Escola Politécnica da USP- Depto. de Enga. Mecatrônica

PMR-3510 Inteligência Artificial

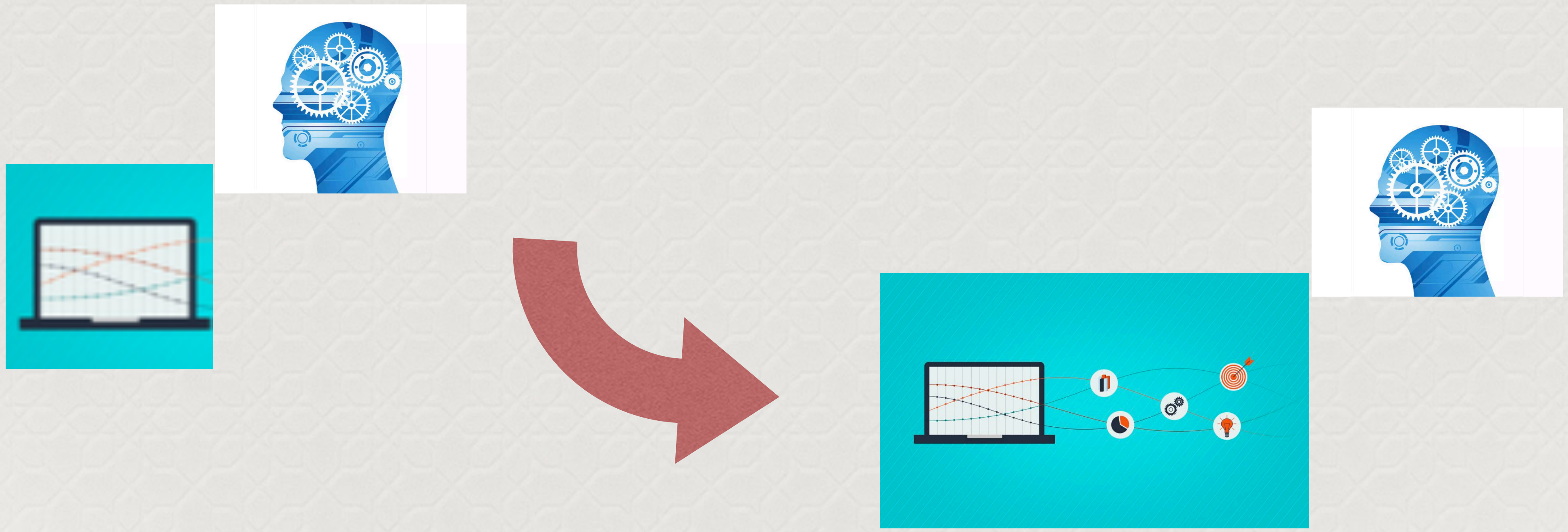
Aula 10- Mecanismo de Inferência

Prof. José Reinaldo Silva
reinaldo@usp.br

d-l@b

Resolução de problemas com IA

Existem duas maneiras de resolver problemas usando IA: uma é algorítmica, usando busca (clássica ou informada); a outra é usando inferência lógica.



Robô Valkyrie mede 1,8 metro e pesa 125 quilos. (Foto: Nasa)

→ *métodos de busca clássica e informada*

↓ *programação lógica*

↘ *métodos de inferência*

Inference

$KB \vdash_i \alpha$ = sentence α can be derived from KB by procedure i

Consequences of KB are a haystack; α is a needle.
Entailment = needle in haystack; inference = finding it

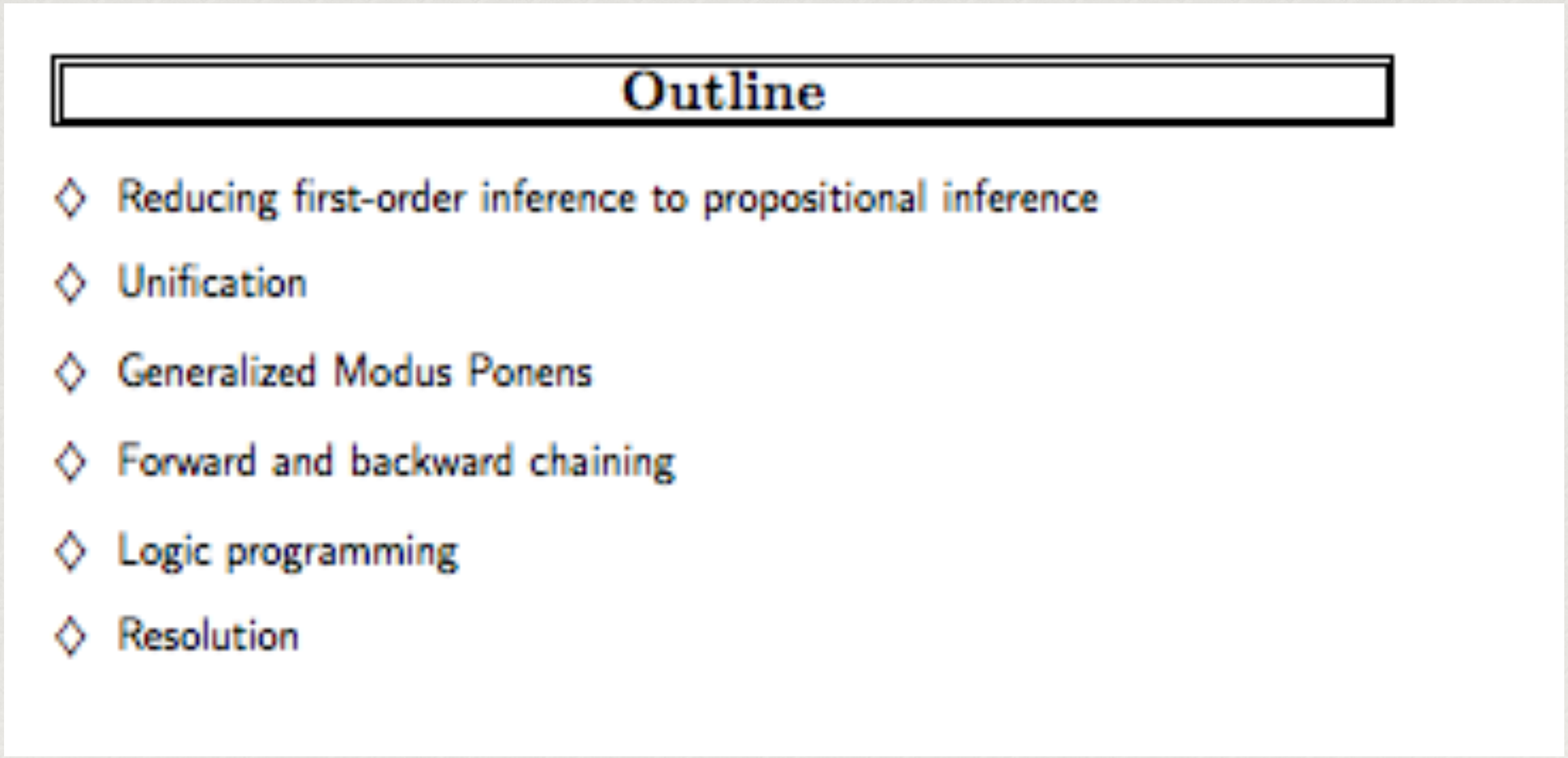
Soundness: i is sound if
whenever $KB \vdash_i \alpha$, it is also true that $KB \models \alpha$

Completeness: i is complete if
whenever $KB \models \alpha$, it is also true that $KB \vdash_i \alpha$

Preview: we will define a logic (first-order logic) which is expressive enough to say almost anything of interest, and for which there exists a sound and complete inference procedure.

That is, the procedure will answer any question whose answer follows from what is known by the KB.

Stuart Russell, Univ. of California - Berkeley, autor do livro texto.



Outline

◇ Reducing first-order inference to propositional inference

◇ Unification

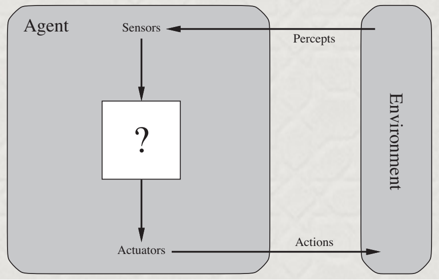
◇ Generalized Modus Ponens

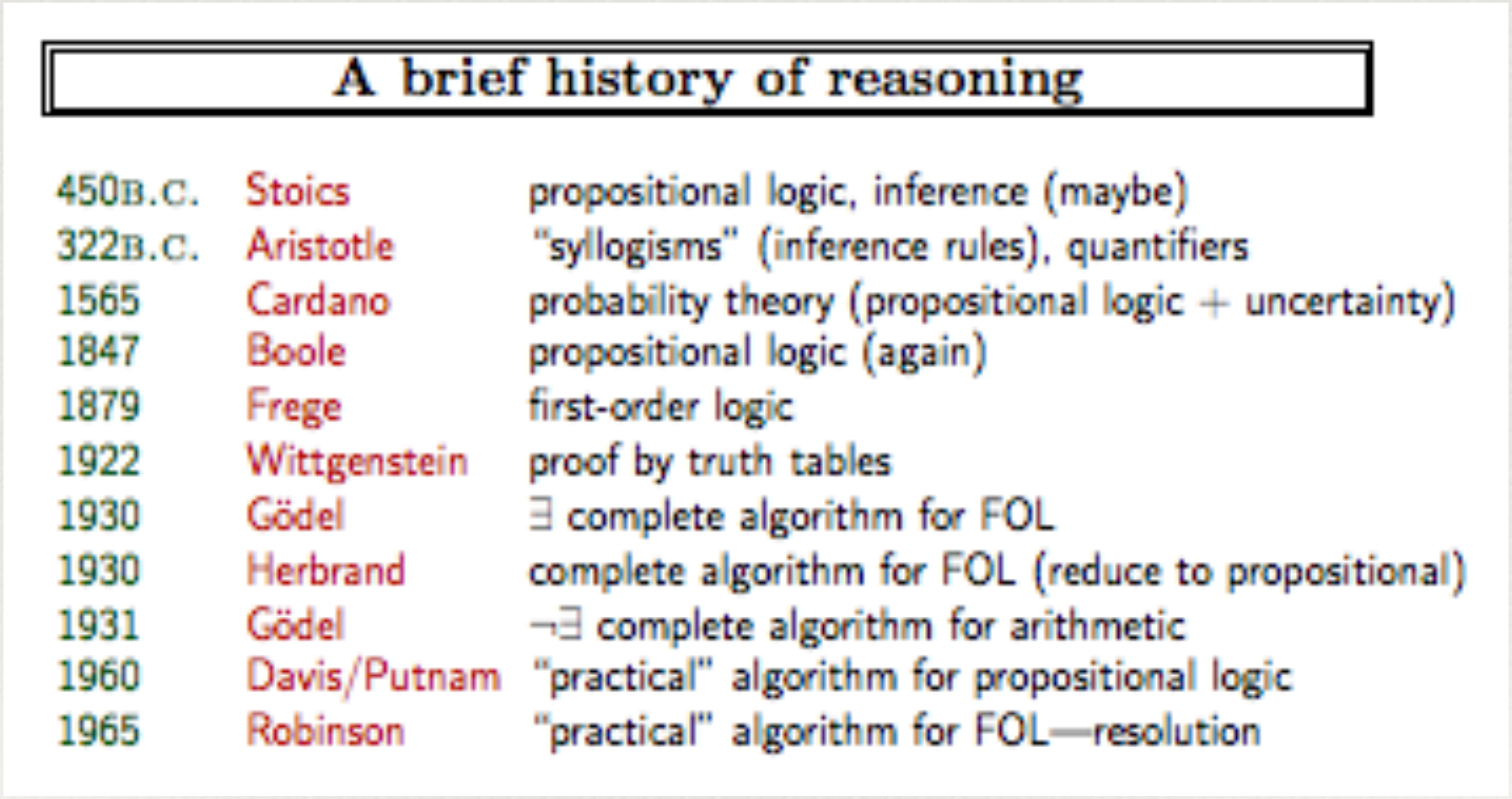
◇ Forward and backward chaining

◇ Logic programming

◇ Resolution

Stuart Russell, Univ. of California - Berkeley, autor do livro texto.

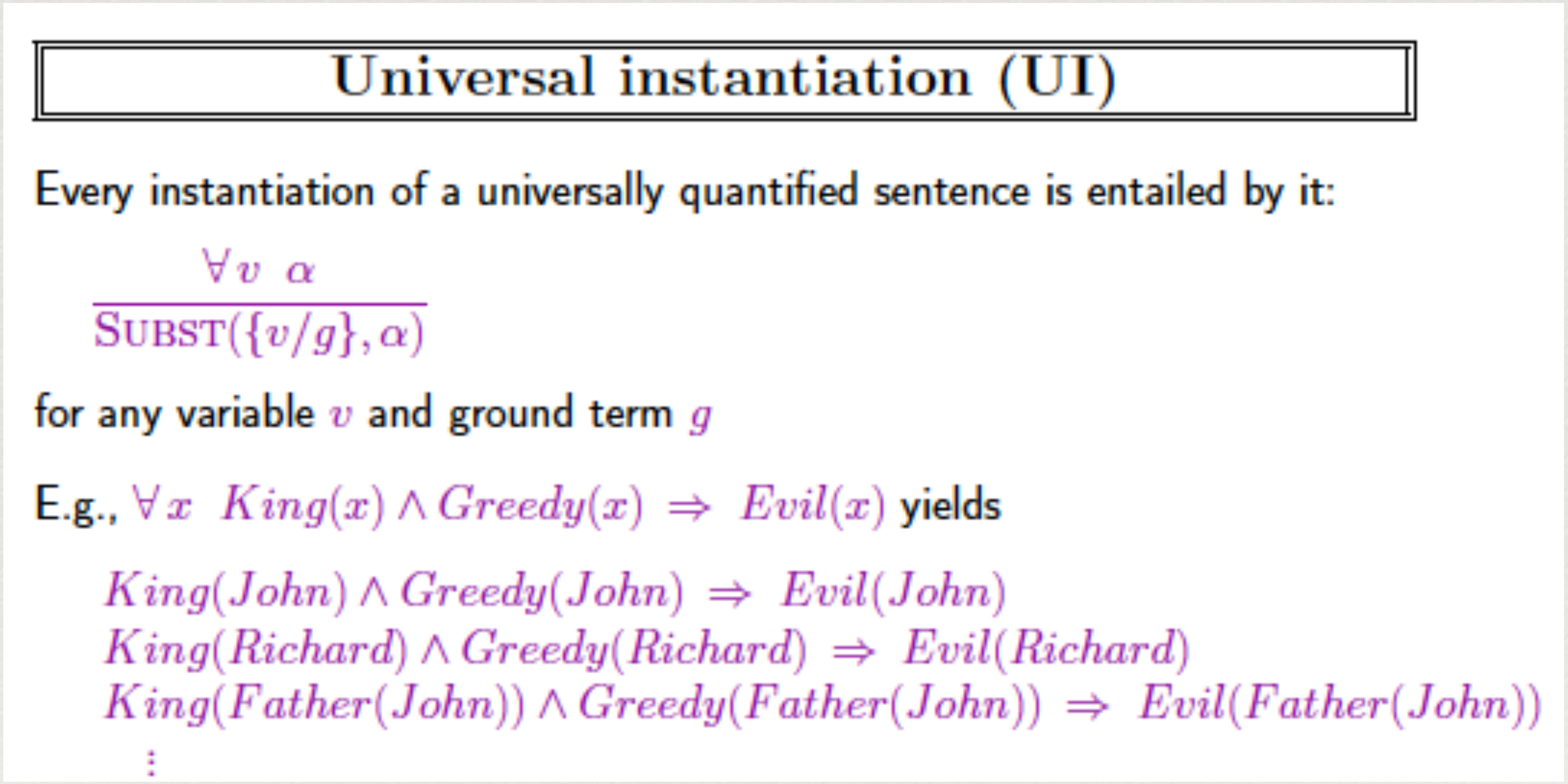




A brief history of reasoning

450B.C.	Stoics	propositional logic, inference (maybe)
322B.C.	Aristotle	"syllogisms" (inference rules), quantifiers
1565	Cardano	probability theory (propositional logic + uncertainty)
1847	Boole	propositional logic (again)
1879	Frege	first-order logic
1922	Wittgenstein	proof by truth tables
1930	Gödel	$\exists$ complete algorithm for FOL
1930	Herbrand	complete algorithm for FOL (reduce to propositional)
1931	Gödel	$\neg\exists$ complete algorithm for arithmetic
1960	Davis/Putnam	"practical" algorithm for propositional logic
1965	Robinson	"practical" algorithm for FOL—resolution

Stuart Russell, Univ. of California - Berkeley, autor do livro texto.



Universal instantiation (UI)

Every instantiation of a universally quantified sentence is entailed by it:

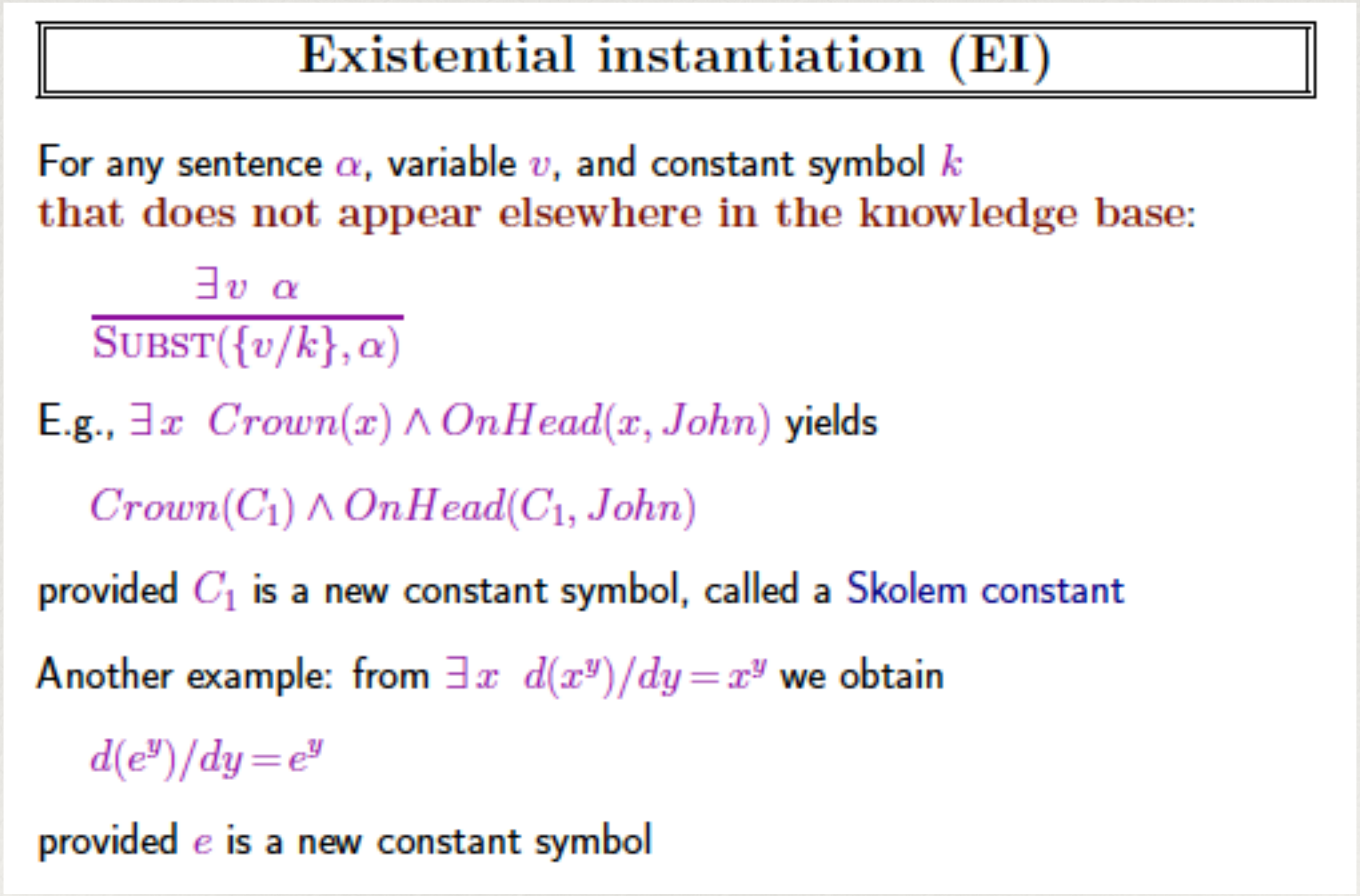
$$\frac{\forall v \;\; \alpha}{\textsc{Subst}(\{v/g\}, \alpha)}$$

for any variable v and ground term g

E.g., $\forall x \;\; King(x) \land Greedy(x) \Rightarrow Evil(x)$ yields

$$King(John) \land Greedy(John) \Rightarrow Evil(John)$$
$$King(Richard) \land Greedy(Richard) \Rightarrow Evil(Richard)$$
$$King(Father(John)) \land Greedy(Father(John)) \Rightarrow Evil(Father(John))$$
$$\vdots$$

Stuart Russell, Univ. of California – Berkeley, autor do livro texto.



Existential instantiation (EI)

For any sentence α, variable v, and constant symbol k
that does not appear elsewhere in the knowledge base:

$$\frac{\exists v \;\; \alpha}{\textsc{Subst}(\{v/k\}, \alpha)}$$

E.g., $\exists x \;\; Crown(x) \land OnHead(x, John)$ yields

$$Crown(C_1) \land OnHead(C_1, John)$$

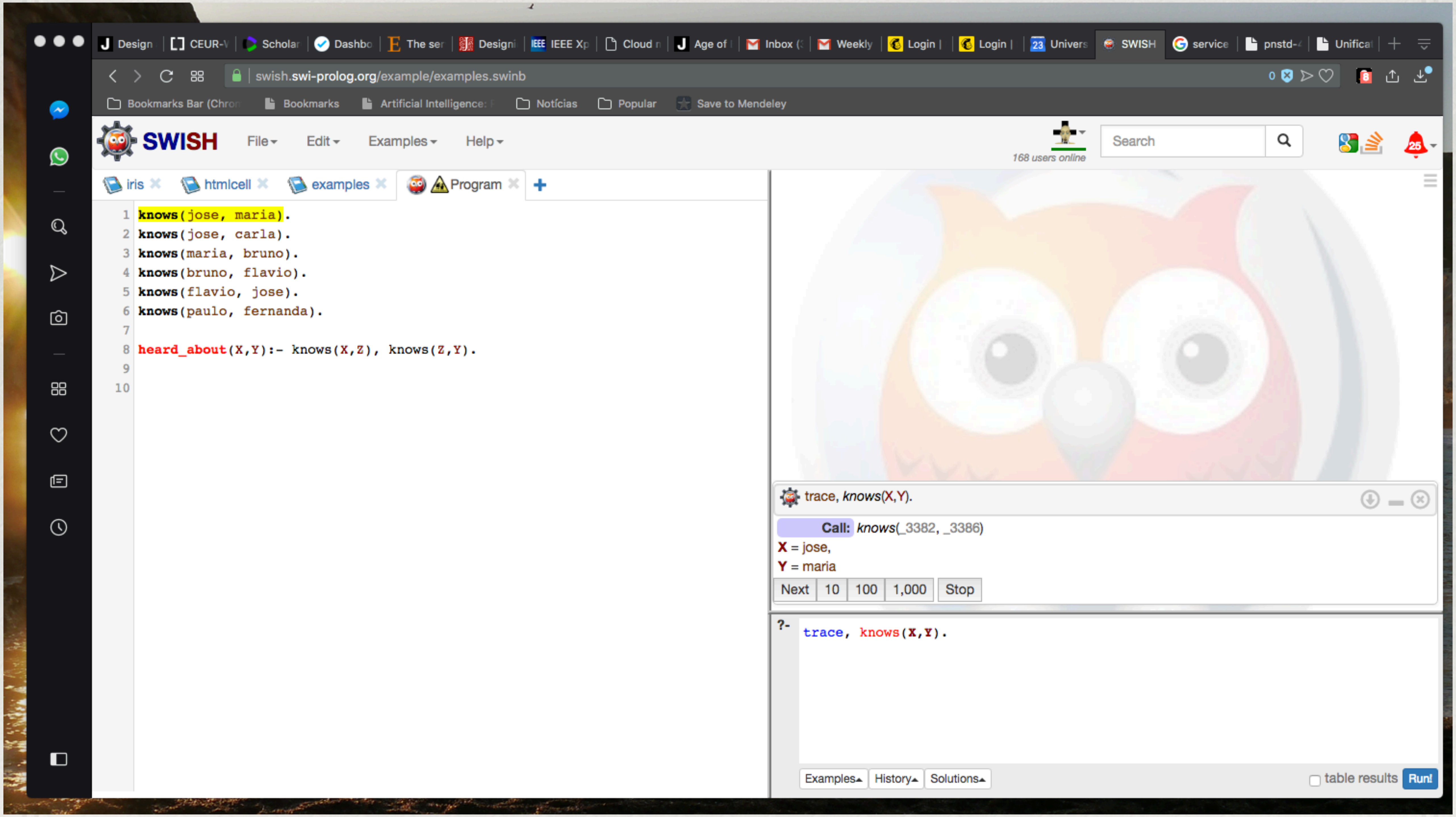
provided C_1 is a new constant symbol, called a Skolem constant

Another example: from $\exists x \;\; d(x^y)/dy = x^y$ we obtain

$$d(e^y)/dy = e^y$$

provided e is a new constant symbol

Stuart Russell, Univ. of California - Berkeley, autor do livro texto.

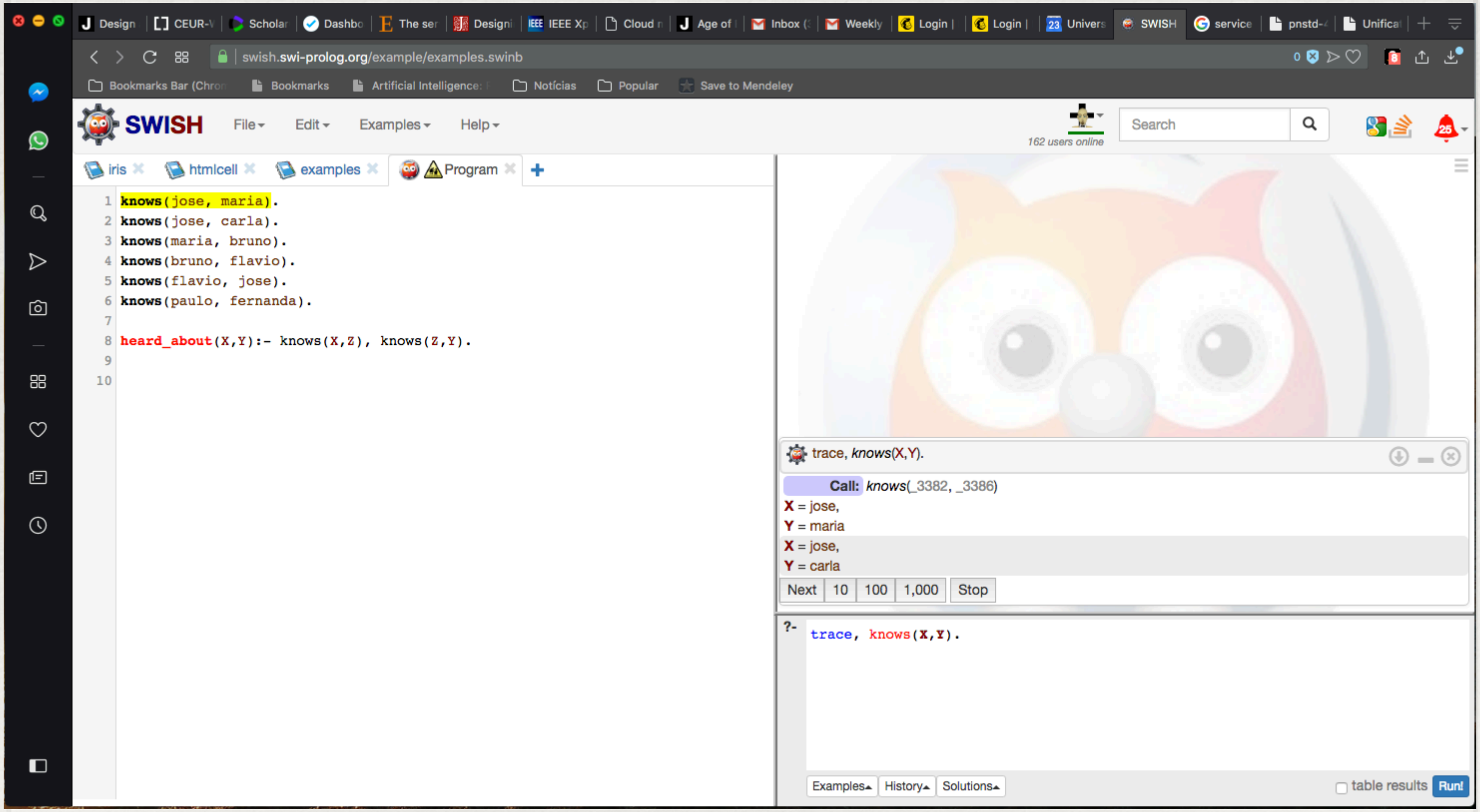


```
knows(jose, maria).
knows(jose, carla).
knows(maria, bruno).
knows(bruno, flavio).
knows(flavio, jose).
knows(paulo, fernanda).

heard_about(X,Y):- knows(X,Z), knows(Z,Y).
```

```
trace, knows(X,Y).
Call: knows(_3382, _3386)
X = jose,
Y = maria
```

```
?- trace, knows(X,Y).
```



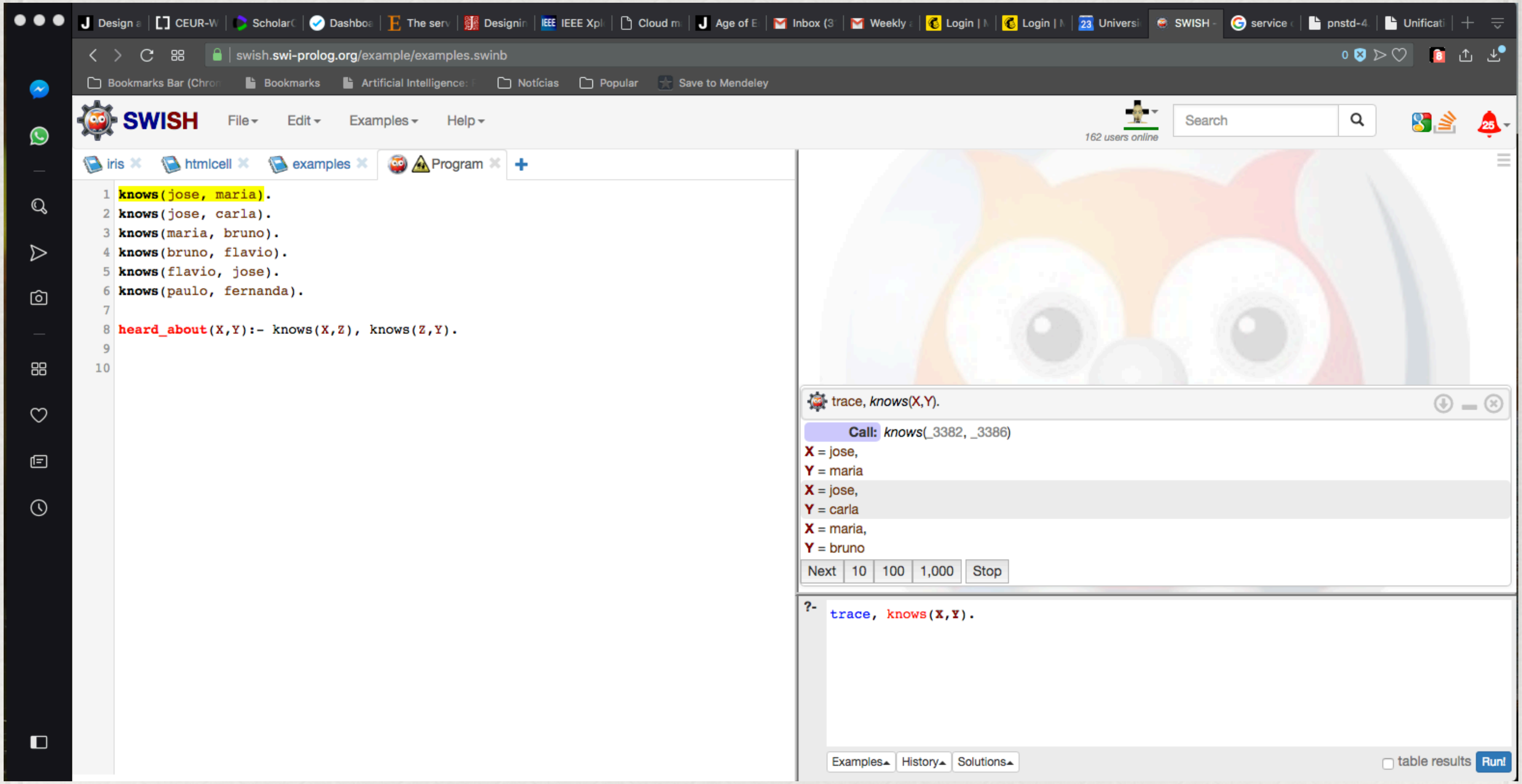
```
knows(jose, maria).
knows(jose, carla).
knows(maria, bruno).
knows(bruno, flavio).
knows(flavio, jose).
knows(paulo, fernanda).

heard_about(X,Y):- knows(X,Z), knows(Z,Y).
```

```
trace, knows(X,Y).

Call: knows(_3382, _3386)
X = jose,
Y = maria
X = jose,
Y = carla
```

```
?- trace, knows(X,Y).
```

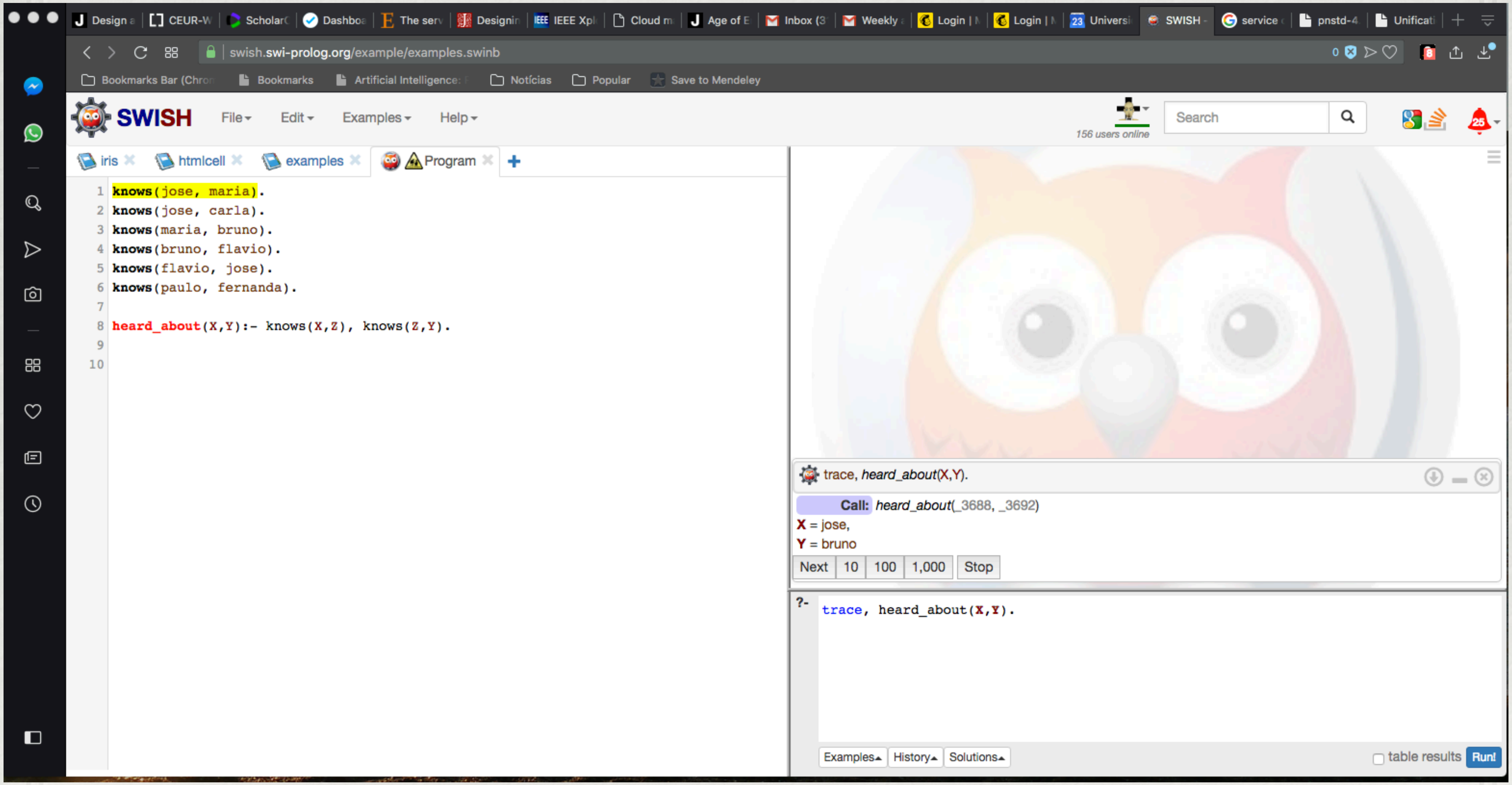


```
knows(jose, maria).
knows(jose, carla).
knows(maria, bruno).
knows(bruno, flavio).
knows(flavio, jose).
knows(paulo, fernanda).

heard_about(X,Y):- knows(X,Z), knows(Z,Y).
```

```
trace, knows(X,Y).
Call: knows(_3382, _3386)
X = jose,
Y = maria
X = jose,
Y = carla
X = maria,
Y = bruno
Next 10 100 1,000 Stop
```

```
?- trace, knows(X,Y).
```



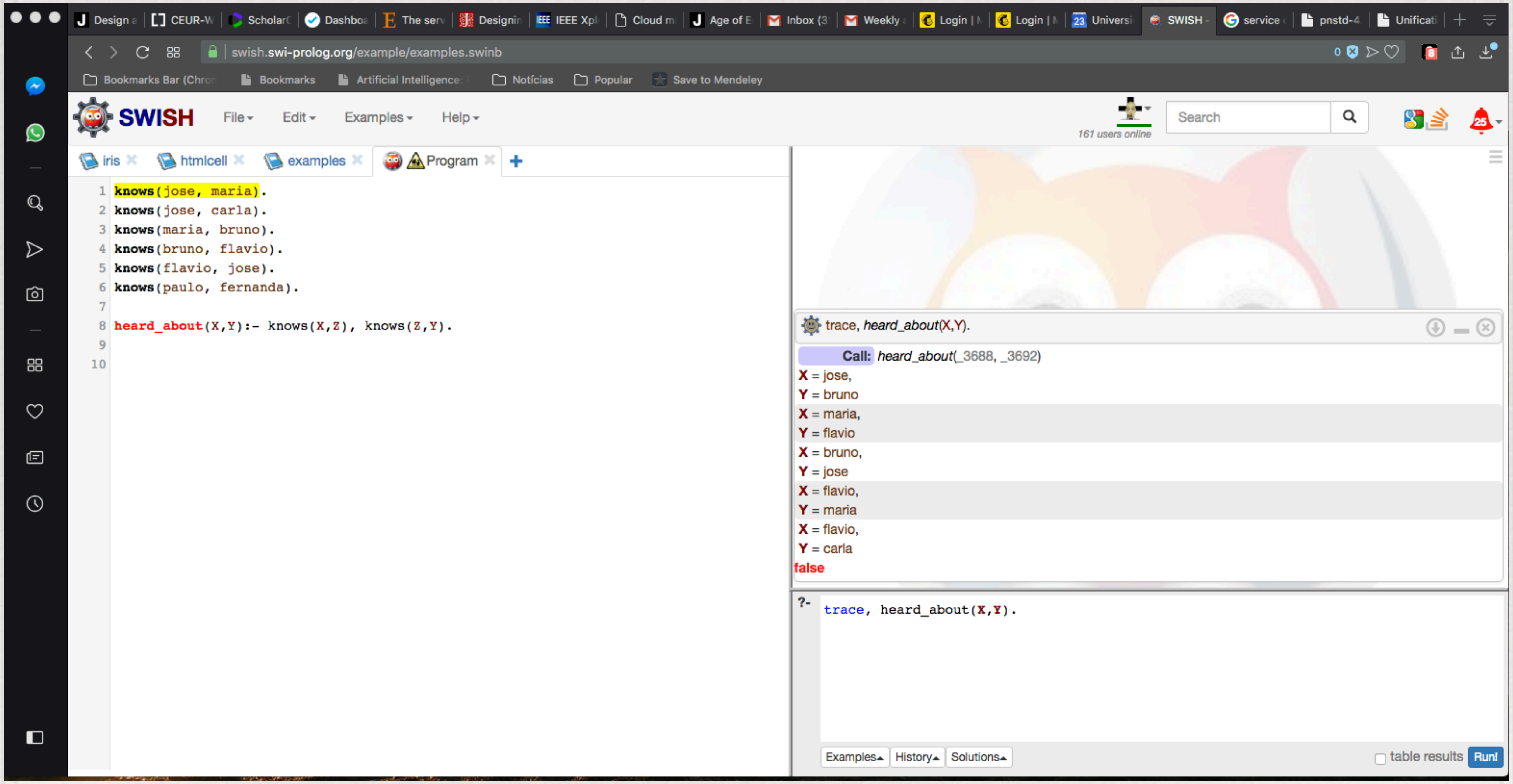
```
knows(jose, maria).
knows(jose, carla).
knows(maria, bruno).
knows(bruno, flavio).
knows(flavio, jose).
knows(paulo, fernanda).

heard_about(X,Y):- knows(X,Z), knows(Z,Y).
```

```
trace, heard_about(X,Y).
Call: heard_about(_3688, _3692)
X = jose,
Y = bruno
```

```
?- trace, heard_about(X,Y).
```

Stuart Russell, Univ. of California – Berkeley, autor do livro texto.



```
knows(jose, maria).
knows(jose, carla).
knows(maria, bruno).
knows(bruno, flavio).
knows(flavio, jose).
knows(paulo, fernanda).

heard_about(X,Y):- knows(X,Z), knows(Z,Y).
```

```
trace, heard_about(X,Y).
Call: heard_about(_3688, _3692)
X = jose,
Y = bruno
X = maria,
Y = flavio
X = bruno,
Y = jose
X = flavio,
Y = maria
X = flavio,
Y = carla
false
```

```
?- trace, heard_about(X,Y).
```

Resumindo... o processo de unificação consiste em achar uma unificação $\theta(X/s, Y/g, \ldots W/n)$ onde a KB possa ser interpretada como verdadeira.

Um agente inteligente

Em cada estado o agente percebe o ambiente → Infere qual a melhor ação em função desta informação → Executa a ação

memória

Proof methods

Proof methods divide into (roughly) two kinds:

Application of inference rules

– Legitimate (sound) generation of new sentences from old

– Proof = a sequence of inference rule applications

  Can use inference rules as operators in a standard search alg.

– Typically require translation of sentences into a normal form

Model checking

truth table enumeration (always exponential in n)

improved backtracking, e.g., Davis–Putnam–Logemann–Loveland

heuristic search in model space (sound but incomplete)

  e.g., min-conflicts-like hill-climbing algorithms

Forward and backward chaining

Horn Form (restricted)

KB = conjunction of Horn clauses

Horn clause =

◇ proposition symbol; or

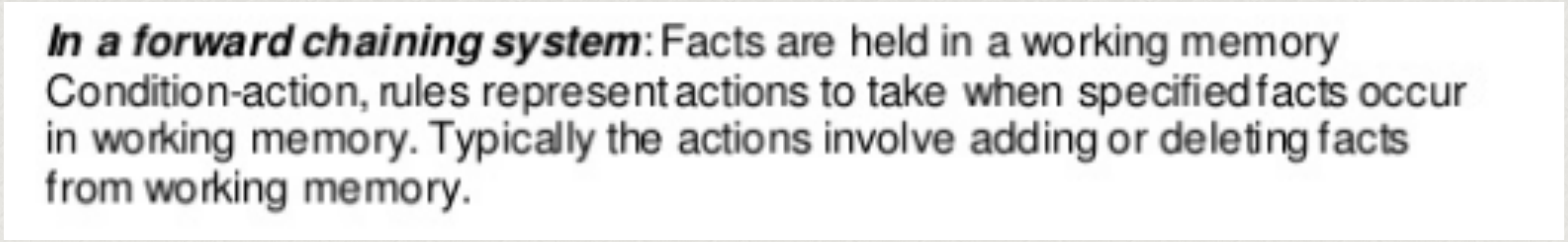
◇ (conjunction of symbols) $\Rightarrow$ symbol

E.g., $C \wedge (B \Rightarrow A) \wedge (C \wedge D \Rightarrow B)$

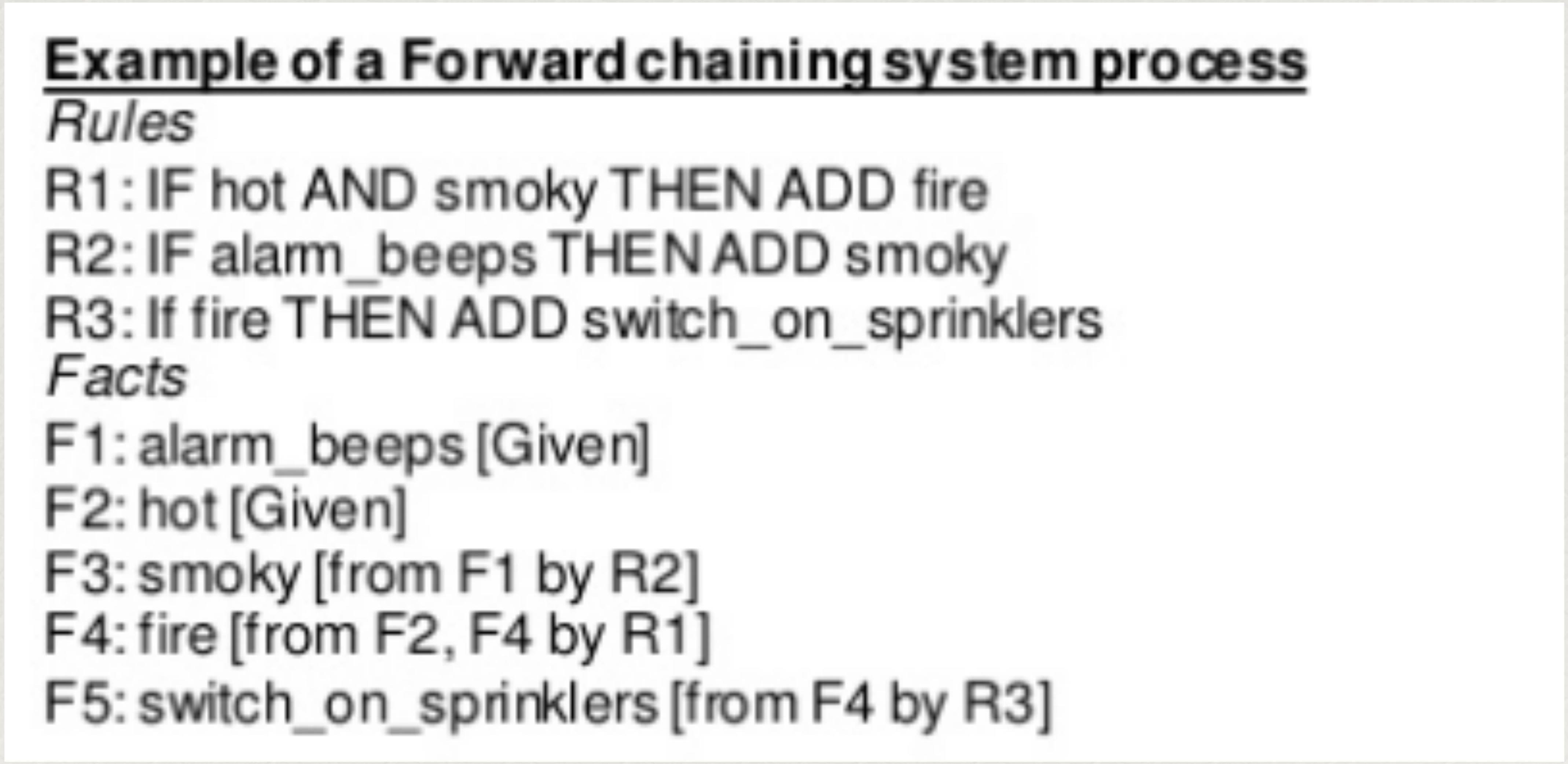
Modus Ponens (for Horn Form): complete for Horn KBs

$$\frac{\alpha_1, \ldots, \alpha_n, \qquad \alpha_1 \wedge \cdots \wedge \alpha_n \Rightarrow \beta}{\beta}$$

Can be used with forward chaining or backward chaining.
These algorithms are very natural and run in linear time



In a forward chaining system: Facts are held in a working memory Condition-action, rules represent actions to take when specified facts occur in working memory. Typically the actions involve adding or deleting facts from working memory.



Example of a Forward chaining system process

Rules

R1: IF hot AND smoky THEN ADD fire
R2: IF alarm_beeps THEN ADD smoky
R3: If fire THEN ADD switch_on_sprinklers

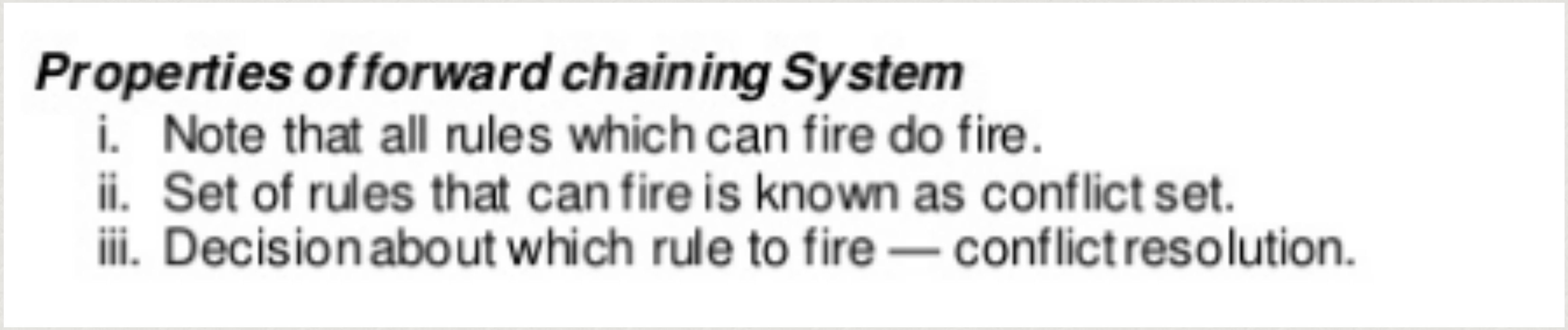
Facts

F1: alarm_beeps [Given]
F2: hot [Given]
F3: smoky [from F1 by R2]
F4: fire [from F2, F4 by R1]
F5: switch_on_sprinklers [from F4 by R3]

Intelligent Fire Alarm: forward chaining with multiple actions:

1. If smoky is detected go to alarm_mode_1
2. If in alarm_mode_1 then {check(temp1), wait(10), check(temp2) and if temp2 > temp1 go to alarm_mode_2}
3. If in alarm_mode_2 then if visual_detect(flames) go to alarm_mode_3
4. If in alarm_mode_3 then {turn_on(sprinklers), close(elevators), close(cutting_fire_doors), sound_alerts, call(fire_force)}.

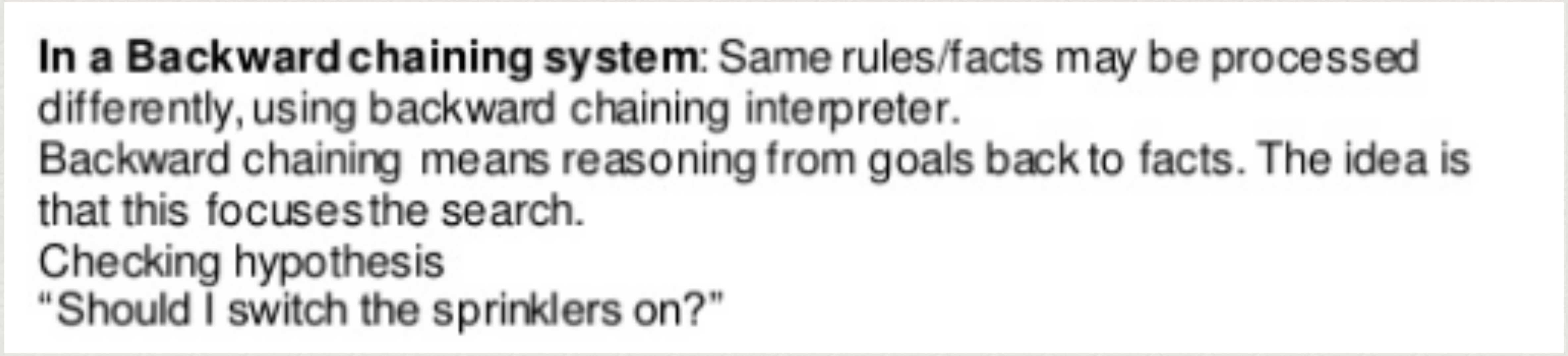
Stuart Russell, Univ. of California - Berkeley, autor do livro texto.



Properties of forward chaining System

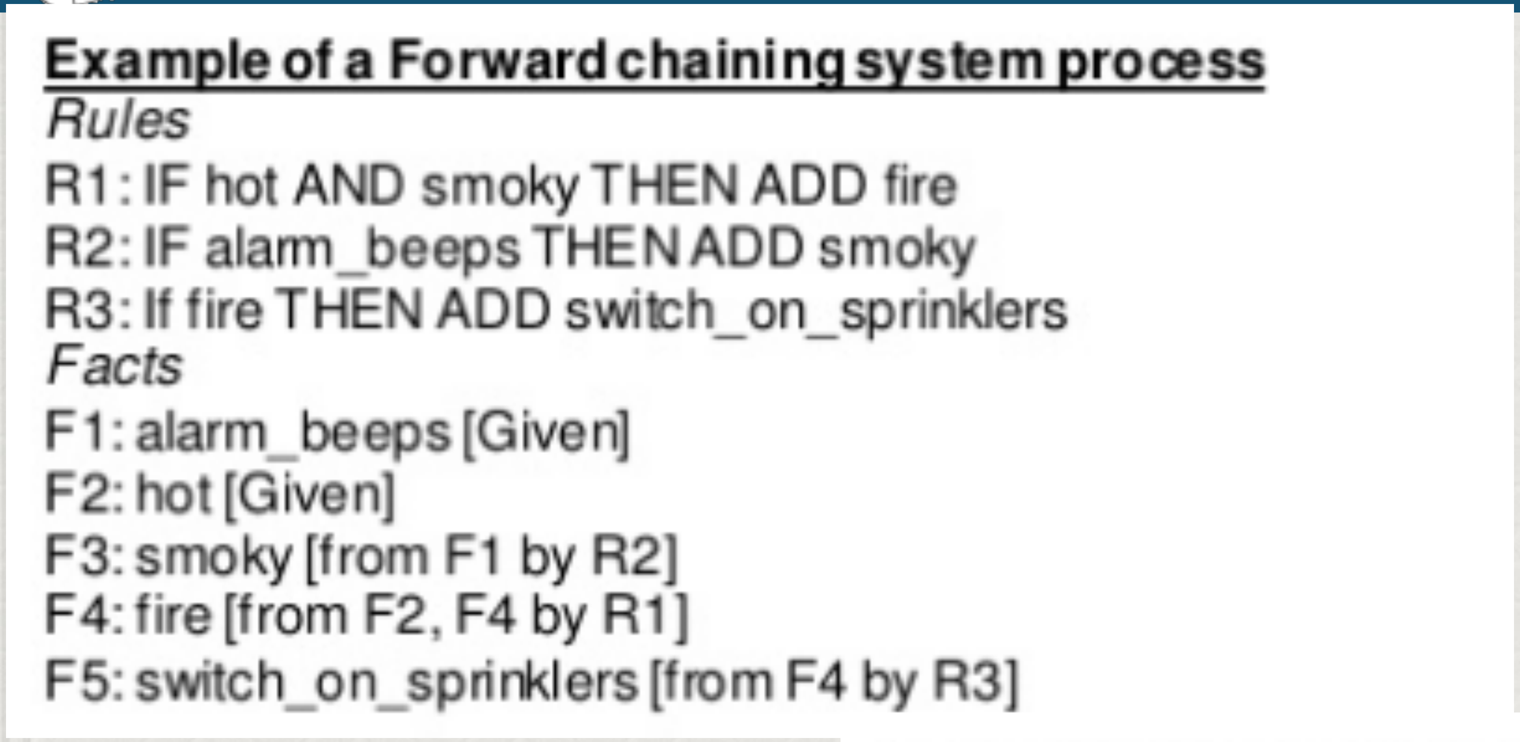
i. Note that all rules which can fire do fire.
ii. Set of rules that can fire is known as conflict set.
iii. Decision about which rule to fire — conflict resolution.

Stuart Russell, Univ. of California - Berkeley, autor do livro texto.



In a Backward chaining system: Same rules/facts may be processed differently, using backward chaining interpreter.
Backward chaining means reasoning from goals back to facts. The idea is that this focuses the search.
Checking hypothesis
"Should I switch the sprinklers on?"

Stuart Russell, Univ. of California - Berkeley, autor do livro texto.



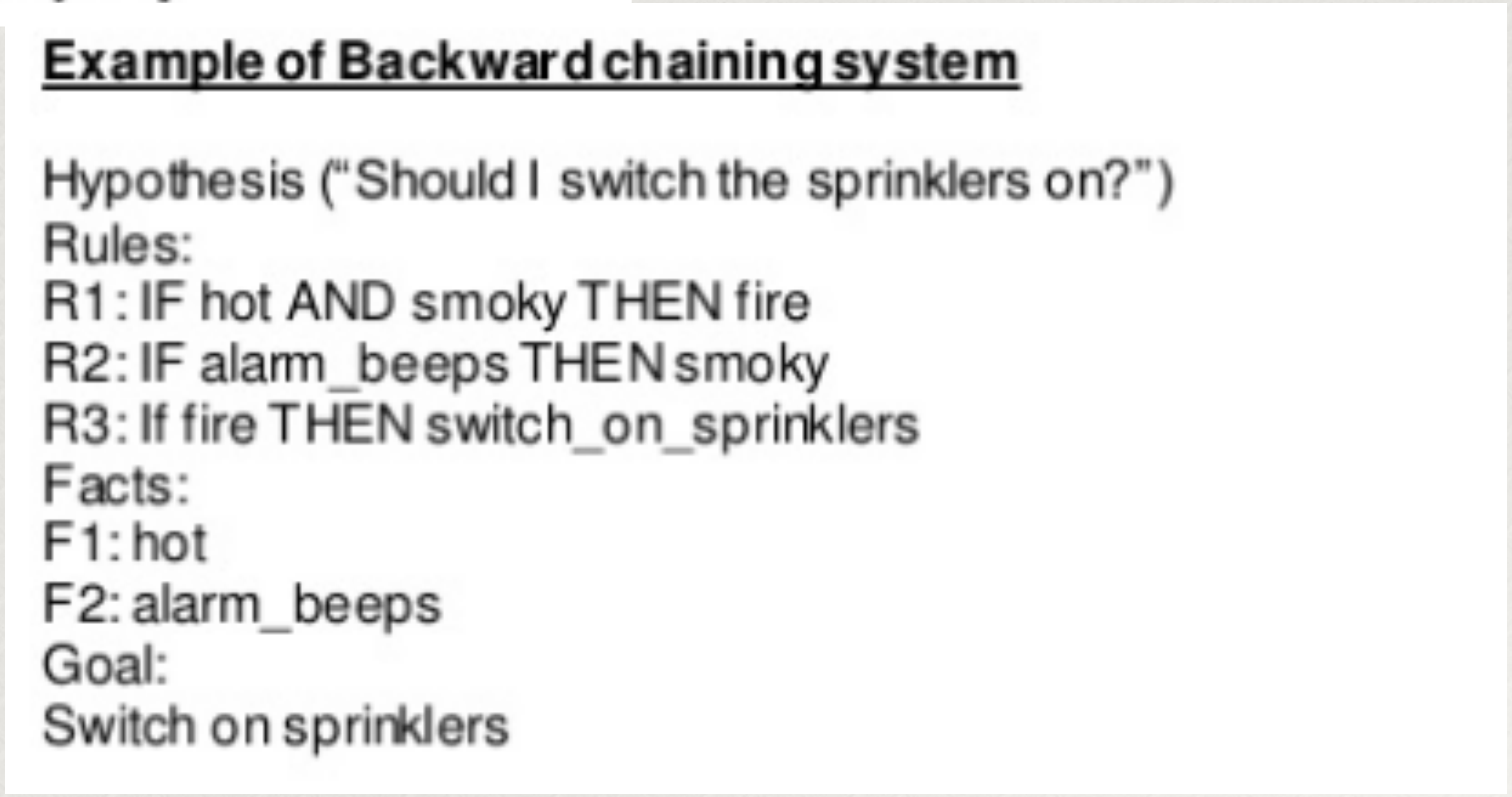
Example of a Forward chaining system process

Rules

R1: IF hot AND smoky THEN ADD fire
R2: IF alarm_beeps THEN ADD smoky
R3: If fire THEN ADD switch_on_sprinklers

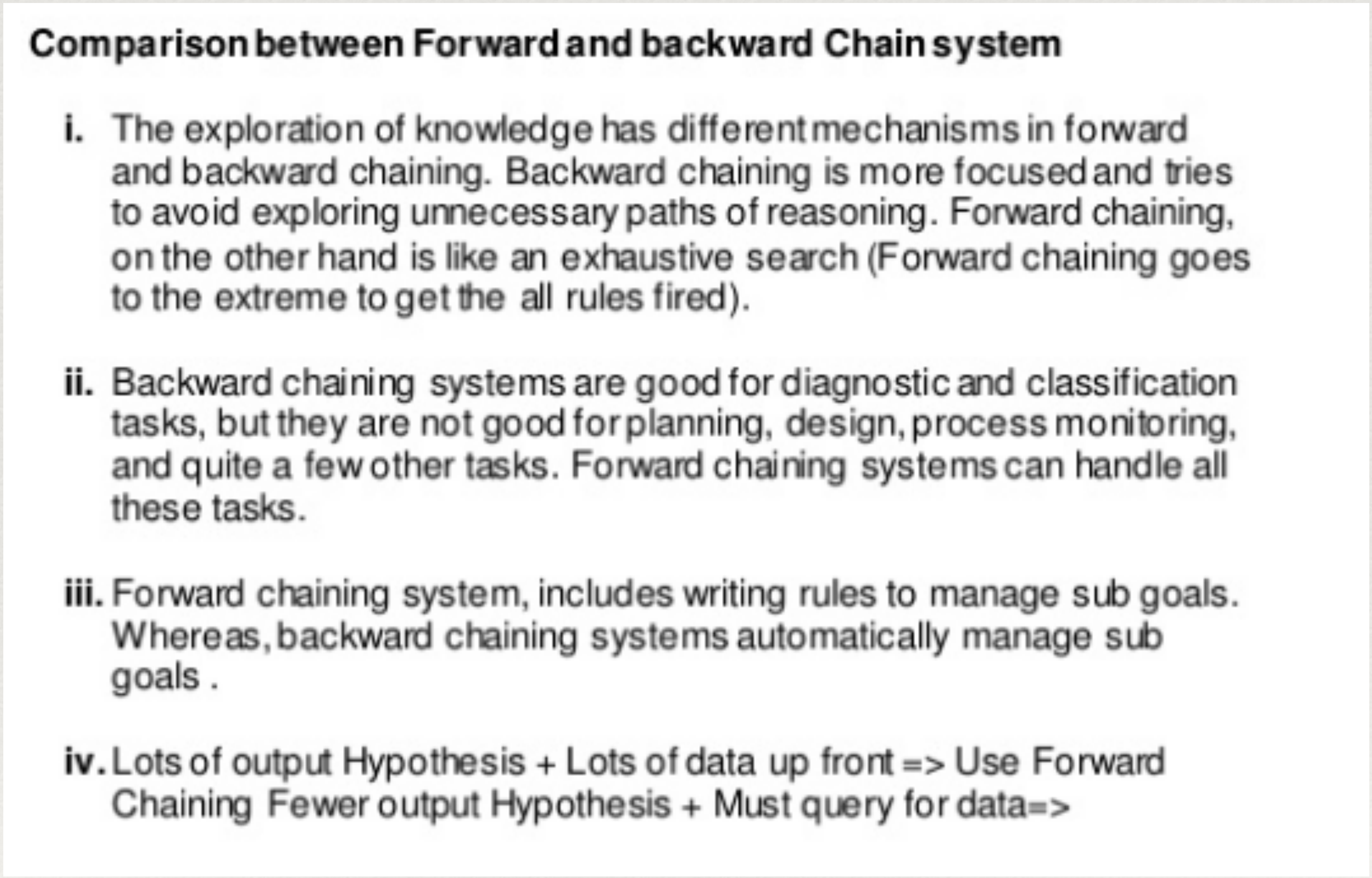
Facts

F1: alarm_beeps [Given]
F2: hot [Given]
F3: smoky [from F1 by R2]
F4: fire [from F2, F4 by R1]
F5: switch_on_sprinklers [from F4 by R3]



Example of Backward chaining system

Hypothesis ("Should I switch the sprinklers on?")
Rules:
R1: IF hot AND smoky THEN fire
R2: IF alarm_beeps THEN smoky
R3: If fire THEN switch_on_sprinklers
Facts:
F1: hot
F2: alarm_beeps
Goal:
Switch on sprinklers



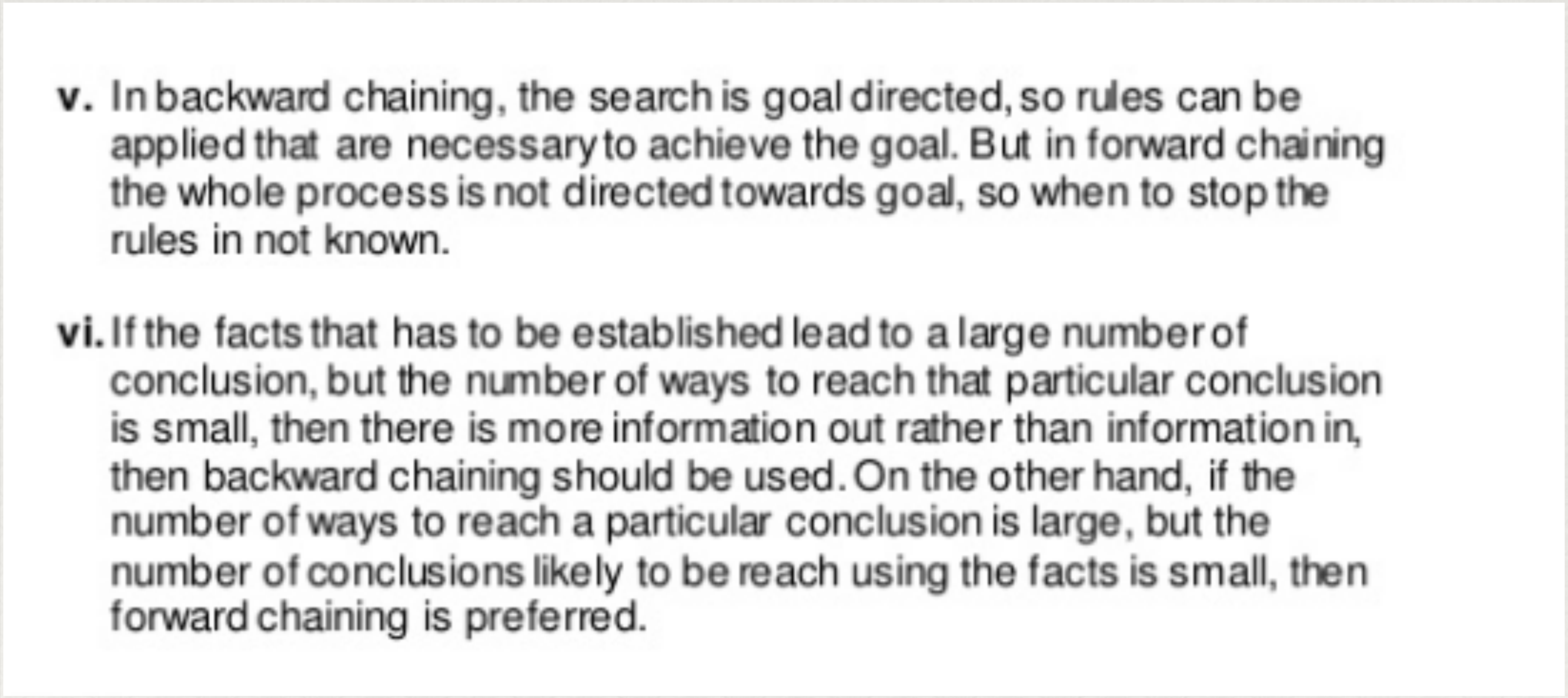
Comparison between Forward and backward Chain system

i. The exploration of knowledge has different mechanisms in forward and backward chaining. Backward chaining is more focused and tries to avoid exploring unnecessary paths of reasoning. Forward chaining, on the other hand is like an exhaustive search (Forward chaining goes to the extreme to get the all rules fired).

ii. Backward chaining systems are good for diagnostic and classification tasks, but they are not good for planning, design, process monitoring, and quite a few other tasks. Forward chaining systems can handle all these tasks.

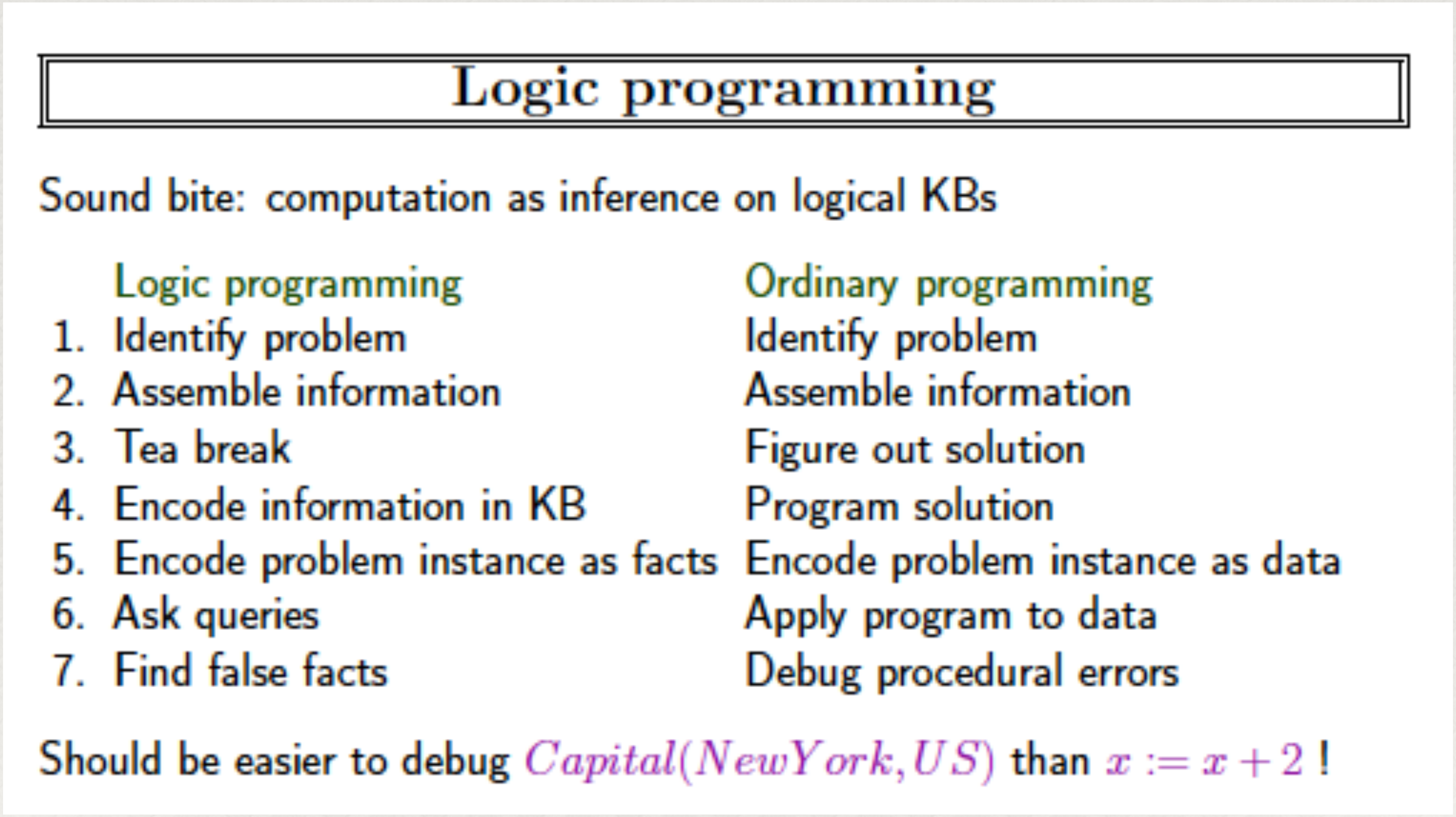
iii. Forward chaining system, includes writing rules to manage sub goals. Whereas, backward chaining systems automatically manage sub goals .

iv. Lots of output Hypothesis + Lots of data up front => Use Forward Chaining Fewer output Hypothesis + Must query for data=>



v. In backward chaining, the search is goal directed, so rules can be applied that are necessary to achieve the goal. But in forward chaining the whole process is not directed towards goal, so when to stop the rules in not known.

vi. If the facts that has to be established lead to a large number of conclusion, but the number of ways to reach that particular conclusion is small, then there is more information out rather than information in, then backward chaining should be used. On the other hand, if the number of ways to reach a particular conclusion is large, but the number of conclusions likely to be reach using the facts is small, then forward chaining is preferred.



Logic programming

Sound bite: computation as inference on logical KBs

	Logic programming	Ordinary programming
1.	Identify problem	Identify problem
2.	Assemble information	Assemble information
3.	Tea break	Figure out solution
4.	Encode information in KB	Program solution
5.	Encode problem instance as facts	Encode problem instance as data
6.	Ask queries	Apply program to data
7.	Find false facts	Debug procedural errors

Should be easier to debug $Capital(NewYork, US)$ than $x := x + 2$!

Prolog

- Linguagem de programação lógica e declarativa
- Foi criado em 1970 por R. Kowalski e Maarten van Emden em Edimburgo e Alain Colmerauer em Marsailles

algoritmo = lógica + controle

- Junto com Lisp, são as linguagens de programação simbólica mais usadas em Inteligência Artificial

Prolog

- Origem: pesquisas em provadores automáticos de teoremas e sistemas de dedução dos anos 60 e 70
- Faz inferência a partir de um processo de refutação (resolução).
  - Cláusulas de Horn com encadeamento para trás
- Usuários: *algumas centenas de milhares*

Sintaxe: *símbolos*

- Variáveis são representadas por ***letras maiúsculas*** (A, B, …) ou "_" (chamada de variável anônima)
- Símbolos funcionais (funtores) e símbolos relacionais (predicados) são representa-dos por ***letras minúsculas***. Exemplos:
  - p(X, Y, _) : aridade 3 (p/3)
  - q : aridade 0, i.e., constante

Sintaxe: *cláusula*

- São escritas na forma de uma implicação invertida:

  H :- B

  em que **H** é uma conclusão (átomo ou literal positivo chamado de cabeça), **B** é um conjunto de premissas (corpo), i.e., é uma conjunção de átomos e **:-** é a implicação clássica invertida **(**←**)**

  H :- (*cláusula unária ou fato*)

  :- B (*cláusula objetivo ou consulta*)
- Se uma variável aparecer só em B , ela é existencialmente quantificada
- Se uma variável aparecer em H, ela é universalmente quantificada

Sintaxe

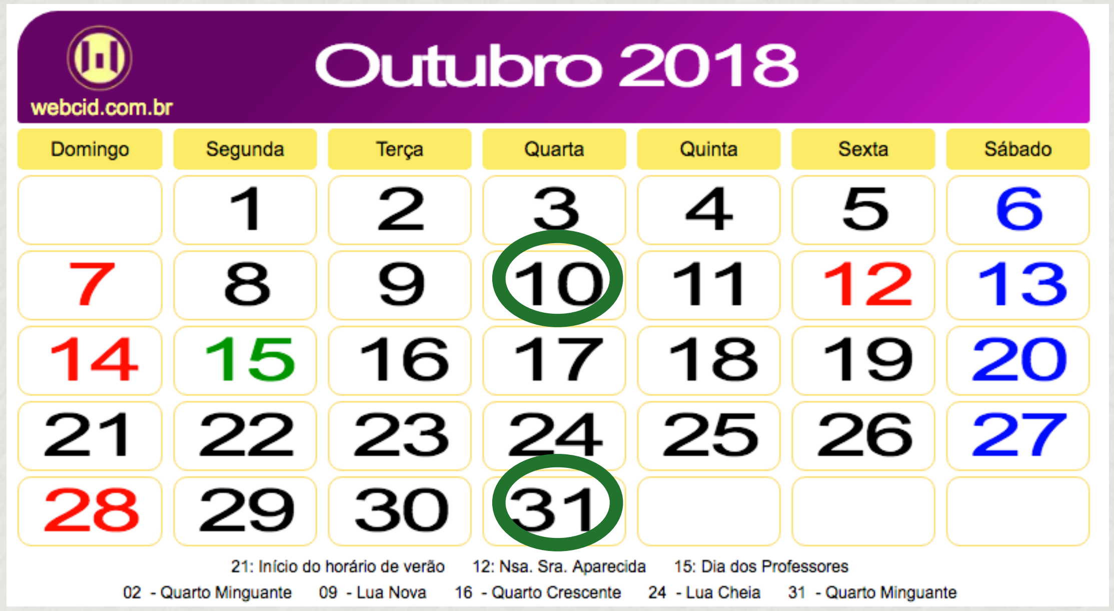
- Conjunção: ‘,’
- Disjunção: ‘;’
- Toda cláusula deve terminar com um ponto
- Elementos não lógicos:
  - negação: \+ ou not()
  - cut: !
  - assert, retract, etc...

Definição de programas

- Programas Prolog puros: conjuntos de cláusulas sem elementos extra-lógicos
- Programa prolog generalizado: conjunto de cláusulas com negação

Até a próxima aula!

A competição entre equipes

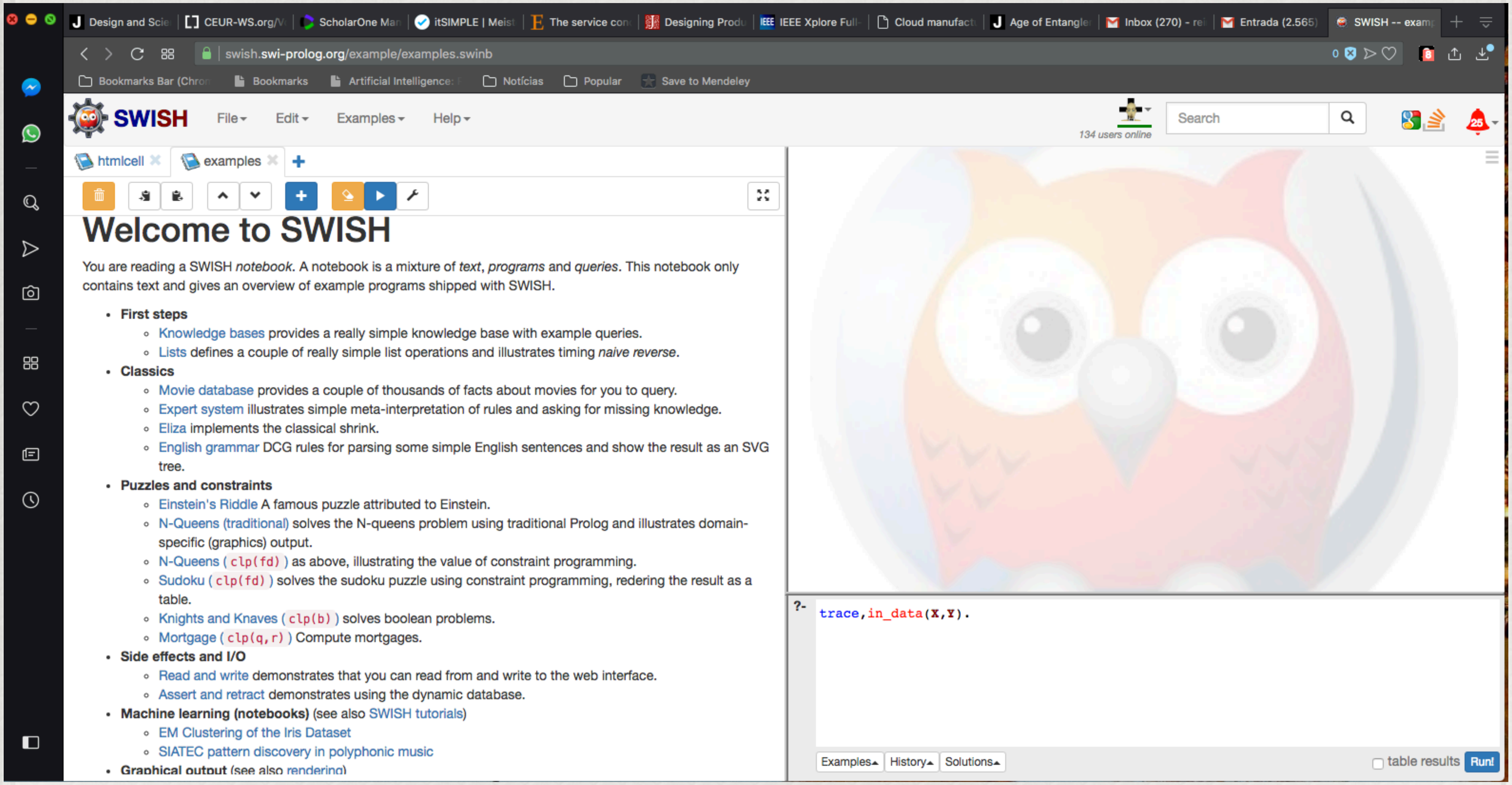

webcid.com.br

Outubro 2018

Domingo	Segunda	Terça	Quarta	Quinta	Sexta	Sábado
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

21: Início do horário de verão 12: Nsa. Sra. Aparecida 15: Dia dos Professores
02 - Quarto Minguante 09 - Lua Nova 16 - Quarto Crescente 24 - Lua Cheia 31 - Quarto Minguante

No dia 31, segundo o nosso cronograma, teremos o desfecho do trabalho em grupo e faremos isso na forma de uma competição onde todos os grupos deverão resolver o mesmo problema dado como estado inicial. Todos devem ter o seu programa instalado no sistema Swish e pronto para rodar sua implementação do A*.

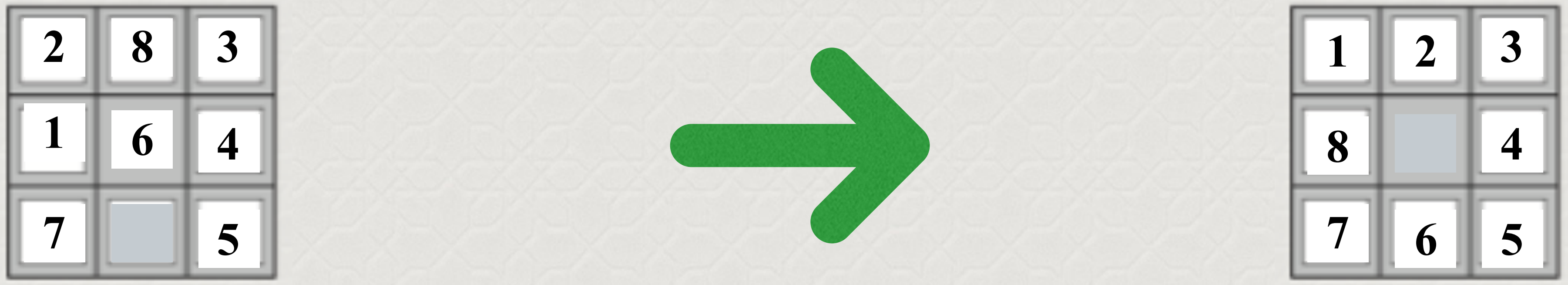


Welcome to SWISH

You are reading a SWISH *notebook*. A notebook is a mixture of *text*, *programs* and *queries*. This notebook only contains text and gives an overview of example programs shipped with SWISH.

- **First steps**
  - Knowledge bases provides a really simple knowledge base with example queries.
  - Lists defines a couple of really simple list operations and illustrates timing *naive reverse*.
- **Classics**
  - Movie database provides a couple of thousands of facts about movies for you to query.
  - Expert system illustrates simple meta-interpretation of rules and asking for missing knowledge.
  - Eliza implements the classical shrink.
  - English grammar DCG rules for parsing some simple English sentences and show the result as an SVG tree.
- **Puzzles and constraints**
  - Einstein's Riddle A famous puzzle attributed to Einstein.
  - N-Queens (traditional) solves the N-queens problem using traditional Prolog and illustrates domain-specific (graphics) output.
  - N-Queens (`clp(fd)`) as above, illustrating the value of constraint programming.
  - Sudoku (`clp(fd)`) solves the sudoku puzzle using constraint programming, redering the result as a table.
  - Knights and Knaves (`clp(b)`) solves boolean problems.
  - Mortgage (`clp(q,r)`) Compute mortgages.
- **Side effects and I/O**
  - Read and write demonstrates that you can read from and write to the web interface.
  - Assert and retract demonstrates using the dynamic database.
- **Machine learning (notebooks)** (see also SWISH tutorials)
  - EM Clustering of the Iris Dataset
  - SIATEC pattern discovery in polyphonic music
- **Graphical output** (see also rendering)

```
?- trace,in_data(X,Y).
```



g(0) = 0 + 4

estado final

O estado final será o mesmo que usamos na aula passada no exemplo de resolução da busca baseada no algoritmo Best-first.

O procedimento

- Todos devem estar preparados com o Swish e o seu algoritmo, mas sem tocar no teclado ou no mouse;
- O estado inicial será colocado em um slide;
- Todos esperam o "sinal de largada" para começar o processo;
- Devem inserir o estado inicial no programa e resolvê-lo;
- Chegando ao estado final o programa deve descrever este estado e colocar na tela o custo da busca (soma de g(x)).
- com esta informação na saída do programa a equipe faz um sinal de termino e o tempo é anotado.

Após a competição

Após a competição cada equipe deve fazer o upload no sistema e-disciplinas de um texto que descreve a heurística (g(x) e h(x)), suas propriedades, e uma listagem do código Prolog. A nota final será a média da nota atribuída a este documento e a nota da competição.

A nota da competição

A nota da competição será a composição da classificação por tempo (de zero a cinco), e cinco pontos se a equipe conseguiu resolver mesmo que em um tempo mais longo, e zero se não fechou o processo de busca.

Trabalho em grupo

Os grupos estão já definidos e registrados no e-disciplinas. A configuração é a seguinte:

Grupo 1:
Alyson Akio Haro
Alexandre Inoue
Eduardo Kose
Vitor Fukuda

Grupo 2:
Alex Majima
Gabriel Ferreira
Lucas de Angelis
Thiago Ferraz

Grupo 3:
Gabriel Tutia
Lucas Palopoli
Pedro dos Santos Melo
Samuel Monção

Grupo 4:
Gustavo Novello
Luiz Guilherme Sabino
Alessandro Ezequiel Junior
Gabriel Negre

Grupo 5:
Gabriel Yida
Vinicius Santiago
Danilo Polidoro

Grupo 6:
Guilherme Aires de França
Henrique Peterlevitz
Wagner Geraldo Ferreira