Representation Using Logic

School of Electronic and Computer Engineering
Peking University

Wang Wenmin

Contents

Procedural vs. Declarative Approaches 过程性与陈述性方法

- Procedural approaches 过程性方法
  - use procedural languages, such as 采用过程性语言，例如
    - C/C++/C#/Java,
    - Lisp,
    - Python.
- Declarative approaches 陈述性方法
  - use declarative languages, such as 采用陈述性语言，例如
    - Propositional logic, 命题逻辑,
    - First-order logic, 一阶逻辑,
    - Temporal logic. 时序逻辑。

Five Different Logics 五种不同的逻辑

Formal Language 形式语言	Ontological Commitment 本体论约定	Epistemological Commitment 认识论约定
Propositional logic 命题逻辑	facts 事实	true/false/unknown 真/假/未知
First-order logic 一阶逻辑	facts, objects, relations 事实、对象、关系	true/false/unknown 真/假/未知
Temporal logic 时序逻辑	facts, objects, relations, times 事实、对象、关系、时间	true/false/unknown 真/假/未知
Probability theory 概率论	Facts 事实	degree of belief $\in [0, 1]$ 可信度
Fuzzy logic 模糊逻辑	facts with degree of truth $\in [0, 1]$ 事实具有真实度	known interval value 已知区间值

Logical Symbols 逻辑符号

Category 类别	Symbol 符号	Mean 含义	
Connectives 连接词	$\neg$	not	非
	$\wedge$	and	与
	$\vee$	or	或
	$\Rightarrow$	implies	蕴含
	$\Leftrightarrow$	if and only if ($\equiv$)	当且仅当
	$\models$	entailment	导出
	$\not\models$		
Quantifiers 限量词	$\forall$	for all	所有
	$\exists$	there exist	存在
Equality 等量词	$=$	equal	等于

Propositional Logic vs. First-order Logic 命题逻辑与一阶逻辑

- Propositional logic: 命题逻辑：

  also known as propositional calculus,

  亦被称为命题演算

  - use of logical connectives, deal with simple declarative propositions (if they are true or false).

    使用逻辑连接词，用于处理简单的陈述性命题。

- First-order logic: 一阶逻辑:

  also known as first-order predicate calculus,

  亦被称为一阶谓词演算，

  - additionally, use quantifiers, equality, and use predicates (often associated with sets).

    此外，还使用限量词、等量词、以及谓词（通常与集合相关联）。

Propositional Logic Syntax with BNF 用BNF表述的命题逻辑语法

$$
\begin{array}{rcl}
Sentence & \rightarrow & AtomicSentence \mid ComplexSentence \\
AtomicSentence & \rightarrow & True \mid False \mid P \mid Q \mid R \mid \ldots \\
ComplexSentence & \rightarrow & <\ Sentence\ > \mid [\ Sentence\ ] \\
 & \mid & \neg\ Sentence \\
 & \mid & Sentence\ \wedge\ Sentence \\
 & \mid & Sentence\ \vee\ Sentence \\
 & \mid & Sentence \Rightarrow Sentence \\
 & \mid & Sentence \Leftrightarrow Sentence \\
OPERATOR\ PRECEDENCE & : & \neg,\ \wedge,\ \vee,\ \Rightarrow,\ \Leftrightarrow
\end{array}
$$

BNF: Backus–Naur Form
巴科斯-诺尔范式

First-Order Logic Syntax with BNF 用BNF表述的一阶逻辑的语法

$$
\begin{array}{rcl}
\textit{Sentence} & \rightarrow & \textit{AtomicSentence} \mid \textit{ComplexSentence} \\
\textit{AtomicSentence} & \rightarrow & \textit{Predicate} \mid \textit{Predicate}(\textit{Term}, \ldots) \mid \textit{Term} = \textit{Term} \\
\textit{ComplexSentence} & \rightarrow & <\textit{Sentence}> \mid [\,\textit{Sentence}\,] \mid \neg\,\textit{Sentence} \mid \textit{Sentence} \wedge \textit{Sentence} \\
 & \mid & \textit{Sentence} \vee \textit{Sentence} \mid \textit{Sentence} \Rightarrow \textit{Sentence} \mid \textit{Sentence} \Leftrightarrow \textit{Sentence} \\
 & \mid & \textit{Quantifier Variable}, \ldots\ \textit{Sentence} \\
\textit{Term} & \rightarrow & \textit{Function}(\textit{Term}, \ldots) \mid \textit{Constant} \mid \textit{Variable} \\
\textit{Quantifier} & \rightarrow & \forall \mid \exists \\
\textit{Constant} & \rightarrow & A \mid X_1 \mid \textit{John} \mid \ldots \\
\textit{Variable} & \rightarrow & a \mid x \mid s \mid \ldots \\
\textit{Predicate} & \rightarrow & \textit{True} \mid \textit{False} \mid \textit{After} \mid \textit{Loves} \mid \textit{Raining} \mid \ldots \\
\textit{Function} & \rightarrow & \textit{Mother} \mid \textit{LeftLeg} \mid \ldots \\
\textit{OPERATOR PRECEDENCE} & : & \neg, =, \wedge, \vee, \Rightarrow, \Leftrightarrow
\end{array}
$$

Formation Rules in First Order Logic 一阶逻辑的形式规则

- The formation rules define
  该形式规则定义
  - terms, and
    项，以及
  - formulas.
    公式
- The formation rules can be used to write a formal grammar for terms and formulas.
  该形式规则可以用于书写项和公式的形式文法。
- Formation rules are generally context-free, i.e.,
  形式规则通常是上下文无关的，即
  - each production has a single symbol on the left side.
    每个产生式左侧有一个单一的符号。

Formation Rules of First Order Logic: Terms 一阶逻辑的形式规则：项

- **Rule1: Variables** 规则1：变量

  Any variable is a term.

  任何变量都是一个项。

- **Rule2: Constants** 规则2：常数

  Any constant is also a term.

  任何常数也都是一个项。

- **Rule3: Functions** 规则3：函数

  Any expression $f(t_1,...,t_n)$ of n arguments is a term, where each argument t_i is a term, and f is a function symbol of valence n. In particular, symbols denoting individual constants are 0-ary function symbols, and are thus terms.

  任何n个参数的表达式f(t1,...,tn)都是一个项，其中每个参数ti是一个项，并且f是一个价n的函数符号。尤其是，表示个体常量的符号是0元函数符号，因此也是一个项。

Formation Rules of First Order Logic: Formulas 一阶逻辑的形式规则：公式

- **Predicate symboles.** If P is an *n-ary* predicate symbol and $t_1,\dots t_n$ are terms, then $P(t_1,\dots t_n)$ is a formula.

  谓词符号：若P是一个n元谓词符号并且$t_1,\dots t_n$是项，则 $P(t_1,\dots t_n)$ 是一个公式。

- **Equality.** If the equality symbol is considered part of logic, and t_1 and t_2 are terms, then $t_1=t_2$ is a formula.

  等量：若等量符号被认为是逻辑的一部分，并且 t_1和 t_2是项，则 $t_1=t_2$ 是一个公式。

- **Negation.** If φ is a formula, then $\neg\varphi$ is a formula.

  否定：若φ 是一个公式，则 $\neg\varphi$ 是一个公式。

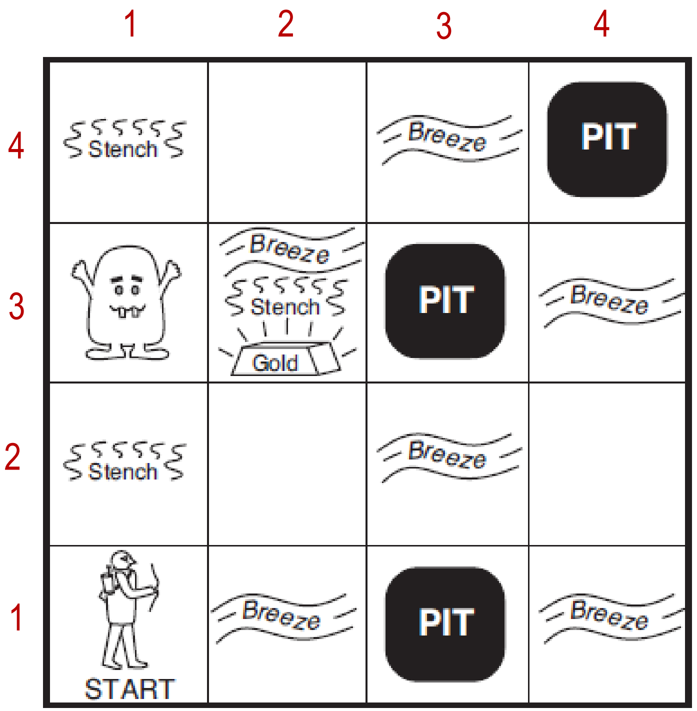
- **Binary connectives.** If φ and ψ are formulas, then $(\varphi \Rightarrow \psi)$ is a formula. Similar rules apply to other binary logical connectives.

  二元连接：若φ 和 ψ 是公式，则 $(\varphi \Rightarrow \psi)$是一个公式。类似的规则可用于其他二元逻辑连接。

- **Quantifiers.** If φ is a formula and x is a variable, then $\forall x\varphi$ and $\exists x\varphi$ are formulas.

  限量：若φ 是一个公式并且x是一个变量，则 $\forall x\varphi$和 $\exists x\varphi$ 是公式。

Example: Wumpus world 魔兽世界

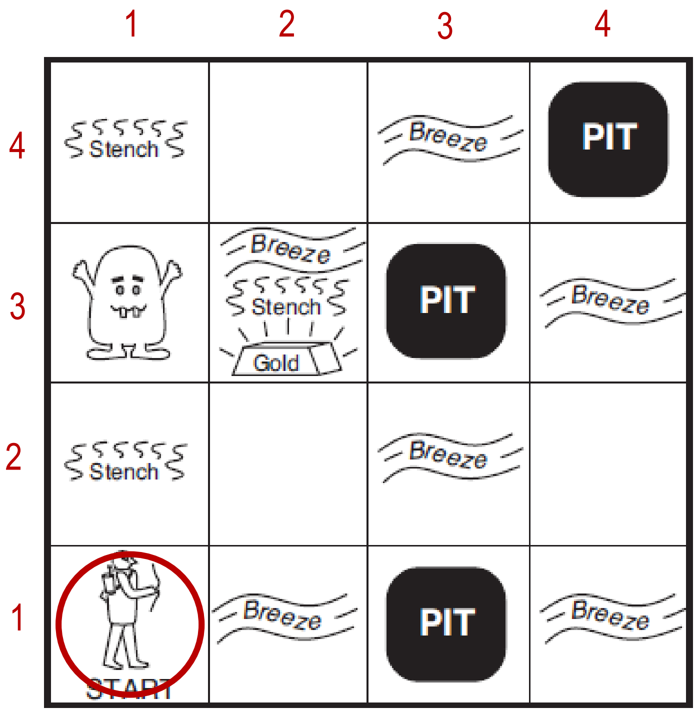


- Environment: 环境：
  - Agent 智能体，
  - Wumpus 魔兽，
  - Gold 黄金，
  - Pit (probability 0.2) 陷阱（概率0.2）

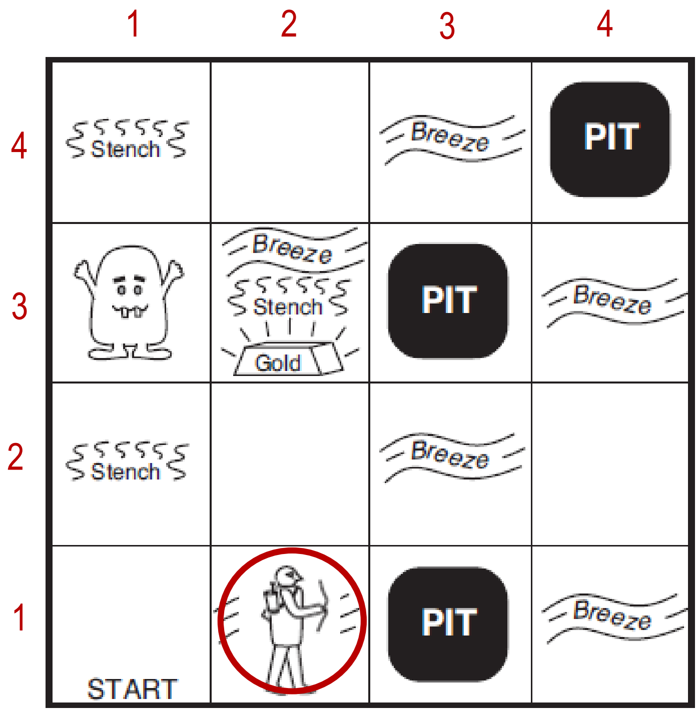
- Performance measure: 性能指标：
  - +1000: gold, 黄金
  - -1000: death 死亡 (enters a PIT or a wumpus),
  - -1: per step, 每一步
  - -10: using the arrow. 用箭
- Actuators: 执行器：
  - *Turn Left*, *Turn Right*, *Forward*, 向左、向右、前进
  - *Shoot*: to fire an arrow, 射击：发射一只箭
  - *Grab*: to pick up gold, 抓住：拾起黄金
  - *Climb*: to climb out of cave. 攀爬：攀越陷阱
- Sensors: 感受器：
  - *Stench* 臭气，
  - *Breeze* 微风，
  - *Glitter* 闪光，
  - *Bump* 碰撞，
  - *Scream* 尖叫.

Percept [*Stench*, *Breeze*, *Glitter*, *Bump*, *Scream*]

Example: Wumpus world 魔兽世界



(a)

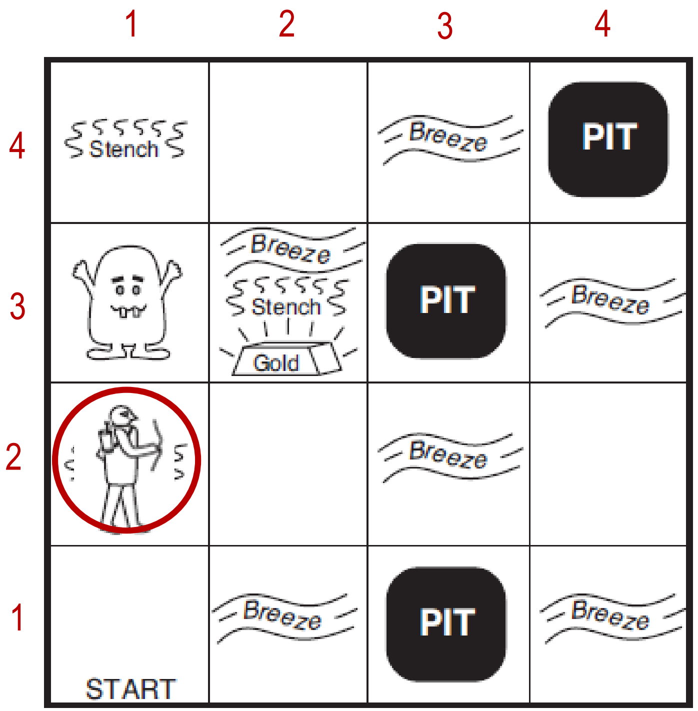


(b)

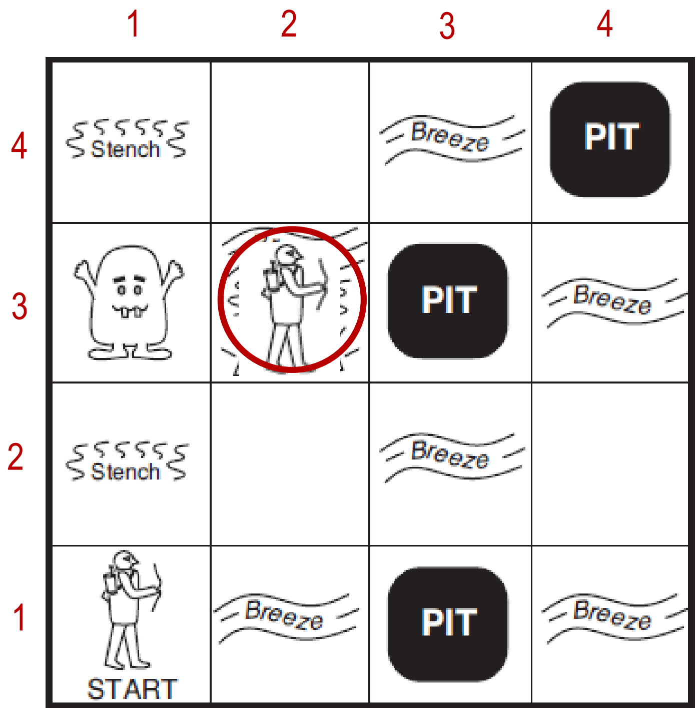
The first step taken by the agent. (a) Initial situation, after *Percept* [*None*, *None*, *None*, *None*, *None*].
(b) After one move, with *Percept* [*None*, *Breeze*, *None*, *None*, *None*].

智能体所采取的第一步。(a) 初始状态，在*Percept* [*None*, *None*, *None*, *None*, *None*]之后。
(b) 移动一步后，具有*Percept* [*None*, *Breeze*, *None*, *None*, *None*]。

Example: Wumpus world 魔兽世界



(c)



(d)

Two later stages in the progress of the agent. (c) After third move, with *Percept* [*Stench*, *None*, *None*, *None*, *None*].
(d) After fifth move, with *Percept* [*Stench*, *Breeze*, *Glitter*, *None*, *None*].

智能体进展的两个后期阶段。(c) 移动第三步后，具有*Percept* [*Stench*, *None*, *None*, *None*, *None*]。
(d) 移动第四步后，具有*Percept* [*Stench*, *Breeze*, *Glitter*, *None*, *None*]。

Using First-Order Logic for Wumpus World 用一阶逻辑描述魔兽世界

- Percept sentence 感知语句

$$Percept([Stench, Breeze, Glitter, None, None], 5)$$
$$\forall t, s, g, m, c \;\; Percept([s, Breeze, g, m, c], t) \Rightarrow Breeze(t)$$
$$\forall t, s, b, m, c \;\; Percept([s, b, Glitter, m, c], t) \Rightarrow Glitter(t)$$

- Action sentence 动作语句

$$Turn(Right), Turn(Left), Forward\,, Shoot\,, Grab, Climb.$$

- Query sentence 查询语句 (To determine which is best, 确定那个是最好的)

$$\textsc{AskVars}(\exists a, BestAction(a, 5))$$

Prolog Language Prolog语言

- Prolog language has its roots in first-order logic.
  Prolog语言起源于一阶逻辑。
- Prolog is a general purpose logic programming language, has been used for theorem proving, expert systems, natural language processing, and so on.
  Prolog是一种通用的逻辑编程语言，已经被用于定理证明、专家系统、自然语言处理，等等。
- Unlike other programming languages, Prolog is declarative: the program logic is expressed in terms of relations, represented as facts and rules.
  不同于其它编程语言，Prolog是陈述性的：程序逻辑由关系来表达，表示为事实与规则。

```
likes(bill, car).
animal(X) :- cat(X).
bird(X) :- animal(X), has(X, feather).
```

Thank you for your attention!

AI