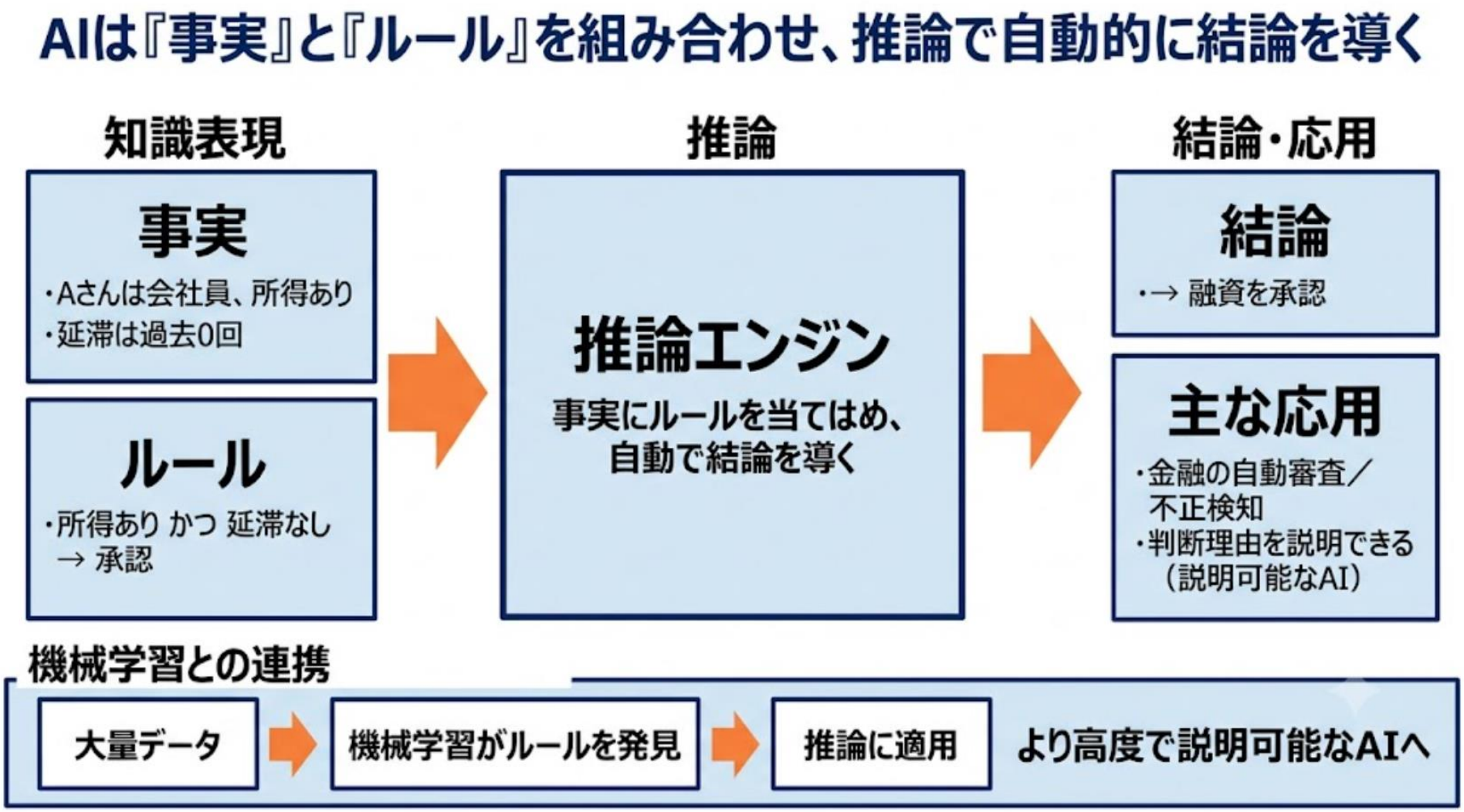
9. 知識表現と推論

（人工知能）

金子邦彦

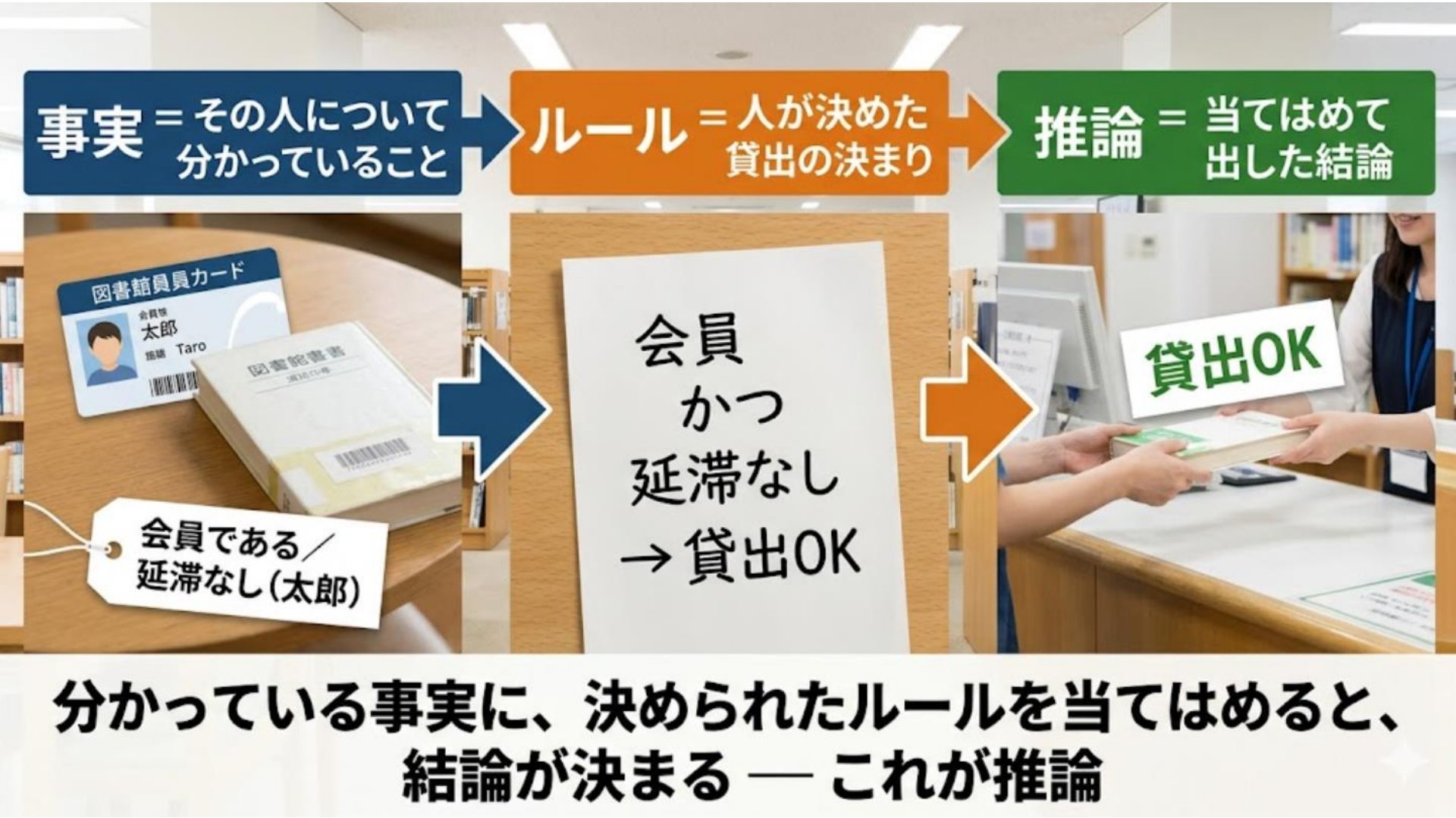
資料中の図などは Google Nano Banana 2 を用いて作成

「人工知能」第９回の内容

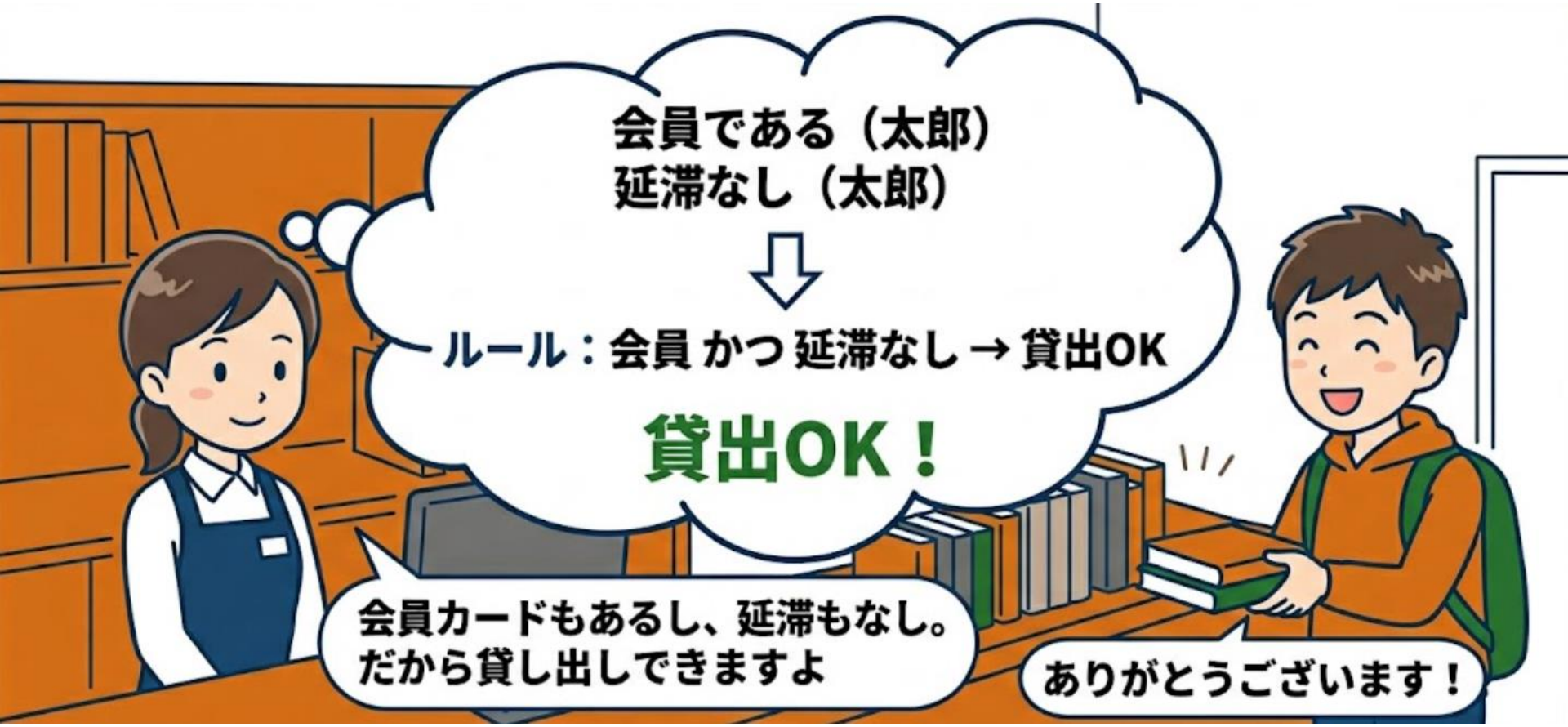


本授業の図は生成AIである Google Nano Banana 2 を用いて作成

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)

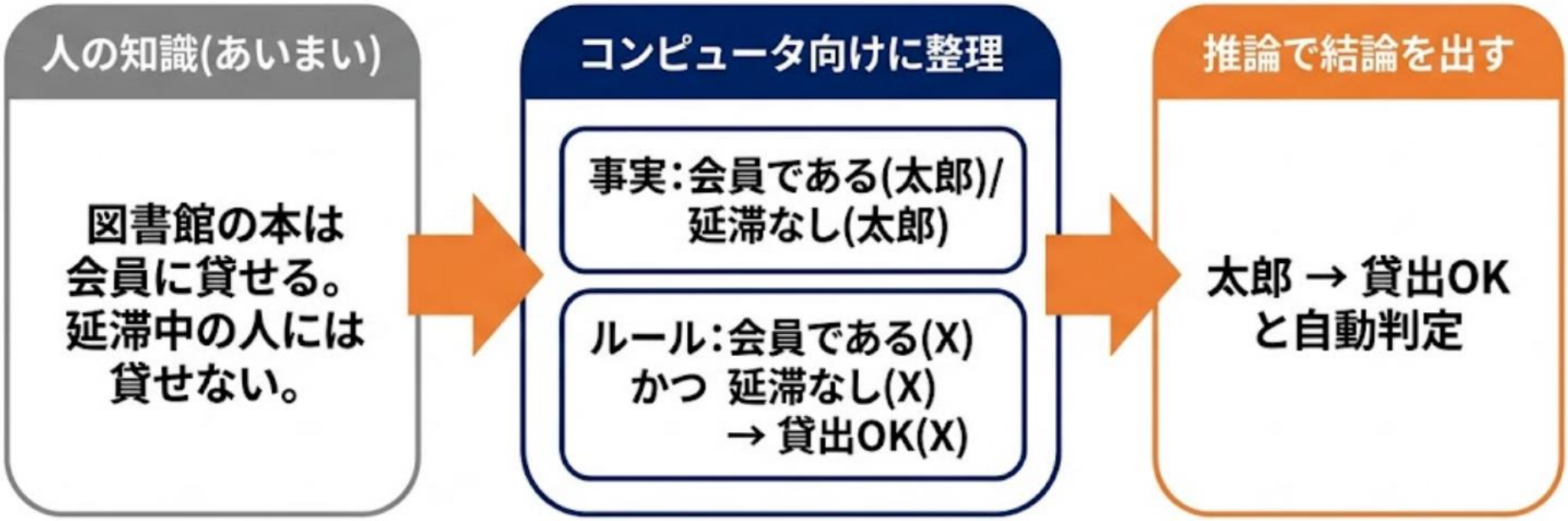
会員である　かつ

延滞なし

貸出OK

知識表現とは

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

推論

『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

推論

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?

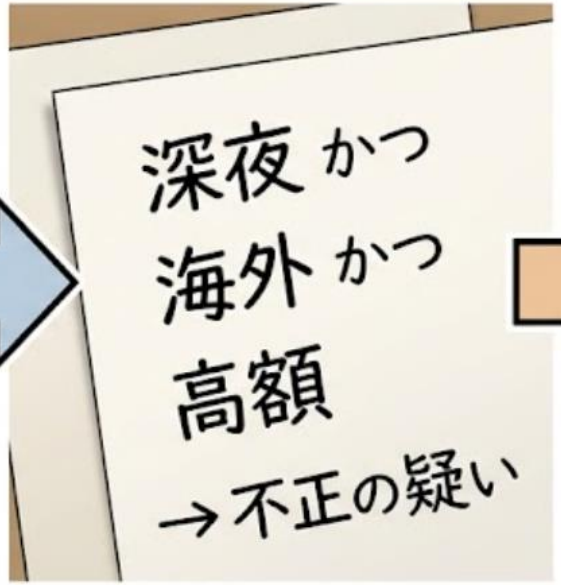
↓ ルールを適用

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
① Google検索の情報パネル 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まって表示され、サイトを見て回る手間が省ける
② クレジットカードの不正検知 国内各社が導入	事実 カード保有者の利用場所・時間帯・金額の傾向を事実として保持	ルール 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	役立つこと 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
③ 金融・保険の自動審査 金融業で広く普及	事実 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	ルール 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	役立つこと 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

知識表現と推論の有用性

『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)

↓

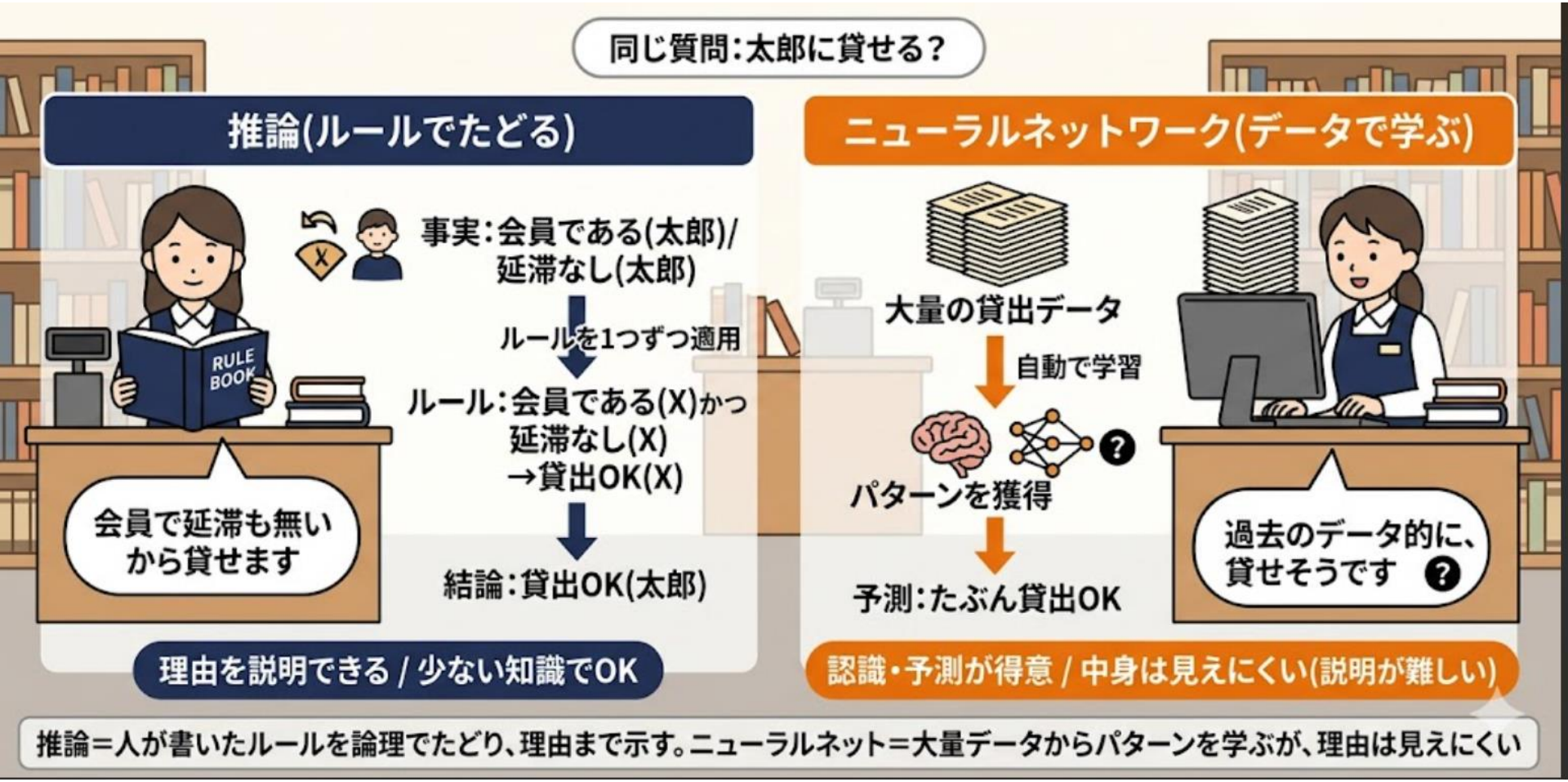
一件ずつ確実に結論を出す：

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



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

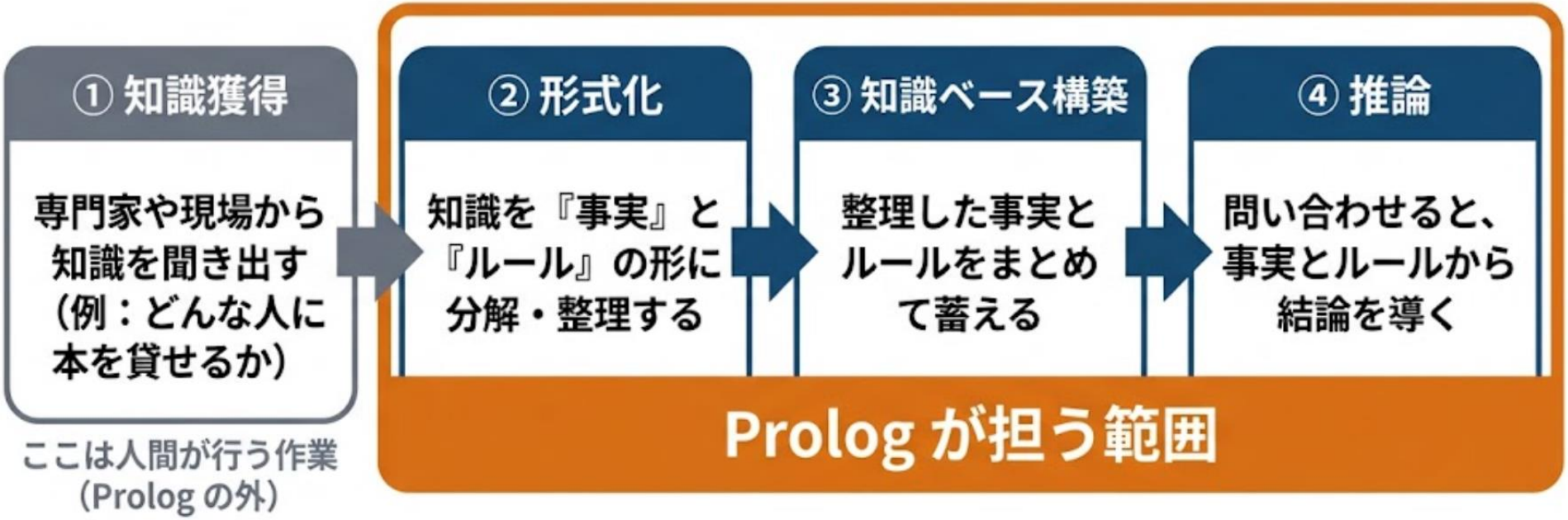
大量データ →(学習)→ パターンを獲得

9-2. 述語と Prolog

データが学んだルールを、Prolog が受け取って推論する

知識工学と Prolog

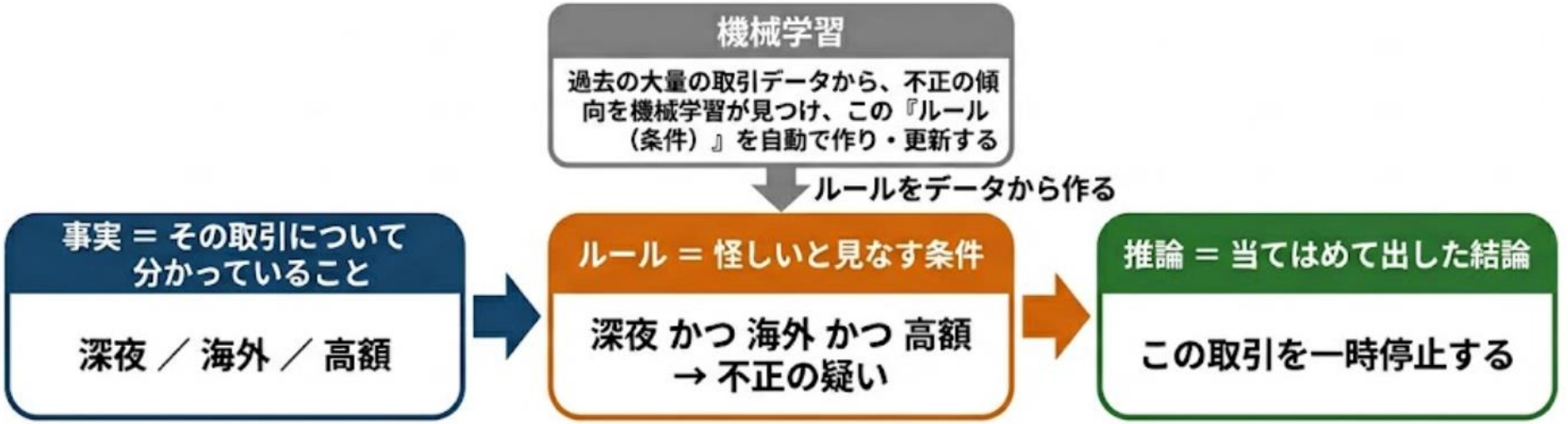
知識工学は『知識を集めて整理し、推論する』営み全体。Prolog はその後半を担う言語



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

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

知識の例

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

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).
```

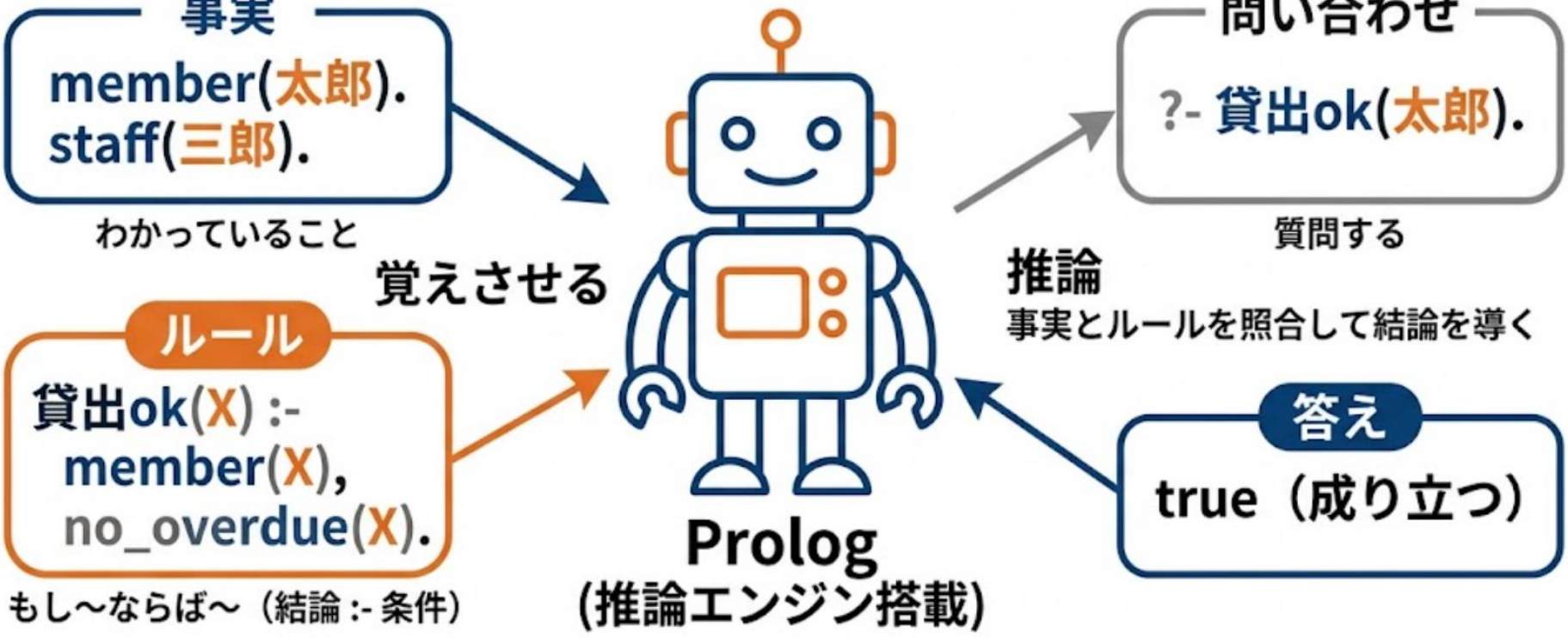
この書き方なら、太郎でも花子でも、同じルールで判定できる

Prologの構文に従い事実とルールを記述。この先も同じ。

Prolog

Database Lab.

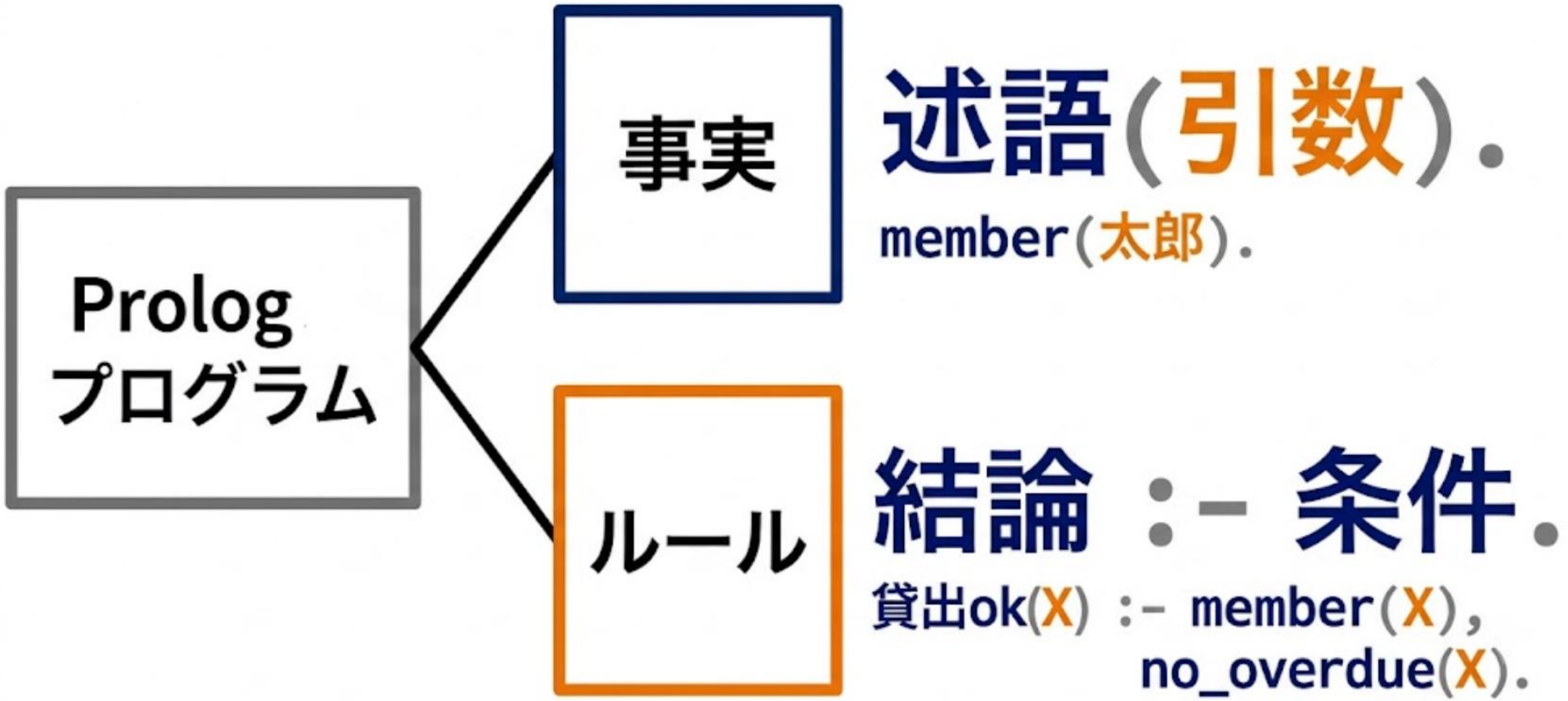
Prologマシンは、事実とルールを覚え、問い合わせに推論で答える



※ 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

Prologプログラムは『事実』と『ルール』の2つでできている



事実の表現 (Prolog の場合)

事実は『述語(引数)』の形で書く

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

member(太郎).

述語（属性）

引数（値）

述語：member・no_overdue・staff ／ 引数：太郎・花子・次郎・三郎

ルールの表現 (Prolog の場合)

【結論】 :- 【条件】

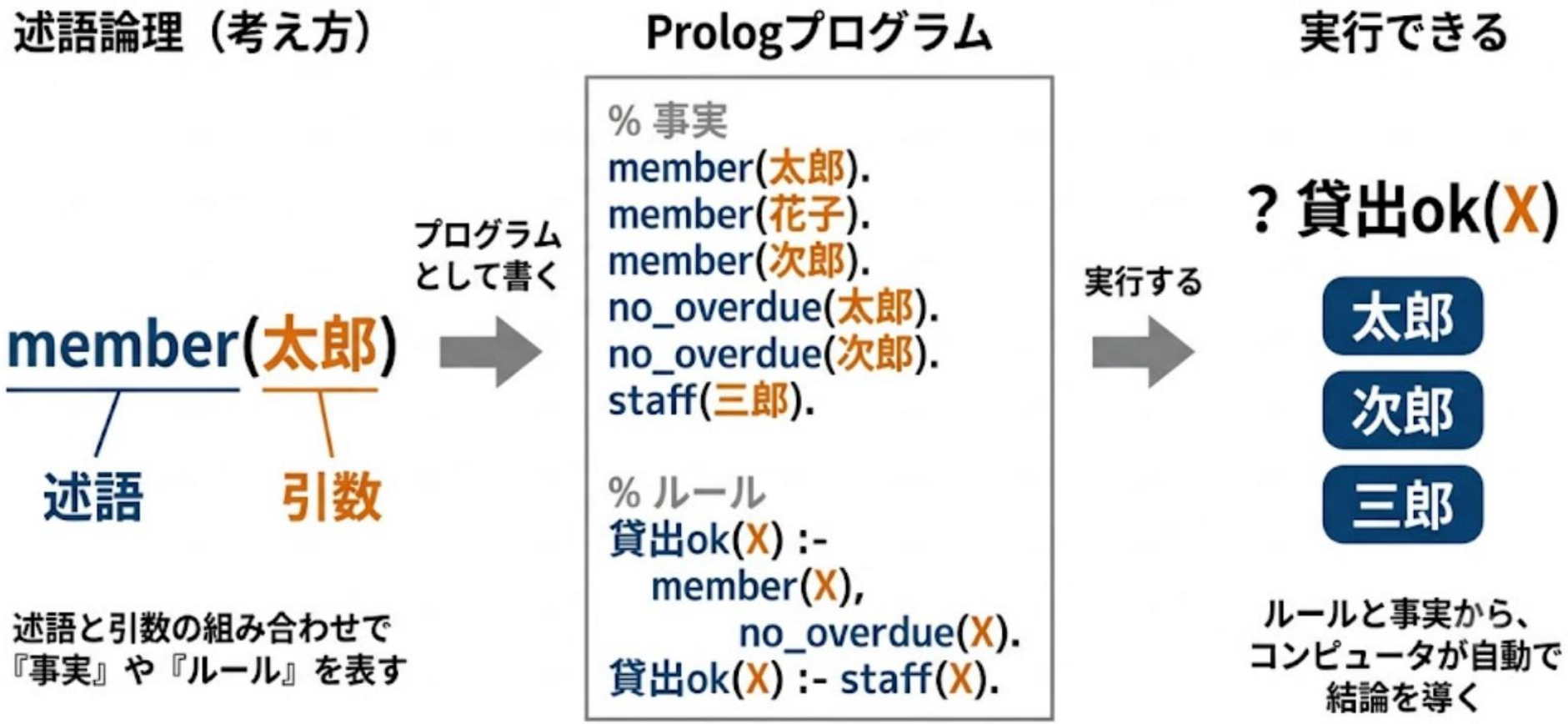
ルール1 もし 会員 かつ 延滞なし ならば 貸出OK
`貸出ok(X) :- member(X), no_overdue(X).`

= 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

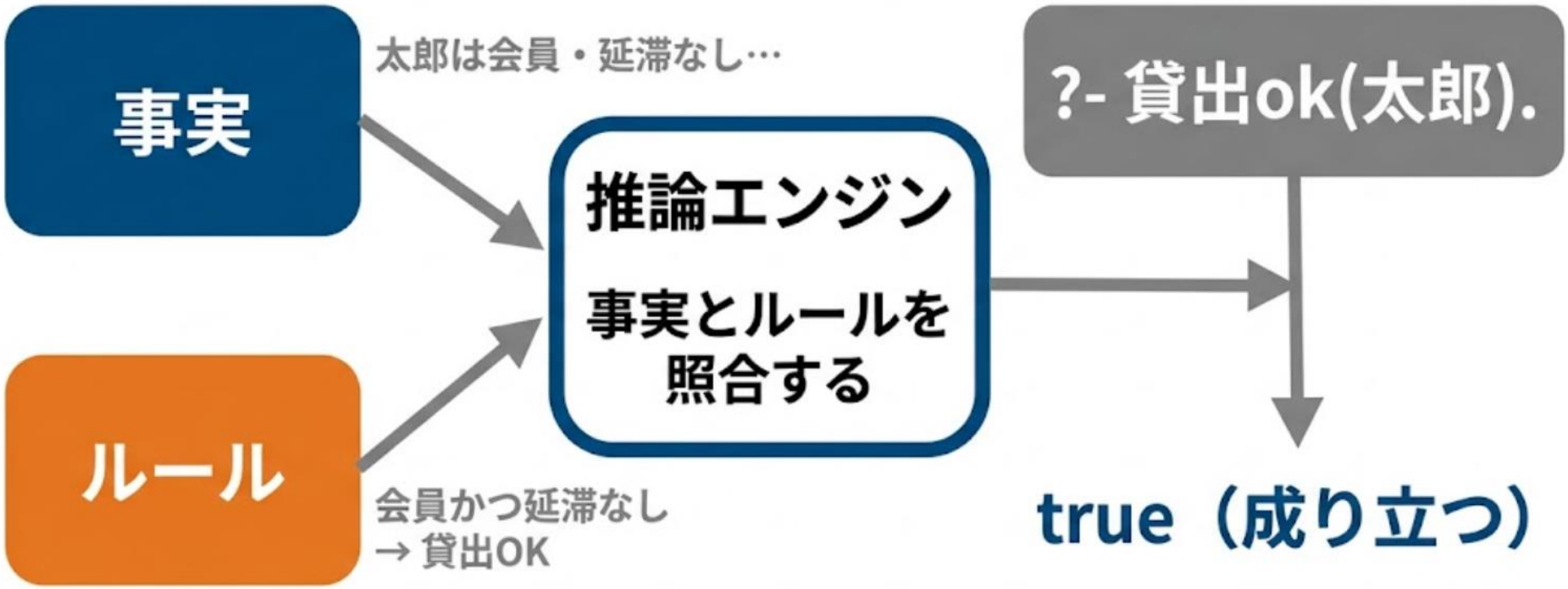
述語論理は『**述語＋引数**』で**事実やルールを表現**する。
Prolog はそれをプログラムとして書き、実行できる



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

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)	結果
?- 貸出ok(太郎).	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)	結果
?- 貸出ok(X).	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 ∧ no_overdue）、三郎はルール2（staff）で成り立つ

質問 (Prolog inquiry format)	結果
?- 貸出ok(花子).	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).

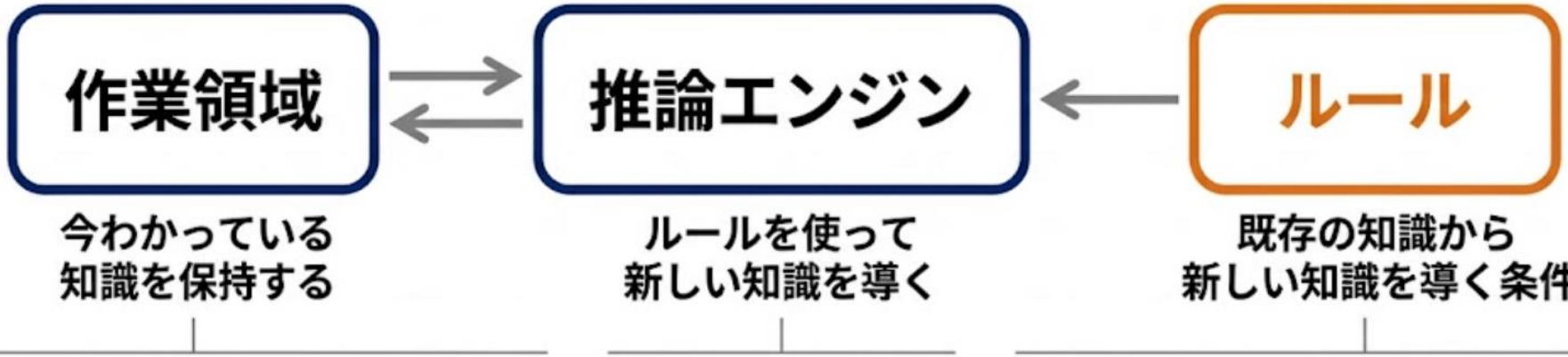
花子が貸出okだと証明できない (不明)

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**

推論エンジン

『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）

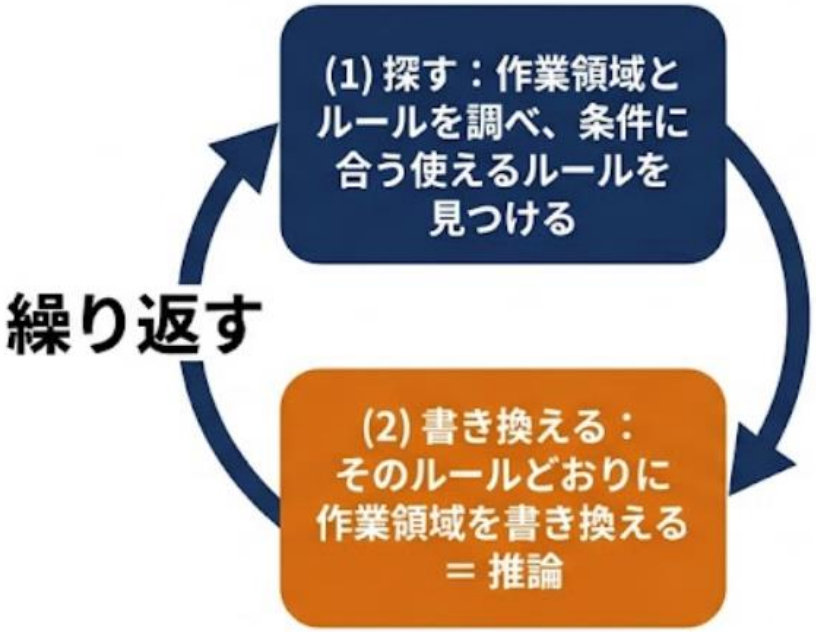
- 貸出ok(太郎)
- 貸出ok(次郎)
- 貸出ok(三郎)

```
% 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



(3) 使えるルールが無くなったら終了
→ 結論を得る

作業領域の変化

```
member(taro).
member(hanako).
member(jiro).
no_overdue(taro).
staff(saburo).
no_overdue(saburo).
```

貸出ok(太郎)

貸出ok(次郎)

貸出ok(三郎)

使えるルールが尽きる→終了

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

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出る

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

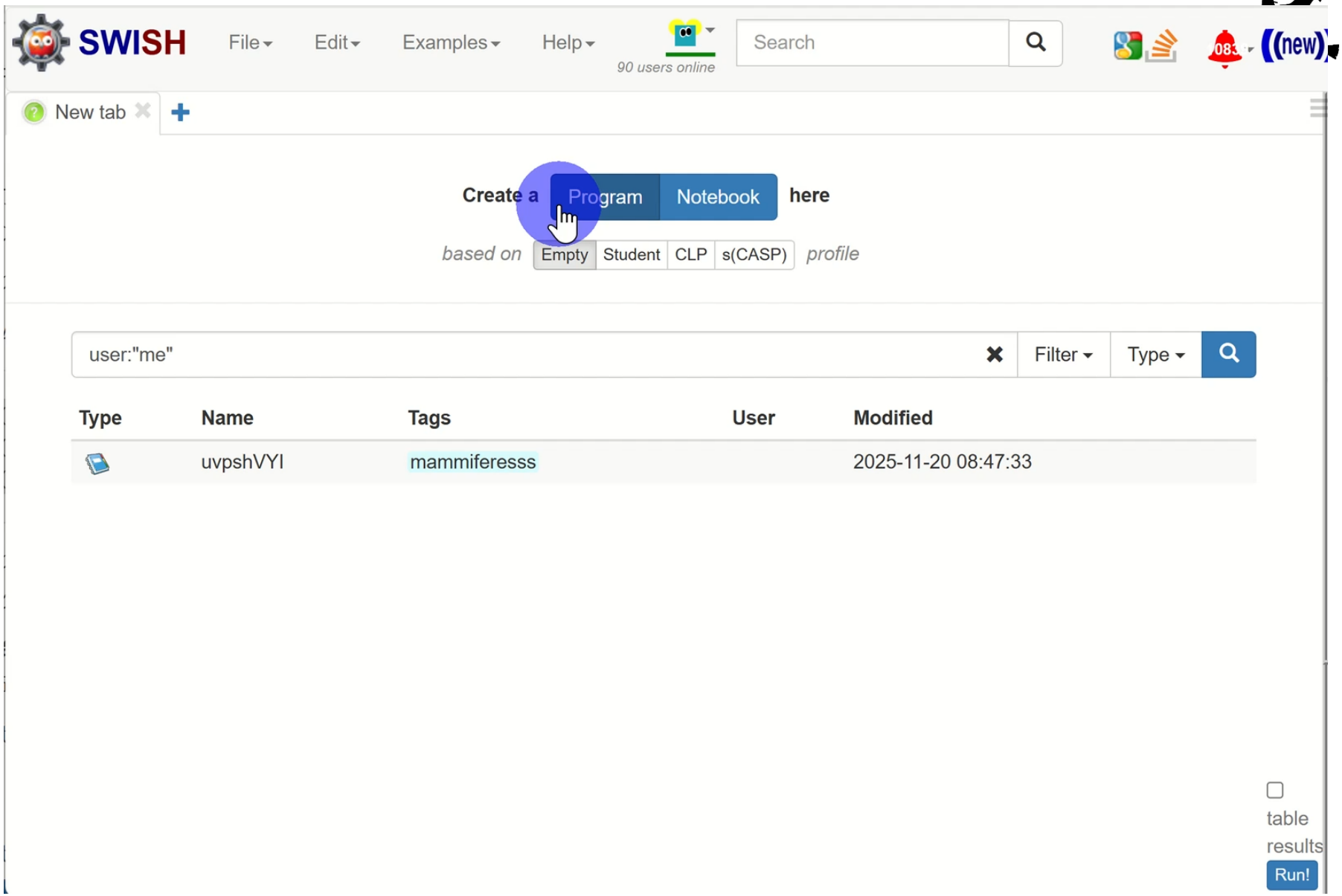
貸出OK
太郎
次郎
三郎

2つの方法は得意が違う ― 目的で使い分ける

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

ルールが増えると、SQLは書き換えが必要になる

演習


SWISH
File
Edit
Examples
Help
90 users online
Search
New tab
Create a Program Notebook here
based on Empty Student CLP s(CASP) profile
user:"me"
Filter
Type
Type Name Tags User Modified
uvpshVYI mammiferesss 2025-11-20 08:47:33
table results
Run!

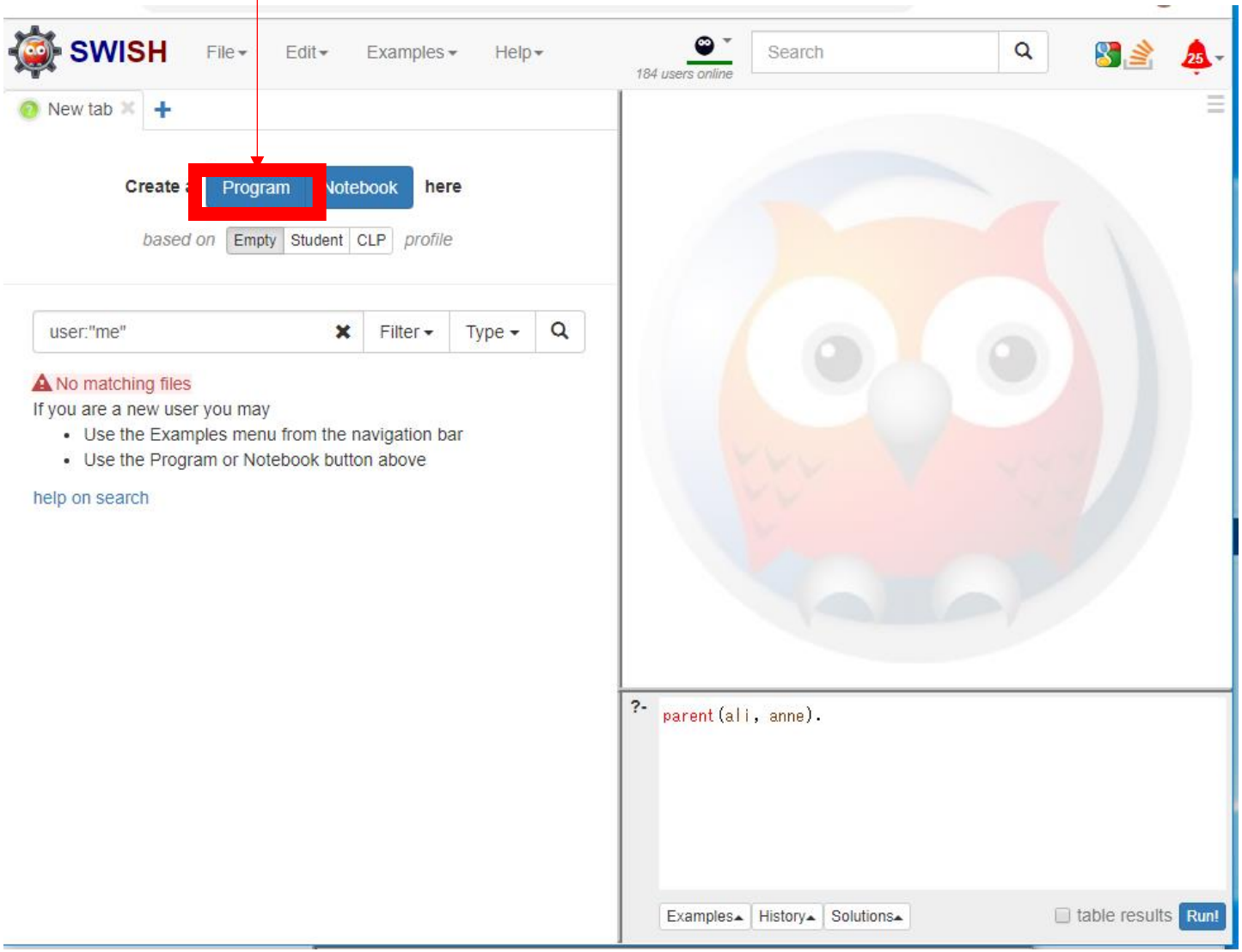
演習１．Prolog による推論

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

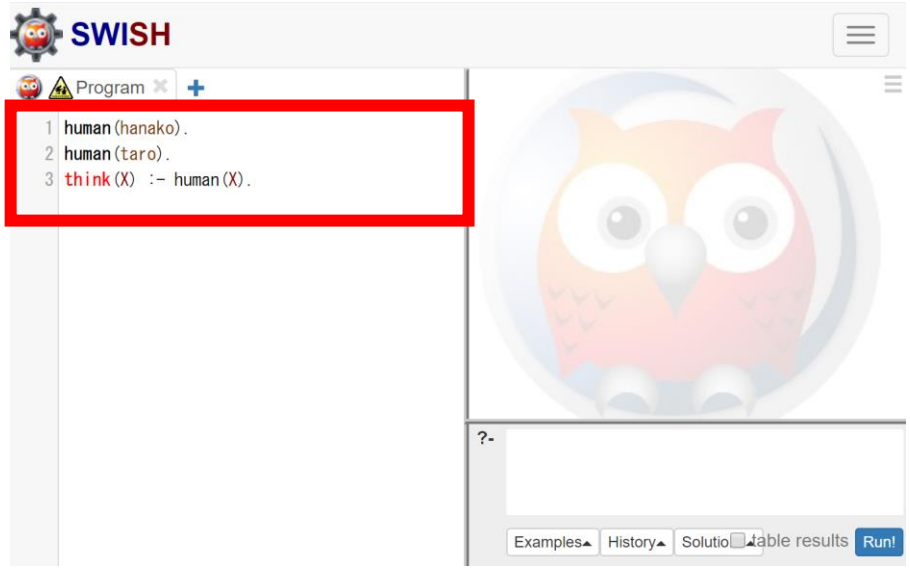
② Prologの画面に変わる.
「Program」をクリック.




SWISH
File
Edit
Examples
Help
181 users online
Search
Program
Your Prolog rules and facts go here ...
事実，規則
の編集
答えが表示される
parent(ali, anne).
問い合わせをコンピュータに
与える
table results
Run!
事実，規則
をコンピュータに
読み込ませるボタン

③ Prologプログラムの準備

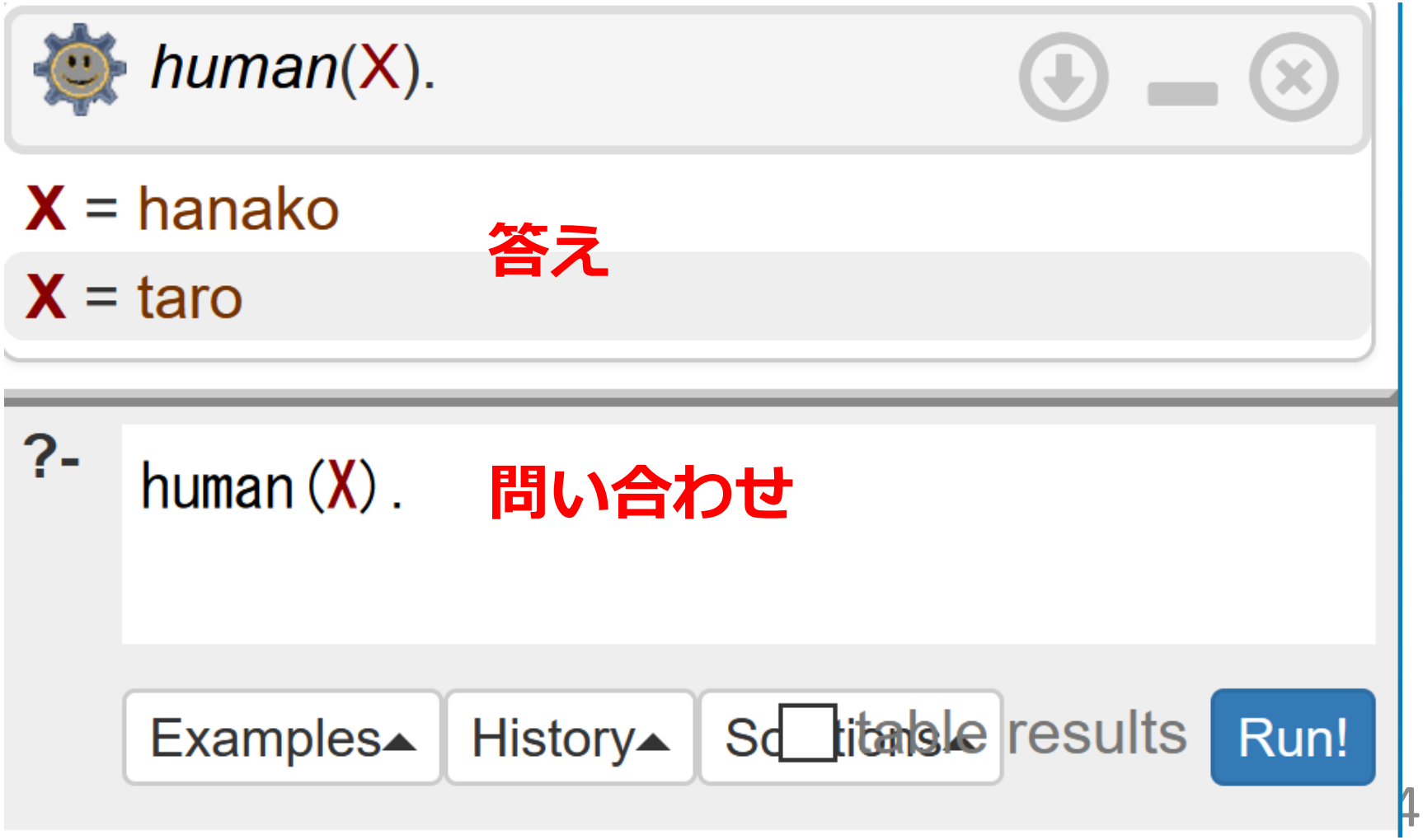
```
human(hanako).
human(taro).
think(X) :- human(X).
```



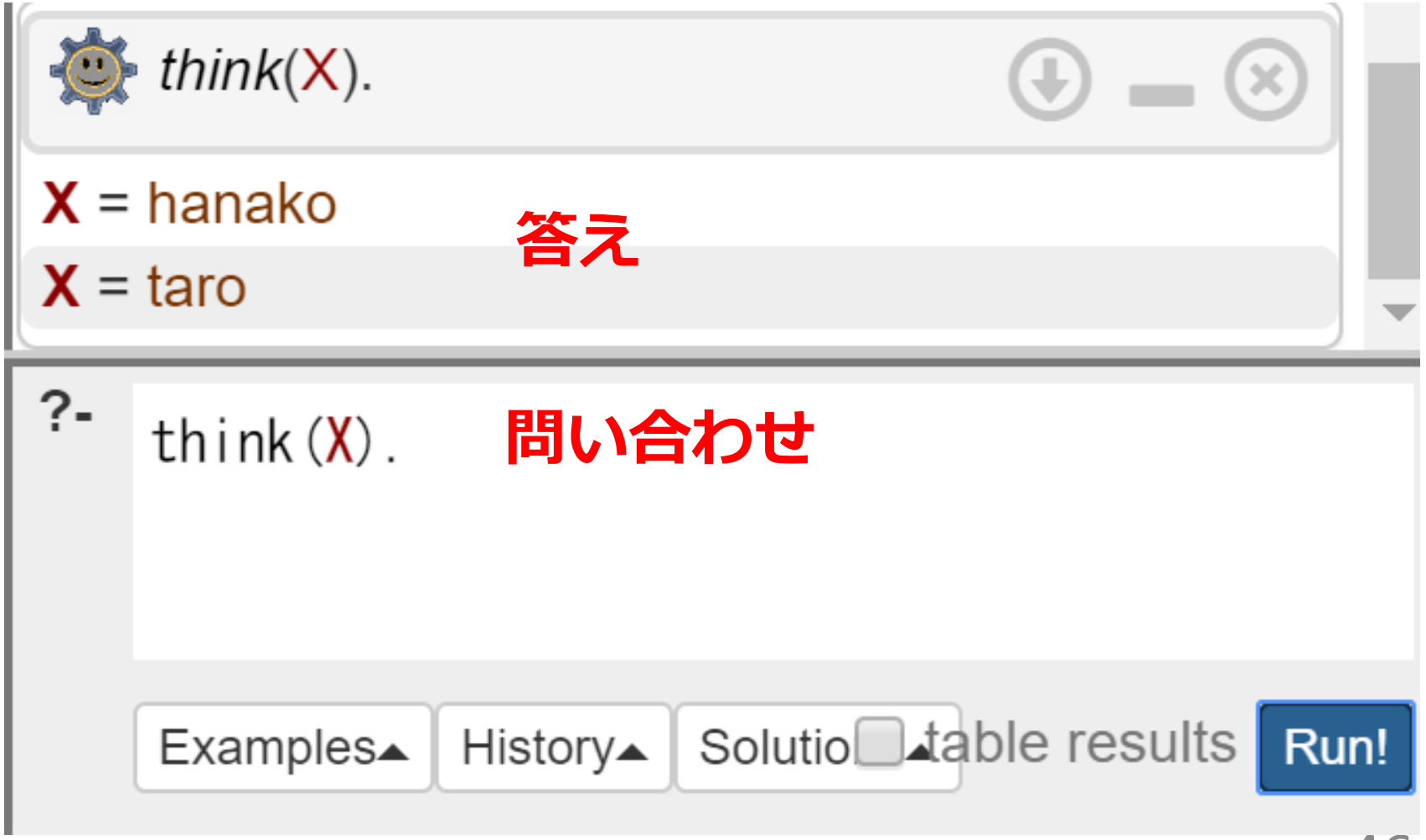
Prologプログラム

編集画面を使用

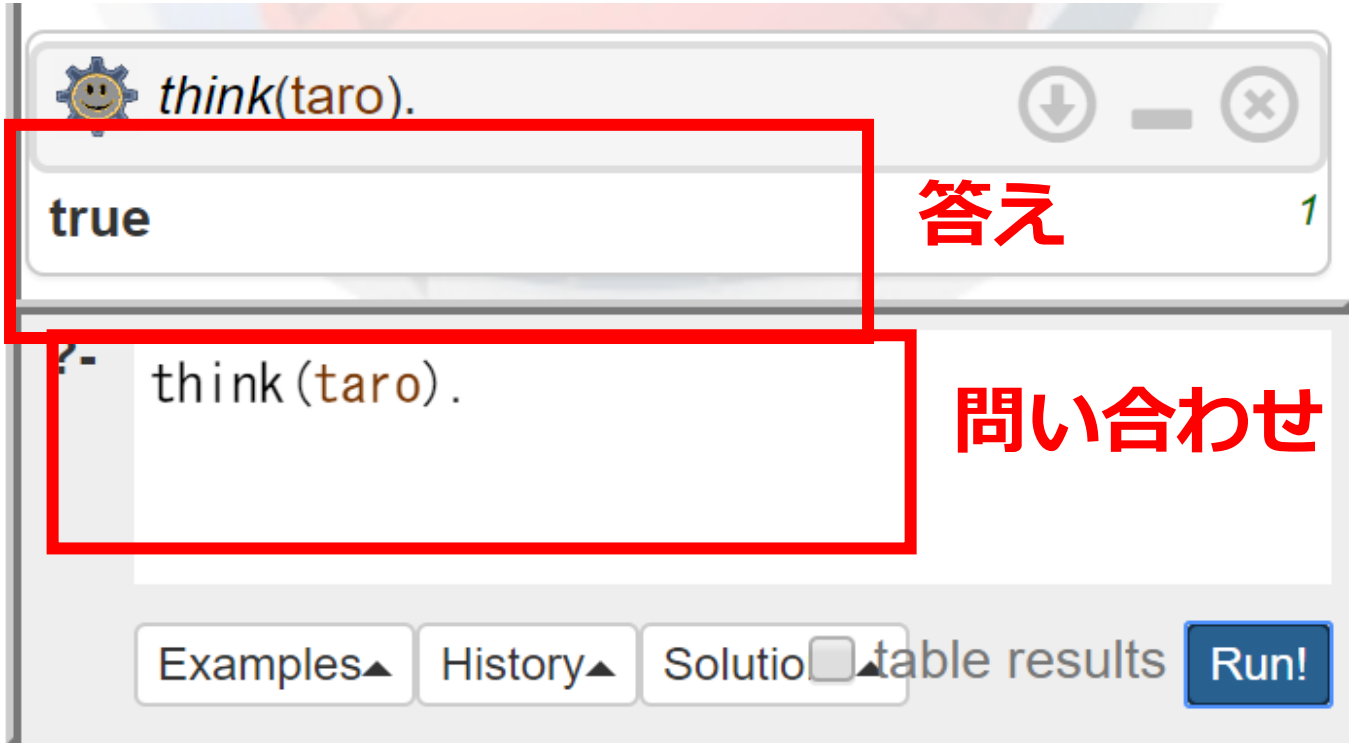
④ 問い合わせ「 **human(X).** 」を入れ，「**Run**」ボタン
そして「**Next**」ボタンをクリック
（答えが増えなくなるまで）



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）

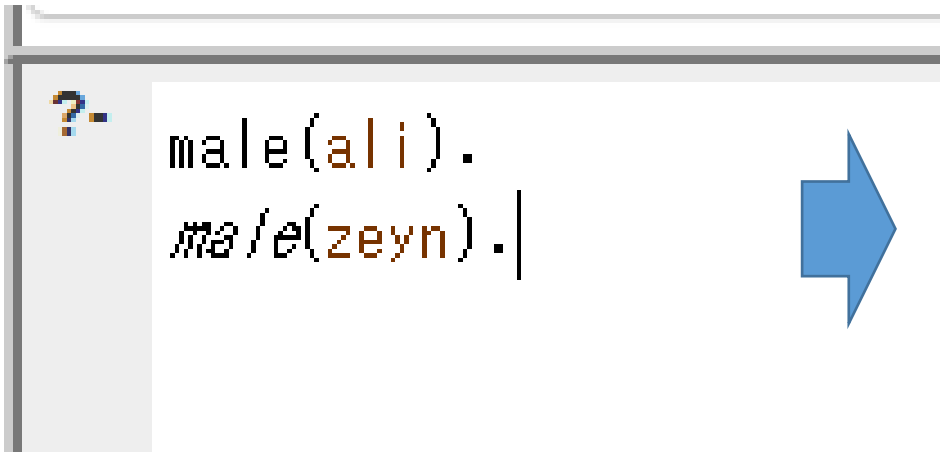


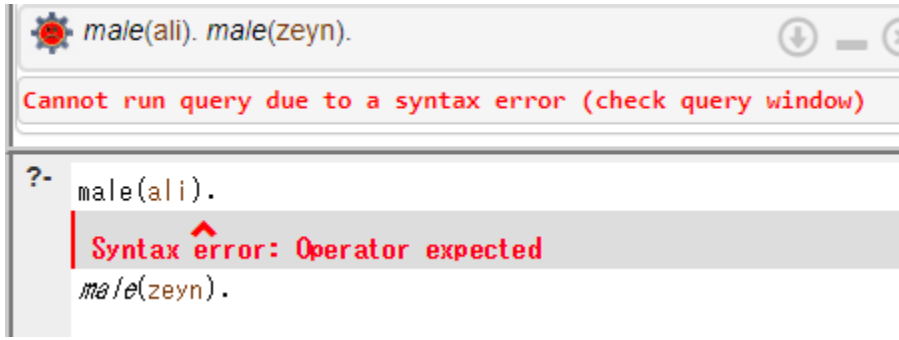
⑥ 問い合わせ「 **think(taro).** 」を入れ，「**Run**」ボタン
think かっこ taro かっこ ピリオド



「true」を確認

問い合わせを，1度に 2個書くことはできない





問い合わせを書くたびに，
前の問い合わせは消す．

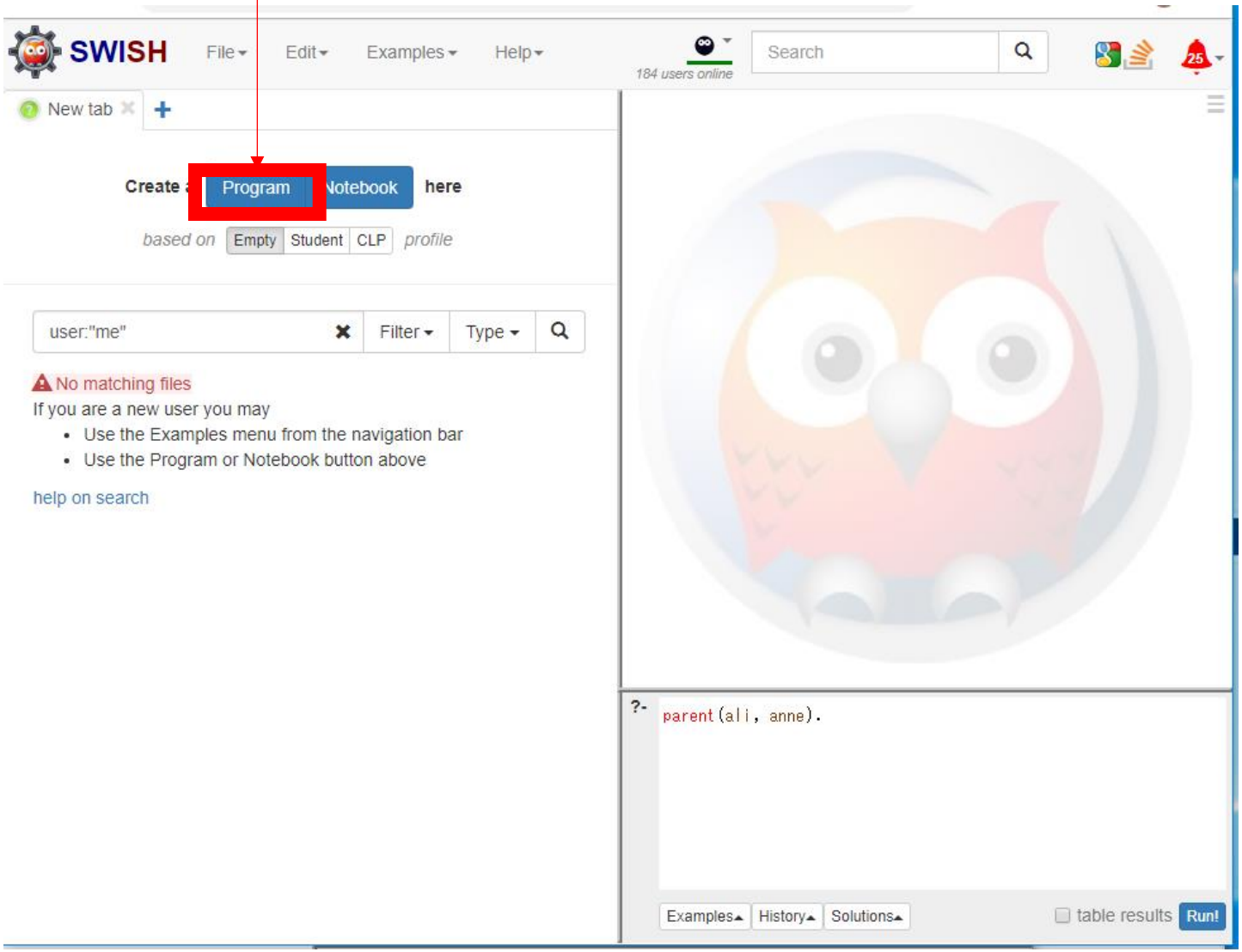
Prolog の true, false

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

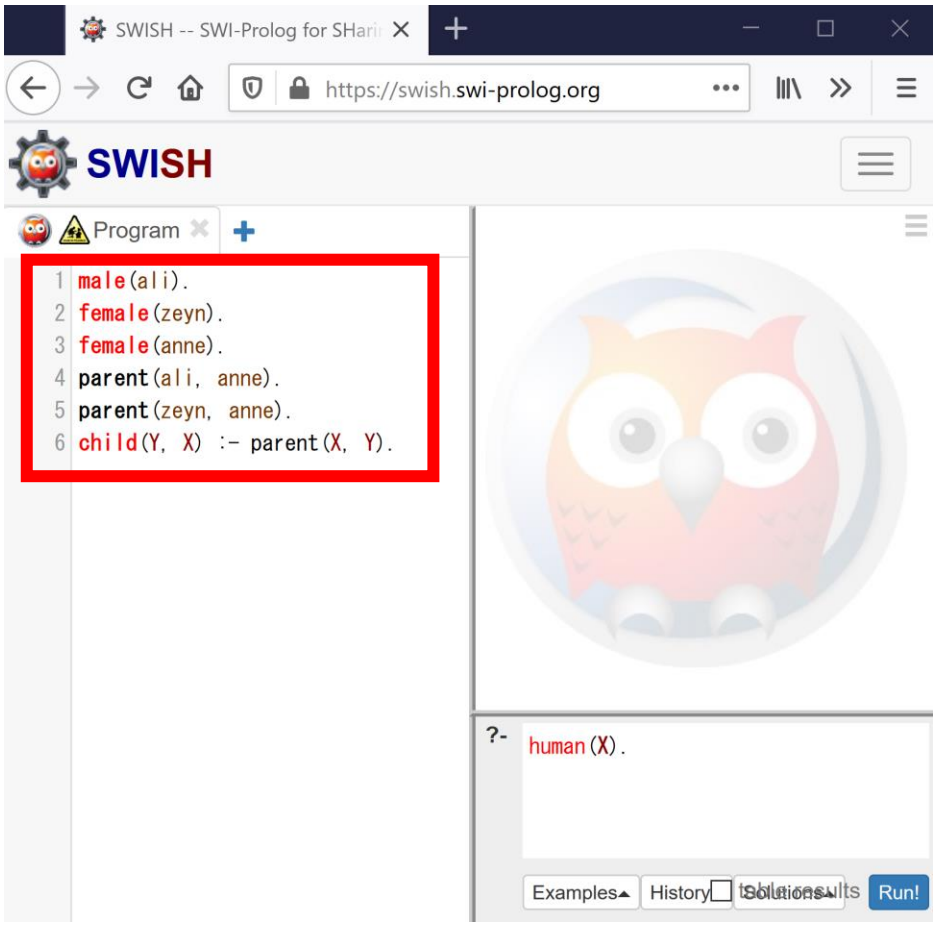
Prolog の画面に変わる.
「Program」をクリック.



Prologプログラムの準備

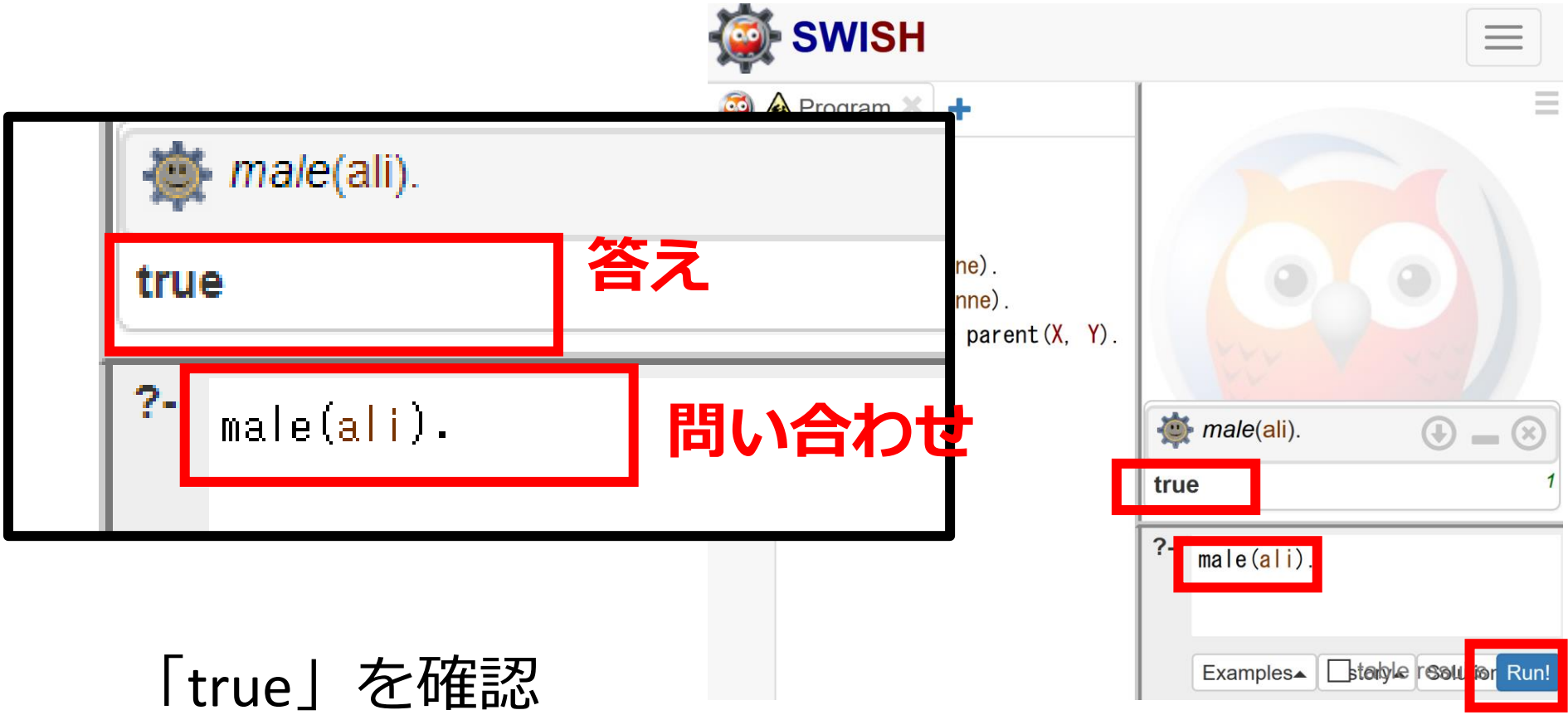
```
male(ali).
female(zeyn).
female(anne).
parent(ali, anne).
parent(zeyn, anne).
child(Y, X) :- parent(X, Y).
```

Prologプログラム



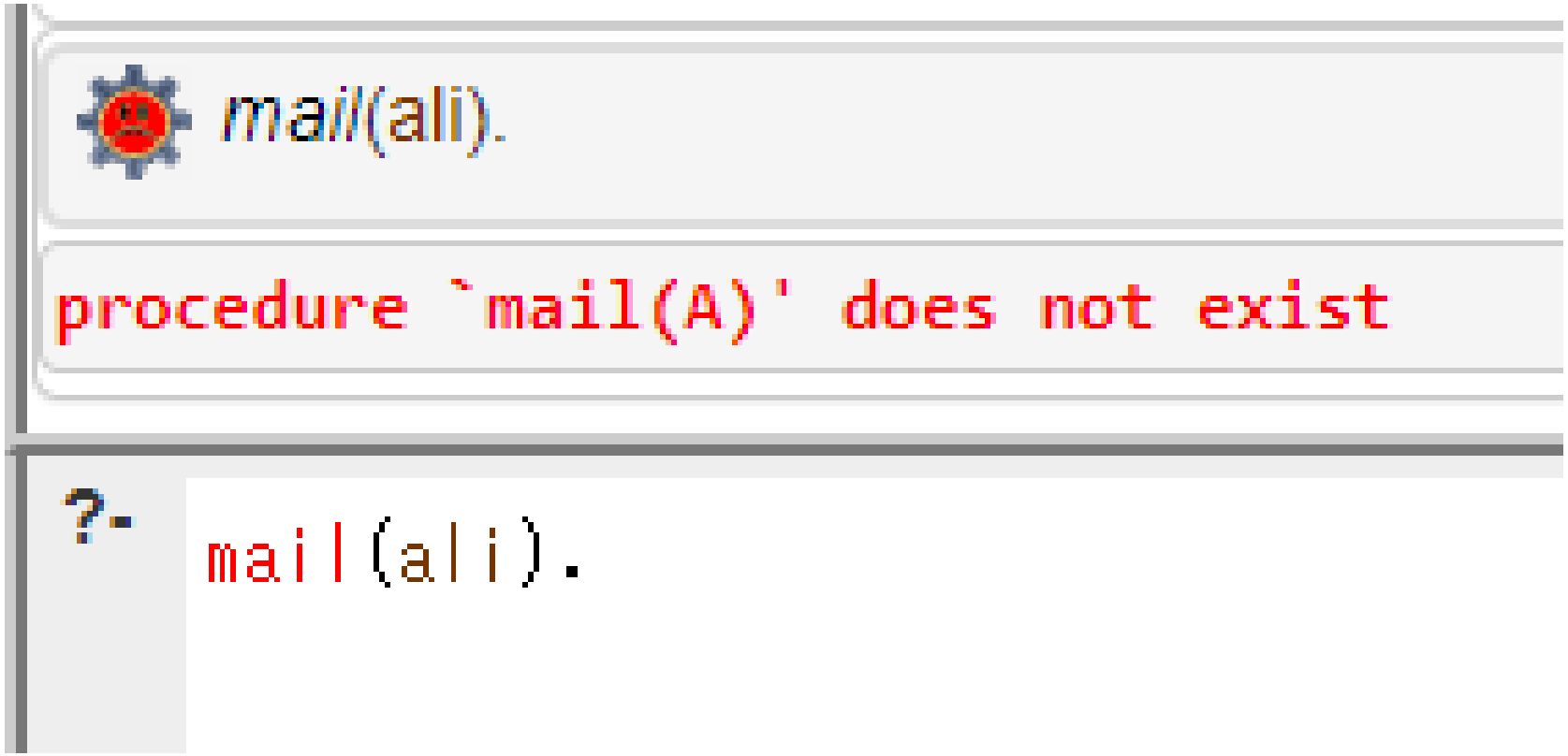
編集画面を使用

問い合わせ「**male(ali).**」を入れ，「**Run**」ボタン
male　かっこ　ali　かっこ　ピリオド　Enterキー



「true」を確認

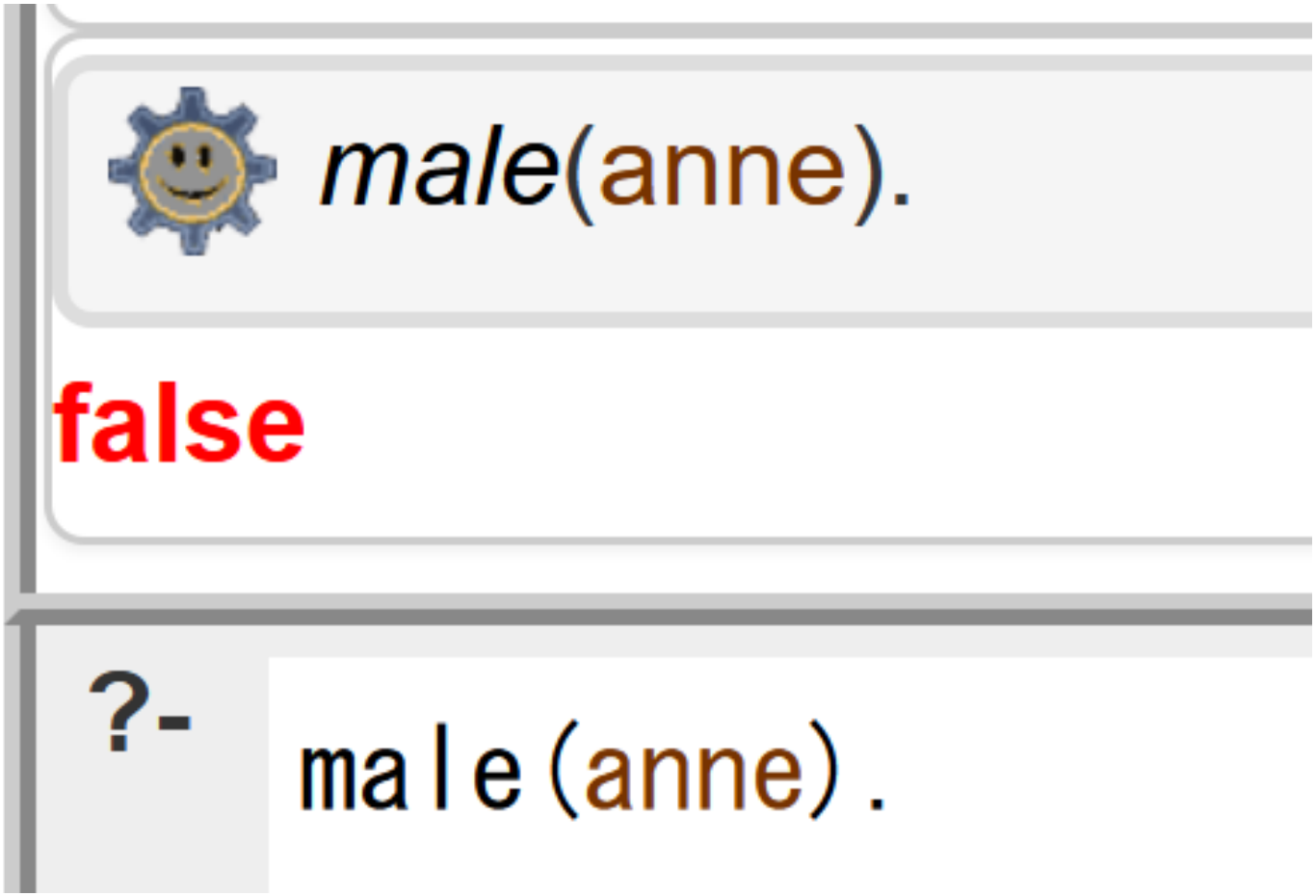
エラーメッセージが出ることがある



正しくは「male」

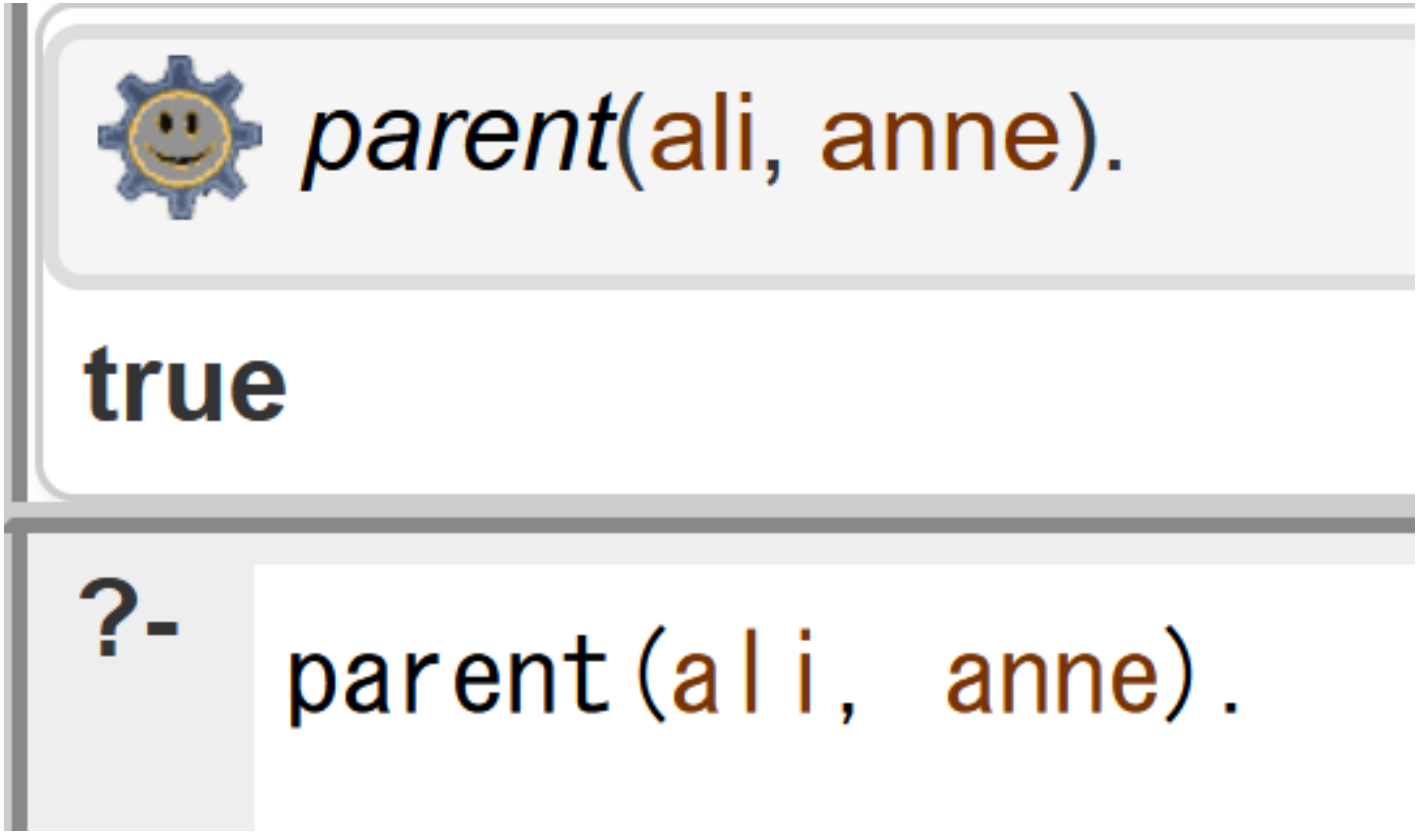
間違って「mail」と書いてしまった

問い合わせ「 **male(anne).** 」を入れ，「**Run**」ボタン
male　かっこ　anne　かっこ　ピリオド　Enterキー



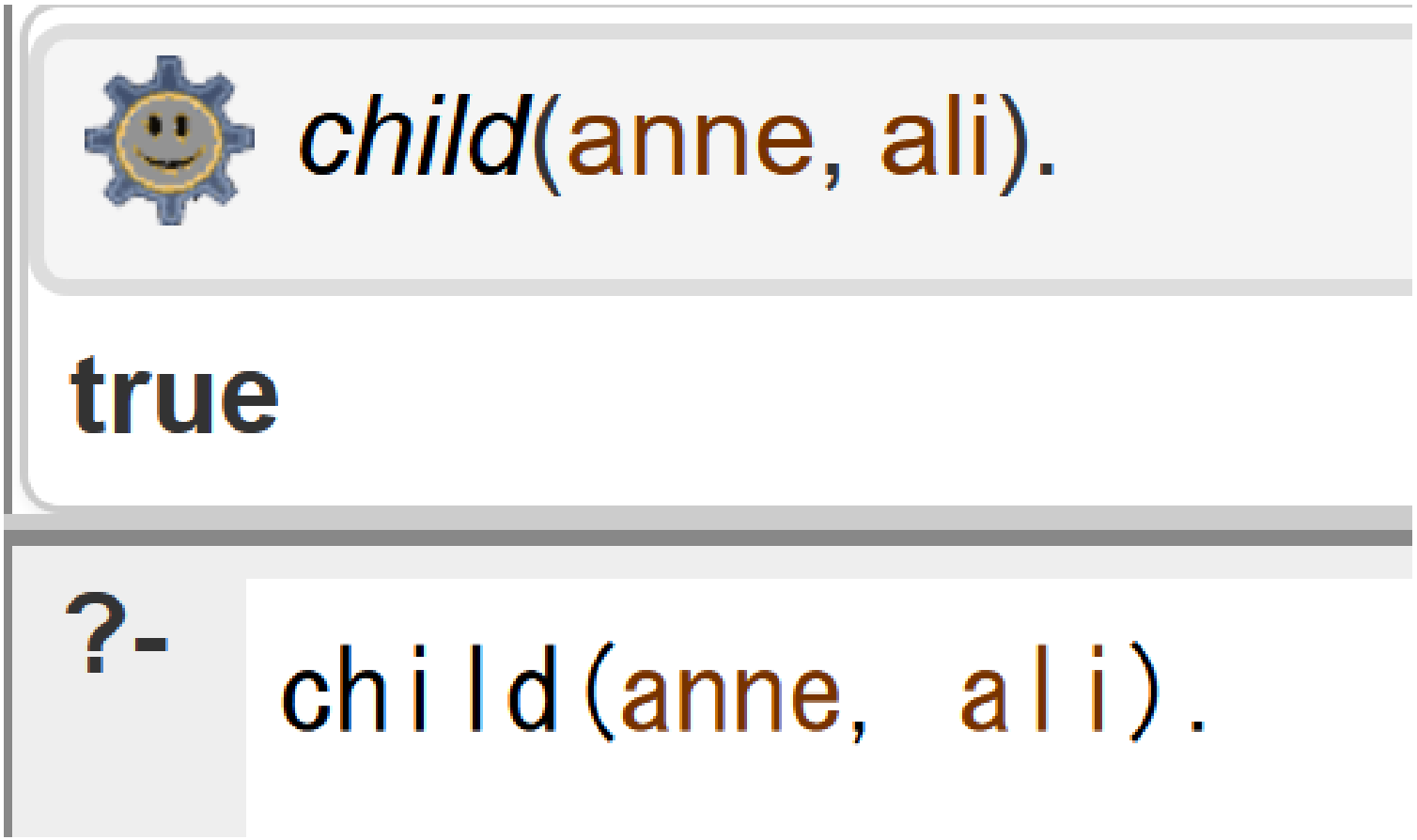
「false」を確認

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



「true」を確認

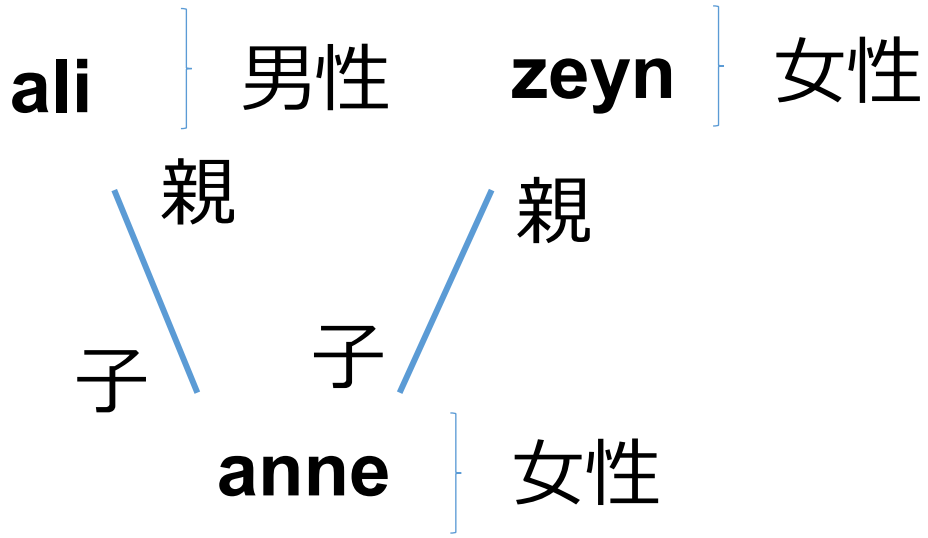
問い合わせ「 **child(anne, ali).** 」を入れ，「**Run**」ボタン



「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

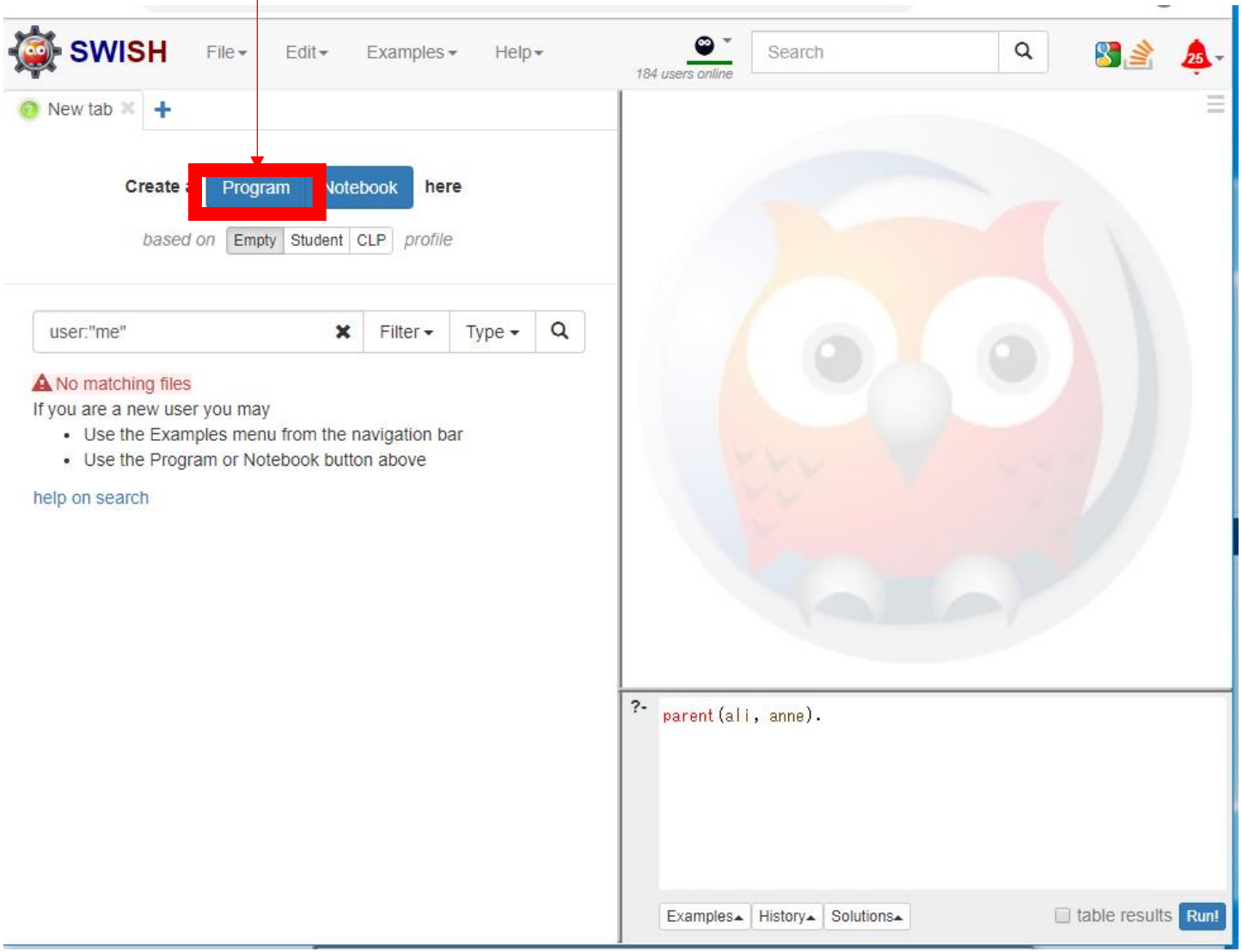
演習３．様々な問い合わせ

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

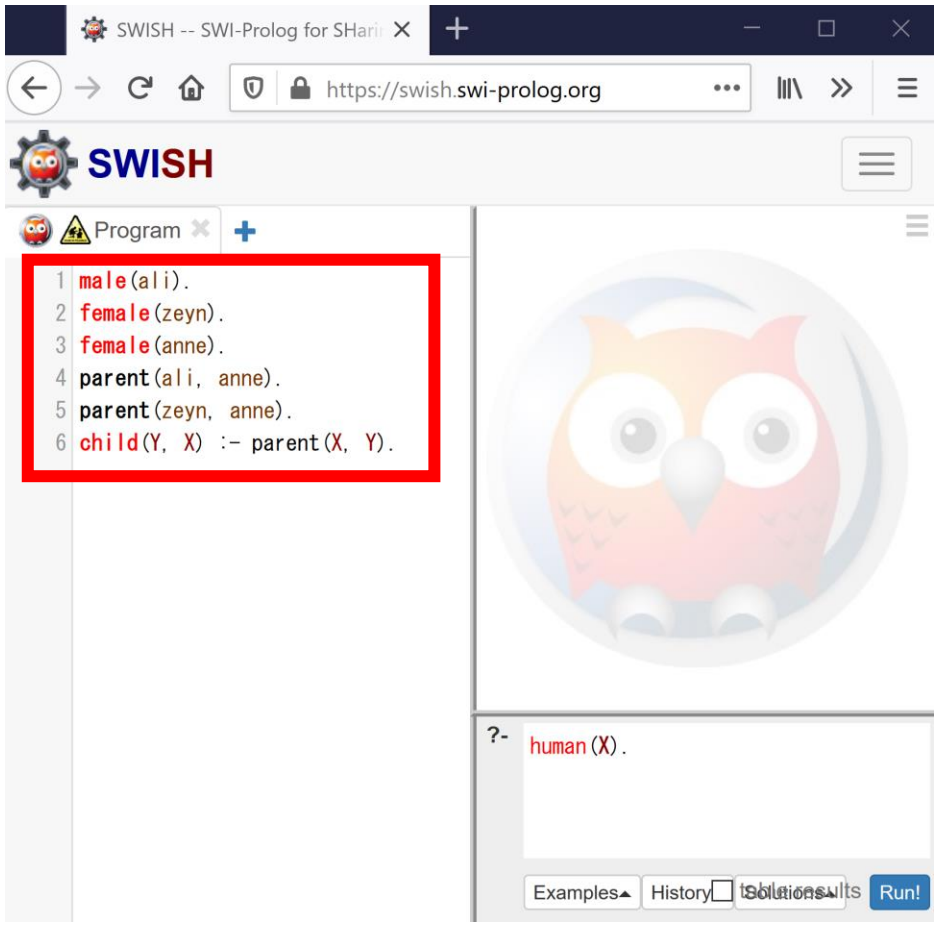
Prolog の画面に変わる.
「Program」をクリック.



「プログラム」の準備

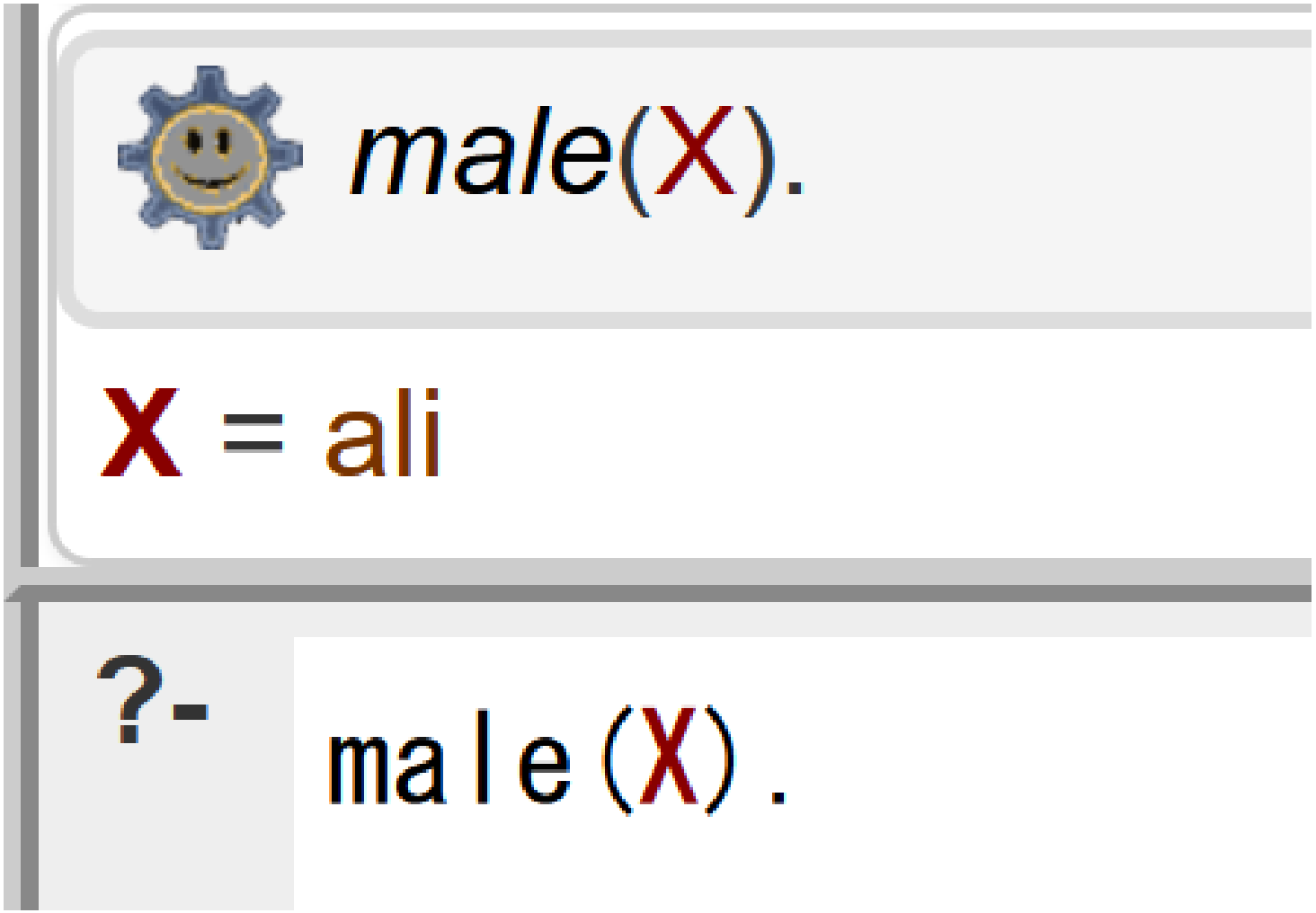
```
male(ali).
female(zeyn).
female(anne).
parent(ali, anne).
parent(zeyn, anne).
child(Y, X) :- parent(X, Y).
```

Prolog プログラム

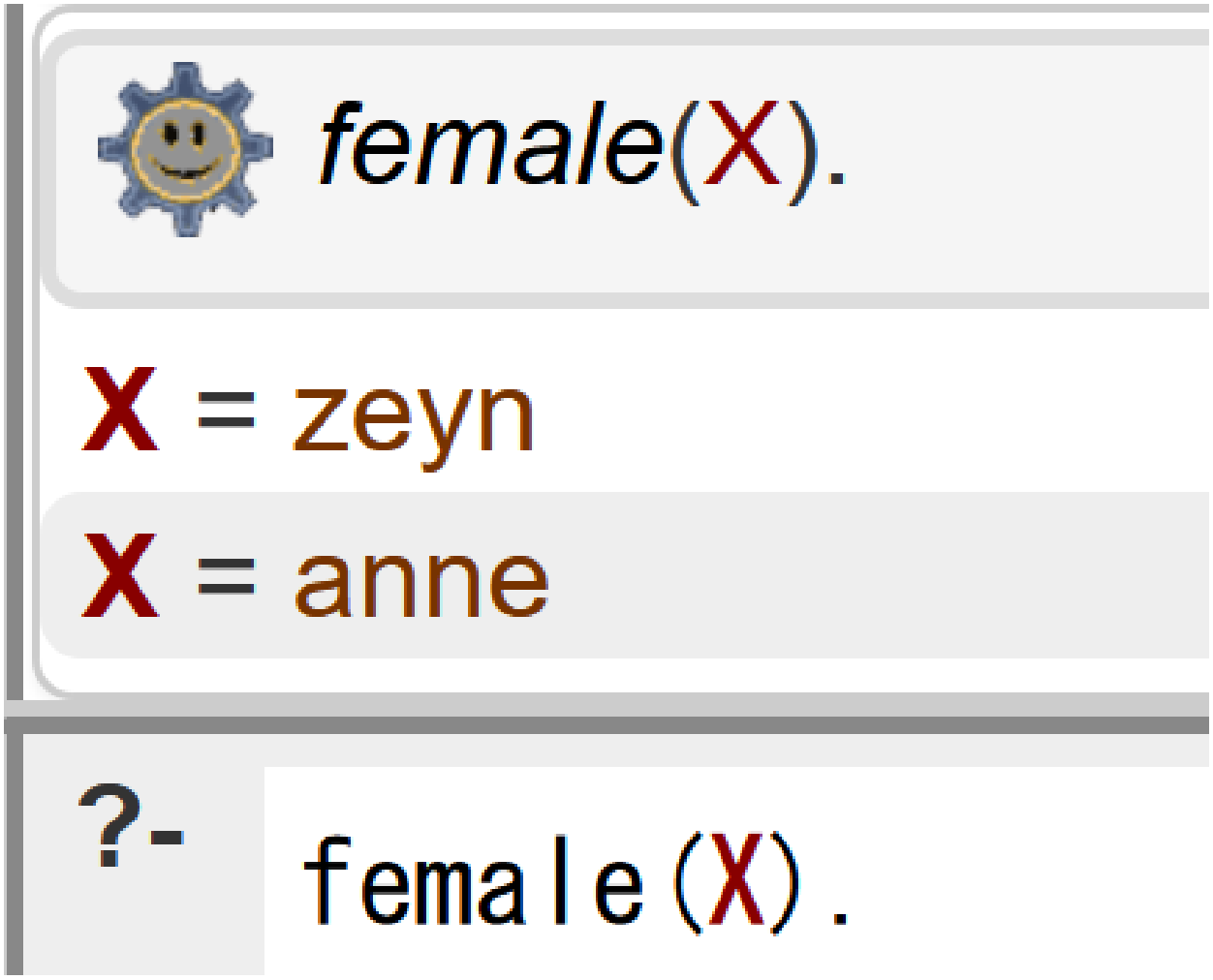


編集画面を使用

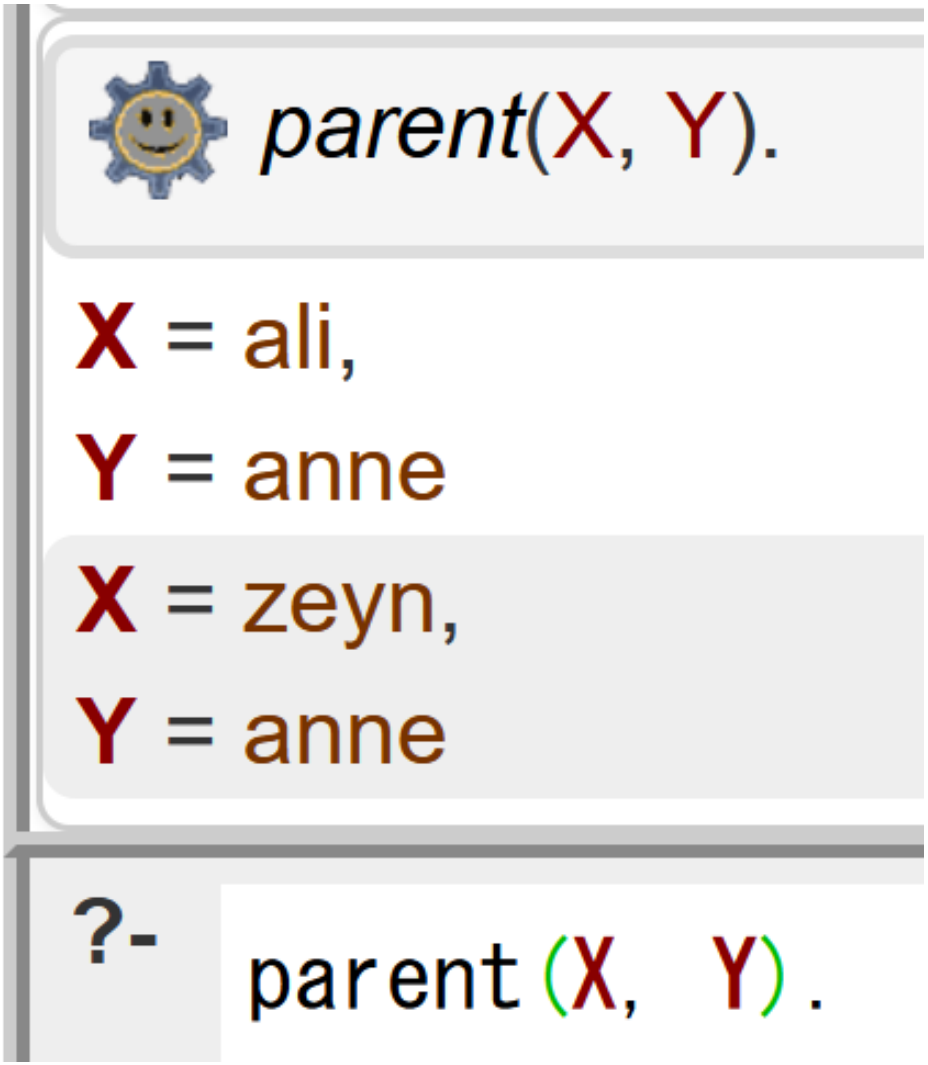
問い合わせ「 **male(X).** 」を入れ，「Run」ボタン



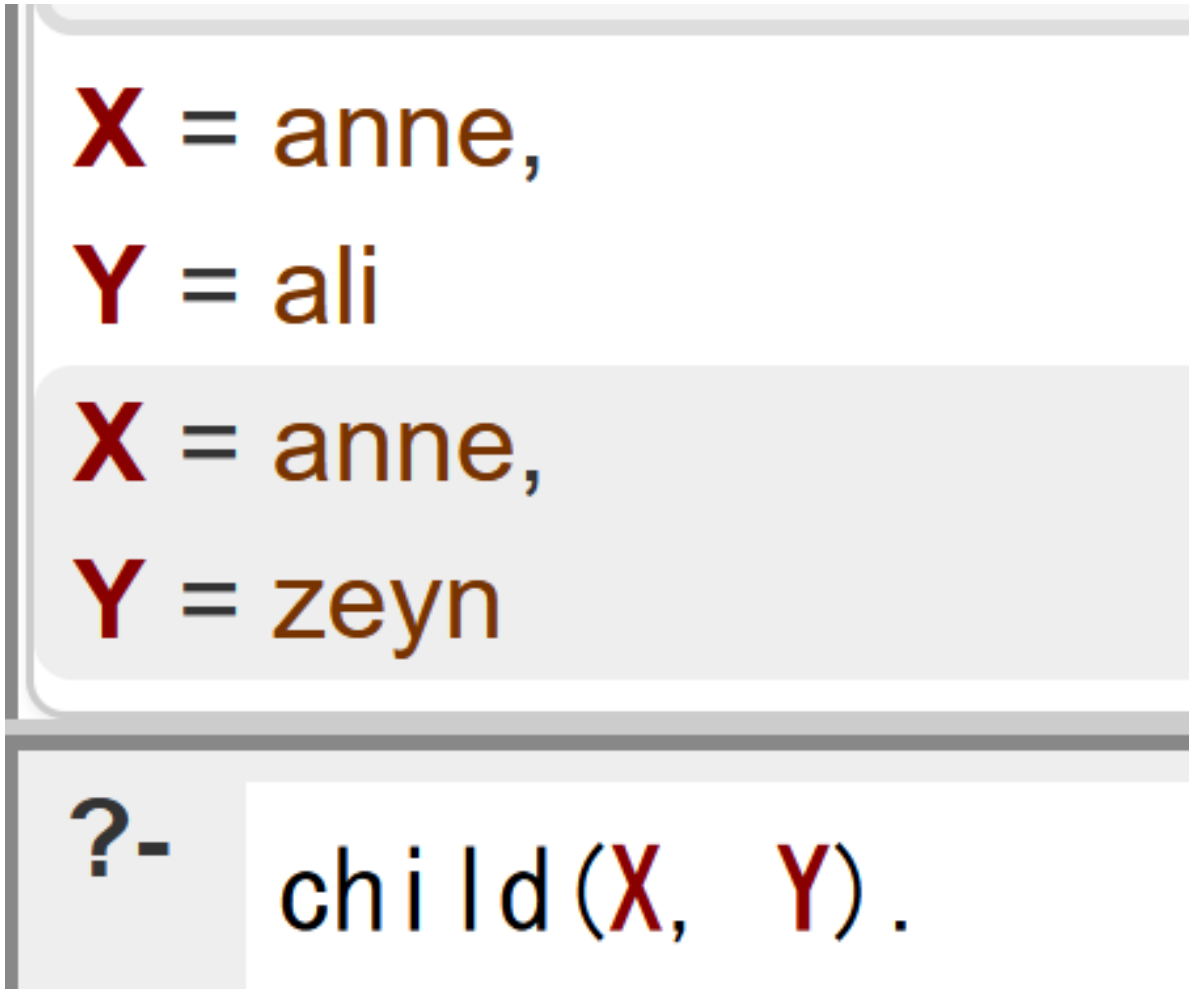
問い合わせ「 **female(X).** 」を入れ，「Run」ボタン
そして「Next」ボタンをクリック
（答えが増えなくなるまで）



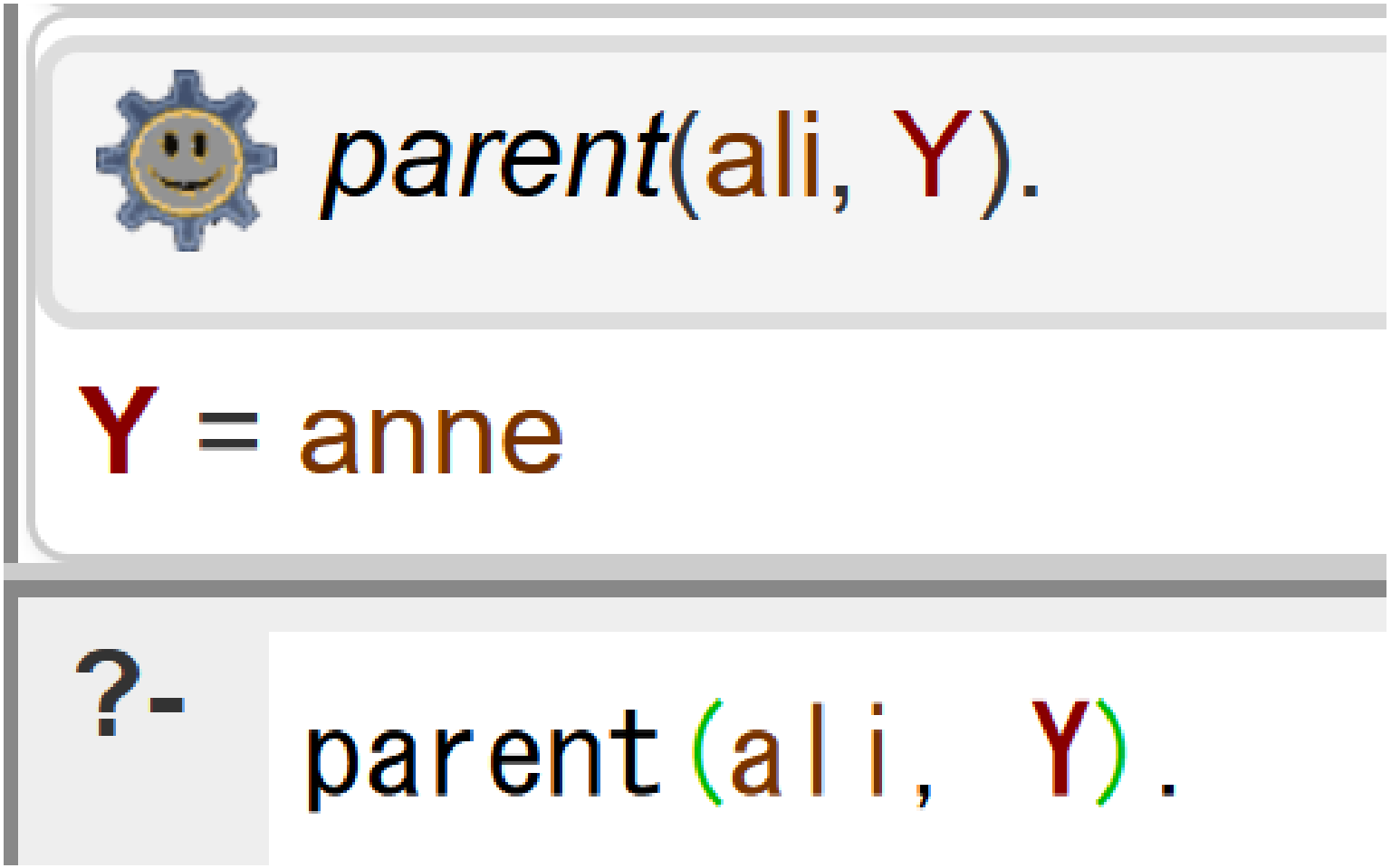
問い合わせ「 **parent(X, Y).** 」を入れ，「Run」ボタン
そして「Next」ボタンをクリック
（答えが増えなくなるまで）



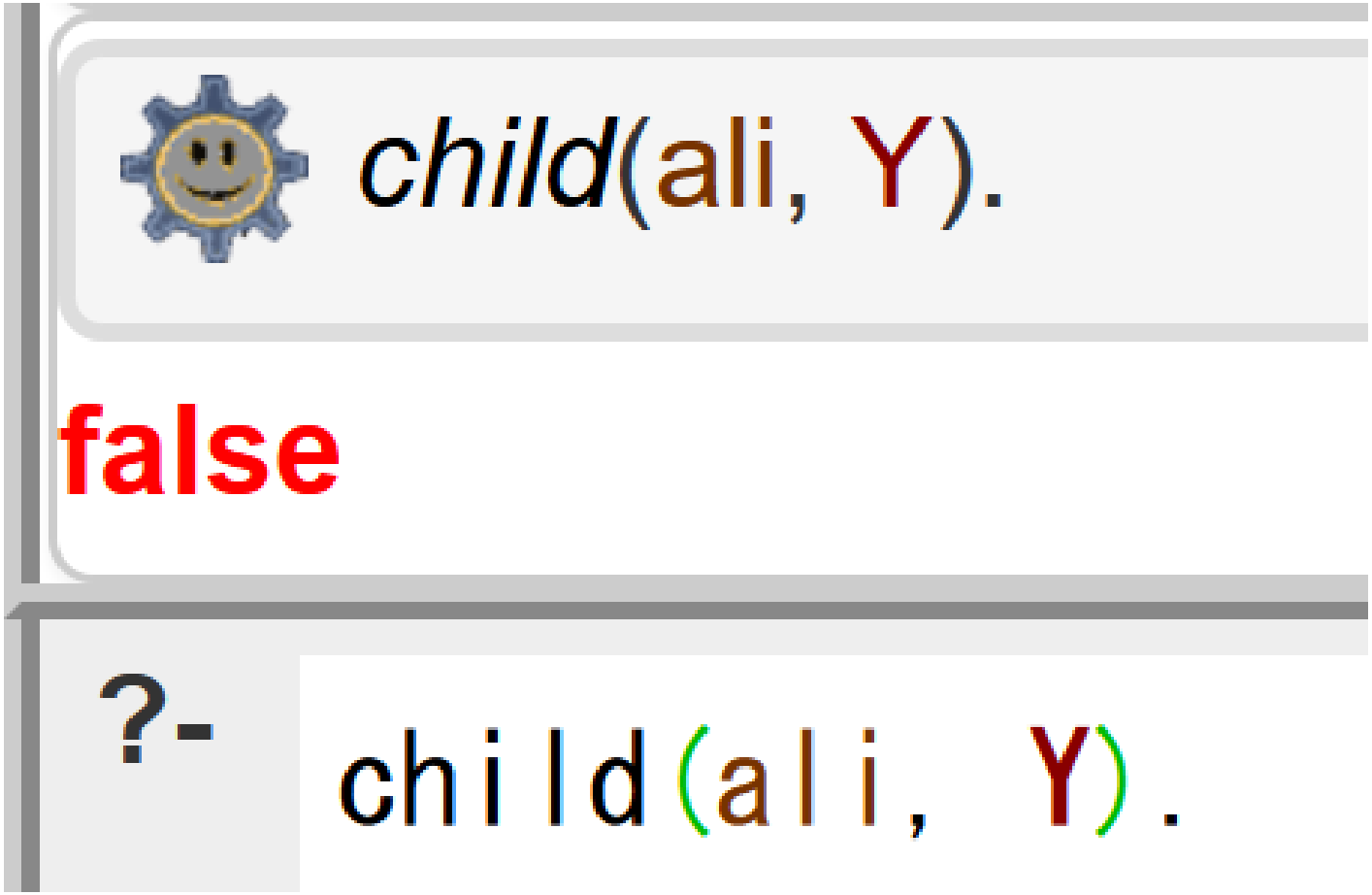
問い合わせ「 **child(X, Y).** 」を入れ，「Run」ボタン
そして「Next」ボタンをクリック
（答えが増えなくなるまで）



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



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



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