **success**, to its **parent** OR process

- **Backward** execution

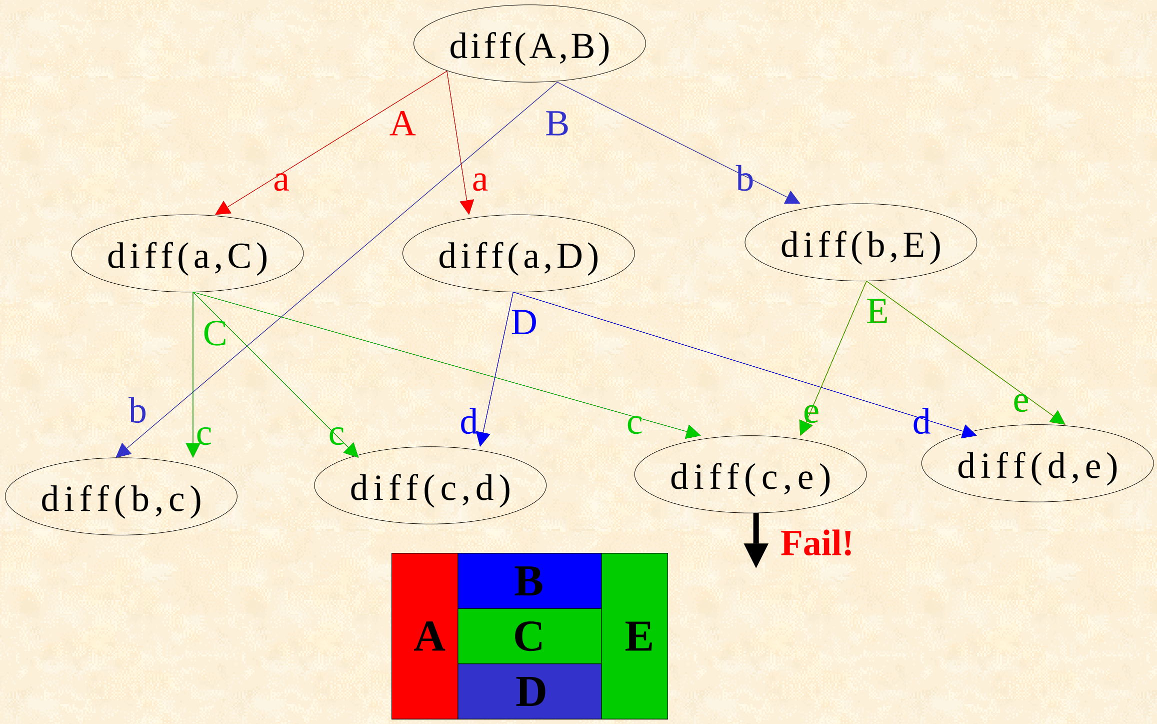
  a child OR process report **fail**

  next binding should be **generated**

  systematically(**generator**)

  exhaustive and **intelligent**

Data dependency graph(forward execution)



Nested loop model

- for A in colors
  for B in colors
  …
  for E in colors
  if diff(A, B) and … and diff(D, E) then …

  for (A, ... ,E) in colors^5

- for (A, B) in colors^2 **where diff(A, B)**
  for C in colors **where diff(a, C)**
  for D in colors **where diff(a, D)**
  for E in colors **where diff(b, E)**
  if diff(b,c) diff(c,d), diff(c,e) diff(d,e) then …

Intelligent backtracking

For (A, B) in color[2] **where** diff(A, B) do
for C in color **where** diff(a, C) do
for D in color **where** diff(a, D) do
for E in color **where** diff(b, E) do
if diff(b, c) and … and diff(d, e) then …

When diff(b, c) fails
naive backtracking(in nested loop model)
diff(b, E) → diff(a, D) → diff(a, C) → diff(A, B)
intelligent backtracking
diff(a, C) → diff(A, B)

Backward execution

- **Nested loop model**
  - **outer** loop variable with next constant(**redo**)
  - variables **inner** loop with the **first** value(**reset**)
- **Intelligent backtracking**
  - a child OR process report **fail**
  - the OR process can not find any solutions
    with the **binding**(**given** by **generator**)
  - **generator** should give **new** binding(**redo**)
  - **generators** in **inner loop**
    restart with the **first** binding(**reset**)

Lineally ordered literal list(LOLL)

- **Linear**(total) order among **generator** literals(variables)

  next(A, B) → next(a, C) → next(a, D) → next(b,E)

  (A, B) → C → D → E
- No dependency relation among C, D, and E.

  But there must be an artificial **linear** order for **systematic** backtracking(**reset**, the **first** solution)
- When next(a, C) is **redone**

  next(b, c), next(c, d), next(c, e) **cancelled**(consumer)

  next(a, D), next(b, E) **reset**(inner loop)

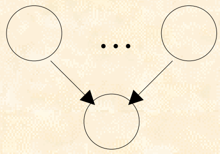
  next(d, e) **cancelled**(cons)

Intelligent backtracking

1. **Failure** is reported by a literal L_f.
2. Find a **proper** literal L_b to be **redone**.
3. **Reset all** of the **generator** literals whose order is later than L_b in LOLL.
4. **Cancel** messages to **all** of the **consumer** literals of **redone**, **reset**, and **canceled** literals

- Tuple generation model vs nested loop model

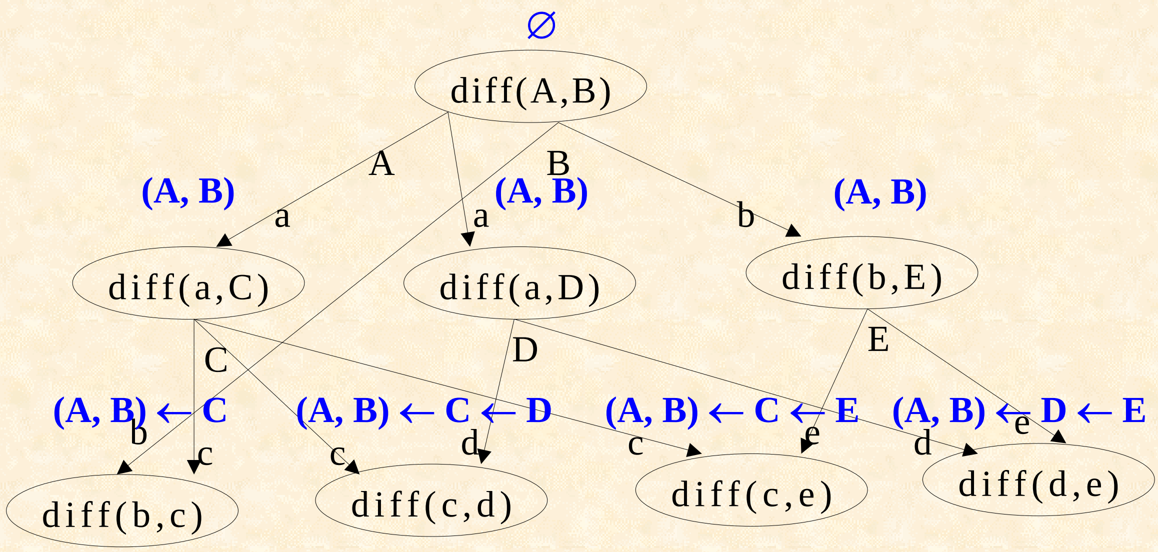
Cause of consecutive failures



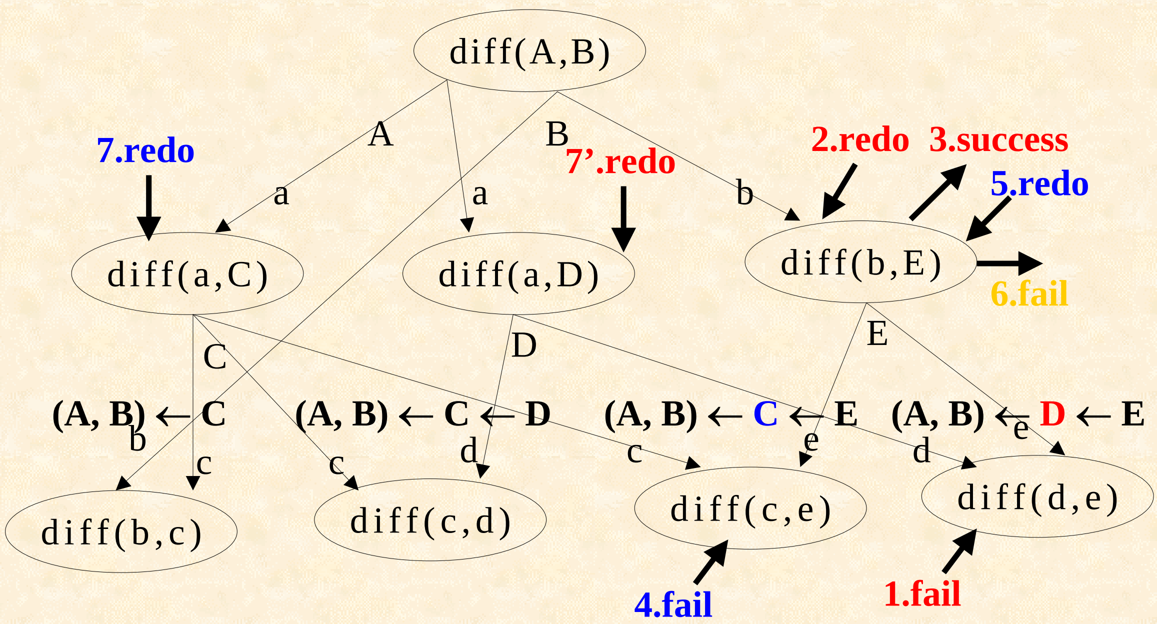
- If failed literal consumes more than one variable,
  the **last generator** in LOLL is **redone**
- If the last generator **fails** again
  the **remaining generators** of failed literals
  should be **redone** again.
- The **remaining generators** should be **stored**
  to consider the **other** failures.
  multiple failure

Conery's model

- Consecutive failure
  redo list for each literal(**static**)
- A **sequence** of the **last generators** of the literal



Improper redo – multiple failures



Redo Cause Set(RCS)

- Remaining **generators** for each generator
  - they must be redone if **this** generator **fails again**
  - type 2 backtracking in Conery
- **Multiple failure**
- remaining generators are added to
  - RCS of the **redone** generator

  When the failed literal report **success**
  - RCS of the **redone** generator is **updated**
- Lin, Kumar, and Lung in U. Texas
  - B-list

Implementation

- Prolog **parallel evaluator**
- **Front end**
  prolog program → internal representation(AST)
- **Back end**
  AST → creation of AND/OR processes
  written in **concurrent C**
- process management
  AND process
  OR process

Load Balancing

- **Dynamically** glowing tree
  static processor configuration
- number of processes >> number of processors
- **load balancing** vs **hop count**
  local optimization
  hop count = 1
- mesh or cube
  recursively circulant graph
  performance analysis

Research topics

- **Forest** model(이명준)
  - affection relation
  - multiple resetting
  - parallel backtracking
- **Unified** model(김도형)
  - Full solution level selective resetting
- **Algebraic** model(이수현)
  - Calculus of Communicating System(CCS)
  - Parallel model

Conclusion

- Parallel evaluation of logic programs
- **Intelligent backtracking**

그리고,

學而之銘名	名可名, 非常名
자연과학, 공학	인문, 사회과학
서양	동양
Evolution	**Revolution**
敎	禪
끊임없는 노력	한 순간에 오는 깨달음