南京大学 人工智能学院
SCHOOL OF ARTIFICIAL INTELLIGENCE, NANJING UNIVERSITY

第三部分：通信与合作

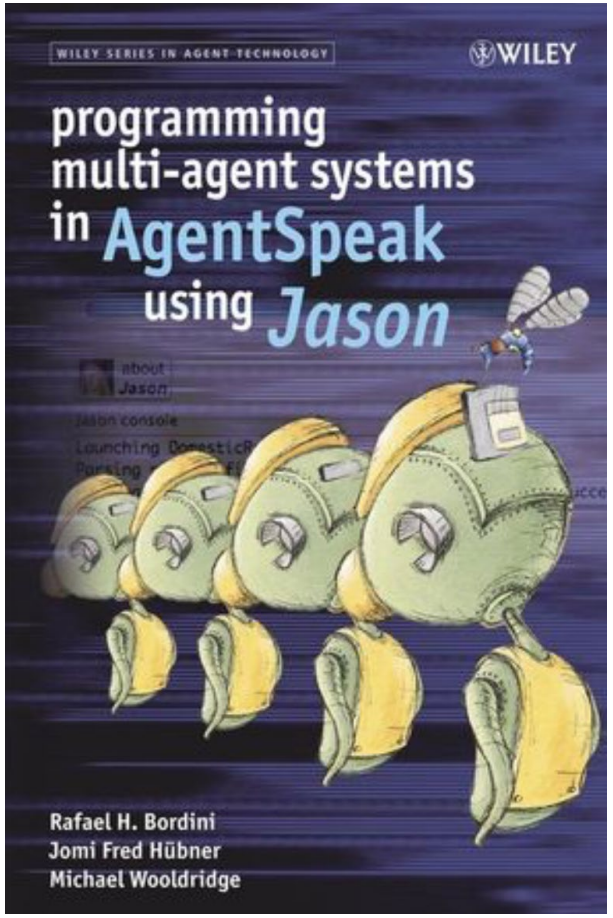
章宗长

2023年4月4日

内容安排

实践：使用Jason解释器的多Agent编程

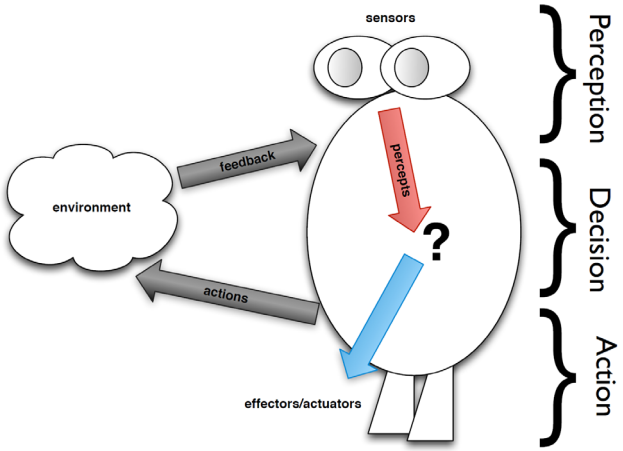
- **多Agent编程语言概览**
- Jason语法简介
- Jason中的通信与交互
- Jason编程实例 – 家政机器人
- Jason编程实例 – 合同网协议



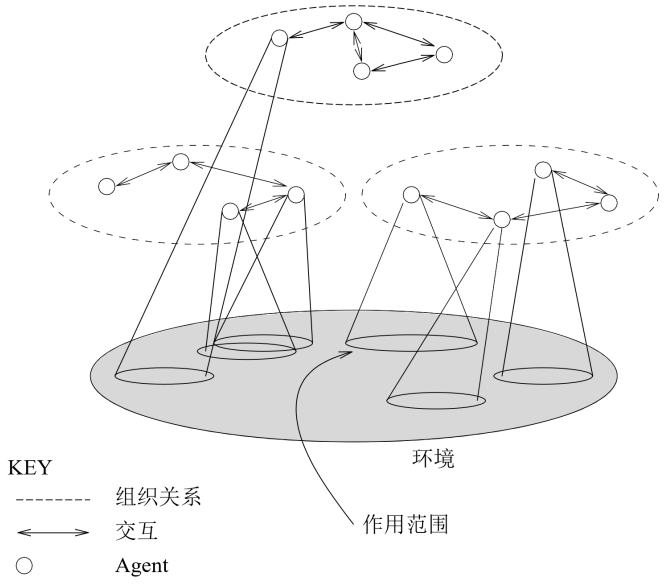
智能Agent & 多Agent系统

- 智能Agent应该具有的属性：
  - 自治性
  - 预动性
  - 反应性
  - 社会能力

- 多Agent系统包含一些Agent：
  - 通过通信相互交互
  - 有不同的“作用范围”
  - 影响的范围可能重叠 … …



Agent与环境



多Agent系统的标准结构

如何设计多Agent编程语言？

- 要设计的语言需要满足Agent和多Agent系统的各种特性，应当具有下面这些性质：
  - 这门语言需要支持目标层面的委托（delegation）
  - 这门语言能够提供目标导向型的问题解决方法
  - 这门语言能够产生反应式系统
  - 这门语言能够集成目标导向式和反应式行为
  - 这门语言需要支持知识层面的通信与合作行为
- AgentSpeak：具有这些性质的多Agent编程语言
- Jason：AgentSpeak语言的实现/解释器/开发环境

AgentSpeak – Hello World

```
started.

+started <- .print("Hello World!").
```

Jason中定义好的内置动作

符号‘.’：一行代码的结束符，类似C或Java中的分号

触发条件

规划体

- 上面的两行代码定义了一个Agent：
  - 初始信念：started
  - 规划（plan）：当你相信started，打印文本“Hello World”
    - 符号‘+started’：当Agent获得信念“started”
    - 当满足触发条件时，规划被触发（激活）
    - 规划体包含一个动作：显示文本‘Hello World!’

AgentSpeak – 运行Hello World

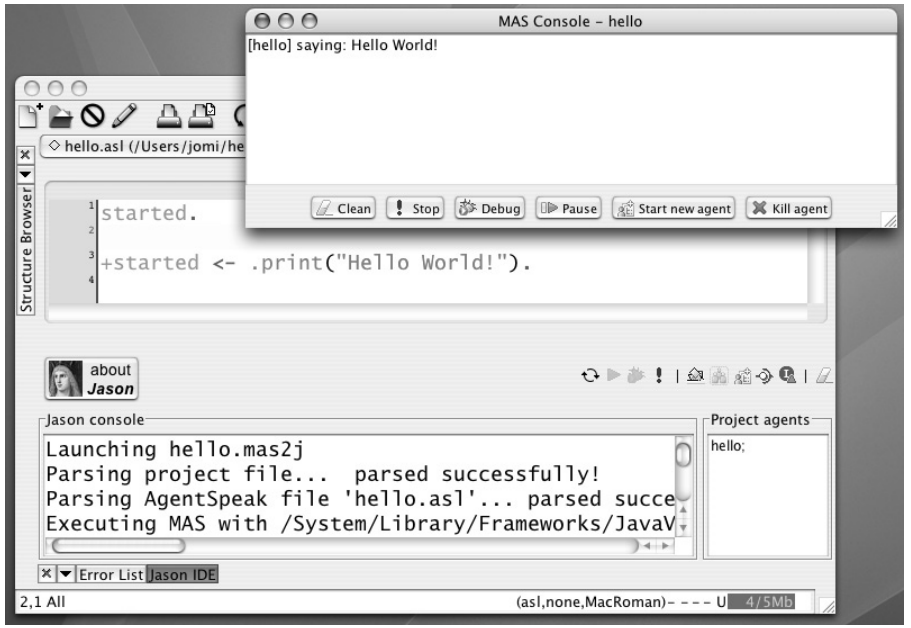
- 通常将这类定义了一个Agent的代码保存在后缀名为.asl 的AgentSpeak语言文件中，如hello.asl

```
started.

+started <- .print("Hello World!").
```

- 在hello.mas2j文件中定义多Agent系统的配置文件：

```
MAS hello {
    infrastructure:
        Centralised
    agents:
        hello;
}
```



AgentSpeak – 阶乘计算实例

```
fact(0,1).

+fact(X,Y)
   :  X < 5
   <- +fact(X+1, (X+1)*Y).

+fact(X,Y)
   :  X == 5
   <- .print("fact 5 == ", Y).
```

规划的上下文

- 信念fact(X, Y)表示Agent相信X的阶乘为Y
- 在定义Agent的规划时，我们可以在冒号后添加对应的条件，表示条件满足时才执行该规划
- 运行结果：
  - `[fact] saying: fact 5 == 120`

AgentSpeak – 阶乘计算实例 (2)

```
!print_fact(5).

+!print_fact(N)
   <- !fact(N,F);
      .print("Factorial of ", N, " is ", F).

+!fact(N,1) : N == 0.

+!fact(N,F) : N > 0
   <- !fact(N-1,F1);
      F = F1 * N.
```

- !print_fact(5) 是Agent的初始目标，其与初始信念的区分在于以“!”开头
- 为了实现 !print_fact(5) 的目标，Agent需要先完成对应的目标!fact(N, F)，然后输出N的阶乘F
- 完成 !fact(N, F) 目标需要完成 !fact(N-1, F1) 目标

实践：使用Jason解释器的多Agent编程

- 多Agent编程语言概览
- **Jason语法简介**
- Jason中的通信与交互
- Jason编程实例 – 家政机器人
- Jason编程实例 – 合同网协议

Jason

- Jason是一种基于BDI体系结构、支持扩展的AgentSpeak语言的解释器
  - Jason提供了支持自定义功能的多Agent系统开发环境
  - 我们也称能够在Jason解释器中运行的语言为Jason语言
- Jason语言中的三个主要结构：
  - 信念（ belief ）
  - 目标 （ goal ）
  - 规划 （ plan ）
- Jason解释器的获取
  - GitHub源码：https://github.com/jason-lang/jason
  - Sourceforge下载：https://sourceforge.net/projects/jason

信念

- Agent有一个信念库（belief base）
- 信念反映了Agent的某些认知（不见得是真实情况），常常以逻辑编程的形式表示：

```
tall(john).
```

对象/个体john有tall的属性

```
likes(john, music).
```

John喜欢音乐

- 在信念后添加标注（annotation）可以更好的表示某些具体的信念，如下面的语句：

```
busy(john)[expires(autumn)].
```

John是忙碌的，但是一旦秋天到来，他就不再忙碌了

- 可以用source标注记录信息来源

```
likes(john,music)[source(john)].
```

“John喜欢音乐”这条信念的消息来源是John

3种类型的Source标注

- 感知信息 [source(percept)]
  - 如果Agent通过感知环境获得了一组信念，则它们被标注为percept
- 通信 [source(agentID)]
  - 如果Agent通过通信获得了一组信念，则它们被标注为分享该消息的发送者的ID
- 读心记事 [source(self)]
  - 如果Agent自身推理出了一组信念，则它们被标注为self

如果没有source标注，则默认为读心记事（mental notes）

强否定 & 弱否定

- 在语句前添加~以表示强否定，如

```
~colour(box1,white)
```

box1的颜色不是白色

- 对应地，在语句前加 not 表示弱否定
  - 弱否定表示Agent对该语句没有信念，而强否定表示Agent认定该语句一定为假

- 例子：Agent的信念库

```
colour(box1,blue)[source(bob)].
~colour(box1,white)[source(john)].
colour(box1,red)[source(percept)].
colourblind(bob)[source(self),degOfCert(0.7)].
lier(bob)[source(self),degOfCert(0.2)].
```

规则

- 基于信念的逻辑表示，可以定义新的规则 (rule)：
  - a :- b　如果b成立，则a成立

```
likely_colour(C,B)
   :- colour(C,B)[source(S)] & (S == self | S == percept).
```

如果盒子B的颜色C是Agent演绎出的或者是Agent感知到的，则C是盒子B最可能的颜色

```
likely_colour(C,B)
   :- colour(C,B)[degOfCert(D1)] &
      not (colour(_,B)[degOfCert(D2)] & D2 > D1) &
      not ~colour(C,B).
```

如果颜色C是有最高确定程度的颜色，并且没有强证据表明盒子B的颜色不是C，则C是盒子B最可能的颜色

基于信念和规则的推理

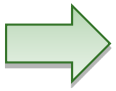
- 基于一组信念和规则，可以推理出新的信念

一组规则

- grandparent(X, Z) :- parent(X, Y) & parent (Y, Z).
- child(X, Y) :- parent(Y, X).
- son(X, Y) :- child(X, Y) & male(X).
- daughter(X, Y) :- child(X, Y) & female(X).

一组信念

- parent(eric, hilary)
- parent(hilary, jane)
- parent(hilary, david)
- female(jane)
- male(david)



推理出的一组新的信念

- grandparent(eric, jane)
- child(hilary, eric)
- child(jane, hilary)
- child(david, hilary)
- son(david, hilary)
- daughter(jane, hilary)

这是一个**演绎推理**的例子
Jason通过**规划**实现**实用推理**

目标

- 目标表示Agent想要得到具有某些属性的世界状态
- 有两类目标：
  - 成就目标（achievement goal）
    - 是Agent想要完成的
    - 目前Agent不认为这个目标是完成的（即它不在Agent的当前信念中）
    - 通常在语句前加“!”，如 `!own(house)`
  - 测试目标（test goal）
    - 类似于Prolog语言中的目标（或查询）
    - 通常在语句前加“?”，如 `?bank_balance(BB)`，其中BB为变量，表示需要确定变量BB的值使得语句为Agent的当前信念

规划

- 规划是一份Agent可执行动作的清单，表示Agent如何完成某个指定的目标

- 规划应当包含下面**三个部分**：

```
triggering event: context <- body
```

  - 触发事件（triggering event）：该规划用于应对何种事件
  - 上下文（context）：在何种上下文时可使用该规划
  - 规划体（body）：当事件触发且上下文满足时，应当执行的内容

- 规划的语法也可以包含一个可选的标签：
  - `@label triggering event: context <- body`

触发事件

- 触发事件：当Agent的信念或目标变化时发生
  - Agent通过执行规划来对事件进行反应
- 触发事件包括以下几种类型：

	Notation	Name
信念变化	$+l$	Belief addition
	$-l$	Belief deletion
目标变化	$+!l$	Achievement-goal addition
	$-!l$	Achievement-goal deletion
	$+?l$	Test-goal addition
	$-?l$	Test-goal deletion

上下文

- 在规划中，上下文的文字类型：

Syntax	Meaning
l	The agent believes *l* is true
~*l*	The agent believes *l* is false
not *l*	The agent does not believe *l* is true
not ~*l*	The agent does not believe *l* is false

- 上下文的例子：

```
+!prepare(Something)
    :  number_of_people(N) & stock(Something,S) & S > N
    <-  ....
```

上下文：库存中有足够N个人吃的食物

```
+!buy(Something)
    :  not ~legal(Something) & price(Something,P)
       & bank_balance(B) & B > P
    <-  ....
```

上下文：银行存款中有比这件商品的价格更多的钱

动作

- 动作是体现在规划中的Agent与环境交互的手段
  - 动作与信念的语句在语法表示上没有区别，它们的区分体现在出现位置的不同：
    - 信念语句出现在规划的触发事件、上下文部分
    - 动作语句出现在规划体部分
  - 动作语句是**无歧义的**，即语句中的任何变量在执行时都必须有确定值
- 具有前缀“.”的动作为Jason中的系统内置动作
  - 如 `.print`，`.send` 等等

规划的实例

```
+!at(Coordinates): not at(Coordinates)
                    & ~unsafe_path(Coordinates)
                <- move_towards(Coordinates);
                    !at(Coordinates).
```

- 触发事件：添加目标 +!at(Coordinates)
- 上下文：Agent当前不在坐标Coordinates且确信到坐标Coordinates的道路安全
- 规划体：向坐标Coordinates移动，随后完成目标!at(Coordinates)

实践：使用Jason解释器的多Agent编程

- 多Agent编程语言概览
- Jason语法简介
- **Jason中的通信与交互**
- Jason编程实例 – 家政机器人
- Jason编程实例 – 合同网协议

AgentSpeak通信结构

- 在AgentSpeak中，Agent的通信结构相较于KQML/ACL更为简单，其信息结构为

<sender, performative, content>

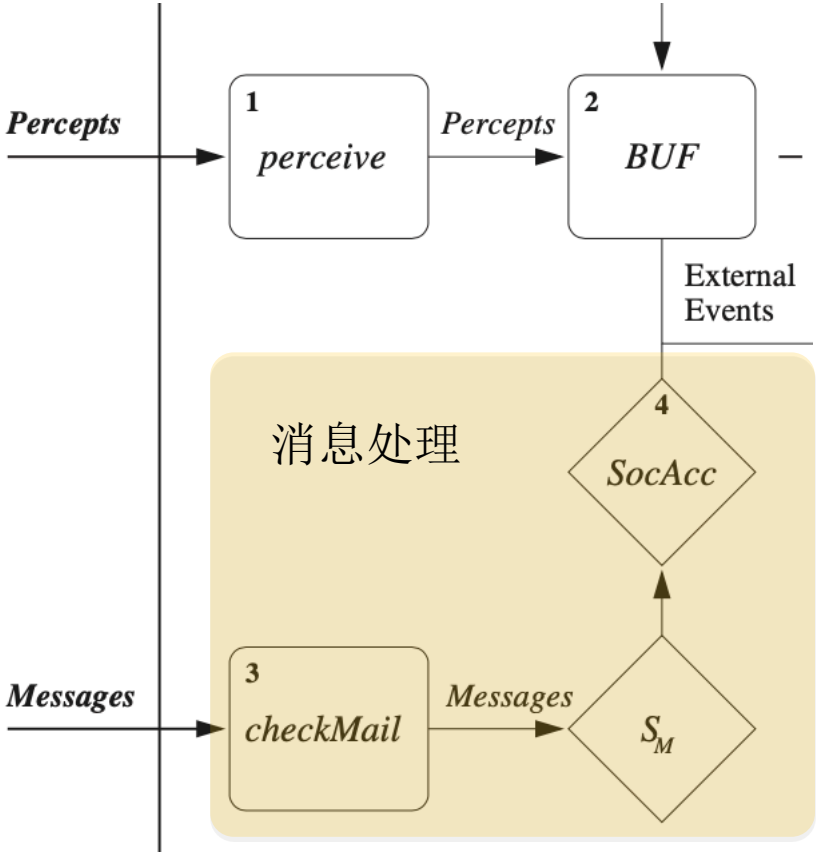
  - 发送者（sender）：消息的发送者
    - 通常为发送消息的AgentID
  - 语用词（performative）：表示发送者发送该消息期望达成的目标
    - 例如，tell, achieve, askOne, … …
  - 内容（content）：遵循AgentSpeak语法的消息内容
    - 消息的内容根据语用词的不同可能有所区别

Jason中的消息

- 在Jason中，消息通过预定义的内置动作产生，常用的发送消息的动作有以下两种：
  - .send(receiver, performative, content)
  - .broadcast(performative, content)
- 动作.send将消息发送给指定的Agent
  - 使用时需要指定对应的Agent，或是Agent的列表
  - 消息由接收者、语用词和内容组成
- 动作.broadcast将消息发送给注册到该多Agent系统中的所有Agent
  - 消息由语用词和内容组成

Jason的消息处理

- 在Jason的内部运行循环中，消息被投放到每个Agent的MailBox中，由checkMail函数接收
- 每次推理过程，Agent会选择一条消息进行处理
  - 由消息选择函数S_M选出下一条要处理的消息
  - 由过程SocAcc决定是否应该拒绝消息
- 如果消息被最终接受，则Jason解释器会将消息内容编译为其实际语义
  - 如：产生新的目标、信念等等



Jason中的语用词

- 共享信念（信息交换型）
  - tell/untell: 发送者想要接收者相信/不相信消息内容，且这一信念是发送者本身相信的
- 共享规划（慎思型）
  - tellHow/untellHow: 发送者请求接收者（不）接受内容中的规划
  - askHow: 发送者想要知道接收者关于指定事件的规划
- 委派一个成就目标（目标委派型）
  - achieve/unachieve: 发送者希望接收者完成/撤销消息内容中的目标
- 委派一个测试目标（信息查询型）
  - askOne/askAll: 发送者想要知道接收者是否知道消息内容中的一个答案/所有答案

Jason中的语用词 – tell/untell

Information Exchange

Cycle #	sender (s) actions	recipient (r) belief base	recipient (r) events
1	.send (r, tell, open(left_door))		
2		open(left_door)[source(s)]	〈+open(left_door) [source(s)],⊤〉
3	.send (r, untell, open(left_door))		
4			〈-open(left_door) [source(s)],⊤〉

- 事件-意图元组 <event, intent>
  - intent表示产生事件event的意图
- 对于通信产生的事件而言，没有与该事件对应的意图，在此使用T代替

<+/- open(left_door)[source(s)], T>

Jason中的语用词 – achieve/unachieve

Goal Delegation

Cycle #	sender (s) actions	recipient (r) intentions	recipient (r) events
1	.send (r, achieve, open(left_door))		
2			〈+!open(left_door) [source(s)],⊤〉
3		!open(left_door)[source(s)]	
3	.send (r, unachieve, open(left_door))	!open(left_door)[source(s)]	
4		<<< intention has been removed >>>	

- 语用词achieve将为接收者添加对应目标
- 语用词unachieve将删除接收者目标列表中的对应目标

Jason中的语用词 – askOne/askAll

- 假设接收者的信念库中包含两条信念：

r's belief base
open(left_door)
open(right_door)

  - askOne仅会提供一个可能的答案，而askAll会提供所有答案

Information Seeking

Cycle #	sender (s) actions	recipient (r) actions	sender (s) events
1	.send (r, askOne, open(Door))		
2		.send(s, tell, open(left_door))	
3			〈+open(left_door)[source(r)],⊤〉
4	.send (r, askAll, open(Door))		
5		.send(s, tell, [open(left_door), open(right_door)])	
6			〈+open(left_door)[source(r)],⊤〉 〈+open(right_door)[source(r)],⊤〉

Jason中的语用词 – tell/untell/askHow

- 语用词tellHow：向接收者的规划库中添加一条规划
  - 要添加的规划具有一个以@p开头的标签

```
.send(r, tellHow,
  “@pOD +!open(Door): not locked(Door)
  <- turn_handle(Door); push(Door); ?open(Door).”)
```

- 语用词untellHow：从接收者的规划库中删除对应标签的规划
  - `.send(r, untellHow, “@pOD”)`
- 语用词askHow：请求接收者提供所有关于给定事件的规划
  - `.send(r, askHow, “+!open(Door)”)`

Jason中的语用词 – 自定义语用词

- Jason允许Agent的开发者自定义语用词
  - 可以覆盖（override）已有的语用词
- 例：定义语用词tell_rule
  - 发送者用它向接收者发送类似“a :- b & c”的规则

```
//tell_rule.mas2j

/*
This example shows how to customise the
KQML to add a new performative,
identified by "tellRule", used by one agent
to send rules like "a :- b & c" to another
agent.
*/

MAS tell_rule {

    infrastructure: Centralised

    agents:
        receiver;
        sender;

    aslSourcePath:
        "src/asl";
}
```

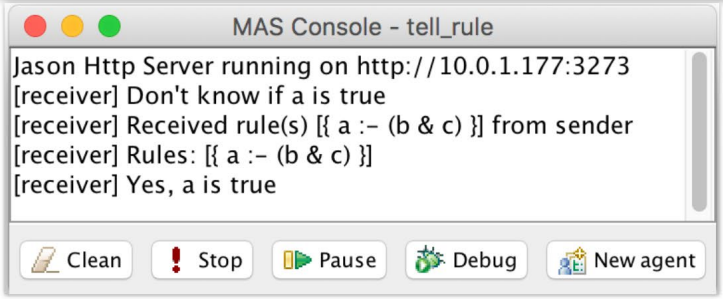
Jason中的语用词 – 自定义语用词tell_rule

- 利用Jason解释器提供的!kqml_received目标，可以在任何满足条件的消息到达时触发事件，从而实现自定义语用词

```
// Agent sender in project tell_rule

/* Initial goals */
!start.

/* Plans */
+!start : true
  <-   // ask the receiver to achieve the goal test
       .send(receiver,achieve,test);
       // send a list with a single rule
       .send(receiver,tellRule, [{a :- b & c}]);
       // ask the receiver to achieve the goal test
       .send(receiver,achieve,test).
```




```
// Agent receiver in project tell_rule

/* Initial beliefs */
b.
c.

/* Plans */
+!test : a <- .print("Yes, a is true").
+!test     <- .print("Don't know if a is true").

// customisation of KQML performative tellRule
+!kqml_received(A,tellRule,Rules,_)
  <- .print("Received rule(s) ", Rules, " from ", A);
     for ( .member(R, Rules) ) {
       +R[source(A)];
     }
     // get all rules and print them
     .relevant_rules(_,LR);
     .print("Rules: ",LR).
```

实践：使用Jason解释器的多Agent编程

- 多Agent编程语言概览
- Jason语法简介
- Jason中的通信与交互
- **Jason编程实例 – 家政机器人**
- Jason编程实例 – 合同网协议

多Agent系统实例：家政机器人

- 假设有一台家政机器人，其唯一任务是为主人提供啤酒。当主人需要它提供啤酒时会向它发送请求，此时它需要走到冰箱的位置，从冰箱中取出啤酒，并将啤酒带回主人身边。
- 然而，机器人需要注意到啤酒的库存可能并不充分，因此它还需要向超市订购更多的啤酒。
- 此外，机器人需要遵循健康卫生部门写入的规则，即每日主人喝的啤酒不能超过一定的数额。
- 需要设计机器人、主人和超市的Agent逻辑。

家政机器人：符号定义

- 定义在实例中会使用到的一些环境感知：
  - at(robot, Place)：机器人当前所处的位置
    - 为了简化这个实例，假设只能观察到机器人出现在冰箱（fridge）和主人（owner） 两个位置
  - stock(beer, N)：当冰箱打开时，Agent能够观察到当前冰箱中的啤酒库存，用变量N表示
  - has(owner, beer)：机器人观察到主人手里是否有啤酒
- 定义每个Agent可以采取的动作：
  - 主人owner：喝一口啤酒
  - 机器人robot：开冰箱、取啤酒、关冰箱、移动、给主人递啤酒
  - 超市supermarket：配送啤酒

家政机器人：Owner设计

```
!get(beer).  // initial goal

/* Plans */

@g
+!get(beer) : true
   <- .send(robot, achieve, has(owner,beer)).

@h1
+has(owner,beer) : true
   <- !drink(beer).
@h2
-has(owner,beer) : true
   <- !get(beer).

// while I have beer, sip
@d1
+!drink(beer) : has(owner,beer)
   <- sip(beer);
      !drink(beer).
@d2
+!drink(beer) : not has(owner,beer)
   <- true.
```

将has(owner, beer)的目标委托给机器人

得到啤酒后将目标设为喝酒，没有啤酒后将目标设为获取啤酒

目标为喝酒且当前有啤酒时，喝一口啤酒，否则什么都不做

家政机器人：Supermarket设计

```
last_order_id(1). // initial belief
```

初始订单号为1

```
// plan to achieve the the goal "order" from agent Ag
+!order(Product,Qtd)[source(Ag)] : true
   <- ?last_order_id(N);
      OrderId = N + 1;
      -+last_order_id(OrderId);
      deliver(Product,Qtd);
      .send(Ag, tell, delivered(Product,Qtd,OrderId)).
```

检查当前订单号并将其加1，同时对信念进行更新

配送后通知订单配送完成

家政机器人：Robot设计

```
limit(beer,10).
```

假设本例中啤酒限额为10

```
/* Rules */
```

```
too_much(B) :-
    .date(YY,MM,DD) &
    .count(consumed(YY,MM,DD,_,_,_,B),QtdB) &
    limit(B,Limit) &
    QtdB > Limit.
```

定义饮用过量的规则为：
统计当天所有时段的啤酒饮用数目，且数目超出限额

consumed(YY, MM, DD, HH, NN, SS, B)表示对已使用物品的信念

家政机器人：Robot设计（续）

```
/* Plans */

@h1
+!has(owner,beer)
   :  available(beer,fridge) & not too_much(beer)
   <- !at(robot,fridge);
      open(fridge);
      get(beer);
      close(fridge);
      !at(robot,owner);
      hand_in(beer);
      // remember that another beer will be consumed
      .date(YY,MM,DD); .time(HH,NN,SS);
      +consumed(YY,MM,DD,HH,NN,SS,beer).

@h2
+!has(owner,beer)
   :  not available(beer,fridge)
   <- .send(supermarket, achieve, order(beer,5));
      !at(robot,fridge). // go to fridge and wait there.

@h3
+!has(owner,beer)
   :  too_much(beer) & limit(beer,L)
   <- .concat("The Department of Health does not allow me ",
              "to give you more than ", L,
              " beers a day! I am very sorry about that!",M);
      .send(owner,tell,msg(M)).
```

当前啤酒数量充足且主人没有喝酒过量时，前往冰箱取啤酒，并回到主人位置交给主人，同时记录啤酒饮用情况

没有啤酒时向超市订购

告诉主人啤酒饮用过量

家政机器人：Robot设计（续）

```
@m1
+!at(robot,P) : at(robot,P) <- true.
@m2
+!at(robot,P) : not at(robot,P)
  <- move_towards(P);
     !at(robot,P).
```

当不在目标位置时进行移动

```
// when the supermarket finishes the order, try the 'has'
// goal again
@a1
+delivered(beer,Qtd,OrderId)[source(supermarket)] : true
  <- +available(beer,fridge);
     !has(owner,beer).
```

得到啤酒时设置目标为向主人提供啤酒

```
// when the fridge is openned, the beer stock is perceived
// and thus the available belief is updated
@a2
+stock(beer,0)
   :  available(beer,fridge)
   <- -available(beer,fridge).

@a3
+stock(beer,N)
   :  N > 0 & not available(beer,fridge)
   <- +available(beer,fridge).
```

根据库存设置当前的啤酒情况

家政机器人：系统运行

- 多Agent系统定义：

```
MAS domestic_robot {

    environment: HouseEnv(gui)

    agents: robot;
            owner;
            supermarket agentArchClass SupermarketArch;
}
```

- 运行结果：

```
[robot] doing: move_towards(fridge)
[robot] doing: move_towards(fridge)
[robot] doing: move_towards(fridge)
[robot] doing: open(fridge)
[robot] doing: get(beer)
[robot] doing: close(fridge)
[robot] doing: move_towards(owner)
[robot] doing: move_towards(owner)
...
[supermarket] doing: deliver(beer,5) ...
[owner] saying: Message from robot: The Department of
Health does not allow me to give you more than 10 beers
a day! I am very sorry about that!
```

实践：使用Jason解释器的多Agent编程

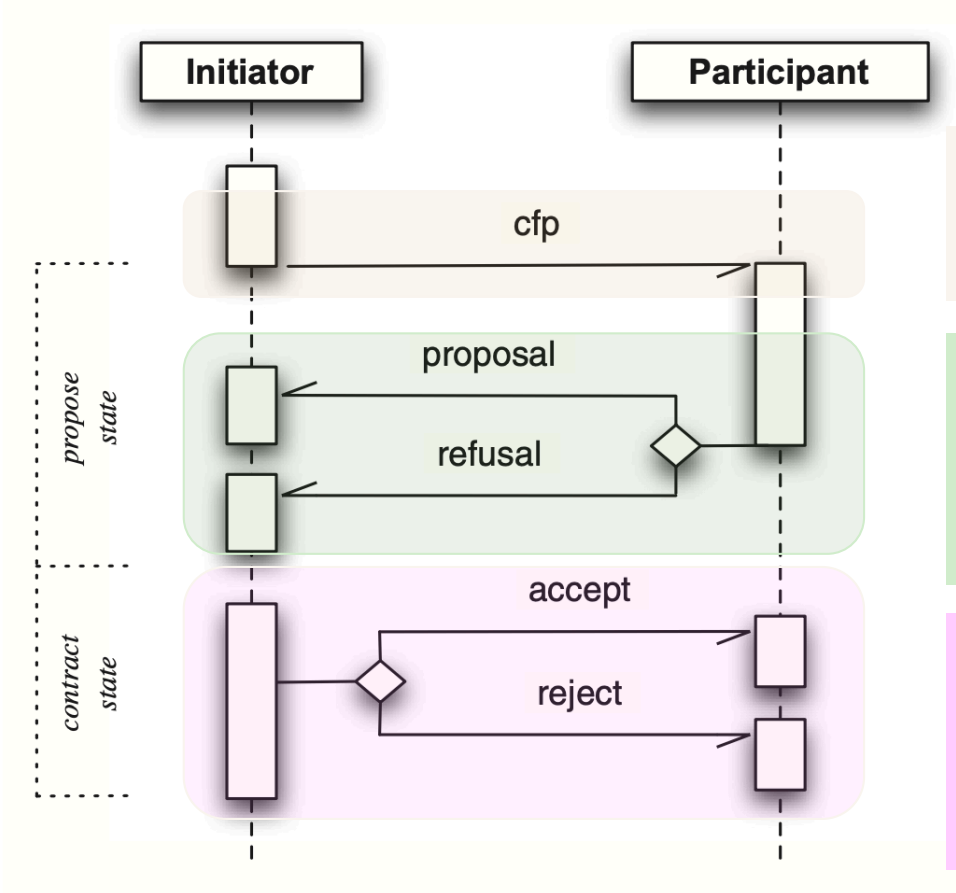
- 多Agent编程语言概览
- Jason语法简介
- Jason中的通信与交互
- Jason编程实例 – 家政机器人
- **Jason编程实例 – 合同网协议**

合同网协议：成员

- 考虑使用Jason搭建一个合同网协议。合同网中的成员有：
  - 1个合同发起者（initiator）
    - 希望将它的任务委托给其它Agent（参与者），这些参与者通过竞标来决定执行这些任务
  - 3个进行正常竞标的参与者
  - 1个总是拒绝报价的参与者
  - 1个什么都不做的参与者

```
MAS cnp {
    agents:
        c;     // the CNP initiator
        p #3; // three participants able to send proposals
        pr;    // a participant that always refuses
        pn;    // a participant that does not answer
}
```

合同网协议：顺序图



每个任务请求通过cfp (call for proposals) 进行标识

每个Agent（参与者）在收到任务请求后给发起者发送报价或是拒绝报价

在任务委托到期时，发起者会根据收到的报价，选择一个参与者执行任务

合同网协议：沉默者

- 沉默者：合同网协议中最简单的参与者
  - 不会回应任何由发起者提出的报价
  - 唯一需要做的任务：在初始化时，告诉发起人自己的存在
  - 使用内置动作.my_name获取Agent标识并发送给发起者

```
// the name of the agent playing initiator in the CNP
plays(initiator,c).

// send a message to the initiator introducing myself
// as a participant
+plays(initiator,In)
   :   .my_name(Me)
   <-  .send(In,tell,introduction(participant,Me)).
```

合同网协议：拒绝者

- 拒绝者：拒绝发起者提出的任何委托

与沉默者相同：在初始化时，告诉发起人自己的存在

```
plays(initiator,c).
+plays(initiator,In)
   :   .my_name(Me)
   <-  .send(In,tell,introduction(participant,Me)).
```

```
// plan to answer a CFP
+cfp(CNPId,Task)[source(A)]
   :    plays(initiator,A)
   <-  .send(A,tell,refuse(CNPId)).
```

在收到任务请求后给发起者发送拒绝报价的消息

合同网协议：正常报价者

- 正常报价者：根据完成任务的预算进行报价
  - 使用内置的.random行动生成0~1范围的随机数

```
// gets the price for the product,
// (a random value between 100 and 110).
price(Task,X) :- .random(R) & X = (10*R)+100.
```

为每个Agent完成任务设置随机的预算

```
plays(initiator,c).

/* Plans */

// send a message to initiator introducing myself
// as a participant
+plays(initiator,In)
   :  .my_name(Me)
   <- .send(In,tell,introduction(participant,Me)).
```

与沉默者、拒绝者相同：在初始化时，告诉发起人自己的存在

合同网协议：正常报价者（续）

收到cfp消息后，根据完成任务的预算进行报价，并记录报价信息到信念库中

```
// answer a Call For Proposal
@c1 +cfp(CNPId,Task)[source(A)]
   :  plays(initiator,A) & price(Task,Offer)
   <- +proposal(CNPId,Task,Offer); // remember my proposal
      .send(A,tell,propose(CNPId,Offer)).
```

```
@r1 +accept_proposal(CNPId)
   :  proposal(CNPId,Task,Offer)
   <- .print("My proposal '",Offer,"' won CNP ",CNPId,
             " for ",Task,"!").
      // do the task and report to initiator
```

报价被接受时输出报价信息，并完成任务

```
@r2 +reject_proposal(CNPId)
   <- .print("I lost CNP ",CNPId, ".");
      -proposal(CNPId,_,_). // clear memory
```

报价被拒绝时输出失败信息，并在信念库中清除该报价的记录

合同网协议：发起者

- 发起者需要完成的工作包括：初始化合同网协议、发送cfp、接收回复、选中竞标成功者并宣布结果

```
/* Initial goals */

!startCNP(1,fix(computer_123)).

/* Plans */

// start the CNP
+!startCNP(Id,Object)
   <- .wait(2000);  // wait participants introduction
      +cnp_state(Id,propose);   // remember the state of the CNP
      .findall(Name,introduction(participant,Name),LP);
      .print("Sending CFP to ",LP);
      .send(LP,tell,cfp(Id,Object));
      .concat("+!contract(",Id,")",Event);
      // the deadline of the CNP is now + 4 seconds, so
      // the event +!contract(Id) is generated at that time
      .at("now +4 seconds", Event).
```

存储当前状态的信念，发送任务请求

超时后自动触发事件

合同网协议：发起者（续）

- 定义规则：判断是否收到了所有报价

```
/* Rules */

all_proposals_received(CNPId) :-
   .count(introduction(participant,_),NP) & // number of participants
   .count(propose(CNPId,_), NO) &      // number of proposes received
   .count(refuse(CNPId), NR) &         // number of refusals received
   NP = NO + NR.
```

发送报价和拒绝报价总数等于参与者数目

- 当收到参与者报价或者拒绝报价的消息时

```
// receive proposal
// if all proposal have been received, don't wait for the deadline
@r1 +propose(CNPId,Offer)
    :  cnp_state(CNPId,propose) & all_proposals_received(CNPId)
    <- !contract(CNPId).

// receive refusals
@r2 +refuse(CNPId)
:  cnp_state(CNPId,propose) & all_proposals_received(CNPId)
<- !contract(CNPId).
```

如果合同网协议的状态为propose，且已收到所有参与者的报价，则触发目标为contract的规划

合同网协议：发起者（续）

- 添加了目标contract后，合同网协议的状态变为contract，计算得到报价最低的Agent

```
// this plan needs to be atomic so as not to accept
// proposals or refusals while contracting
@lc1[atomic]
+!contract(CNPId)
    :  cnp_state(CNPId,propose)
    <- -+cnp_state(CNPId,contract);
       .findall(offer(O,A),propose(CNPId,O)[source(A)],L);
       .print("Offers are ",L);
       L \== []; // constraint the plan execution to at least one offer
       .min(L,offer(WOf,WAg)); // sort offers, the first is the best
       .print("Winner is ",WAg," with ",WOf);
       !announce_result(CNPId,L,WAg);
       -+cnp_state(Id,finished).
```

将状态从propose改为contract

确保至少收到一份报价，并得到报价最低的Agent

宣布竞标结果，并将状态从contract改为finished

合同网协议：发起者（续）

- 如果当前状态不是propose，则什么都不做

```
// nothing todo, the current phase is not 'propose'
@lc2 +!contract(CNPId).
```

- 当删除目标contract时，输出合同网协议失败的信息

```
-!contract(CNPId)
   <- .print("CNP ",CNPId," has failed!").
```

- 当竞标结束时，通知所有参与者竞标结果

```
+!announce_result(_,[],_).
// announce to the winner
+!announce_result(CNPId,[offer(O,WAg)|T],WAg)
   <- .send(WAg,tell,accept_proposal(CNPId));
      !announce_result(CNPId,T,WAg).
// announce to others
+!announce_result(CNPId,[offer(O,LAg)|T],WAg)
   <- .send(LAg,tell,reject_proposal(CNPId));
      !announce_result(CNPId,T,WAg).
```

通知获胜Agent竞标成功

通知其他Agent竞标失败

合同网协议：运行结果

- 运行一次合同网协议，可能得到的结果为

```
[c]  saying: Sending CFP to [p2,p3,pr,pn,p1]
[c]  saying: Offers are [offer(105.42,p3),
                         offer(104.43,p2),
                         offer(109.80,p1)]
[c]  saying: Winner is p2 with 104.43
[p3] saying: I lost CNP 1.
[p1] saying: I lost CNP 1.
[p2] saying: My proposal '104.43' won CNP 1
             for fix(computer\_123)!
```

小结

- AgentSpeak：多Agent编程语言
- Jason：AgentSpeak语言的实现/解释器/开发环境
- Jason语法简介
  - 三部分：信念、目标、规划
- Jason中的通信与交互
  - 消息、语用词
- Jason编程实例
  - Hello World、阶乘计算
  - 家政机器人、合同网协议

编程作业2-1

- 在网站上下载Jason软件包，熟悉jEdit环境
  - https://sourceforge.net/projects/jason
- 阅读examples中的家政机器人 (domestic-robot) 实例的相关内容并理解代码含义。
- 在家政机器人实例中，机器人Agent为完成目标!has(owner,beer) 制定了三个规划（在课件中的标签为h1, h2, h3），但是此处的限制不够严格，因为机器人在执行规划时没有确认**目标来源**是否来自于主人owner Agent（或是其他合理的来源）。
- 请为相应的代码（课件中的robot Agent的h1, h2, h3三个规划）添加上合适的条件，使得在不影响执行结果的情况下执行该目标时限制更为严格。给出修改后的代码和修改的理由。

编程作业2-2

- 在家政机器人实例中，对超市Agent进行修改，使得超市在商品啤酒上具有一定的库存，每次完成订单时打印当前的库存量，当库存不足以满足订单要求时，打印相关的失败信息。

- 提交修改后的超市Agent代码，并在初始啤酒库存分别为100和8时，给出程序的运行结果。

编程作业2-3

- 课件中关于robot Agent的目标!at具有两个规划（在课件中以标签m1和m2表示），请通过运行程序来说明：

```
+!at(robot,P) : at(robot,P) <- true.
+!at(robot,P) : not at(robot,P)
<- move_towards(P);
!at(robot,P).
```

```
+!at(robot,P) : at(robot,P) <- true.
+!at(robot,P) : true
<- move_towards(P);
!at(robot,P).
```

  - 将原先的两个规划的实现（左图）顺序调换，会对运行结果有影响吗？
  - 将实现方式改为右图（即将第二个规划的条件改为true），会对运行结果有影响吗？如果再将两个规划的顺序调换，运行结果仍正确吗？

编程作业2-4

- 在合同网协议实例中，发起者可能想要对已经发送的cfp进行取消，此时需要对所有参与者发送取消信息。假设在事件+!abort(CNPId)中完成，请实现处理该事件的规划，并保证参与者收到取消指令后进行相应的处理。

  注：可以使用语用词untell来撤销某一信念。

编程作业2-5

- 在合同网协议实例中，参与者可能想要撤销自己之前提出的报价，假设在事件+cancel(CNPId)中触发，其需要向发起者提出撤销申请。发起者处于proposal状态时，可以成功撤销该报价，否则将通知参与者撤销失败（可以简化为print）。
- 请编程实现上述事件逻辑。题目中未交代的语句和变量名可以自己设计。

提交Jason代码和实验报告。

截止时间为：**2023年5月9日**