

Lecture 2

Agents and Problem Solving

Tuesday, August 29, 2000

William H. Hsu

Department of Computing and Information Sciences, KSU

<http://www.kddresearch.org>

<http://www.cis.ksu.edu/~bhsu>

Reading for Next Class:

Chapter 3, Appendix A, Russell and Norvig

Handout, *Essentials of Artificial Intelligence*, M. Ginsberg

Lecture Outline

- **Today's Reading: Section 3.1-3.4, Russell and Norvig**
- **Intelligent Agent Frameworks**
 - Reactive
 - With state
 - Goal-based
 - Utility-based
- **Thursday: Problem Solving and Search**
 - **Background in combinatorial algorithms**
 - **Asymptotic analysis**
 - **Essentials of graph theory (definitions)**
 - **State space search handout (Winston)**
 - **Search handout (Ginsberg)**

Review: Intelligent Agent Framework

- **Agent: Definition**

- Any entity that perceives its environment through sensors and acts upon that environment through effectors
- Examples (class discussion): human, robotic, software agents

- **Perception**

- Signal from environment
- May exceed sensory capacity

- **Sensors**

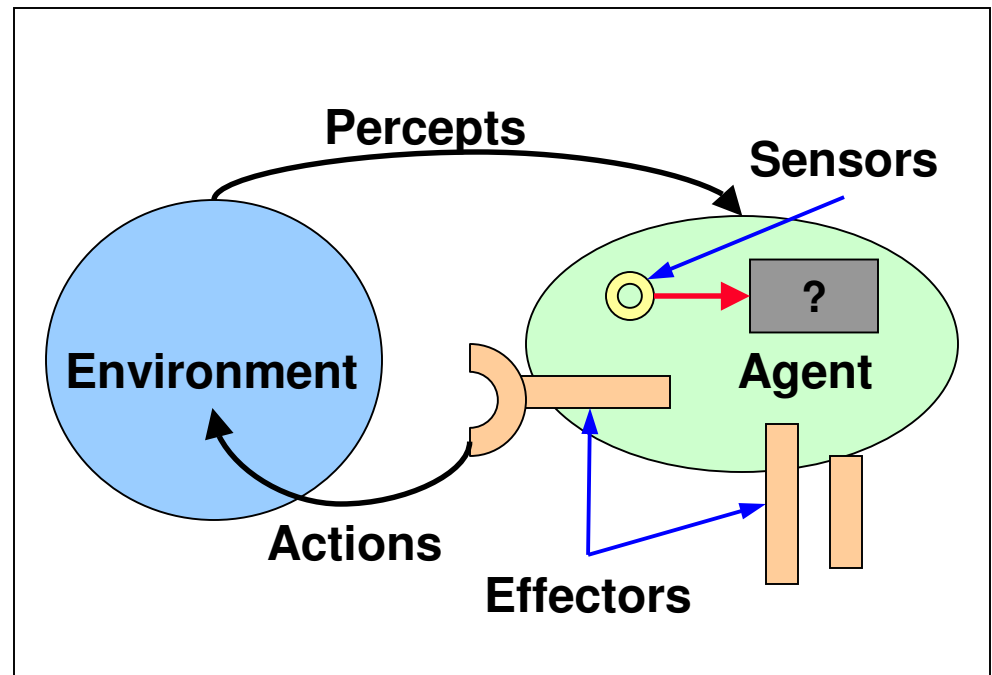
- Acquires percepts
- Possible limitations

- **Action**

- Attempts to affect environment
- Usually exceeds effector capacity

- **Effectors**

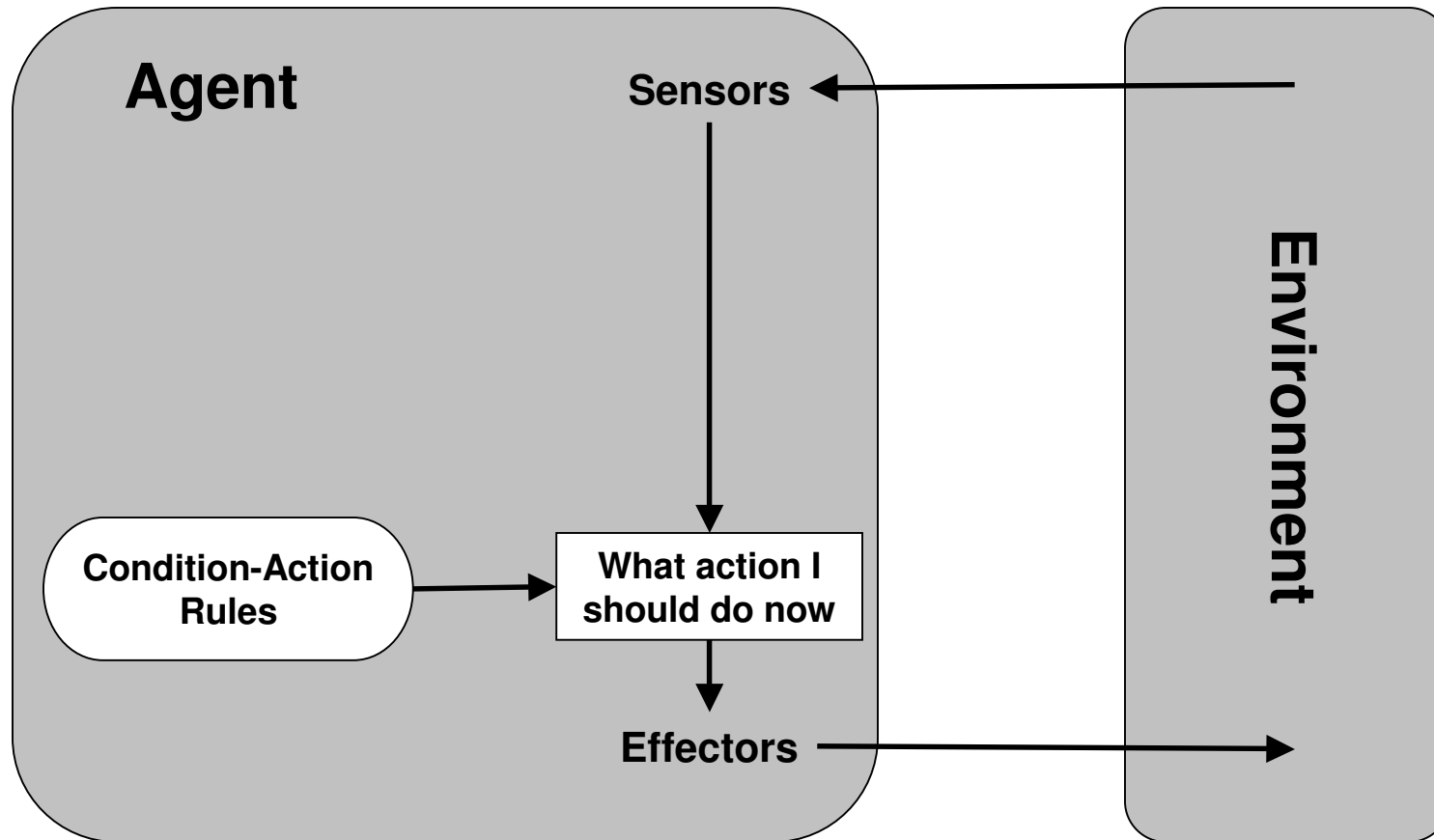
- Transmits actions
- Possible limitations



Review: Agent Programs

- **Software Agents**
 - Also known as (aka) software robots, softbots
 - Typically exist in very detailed, unlimited domains
 - Example
 - (Real-time) critiquing, automation of avionics, shipboard damage control
 - Indexing (spider), information retrieval (IR; e.g., web crawlers) agents
 - Plan recognition systems (computer security, fraud detection monitors)
 - See: Bradshaw (*Software Agents*)
- **Focus of This Course: Building IAs**
 - Generic skeleton agent: Figure 2.4, R&N
 - function *SkeletonAgent* (*percept*) returns action
 - static: *memory*, agent's memory of the world
 - *memory* ← *Update-Memory* (*memory*, *percept*)
 - *action* ← *Choose-Best-Action* (*memory*)
 - *memory* ← *Update-Memory* (*memory*, *action*)
 - return *action*

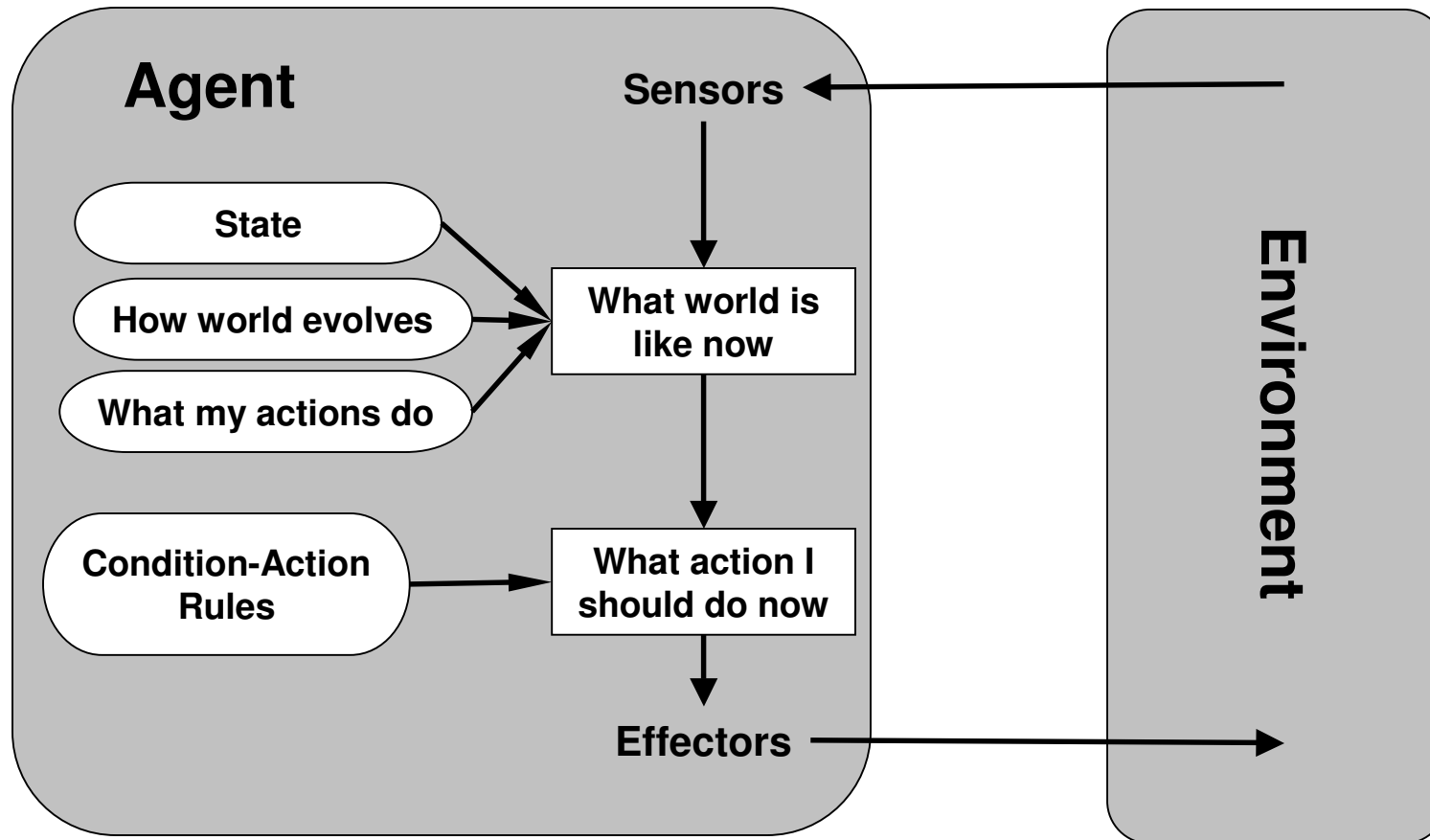
Agent Framework: Simple Reflex Agents [1]



Agent Framework: Simple Reflex Agents [2]

- **Implementation and Properties**
 - Instantiation of generic skeleton agent: Figure 2.8
 - function *SimpleReflexAgent* (*percept*) returns action
 - static: *rules*, set of condition-action rules
 - *state* ← *Interpret-Input* (*percept*)
 - *rule* ← *Rule-Match* (*state*, *rules*)
 - *action* ← *Rule-Action* {*rule*}
 - return *action*
- **Advantages**
 - Selection of best action based only on current state of world and rules
 - Simple, very efficient
 - *Sometimes* robust
- **Limitations and Disadvantages**
 - No memory (doesn't keep track of world)
 - Limits range of applicability

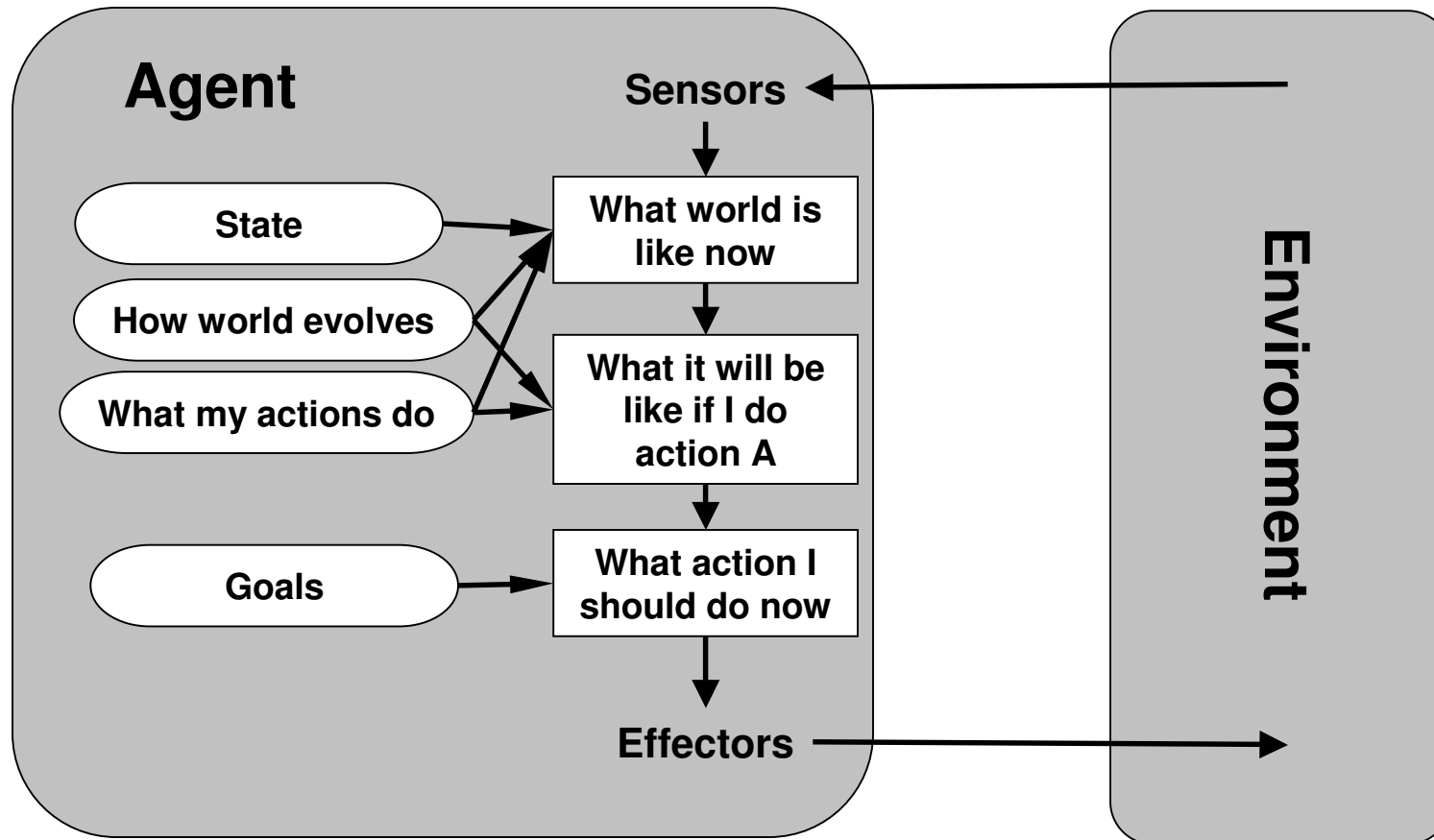
Agent Frameworks: (Reflex) Agents with State [1]



Agent Frameworks: (Reflex) Agents with State [2]

- **Implementation and Properties**
 - Instantiation of generic skeleton agent: Figure 2.10
 - function *ReflexAgentWithState* (*percept*) returns action
 - static: *state*, description of current world state;
rules, set of condition-action rules
 - *state* ← *Update-State* (*state*, *percept*)
 - *rule* ← *Rule-Match* (*state*, *rules*)
 - *action* ← *Rule-Action* {*rule*}
 - return *action*
- **Advantages**
 - Selection of best action based only on current state of world and rules
 - Able to reason over past states of world
 - Still efficient, *somewhat* more robust
- **Limitations and Disadvantages**
 - No way to express goals and preferences relative to goals
 - Still limited range of applicability

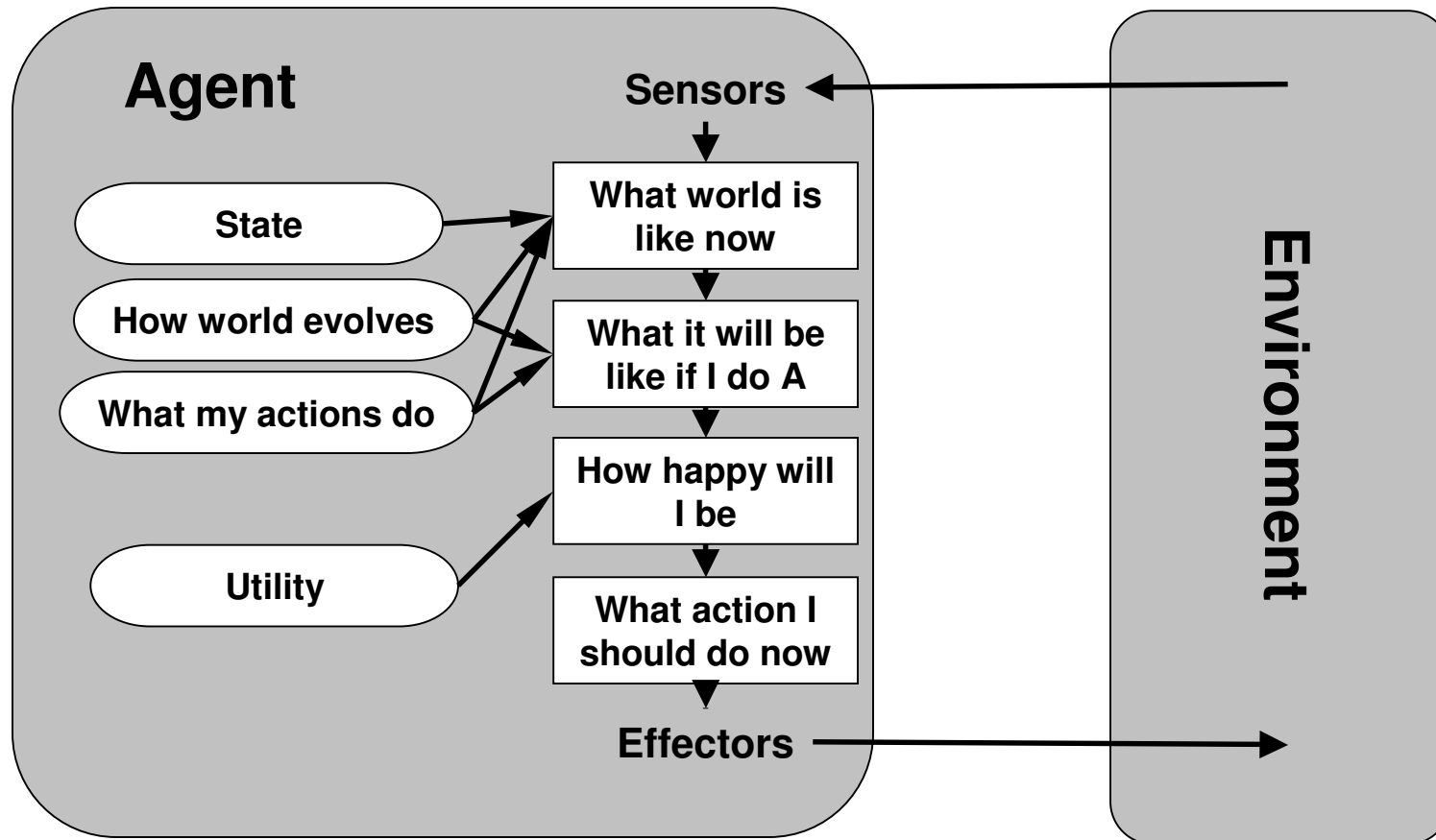
Agent Frameworks: Goal-Based Agents [1]



Agent Frameworks: Goal-Based Agents [2]

- **Implementation and Properties**
 - Instantiation of generic skeleton agent: Figure 2.11
 - Functional description
 - Chapter 13: classical planning
 - Requires more formal specification
- **Advantages**
 - Able to reason over goal, intermediate, and initial states
 - Basis: automated reasoning
 - One implementation: theorem proving (first-order logic)
 - Powerful representation language and inference mechanism
- **Limitations and Disadvantages**
 - Efficiency limitations: can't feasible solve many general problems
 - No way to express preferences

Agent Frameworks: Utility-Based Agents [1]



Agent Frameworks: Utility-Based Agents [2]

- **Implementation and Properties**
 - Instantiation of generic skeleton agent: Figure 2.8
 - function *SimpleReflexAgent* (*percept*) returns action
 - static: *rules*, set of condition-action rules
 - *state* ← *Interpret-Input* (*percept*)
 - *rule* ← *Rule-Match* (*state*, *rules*)
 - *action* ← *Rule-Action* (*rule*)
 - return *action*
- **Advantages**
 - Selection of best action based only on current state of world and rules
 - Simple, very efficient
 - *Sometimes* robust
- **Limitations and Disadvantages**
 - No memory (doesn't keep track of world)
 - Limits range of applicability

Looking Ahead: Search

- **Thursday's Reading: Sections 3.1-3.4, Russell and Norvig**
- **Thinking Exercises (Discussion in Next Class): 3.3 (a, b, e), 3.9**
- **Solving Problems by Searching**
 - Problem solving agents: design, specification, implementation
 - Specification components
 - Problems – formulating *well-defined* ones
 - Solutions – requirements, constraints
 - Measuring performance
- **Formulating Problems as (State Space) Search**
- **Example Search Problems**
 - Toy problems: 8-puzzle, 8-queens, cryptarithmic, toy robot worlds, constraints
 - Real-world problems: layout, scheduling
- **Data Structures Used in Search**
- **Next Tuesday: Uninformed Search Strategies**
 - State space search handout (Winston)
 - Search handouts (Ginsberg, Rich and Knight)

Problem-Solving Agents [1]: Preliminary Design

- **Justification**
 - Rational IAs: act to *reach* environment that maximizes performance measure
 - Need to formalize, operationalize this definition
- **Practical Issues**
 - Hard to find appropriate *sequence of states*
 - Difficult to translate into IA design
- **Goals**
 - Chapter 2, R&N: simplifies task of translating agent specification to formal design
 - First step in problem solving: formulation of goal(s) – “accept no substitutes”
 - Chapters 3-4, R&N: goal \equiv {world states | goal test is satisfied}
- **Problem Formulation**
 - Given: initial state, desired goal, specification of actions
 - Find: *achievable* sequence of states (actions) mapping from initial to goal state
- **Search**
 - Actions: cause transitions between world states (e.g., applying effectors)
 - Typically specified in terms of finding sequence of states (operators)

Problem-Solving Agents [2]: Specification

- **Input: Informal Objectives; Initial, Intermediate, Goal States; Actions**
- **Output**
 - Path from initial to goal state
 - Leads to design requirements for state space search problem
- **Logical Requirements**
 - States: representation of state of world (example: starting city, graph representation of Romanian map)
 - Operators: descriptors of possible actions (example: moving to adjacent city)
 - Goal test: state → boolean (example: at destination city?)
 - Path cost: *based on search, action costs* (example: number of edges traversed)
- **Operational Requirements**
 - Search algorithm to find path
 - Objective criterion: minimum cost (this and next 3 lectures)
- **Environment**
 - Agent can search in environment according to specifications
 - Sometimes has full state and action descriptors; *sometimes not!*

Problem-Solving Agents [3]: Implementation

- **function *Simple-Problem-Solving-Agent* (*p*: percept) returns *a*: action**
 - **inputs**: *p*, percept
 - **static**:
 - s*, action sequence (initially empty)
 - state*, description of current world state
 - g*, goal (initially null)
 - problem*, problem formulation
 - *state* ← *Update-State* (*state*, *p*)
 - if *s.Is-Empty*() then
 - *g* ← *Formulate-Goal* (*state*) // focus of today's class
 - *problem* ← *Formulate-Problem* (*state*, *g*) // focus of today's class
 - *s* ← *Search* (*problem*) // next 3 classes
 - *action* ← *Recommendation* (*s*, *state*)
 - *s* ← *Remainder* (*s*, *state*) // discussion: meaning?
 - return (*action*)
- **Chapters 3-4: Implementation of *Simple-Problem-Solving-Agent***

Terminology

- **Intelligent Agents**
- **Topics and Methodologies**
 - Knowledge representation
 - Search
 - Machine learning
 - Planning
- **Agent Frameworks**
 - State-based vs. memoryless agents
 - Reactivity vs. deliberation
 - From goals to preferences (utilities)
- **Applications**
 - Problem solving, optimization, scheduling, design
 - Decision support, data mining
 - Natural language processing, conversational and information retrieval agents
 - Pattern recognition and robot vision

Summary Points

- **Artificial Intelligence: Conceptual Definitions and Dichotomies**
 - Human cognitive modelling vs. rational inference
 - Cognition (thought processes) versus behavior (performance)
 - Some viewpoints on defining intelligence
- **Agent Frameworks**
 - Reactivity vs. state
 - From goals to preferences (utilities)
- **Applications and Automation Case Studies**
 - Search: game-playing systems, problem solvers
 - Planning, design, scheduling systems
 - Control and optimization systems
 - Machine learning: pattern recognition, data mining (business decision support)
- **Things to Check Out Online**
 - Resources page:
<http://www.kddresearch.org/Courses/Fall-2001/CIS730/Resources>
 - Yahoo! Group discussions: <http://groups.yahoo.com/group/ksu-cis730-fall2001>
 - Suggested project topics (to be posted online), resources