

# 9. 知識表現と推論

(人工知能)

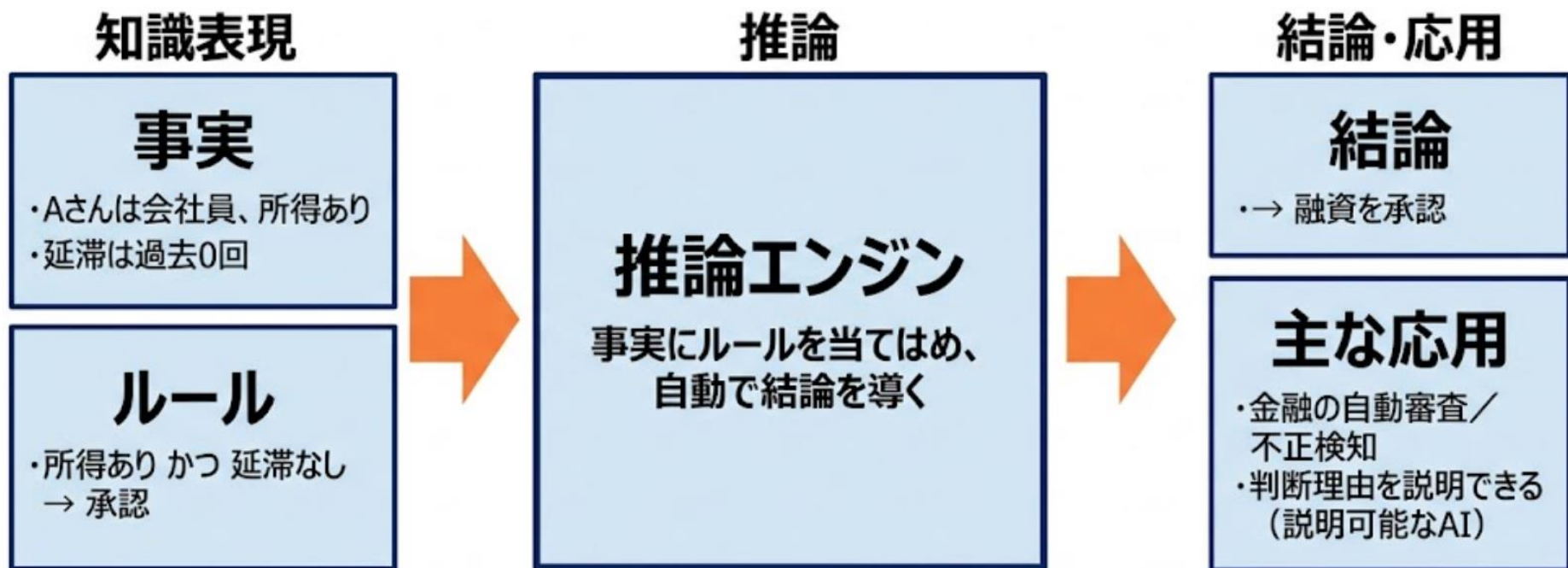
金子邦彦



# 「人工知能」第9回の内容



AIは『事実』と『ルール』を組み合わせ、推論で自動的に結論を導く



## 機械学習との連携

大量データ

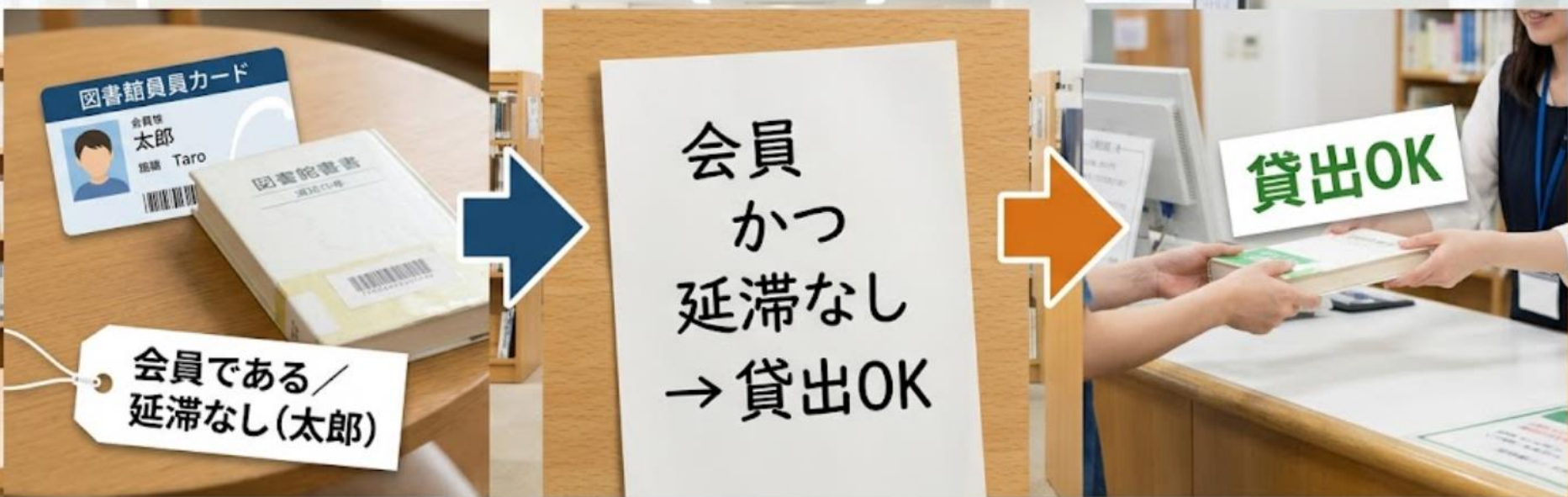
機械学習がルールを発見

推論に適用

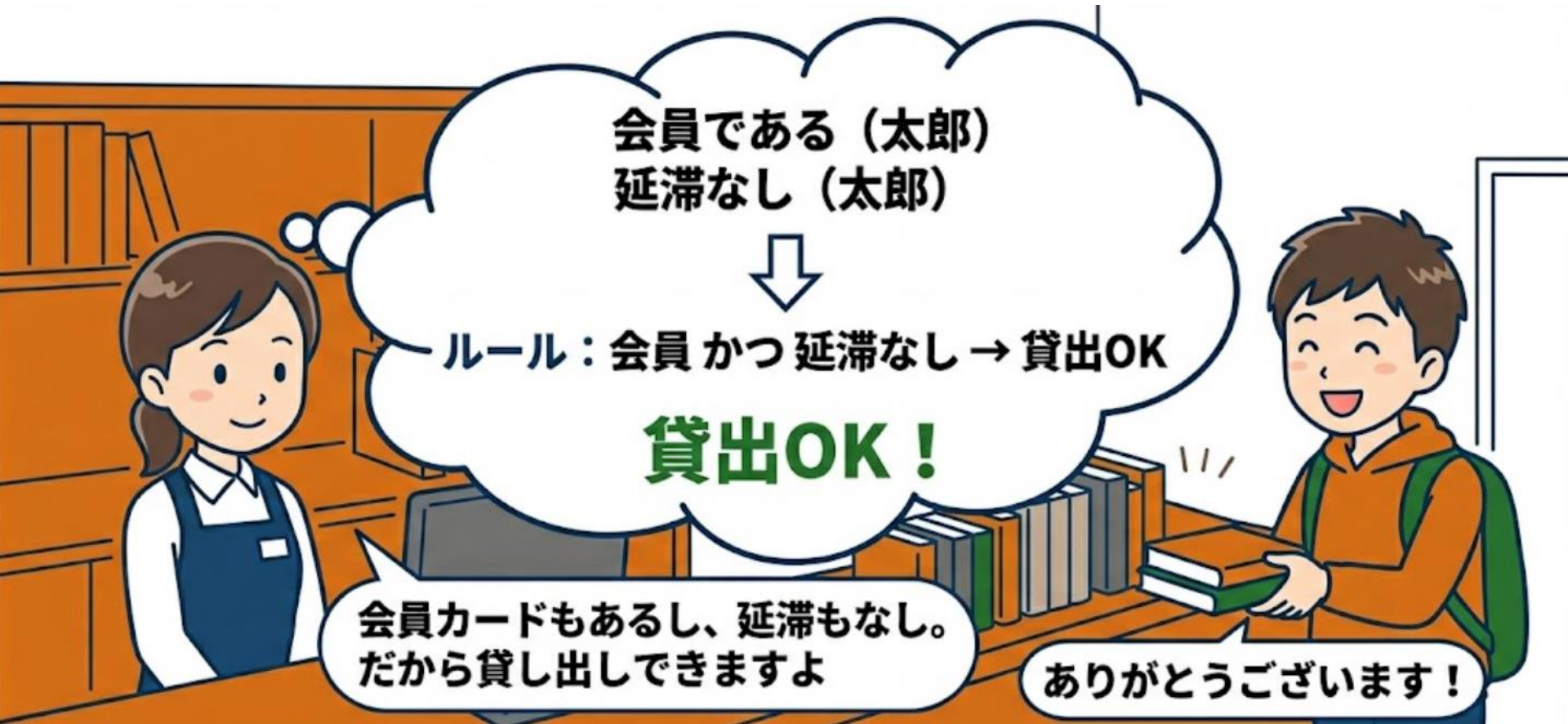
より高度で説明可能なAIへ



## 9-1. 知識表現と推論



**分かっている事実に、決められたルールを当てはめると、結論が決まる — これが推論**



会員である (太郎)  
延滞なし (太郎)



ルール：会員かつ延滞なし → 貸出OK

**貸出OK!**

会員カードもあるし、延滞もなし。  
だから貸し出しできますよ

ありがとうございます!

## 事実(太郎について分かっていること)



太郎



会員である(太郎)



延滞なし(太郎)

## ルール(どんな人なら借りられるか)



会員である



かつ

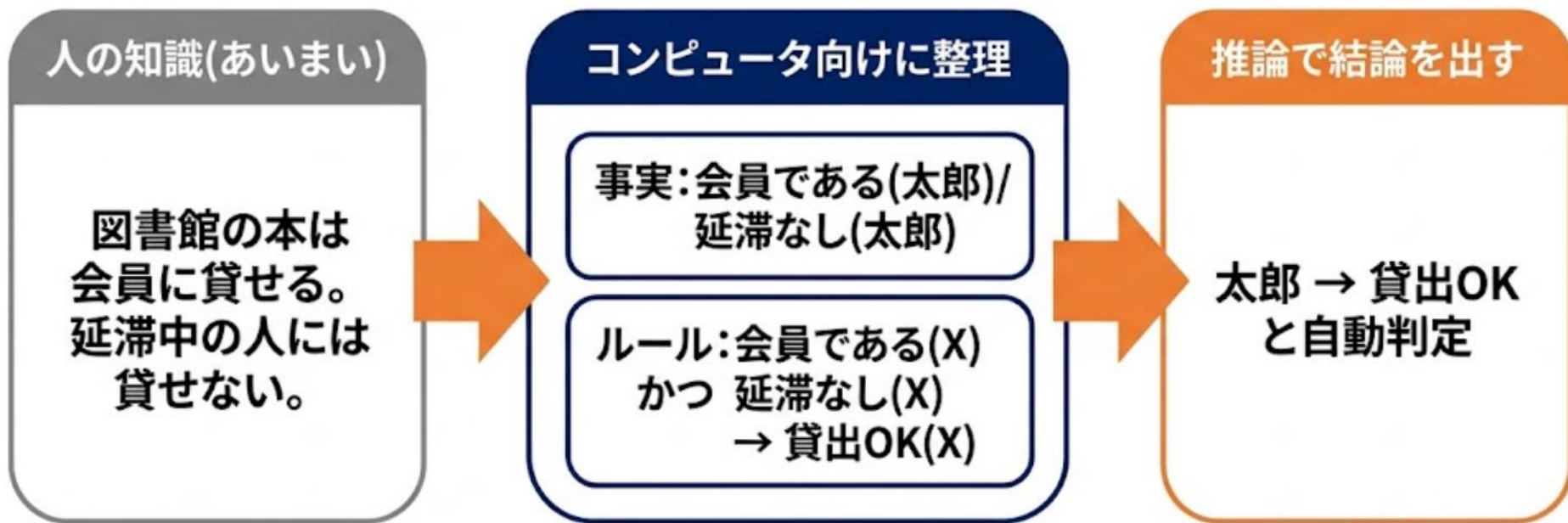


延滞なし



貸出OK

人のあいまいな知識を、コンピュータが扱える形に整理する



知識を【事実】と【ルール】に分けると、コンピュータは自分で結論を導ける



『事実』と『ルール』を照らし合わせ、成り立つか調べる — これが推論

# 推論する AI : 質問に自動で答える



AIは『事実』と『ルール』をもとに、質問が成り立つか調べる

『事実:会員である(太郎)/ 延滞なし(太郎)』  
『ルール:会員である(X) かつ  
延滞なし(X) → 貸出OK(X)』



推論AI

質問:貸出OK(太郎)?  True

太郎は会員かつ延滞なし。ルールが成り立つ

質問:貸出OK(X)?  X = 太郎

Xに当てはまるのは太郎だけ

質問:貸出OK(花子)?  False

花子が会員だという事実が無いので成り立たない(延滞しているわけではない)

※ False は『根拠が無い』という意味。この考え方は将来学ぶ Prolog につながる

## 事実とルールから 結論を導く(推論)

知識を使って推論し、結論が新しい知識として加わる

### 知識表現:事実とルール

- 事実(成り立っている事柄):  
会員である(太郎)/  
延滞なし(太郎)
- ルール(条件→結論):  
会員である(X) かつ  
延滞なし(X) → 貸出OK(X)

### 推論:結論を導く

質問:太郎に貸せる?

ルールを適用

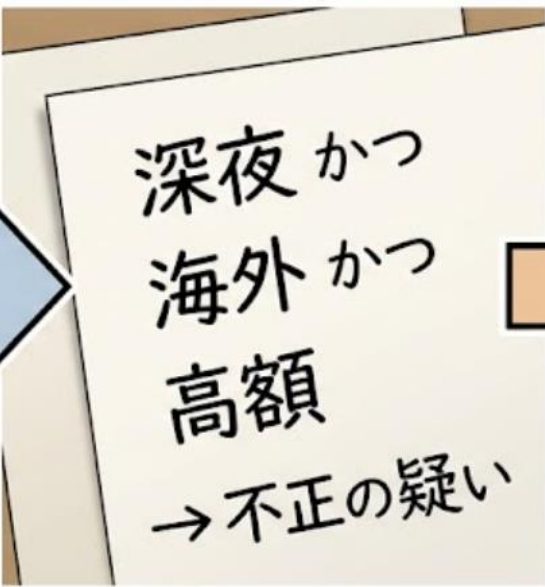
結論:貸出OK(太郎)

太郎は『定数』、Xは『変数』。Xには誰でも当てはめられる

**事実 =**  
ふだんの使い方の記録



**ルール =**  
怪しいと見なす条件



**推論 =**  
自動で判定した結論



**ふだんの記録 (事実) に怪しさの条件 (ルール) を当てはめ、  
自動で危険を判定する — これが推論**

# 知識表現と推論の身近な例



どれも『事実』に『ルール』を適用して、結論を自動で出している

	事実(持っている知識)	ルール(条件 → 結論)	役立つこと
<b>① Google検索 の情報パネル</b> 2012年～	「東京タワーは高さ333m、東京都港区にある」など、数億件の事実を保持	場所Aが都市Bにある かつ 都市BがX国にある → 場所AはX国にある	検索するだけで関連する事実がまとめて表示され、サイトを見て回る手間が省ける
<b>② クレジットカード の不正検知</b> 国内各社が導入	<b>事実</b> カード所有者の利用場所・時間帯・金額の傾向を事実として保持	<b>ルール</b> 普段は国内利用 かつ 深夜に海外で高額決済 → 不正の疑いあり	<b>役立つこと</b> 決済のたびに照合し、数ミリ秒でリスクを判定して不正を自動でブロック
<b>③ 金融・保険の 自動審査</b> 金融業で広く普及	<b>事実</b> 申込者の年齢・年収・勤続年数・ローン残高を事実として受け取る	<b>ルール</b> 年収300万円以上 かつ 勤続3年以上 かつ 他のローンなし → 承認	<b>役立つこと</b> 担当者なしで、条件を満たす申請を自動で承認・否認できる

『事実』と『ルール』で知識を表すと、判断にもデータの自動処理にも使える

## ① 人工知能:判断に役立つ



事実 + ルール

↓ 推論(ルールを適用)

判断:貸出OK(太郎)

同じしくみで、診断・予測などの判断にも使える  
例)図書館:貸出の可否 / 医療:症状からの診断

理由を説明できる判断ができる

## ② データサイエンス:大量データを自動で処理



大量のデータを『事実』として取り込む

例)会員である(太郎) / 延滞なし(太郎) / 会員である(花子) ...

↓ ルールを適用(推論)

ルール:会員である(X) かつ 延滞なし(X) → 貸出OK(X)

↓

一件ずつ確実に結論を出す:

貸出OK(太郎) / 貸出OK(花子) / ... (全データを自動判定)

導かれた結論は『新しい事実』として知識に加わる

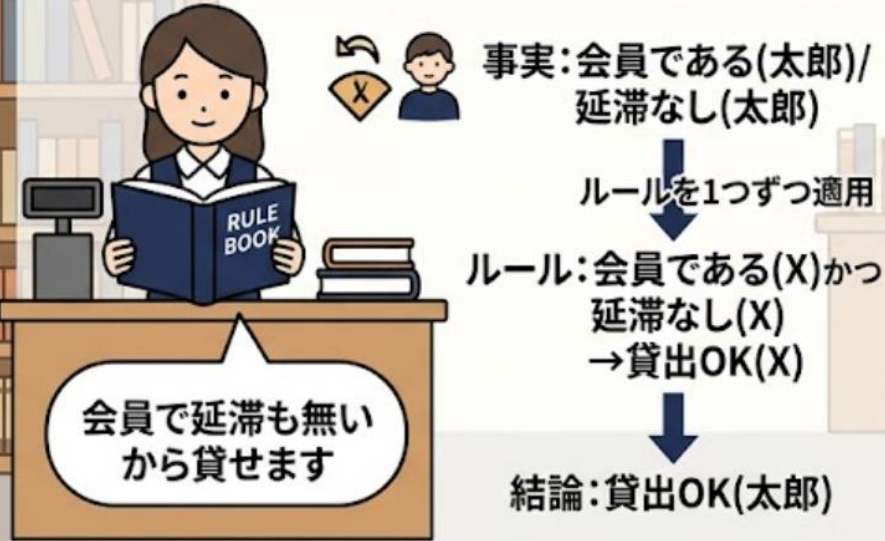
ルールから論理的に正しい結論だけを、大量に自動で導ける

# 推論と機械学習 (ニューラルネットワーク) の違い



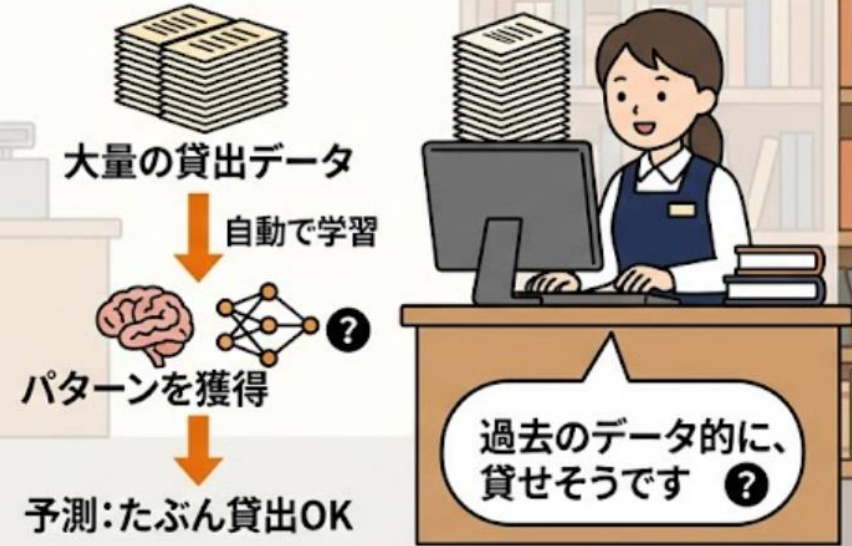
同じ質問: 太郎に貸せる?

## 推論(ルールでたどる)



理由を説明できる / 少ない知識でOK

## ニューラルネットワーク(データで学ぶ)



認識・予測が得意 / 中身は見えにくい(説明が難しい)

推論=人が書いたルールを論理でたどり、理由まで示す。ニューラルネット=大量データからパターンを学ぶが、理由は見えにくい

# 知識表現・推論とディープラーニングの違い



	知識表現・推論	ディープラーニング(データに基づく学習)
知識の作り方	人が事実とルールを書く	大量データから自動で学習
得意なこと	論理・ルールの処理	画像や言葉の認識・予測
判断の説明	結論に至る理由をたどれる	中身が見えにくい(説明が難しい)
必要なもの	明確に書けるルール	大量の学習データ
向いている課題	ルールがはっきりした問題 (例:貸出OKの判定)	ルール化しにくい問題 (例:画像の分類)

ルール + 事実 →(論理でたどる)→ 結論

大量データ →(学習)→ パターンを獲得



## 9-2. 述語と Prolog

取扱取引データ



データから  
ルールを学ぶ



機械学習

データから  
ルールを学ぶ

学んだルールを  
そのまま渡す



Prolog

事実に  
当てはめて  
推論する

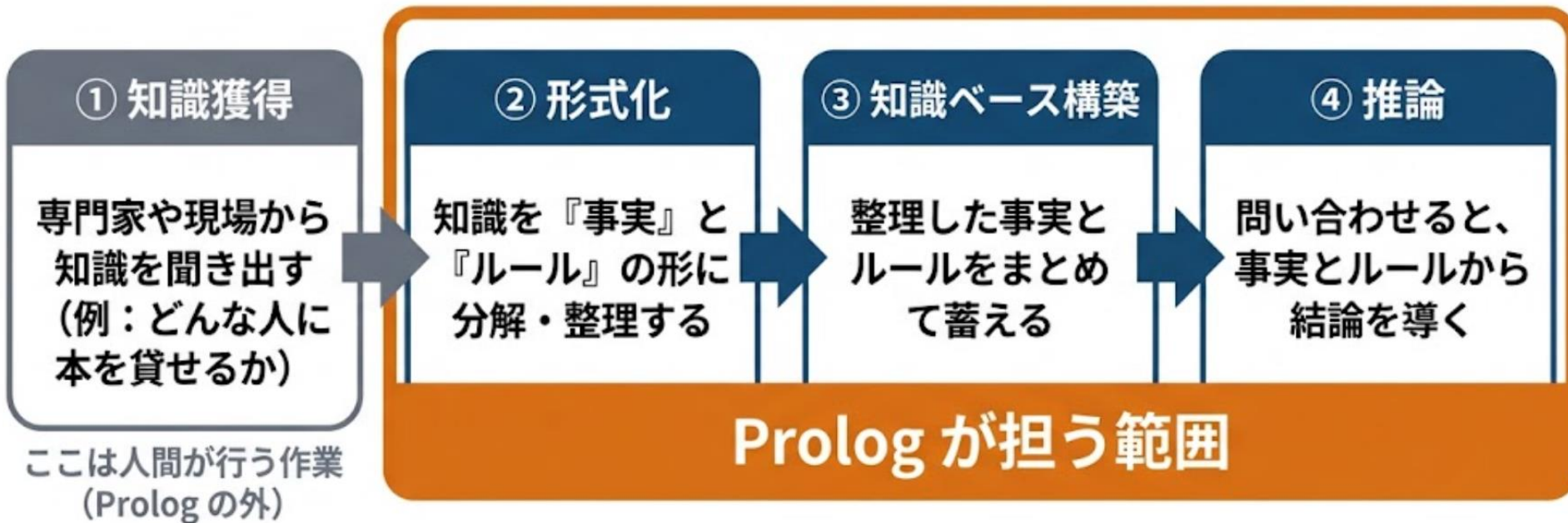
深夜かつ海外  
かつ高額  
→不正の疑い

深夜/海外/高額

推論：この取引を一時停止

データが学んだルールを、Prologが受け取って推論する

知識工学は『知識を集めて整理し、推論する』営み全体。Prolog はその後半を担う言語

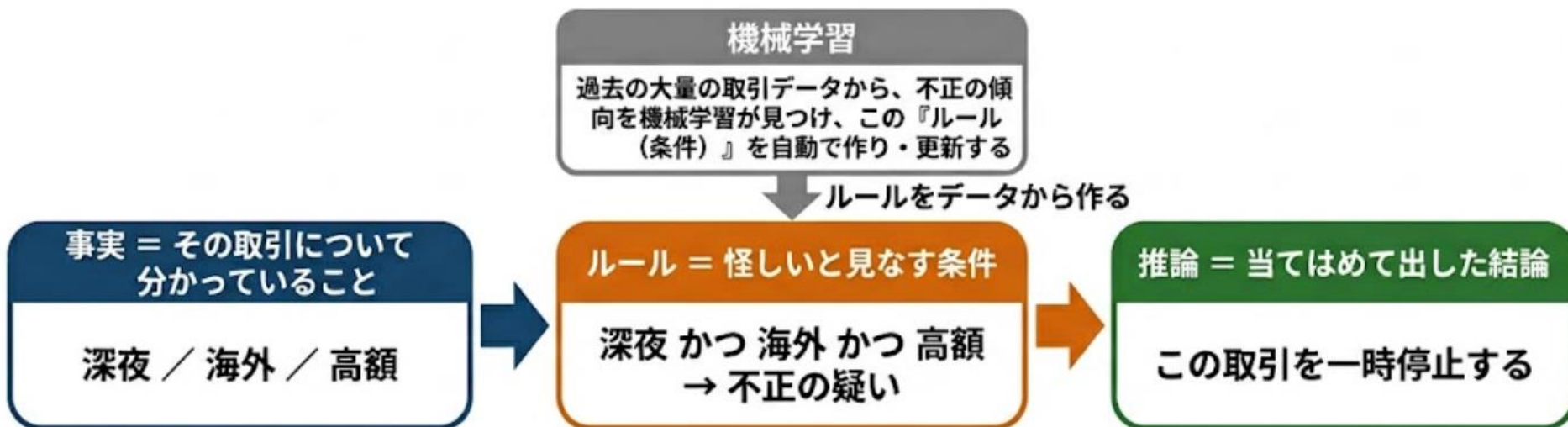


**知識を集めるのは人の仕事。集めた知識を書いて推論する部分を Prolog が引き受ける**

# 応用：データとAIで知識を増やし続ける



推論の仕組みは同じ。ちがうのは『ルールを人が書くか、データから学ぶか』だけ



**事実**に**ルール**を当てはめて**結論**を出す＝**推論**。  
その**ルール**を人ではなく**データ**から**学んで作る**のが**機械学習**

# 知識工学と Prolog



知識工学=人の知識を『事実とルール』に整理する営み。Prologはそれを書いて推論できる言語

知識工学：  
人の知識を整理する

『会員で延滞のない  
人には貸せる』  
といった人の知識

知識ベース（Prologで記述）

```
member(太郎).  
member(花子).  
no_overdue(太郎).  
staff(三郎). など
```

```
貸出ok(X) :- member(X),  
               no_overdue(X).  
貸出ok(X) :- staff(X). など
```

問い合わせ  
推論エンジンが答える

```
「?- 貸出ok(太郎).  
   → true」  
「?- 貸出ok(X).  
   → X = 太郎・三郎」  
「?- 貸出ok(花子).  
   → false」
```

事実とルールを照合し、  
自動で結論を導く

人の知識を事実とルールに整理し（知識工学）、  
Prolog に書いて問い合わせると、推論エンジンが結論を導く

各人が持つ事実（会員・延滞なし・職員）を2本のルールに当てはめると、貸出OKかどうかが決まる



太郎

会員 延滞なし



花子

会員



次郎

会員 延滞なし



三郎

職員

ルール1

会員 **かつ** 延滞なし → 貸出OK

ルール2

職員 → 貸出OK

または (どちらか成り立てばOK)

貸出OK

(ルール1)

貸出NG

(延滞なしの事実が無い)

貸出OK

(ルール1)

貸出OK

(ルール2)



知識表現:事実トールルールをコンピュータが扱える形で書く。

## Prolog での書き方

事実

```
member(太郎). member(花子). member(次郎).  
no_overdue(太郎). no_overdue(次郎).  
staff(三郎).
```

ルール

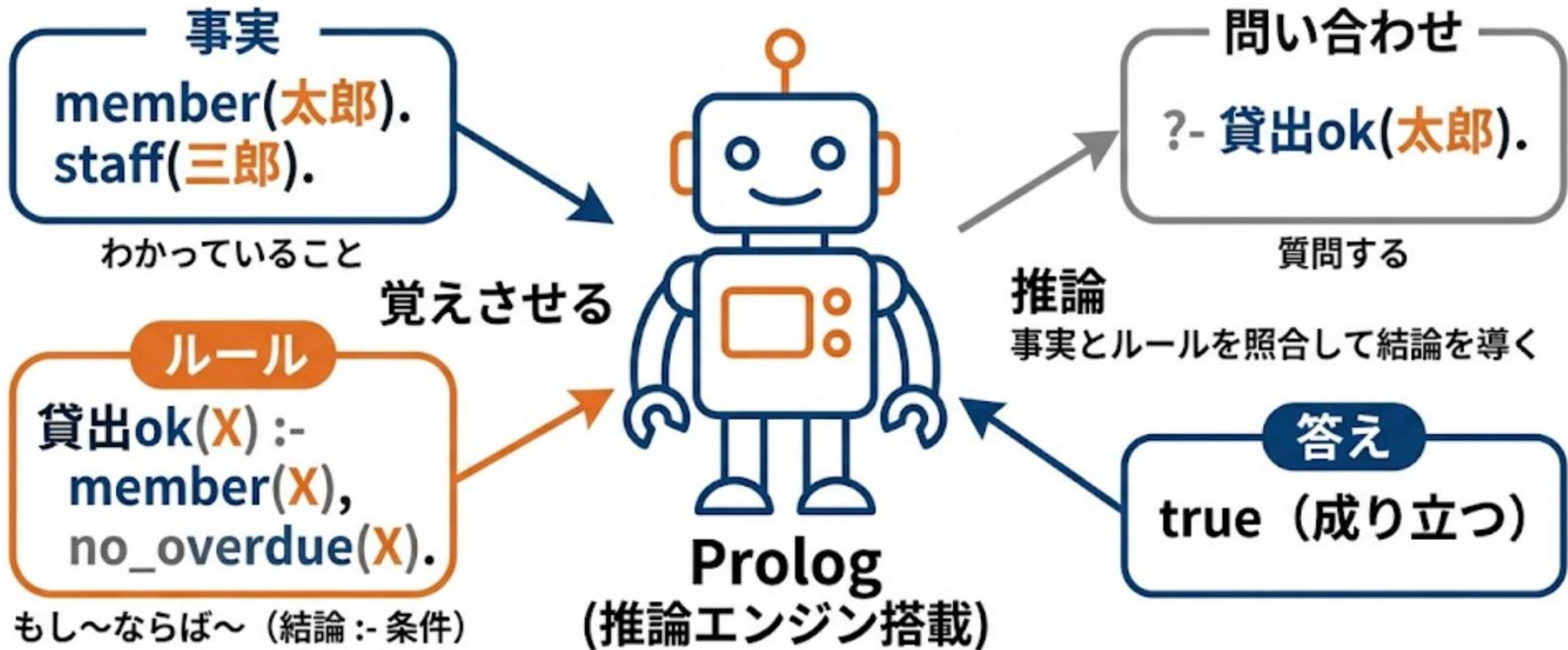
```
貸出ok(X) :- member(X), no_overdue(X).  
貸出ok(X) :- staff(X).
```

この書き方なら、太郎でも花子でも、同じルールで判定できる

Prologの構文に従い事実とルールを記述。この先も同じ。

# Prolog

Prologマシンは、事実とルールを覚え、問い合わせに推論で答える



- ※ 事実もルールも問い合わせも、すべて『述語と引数』を用いて書く。述語=関係や状態、引数=対象
- ※ 日本語の述語・定数が使えるかは処理系による

Prologプログラムは『事実』と『ルール』の2つでできている

Prolog  
プログラム

事実

述語(引数).

member(太郎).

ルール

結論 :- 条件.

貸出ok(X) :- member(X),  
no\_overdue(X).



# ルールの表現 (Prolog の場合)



**【結論】 :- 【条件】**

ルール1 もし 会員 **かつ** 延滞なし **ならば** 貸出OK

貸出ok(X) :- member(X), no\_overdue(X).

= かつ

ルール2           もし 職員 **ならば** 貸出OK

貸出ok(X) :- staff(X).

# 述語論理と Prolog



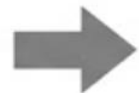
述語論理は『**述語 + 引数**』で**事実**や**ルール**を表現する。  
**Prolog** はそれを**プログラム**として書き、**実行**できる

述語論理 (考え方)

**member(太郎)**  
└───┬───┘  
述語 引数

述語と引数の組み合わせで  
『事実』や『ルール』を表す

プログラム  
として書く



Prologプログラム

```
% 事実
member(太郎).
member(花子).
member(次郎).
no_overdue(太郎).
no_overdue(次郎).
staff(三郎).

% ルール
貸出ok(X) :-
    member(X),
    no_overdue(X).
貸出ok(X) :- staff(X).
```

実行する



実行できる

? 貸出ok(X)

太郎

次郎

三郎

ルールと事実から、  
コンピュータが自動で  
結論を導く

## 9-3. 推論エンジン

# 推論エンジン



事実とルールがあっても、それを動かす『推論エンジン』がないと答えは出ない



『事実』を置く場所を作業領域、それを動かす働きを推論エンジンと呼ぶ

## 推論エンジンは『事実』と『ルール』をもとに、質問に回答する



推論エンジン

### 事実

```
member(`太郎`),
member(`花子`),
member(`次郎`),
no_overdue(`太郎`),
no_overdue(`次郎`),
staff(`三郎`).
```

### ルール

```
貸出ok(`X`) :-
    member(`X`),
    no_overdue(`X`).
貸出ok(`X`) :-
    staff(`X`).
```

### 質問 (Prolog inquiry format)

?- 貸出ok(太郎).

### 結果

true (成り立つ)

太郎は member かつ no\_overdue。ルール1で導出される

### 質問 (Prolog inquiry format)

?- 貸出ok(X).

### 結果

X = 太郎・次郎・三郎

Xを総当たりで探索。太郎と次郎はルール1 (member  $\wedge$  no\_overdue)、三郎はルール2 (staff) で成り立つ

### 質問 (Prolog inquiry format)

?- 貸出ok(花子).

### 結果

false (成り立たない)

花子は no\_overdue の事実が無く staff でもないため、どちらのルールにも単一化できない

事実とルールを保持する場所が『作業領域』、それらを照合して質問に答える働きが『推論エンジン』

# Prolog における true, false の意味

Prolog の false は『偽』ではなく『証明できない (知られていない)』という意味

**true (成り立つ)**

**?- 貸出ok(太郎).**

太郎は貸出okである

太郎は member かつ no\_overdue。  
ルールで証明できる

**false (成り立たない)**

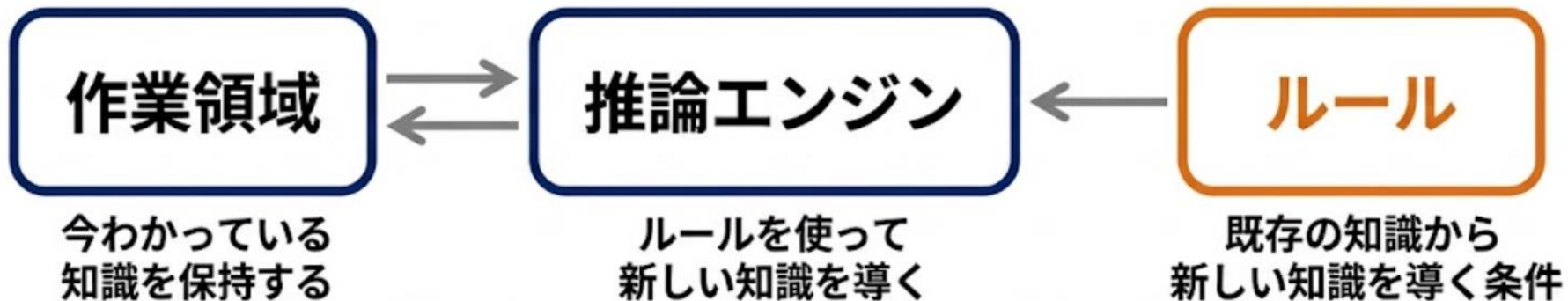
**?- 貸出ok(花子).**

花子が貸出okだと証明できない (不明)

花子は no\_overdue の事実が無く staff  
でもないため、どのルールも成り立たない

**注意：false は『花子は貸出禁止』という意味ではない。  
ただ『貸出okだと証明する根拠が無い』だけ**

『もし～ならば～』形式のルールは、作業領域・推論エンジン・ルールの3要素で動く



```
% 作業領域 (初期の事実)
member(太郎). member(花子).
member(次郎).

no_overdue(太郎).
no_overdue(次郎).
staff(三郎).
```

推論の結果  
(作業領域に追加)

貸出ok(太郎)

貸出ok(次郎)

貸出ok(三郎)

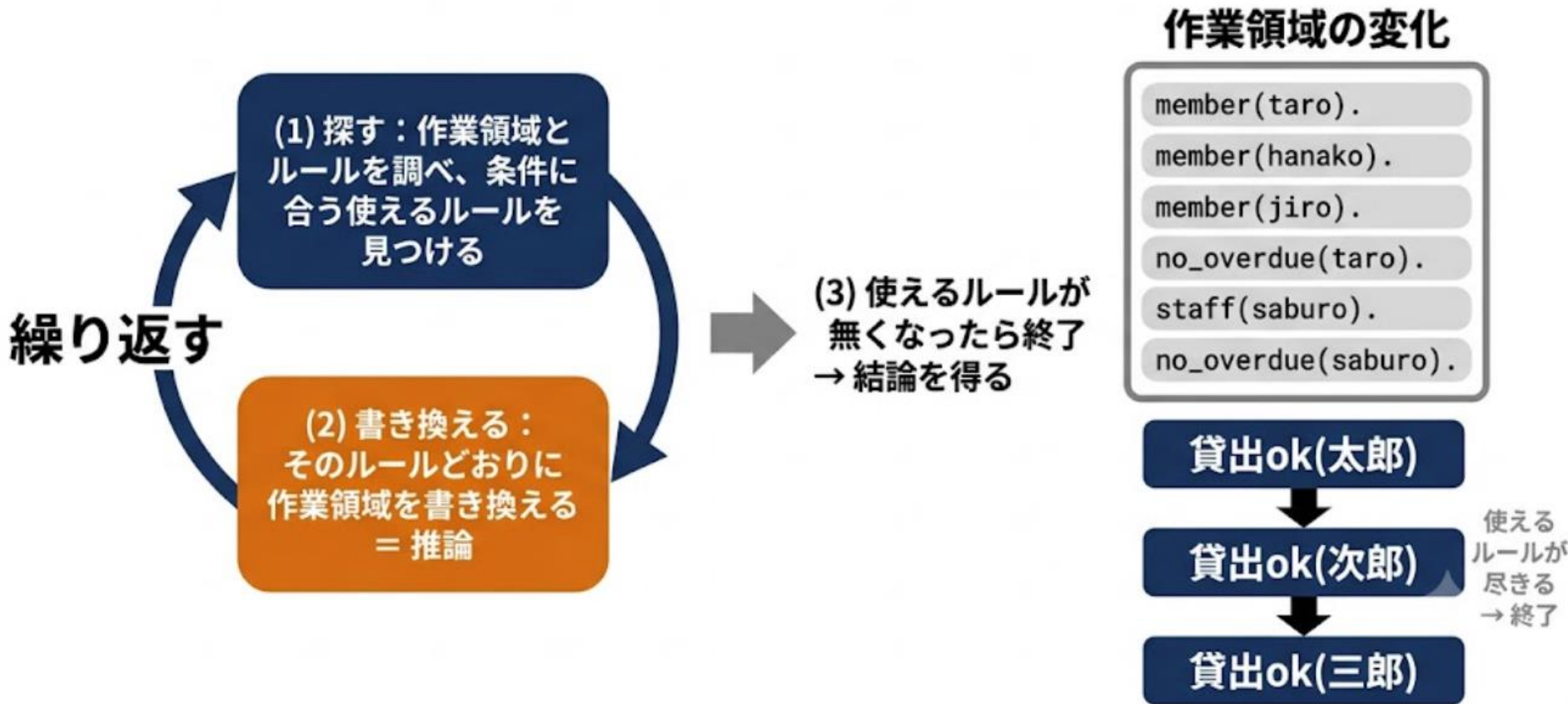
```
% ルール
貸出ok(X) :- member(X),
              no_overdue(X).
貸出ok(X) :- staff(X).
```

花子は会員だが延滞なしでなく職員でもないため、どのルールにも該当せず導出されない

# 推論エンジンの仕組み



推論エンジンは『探す→書き換える』を、使えるルールが尽きるまで繰り返す



# Prolog、データベースの比較



知識は、Prologの『事実』でも、表（テーブル）でも表せる

## Prolog (述語論理)

```
member(太郎).  
member(花子).  
no_overdue(太郎).  
staff(三郎).
```

事実を1つずつ書き並べる



## リレーショナルDB (表)

### 会員表

名前	延滞なし
太郎	○
花子	×

### 職員表

名前  
三郎

行と列でまとめて表す

述語の名前 → 表の名前 / 述語の引数 → 表の列・行

# Prolog、データベースの比較



知識を表現する方法には、これまで見た『ルールベース』のほかに『リレーショナルデータベース』がある

知識を表現する方法

**ルールベース (述語論理)**

事実とルールを書き、  
推論エンジンが結論を導く

これまで見てきた方法

**リレーショナルデータベース**

データを表 (テーブル) の  
形で整理して扱う

次に見ていく方法

この2つを比べながら、それぞれの考え方の違いを見ていく

# 知識表現と推論の実現：データベースシステムを利用



ルールが1本なら、述語論理の演繹（ルール+事実→結論）はSQLのテーブルとJOINで再現できる

事実=テーブル

1. member (会員である)

person
太郎
花子
次郎

no\_overdue (延滞なし)

person
太郎
次郎

↓ JOIN

2. ルール=固定のJOIN

```
SELECT m.person AS 貸出OK
FROM member m
JOIN no_overdue n ON m.person = n.person;
```

3. 実行結果

貸出OK
太郎
次郎

花子は member にいるが no\_overdue にいないため脱落

# 知識表現と推論の実現：データベースシステムを利用



ルールが増えると、結論を出すSQL本体（プログラム）を書き換える必要がある

## ルール追加

職員である(X) → 貸出OK(X) を追加

staff (職員である)

person
三郎



## 書き換わったSQLと結果

```
SELECT m.person AS 貸出OK
FROM member m
JOIN no_overdue n ON m.person = n.person
UNION
SELECT s.person FROM staff s;
```



## 実行結果

貸出OK
太郎
次郎
三郎

## 2つの方法は得意が違う — 目的で使い分ける

	<b>Prolog</b>	<b>リレーショナルデータベース</b>
<b>得意</b>	<b>ルールを足すだけで 知識を増やせる</b>	<b>大量のデータを 高速に検索できる</b>
<b>向く 場合</b>	<b>ルールが増えたり 変わったりする</b>	<b>ルールが固定で、 表の検索で答えが出る</b>

ルールが増えると、SQLは書き換えが必要になる

# 演習





New tab × +

Create a **Program** Notebook here

based on **Empty** Student CLP s(CASP) profile

user:"me" × Filter ▾ Type ▾ Q

Type	Name	Tags	User	Modified
	uvpshVYI	mammiferesss		2025-11-20 08:47:33

table results Run!

# 演習 1 . Prolog による推論

① swish のページを Web ブラウザで開く

<https://swish.swi-prolog.org>

このサイトは, オンラインで Prolog の体験,  
学習ができる

## ② Prolog の画面に変わる. 「Program」をクリック.

The screenshot shows the SWISH Prolog IDE interface. At the top, there is a menu bar with 'File', 'Edit', 'Examples', and 'Help'. To the right of the menu bar is a search bar and a notification icon showing '25'. Below the menu bar, there is a 'Create' section with two buttons: 'Program' (highlighted with a red box and a red arrow) and 'Notebook'. Below these buttons, it says 'based on Empty Student CLP profile'. There is also a search bar with the text 'user:"me"' and a 'Filter' dropdown. Below the search bar, there is a message: 'No matching files. If you are a new user you may: Use the Examples menu from the navigation bar, Use the Program or Notebook button above. help on search'. The main area of the interface is a large owl logo. Below the owl logo, there is a code editor with the text '?- parent(ali, anne).' and a 'Run!' button. At the bottom of the code editor, there are buttons for 'Examples', 'History', and 'Solutions', and a checkbox for 'table results'.

Your Prolog rules and facts go here ...

事実, 規則  
の編集

答えが表示される

```
?- parent(alj, anne).
```

問い合わせをコンピュータに  
与える

事実, 規則

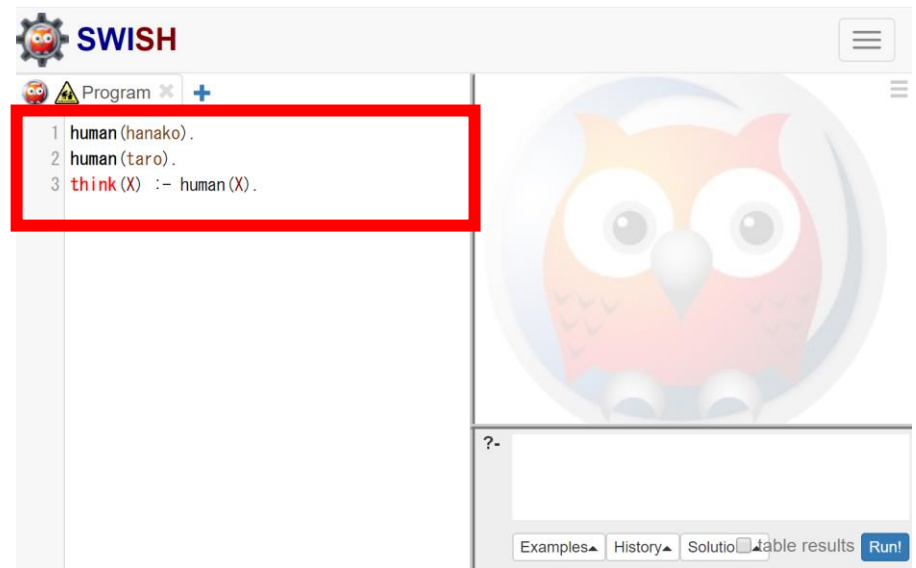
をコンピュータに  
読み込ませるボタン

table results

Run!

### ③ Prologプログラムの準備

```
human(hanako).  
human(taro).  
think(X) :- human(X).
```



Prolog プログラム

編集画面を使用

- ④ 問い合わせ「`human(X).`」を入れ, 「Run」ボタン  
そして「Next」ボタンをクリック  
(答えが増えなくなるまで)



`human(X).`



**X** = hanako

答え

**X** = taro

?-

`human(X).`

問い合わせ

Examples▲

History▲

Solutions▲

results

Run!

- ⑤ 問い合わせ「**think(X).**」を入れ、「Run」ボタン  
そして「Next」ボタンをクリック  
(答えが増えなくなるまで)

The image shows a screenshot of a programming environment. At the top, there is a window titled `think(X).` with a gear icon and a smiley face. Inside this window, there is a list of answers: `X = hanako` and `X = taro`. The word **答え** (Answer) is written in red next to the list. Below this window is a larger window with a question mark icon and the text `think (X).`. The word **問い合わせ** (Inquiry) is written in red next to the text. At the bottom of this window, there are several buttons: `Examples▲`, `History▲`, `Solutio` (partially visible), `table results`, and a blue `Run!` button.


⑥ 問い合わせ 「**think(taro).**」 を入れ, 「Run」 ボタン  
think っこ taro っこ ピリオド



「true」を確認

# 問い合わせを，1度に2個書くことはできない

```
?- male(ali).  
ma/e(zeyn).|
```



```
male(ali). male(zeyn).  
Cannot run query due to a syntax error (check query window)  
?- male(ali).  
Syntax error: Operator expected  
ma/e(zeyn).
```

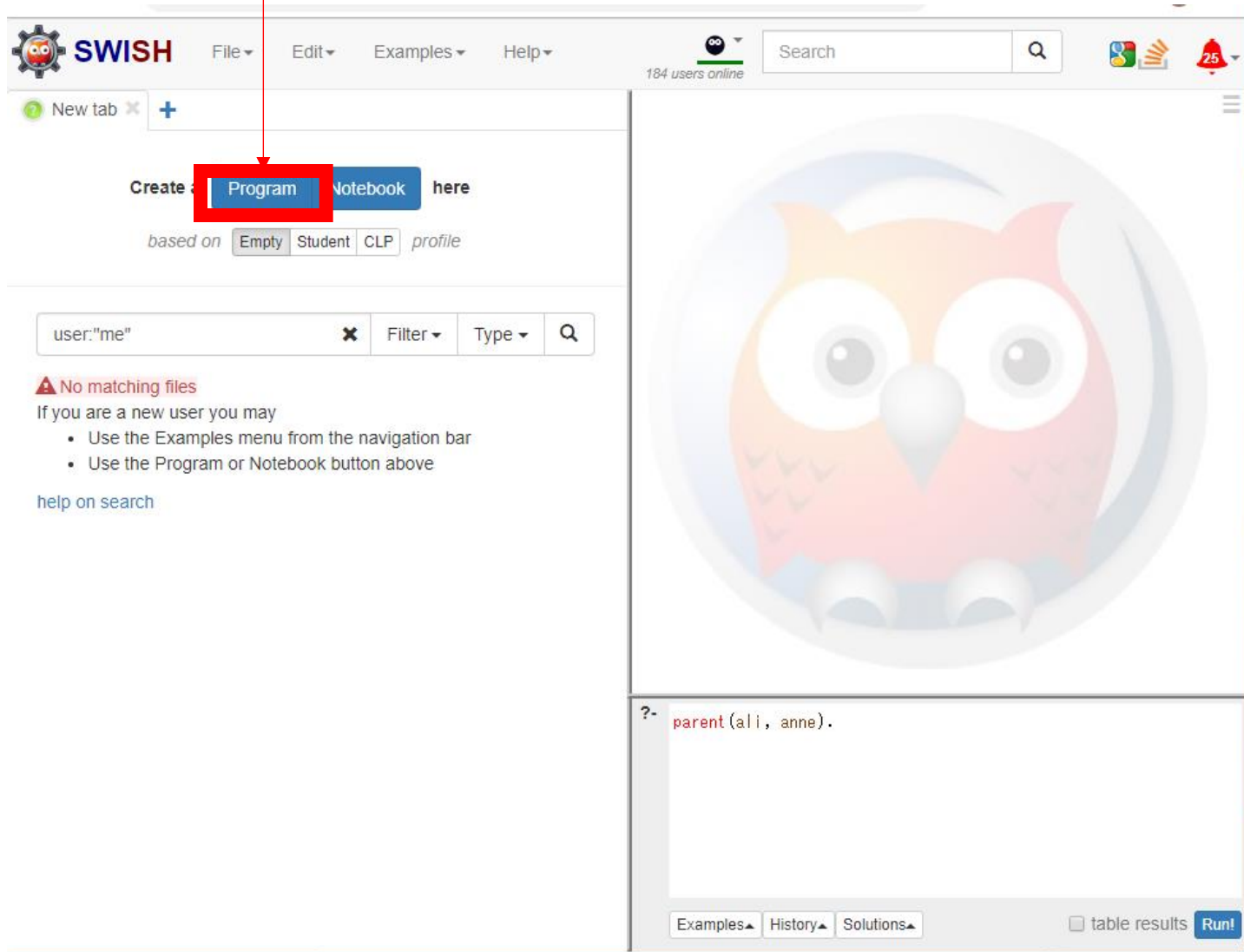
問い合わせを書くたびに，  
前の問い合わせは消す。

swish のページを Web ブラウザで開く

<https://swish.swi-prolog.org>

このサイトは、オンラインで Prolog の体験、  
学習ができる

# Prolog の画面に変わる. 「Program」をクリック.



The screenshot shows the SWISH Prolog IDE interface. At the top, there is a menu bar with 'File', 'Edit', 'Examples', and 'Help'. To the right of the menu bar is a search bar and a notification icon showing '25'. Below the menu bar, there is a 'Create' section with two buttons: 'Program' (highlighted with a red box and a red arrow) and 'Notebook'. Below these buttons, there is a 'based on' section with 'Empty', 'Student', 'CLP', and 'profile' options. A search bar contains the text 'user:"me"'. Below the search bar, there is a message: 'No matching files' and instructions for new users. The main workspace is divided into two panes. The top pane shows a large, stylized owl logo. The bottom pane shows a code editor with the text '?- parent(ali, anne).' and a 'Run!' button.

# Prolog プログラムの準備

```
male(ali).  
female(zeyn).  
female(anne).  
parent(ali, anne).  
parent(zeyn, anne).  
child(Y, X) :- parent(X, Y).
```

Prolog プログラム



The screenshot shows the SWISH web interface in a browser window. The address bar shows the URL <https://swish.swi-prolog.org>. The page title is "SWISH". The main content area is titled "Program" and contains the following Prolog code, which is highlighted with a red box:

```
1 male(ali).  
2 female(zeyn).  
3 female(anne).  
4 parent(ali, anne).  
5 parent(zeyn, anne).  
6 child(Y, X) :- parent(X, Y).
```

Below the code editor, there is a search bar with the text "?- human(X)." and buttons for "Examples", "History", "Solutions", and "Run!". A large owl logo is visible in the background of the interface.

編集画面を使用

問い合わせ「male.ali)」を入れ、「Run」ボタン  
male かつこ ali かつこ ピリオド Enterキー

The image shows a screenshot of the SWISH programming environment. The top bar displays the SWISH logo and a hamburger menu. Below it, a code editor window shows the code `male.ali)`. A red box highlights the text `true` in the console output, with the Japanese label **答え** (Answer) next to it. Another red box highlights the code `male.ali)` in the console input area, with the Japanese label **問い合わせ** (Inquiry) next to it. Below the code editor, the text 「true」を確認 (Check 'true') is written. On the right side, a smaller window shows the same code editor and console, with a red box around the `true` output and another red box around the `male.ali)` input. At the bottom right of this window, the **Run!** button is highlighted with a red box.

## エラーメッセージが出ることもある



```
mail(ali).
```

```
procedure `mail(A)' does not exist
```

```
?-
```

```
mail(ali).
```

正しくは「male」

間違って「mail」と書いてしまった

問い合わせ「`male(anne).`」を入れ、「Run」ボタン  
male かつこ anne かつこ ピリオド Enterキー



`male(anne).`

**false**

?-

`ma l e (anne).`

「false」を確認

問い合わせ 「parent(ali, anne).」 を入れ, 「Run」  
ボタン



*parent(ali, anne).*

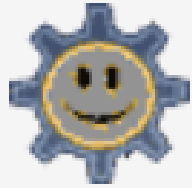
**true**

?-

*parent(ali, anne).*

「true」を確認

問い合わせ 「child(anne, ali).」 を入れ, 「Run」 ボタン



*child*(anne, ali).

**true**

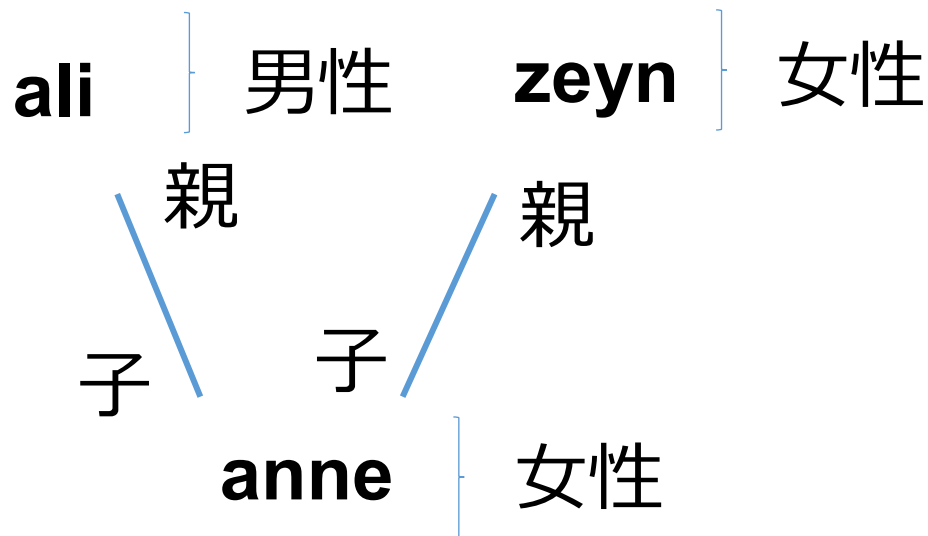
?-

*child*(anne, ali).

「true」を確認

## 確認クイズ (余裕のある人向け)

問い合わせの答えは何になるか, 空欄の true または false を考えてみよう



問い合わせ	答え
female(anne).	
parent(zeyn, ali)	
child(zeyn, ali).	
child(ali, anne).	

そして Prolog で実際に実行して確かめてみよう

## 演習 3 . 様々な問い合わせ

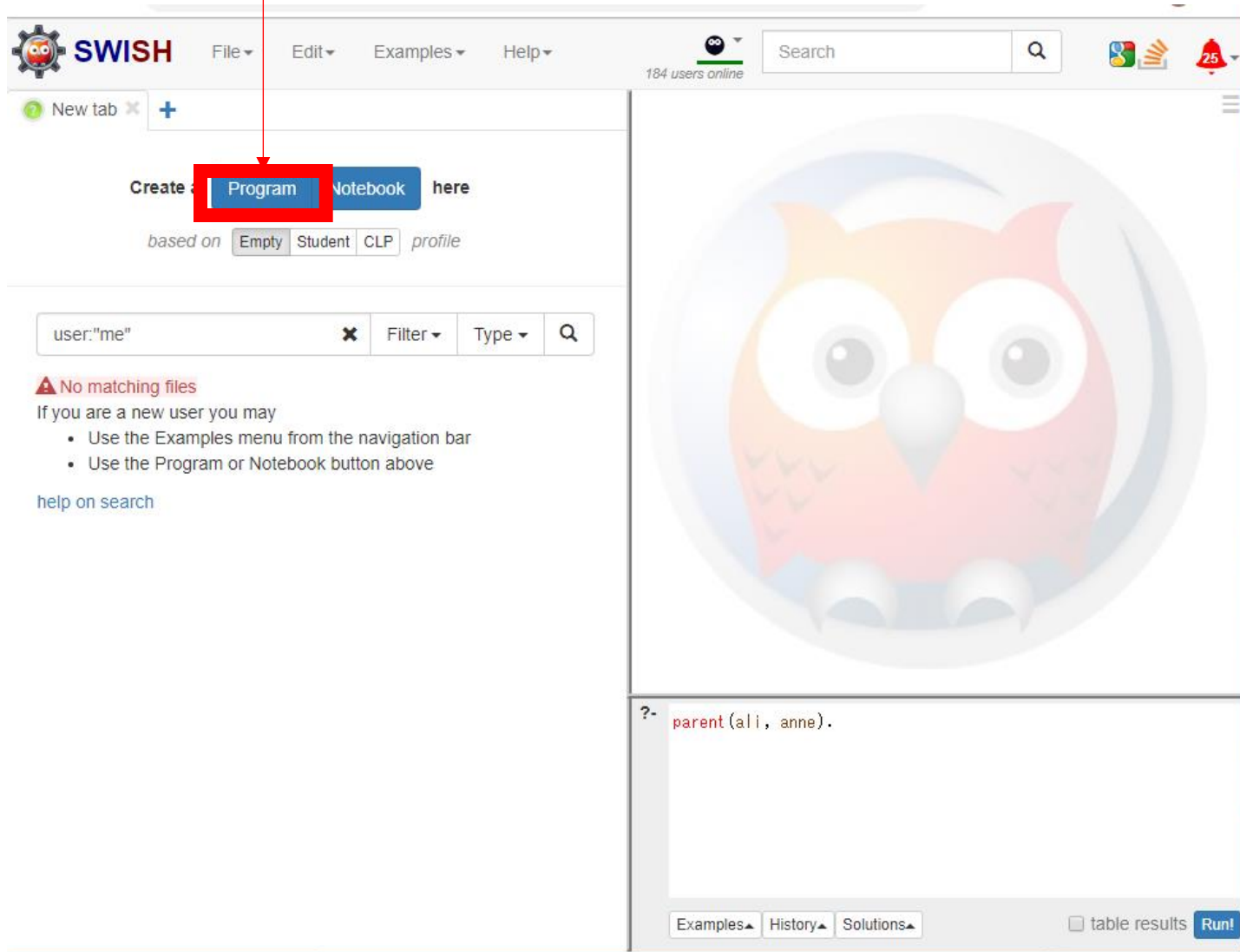


swish のページを Web ブラウザで開く

<https://swish.swi-prolog.org>

このサイトは、オンラインで Prolog の体験、  
学習ができる

# Prolog の画面に変わる. 「Program」をクリック.



The screenshot shows the SWISH Prolog IDE interface. At the top, there is a menu bar with 'File', 'Edit', 'Examples', and 'Help'. To the right of the menu bar is a search bar and a notification icon showing '25'. Below the menu bar, there is a 'Create' section with two buttons: 'Program' (highlighted with a red box and a red arrow) and 'Notebook'. Below these buttons, it says 'based on' followed by 'Empty', 'Student', 'CLP', and 'profile' buttons. A search bar contains the text 'user:"me"'. Below the search bar, there is a message: 'No matching files' and instructions for new users. The main workspace is divided into two panes. The top pane shows a large, stylized owl logo. The bottom pane is a code editor with the text '?- parent(ali, anne).' and a 'Run!' button at the bottom right.

# 「プログラム」の準備

```
male(ali).  
female(zeyn).  
female(anne).  
parent(ali, anne).  
parent(zeyn, anne).  
child(Y, X) :- parent(X, Y).
```

Prolog プログラム



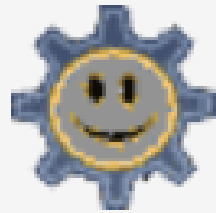
```
1 male(ali).  
2 female(zeyn).  
3 female(anne).  
4 parent(ali, anne).  
5 parent(zeyn, anne).  
6 child(Y, X) :- parent(X, Y).
```

?- human(X).

Examples History Solutions Run!

編集画面を使用

問い合わせ「male(X).」を入れ, 「Run」ボタン



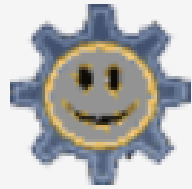
*male(X).*

**X** = ali

?-

*male(X).*

問い合わせ「**female(X).**」を入れ, 「Run」ボタン  
そして「**Next**」ボタンをクリック  
(答えが増えなくなるまで)



*female(X).*

**X** = zeyn

**X** = anne

?-

*female(X).*

問い合わせ「`parent(X, Y).`」を入れ、 「Run」 ボタン  
そして「Next」 ボタンをクリック  
(答えが増えなくなるまで)



`parent(X, Y).`

`X = ali,`

`Y = anne`

`X = zeyn,`

`Y = anne`

?-

`parent(X, Y).`

問い合わせ「**child(X, Y).**」を入れ, 「Run」ボタン  
そして「Next」ボタンをクリック  
(答えが増えなくなるまで)

**X** = anne,

**Y** = ali

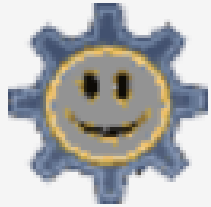
**X** = anne,

**Y** = zeyn

?-

**child(X, Y).**

問い合わせ 「parent(**ali**, Y).」 を入れ, 「Run」 ボタン



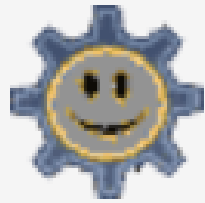
*parent*(ali, Y).

**Y** = anne

?-

parent (ali, Y).

問い合わせ 「child(**ali**, Y).」 を入れ, 「Run」 ボタン



*child*(ali, Y).

**false**

?-

*child*(ali, Y).

false は「不明」.

## 確認クイズ (余裕のある人向け)

それぞれの問い合わせの答えは何か考えてみよう。  
そして、実際に Prolog で動かして確かめてみよう

1. parent(zeyn, Y).

- ① false   ② true   ③ Y = anne   ④ Y = ali

2. child(zeyn, Y).

- ① false   ② true   ③ Y = anne   ④ Y = ali

