MACQUARIE University

Controlled Natural Language

Rolf Schwitter

Rolf.Schwitter@mq.edu.au

Today's Agenda

- What are Controlled Natural Languages?
- Types of Controlled Natural Languages
- Focus: Controlled Natural Languages for Semantic Systems
- Processable English (PENG^{ASP})
- Specifying Strong and Weak Constraints
- Working with Temporal Information

A Motivating Example

Every student who works is successful.

Every student who studies at Macquarie University works or parties.

It is not the case that a student who is enrolled in Information Technology parties.

Tom is a student.

Tom studies at Macquarie University and is enrolled in Information Technology.

Bob is a student who studies at Macquarie University and does not work.

A Motivating Example

Every student who works is successful.

Every student who studies at Macquarie University works or parties.

It is not the case that a student who is enrolled in Information Technology parties.

Tom is a student.

Tom studies at Macquarie University and is enrolled in Information Technology.

Bob is a student who studies at Macquarie University and does not work.

A Motivating Example

Every student who works is successful.

Every student who studies at Macquarie University works or parties.

It is not the case that a student who is enrolled in Information Technology parties.

Tom is a student.

Tom studies at Macquarie University and is enrolled in Information Technology.

Bob is a student who studies at Macquarie University and does not work.

© Macquarie University 2019

A Motivating Example

Every student who works is successful.

Every student who studies at Macquarie University works or parties.

It is not the case that a student who is enrolled in Information Technology parties.

Tom is a student.

Tom studies at Macquarie University and is enrolled in Information Technology.

Bob is a student who studies at Macquarie University and does not work.

© Macquarie University 2019

What the Heck is that?

Every student who works is successful.

Every student who studies at Macquarie University works or parties.

It is not the case that a student who is enrolled in Information Technology parties.

Tom is a student.

Tom studies at Macquarie University and is enrolled in Information Technology.

Bob is a student who studies at Macquarie University and does not work.

© Macquarie University 2019

It’s a Program

Every student who works is successful.

Every student who studies at Macquarie University works or parties.

It is not the case that a student who is enrolled in Information Technology parties.

Tom is a student.

Tom studies at Macquarie University and is enrolled in Information Technology.

Bob is a student who studies at Macquarie University and does not work.

© Macquarie University 2019

It’s a Logic Program (LP)

Every student who works is successful.

Every student who studies at Macquarie University works or parties.

It is not the case that a student who is enrolled in Information Technology parties.

Tom is a student.

Tom studies at Macquarie University and is enrolled in Information Technology.

Bob is a student who studies at Macquarie University and does not work.

© Macquarie University 2019

It’s a LP in Controlled Natural Language

Every student who works is successful.

Every student who studies at Macquarie University works or parties.

It is not the case that a student who is enrolled in Information Technology parties.

Tom is a student.

Tom studies at Macquarie University and is enrolled in Information Technology.

Bob is a student who studies at Macquarie University and does not work.

© Macquarie University 2019

We would like to …

- Feed this logic program to a computer.
- Automatically infer that
  - Tom is successful.
  - Tom works.
  - Bob parties.
- Ask questions such as:
  - Who is successful?
  - Who works?
  - Who does not work?

© Macquarie University 2019

Logic Program: Answer Set Program

```
prop(A, successful) :-
   class(A, student), pred(A, work).
pred(B, work) ; pred(B, party) :-
   class(B, student), pred(B, C, study_at), named(C, macquarie_university).
:- class(D, student), prop(D, E, enrolled_in), named(E, information_technology), pred(D, party).
named(1, tom).
class(1, student).
pred(1, 2, study_at).
named(2, macquarie_university).
prop(1, 3, enrolled_in).
named(3, information_technology).
named(4, bob).
class(4, student).
pred(4, 2, study_at).
-pred(4, work).
answer(named(F, G)) :-
  named(F, G), prop(F, successful).
```

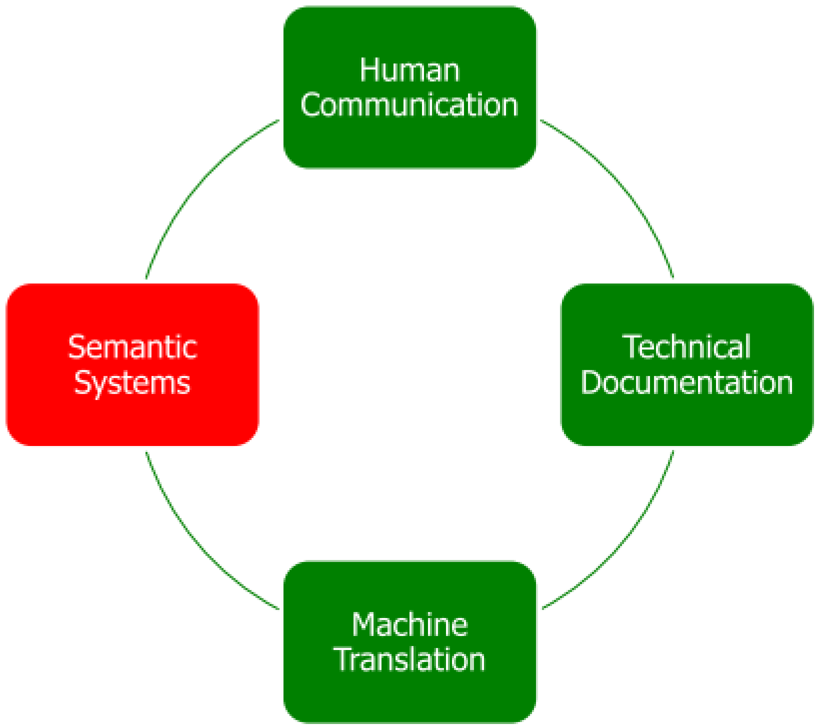
What are Controlled Natural Languages?

Definition:

Controlled natural languages are simplified forms of natural languages; they are constructed from natural languages by restricting the size of the grammar and the vocabulary in order to reduce or eliminate ambiguity and complexity.

© Macquarie University 2019

Types of Controlled Natural Languages



© Macquarie University 2019

CNLs for Semantic Systems

- There exist a number of general-purpose CNLs for KR:
  - Attempto Controlled English
  - Computer Processable Language
  - Processable English (PENG^{ASP}).
- Start from a simple sentence pattern:

  subject + verb + complements + adjuncts
- Use constructors to build complex sentences.
- CNLs are translated into a formal target language.
- Reasoning: consistency checking & question answering.

© Macquarie University 2019

PENG^{ASP} (Processable English)

- PENG^{ASP} is a controlled natural language (CNL) that serves as a high-level specification language for ASP programs.
- The language processor of the PENG^{ASP} system translates a CNL specification into an ASP program (and vice versa).
- That means the grammar of PENG^{ASP} is bi-directional and can be used for processing and verbalization.

R. Schwitter. Specifying and Verbalising Answer Set Programs in Controlled Natural Language. In: Journal of Theory and Practice of Logic Programming, Vol. 18, Special Issue 3-4, pp. 691-705, 2018.

Answer Set Programming

- Answer Set Programming (ASP) is a form of declarative programming.
- ASP has its roots in:
  - knowledge representation
  - logic programming
  - deductive databases
  - constraint solving.
- The programmer does not specify how to solve a problem but instead what the problem is.

© Macquarie University 2019

Answer Set Programming

```
p(X) ; q(X) :- r(X), not s(X).
r(1).
r(2).
```

```
Answer: 1
r(1) r(2) p(1) q(2)
Answer: 2
r(1) r(2) p(1) p(2)
Answer: 3
r(1) r(2) q(1) q(2)
Answer: 4
r(1) r(2) q(1) p(2)
SATISFIABLE
```

Answer Set Programming

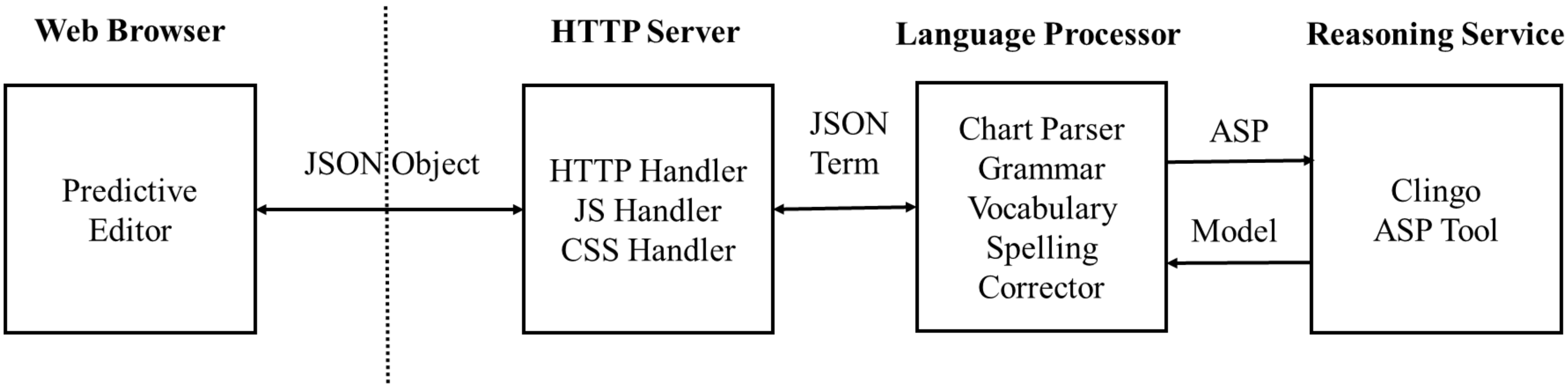
```
p(X) ; q(X) :- r(X), not s(X).
r(1).
r(2).
:- q(2).

Answer: 1
  r(1) r(2) p(2) p(1)
Answer: 2
  r(1) r(2) p(2) q(1)
SATISFIABLE
```

© Macquarie University 2019

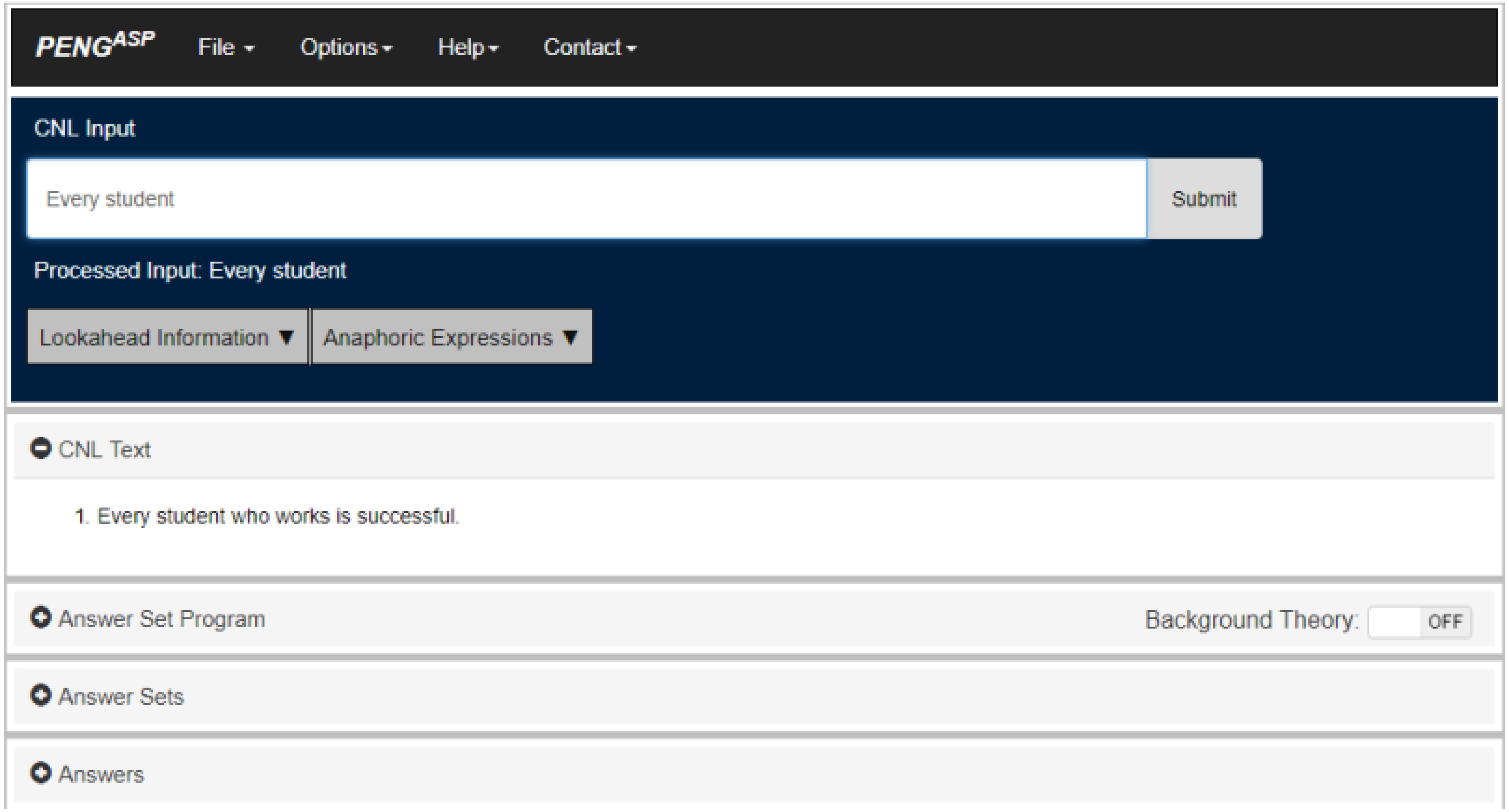
The PENG^{ASP} System

- A predicative text editor is used to guide the writing of a specification in controlled natural language.
- The specification is parsed incrementally during the writing process with the help of a chart parser.
- All natural language processing tasks occur in parallel:
  - anaphoric expressions are resolved,
  - a paraphrase is produced,
  - look-ahead information is generated, and
  - an executable answer set program is produced.

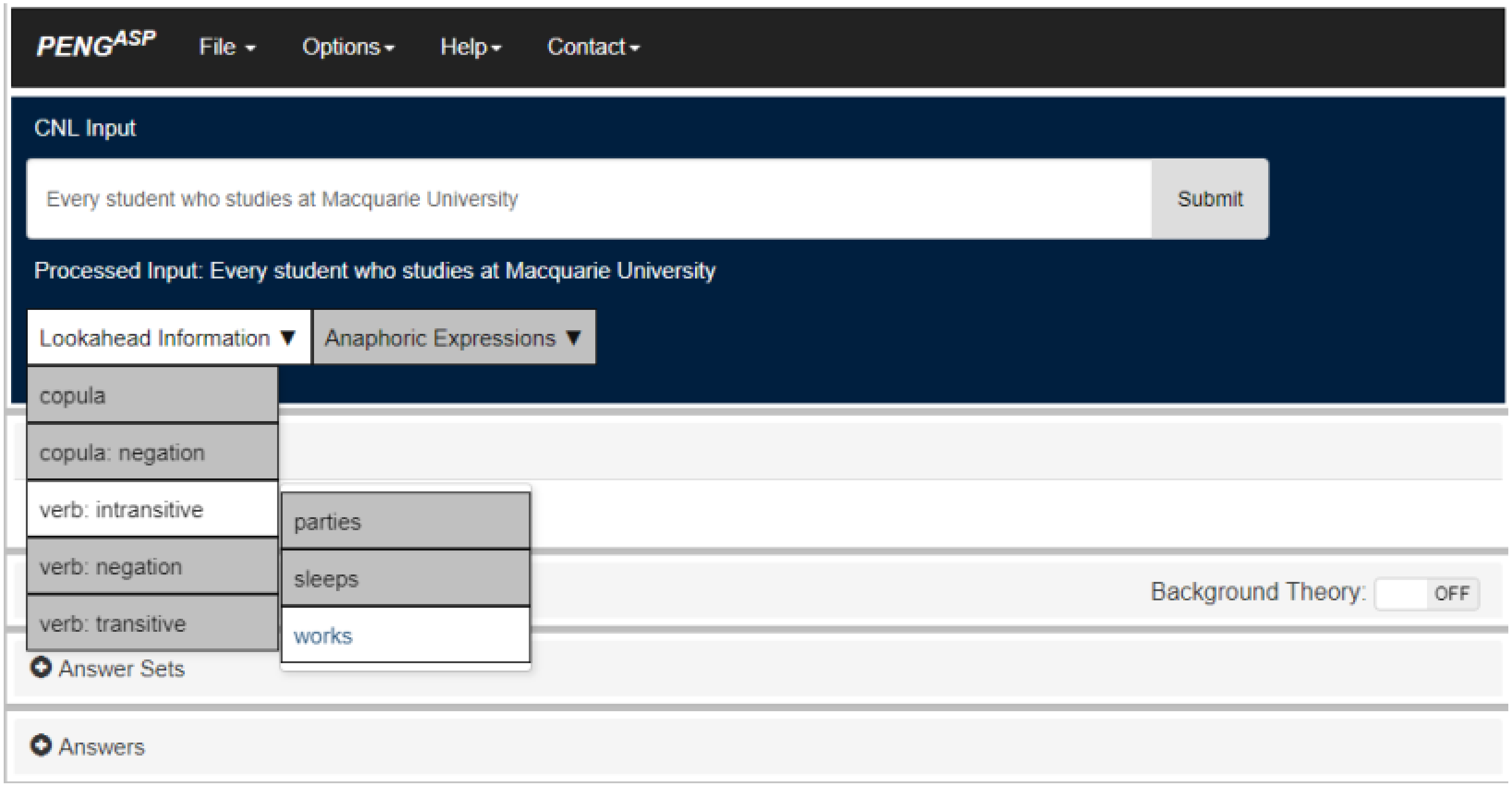
PENG^{ASP} Architecture



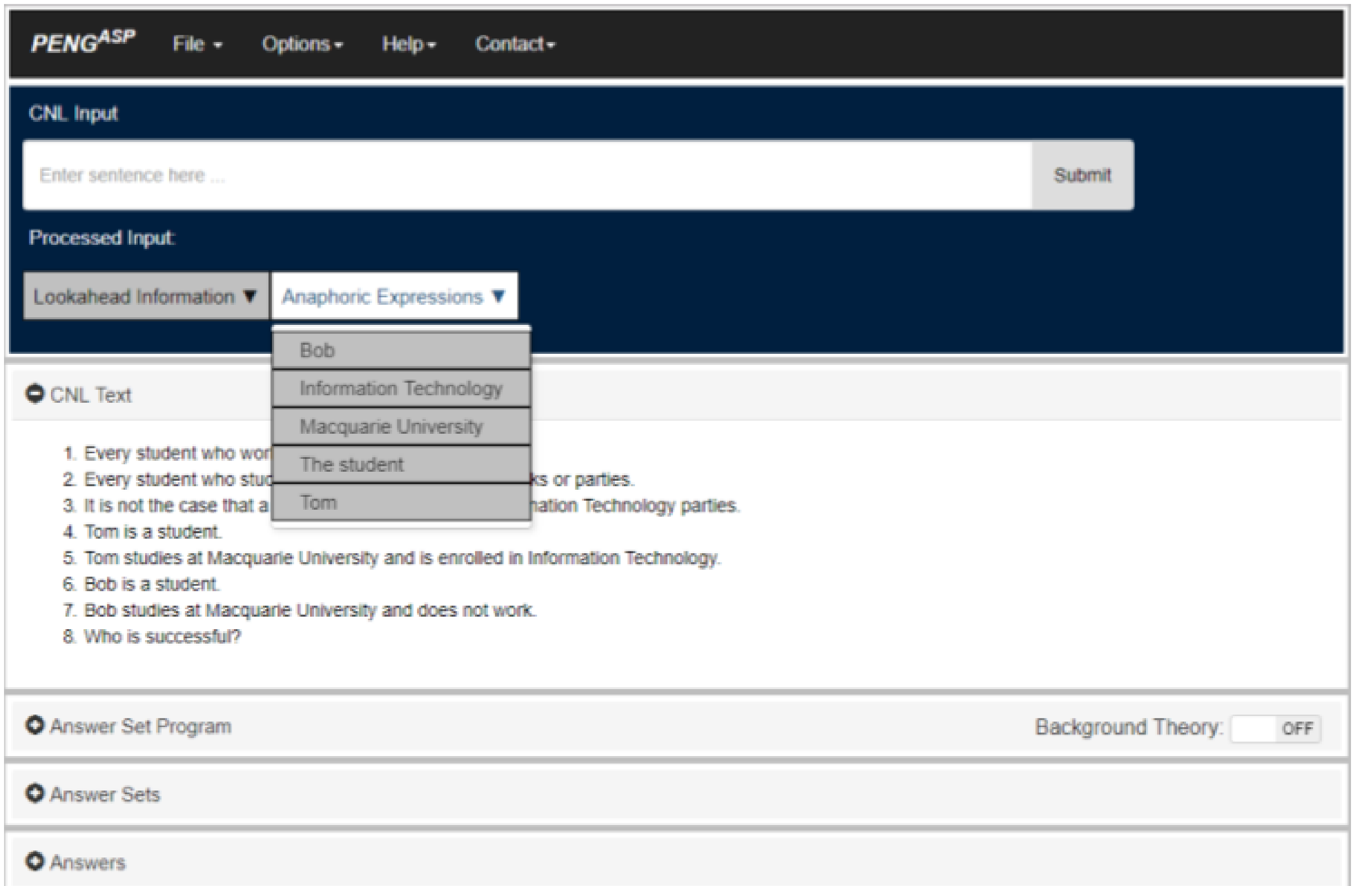
PENG^{ASP}: Predictive Text Editor



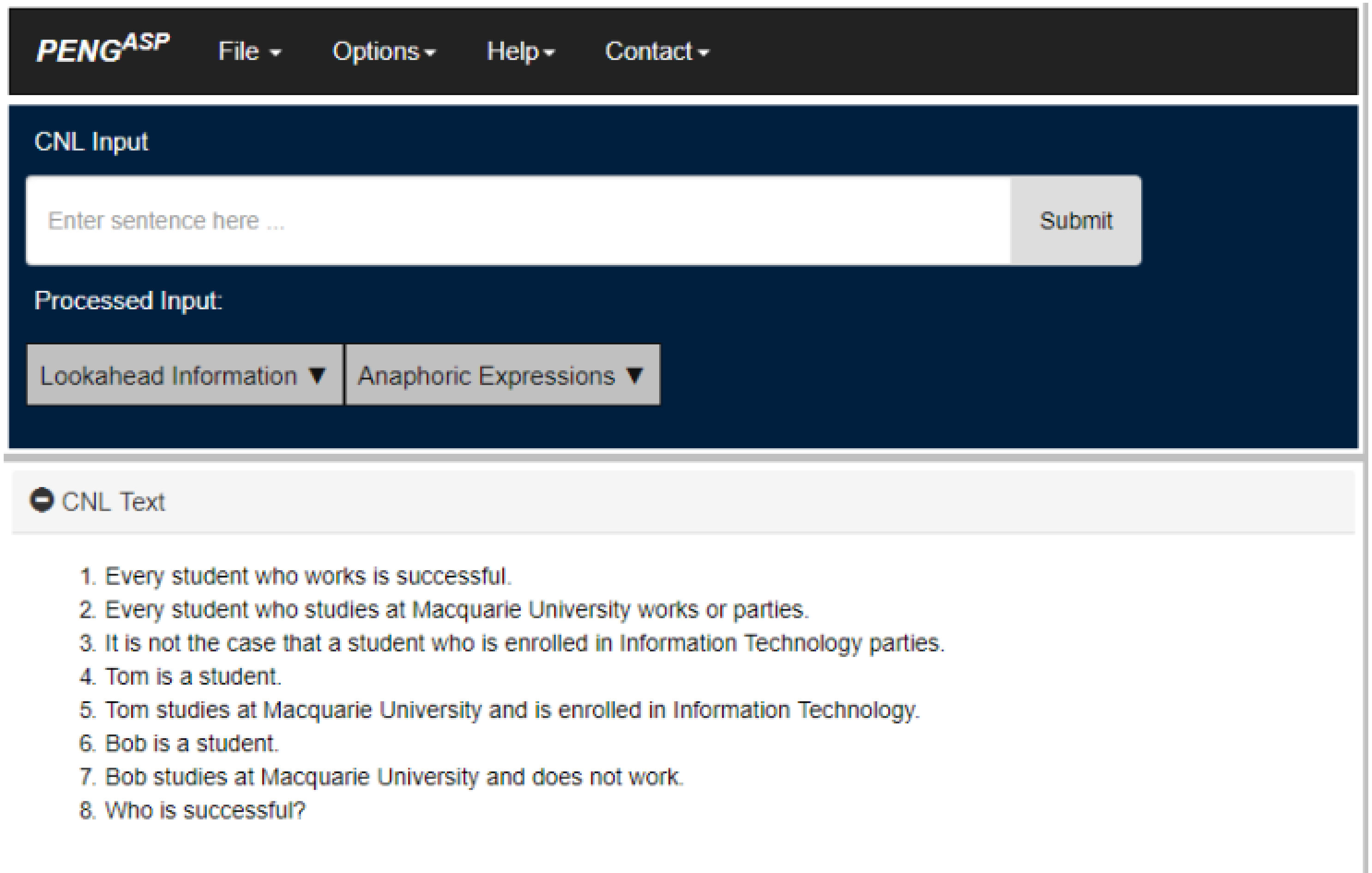
PENG^{ASP}: Lookahead Information



PENG^{ASP}: Anaphoric Expressions



PENG^{ASP}: Entire Specification



© Macquarie University 2019

Answer Set Program

Answer Set Program   Background Theory: OFF

```
prop(A,successful) :- class(A,student), pred(A,work).

pred(B,work) ; pred(B,party) :- class(B,student), pred(B,C,study_at), named(C,macquarie_university).

:- class(D,student), prop(D,E,enrolled_in), named(E,information_technology), pred(D,party).

named(1,tom).

class(1,student).

pred(1,2,study_at).

named(2,macquarie_university).

prop(1,3,enrolled_in).

named(3,information_technology).

named(4,bob).

class(4,student).

pred(4,2,study_at).

-pred(4,work).

answer(named(F,G)) :- named(F,G), prop(F,successful).
```

© Macquarie University 2019

Answer Set

Answer Sets

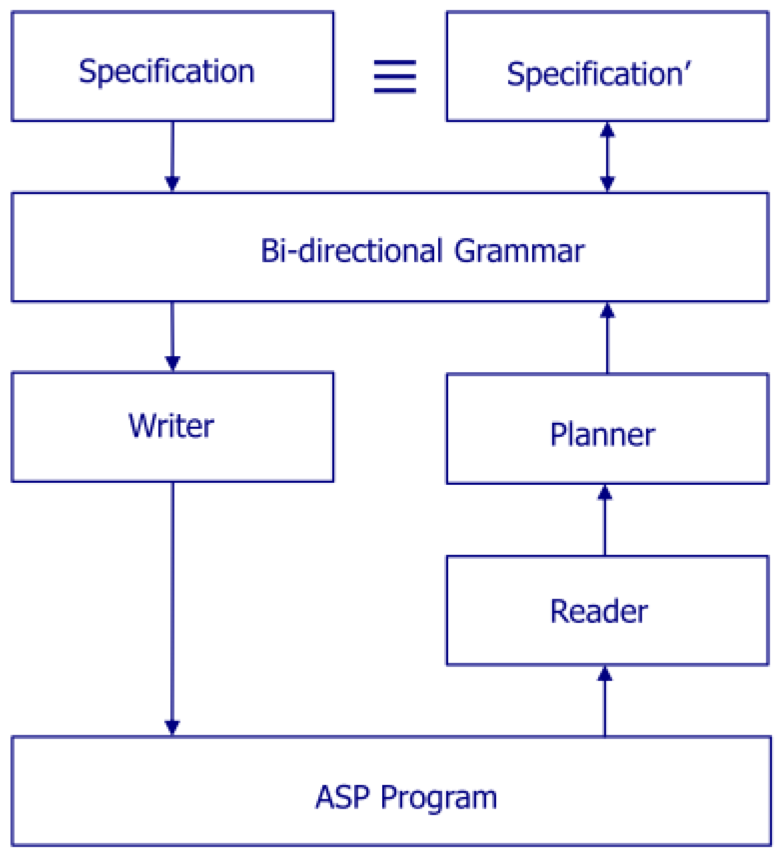
```
clingo version 5.3.0
Reading from asp.lp
Solving...
Answer: 1
-pred(4,work) class(1,student) class(4,student) named(1,tom) named(2,macquarie_university) named(3,infor
mation_technology) named(4,bob) pred(1,2,study_at) pred(4,2,study_at) prop(1,3,enrolled_in) pred(1,work)
pred(4,party) prop(1,successful) answer(named(1,tom))
SATISFIABLE

Models       : 1
Calls        : 1
Time         : 0.001s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time     : 0.000s
```

Answer

Answers

```
Tom
```

PENG^{ASP}: Round-Tripping



Test: Round-Tripping

```
round_tripping(FileName) :-
   peng_file(FileName, Specification1),
   tokeniser(Specification1),
   process_sentences(Specification1, Clauses1),
   writer(Clauses1),
   execute(Solution1),
   reader(Clauses2),
   generate_sentences(Clauses2, Specification2),
   process_sentences(Specification2, Clauses3),
   writer(Clauses3),
   execute(Solution2),
   Clauses1 == Clauses2,
   Solution1 == Solution2,
   write('Round tripping is successful.').
```

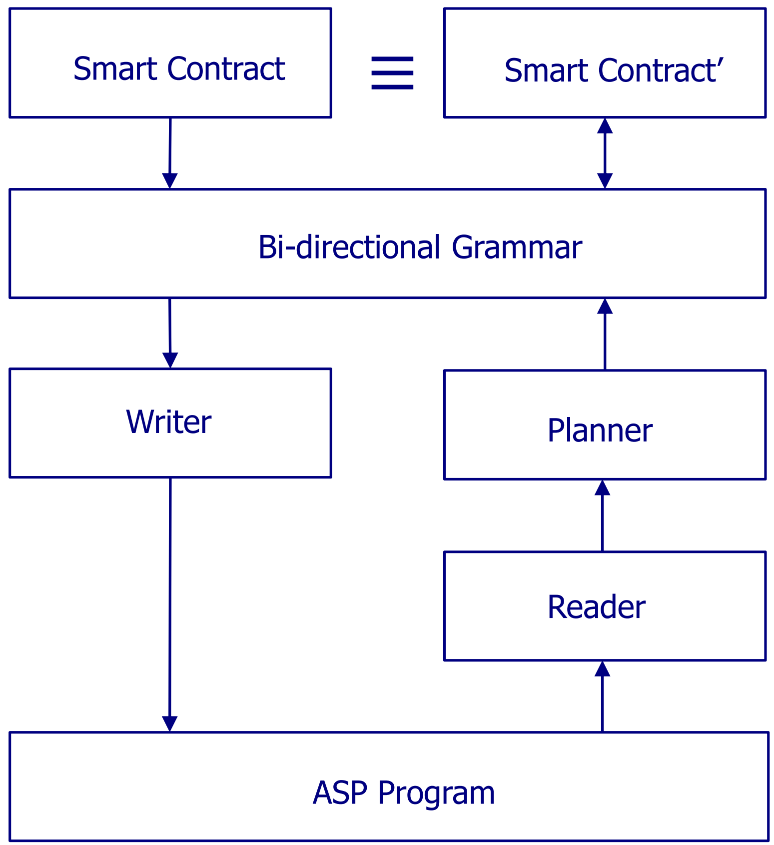
© Macquarie University 2019

Answer Set Program

```
named(1, rolf_schwitter).
class(1, seller).
named(2, tracy_yap_realty).
class(2, listing_brokerage).
pred(1, 2, appoint).
...
pred(E, F, deal_with) :-
  class(E, brokerage),
  pred(E, G, represent),
  class(G, seller),
  prop(G, H, included_in),
  class(H, agreement),
  pred(E, F, receive),
  class(F, notice),
  pred(F, H, belong_to).
```

© Macquarie University 2019

PENG^{ASP}: Round-Tripping



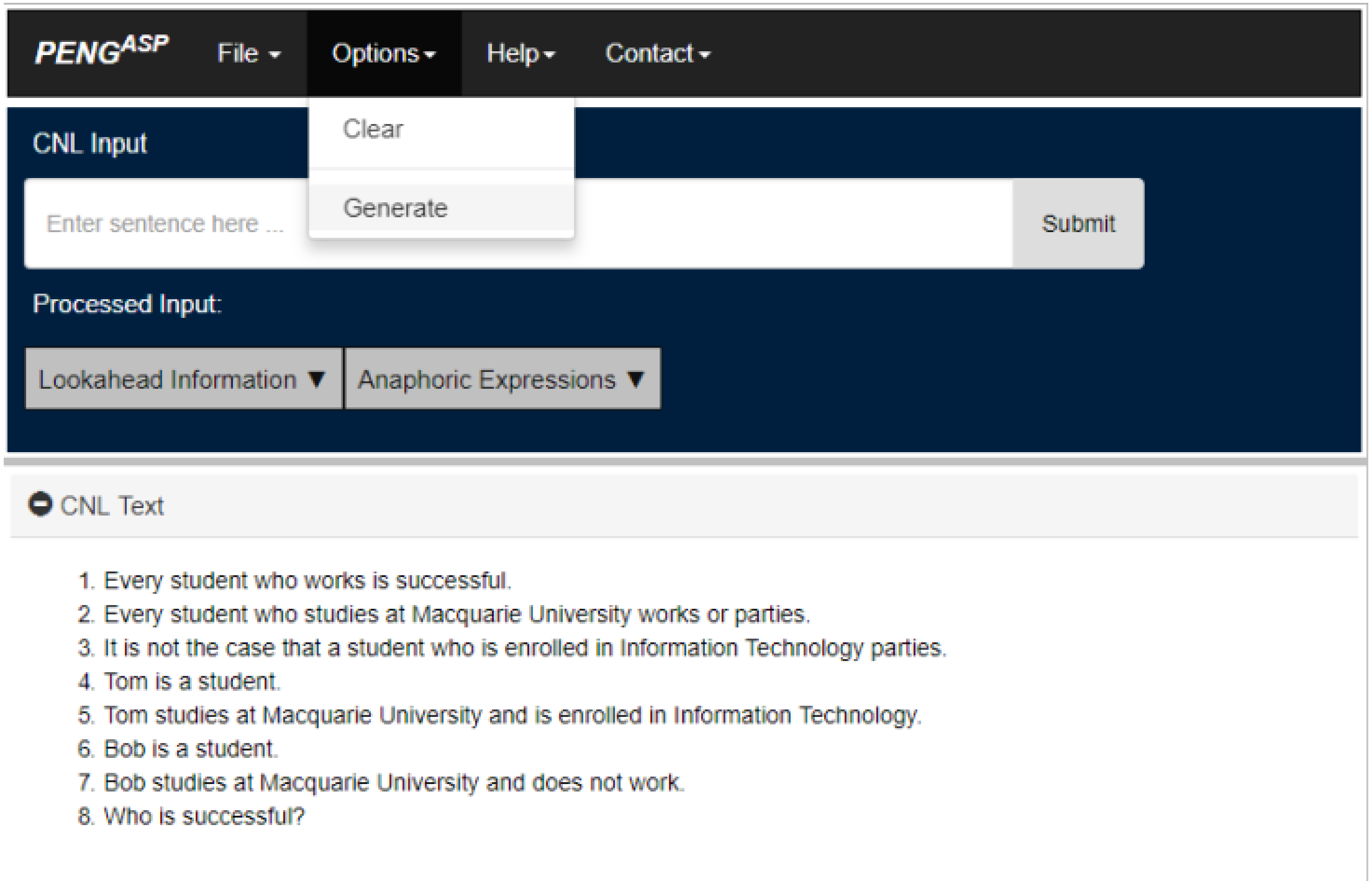
© Macquarie University 2019

Test: Round-Tripping

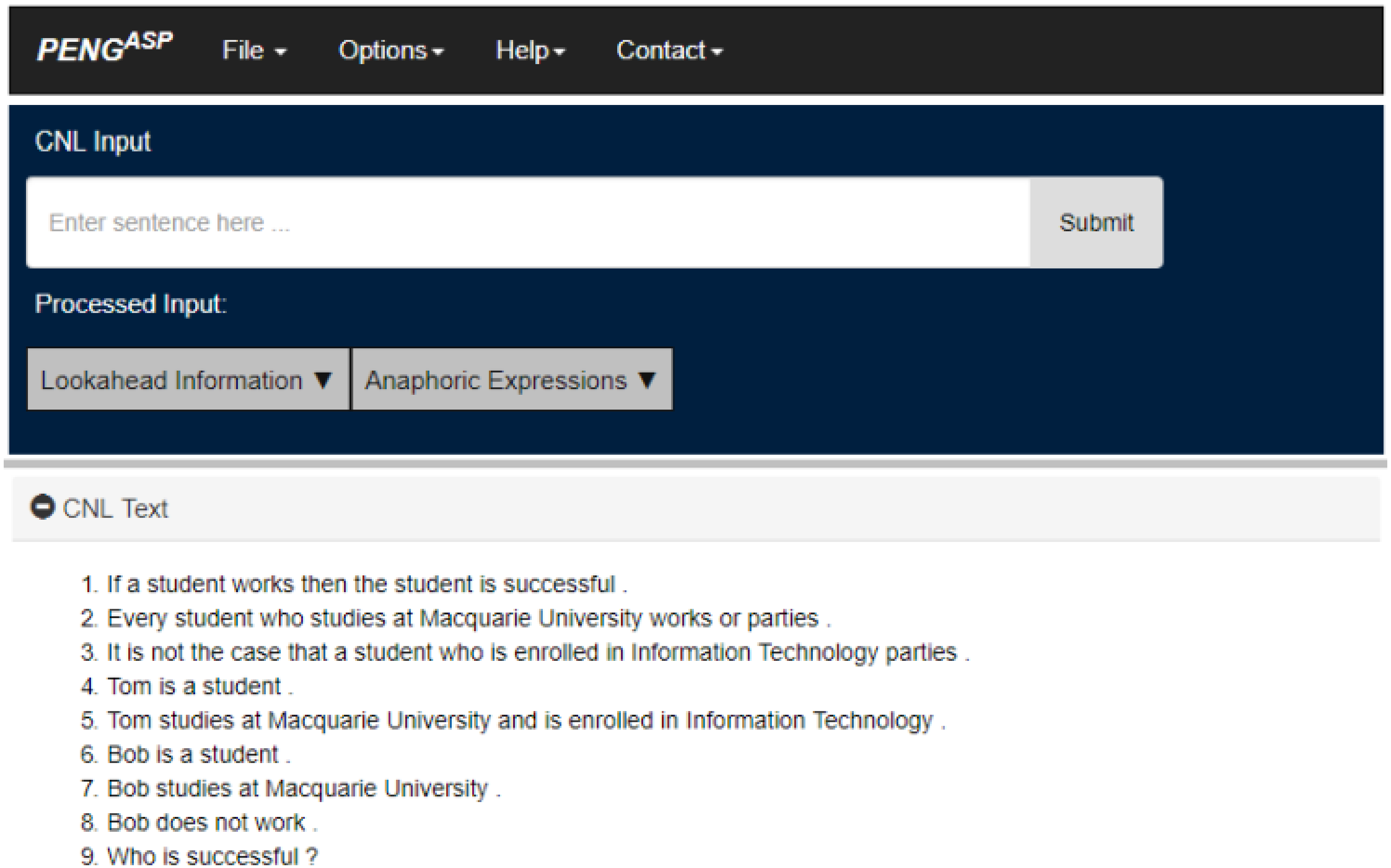
```
round_tripping(FileName) :-
   peng_file(FileName, Contract1),
   tokeniser(Contract1),
   process_sentences(Contract1, Clauses1),
   writer(Clauses1),
   execute(Solution1),
   reader(Clauses2),
   generate_sentences(Clauses2, Contract2),
   process_sentences(Contract2, Clauses3),
   writer(Clauses3),
   execute(Solution2),
   Clauses1 == Clauses2,
   Solution1 == Solution2,
   write('Round tripping is successful.').
```

© Macquarie University 2019

PENG^{ASP}: Round-Tripping



PENG^{ASP}: Result Round-Tripping



© Macquarie University 2019

Strong and Weak Constraints

- ASP supports strong constraints and weak constraints.
- Strong constraints weed out answer sets.
- Weak constraints should be satisfied if possible, they rank answer sets.
- Strong constraint:

```
:- Literal.
```

- Weak constraint:

```
:~ Literal. [Weight@Level, t]
```

© Macquarie University 2019

Weak Constraints

- Weak constraints can be weighted according to their importance.
- In the presence of weights, best models minimize the sum of the weights of the violated weak constraints.
- Weak constraints can also be prioritized.
- Under prioritization, the semantics minimizes the violation of the constraints with respect to the priority levels in descending order.

© Macquarie University 2019

Weak Constraints

- The weak constraint:

```
:~ pred(I, N, W). [W@L, I]
```

can be expressed alternatively as a minimize statement:

```
#minimize { W@L, I : pred(I, N, W) }
```

- Minimize statements can be interpreted as maximize statements with inverse weights:

```
#maximize { -W@L, I : pred(I, N, W) }.
```

Scenario in Full Natural Language

We want to choose one among five available accommodations. These accommodations are identified via their names (Amora, Grace, Metro, Posh and Adina), and each accommodation owns a certain number of stars and costs a certain amount of money. Of course, the more stars an accommodation has, the more it costs per night. Additionally, we know that the motel Posh is located on a main street, which is why we expect its rooms to be noisy. Avoiding noise is very important to us, minimizing the cost per star is less important, and maximizing the star rating is least important to us.

© Macquarie University 2019

Reconstructed Scenario in PENG^{ASP}

1. Choose either the accommodation Amora or Grace or Metro or Posh or Adina.
2. The hotel Amora owns five stars and costs 170 dollars.
3. The hotel Grace owns four stars and costs 140 dollars.
4. The hotel Metro owns three stars and costs 90 dollars.
5. The guesthouse Adina owns two stars and costs 60 dollars.
6. The motel Posh that is located on the main street owns three stars and costs 75 dollars.

© Macquarie University 2019

Reconstructed Scenario in PENG^{ASP}

7. If an accommodation is located on a main street then the accommodation is noisy.
8. If an accommodation costs N dollars and owns M stars then N / M is the cost per star of the accommodation.
9. If an accommodation owns N stars then N is the star rating of the accommodation.

Reconstructed Scenario in PENG^{ASP}

10. Minimizing that an accommodation is noisy has the high priority H.
11. Minimizing the cost per star of an accommodation has the medium priority M.
12. Maximizing the star rating of an accommodation has the low priority L.
13. The high priority is 3.
14. The medium priority is 2.
15. The low priority is 1.

© Macquarie University 2019

Sentence 1

Choose either the accommodation Amora or Grace or Metro or Posh or Adina.

```
1 { class(X, accommodation) :
    named(X, (amora ; grace ; metro ; posh ; adina)) } 1.
```

Sentences 2-5: 4

The hotel Metro owns three stars and costs 90 dollars.

```
class(7, hotel).
named(7, metro).
pred(7, 8, own).
data_prop(8, pos_int(3), cardinal).
class(8, star).
pred(7, 9, cost).
data_prop(9, pos_int(90), cardinal).
class(9, dollar).
```

© Macquarie University 2019

Sentence 6

The motel Posh that is located on the main street owns three stars and costs 75 dollars.

```
class(13, motel).
named(13, posh).
prop(13, 14, located_on).
class(14, main_street).
pred(13, 8, own).
pred(13, 15, cost).
data_prop(15, pos_int(75), cardinal).
class(15, dollar).
```

© Macquarie University 2019

Sentence 6: Relative Clause

The motel Posh that is located on the main street owns three stars and costs 75 dollars.

```
class(13, motel).
named(13, posh).
prop(13, 14, located_on).
class(14, main_street).
pred(13, 8, own).
pred(13, 15, cost).
data_prop(15, pos_int(75), cardinal).
class(15, dollar).
```

© Macquarie University 2019

Sentence 6: Anaphoric Reference

The motel Posh that is located on the main street owns three stars and costs 75 dollars.

```
class(13, motel).
named(13, posh).
prop(13, 14, located_on).
class(14, main_street).
pred(13, 8, own).
pred(13, 15, cost).
data_prop(15, pos_int(75), cardinal).
class(15, dollar).
```

© Macquarie University 2019

Sentence 7

If an accommodation is located on a main street then the hotel is noisy.

```
prop(X, noisy) :-
  class(X, accommodation),
  prop(X, Y, located_on),
  class(Y, main_street).
```

Sentence 8

If an accommodation costs N dollars and owns M stars then N / M is the cost per star of the accommodation.

```
rel(N/M, X, cost_per_star) :-
    class(X, accommodation),
    pred(X, Y, cost),
    data_prop(Y, pos_int(N), cardinal),
    class(Y, dollar),
    pred(X, Z, own),
    data_prop(Z, pos_int(M), cardinal),
    class(Z, star).
```

Sentence 9

If an accommodation owns N stars then N is the star rating of the accommodation.

```
rel(N, X, star_rating) :-
  class(X, accommodation),
  pred(X, Y, own),
  data_prop(Y, pos_int(N), cardinal),
  class(Y, star).
```

© Macquarie University 2019

Sentence 10

Minimizing that an accommodation is noisy has the high priority H.

```
#minimize { 1@H,
            X : class(X, accommodation),
                prop(X, noisy),
                prop(P, high),
                class(P, priority),
                data_prop(P, pos_int(H), nominal) }.
```

Sentence 11

Minimizing the cost per star of an accommodation has the medium priority M.

```
#minimize { X@M,
            Y : rel(X, Y, cost_per_star),
                class(Y, accommodation),
                prop(P, medium),
                class(P, priority),
                data_prop(P, pos_int(M), nominal) }.
```

© Macquarie University 2019

Sentence 12

Maximizing the star rating of an accommodation has the low priority L.

```
#maximize { X@L,
            Y : rel(X, Y, star_rating),
                class(Y, accommodation),
                prop(P, low),
                class(P, priority),
                data_prop(P, pos_int(L), nominal) }.
```

Sentences 13-15: 13

The high priority is 3.

```
prop(16, high).
class(16, priority).
data_prop(16, pos_int(3), nominal).
```

Evaluation

```
:~ prop(13, noisy), class(13, accommodation). [1@3, 13]

:~ class(1,  accommodation). [34@2,  1]
:~ class(4,  accommodation). [35@2,  4]
:~ class(7,  accommodation). [30@2,  7]
:~ class(10, accommodation). [30@2, 10]
:~ class(13, accommodation). [25@2, 13]

:~ class(1,  accommodation). [-5@1,  1]
:~ class(4,  accommodation). [-4@1,  4]
:~ class(7,  accommodation). [-3@1,  7]
:~ class(10, accommodation). [-2@1, 10]
:~ class(13, accommodation). [-3@1, 13]
```

© Macquarie University 2019

Evaluation

```
:~ prop(13, noisy), class(13, accommodation). [1@3, 13]

:~ class(1,  accommodation). [34@2,  1]
:~ class(4,  accommodation). [35@2,  4]
:~ class(7,  accommodation). [30@2,  7]
:~ class(10, accommodation). [30@2, 10]
:~ class(13, accommodation). [25@2, 13]

:~ class(1,  accommodation). [-5@1,  1]
:~ class(4,  accommodation). [-4@1,  4]
:~ class(7,  accommodation). [-3@1,  7]
:~ class(10, accommodation). [-2@1, 10]
:~ class(13, accommodation). [-3@1, 13]
```

© Macquarie University 2019

Evaluation

```
:~ prop(13, noisy), class(13, accommodation). [1@3, 13]

:~ class(1,  accommodation). [34@2,  1]
:~ class(4,  accommodation). [35@2,  4]
:~ class(7,  accommodation). [30@2,  7]
:~ class(10, accommodation). [30@2, 10]
:~ class(13, accommodation). [25@2, 13]

:~ class(1,  accommodation). [-5@1,  1]
:~ class(4,  accommodation). [-4@1,  4]
:~ class(7,  accommodation). [-3@1,  7]
:~ class(10, accommodation). [-2@1, 10]
:~ class(13, accommodation). [-3@1, 13]
```

© Macquarie University 2019

Evaluation

```
:~ prop(13, noisy), class(13, accommodation). [1@3, 13]

:~ class(1,  accommodation). [34@2,  1]
:~ class(4,  accommodation). [35@2,  4]
:~ class(7,  accommodation). [30@2,  7]
:~ class(10, accommodation). [30@2, 10]
:~ class(13, accommodation). [25@2, 13]

:~ class(1,  accommodation). [-5@1,  1]
:~ class(4,  accommodation). [-4@1,  4]
:~ class(7,  accommodation). [-3@1,  7]
:~ class(10, accommodation). [-2@1, 10]
:~ class(13, accommodation). [-3@1, 13]
```

© Macquarie University 2019

Solution

```
% named(7, metro) ... class(7, hotel) ...
% class(7, accommodation)
% Optimization: 0 30 -3
% OPTIMUM FOUND
% Time : 0.004s
```

© Macquarie University 2019

Working with Temporal Information

Specification:

1. The train AV8504 is located at Roma Termini on 2019-02-20 at 06:30.
2. The train departures from Roma Termini at 06:45.
3. The train arrives at Firenze Campo di Marte at 08:03 and departures from there at 08:10
4. The train arrives at Bozen/Bolzano at 11:17.

© Macquarie University 2019

ASP Representation

```
class(1,train).
named(1,av8504).
holds_at(fluent(1,2,located_at),1550644200).
named(2,roma_termini).
data_prop(3,2019,2,20,date).
data_prop(3,6,30,time).
data_prop(3,1550644200,date_time).
happens(event(1,2,departure_from),1550645100).
data_prop(5,2019,2,20,date).
data_prop(5,6,45,time).
data_prop(5,1550645100,date_time).
```

© Macquarie University 2019

Working with Temporal Information

- Event Calculus:
  - events,
  - fluents (= time-varying properties)
  - time points.
- Event: ***happens(Event, TimePoint)***
- Fluent: ***holds_at(Fluent, TimePoint)***
- Effect axioms:
  - ***initiates(Event, Fluent, TimePoint)***
  - ***terminates(Event, Fluent, TimePoint***

Working with Temporal Information

Effect Axioms:

1. If a vehicle departures from a location at a time point T then the vehicle will be in transit after the time point T.
2. If a vehicle departures from a location at a time point T then the vehicle will no longer be located at that location after the time point T.

© Macquarie University 2019

ASP Representation

```
initiates(event(A, B, departure_from), fluent(A, in_transit), T) :-
   class(A, vehicle),
   happens(event(A, B, departure_from), T),
   class(B, location),
   class(C, time_point),
   data_prop(C, T, date_time).

terminates(event(A, B, departure_from), fluent(A, B, located_at), t) :-
   class(A, vehicle),
   happens(event(A, B, departure_from), T),
   class(B, location),
   class(C, time_point),
   data_prop(C, T, date_time).
```

© Macquarie University 2019

Working with Temporal Information

Effect Axioms:

3. If a vehicle arrives at a location at a time point T then the vehicle will be located at that location after the time point T.
4. If a vehicle arrives at a location at a time point T and the vehicle does not provably make a stopover at the time point T then the vehicle will no longer be in transit after the time point T.

© Macquarie University 2019

ASP Representation

```
terminates(event(A, B, departure_from), fluent(A, B, located_at), T) :-
   class(A, vehicle),
   happens(event(A, B, departure_from), T),
   class(B, location),
   class(C, time_point),
   data_prop(C, T, date_time).

initiates(event(A, B, arrive_at), fluent(A, B, located_at), T) :-
   class(A, vehicle),
   happens(event(A, B, arrive_at), T),
   class(B, location),
   class(C, time_point),
   data_prop(C, T, date_time).
```

© Macquarie University 2019

Working with Temporal Information

Ontological commitments:

1. Roma Termini is a railway station.
2. Firenze Campo di Marte is a railway station.
3. Bozen/Bolzano is a railway station.
4. Every railway station is a location
5. Every train is a vehicle.
6. If a vehicle arrives at a location at a time point T1 and departures from that location at a time point T2 and T1 is before T2 then the vehicle makes a stopover at that location between T1 and T2.

© Macquarie University 2019

ASP Representation

```
class(2, railway_station).
class(7, railway_station).
class(12, railway_station).
class(A, location) :- class(A, railway_station).
class(A, vehicle) :- class(A, train).
```

© Macquarie University 2019

ASP Representation

```
happens(event(A, make_stopover), T1, T2) :-
    class(A, vehicle),
    happens(event(A, B, arrive_at), T1),
    class(B, location),
    class(C, time_point),
    data_prop(C, T1, date_time),
    happens(event(A, B, departure_from), T2),
    class(D, time_point),
    data_prop(D, T2, date_time),
    T1 < T2.
```

© Macquarie University 2019

Event Calculus Axioms in ASP (Excerpt)

```
holds_at(F, T2) :-
   initiates(E, F, T1),
   time_point(T2),
   T1 < T2,
   not clipped(T1, F, T2).

clipped(T1, F, T2) :-
   time_point(T1),
   time_point(T2),
   T1 <= T, T < T2,
   terminates(E, F, T).
```

© Macquarie University 2019

Evaluation

```
holds_at(fluent(1, 2, located_at), 630)
holds_at(fluent(1, 2, located_at), 640)
holds_at(fluent(1, in_transit), 730)
holds_at(fluent(1, in_transit), 803)
holds_at(fluent(1, 3,located_at), 805)
holds_at(fluent(1, in_transit), 805)
holds_at(fluent(1, 3, located_at), 810)
holds_at(fluent(1, in_transit), 810)
holds_at(fluent(1, in_transit), 1115)
holds_at(fluent(1, in_transit), 1117)
holds_at(fluent(1, 4, located_at), 1118)

-holds_at(fluent(1, 2, located_at), 730)
...
-holds_at(fluent(1, in_transit), 1118)
```

Questions

1. When is the train located at a railway station?
2. Is the train in transit at 08:05?
3. How far away is the train from Roma Termini at 08:30?

Note: 3 requires a trajectory axiom of the Event Calculus (not show on the previous slides)

Take-Home Messages

- PENG^{ASP} is human-understandable and machine-processable controlled natural language.
- PENG^{ASP} can serve as a high-level (bi-directional) interface language to semantic systems.
- PENG^{ASP} can bridge the gap between English and formal languages.
- Future research:
  - PENG^{ASP} for smart contracts
  - PENG^{ASP} for dynamic domains
  - PENG^{ASP} for conceptual modelling.