

Introduction to Distributed and Embedded Multi-agent Systems

Carlos Eduardo Pantoja¹
Nilson Mori Lazarin^{1,2}

1. Centro Federal de Educação Tecnológica (CEFET/RJ) - 2. Universidade Federal Fluminense (UFF), Brasil



Março, 2024

BELIEFS AND RULES



Beliefs: Format

belief **[source(type)]**



**the source of
the belief.**

Jason Framework



HÜBNER, Jomi Fred; BORDINI, Rafael Heitor; VIEIRA, Renata. Introdução ao desenvolvimento de sistemas multiagentes com Jason. **XII Escola de Informática da SBC**, v. 2, p. 51–89, 2004.

Jason Framework



Jason | a Java-based interpreter for an extended version of AgentSpeak.

HÜBNER, Jomi Fred; BORDINI, Rafael Heitor; VIEIRA, Renata. Introdução ao desenvolvimento de sistemas multiagentes com Jason. **XII Escola de Informática da SBC**, v. 2, p. 51–89, 2004.

Jason Framework

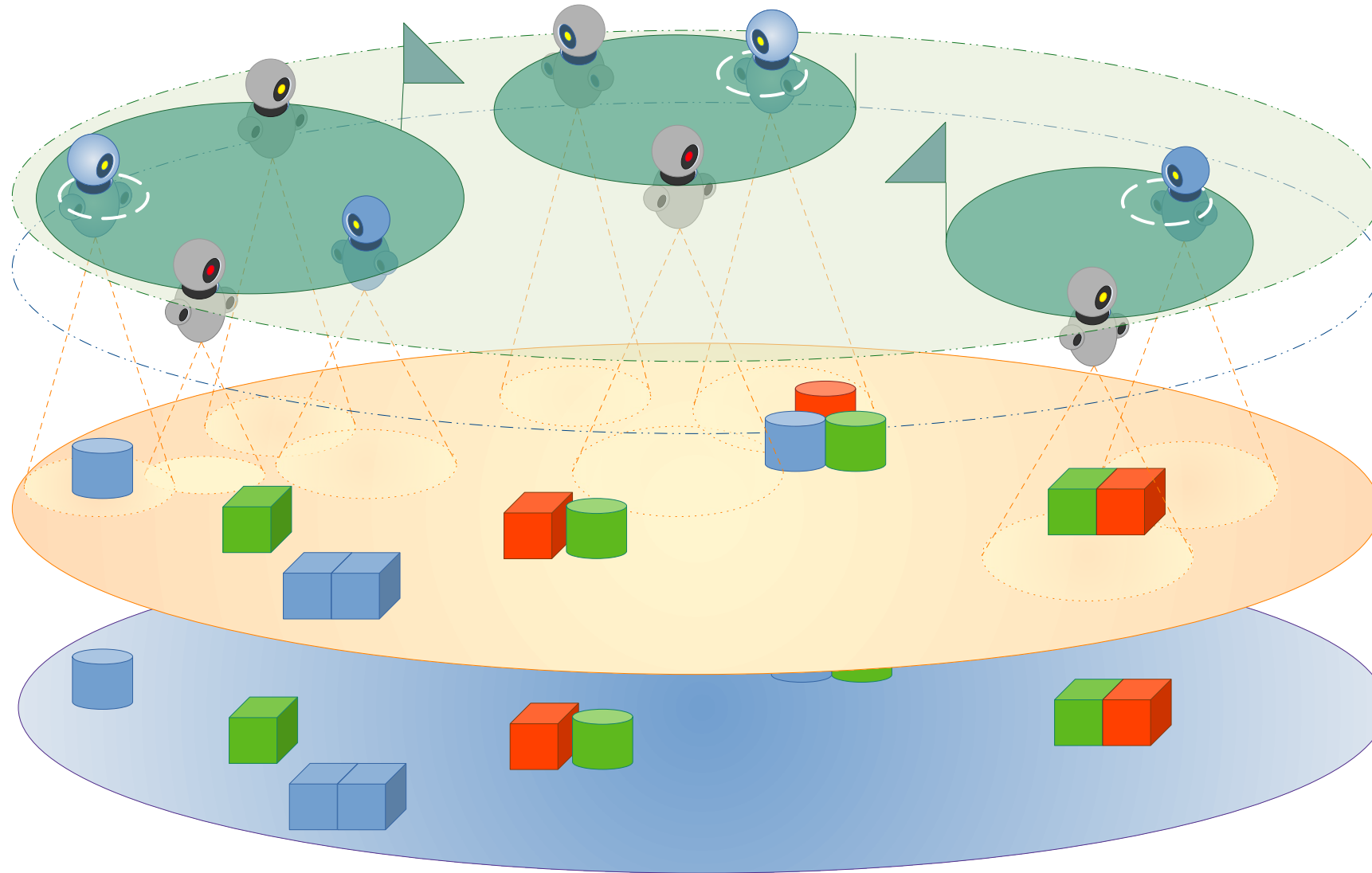


Jason | a Java-based interpreter for an extended version of AgentSpeak.

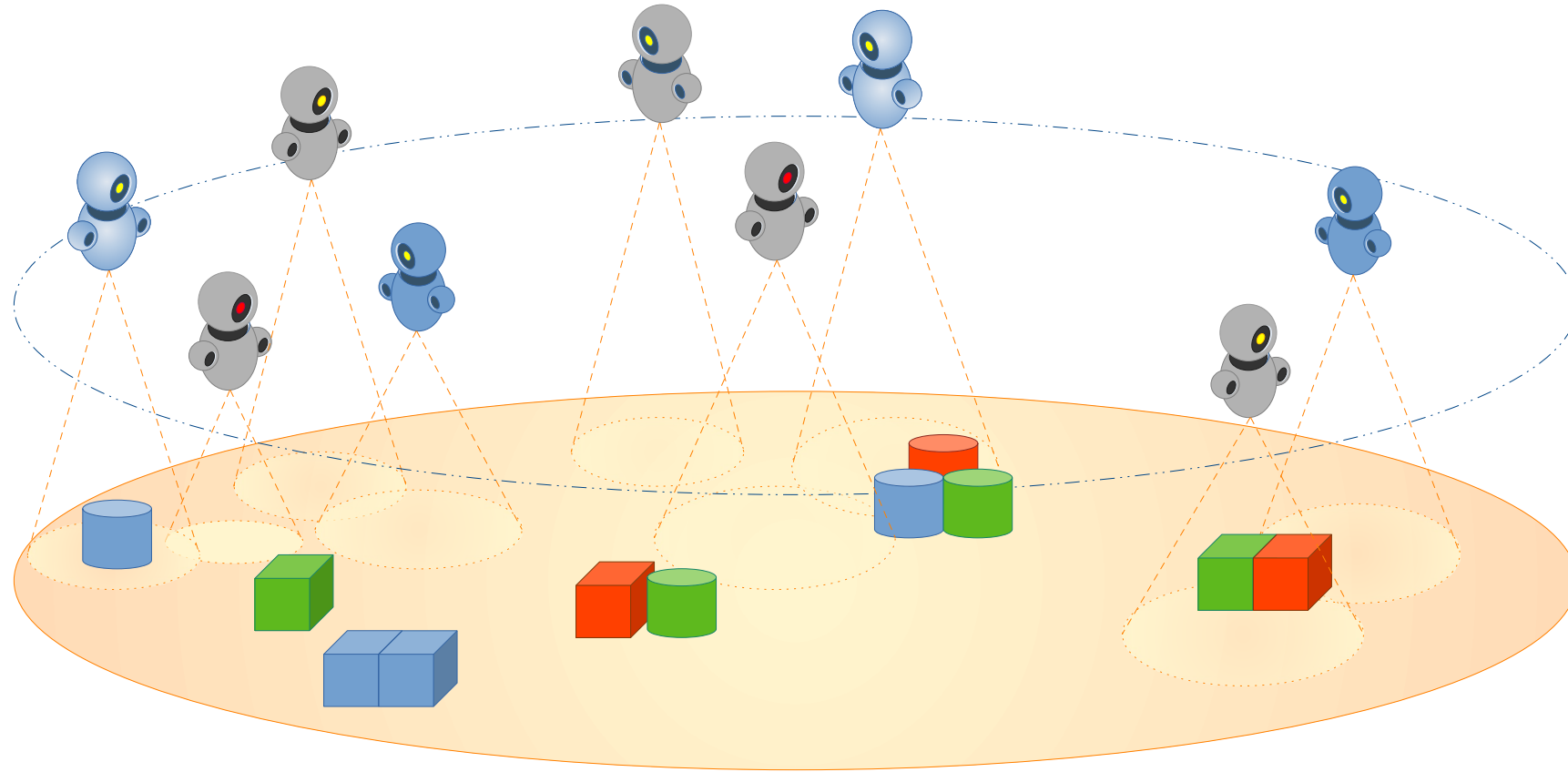
<http://jason.sf.net>

HÜBNER, Jomi Fred; BORDINI, Rafael Heitor; VIEIRA, Renata. Introdução ao desenvolvimento de sistemas multiagentes com Jason. **XII Escola de Informática da SBC**, v. 2, p. 51–89, 2004.

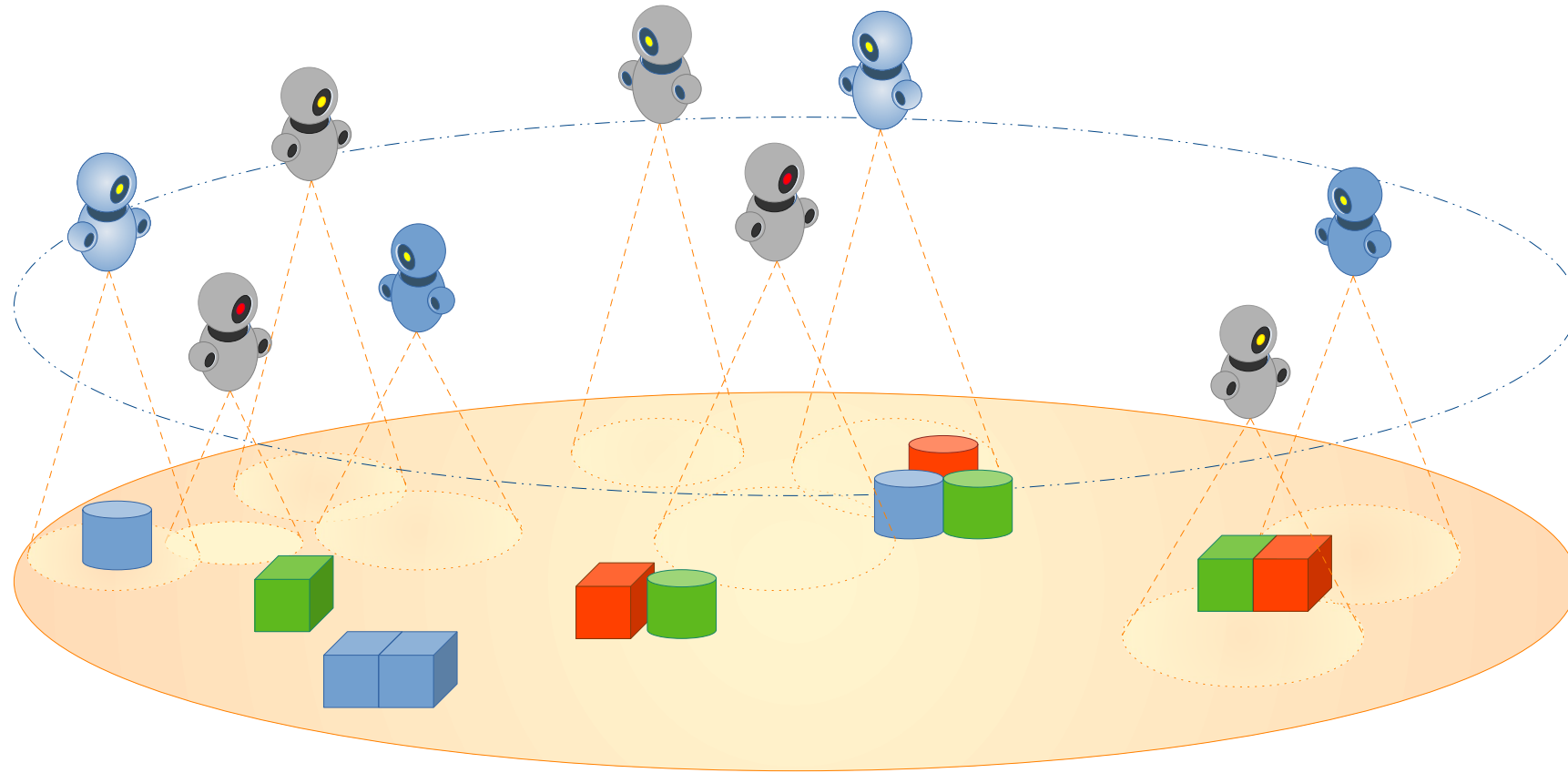
Jason Framework



Jason Framework



Jason Framework



Using Jason, it is possible to program the **agent** and the **endogenous environment** dimensions.

Jason Framework: Crenças

Em Jason, um agente armazena as **informações** percebidas do ambiente; as informações internas; e informações de comunicação através de crenças.

Jason Framework: Crenças

Em Jason, um agente armazena as **informações** percebidas do ambiente; as informações internas; e informações de comunicação através de crenças.

As **crenças** são armazenadas em uma **Base de Crenças** (Belief Base).

Jason Framework: Crenças

Em Jason, um agente armazena as **informações** percebidas do ambiente; as informações internas; e informações de comunicação através de crenças.

As **crenças** são armazenadas em uma **Base de Crenças** (Belief Base).

As crenças são representadas como predicados da **lógica tradicional**. Os **predicados** representam propriedades particulares.

Jason Framework: Beliefs Types

1. Mental Notes (self)

- Informações adicionadas na base de crenças pelo próprio agente.

Jason Framework: Beliefs Types

1. Mental Notes (self)

- Informações adicionadas na base de crenças pelo próprio agente.

2. Communication (agent)

- Informações obtidas pelo agente através da interação com outros agentes.

Jason Framework: Beliefs Types

1. Mental Notes (self)

- Informações adicionadas na base de crenças pelo próprio agente.

2. Communication (agent)

- Informações obtidas pelo agente através da interação com outros agentes.

3. Environmental Perceptions (percepts)

- Informações coletadas pelo agente que são relativas ao sensoramento constante do ambiente.

belief [source(type)]

belief[source(type)]

Beliefs: Format

belief[source(type)]

**a predicate
from logic.**

belief **[source(type)]**

Beliefs: Format

belief **[source(type)]**



**the source of
the belief.**

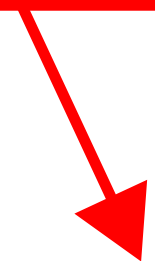
belief [source(type**)]**

belief [source(type**)]**



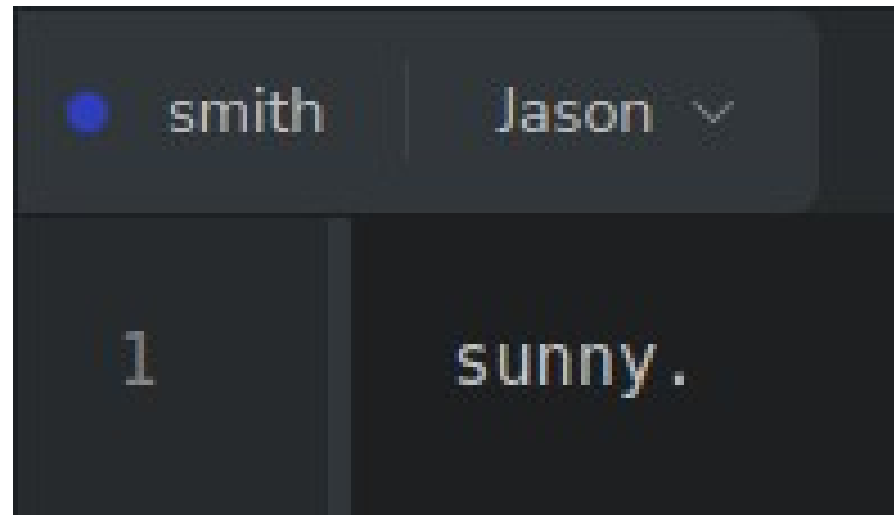
**The belief's
source type .**

belief [source(type**)]**

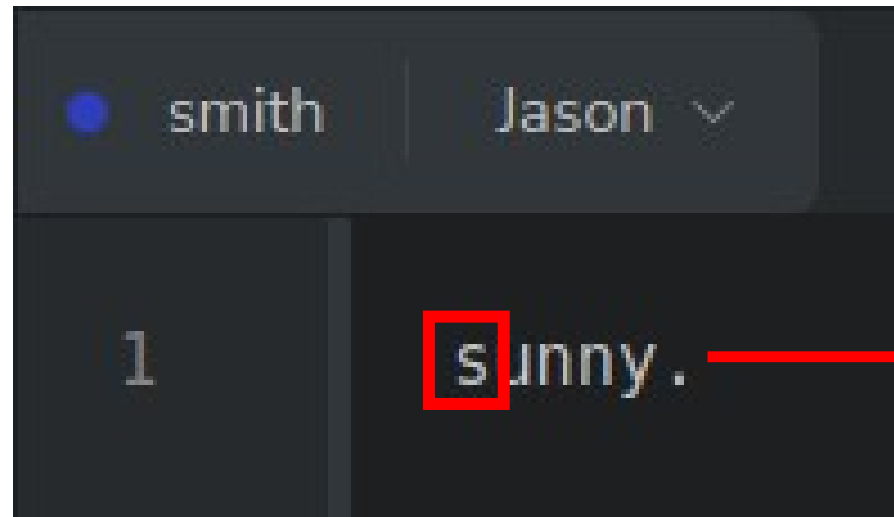


**It can be self,
percepts, or from
other agents.**

Initial Beliefs: Mental Notes

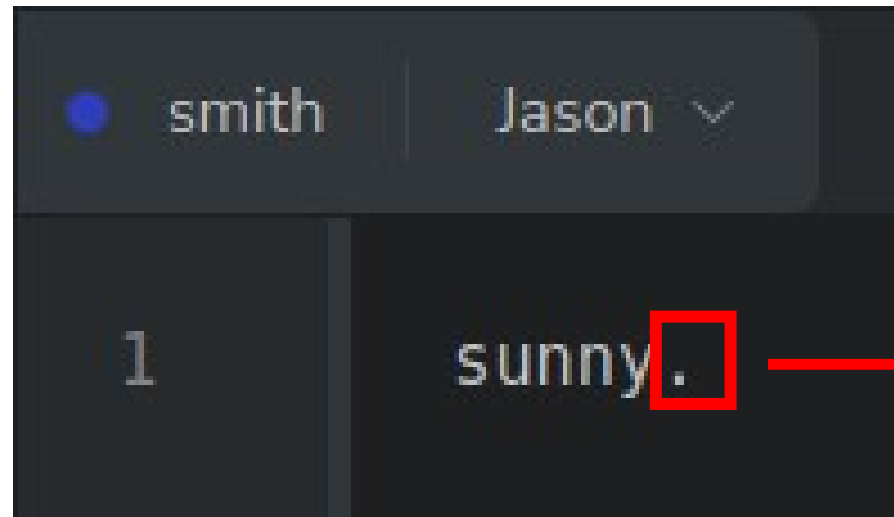


Initial Beliefs: Mental Notes



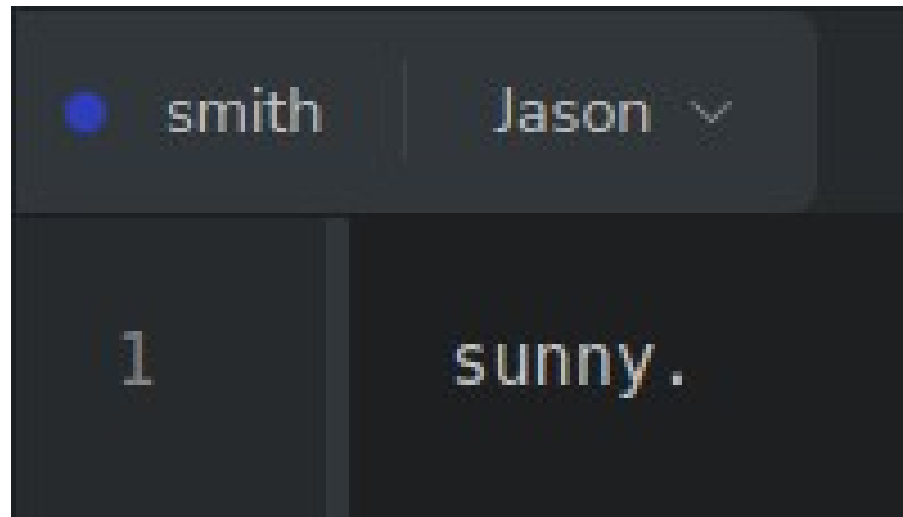
every belief must
start with a
lowercase letter.

Initial Beliefs: Mental Notes

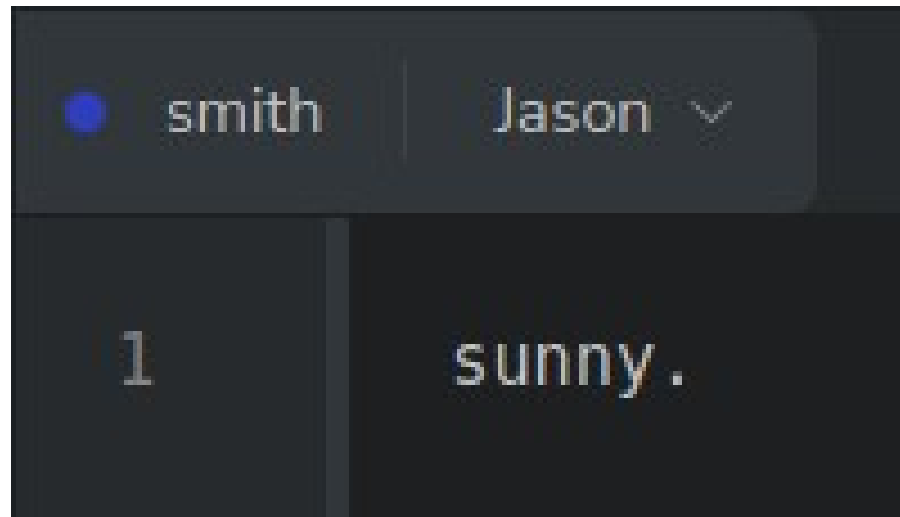


and it must end
with a period.

Initial Beliefs: Mental Notes



Initial Beliefs: Mental Notes



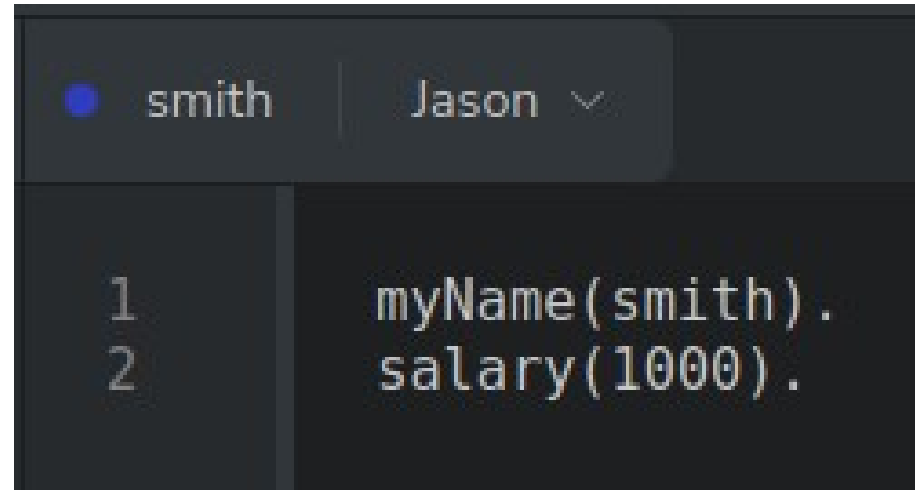
Inspection of agent **smith**

- Beliefs

`sunny[source(self)].`

- Annotations

Initial Beliefs: Mental Notes



A screenshot of a code editor interface. At the top, there is a tab labeled 'smith' with a blue dot, and another tab labeled 'Jason' with a dropdown arrow. Below the tabs, the code editor shows two lines of code: '1 myName(smith).' and '2 salary(1000).'.

```
smith Jason ▾  
1 myName(smith).  
2 salary(1000).
```

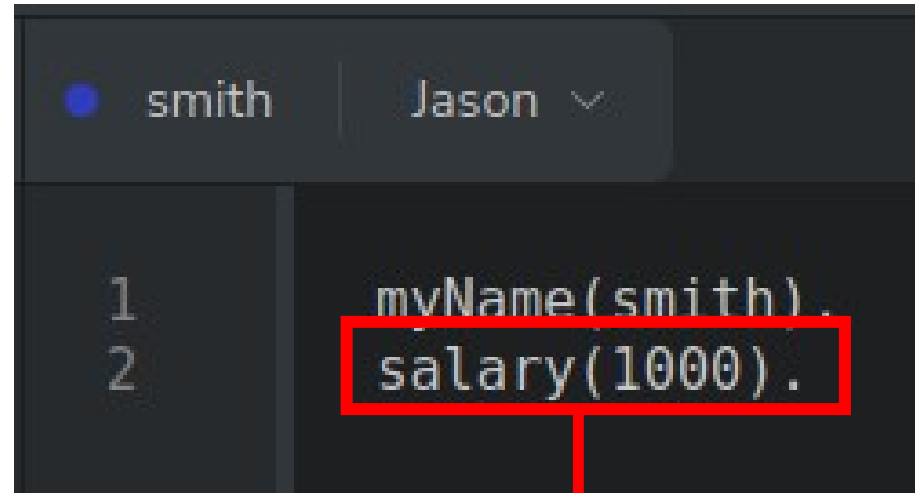
Initial Beliefs: Mental Notes

smith Jason ▾	
1	myName(smith).
2	salary(1000).



predicate(predicate)

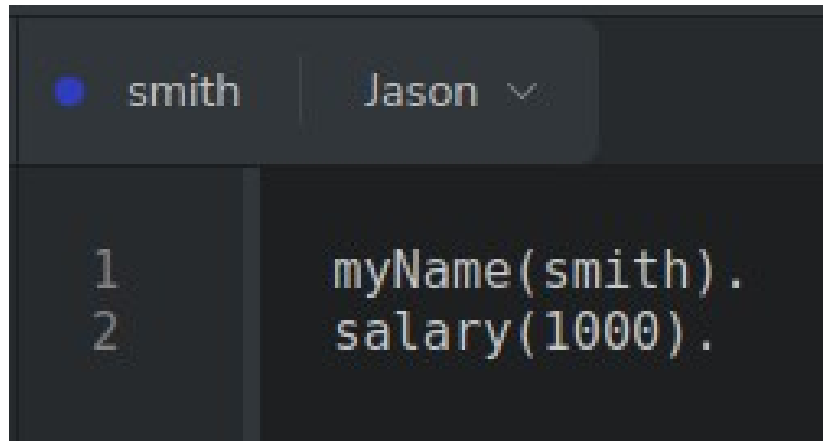
Initial Beliefs: Mental Notes



```
smith | Jason v
1 myName(smith).
2 salary(1000).
```

predicate(int)

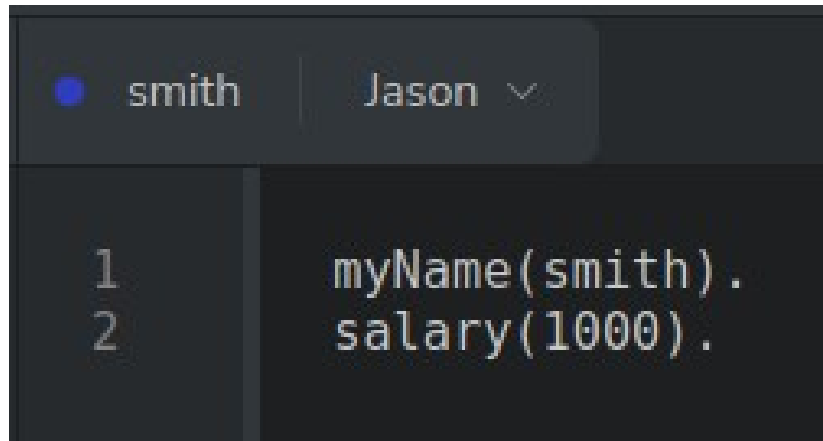
Initial Beliefs: Mental Notes



The screenshot shows a code editor with a dark theme. At the top, there is a tab bar with two tabs: 'smith' (selected with a blue dot) and 'Jason' (with a dropdown arrow). Below the tab bar, the code is displayed in a monospaced font. Line 1 contains 'myName(smith).' and line 2 contains 'salary(1000).'. A vertical line is positioned between the line numbers and the code.

```
1 myName(smith).  
2 salary(1000).
```


Initial Beliefs: Mental Notes



A screenshot of a code editor interface. At the top, there are two tabs: 'smith' (selected with a blue dot) and 'Jason' (with a dropdown arrow). Below the tabs, the code is displayed on a dark background with light-colored text. Line 1 contains 'myName(smith).' and line 2 contains 'salary(1000).'.

```
smith Jason ▾  
1 myName(smith).  
2 salary(1000).
```



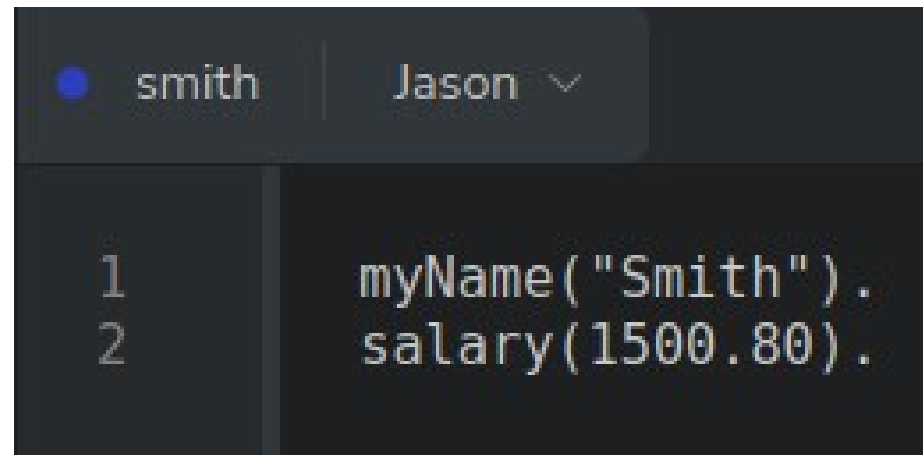
Inspection of agent **smith**

- Beliefs

`myName(smith)[source(self)].`
`salary(1000)[source(self)].`

- Annotations

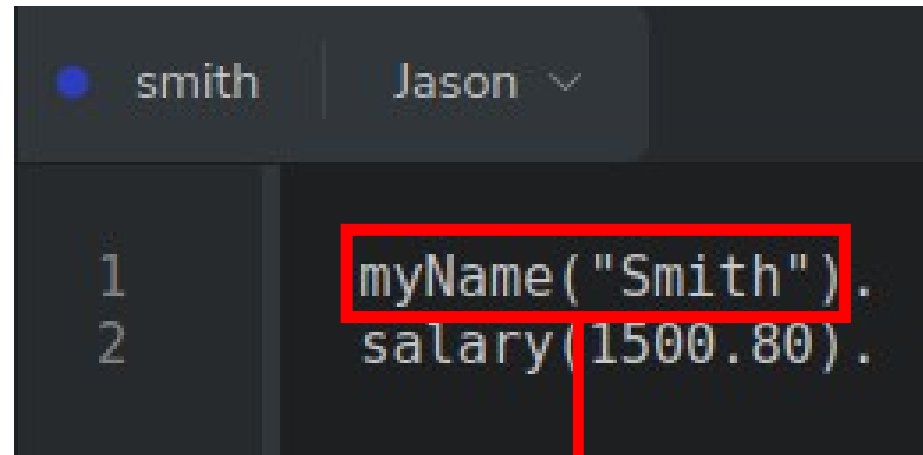
Initial Beliefs: Mental Notes



The screenshot shows a code editor with a dark background. At the top, there is a tab labeled 'smith' with a blue dot next to it, and another tab labeled 'Jason' with a dropdown arrow. Below the tabs, there is a list of two items, numbered 1 and 2, followed by a code block containing two lines of text: 'myName("Smith").' and 'salary(1500.80).'. The code is written in a light gray font.

```
smith | Jason ▾  
1 myName("Smith").  
2 salary(1500.80).
```

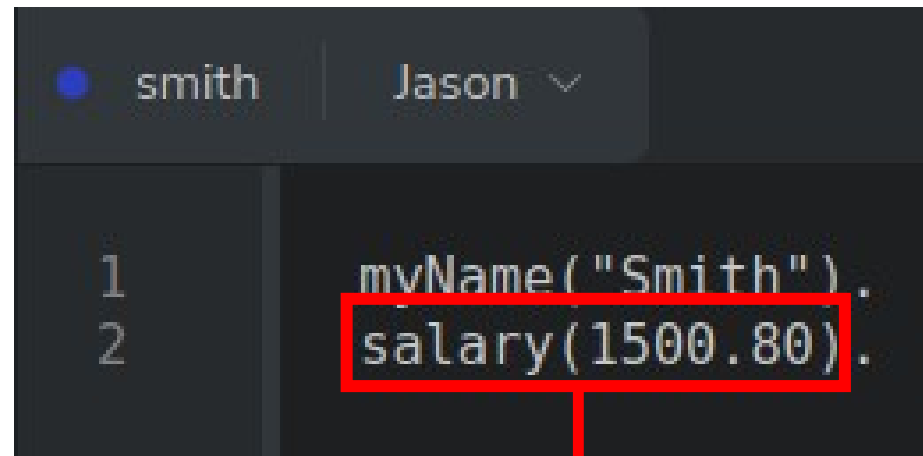
Initial Beliefs: Mental Notes



```
smith | Jason ▾  
1 myName("Smith");  
2 salary(1500.80);
```

predicate(String)

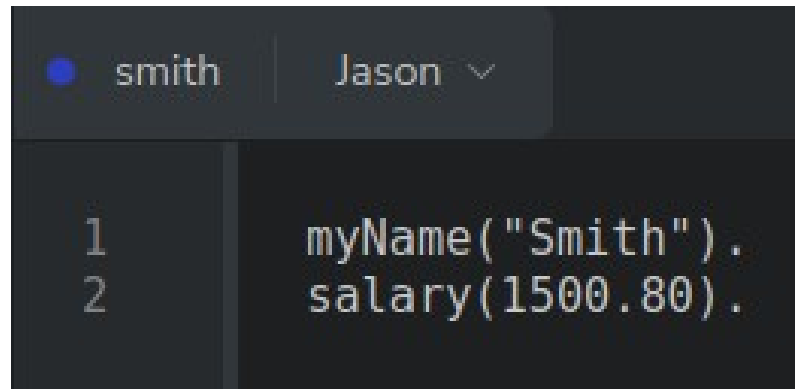
Initial Beliefs: Mental Notes



```
smith | Jason ▾  
1 myName("Smith");  
2 salary(1500.80);
```

predicate(float)

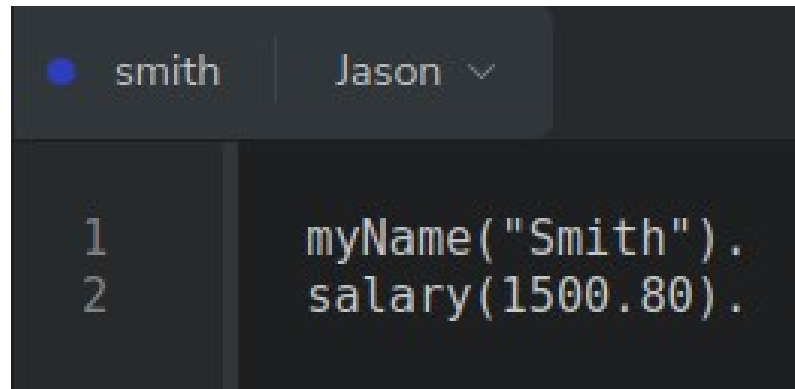
Initial Beliefs: Mental Notes



The image shows a screenshot of a code editor interface. At the top, there are two tabs: 'smith' (selected with a blue dot) and 'Jason' (with a dropdown arrow). Below the tabs, the code is displayed on a dark background with light-colored text. The code consists of two lines, numbered 1 and 2 in the left margin. Line 1 is 'myName("Smith").' and line 2 is 'salary(1500.80).'. The code is written in a monospaced font.

```
smith Jason ▾  
1 myName("Smith").  
2 salary(1500.80).
```

Initial Beliefs: Mental Notes



A screenshot of a code editor interface. At the top, there are two tabs: 'smith' (selected with a blue dot) and 'Jason' (with a dropdown arrow). The editor area shows two lines of code: line 1 is `myName("Smith").` and line 2 is `salary(1500.80).`



Inspection of agent **smith**

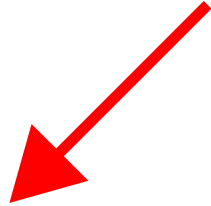
- Beliefs

`myName("Smith")[source(self)].`
`salary(1500.8)[source(self)].`

- Annotations

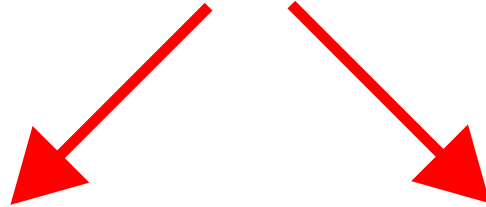
predicate(value)

predicate(value)



predicate

predicate(value)



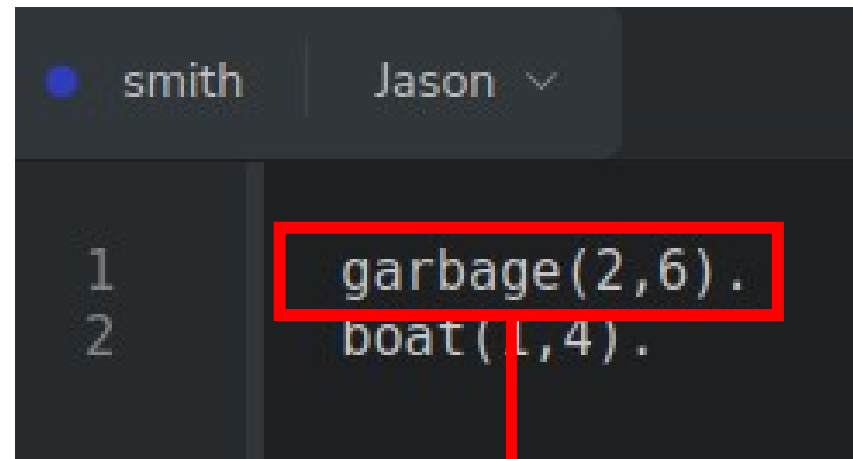
predicate

int, float, String, etc.

Initial Beliefs: Mental Notes

smith	Jason ▾
1	garbage(2,6) .
2	boat(1,4) .

Initial Beliefs: Mental Notes



```
smith | Jason v
1 garbage(2,6).
2 boat(1,4).
```

predicate(int, int)

Initial Beliefs: Mental Notes

smith	Jason ▾
1	garbage(2,6) . boat(1,4) .
2	

Initial Beliefs: Mental Notes

```
smith Jason ▾  
1 garbage(2,6) .  
2 boat(1,4) .
```



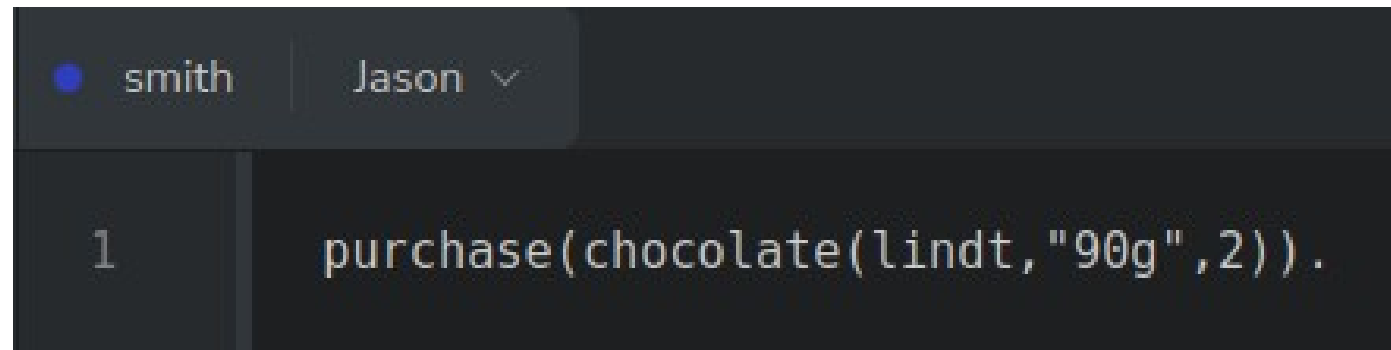
Inspection of agent **smith**

- Beliefs

```
boat(1,4)[source(self)].  
garbage(2,6)[source(self)].
```

- Annotations

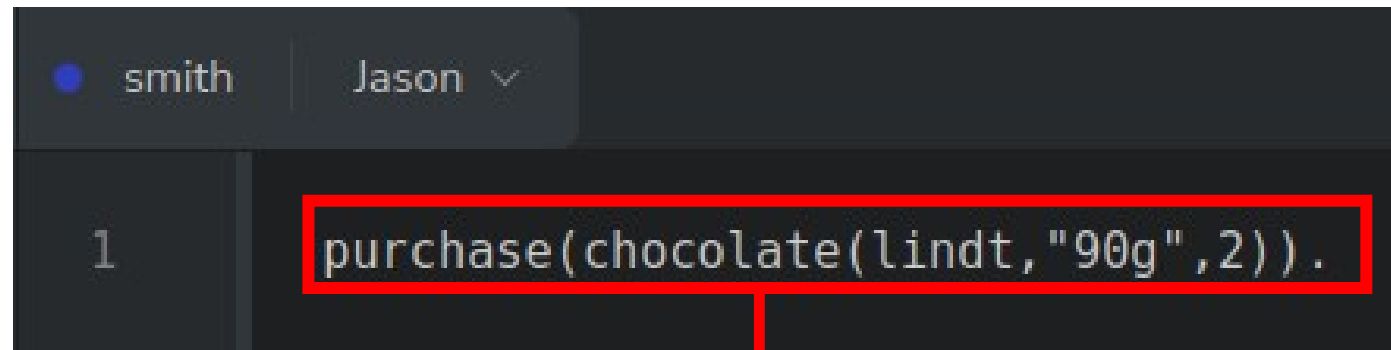
Initial Beliefs: Mental Notes



The screenshot shows a dark-themed interface for a multi-agent system. At the top, there is a header bar with two tabs: 'smith' (selected with a blue dot) and 'Jason' (with a dropdown arrow). Below the tabs, there is a list of mental notes. The first note is numbered '1' and contains the text 'purchase(chocolate(lindt,"90g",2)).'.

Agent	Mental Note
smith	1 purchase(chocolate(lindt,"90g",2)).

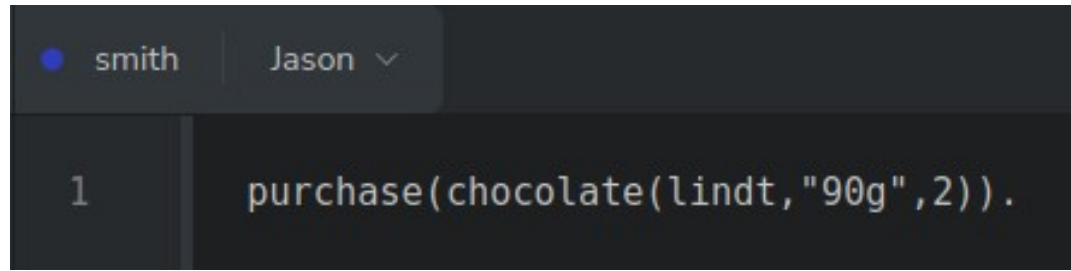
Initial Beliefs: Mental Notes



The screenshot shows a code editor interface. At the top, there are two tabs: 'smith' (selected with a blue dot) and 'Jason' (with a dropdown arrow). Below the tabs, on the left, is a line number '1'. To the right of the line number, the text `purchase(chocolate(lindt, "90g", 2)).` is displayed. This text is enclosed in a red rectangular box. A red arrow points from the bottom center of this box down towards the text below.

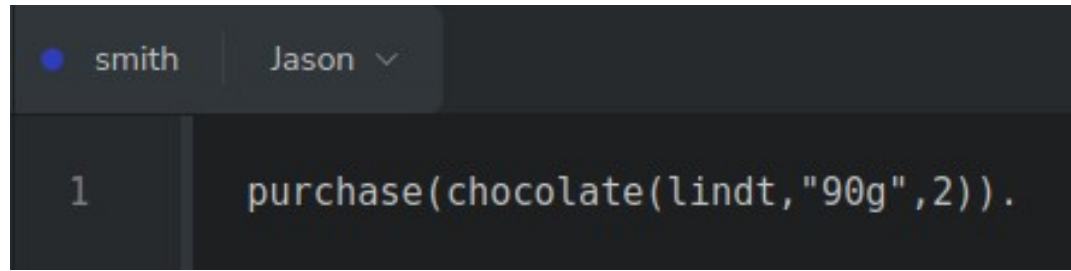
predicate(predicate(predicate, String, int)

Initial Beliefs: Mental Notes



A screenshot of a code editor interface. At the top, there is a tab labeled 'smith' with a blue dot next to it, and a dropdown menu showing 'Jason' with a downward arrow. Below the tab, the code editor displays a single line of code: `1 purchase(chocolate(lindt, "90g", 2)).`

Initial Beliefs: Mental Notes



```
smith Jason ▾  
1 purchase(chocolate(lindt, "90g", 2)).
```

Inspection of agent **smith**

- Beliefs

`purchase(chocolate(lindt, "90g", 2))[source(self)].`

- Annotations

predicate(predicate)

predicate(predicate)

predicate(predicate(predicate))

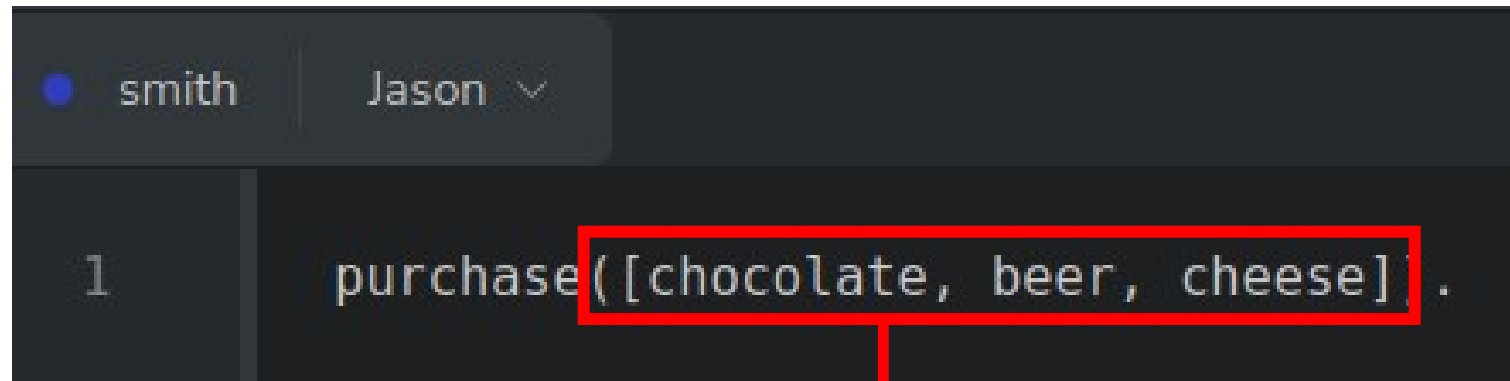
predicate(predicate(predicate))

predicate(predicate(predicate(...)))

Initial Beliefs: Mental Notes

smith	Jason ▾
1	<code>purchase([chocolate, beer, cheese]).</code>

Initial Beliefs: Mental Notes

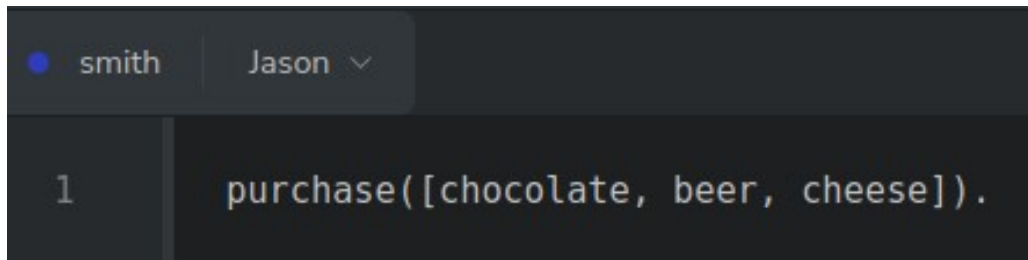


The screenshot shows a user interface for a multi-agent system. At the top, there are two tabs: 'smith' (selected with a blue dot) and 'Jason' (with a dropdown arrow). Below the tabs, there is a list of mental notes. The first note, indexed '1', contains the text 'purchase([chocolate, beer, cheese]).'. The list item '1' is in a dark grey box, and the text is in a lighter grey box. A red rectangular box highlights the list item '1', and a red arrow points from this box down towards the text below.

```
smith Jason ▾  
1 purchase([chocolate, beer, cheese]).
```

predicate([predicate, predicate, predicate])

Initial Beliefs: Mental Notes



The screenshot shows a dark-themed code editor. At the top, there is a header bar with a blue dot, the text "smith", and a dropdown menu labeled "Jason" with a downward arrow. Below the header, the first line of code is numbered "1" and contains the text `purchase([chocolate, beer, cheese]).`

Initial Beliefs: Mental Notes

```
smith Jason ▾  
1 purchase([chocolate, beer, cheese]).
```



Inspection of agent **smith**

- Beliefs

`purchase([chocolate,beer,cheese])[source(self)].`

- Annotations

predicate([list])

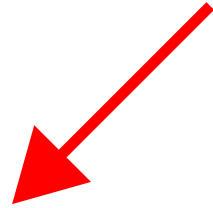
Beliefs: Format

predicate([list])

[value, value, ...]

Beliefs: Format

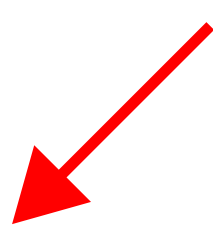
[value, value, ...]



predicate

Beliefs: Format

[value, value, ...]

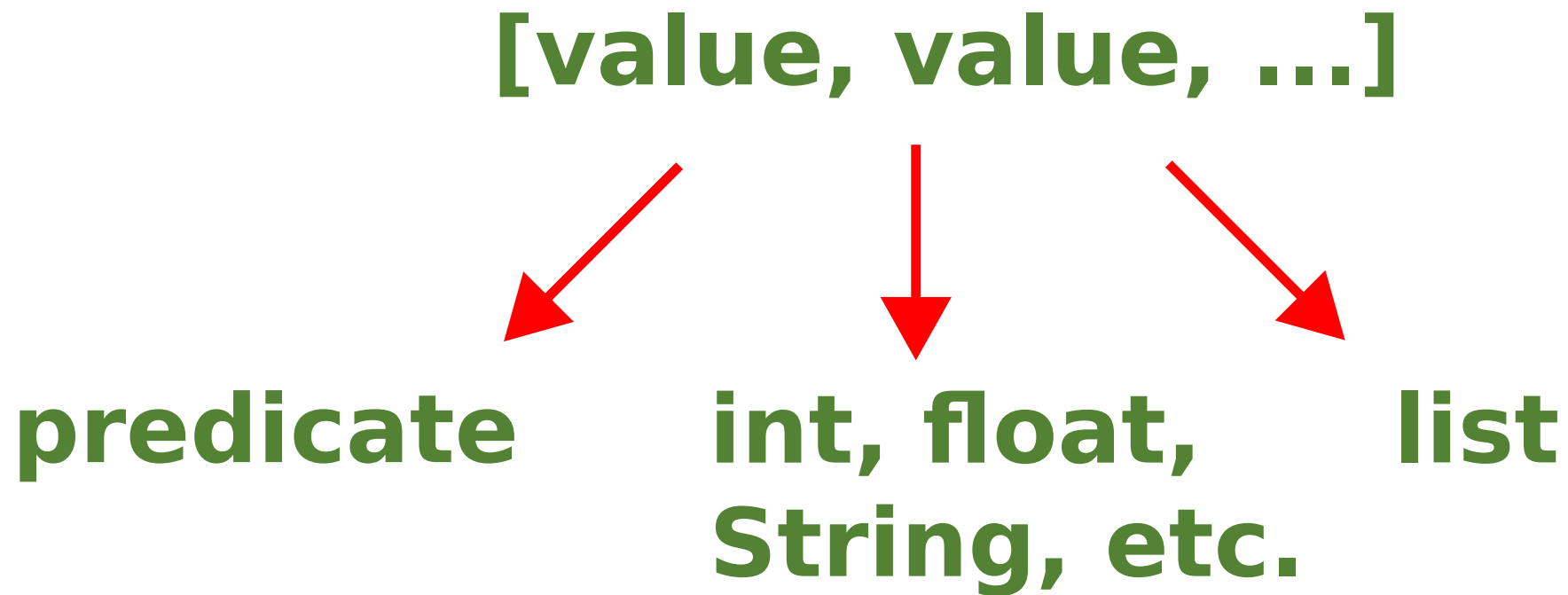


predicate



**int, float,
String, etc.**

Beliefs: Format



Beliefs: Strong Negation

\sim belief [source(type)]

Beliefs: Strong Negation

\sim belief [source(type)]

**every strong
negation starts
with \sim .**

Initial Beliefs: Mental Notes

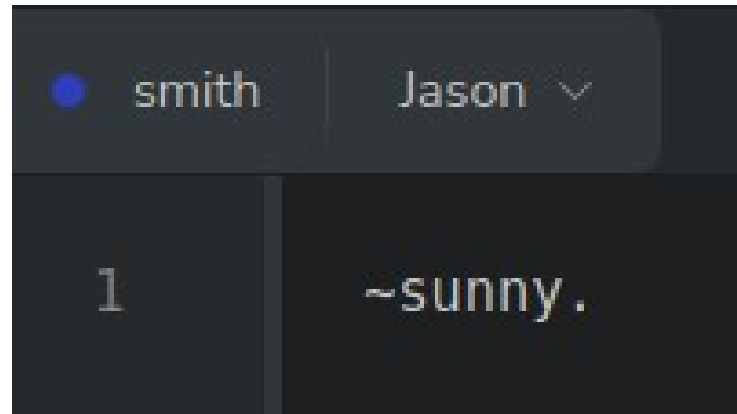
smith Jason ▾	
1	~ sunny.

it negates the
predicate.

Initial Beliefs: Mental Notes

smith	Jason ▾
1	~sunny.

Initial Beliefs: Mental Notes



Inspection of agent **smith**

- Beliefs

`~sunny[source(self)].`

- Annotations

Jason Framework: Rules

A **rule** is a statement that defines relationships between facts (beliefs).

Jason Framework: Rules

A **rule** is a statement that defines relationships between facts (beliefs).

The rule operator can be understood as a **logical implication**. It reads as **"if"** or **"is true when"**.

Rules: Format

head :- body.

Rules: Format

head :- body.



**A consequent
fact or belief.**

Rules: Format

head :- **body**.



**A consequent
fact or belief.**

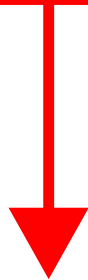
**contains conditions that
must be satisfied
for the rule to hold.**

Rules: Predicate

```
• rule Jason ▾  
1 happy(bob) :- money(bob).  
2  
3 money(bob).  
4
```

Rules: Predicate

```
rule Jason  
1 happy(bob) :- money(bob).  
2  
3 money(bob).
```



**When the agent
believes...**

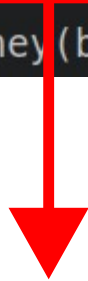
Rules: Predicate

```
rule Jason {  
  1 happy(bob) :- money(bob).  
  2  
  3 money(bob).  
}
```

... this condition is
satisfied.

Rules: Predicate

```
rule Jason
1  happy(bob) :- money(bob) .
2
3  money(bob) .
```



So, it holds ...

Rules: Predicate

```
rule Jason
1  happy(bob) :- money(bob) .
2
3  money(bob) .
```

... but it is NOT A BELIEF.

Rules: Predicate

```
rule Jason
1 happy(bob) :- money(bob) .
2
3 money(bob) .
```



Inspection of agent **rule**

- Beliefs

`money(bob)[source(self)]`.

- Rules

`happy(bob) :-
money(bob).`

- Annotations

Jason Framework: Variables

In Jason, a variable is a **placeholder** for an **unknown value**.

Jason Framework: Variables

In Jason, a variable is a **placeholder** for an **unknown value**.

- denoted by a string consisting of an uppercase letter or an underscore;

Jason Framework: Variables

In Jason, a variable is a **placeholder** for an **unknown value**.

- denoted by a string consisting of an uppercase letter or an underscore;
- it must be instantiated with specific values while executing the agent's reasoning;

Jason Framework: Variables

In Jason, a variable is a **placeholder** for an **unknown value**.

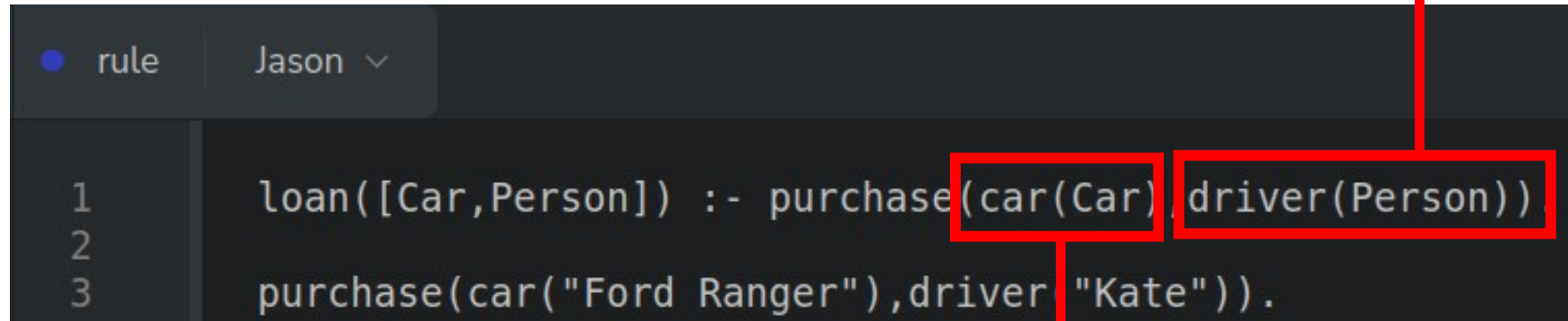
- denoted by a string consisting of an uppercase letter or an underscore;
- it must be instantiated with specific values while executing the agent's reasoning;
- The underscore is a special variable that is often used when the value of a variable is not needed or not used. It is sometimes referred to as a "**don't care**" variable.

Rules: With Variables

```
● rule Jason ▾  
1      loan([Car,Person]) :- purchase(car(Car),driver(Person)).  
2  
3      purchase(car("Ford Ranger"),driver("Kate")).
```

Rules: With Variables

predicate(Variable)

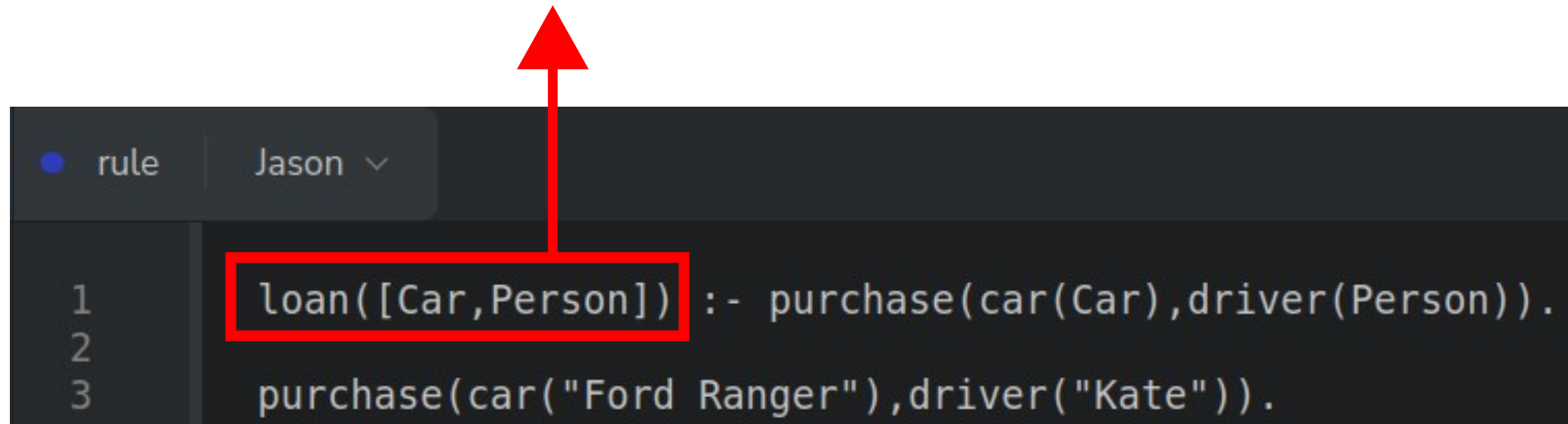


```
rule Jason
1 loan([Car, Person]) :- purchase(car(Car), driver(Person))
2
3 purchase(car("Ford Ranger"), driver("Kate")).
```

predicate(Variable)

Rules: With Variables

predicate([Variable, Variable])



```
rule Jason
1 loan([Car, Person]) :- purchase(car(Car), driver(Person)).
2
3 purchase(car("Ford Ranger"), driver("Kate")).
```

Rules: With Variables

```
rule Jason  
1 loan([Car,Person]) :- purchase(car(Car),driver(Person)).  
2  
3 purchase(car("Ford Ranger"),driver("Kate")).
```



If a belief exists...

Rules: With Variables

```
rule Jason
1   loan([Car,Person]) :- purchase(car(Car),driver(Person)).
2
3   purchase(car("Ford Ranger"),driver("Kate")).
```

Rules: With Variables

The screenshot shows a code editor with a dark theme. At the top, there's a tab labeled 'rule' and a dropdown menu showing 'Jason'. Below, there's a list of three lines of code. Line 1: `loan([Car, Person]) :- purchase(car(Car), driver(Person)).` Line 2: `purchase(car("Ford Ranger"), driver("Kate")).` Line 3: `purchase(car("Ford Ranger"), driver("Kate")).` Red boxes highlight the variables `Car` and `Person` in line 1, and the concrete values `"Ford Ranger"` and `"Kate"` in line 2. Dashed red arrows show the mapping: one arrow from the `Car` box to the `"Ford Ranger"` box, and another from the `Person` box to the `"Kate"` box, illustrating the unification process.

```
1 loan([Car, Person]) :- purchase(car(Car), driver(Person)).
2
3 purchase(car("Ford Ranger"), driver("Kate")).
```

... it binds the value to a variable (unification).

Rules: With Variables

```
rule Jason
1   loan([Car,Person]) :- purchase(car(Car),driver(Person)).
2
3   purchase(car("Ford Ranger"),driver("Kate")).
```

Rules: With Variables

```
rule Jason
1   loan([Car,Person]) :- purchase(car(Car),driver(Person)).
2
3   purchase(car("Ford Ranger"),driver("Kate")).
```



Inspection of agent **rule**

- Beliefs

`purchase(car("Ford Ranger"),driver("Kate"))[source(self)].`

- Rules

`loan([Car,Person]) :-
 purchase(car(Car),driver(Person)).`


- Annotations

Rules: Predicate and Variable

```
• rule Jason ▾  
1 day :- (sky(blue) | sky(grey)) & time(Hour) & Hour <= 17.  
2  
3 sky(blue).  
4 time(14).
```

Rules: Predicate and Variable

```
rule Jason
1  day :- (sky(blue) | sky(grey)) & time(Hour) & Hour <= 17.
2
3  sky(blue).
4  time(14).
```



The verified conditions.

Rules: Predicate and Variable

```
rule Jason
1   day :- (sky(blue) | sky(grey)) & time(Hour) & Hour <= 17.
2
3   sky(blue).
4   time(14).
```



If the beliefs exist...

Rules: Predicate and Variable

```
rule Jason
1  day :- (sky(blue) | sky(grey)) & time(Hour) & Hour <= 17.
2
3  sky(blue).
4  time(14).
```

... the predicate holds, and
agents can use it ...

Rules: Predicate and Variable

```
rule Jason
1  day :- (sky(blue) | sky(grey)) & time(Hour) & Hour <= 17.
2
3  sky(blue).
4  time(14).
```

... but it is NOT A BELIEF.

Rules: Predicate and Variable

```
rule Jason  
1   day :- (sky(blue) | sky(grey)) & time(Hour) & Hour <= 17.  
2  
3   sky(blue).  
4   time(14).
```

Rules: Predicate and Variable

```
rule Jason  
1 day :- (sky(blue) | sky(grey)) & time(Hour) & Hour <= 17.  
2  
3 sky(blue).  
4 time(14).
```



Inspection of agent rule

- Beliefs

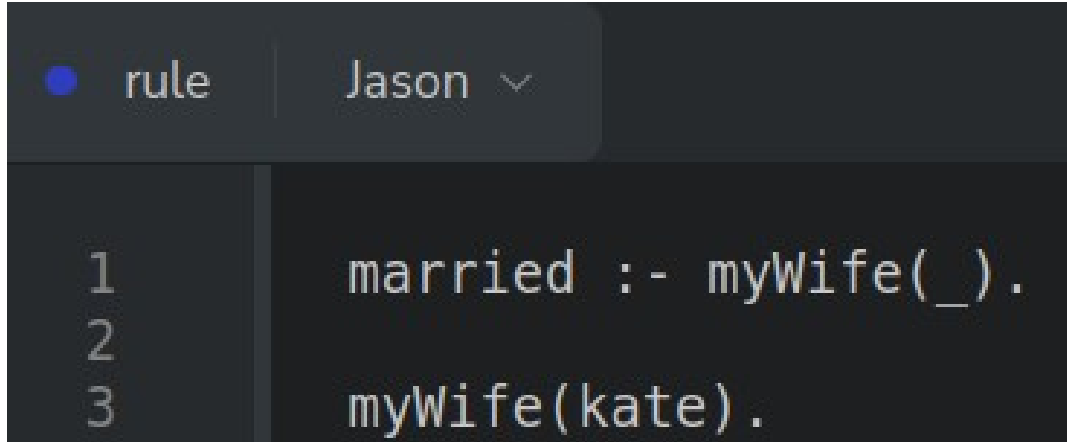
```
sky(blue)[source(self)].  
time(14)[source(self)].
```

- Rules

```
day :-  
    ((sky(blue) | sky(grey)) & (time(Hour) & (Hour <= 17))).
```

- Annotations

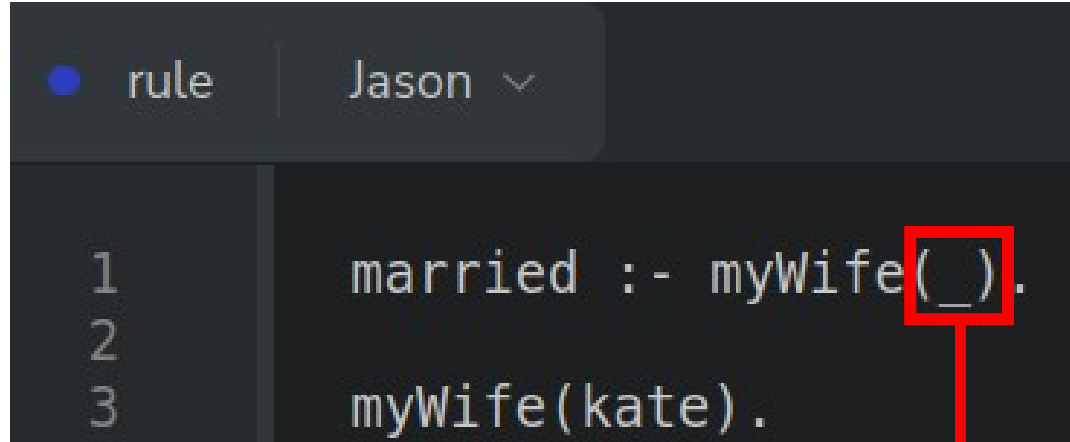
Rules: Underscore (Don't Care)



A screenshot of a Prolog rule editor. At the top, there is a tab labeled 'rule' with a blue dot, and a dropdown menu showing 'Jason' with a downward arrow. Below this, a code editor displays a rule named 'married' with two clauses. The first clause is 'married :- myWife(_).' and the second clause is 'myWife(kate).' The lines are numbered 1, 2, and 3 on the left side of the editor.

```
1 married :- myWife(_).  
2  
3 myWife(kate).
```

Rules: Underscore (Don't Care)



```
rule Jason ▾  
1 married :- myWife(_).  
2  
3 myWife(kate).
```

Don't care its value.

Rules: Underscore (Don't Care)

```
rule Jason
1 married :- myWife(_).
2
3 myWife(kate).
```



Inspection of agent **rule**

- Beliefs

`myWife(kate)[source(self)].`

- Rules

`married :-`
`myWife(_39).`

- Annotations

Rules: Strong Negation

```
rule Jason ▾  
1 ~night :- (sky(blue) | sky(grey)) & time(Hour) & Hour <= 17.  
2  
3 sky(blue).  
4 time(14).
```

Rules: Strong Negation

```
rule Jason
1  ~night :- (sky(blue) | sky(grey)) & time(Hour) & Hour <= 17.
2
3  sky(blue).
4  time(14).
```

~predicate

Rules: Strong Negation

```
rule Jason  
1 ~night :- (sky(blue) | sky(grey)) & time(Hour) & Hour <= 17.  
2  
3 sky(blue).  
4 time(14).
```

Rules: Strong Negation

```
rule Jason
1 ~night :- (sky(blue) | sky(grey)) & time(Hour) & Hour <= 17.
2
3 sky(blue).
4 time(14).
```



Inspection of agent **rule**

- Beliefs	sky(blue)[source(self)]. time(14)[source(self)].
- Rules	~night :- ((sky(blue) sky(grey)) & (time(Hour) & (Hour <= 17))).
- Annotations	



OBRIGADO!

pantoja@cefet-rj.br
nilson.lazarin@cefet-rj.br

