# Logic Programming

Rodica Potolea
Camelia Lemnaru

Lecture #2, 2024
Cluj-Napoca

# Agenda

- LP paradigms – review
- Operational Semantic - review
- Execution tree
  - Representation
  - Mechanism

# Prolog PREDICATES
## (rules, facts, queries)

- **Clause** general form:

```
p(X):-q1(Y),q2(X,Y),…, qn(X,Z).
```

  **=** a **TEOREM** in form "conclusion if hypotheses"
  meaning q1^q2^…^qn->p
  Horn clause with at most one non-negated literal
  ¬q1 V ¬q2 V … ¬qn V p

- **Fact**

```
p(a).
```

  = a clause without body
  = a theorem WITHOUT any hypothesis = **AXIOM** (no need for proof)

- **Query**

```
?- q2(a,b).
```

  = a clause without head
  = a theorem WITHOUT conclusion

They define the **declarative semantics** of Prolog
= interpretation of the statements

# Execution of LP programs

## Unification mechanism – the core of LP

- **Q1**: For the query/head unification take
  - A1: first clause first = **top-down**
- **Q2:** In the query/head successful unification the body becomes the new goal. In a conjunction of goals
  - A2: first subgoal (sub-body) first = **left-right**
- **Q3:** In the query/head failed unification
  - A3: unification fails = **backtracking**

# Operational semantics in action

When building the tree, we start from the initial query in the matching process. For each (sub)goal (one at a time):

- C1: current goal succeeds? If *yes*, current node successful built & go C2, *else*, backtrack.
- C2: does it match a fact? If *yes*, current node is a leaf & go C3,

  *else*, go take and execute the entire body as goal (push it all on the execution stack, and pop stack's top).
- C3: is the stack empty? If *yes*, over (the whole execution ends successfully, the execution tree becomes at this very moment the deduction tree),

  *else* pop the top of the execution stack & go C1

# First conclusions

- a Prolog **program** is a set of **theorems** (complete clauses) and **axioms** (facts)
- Executing a Prolog program means **proving a new theorem** (the query/goal) from the existing ones (program)
- The execution relies on **solving** a set of **systems of linear equations**
- The number of systems = number of nodes in the deduction tree
- Each system has a number of equations equal to the number of arguments of the goal  executed in the corresponding node

# Main elements of Prolog

Built-in predicates = ready to use

| | | | |
|---|---|---|---|
| **var(X)** | if X unbound | then T | else F |
| **nonvar(X)** | if X bound | then T | else F |
| **atom(X)** | if X constant | then T | else F |
| **integer(X)** | self-explanatory | | |
| **atomic(X)** | if X atom or int | then T | else F |
| **call(X)** | executes X, where X is the name of a predicate<br>used in metaprogramming, where X gets instantiated at runtime | | |

# Main elements of Prolog – contd.

**=** infix operator (*equality*)

Prolog attempts to match (unify) the lhs with rhs

if successful, they are bound together from this point on

an un-instantiated var will become equal to ANYTHING as the unification succeeds

**==** infix operator (*identity*)

if X==Y then X=Y

if X=Y then X==Y NOT MANDATORY!!!!

an uninstantiated var will become identical to another uninstantiated variable ONLY if they are already sharing same location, otherwise FAILS

# Examples

?-X==Y no (Fails; even if both X and Y are free variables)

?-X==X yes, X=_some_number

?-X=Y,X==Y

yes, X=_some_number; Y =_some_number (SAME)

?-X=[a,b], Y=[a,b],X==Y

no (although same list [a,b] is JUST same content, they are NOT shared in memory, NOT same location)

?-X=[a,b], Y=X,X==Y

yes

**is** infix operator; rhs gets evaluated and the result instantiates the lhs (if successful) or else fails. ONLY lhs may get instantiated

X **is** some_expression

X is 2+3

yes, X=5

X=2+3

yes, X=2+3

2+3 **is** X

instantiation error

# Matching rules

| Goal(q) | Rule head (r) | Outcome |
|---|---|---|
| a | b | fails |
| a | a | succeeds |
| X | a | succeeds, and X gets instantiated to a |
| a | X | succeeds, and X gets instantiated to a |
| Q | R | succeeds, and Q and R become the same |

# Execution tree representation

**Tree built top-down and left-right (DFS)**

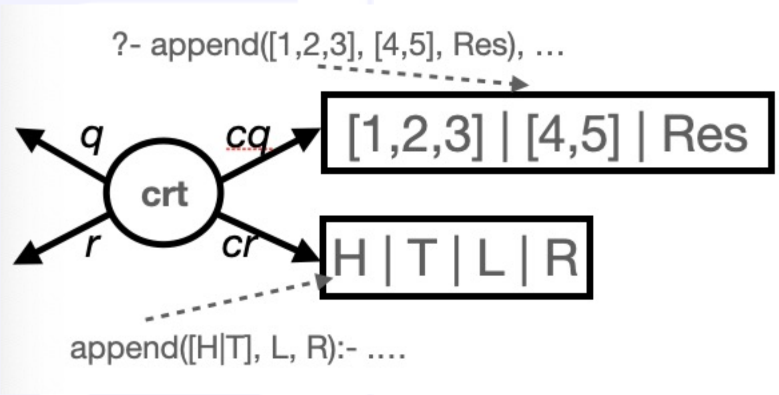
Is a multiway tree (represented as binary)

- Each node in the exe tree contains **pointers** to several structures
  - Data: Program and memory locations of the variables (**1**)
  - Linking: connections in the tree (to navigate during execution, including backtracking) (**2**)
- Several **actions** to build the tree (**3**)

# Execution tree: pointers to program and variables (1)

- A set of **4 pointers** to the:

| | |
|---|---|
| **query (goal)** | **q** |
| **head of the rule** | **r** |
| **context of the query (goal)** | **cq** |
| **context of the rule** | **cr** |



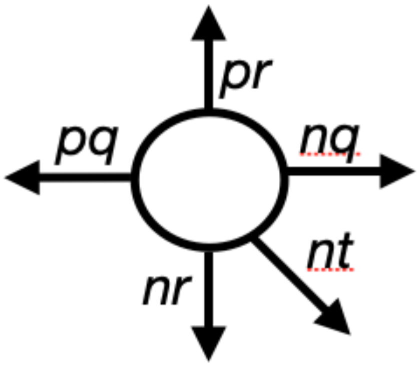
?- append([1,2,3], [4,5], Res), …

append([H|T], L, R):- ….

- The **query context** inherited from the parent node = *rule context of the parent node becomes query context of the child node* (it may be enhanced by the query context of a brother to the left due to hidden variables, or enhanced with new variables here)
- The **rule context** is allocated at the level of the current node = *ALL formal variables are indexed with the number of the node* (to keep track of the variable occurrence)

# Execution tree: Linking pointers (2)

- A set of **5 pointers** to other nodes in the tree (they provide the ability to make the sound actions according to the operational semantics):



**parent rule** **pr**
(parent node; for root, null)

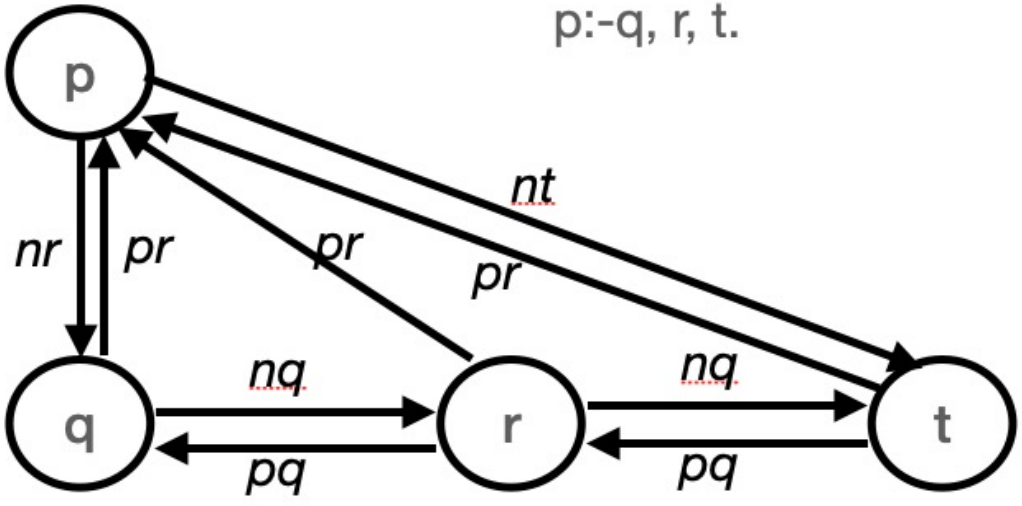
**previous query** **pq**
(sibling node to the left; for first child, null)

**next rule** **nr**
(first child; if any; for leaves, null)

**next query** **nq**
(sibling node to the right; for last child, null)

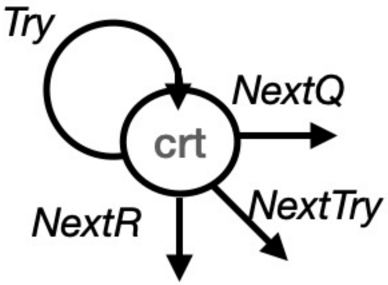
**next try** **nt**
(the next alternative to try in case of failure/backtracking; points to the rightmost child, to provide fast reachability to the last node in the tree)

# Execution tree:
## Linking pointers (2)



p:-q, r, t.

# Execution tree: Actions (3)



- **Try** - tries the definition of the predicate with the same name as the query (represents the matching between a query and the head of the rule)

the action BUILDS the **current node**

- **NextR-** enters the body of the clause at the first subgoal in the body, passing the conditional (**:-**)

the action BUILDS the **first child** of the current node

- **NextQ-** continues the body of the clause at the next subgoal in the body, passing a conjunction (**,**)

the action BUILDS the **first sibling to the right** of the current node

- **NextTry** – determines the first node to backtrack and launches the backtracking mechanism

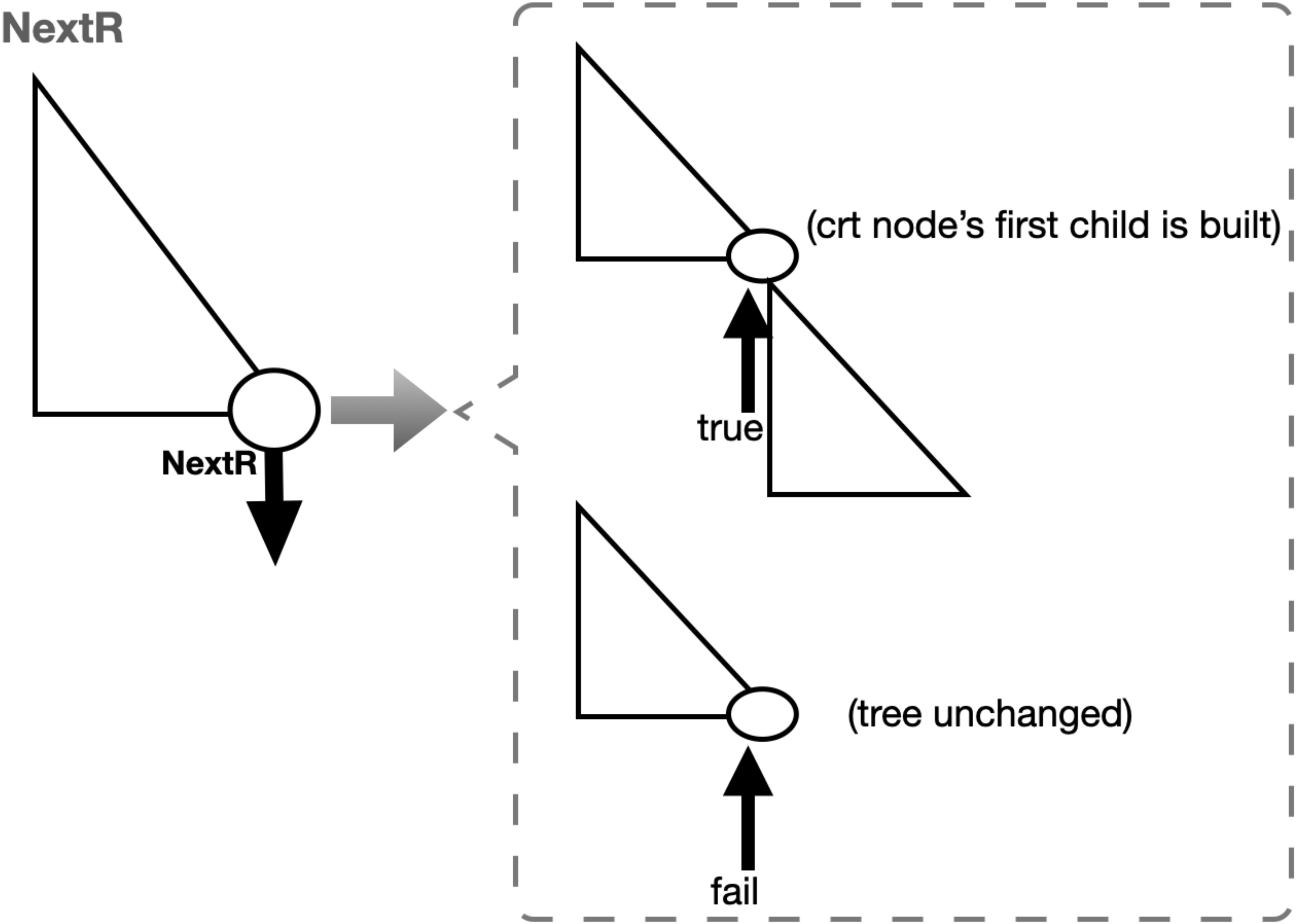
the action takes place in a tree already built, by trying an alternative solution for the node identified as responsible to backtrack (examples during seminars)

# Execution tree: Pointers and Actions – NextR

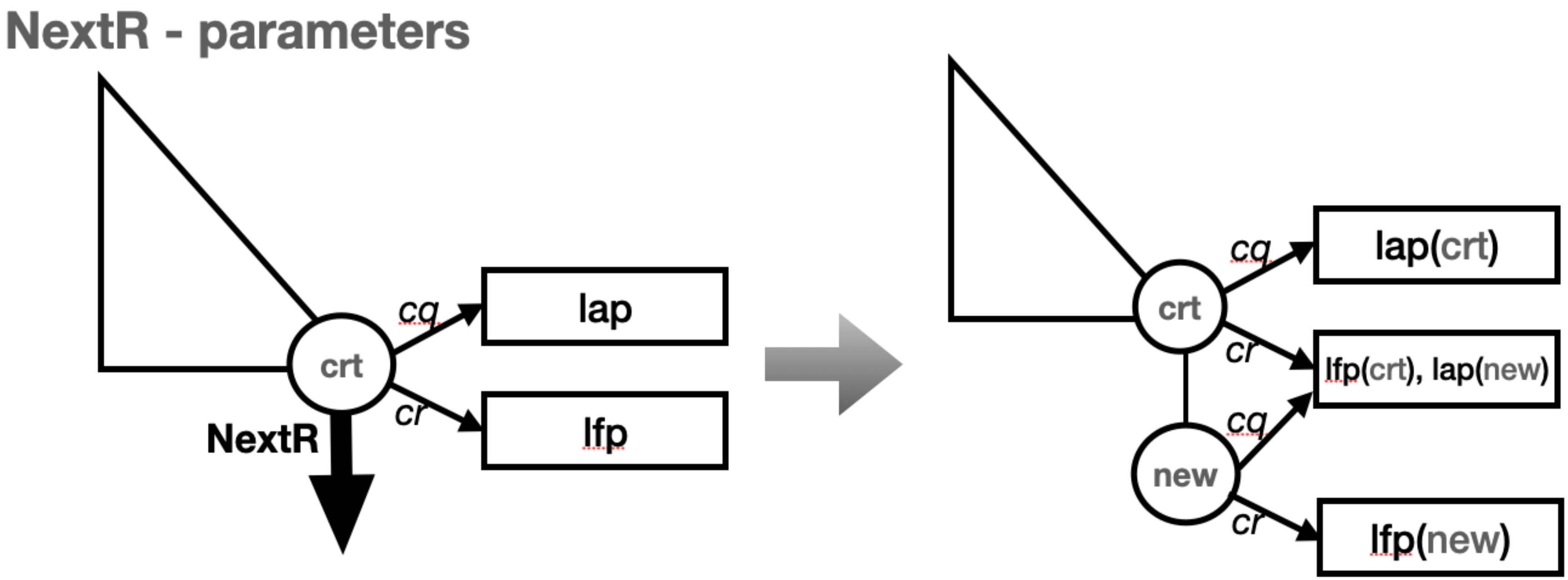
## NextR

- At least one *new* node is generated, the left child of the current node
- If it (*new* node; first child of current) matches a fact, it is a leaf
- If matches a complete rule, the entire tree rooted by it (*new* node; first child of current) is generated
- the list of **actual parameters** of the new node is **inherited** from the list of formal parameters of the current node
- The list of **formal parameters** of the new node is **allocated now** (thus, the corresponding variables are indexed with the number of the node)

# NextR


NextR
(crt node's first child is built)
true
(tree unchanged)
fail

# NextR – how arguments evolve



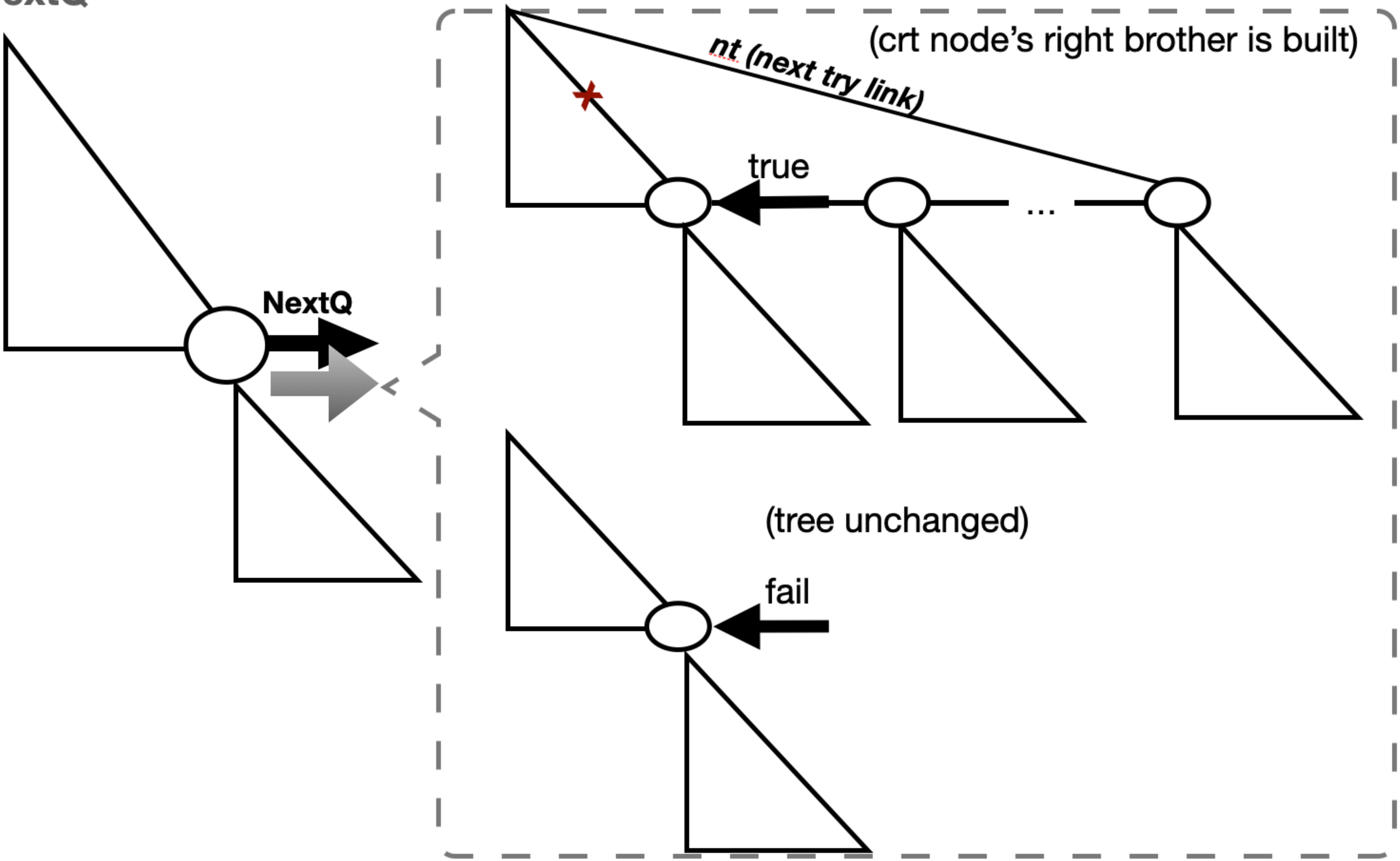
- the list of **actual parameters** of the new node is **inherited** from the list of **formal parameters** of the current node
- the list of **formal parameters** of the new node is **allocated** now (thus, the corresponding variables are indexed with the number of the node)

# Execution tree: Pointers and Actions – NextQ

**NextQ**

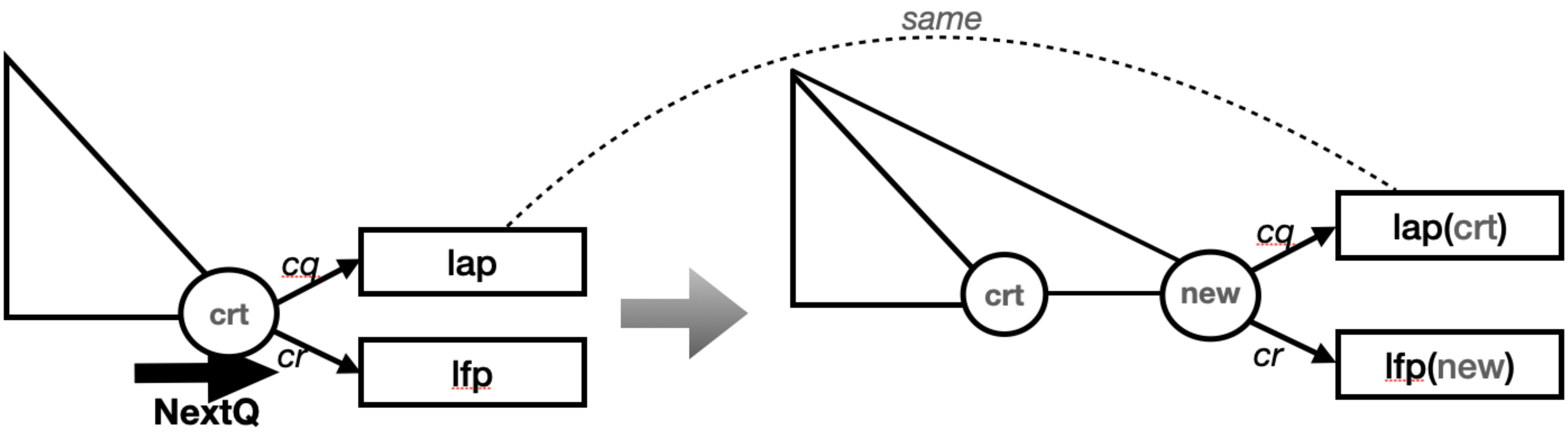
- At least one *new* node is generated, the right brother of the current node
- If it (*new* node; right brother of current) matches a fact, it is a leaf; else the entire tree rooted by is generated
- If it is NOT the rightmost sibling (NOT the last in conjunction, NOT before.), all its right siblings (with subtrees) are generated when completed
- the list of **actual parameters** of the new node is **inherited** from the list of actual parameters of the current node
- The list of **formal parameters** of the new node is **allocated now** (thus, the corresponding variables are indexed with the number of the node)

## NextQ


NextQ
(crt node's right brother is built)
nt (next try link)
true
...
(tree unchanged)
fail

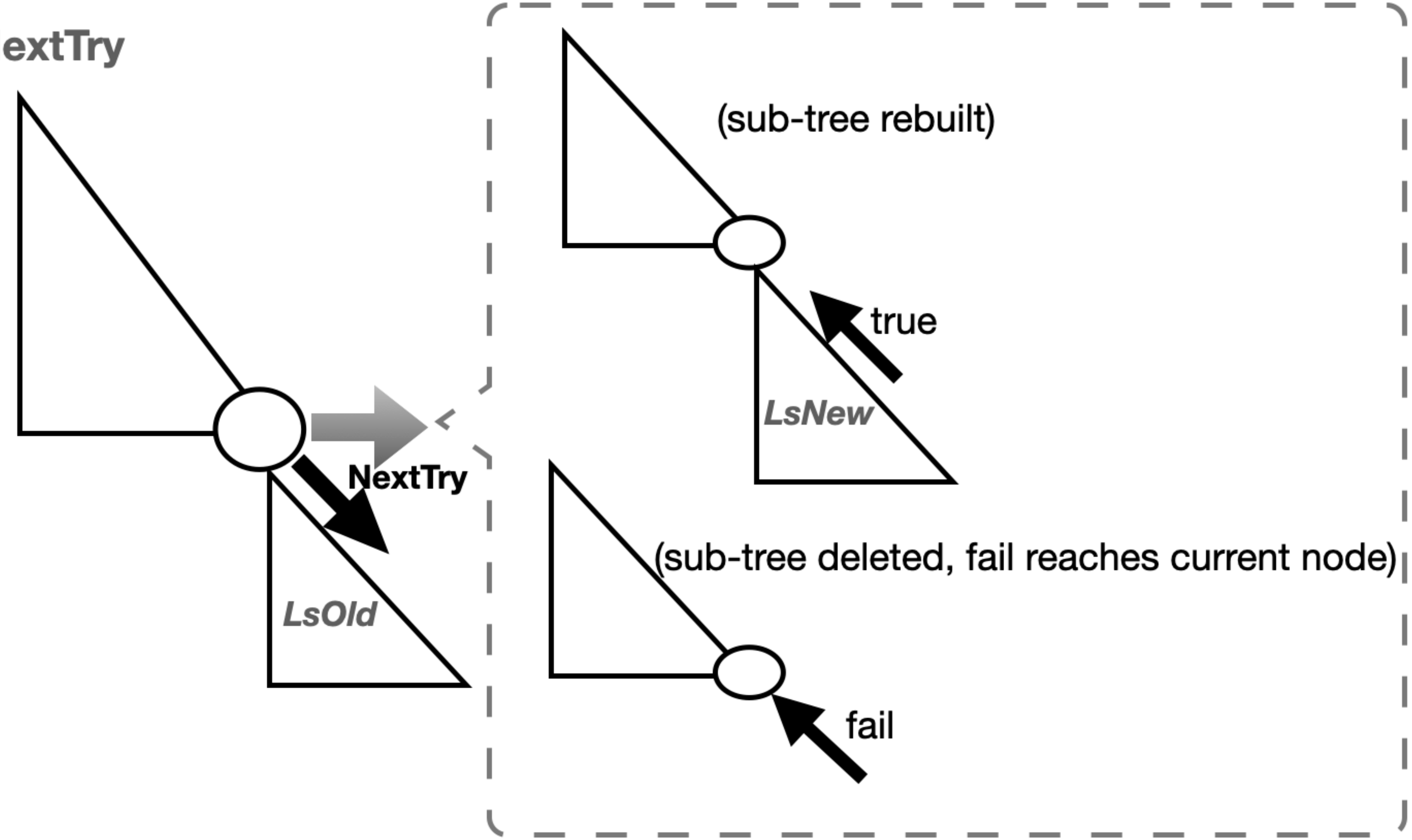
# NextQ – how arguments evolve

## NextQ - parameters



- the list of **actual parameters** of the new node is **inherited** from the list of **actual parameters** of the current node
- the list of **formal parameters** of the new node is **allocated** now (thus, the corresponding variables are indexed with the number of the node)

# NextTry


NextTry
LsOld
(sub-tree rebuilt)
true
LsNew
(sub-tree deleted, fail reaches current node)
fail

# Example 3rd

Check if an element is present in a list

How many arguments/why?

Predicate's signature is `member/2 (element, list)`

```
member(X,L):-
      L=[H|T],
      X=H.
member(X,L):-
      L=[H|T],
      X\=H,
      member(X,T).
member(X,[]):-
      fail.
```

# Example 3rd contd.

```
member(X,L):-
        L=[H|T],           //list not empty
        X=H.               //searched element matches the head of the list
member(X,L):-
        L=[H|T],           //list not empty
        X\=H,              //searched element does not match the head of the list
        member(X,T). // searched element found in the rest of the list
member(X,L):-
        L=[],               //if reached the empty list
        fail.               // searched element not found in the list
```

With default unifications, the predicate becomes:

```
member(H,[H|T]).  //default decomposition and unification
member(X,[H|T]):- //default decomposition
        X\=H,          //Is it necessary? Why/why not?
        member(X,T).
member(X,[]):-
        fail.          //Is it necessary? Why/why not?
```

# Example 3rd contd.

```
member(H,[H|T]).  //default decomposition and unification
member(X,[H|T]):- //default decomposition
        X\=H,  //Is it necessary?
        member(X,T).  //not really due to the search rule (A1)
member(X,[]):-        //Is it necessary?
        fail.         //not really due to the default failure by absence.
```

Thus, the predicate becomes:

```
member(H,[H|T]).
member(X,[H|T])
        member(X,T).
```

Qs:

- What happens if we reverse the order of clauses? Would it still be a correct predicate? Why/why not?
- How is better? Why?
- Trace a deterministic query
- Trace a nondeterministic query