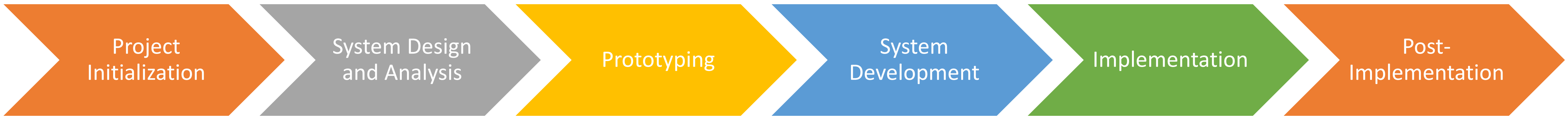
CHAPTER V

APPROACH TO THE DEVELOPMENT OF EXPERT SYSTEMS

DEVELOPMENT CYCLE



DEVELOPMENT CYCLE

- **Phase 1: Project Initialization**
  - Problem Definition
  - Needs Analysis
  - Evaluation of Alternative Solutions
  - Verify if the ES approach is appropriate
  - Consider Management Issues

DEVELOPMENT CYCLE

- **Phase 2: System Design and Analysis**
  - Define a conceptual design
  - Define a development strategy
  - Identify knowledge sources and ensure cooperation
  - Select computer resources
  - Ensure cost-benefit analysis

DEVELOPMENT CYCLE

- **Phase 3: Prototyping**
  - Build a small prototype
  - Test, improve, and expand it
  - Analyze feasibility
  - Complete the design

DEVELOPMENT CYCLE

- **Phase 4: System Development**
  - Build the knowledge base
  - Test, evaluate, and improve this base
  - Plan for integration

DEVELOPMENT CYCLE

- **Phase 5: Implementation**
  - Ensure user acceptance
  - Install and deploy the system
  - Guide and train users
  - Ensure security
  - Provide documentation

DEVELOPMENT CYCLE

- **Phase 5: Post-Implementation**
  - Maintenance
  - Updates
  - Periodic evaluation

DESIGN AND IMPLEMENTATION

- **Conventional design (engine programming, etc…)**
  - Programming Languages (C, C#, Java, Python, ..)
- **Use of ES generators (bare inference engine)**
  - M1, OPS5, MP-LRO
- **Use of AI programming languages**
  - Functional languages: LISP, ML, CAML
  - Logical languages: PROLOG
  - Interpreters: CLIPS, JESS
  - Graphical tools: ES Builder

PROLOG

Characteristics

Prolog is logical:
The program can be seen as a series of axioms that describe a problem.

Prolog uses a true notion of variables:
Variables refer to unknown objects in search. They are managed by the system, which assigns them values and undoes them.

Prolog is non-deterministic:
Defined functions can have multiple values, and Prolog handles the search for these values by backtracking where necessary.

PROLOG REASONING

Stucture of a program

A PROLOG program is composed of three parts: Facs, Rules and Queries (Goals)

Example :

```
%facts
personne(léon,35).
personne(lucie,27).
personne(louis,40).
personne(pauline,9).
personne(luc,27).

%rules
individu(x) :- personne(x,_).
majeur(x) :- personne(x,y),y=>18.
mineur(x) :- personne(x,y),y<18.
```

Implementation (queries)

```
?- individu(pauline).
→true

?- individu(jacque).
→ false

?- personne(X,27).
→ X=lucie;
→ X=luc.

?- personne(louis,X)
→ X=40.

?- mineur(X).
→ X=pauline.
```

Prolog Properties

Remarks:

- The mechanism of Prolog is **correct**:
  - It only provides logically correct answers.
- The mechanism of Prolog is **complete**:
  - When it stops, we can be sure that it has provided all the necessary answers.

Meaning of rules in the Clauses section

Simple Clauses (Facts):

Example: Person(leon, 35).

Reads as: Leon is a person of 35 years.

Complete Clauses (rules): Contain the symbol :-

They correspond to general statements involving variables.

- The symbol ':-': Means If, can be replaced by if.
- The symbol ',': Means AND, can be replaced by and.
- The symbol ';': Means OR, can be replaced by or.
- The negation is represented by: not.
- Variables: Represent arbitrary objects.
  Example: X, Y, Person.
- The symbol '_': Anonymous variable (means forall)

PROLOG REASONING

Meaning of rules in the Clauses section

- **Meaning:**

- personne(léon,35). means:
It is true that Léon is a person of 35 years old **OR**
The goal Personne(Léon,35) is satisfied.

- individu(x):-personne(x,_). means:
X is an individual if it exists a fact personne(X,_). **OR**
The goal individu(x) is satisfied for each X so that Personne(x,_) is staisfied.

- mineur(x) :- personne(x,y),y<18. means:
X is a mineur if X is a personne of age Y and Y<18 **OR**
The goal Mineur(x) is satisfied for each X so that the goal Personne(X,Y) is satisfied with Y<18.

PROLOG REASONING

Meaning of rules in the Clauses section

- **Other precisions:**

\- The symbol ';' can be used within the body of a rule, it means **or**:
a(x):-b(x);c(x). This rule could be written:
a(x):-b(x).
a(x):-c(x).

\- The logical meaning of a clause is the meaninf of an implication between the body of the rule and its head and where the variables are all universally quantified.
Examples :
a(x):-b(x),c(x). means: $\forall$ x b(x) ∧ c(x) $\Rightarrow$ a(x)
a(x):-b. means: b $\Rightarrow$ $\forall$ x a(x)
a(x):-b(x,y). means: $\forall$ x ($\exists$ y b(x,y)) $\Rightarrow$ a(x)

PROLOG REASONING

Reasoning principle

1- Handle the leftmost goal.
2- To satisfy the goal being handled:

+ Search in the program order for the first rule not yet attempted at this point and whose head is compatible with the goal
+ Then replace in the goal list the first goal with the body of this rule, making all substitutions.

3- Whenever a failure occurs, backtrack to the most recent list of goals where a rule could be used and has not been (backtracking).

Note: This systematic search type works very well with trees. Each node is a list of goals, and the traversal is according to the technique:

Left - Right - Depth-first with Backtracking.

Example:

R1: Est-de-bonne-humeur(x):-A-de-l-argent(x),Est-en-vacances(x),Il-y-a-du-soleil.

R2: Est-de-bonne-humeur(x):-Réussit-dans-le-travail(x),Réussit-dans-sa-famille(x).

R3: A-de-l-argent(Jean).

R4: A-de-l-argent(Alain).

R5: Est-en-vacances(Jean):-On-est-en(Aout).

R6: Est-en-vacances(Alain):-On-est-en(Juillet).

R7: On-est-en(Juillet).

R8: Il-y-a-du-soleil:-On-est-en(Aout).

R9: Réussit-dans-le-travail(Jean).

R10: Réussit-dans-le-travail(Alain).

R11: Réussit-dans-sa-famille(Alain).

Goal: Est-de-bonne-humeur(x) ?

R1: Est-de-bonne-humeur(x):-A-de-l-argent(x),Est-en-vacances(x),Il-y-a-du-soleil.

R2: Est-de-bonne-humeur(x):-Réussit-dans-le-travail(x),Réussit-dans-sa-famille(x).

R3: A-de-l-argent(Jean).

R4: A-de-l-argent(Alain).

R5: Est-en-vacances(Jean):-On-est-en(Aoute).

R6: Est-en-vacances(Alain):-On-est-en(Juillet).

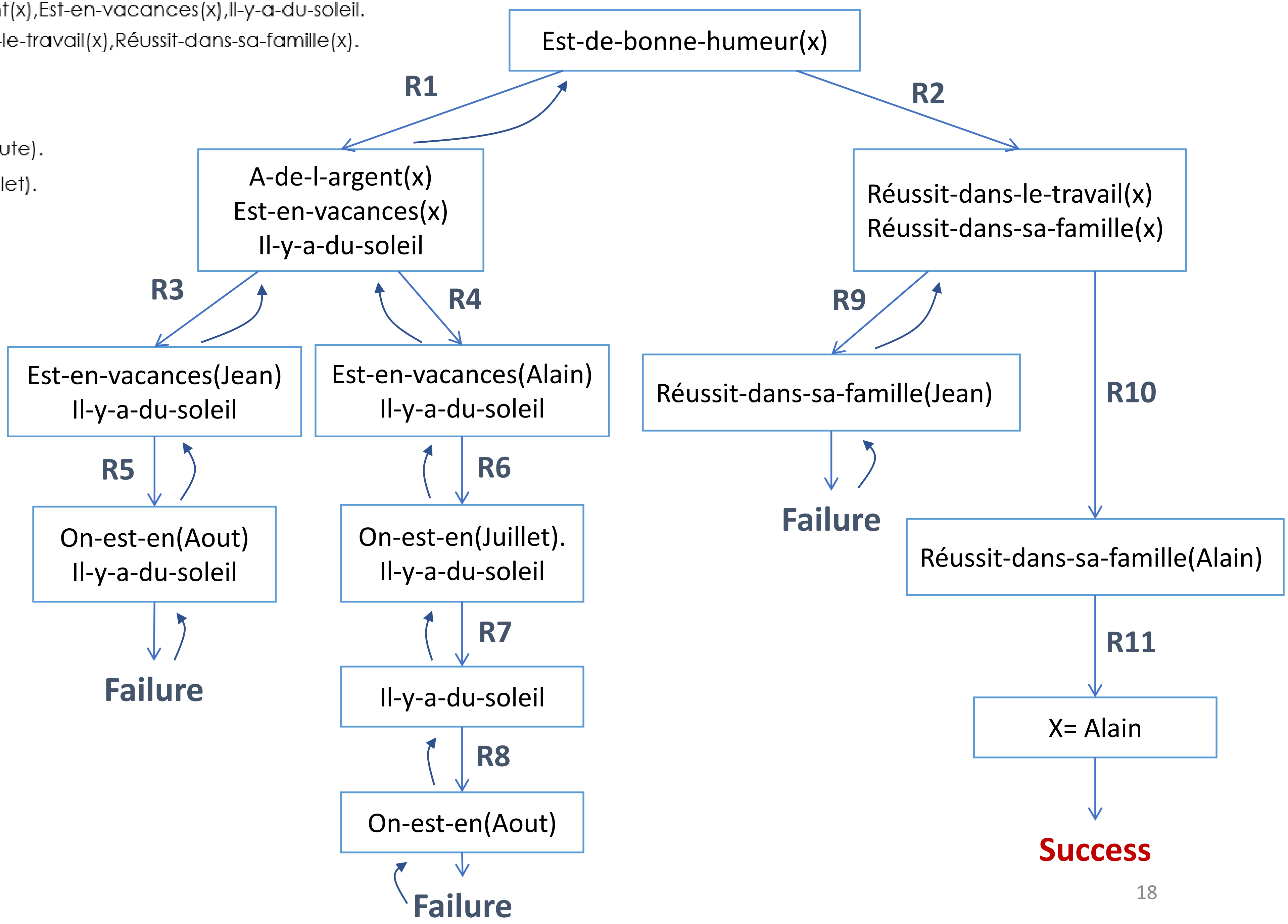
R7: On-est-en(Juillet).

R8: Il-y-a-du-soleil:-On-est-en(Aoute).

R9: Réussit-dans-le-travail(Jean).

R10: Réussit-dans-le-travail(Alain).

R11: Réussit-dans-sa-famille(Alain).



Exercise :

Redo the reasoning by replacing

On-est-en(Juillet) by ***On-est-en(Aout)***

PROLOG REASONING

Processing the negation

When encountering negations, Prolog reasons with the principle of **Negation by Failure**.

- If Prolog reasoning applied to goal P yields success, then the goal ¬P is considered to result in failure.
- If Prolog reasoning applied to goal P yields failure, then the goal ¬P is considered to result in success.

Whenever a goal of the form ¬P is encountered, it must be processed as follows:

- Build an auxiliary reasoning tree with P as its root.
- If this tree yields success, return to the main tree marking ¬P with a failure.
- If this tree yields failure, return to the main tree marking ¬P with success.

PROLOG REASONING

Negation by failure

¬P → (2) → Failure

P → (1) → Success

¬P → (2) → Success

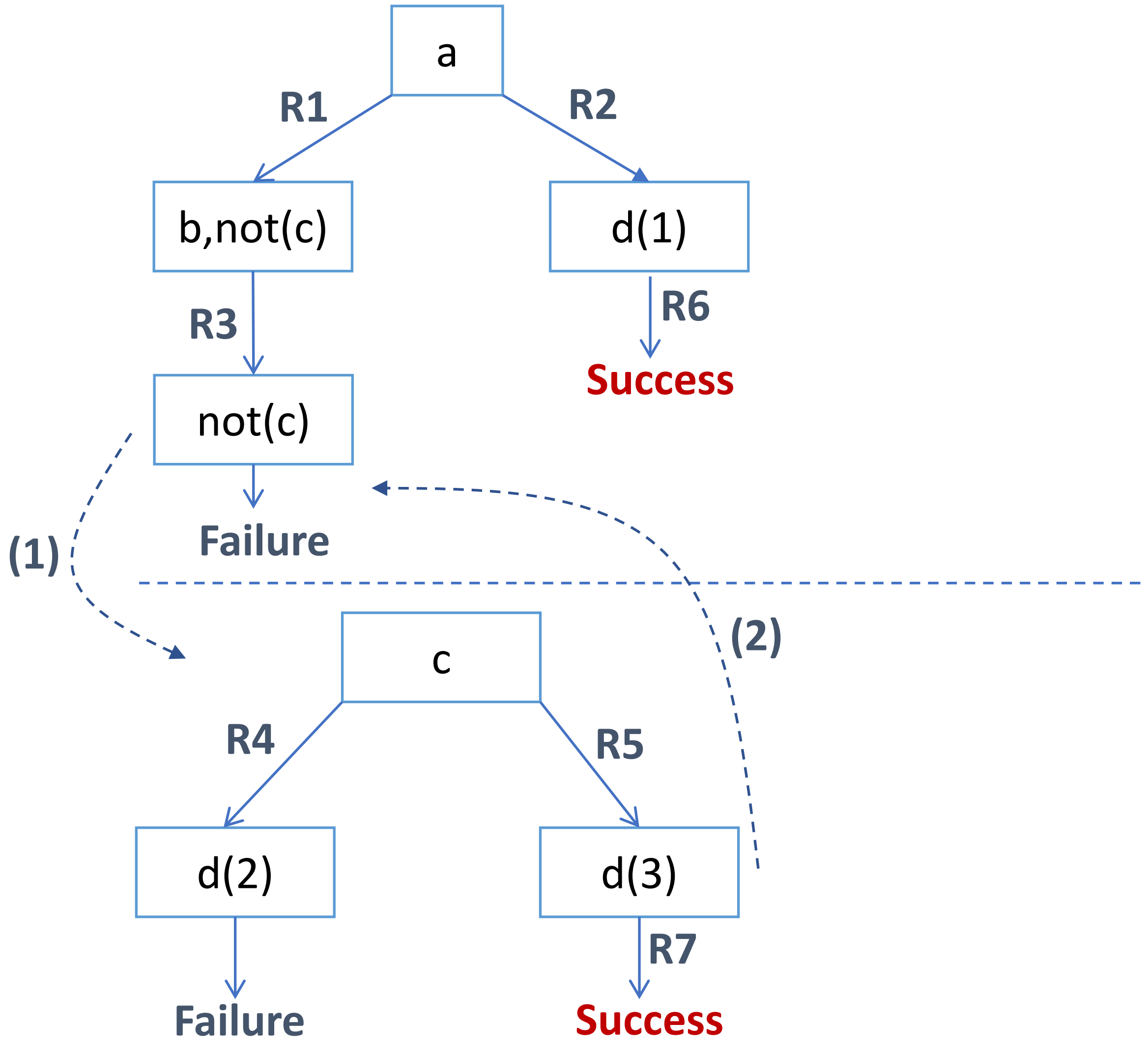
P → (1) → Failure

Negation by failure

Example 1:

R1: a:-b,not(c).
R2: a:-d(1).
R3: b.
R4: c:-d(2).
R5: c:-d(3).
R6: d(1).
R7: d(3).

Query: a ?



PROLOG REASONING

Negation by failure

Example 2:

R1: a:-b,not(c).
R2: a:-d(1).
R3: b.
R4: c:-d(2).
R5: c:-d(3).
R6: d(1).
R7: d(3).
R8: e:-d(1),not(a).
R9: e:-not d(2),d(3).

Query: e ?

The CUT « ! »

- The **cut "!"** is a very special predicate; it always succeeds (true) and it affects the way the reasoning tree is traversed.
- During the traversal of the reasoning tree, if a list of goals starting with a cut "!" is encountered:
  - It is necessary to backtrack directly above the goal that introduced this cut.
  - When the cut is encountered, it prunes all choices originating from the goal that introduced the cut.

The CUT « ! »

Example: A ?

A:-B,C.

A:-D.

A:-E.

B:-F,!.

B:-H.

B:-I.

E.

F:-G.

F:-J.

F:-K.

G.

G:-L.

G:-M.