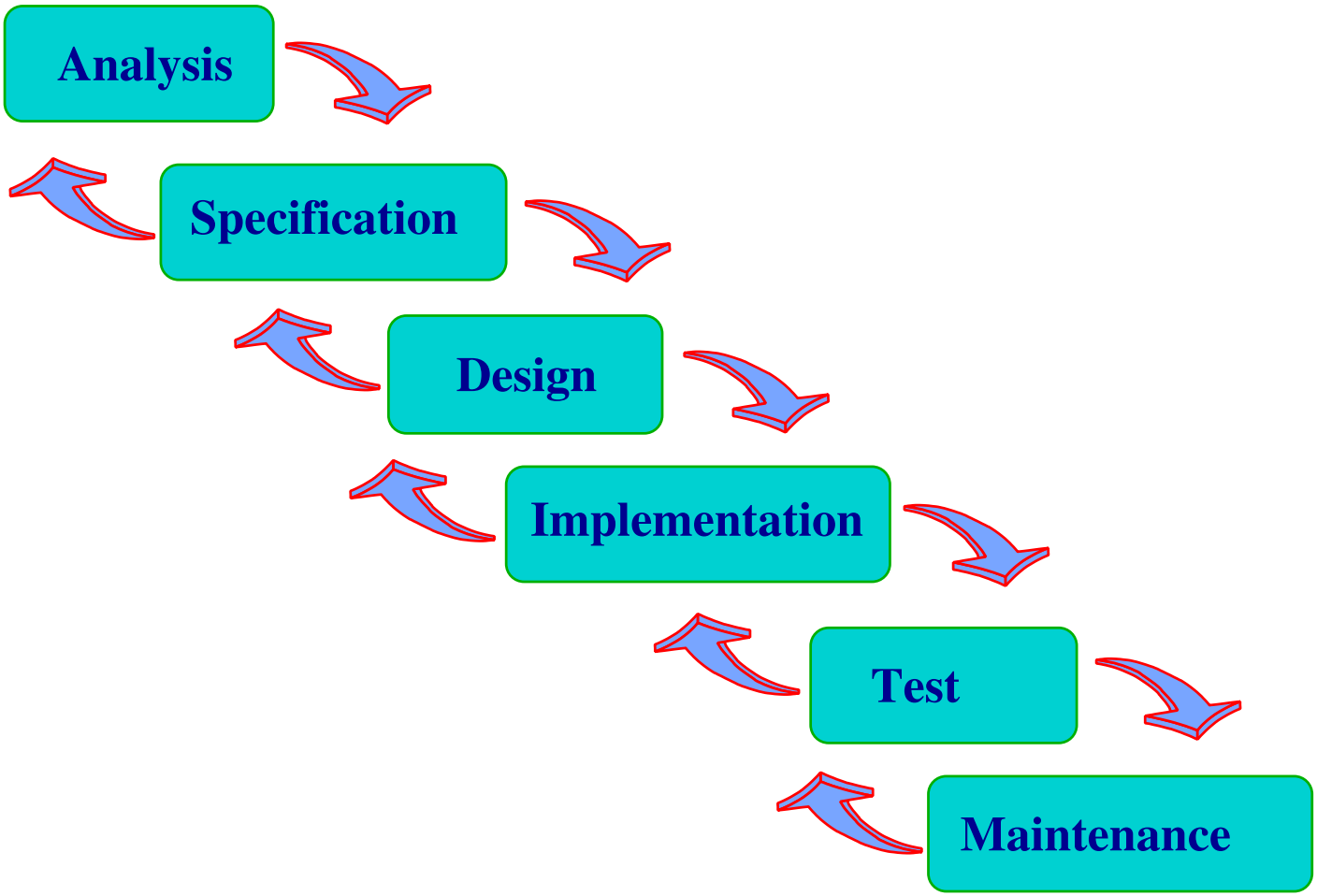
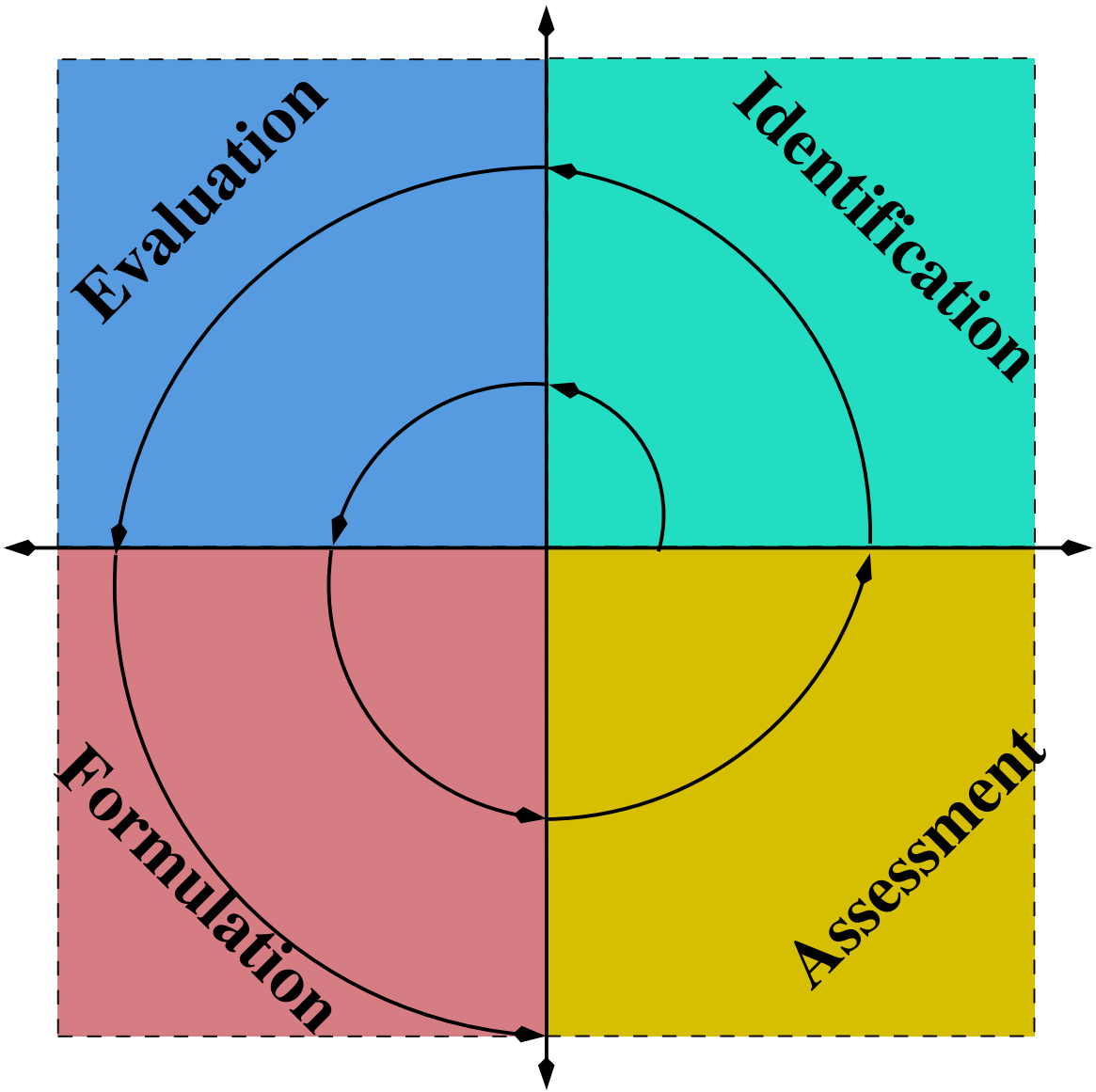
Development of KBS

- The most important phase in the development of a KBS is knowledge elicitation (knowledge extraction))
- This requires the interaction between the **Knowledge Engineer** and the expert
- The methodologies of software engineering have to include this step in their development process
- The methodologies of software engineering have to be adapted to the specific needs of KBS

SI: Waterfall model



SI: Spiral model



Particularities of the KBS

- Conventional software systems $\Longrightarrow$ Known and common algorithms
- KBS $\Longrightarrow$ incomplete, uncertain and heuristic knowledge
- Conventional software systems $\Longrightarrow$ It is possible to give an estimate of what knowledge is needed and its volume
- KBS $\Longrightarrow$ It is difficult to know what knowledge will be needed and its volume

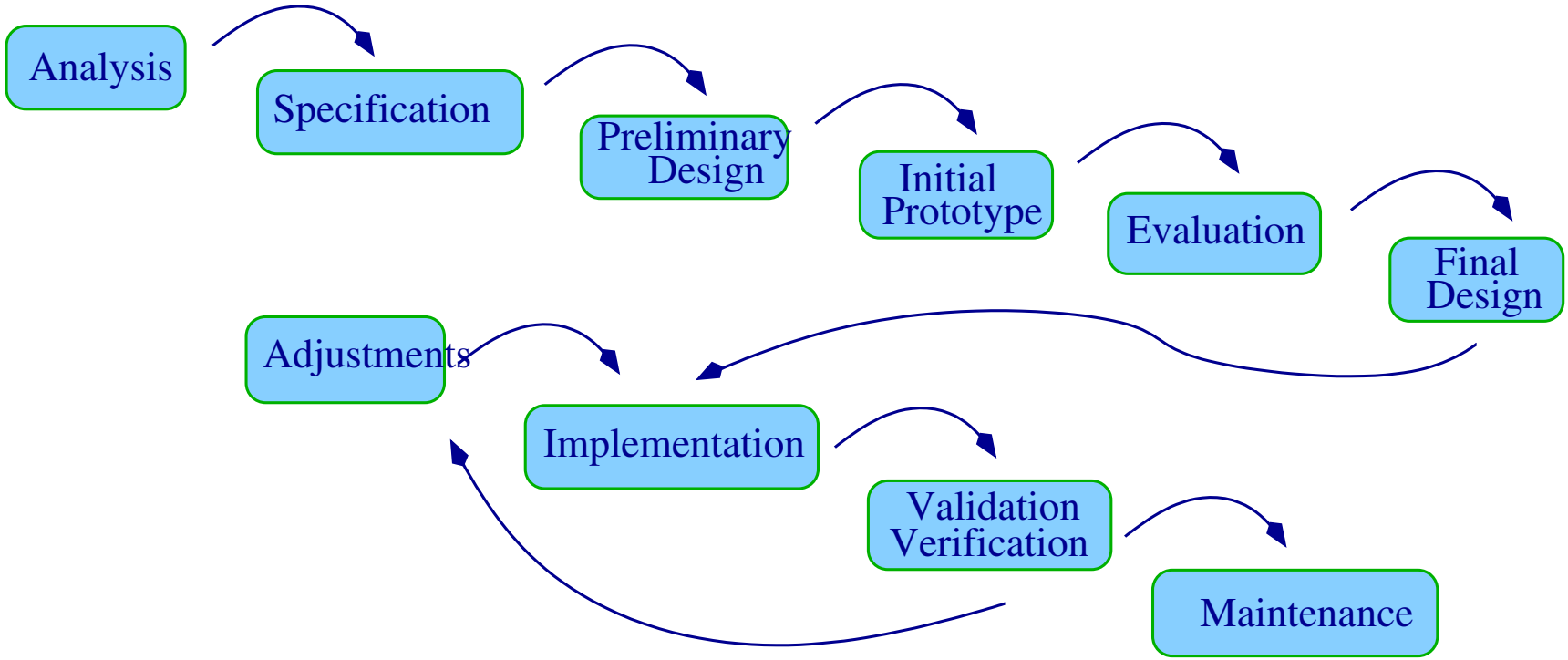
Particularities of the KBS

- It is difficult to obtain a clear design at the early stages of the development
- Wrong initial decisions imply radical changes in the design during the development
- The **knowledge engineer** has to perform the task of knowledge acquisition $\Longrightarrow$ Interviews with the experts
- Problems:
  - The KE has to learn the basic principles of the domain
  - To find a knowledge representation formalism that the expert can understand
  - The experts prefer to reason from particular cases rather than from general definitions
  - It is difficult for the experts to detail explicitly their knowledge (expert's paradox)

Particularities of the KBS

- Solution: **Incremental design and rapid prototyping**
- Goal: To develop a functional prototype that has all the basic functionalities of the system
- The analysis and the specification has to take in account the whole system
- The design and implementation are limited to the initial prototype
- This prototype is completed incrementally
- Advantage: A functional system is always available during all the development process

The life cycle of a KBS



The life cycle of a KBS (I)

1. **Analysis of the problem:** Gather information about the project and assess its viability
2. **Requirements specification:** Determine the goals of the project and the methodologies to achieve them
3. **Preliminary design:** High level decisions about the design (Knowledge representation formalism, tools, sources of knowledge)
4. **Initial prototype and evaluation:** Build a prototype with limited coverage, evaluate the design decisions from the prototype
5. **Final design:** Validate the decisions and propose a design that allows an incremental development

The life cycle of a KBS (II)

6. **Implementation:** Complete the knowledge acquisition process and complete incrementally the initial prototype
7. **Validation and verification:** Test and validate that the system works accordingly to specifications
8. **Design adjustments:** Apply the feedback from the tests (Design changes should be minimal)
9. **Maintaining:** Maintain the system.

Specialized methodologies

- **CommonKADS**
  - Spiral life cycle and modeling using UML-like formalism
  - Six models are built: Organization, tasks, agents, communication, knowledge and design.
- **MIKE**
  - Spiral life cycle: Knowledge acquisition (acquisition model and knowledge structure model), design, implementation, evaluation.

A simplified methodology

- For developing small KBS a methodology that follows the waterfall model can be applied
  1. **Problem identification**
  2. **Conceptualization**
  3. **Formalization**
  4. **Implementation**
  5. **Validation and Test**

Identification

- We have to assess if the problem is adequate for developing a KBS
  - Is there an algorithmic solution?
  - Are there available knowledge sources?
  - Is the size/goal/complexity of the problem adequate?
- Search and evaluate knowledge sources
- Determine what knowledge is necessary to build the system
- Determine the goals of the system (what do we expect from it?)

Conceptualization

This phase will give us a perspective of the problem from the expert point of view

- We have to:
  - Determine the elements of the domain $\Longrightarrow$ Informal description of the ontology
  - Decompose the problem in subproblems by means of successive refinement, discovering the reasoning blocks
  - Detail the flow of reasoning and the inputs and outputs of each subproblem
  - Detail and distinguish among evidences, hypothesis and actions and discover their relations
- All this information has to be acquired by means of interviews with the experts and from the knowledge sources
- The result will be a semiformal model of the domain, the subproblems and the problem solving methods used by the expert

Formalization

This phase will transform the perspective of the problem from the expert point of view to the knowledge engineer point of view

- Decide the knowledge representation formalism that is more adequate
- Identify the problem search space
- Analyze the subproblems typology and the reasoning blocks and decide what methodologies for problem solving are more adequate
- Analyze the need for the treatment of uncertainty or incomplete information

Implementation

- Build the ontology of the domain
- Embed the problems identified inside the chosen methodologies for problem solving
- Implement the different modules corresponding to each problem using the knowledge acquired
- If an approach based on rapid prototyping is used, the development will start with an initial prototype that will be incrementally completed

Validation and test

- Some representative cases will be chosen and solved using the system
- These should include cases used to build the systems and also new cases
- If the development is done incrementally this phase will be repeated each implementation cycle
- The validation of a KBS is more complex than conventional software systems

Classification of problems

- The identification of a typology of problems that can be solved using KBS helps to development
- Each type allows to identify:
  - The most common tasks
  - A set of specific problem solving methodologies
  - Adequate methods for knowledge representation and inference
- Two generic tasks will be used to classify the problems that can be solve by KBS:
  - **Analysis tasks:** Interpretation of a system
  - **Synthesis tasks:** Construction of a system

Synthesis - Analysis

Both can be specialized

- Analysis tasks
  - Identification, tells what kind of system we have
    - Monitorization, detects discrepancies on behaviour
    - Diagnostics, explains discrepancies
  - Prediction, tells what kind of output can be expected
  - Control, determines what inputs allow certain outputs
- Synthesis tasks
  - Specification, searchs for the constraints that have to be satisfied
  - Design, generates a configuration of elements following some constraints
  - Assembling, builds a system putting together elements

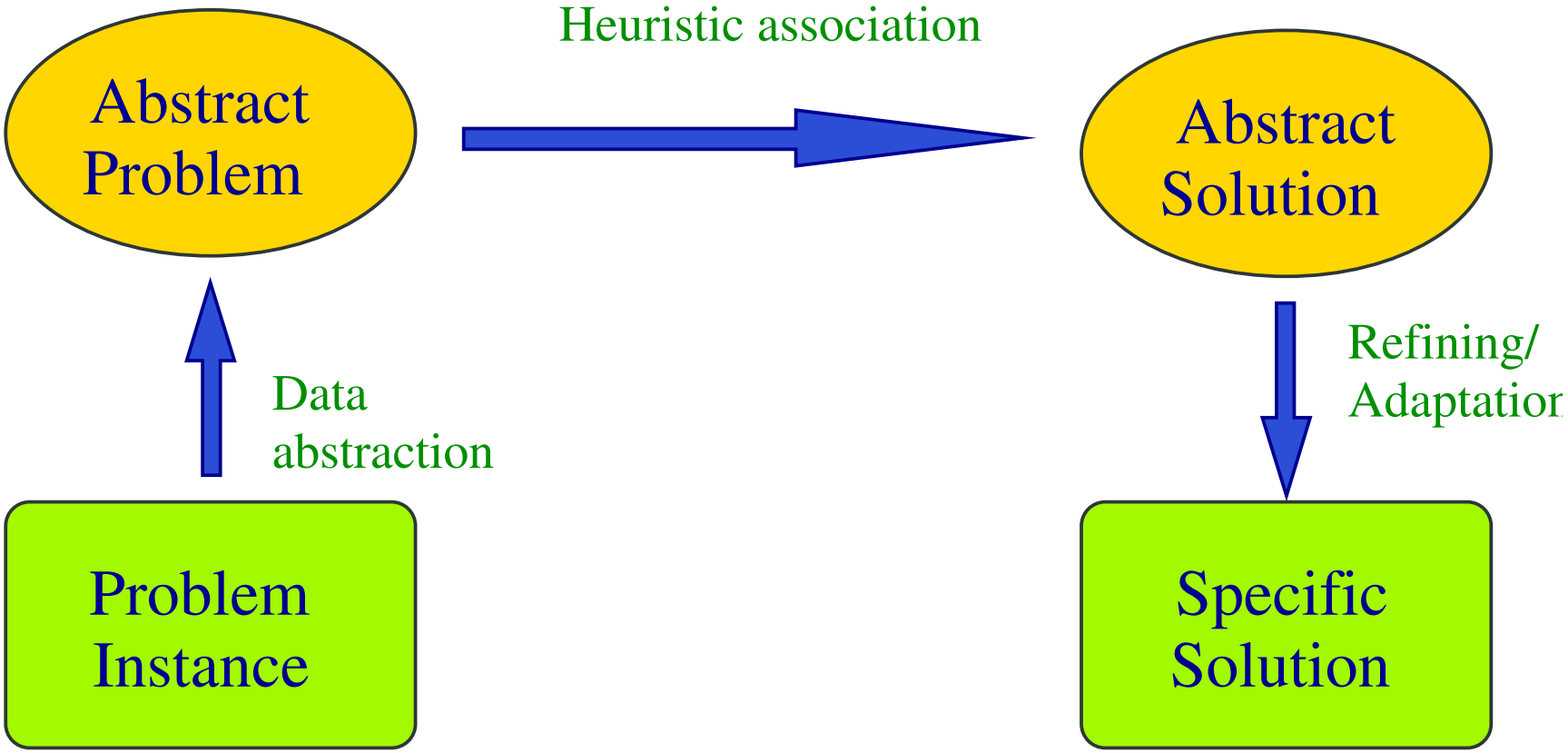
Methodologies for problem solving

- Each one of the generic problems has special characteristics
- There are specific problem solving methodologies for each kind
- We will talk about two methodologies:
  - *Heuristic Classification*
  - *Constructive Problem Solving*

Heuristic Classification

- It is used for **analysis** tasks
- The goal is to choose a solution from a limited set of solutions
- Input data is associated with the solutions (simple direct association or using reasoning)
- Three phases:
  1. **Data Abstraction** (Definitional, qualitative, generalization, ...)
  2. **Heuristic Association**
  3. **Refining**

Heuristic Classification



Heuristic Classification: Knowledge acquisition

- The acquisition of the knowledge to solve problems that use heuristic classification can be done systematically
- Three kinds of concepts can be distinguished:
  - **Hypothesis**: Possible solutions to our problem
  - **Symptoms**: Characteristics that describe the hypothesis
  - **Initial causes**: Initial information about the problem that lead to the Symptoms
- From each set of concepts we need to obtain the set of deductions that go from one to another
- From the initial causes to the symptoms we have the abstraction rules
- From the symptoms to the hypothesis we have the heuristic association rules

Heuristic Classification: Knowledge acquisition

- For each set of concepts we have to:
  - See what concepts from the first set (antecedents) are associated with concepts of the second set (consequents)
  - Choose as antecedents of the rules the concepts that are more specific to each consequent (separability)
  - If it is necessary new intermediate concepts should be added to link the consequents and antecedents and to create the needed chains of deduction
  - Observe the confidence of the association between antecedent and consequent (uncertainty)
- If the hypothesis are abstract solutions $\Longrightarrow$ Determine rules of solution refining

Heuristic Classification: Example (1)

- We want to develop a KBS to assess bank loan requests for creating a business
- There is a limited set of solutions (accept or decline)
- The goal is to decide, given the client characteristics, if the request is accepted and under what conditions, or if the request is declined
- This is an **analysis** problem that can be solved using **Heuristic Classification**.

Heuristic Classification: Example (2)

Lets suppose that a loan request has the following information:

- If the client has bank guarantees.
- If some relative of him guarantees the loan.
- If he has accounts, houses, cars, other properties and their value
- If he has antecedents of unpaid debts
- If he has signed bad checks
- If he has loans already granted
- Kind of business that the client want to create
- Amount of money that he is asking for

Heuristic Classification: Example (3)

Determine the set of characteristics that define the abtract problems

- Financial guarantees (Very good, good, normal, bad, very bad)
- Assets
- Fiability of the loan
- Compromise with the client
- Viability of the business

Heuristic Classification: Example (4)

Determine the set of abstract solutions

- Decline
- Grant
- Grant but reducing the amount
- Grant and give a preferential interest rate

Heuristic Classification: Example (5)

Determine the rules that abstract from the problem data

- if *guarantees* $>$ a million euros or rich uncle then *financial guarantees*=good
- if *guarantees* $<$ 100000 euros then *financial guarantees* bad
- if sum of *assets* $<$ a million then *assets*=bad
- if sum of *assets* $>$ two millions then *assets*=good
- if *bad checks* or *unpaid debts* then *fiability*=very bad
- if *fast food business* or *ice scream business* then *viability*=normal
- if *chain of stores* or *ISP* then *viability*=very good
- if *loans already granted* $>$ a million or *brother of branch director* then *compromise*=good
- ...

Heuristic Classification: Example (6)

Determine rules that associate characteristics to solutions

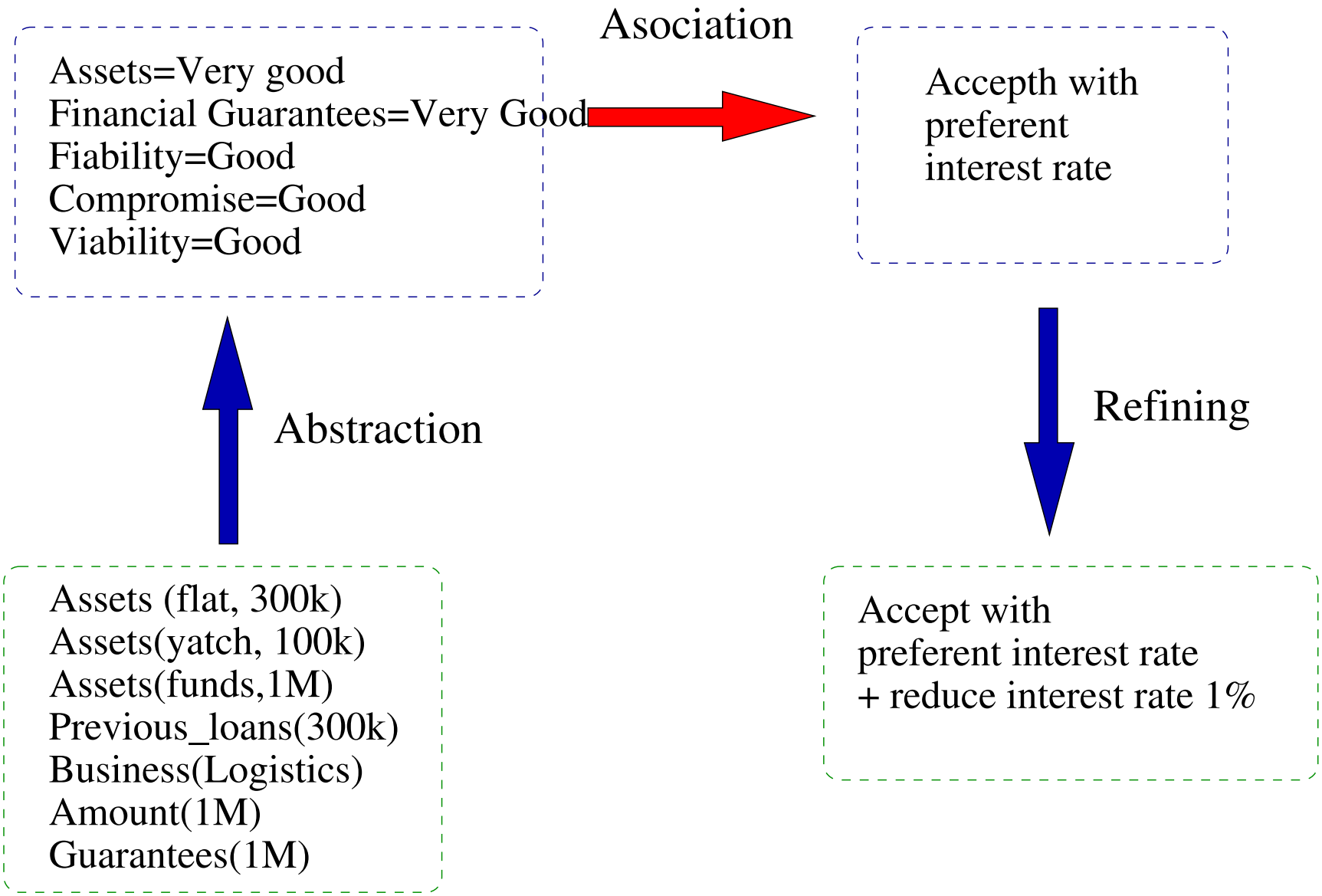
- if *financial guarantees*=normal and *assets*=bad then *decline*
- if *fiability*={bad, very bad} then *decline*
- if *financial guarantees*=normal and *assets*=normal and *viability*=good then *grant but reducing*
- if *financial guarantees*=good and *assets*=normal and *compromise*=normal and *viability*=good then *grant*
- if *financial guarantees*=good and *assets*=good and *compromise*=very good and *viability*=very good then *grant with preferential interest rate*
- ...

Heuristic Classification: Example (7)

Determine rules to refine the solutions

- if *grant but reducing* and *amount* $>$ 500000 euros and *assets* $=$ 500000 euros then *reduce* to 500000 euros
- if *grant with preferential interest rate* and *amount* $>$ a million and *assets* $>$ a million then *reduce* 1 % the interest rate
- if *grant with preferential interest rate* and *brother of branch director* then *reduce* 2 % the interest rate
- ...

Heuristic Classification: Example (8)



Constructive problem solving

- Used for problems that have an unbounded number of solutions
- The resolution implies to build the solution from a set of elements (actions, components, failures, ...)
- It is applied to **synthesis** tasks
- Heuristic and local search methods can be used but with impracticable time cost

Constructive problem solving

- To build a solution we need knowledge about:
  - The model that describes the structure of the solution
  - The model of behaviour of the components of the solution
  - The actions that allow to build the solution
  - The set of constraints between the components and the solution
  - How to evaluate the decisions about the actions needed to build the solution and about the quality of the solution (complete or incomplete) itself
- The constraints could be:
  - About the configuration of the components (Physical, temporal, ...)
  - About the inputs/outputs/preconditions/postconditions of the actions used to build the solution
  - Interactions among the previous constraints

Methods for Constructive problem solving

- **Propose and apply**: We start from an empty solution. Iteratively an action that allows to extend the current partial solution is selected until the complete solution is achieved
- **Least commitment**: We start from a complete initial solution. Iteratively an action that allow to modify the current solution is selected but guaranteeing that the action will impose the minimal constraints to future actions

Propose and apply

- The search is performed in the space of partial solutions
- We start with and empty or incomplete solution
- Each step extends the solution
- The best action is chosen each iteration
- The search is always inside the space of solutions

Propose and apply

- We need knowledge about:
  - Actions used to solve the problem
  - Constraints and relations among the components of the solution
  - Evaluation of the effects of an action on the solution
  - Evaluation of the quality of the solution
- The process to solve a problem can be done in different ways
  - Sequential resolution (Lots of knowledge are needed to be efficient)
  - Hierarchical problem decomposition (more efficient, but it requires to obtain problem decomposition operators)

Propose and apply: Algorithm

1. Initialize the goal: The initial solution is created
2. Propose an action: The possible actions that can be applied on the current solution are selected
3. Prune actions: Actions are pruned using global criteria
4. Evaluate actions: The effects of the actions on the solution are compared and evaluated
5. Select an action: The best action is chosen. If there is no best action backtracking is considered
6. Apply the action: The selected action is applied to the current solution
7. Evaluate the solution: If the current solution is the goal the process stops or another iteration is performed

Least commitment

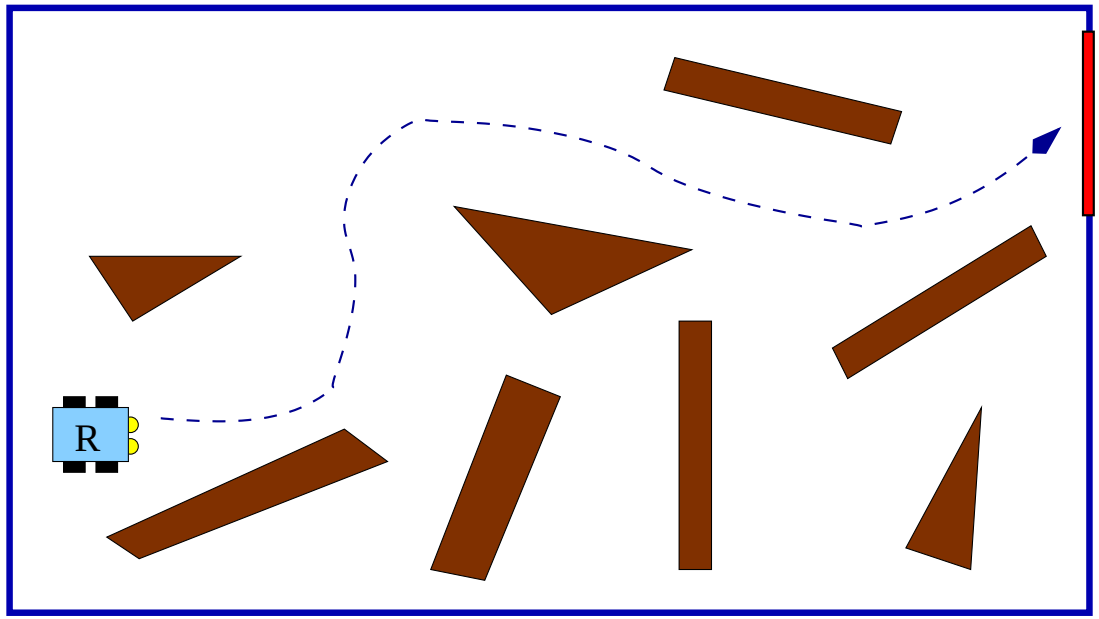
- The space of complete solutions is explored
- The search begins from a solution (it could be a no solution also)
- The solution is modified and corrected
- The choice of actions to apply is defined by the heuristic of least commitment: the minimal modification that impose less future constraints
- The search can pass from the space of solutions to the space of no solutions and viceversa

Least commitment: Algorithm

1. Start with a non optimal solution that satisfies the constraints if possible
2. Modify the solution with an action chosen using the heuristic of least commitment (action that imposes less constraints on the solution)
3. If the modification violates any constraint undo a previous action performing minimal modifications (not necessarily the last action)

Constructive problem solving: Example (1)

- We want to plan the best path for a robot inside a room
- Inside the room there are obstacles that have to be avoided
- We have a set of actions:
  - Move forward or backwards at a specific speed some distance
  - Rotate an specific number of degrees



Constructive problem solving: Example (2)

- Global constraints: Arrive to the exit door, minimal length and time path
- Constraints for the actions: Avoid collisions with obstacles or walls, maintain certain distance to obstacles to maneuver
- Evaluation of the actions:
  - Move: The nearer and faster the robot moves towards the goal the better
  - Rotate: The farther the trajectory of the robot is from the obstacles the better