Imperial College
London

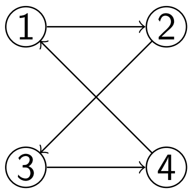
Part III
Unit 7: The Answer Set Semantics

Mark Law

March 8th, 2018

Imperial College
London

Reachability in graphs



```
1 :  node(1..4).
2 :  edge(1,2). edge(2,3).
3 :  edge(3,4). edge(4,1).
```

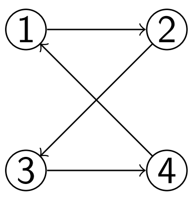
```
4 :  reach(N) :- edge(1,N).
5 :  reach(N) :- reach(N2),edge(N2,N).

6 :  unreachable_node :- node(N),not reach(N).
```

1. Consider the graph above. We wish to test whether every node is "reachable" from node 1 (i.e. whether there is a path from 1 to every node).
2. Line 1 says that there are 4 nodes in the graph (`node(1..4)` is shorthand for `node(1), . . . , node(4)`).
   Lines 2 and 3 specify the edges in the graph.
3. Lines 4 and 5 specify the concept of a node being reachable from node 1. Line 4 is the base case that says that a node N is reachable from 1 if there is an edge from 1 to N.
   Line 5 is the recursive case, saying that a node N is reachable if there is an edge from another reachable node to N.
4. Line 6 says that there is an unreachable node if there exists a node for which we cannot prove reachability.

Imperial College London

Reachability in graphs: Clark Completion



$$\begin{array}{rl} 1: & \texttt{node}(\texttt{X}) \leftrightarrow \texttt{X} = 1 \vee \ldots \vee \texttt{X} = 4. \\ 2: & \texttt{edge}(\texttt{X},\texttt{Y}) \leftrightarrow (\texttt{X} = 1 \wedge \texttt{Y} = 2) \vee \\ & \qquad (\texttt{X} = 2 \wedge \texttt{Y} = 3) \vee \\ & \qquad (\texttt{X} = 3 \wedge \texttt{Y} = 4) \vee \\ & \qquad (\texttt{X} = 4 \wedge \texttt{Y} = 1) \end{array}$$

$$\begin{array}{rl} 3: & \texttt{reach}(\texttt{N}) \leftrightarrow \texttt{edge}(1,\texttt{N}) \vee \exists \texttt{N2}.(\texttt{reach}(\texttt{N2}) \wedge \texttt{edge}(\texttt{N2},\texttt{N})). \\ 4: & \texttt{unreachable\_node} \leftrightarrow \exists \texttt{N}.(\texttt{node}(\texttt{N}) \wedge \texttt{not reach}(\texttt{N})). \\ 5: & \neg(1 = 2).\ \ldots \end{array}$$

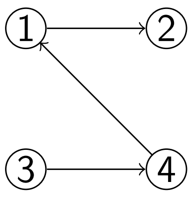
What are the models of *comp*(*P*)?

$$Facts(P) \cup \{\texttt{reach}(1..4)\}$$

1. Earlier in the course, you have seen the idea of the Clark Completion (Clark 1978).
2. Our graph program has exactly one supported model (shown on the slide). Note that *Facts*(*P*) denotes the set of atoms that occur as facts in *P*. This shows that in this graph, every node is reachable from node 1.

Imperial College London

Reachability in graphs: Clark Completion



$$\begin{aligned} 1: &\quad \texttt{node(X)} \leftrightarrow \texttt{X} = 1 \vee \ldots \vee \texttt{X} = 4.\\ 2: &\quad \texttt{edge(X,Y)} \leftrightarrow (\texttt{X} = 1 \wedge \texttt{Y} = 2)\vee\\ &\qquad (\texttt{X} = 3 \wedge \texttt{Y} = 4)\vee\\ &\qquad (\texttt{X} = 4 \wedge \texttt{Y} = 1)\end{aligned}$$

$$\begin{aligned} 3: &\quad \texttt{reach(N)} \leftrightarrow \texttt{edge(1,N)} \vee \exists \texttt{N2}.(\texttt{reach(N2)} \wedge \texttt{edge(N2,N)}).\\ 4: &\quad \texttt{unreachable\_node} \leftrightarrow \exists \texttt{N}.(\texttt{node(N)} \wedge \texttt{not reach(N)}).\\ 5: &\quad \neg(1 = 2).\ \ldots\end{aligned}$$

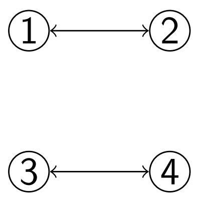
What are the models of *comp*(*P*)?

$$\mathit{Facts}(P) \cup \{\texttt{reach(1)}, \texttt{reach(2)}, \texttt{unreachable\_node}\}$$

1. We can check the program by changing the graph to one which has an unreachable node.
2. This time, the unique model of *comp*(*P*) contains `unreachable_node` as the nodes 3 and 4 are unreachable from 1.

Imperial College London

Reachability in graphs: Clark Completion



$$\begin{array}{ll} 1: & \texttt{node}(\texttt{X}) \leftrightarrow \texttt{X} = 1 \vee \ldots \vee \texttt{X} = 4. \\ 2: & \texttt{edge}(\texttt{X},\texttt{Y}) \leftrightarrow (\texttt{X} = 1 \wedge \texttt{Y} = 2) \vee \\ & \quad (\texttt{X} = 2 \wedge \texttt{Y} = 1) \vee \\ & \quad (\texttt{X} = 3 \wedge \texttt{Y} = 4) \vee \\ & \quad (\texttt{X} = 4 \wedge \texttt{Y} = 3) \end{array}$$

$$\begin{array}{ll} 3: & \texttt{reach}(\texttt{N}) \leftrightarrow \texttt{edge}(1,\texttt{N}) \vee \exists \texttt{N2}.(\texttt{reach}(\texttt{N2}) \wedge \texttt{edge}(\texttt{N2},\texttt{N})). \\ 4: & \texttt{unreachable\_node} \leftrightarrow \exists \texttt{N}.(\texttt{node}(\texttt{N}) \wedge \texttt{not reach}(\texttt{N})). \\ 5: & \neg(1 = 2). \ \ldots \end{array}$$

What are the models of *comp*(*P*)?

$$\mathit{Facts}(P) \cup \{\texttt{reach}(1..2), \texttt{unreachable\_node}\},$$
$$\mathit{Facts}(P) \cup \{\texttt{reach}(1..4)\}$$

1. In this graph, not all nodes are reachable, but there are cycles.
2. This time, there are two models of *comp*(*P*). One says that the graph has unreachable nodes, and the other says that it does not!
3. The problem is that the Clark completion does not specify that the base case must be reached.
4. In the remainder of this course, we will see the Answer Set semantics, which gives a much more intuitive meaning to recursive programs with negation as failure. We will see that our graph program has exactly one answer set (the first model on the slide).

Imperial College London

Part III: Overview

- Unit 7: The Answer Set Semantics
  - Motivation ✓
  - Semantics
  - How does it relate to the Clark completion?
- Unit 8: Disjunction, Constraints, Choice rules and Abduction in ASP
- Unit 9: Aggregation and Optimisation in ASP

In this part of the course, we will study the answer set semantics. In this lecture, we will formalise what it means to be an answer set of a program, and see how this relates to the other semantics you have covered in this course so far. The next lecture will introduce ways of expressing choice and disjunction in ASP. We will see that these constructs are very useful for abduction in ASP. In the final lecture, we will look at how to express aggregation and optimisation in ASP.

Imperial College London

ASP: Overview

- In this lecture, programs are set of rules of the form:

  $$\texttt{h :- b}_1\texttt{,...,b}_\texttt{m}\texttt{,not c}_1\texttt{,...,not c}_\texttt{n}\texttt{.}$$

  Where $\texttt{h}, \texttt{b}_1, \ldots, \texttt{b}_\texttt{m}, \texttt{c}_1, \ldots, \texttt{c}_\texttt{n}$ are all atoms (made of predicates, functions, constants and variables as usual). Note, ASP does not have lists!

- ASP allows other kinds of rules, which we will see next week.
- ASP is declarative – you specify the problem, not the procedure!

The ASP programs we will see in this lecture are similar to the programs you have worked with so far in the course, other than that ASP does not allow lists. We will see next week that ASP allows for other kinds of rules to express choice, optimisation and other concepts. One of the major selling points of ASP is that it is declarative. Unlike in Prolog, where you need to think about structuring your program to avoid loops, ASP programs are declarative. In ASP, you encode the problem you are trying to solve but you do not need to think about the procedure that is required to solve it (this is delegated to an ASP solver).

Imperial College London

Terminology

- A fact is a rule with an empty body.
- $head(R)$ is the head of a rule.
- $body^{+}(R)$ is the set of all positive body literals in a rule.
- $body^{-}(R)$ is the set of all negative body literals in a rule.

Imperial College London

Terminology

- The Herband base of a program is the set of all ground atoms that can be made from predicate symbols, functions and constants in the program.
- A Hebrand interpretation is an assignment from each element of the Hebrand base of a program to either true or false. We usually write an interpretation as the set of things it assigns to true.
- A Herbrand model is a Herbrand interpretation that satisfies every rule in the program (if it satisfies the body of a rule, it must satisfy the head).
- A Herbrand model is said to be minimal if no subset of it is also a model.

Imperial College
London

Relevant Grounding: Iterative Construction

- For any program P, $ground(P)$ denotes the set of all ground instances of rules in P.
- As this is usually infinite, ASP solvers compute the *relevant grounding*, denoted $\mathcal{RG}(P)$ in this course.

```
1: procedure $\mathcal{RG}(P)$
2:     $G = \emptyset$
3:     do
4:         $G' = \left\{ R \in ground(P) \;\middle|\; \begin{array}{c} R \notin G \text{ and each element of} \\ body^{+}(R) \text{ is the head of a rule in } G \end{array} \right\}$
5:         $G = G \cup G'$
6:     while $G' \neq \emptyset$
7:     return $G$
8: end procedure
```

The first step of solving an Answer Set Program is to transform it into an equivalent ground program. This process is called *grounding*. Any program that contains at least one function symbol and at least one constant (and at least one variable) technically has an infinite number of ground instances. For example, consider the program:

```
p(f(X)) :- q(g(X)).
q(a).
```

You may think that there are only two important ground instances to consider here (the fact `q(a)` and the rule with `X` substituted with `a`). In fact `X` can be substituted with `f(a)`, `f(f(a))` and so on. As `q(g(f(a)))` (etc.) can never be proved given this program, these rules are "irrelevant". ASP solvers consider a smaller *relevant* grounding. This can be computed using the algorithm on the slide (although modern ASP grounders such as Gringo (Gebser et al 2007), use much more sophisticated algorithms).

The idea of this simple procedure for computing the relevant grounding is to keep adding ground instances of rules R for which every positive body literal of R occurs in the head of a ground instance that we have already computed. Any ground rule that contains a positive body literal that is not in the head of some other rule is irrelevant as its body cannot be satisfied by the other rules (as its body is guaranteed to not be satisfied, the rule is guaranteed to be satisfied).

Imperial College London

Grounding: safety

- A rule R is safe if every variable that occurs in R occurs in $body^{+}(R)$

Are the following rules safe?

```
p(X) :- q(X), not r(X, Y).   ✗
p(Y) :- q(X), not r(X, X).   ✗
p(Y) :- q(X), r(X, Y).       ✓
```

1. In order for grounding to be possible, a simple condition must hold: *safety*. A rule R is said to be *unsafe* if it contains a variable that does not occur in any positive body literal of R. For an ASP program to be grounded it must be safe (i.e. it cannot contain any unsafe rules).

Imperial College
London

Relevant Grounding: Example

$$P = \begin{cases} 1: & \texttt{node(1..2). edge(1,1). edge(2,2).} \\ 2: & \texttt{reach(N) :- edge(1,N).} \\ 3: & \texttt{reach(N) :- reach(N2), edge(N2,N).} \\ 4: & \texttt{unreachable\_node :- node(N), not reach(N).} \end{cases}$$

What is $\mathcal{RG}(P)$?

Imperial College
London

Relevant Grounding: Example

$$P = \left\{ \begin{array}{ll} 1: & \texttt{node(1..2). edge(1,1). edge(2,2).} \\ 2: & \texttt{reach(N) :- edge(1,N).} \\ 3: & \texttt{reach(N) :- reach(N2), edge(N2,N).} \\ 4: & \texttt{unreachable\_node :- node(N), not reach(N).} \end{array} \right.$$

What is $\mathcal{RG}(P)$?

```
node(1..2). edge(1,1). edge(2,2).
```

Imperial College
London

Relevant Grounding: Example

$$P = \begin{cases} 1: & \texttt{node(1..2). edge(1,1). edge(2,2).} \\ 2: & \texttt{reach(N) :- edge(1,N).} \\ 3: & \texttt{reach(N) :- reach(N2), edge(N2,N).} \\ 4: & \texttt{unreachable\_node :- node(N), not reach(N).} \end{cases}$$

What is $\mathcal{RG}(P)$?

```
node(1..2). edge(1,1). edge(2,2).
reach(1) :- edge(1,1).

unreachable_node :- node(1), not reach(1).
unreachable_node :- node(2), not reach(2).
```

Imperial College
London

Relevant Grounding: Example

$$P = \begin{cases} 1: & \texttt{node(1..2). edge(1,1). edge(2,2).} \\ 2: & \texttt{reach(N) :- edge(1,N).} \\ 3: & \texttt{reach(N) :- reach(N2), edge(N2,N).} \\ 4: & \texttt{unreachable\_node :- node(N), not reach(N).} \end{cases}$$

What is $\mathcal{RG}(P)$?

```
node(1..2). edge(1,1). edge(2,2).
reach(1) :- edge(1,1).
reach(1) :- reach(1), edge(1,1).
unreachable_node :- node(1), not reach(1).
unreachable_node :- node(2), not reach(2).
```

Imperial College
London

Relevant Grounding: Exercise

Please attempt Question 1 (a) (i) from the tutorial sheet.

Imperial College London

Reduct

Given a ground logic program P and an interpretation X, the reduct P^X is constructed in two steps:

- Remove any rule whose body contains the negation of an atom in X
- Delete any negation from the remaining rules

Given an interpretation X and ground logic program P, we can check whether X is an answer set of P using the *reduct* of P with respect to X (denoted P^X). The intuition behind the reduct is that we assume X is "correct" and use it to interpret the negation in the program.

The first step of computing P^X is to remove any rule from P whose body contains the negation of something in X – if we assume that X is "correct" these rule bodies are false.

The second step is to delete the negation in the remaining rules – if we assume X is correct, then "`not a`" is true for any a $\notin X$.

There are many different definitions of the reduct (and answer sets). For the fragment of ASP that we consider in this lecture (normal logic programs), the different definitions of answer set are all equivalent. We will use other definitions later in the course when we start using other types of rules.

Note that this definition of the reduct is guaranteed to have a unique minimal model (as it is a definite logic program).

Imperial College London

Reduct

Given a ground logic program P and an interpretation X, the reduct P^X is constructed in two steps:

- Remove any rule whose body contains the negation of an atom in X
- Delete any negation from the remaining rules

$X = \{\texttt{heads}\}$, $P = \left\{ \begin{array}{l} \texttt{heads :- not tails.} \\ \texttt{tails :- not heads.} \end{array} \right\}$

Given an interpretation X and ground logic program P, we can check whether X is an answer set of P using the *reduct* of P with respect to X (denoted P^X). The intuition behind the reduct is that we assume X is "correct" and use it to interpret the negation in the program.

The first step of computing P^X is to remove any rule from P whose body contains the negation of something in X – if we assume that X is "correct" these rule bodies are false.

The second step is to delete the negation in the remaining rules – if we assume X is correct, then "`not a`" is true for any $a \notin X$.

There are many different definitions of the reduct (and answer sets). For the fragment of ASP that we consider in this lecture (normal logic programs), the different definitions of answer set are all equivalent. We will use other definitions later in the course when we start using other types of rules.

Note that this definition of the reduct is guaranteed to have a unique minimal model (as it is a definite logic program).

Imperial College
London

Reduct

Given a ground logic program P and an interpretation X, the reduct P^X is constructed in two steps:

- Remove any rule whose body contains the negation of an atom in X
- Delete any negation from the remaining rules

$X = \{\texttt{heads}\}$,

$$P = \left\{ \begin{array}{l} \texttt{heads :- not tails.} \\ \texttt{tails :- not heads.} \end{array} \right\}$$

Given an interpretation X and ground logic program P, we can check whether X is an answer set of P using the *reduct* of P with respect to X (denoted P^X). The intuition behind the reduct is that we assume X is "correct" and use it to interpret the negation in the program.

The first step of computing P^X is to remove any rule from P whose body contains the negation of something in X – if we assume that X is "correct" these rule bodies are false.

The second step is to delete the negation in the remaining rules – if we assume X is correct, then "`not a`" is true for any a $\notin X$.

There are many different definitions of the reduct (and answer sets). For the fragment of ASP that we consider in this lecture (normal logic programs), the different definitions of answer set are all equivalent. We will use other definitions later in the course when we start using other types of rules.

Note that this definition of the reduct is guaranteed to have a unique minimal model (as it is a definite logic program).

Imperial College
London

Reduct

Given a ground logic program P and an interpretation X, the reduct P^X is constructed in two steps:

- Remove any rule whose body contains the negation of an atom in X
- Delete any negation from the remaining rules

$X = \{$`heads`$\}$,

$P = \left\{ \begin{array}{l} \texttt{heads :- not tails.} \\ \texttt{tails :- not heads.} \end{array} \right\}$

Given an interpretation X and ground logic program P, we can check whether X is an answer set of P using the *reduct* of P with respect to X (denoted P^X). The intuition behind the reduct is that we assume X is "correct" and use it to interpret the negation in the program.

The first step of computing P^X is to remove any rule from P whose body contains the negation of something in X – if we assume that X is "correct" these rule bodies are false.

The second step is to delete the negation in the remaining rules – if we assume X is correct, then "`not a`" is true for any a $\notin X$.

There are many different definitions of the reduct (and answer sets). For the fragment of ASP that we consider in this lecture (normal logic programs), the different definitions of answer set are all equivalent. We will use other definitions later in the course when we start using other types of rules.

Note that this definition of the reduct is guaranteed to have a unique minimal model (as it is a definite logic program).

Imperial College
London

Reduct: Example

$$X = \left\{ \begin{array}{c} \texttt{node}(1..2), \texttt{edge}(1,1), \texttt{edge}(2,2), \\ \texttt{reach}(1), \texttt{unreachable\_node} \end{array} \right\}$$

$$\mathcal{RG}(P) = \left\{ \begin{array}{l} \texttt{node}(1..2).\ \texttt{edge}(1,1).\ \texttt{edge}(2,2). \\ \texttt{reach}(1)\ \texttt{:-}\ \texttt{edge}(1,1). \\ \texttt{reach}(1)\ \texttt{:-}\ \texttt{reach}(1), \texttt{edge}(1,1). \\ \texttt{unreachable\_node}\ \texttt{:-}\ \texttt{node}(1), \texttt{not}\ \texttt{reach}(1). \\ \texttt{unreachable\_node}\ \texttt{:-}\ \texttt{node}(2), \texttt{not}\ \texttt{reach}(2). \end{array} \right\}$$

Imperial College
London

Reduct: Example

$$X = \left\{ \begin{array}{c} \texttt{node}(1..2), \texttt{edge}(1,1), \texttt{edge}(2,2), \\ \texttt{reach}(1), \texttt{unreachable\_node} \end{array} \right\}$$

$$\mathcal{RG}(P) = \left\{ \begin{array}{l} \texttt{node}(1..2).\ \texttt{edge}(1,1).\ \texttt{edge}(2,2). \\ \texttt{reach}(1)\ \texttt{:-}\ \texttt{edge}(1,1). \\ \texttt{reach}(1)\ \texttt{:-}\ \texttt{reach}(1), \texttt{edge}(1,1). \\ \texttt{unreachable\_node}\ \texttt{:-}\ \texttt{node}(1), \texttt{not}\ \texttt{reach}(1). \\ \texttt{unreachable\_node}\ \texttt{:-}\ \texttt{node}(2), \texttt{not}\ \texttt{reach}(2). \end{array} \right\}$$

Imperial College London

Reduct: Example

$$X = \left\{ \begin{array}{c} \texttt{node}(1..2), \texttt{edge}(1,1), \texttt{edge}(2,2), \\ \texttt{reach}(1), \texttt{unreachable\_node} \end{array} \right\}$$

$$\mathcal{RG}(P) = \left\{ \begin{array}{l} \texttt{node}(1..2).\ \texttt{edge}(1,1).\ \texttt{edge}(2,2). \\ \texttt{reach(1) :- edge}(1,1). \\ \texttt{reach(1) :- reach}(1), \texttt{edge}(1,1). \\ \texttt{unreachable\_node :- node}(1), \texttt{not reach}(1). \\ \texttt{unreachable\_node :- node}(2), \texttt{not reach}(2). \end{array} \right\}$$

Imperial College
London

Reduct: Exercise

Please attempt Question 1 (b) (i) from the tutorial sheet.

Imperial College London

Answer Sets

An interpretation X is an answer set of a program P if and only if X is the unique minimal model of $\mathcal{RG}(P)^X$ (written $X = M(\mathcal{RG}(P)^X)$)

$X = \{\texttt{heads}\}$

$$P = \left\{ \begin{array}{l} \texttt{heads :- not tails.} \\ \texttt{tails :- not heads.} \end{array} \right\}$$

$\mathcal{RG}(P)^X = \{ \texttt{heads.} \}$

$X = M(\mathcal{RG}(P)^X)$

$$X \in AS(P)$$

In summary, the steps of checking whether X is an answer set of a program P are:

1. Compute the relevant grounding $\mathcal{RG}(P)$.
2. Compute the reduct of $\mathcal{RG}(P)$ with respect to X.
3. Check whether $X = M(\mathcal{RG}(P)^X)$.

When P is a ground (propositional) program, we often omit the relevant grounding step ($\mathcal{RG}(P) = P$).

The set of all answer sets of a program P is denoted $AS(P)$

Imperial College
London

Answer Sets

An interpretation X is an answer set of a program P if and only if X is the unique minimal model of $\mathcal{RG}(P)^X$ (written $X = M(\mathcal{RG}(P)^X)$)

$X = \{\texttt{tails}\}$

$$P = \left\{ \begin{array}{l} \texttt{heads :- not tails.} \\ \texttt{tails :- not heads.} \end{array} \right\}$$

$\mathcal{RG}(P)^X = \{ \texttt{ tails. } \}$

$X = M(\mathcal{RG}(P)^X)$

$$X \in AS(P)$$

In summary, the steps of checking whether X is an answer set of a program P are:

1. Compute the relevant grounding $\mathcal{RG}(P)$.
2. Compute the reduct of $\mathcal{RG}(P)$ with respect to X.
3. Check whether $X = M(\mathcal{RG}(P)^X)$.

When P is a ground (propositional) program, we often omit the relevant grounding step ($\mathcal{RG}(P) = P$).

The set of all answer sets of a program P is denoted $AS(P)$

Imperial College London

Answer Sets

An interpretation X is an answer set of a program P if and only if X is the unique minimal model of $\mathcal{RG}(P)^X$ (written $X = M(\mathcal{RG}(P)^X)$)

$X = \{\texttt{heads}, \texttt{tails}\}$ $\mathcal{RG}(P)^X = \{ \quad \}$

$$P = \left\{ \begin{array}{l} \texttt{heads :- not tails.} \\ \texttt{tails :- not heads.} \end{array} \right\}$$

$X \neq M(\mathcal{RG}(P)^X)$

$$X \notin AS(P)$$

In summary, the steps of checking whether X is an answer set of a program P are:

1. Compute the relevant grounding $\mathcal{RG}(P)$.
2. Compute the reduct of $\mathcal{RG}(P)$ with respect to X.
3. Check whether $X = M(\mathcal{RG}(P)^X)$.

When P is a ground (propositional) program, we often omit the relevant grounding step ($\mathcal{RG}(P) = P$).

The set of all answer sets of a program P is denoted $AS(P)$

Imperial College London

Answer Sets

An interpretation X is an answer set of a program P if and only if X is the unique minimal model of $\mathcal{RG}(P)^X$ (written $X = M(\mathcal{RG}(P)^X)$)

$X = \{\}$

$\mathcal{RG}(P)^X = \left\{ \begin{array}{l} \texttt{heads.} \\ \texttt{tails.} \end{array} \right\}$

$P = \left\{ \begin{array}{l} \texttt{heads :- not tails.} \\ \texttt{tails :- not heads.} \end{array} \right\}$

$X \neq M(\mathcal{RG}(P)^X)$

$$X \notin AS(P)$$

In summary, the steps of checking whether X is an answer set of a program P are:

1. Compute the relevant grounding $\mathcal{RG}(P)$.
2. Compute the reduct of $\mathcal{RG}(P)$ with respect to X.
3. Check whether $X = M(\mathcal{RG}(P)^X)$.

When P is a ground (propositional) program, we often omit the relevant grounding step ($\mathcal{RG}(P) = P$).

The set of all answer sets of a program P is denoted $AS(P)$

Imperial College London

Minimal model: Iterative Construction

- A definite logic program P, has a unique minimal model $M(P)$. It can be computed using the simple procedure below:

```
1: procedure $M(P)$
2:     $M = \emptyset$
3:     do
4:         $M' = \left\{ head(R) \;\middle|\; \begin{array}{c} R \in P, \\ head(R) \notin M, \\ body(R) \subseteq M \end{array} \right\}$
5:         $M = M \cup M'$
6:     while $M' \neq \emptyset$
7:     return $M$
8: end procedure
```

The minimal model of any definite logic program can be computed by starting with M as the empty set and iteratively adding to M any atom that is the head of a rule in P whose body is already satisfied by M. In each step, M' is equal to the immediate consequences $T_P(M)$. This procedure computes the *least fixpoint* of T_P, which you have covered earlier in the course.

Imperial College
London

Minimal Model: Example

$$\mathcal{RG}(P)^X = \left\{ \begin{array}{l} \texttt{node}(1..2).\ \texttt{edge}(1,1).\ \texttt{edge}(2,2). \\ \texttt{reach(1) :- edge}(1,1)\texttt{.} \\ \texttt{reach(1) :- reach}(1)\texttt{,edge}(1,1)\texttt{.} \\ \texttt{unreachable\_node :- node}(2)\texttt{.} \end{array} \right\}$$

$$M(\mathcal{RG}(P)^X) = \{ \qquad\qquad\qquad\qquad \}$$

Imperial College
London

Minimal Model: Example

$$\mathcal{RG}(P)^X = \left\{ \begin{array}{l} \texttt{node}(1..2).\ \texttt{edge}(1,1).\ \texttt{edge}(2,2). \\ \texttt{reach(1) :- edge}(1,1)\texttt{.} \\ \texttt{reach(1) :- reach}(1), \texttt{edge}(1,1)\texttt{.} \\ \texttt{unreachable\_node :- node}(2)\texttt{.} \end{array} \right\}$$

$$M(\mathcal{RG}(P)^X) = \{\ \texttt{node}(1..2), \texttt{edge}(1,1), \texttt{edge}(2,2)\ \ \}$$

Imperial College London

Minimal Model: Example

$$\mathcal{RG}(P)^X = \left\{\begin{array}{l} \texttt{node}(1..2).\ \texttt{edge}(1,1).\ \texttt{edge}(2,2). \\ \texttt{reach(1) :- edge}(1,1)\texttt{.} \\ \texttt{reach(1) :- reach}(1), \texttt{edge}(1,1)\texttt{.} \\ \texttt{unreachable\_node :- node}(2)\texttt{.} \end{array}\right\}$$

$$M(\mathcal{RG}(P)^X) = \left\{\begin{array}{c} \texttt{node}(1..2), \texttt{edge}(1,1), \texttt{edge}(2,2), \\ \texttt{reach}(1), \texttt{unreachable\_node} \end{array}\right\}$$

Imperial College
London

Minimal Model: Example

$$\mathcal{RG}(P)^X = \left\{ \begin{array}{l} \texttt{node}(1..2).\ \texttt{edge}(1,1).\ \texttt{edge}(2,2). \\ \texttt{reach}(1)\ \texttt{:-}\ \texttt{edge}(1,1). \\ \texttt{reach}(1)\ \texttt{:-}\ \texttt{reach}(1), \texttt{edge}(1,1). \\ \texttt{unreachable\_node}\ \texttt{:-}\ \texttt{node}(2). \end{array} \right\}$$

$$M(\mathcal{RG}(P)^X) = \left\{ \begin{array}{c} \texttt{node}(1..2), \texttt{edge}(1,1), \texttt{edge}(2,2), \\ \texttt{reach}(1), \texttt{unreachable\_node} \end{array} \right\}$$

$$X = M(\mathcal{RG}(P)^X)$$

Imperial College
London

Minimal Model: Example

$$\mathcal{RG}(P)^X = \left\{ \begin{array}{l} \texttt{node(1..2). edge(1,1). edge(2,2).} \\ \texttt{reach(1) :- edge(1,1).} \\ \texttt{reach(1) :- reach(1), edge(1,1).} \\ \texttt{unreachable\_node :- node(2).} \end{array} \right\}$$

$$M(\mathcal{RG}(P)^X) = \left\{ \begin{array}{c} \texttt{node(1..2), edge(1,1), edge(2,2),} \\ \texttt{reach(1), unreachable\_node} \end{array} \right\}$$

$$X \in AS(P)$$

Imperial College
London

Reachability in Graphs in ASP

$$X = \{\ \texttt{node}(1..2), \texttt{edge}(1,1), \texttt{edge}(2,2), \texttt{reach}(1..2)\ \}$$

$$\mathcal{RG}(P) = \left\{ \begin{array}{l} \texttt{node}(1..2).\ \texttt{edge}(1,1).\ \texttt{edge}(2,2). \\ \texttt{reach}(1)\ \texttt{:-}\ \texttt{edge}(1,1). \\ \texttt{reach}(1)\ \texttt{:-}\ \texttt{reach}(1), \texttt{edge}(1,1). \\ \texttt{unreachable\_node}\ \texttt{:-}\ \texttt{node}(1), \texttt{not}\ \texttt{reach}(1). \\ \texttt{unreachable\_node}\ \texttt{:-}\ \texttt{node}(2), \texttt{not}\ \texttt{reach}(2). \end{array} \right\}$$

$$M(\mathcal{RG}(P)^X) = \left\{ \begin{array}{c} \texttt{node}(1..2), \texttt{edge}(1,1), \texttt{edge}(2,2), \\ \texttt{reach}(1) \end{array} \right\}$$

$$X \notin AS(P)$$

Imperial College London

Reachability in Graphs in ASP

$$X = \{\ \texttt{node}(1..2), \texttt{edge}(1,1), \texttt{edge}(2,2), \texttt{reach}(1..2)\ \}$$

$$\mathcal{RG}(P)^X = \left\{ \begin{array}{l} \texttt{node}(1..2).\ \texttt{edge}(1,1).\ \texttt{edge}(2,2). \\ \texttt{reach}(1)\ \texttt{:-}\ \texttt{edge}(1,1). \\ \texttt{reach}(1)\ \texttt{:-}\ \texttt{reach}(1), \texttt{edge}(1,1). \\ \texttt{unreachable\_node}\ \texttt{:-}\ \texttt{node}(1), \texttt{not}\ \texttt{reach}(1). \\ \texttt{unreachable\_node}\ \texttt{:-}\ \texttt{node}(2), \texttt{not}\ \texttt{reach}(2). \end{array} \right\}$$

$$M(\mathcal{RG}(P)^X) = \left\{ \begin{array}{c} \texttt{node}(1..2), \texttt{edge}(1,1), \texttt{edge}(2,2), \\ \texttt{reach}(1) \end{array} \right\}$$

$$X \notin AS(P)$$

Imperial College
London

Answer Sets: Exercise

Please attempt Question 1 (c) (i) from the tutorial sheet.

Imperial College London

Relationship between ASP and Clark completion

For any propositional program, $A \in AS(P) \Rightarrow A$ is a model of $comp(P)$.

The converse does not hold!
So what is special about answer sets?

We now consider the relationship between ASP and the Clark completion that you saw earlier in the course. In this lecture, we only do this comparison for propositional logic programs, as ASP requires programs to be ground as a first step. All answer sets of a program P are models of its completion $comp(P)$.

We saw for the graphs examples though that for some programs there are models of the completion that are not answer sets. So what is special about answer sets?

Imperial College London

Unfounded Sets

Given a ground program P and an interpretation I, a set $U \subseteq I$ is said to be an unfounded subset of I wrt P if and only if there is no rule R such that (1)-(3) hold:

1. I satisfies $body(R)$
2. $head(R) \in U$
3. $U \cap body^{+}(R) = \emptyset$

Given any program P, the answer sets are equal to the models of $comp(P)$ which have no non-empty unfounded subsets wrt P.
In fact, these are the models of P that have no non-empty unfounded subsets wrt P.

Answer sets can also be defined in terms of unfounded sets. Essentially subset U of an interpretation is unfounded with respect to the program if there is no rule R that proves anything in U without using the elements of U (in the positive body of R).

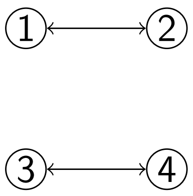
Unfounded sets allow us to highlight the relationship between the earlier semantics in this course and the answer set semantics. The answer sets of a propositional program P are the models of $comp(P)$ with no unfounded subsets (wrt P). In fact, these are also the models of P with no non-empty unfounded subsets wrt P.
You can think of an unfounded subset U of an interpretation I as not having any "external support" – that is, the only way to prove any elements of U is by using other elements of U (as the only rules with elements of U in the head whose body is satisfied by I contain at least one element of U in their positive body).
Consider the program:

```
p :- not q, r.
q :- not r.
r :- p.
```

The completion of this program has two models – {p, r} and {q} – but it only has one answer sets – {q}. {p, r} is an unfounded subset of {p, r} wrt the program. Every rule for p and r uses either p or r in the body.

Imperial College
London

Reachability in graphs and unfounded sets



```
1 :  node(1..4).
2 :  edge(1,2). edge(2,1).
3 :  edge(3,4). edge(4,3).
```

```
4 :  reach(N) :- edge(1,N).
5 :  reach(N) :- reach(N2), edge(N2,N).
6 :  unreachable_node :- node(N), not reach(N).
```

$$I = Facts(P) \cup \{\texttt{reach(1..4)}\}$$

$\{\texttt{reach(3)}, \texttt{reach(4)}\}$ is an unfounded subset of I wrt P. Hence $I \notin AS(P)$

Imperial College
London

Summary

- The completion of some programs can have unintuitive models.
- Any definite program has a unique minimal model $M(P)$
- $X \in AS(P) \Leftrightarrow X = M(\mathcal{RG}(P)^X)$
- $X \in AS(P) \Leftrightarrow X$ is a model of $comp(\mathcal{RG}(P))$ and X has no non-empty unfounded subsets wrt P.