This document summarises the key points from units 6-9 in the Logic-based Learning course (C304). This is provided to aid your revision, but be aware that all concepts mentioned here are explored in greater depth in the slides and the notes. If you prefer to revise from the notes, feel free; there should be nothing in this document which was not covered in the lectures/notes.

1 ASP and the stable model semantics

1.1 Syntax

Definition 1.1. We will consider 5 different types of rules in this course.

1. A *definite rule* is of the form:
   $h \leftarrow b_1, \ldots, b_n$, where $h, b_1, \ldots, b_n$ are all atoms.
2. A *normal rule* is of the form:
   $h \leftarrow b_1, \ldots, b_n, \text{not } c_1, \ldots, \text{not } c_m$, where $h, b_1, \ldots, b_n, c_1, \ldots, c_m$ are all atoms.
3. A *contraint* is of the form:
   $\leftarrow b_1, \ldots, b_n, \text{not } c_1, \ldots, \text{not } c_m$, where $h, b_1, \ldots, b_n, c_1, \ldots, c_m$ are all atoms. The empty head is often called *false*.
4. A *choice rule* is of the form:
   $l\{h_1, \ldots, h_o\}u \leftarrow b_1, \ldots, b_n, \text{not } c_1, \ldots, \text{not } c_m$, where $h_1, \ldots, h_o, b_1, \ldots, b_n, c_1, \ldots, c_m$ are all atoms and l and u are integers. The head of this rule is called an *aggregate*.
5. An *optimisation statement* is of the form:
   $\#minimise[a_1 = w_1, \ldots, a_n = w_n]$. or
   $\#maximise[a_1 = w_1, \ldots, a_n = w_n]$.
   where $a_1, \ldots, a_n$ are ground atoms and $w_1, \ldots, w_n$ are integers called the *weights*.

Definition 1.2. A *definite logic program* is a set of definite rules.

Definition 1.3. A *normal logic program* is a set of normal rules. (Note that definite rules are normal rules too!)

Definition 1.4. Given any normal rule, choice rule or constraint R, R is *safe* iff every variable which occurs in R occurs at least once in $body^+(R)$.

Definition 1.5. An ASP program is a set of normal rules, choice rules, constraints (all of which must be safe) and optimisation statements.

Note that this definition of an ASP program is as we have used it in the course. In fact, there are other types of rule which we did not have time to consider.

1.2 Semantics

Definition 1.6. The *Herbrand base* of a logic program P is the set of all ground atoms which can be made from predicates, functions and constants which appear in P.

Definition 1.7. A *Herbrand interpretation* of a program P assigns every atom in the Herbrand base of P to either true of false. A Herbrand interpretation I is usually written as the set of atoms which I assigns to true.

Definition 1.8. The *relevant grounding* of an ASP program P (written $ground(P)$) can be constructed by starting with the empty program $Q = \{\}$ and continually adding to Q any rule R such that:

1. R is a ground instance of some rule in P.
2. Every atom $b \in body^+(R)$ already occurs in the head of some rule in Q.

Once no further rules can be added to Q in this way, Q is equal to $ground(P)$.

Definition 1.9. Given any rule R, the body is satisfied by a Herbrand interpretation I if and only if $body^+(R) \subseteq I$ and $body^-(R) \cap I = \emptyset$.

- If the head of R is an atom h, then the head is satisfied by I if and only if $h \in I$.
- If R is a constraint, then the head of R is never satisfied.
- If the head of R is an aggregate $l\{h_1, \ldots, h_n\}u$ then the head of R is satisfied by I if and only if $l \leq |\{h_1, \ldots, h_n\} \cap I| \leq u$.

Definition 1.10. For any program P, a Herbrand interpretation I is a *Herbrand model* if and only if for every rule R in $ground(P)$ such that the body of R is satisfied by I, the head of R is also satisfied by I.

Definition 1.11. Given a program P, a Herbrand model M is *minimal* if and only if there is no strict subset M' of M such that M' is also a Herbrand model of P.

Note that for any definite logic program P, there is a unique minimal Herbrand model which we will call the *least Herbrand model* of P (written $M(P)$).

We now present the semantics for ASP programs as used in this course. If you look elsewhere at some of the other ASP literature, you will see that the semantics for aggregates is defined differently (and not as intuitively). This is because, when you allow aggregates in the body, things get more complicated. As in this course we only allow choice rules (with aggregates in the head), we present a simplified semantics which is equivalent for our subset of ASP to the more complicated semantics given in the literature.

The following definition combines the definition of a reduct for a normal logic program (as presented in the lecture) with the extenstions we added in unit 7 to translate choice rules and constraints. In this definition we ignore any optimisation statements in the program (optimisation statements do not effect the answer sets of a program).

Definition 1.12. Given any ground ASP program P and any Herbrand interpretation X, the *reduct* of P with respect to X (written P^X) is construced in 4 steps:

1. Replace the heads of all constraints with $\bot$.
2. For each choice rule R, if its head is not satisfied, replace its head with $\bot$.

   If its head is satisfied then remove R and for each atom h in the aggregate at the head of R such that $h \in X$, add the rule $h \leftarrow body(R)$.
3. Remove any rule R such that the body of R contains the negation of an atom in X.
4. Remove all negation from any remaining rules

The first two steps in constructing the reduct only concern programs which contain choice rules and constraints. If P starts as a normal logic program then you can skip to step 3. The important thing to note is that the reduct is always a definite logic program and thus always has a unique minimal Herbrand model (the least Herbrand model $M(P)$).

Definition 1.13. For any program P and any Herbrand interpretation X, X is an *answer set* of P if and only if X is the least Herbrand model of the reduct of $ground(P)$ with respect to X.

i.e. X is an answer set of P if and only if $X = M((ground(P))^X)$.

Remark 1.14. As no Herbrand interpretation I can assign $\bot$ to true, if the body of any constraint in a program P is satisfied by I, then I can not be an answer set of P ($M((ground(P))^I)$ would contain $\bot$ but I wouldn't).

In this document, we will write $AS(P)$ to denote the set of answer sets of a program P.

Now that we have defined what an answer set is, we can define two different notions of entailment *brave entailment* and *cautious entailment*.

Definition 1.15. Given an ASP program P, P is said to *bravely entail* an atom a (written $P \models_b a$) if and only if there is at least one answer set A of P such that $a \in A$.

P is said to *cautiously entail* a (written $P \models_c a$) if and only if for every answer set A of P, $a \in A$.

Remark 1.16. We also write that $P \models_b$ not a and $P \models_c$ not a if a is false in at least one (respectively every) answer set of P.

You should note that $P \models_b$ not a is *not* the same as $P \not\models_b a$; in fact $P \models_b$ not $a \equiv P \not\models_c a$. Similarly $P \models_c$ not $a \equiv P \not\models_b a$.

1.2.1 Optimisation Statements

Optimisation statements are a useful tool in ASP as they allow us to specify which answer sets are better than others. We have only considered programs with one ground optimisation statement in this course, but if you read around in the literature you will see that you can express far more in ASP using multiple optimisation statements. Again, we present a slightly simplified semantics as we have not considered more general optimisation statements.

Definition 1.17. Consider an program P containing one optimisation statement $\#minimise[a_1 = w_1, \ldots, a_n = w_n]$. Given an answer set A of P, we say that the *optimality* of A is the sum $\sum_{a_i \in A : i \in [1,n]} w_i$.

An answer set A of P is said to be *optimal* if no other answer set A' of P has a lower optimality.

The definition of the semantics of programs with a single maximise statement is similar to that of minimise, but we give it anyway for completeness.

Definition 1.18. Consider an program P containing one optimisation statement $\#maximise[a_1 = w_1, \ldots, a_n = w_n]$. Given an answer set A of P, we say that the *optimality* of A is the sum $\sum_{a_i \in A : i \in [1,n]} w_i$.

An answer set A of P is said to be *optimal* if no other answer set A' of P has a higher optimality.

1.3 Useful Properties

In this section we give some useful properties which you are not expected to recall, but may be helpful in determining the answer sets of a program.

Property 1.19. Every definite logic program P has exactly one answer set. This is the same as its least Herbrand model.

Property 1.20. For any normal logic program P, every answer set of P must be a minimal model of P.

Note that this property doesn't hold for ASP programs which contain choice rules. Also, the converse of this property does not hold! Not all minimal models are answer sets.

Property 1.21. For any ASP program P, all answer sets of P must be models of P.

Again, the converse of this property does not hold. Not all models are answer sets.

These properties may save you time; if you notice that an interpretation is not a model, then there is no point checking whether it is an answer set.

2 Brave and Cautious Induction

Definition 2.1. A *brave induction* task consists of an ASP program B called the background knowledge, and two sets of atoms E^+ and E^- called the positive and negative examples (respectively). We will often write the task as a tuple $\langle B, E^+, E^- \rangle$.

Definition 2.2. Given a brave induction task $\langle B, E^+, E^- \rangle$, an hypothesis H (another ASP program) is an inductive solution (written $H \in ILP_b \langle B, E^+, E^- \rangle$) if and only if there is an answer set A of $B \cup H$ such that:

1. $\forall e \in E^+ : e \in A$
2. $\forall e \in E^- : e \notin A$

Note that the following definition for a cautious induction task is exactly the same as a brave induction task. It is only the definitions of the inductive solutions which differ.

Definition 2.3. A *cautious induction* task consists of an ASP program B called the background knowledge, and two sets of atoms E^+ and E^- called the positive and negative examples (respectively). We will often write the task as a tuple $\langle B, E^+, E^- \rangle$.

Definition 2.4. Given a cautious induction task $\langle B, E^+, E^- \rangle$, an hypothesis H (another ASP program) is an inductive solution (written $H \in ILP_c \langle B, E^+, E^- \rangle$) if and only if:

1. $B \cup H$ is satisfiable (it has at least one answer set).
2. For every answer set A of $B \cup H$:
   (a) $\forall e \in E^+ : e \in A$
   (b) $\forall e \in E^- : e \notin A$

3 ASPAL

Similarly to earlier algorithms, ASPAL relies on mode declarations to define its hypothesis space.

Definition 3.1. An ASPAL *mode declaration* is of the form $modeh(r, s)$ or $modeb(r, s)$, where r is an integer called the recall, and s is an literal (an atom, or the negation by failure of an atom) whose arguments are constants which are all prefixed with either $\#$, $+$ or $-$.

Definition 3.2. Given a set of mode declarations M, an ASPAL *skeleton rule* is a normal rule of the form

$h \leftarrow b_1, \dots b_n, t_1, \dots, t_m$ such that:

1. h is an atom constructed from one of the *modeh*'s by replacing all the input variables with different variables $V1, V2, \ldots$ and all the constants with different variables $C1, C2, \ldots$.
2. Each b_i is a literal constructed from one of the *modeb*'s by replacing all the input variables with variables $V1, V2 \ldots$ which have already occured previously as either the output variable of another atom or as an input variable in the head; all the constants with new variables (which don't appear elsewhere in the rule) $C1, C2, \ldots$, and all of the output variables with new variables $V1, \ldots$.
3. Each variable V_i (excluding the C's) has an extra atom t_i added to the body which specifies the type of the variable (this is the constant which appears after the $+$ or $-$ in the mode declaration).
4. No variable takes two types.
5. The number of literals in the body (excluding the types) n is less than or equal to L_{max}.
6. The number of variables V_i (excluding the C's) is less than or equal to V_{max}.
7. No predicate occurs more times in a single rule than the value of the recall in its mode declaration.

Definition 3.3. A set of skeleton rules S_M is *maximal* given a set of mode declarations M and the values of L_{max} and V_{max} if no further skeleton rules can be added which are not equivalent to a skeleton rule which is already in S_M.

Definition 3.4. For each skeleton rule $R \in S_M$, we give R a unique identifier written R_{id}[1].

A rule R' is an instance of R if R has been constructed by replacing all the constant variable $C1, \ldots$ with constants which are of the correct type (the type is determined from the background knowledge). The meta encoding of R' (written $meta(R')$ in this document) is the atom $rule(R_{id}, c_1, \ldots, c_n)$ where $c_1, \ldots, c_n$ are the constants which appear in R'.

Definition 3.5. The *length* of a skeleton rule R (written $|R|$ in this document) is equal to the number of literals in R (excluding the atoms which are added to enforce the types).

Definition 3.6. Given a brave induction task $\langle B, E^+, E^- \rangle$ and a set of mode declarations M, the ASPAL meta encoding of the task is composed of:

- B (The background knowledge is unchanged).
- For each skeleton rule $R \in S_M$ we add:
  $head(R) \leftarrow body(R), rule(1, C_1, \ldots C_n)$ where the C_i's are the constant variables which occur in the skeleton rule.
- Let $E^+ = \{e_1^+, \ldots, e_n^+\}$ and $E^- = \{e_1^-, \ldots, e_m^-\}$. Then we add:
  $goal \leftarrow e_1^+, \ldots e_n^+, \text{not } e_1^-, \ldots, \text{not } e_m^-$ and
  $\leftarrow \text{not } goal$.

[1]For the purposes of this course, you can assume that the positive integers $1, 2, 3, \ldots$ will be given as identifiers to rules in the order you write them down.

- Let $\{R^1, \ldots, R^n$ be the set of all instances of rules in S_M. Then we add:
  $0\{meta(R^1), \ldots, meta(R^n)\}n$. and
  $\#minimise[meta(R^1) = |R^1|, \ldots, meta(R^n) = |R^n|]$. (where $meta$ is as in definition 3.4).

Given some mode declarations M, the optimal (shortest) inductive solutions of the brave task $\langle B, E^+, E^- \rangle$ (which are compatible with M) can be found using the following steps:

1. Find a maximal set of skeleton rules S_M.
2. Represent the task $\langle B, E^+, E^- \rangle$ with S_M using the ASPAL encoding.
3. Solve this encoding for its optimal answer sets.
4. Map each answer set back to a hypothesis by extracting each atom of the predicate *rule* and mapping it back to the rule instance which it represents.

Remark 3.7. In fact, if we were interested in all inductive solutions, rather than all optimal inductive solutions, we can simply ignore the optimisation statement and solve the encoding for all answer sets. The answer sets of the encoding can be mapped to the whole set of brave inductive solutions.

Remark 3.8. ASPAL is sound and complete with respect to the definition of a brave inductive solution. This means that there is an answer set of the meta encoding which maps back to a hypothesis H if and only if H is a brave inductive solution.

4 Learning from Answer Sets

In this section we recap Unit 9, Learning from Answer Sets and ILASP. We omit the section on ILASP as is not examinable and this document is aimed at being a revision aid.

In the Learning from Answer Sets framework, as with ASPAL (and previous systems), mode declarations are used to express a language bias. As we are now interested in learning full ASP programs this language bias is rather different. Concepts such as input and output variables do not make sense in ASP where all variables are grounded at the same time before solving the program.

Definition 4.1. A Learning from Answer Sets *mode declaration* is of the form $modeh(r, s)$ or $modeb(r, s)$, where r is an integer called the recall, and s is an literal (an atom, or the negation by failure of an atom) whose arguments are terms $var(type)$ or $const(type)$, where $type$ is some constant.

Definition 4.2. Given a set M of Learning from Answer Sets mode declarations and given the parameters H_{min}, H_{max}, V_{max} and B_{max} , the search space S_M includes the rules of the form:

$HEAD \leftarrow b_1, \ldots, b_n, \text{not } c_1, \ldots, \text{not } c_m$

where $HEAD$ is one of the following:

1. empty (meaning the rule is a constraint).

2. an atom which is compatible with the some *modeh* in M replacing all the variables with variables and the constants with constants of the correct type (according to the background knowledge[2]).
3. an aggregate $l\{h_1, \ldots, h_o\}u$ where each h_i is an atom which is compatible with the some *modeh* in M (as above); and $H_{min} \leq o \leq H_{max}$; and finally, $1 \leq l \leq u \leq o$.

and the following conditions are all true:

1. The number of variables in the whole rule is less than or equal to V_{max}.
2. $n + m \leq B_{max}$
3. No variable in the rule can have two different types (where the types are specified from the mode declarations which generated these variables).
4. The rule is safe.

Note that other than the lack of input and output variables there are a few other differences to ASPAL mode declarations. Firstly, we do not add the type atoms to our rules. In a sense, we can see this as our rules not being "strongly typed". If our rule needed these type atoms for it to be an inductive solution then they would actually have to be given in the body declarations. We also now allow constraints and choice rules to be in our hypothesis space.

What this means is that our search space is in practice much bigger than ASPAL's. This can be seen as an advantage in that our solution is more likely to be in there, or a disadvantage in that it makes the computation harder.

Definition 4.3. A *partial interpretation* e is a pair of sets of atoms $\langle e^{inc}, e^{exc} \rangle$. e^{inc} and e^{exc} are called (respectively) the *inclusions* and the *exclusions* of e.

Definition 4.4. Given a Herbrand interpretation I and a partial interpretation e, I is said to *extend* e if and only if $e^{inc} \subseteq I$ and $e^{exc} \cap I = \emptyset$.

In Learning from Answer Sets, our examples are partial interpretations rather than atoms. This is makes more sense when trying to learn ASP programs, as the main "object" in ASP is the answer set (so our examples are now (partial) answer sets); whereas when trying to learn prolog, our main "object" is the query, so our examples can be thought of as the results of queries.

Definition 4.5. A *Learning from Answer Sets* task is a tuple $T = \langle B, S_M, E^+, E^- \rangle$ where B is an ASP program called the background knowledge, S_M is the search space (usually defined by some mode declarations M) and E^+ and E^- are sets of partial interpretations called the positive and negative examples (respectively).

The set of inductive solutions of T (written $ILP_{LAS}(T)$) is the set of hypotheses H such that:

[2]In fact, the implementation on the ILP frameworks site requires that you give the list of constants with their types. This allows a more flexible choice of constants in the search space (and also allows for the case of an unsatisfiable background knowledge).

1. $H \subseteq S_M$
2. $\forall e \in E^+ : \exists A \in AS(B \cup H)$ such that A extends e.
3. $\forall e \in E^- : \nexists A \in AS(B \cup H)$ such that A extends e.

4.1 Relationship to Brave and Cautious Induction

In this section, we omit the search space from our Learning from Answer Sets tasks (as brave and cautious induction were defined with no search space). The task $\langle B, E^+, E^- \rangle$ has inductive solutions as in definition 4.5, but with the condition $H \subseteq S_M$ omitted.

Mapping a brave induction task to a Learning from Answer Sets task is fairly simple. In brave induction, the task is to find a hypothesis such that $B \cup H$ has at least one answer set which contains all of the positive examples (E^+) and none of the negative examples (E^-). This is equivalent to $B \cup H$ having at least one answer set which extends the partial interpretation $\langle E^+, E^- \rangle$ (which is again equivalent to H covering the positive example $\langle E^+, E^- \rangle$).

Property 4.6. Given any Brave Induction task $T = \langle B, \{e_1^+, \ldots, e_n^+\}, \{e_1^-, \ldots, e_m^-\} \rangle$ (where, as specified by the definition, the examples are atoms).

The brave induction task can be mapped to a Learning from Answer Sets task with one positive example: $\langle \{e_1^+, \ldots, e_n^+\}, \{e_1^-, \ldots, e_m^-\} \rangle$.

i.e. the inductive solutions of T are exactly the inductive solutions of $ILP_{LAS}(\langle B, \{\langle \{e_1^+, \ldots, e_n^+\}, \{e_1^-, \ldots, e_m^-\} \rangle\}, \emptyset \rangle)$.

The representation of cautious induction in Learning from Answer Sets is slightly more involved. We must have one positive example (which is empty) which just says that there is at least one answer set of $B \cup H$. And one negative example for each example in the cautious tasks.

Each positive example a in cautious induction says that every answer set of $B \cup H$ must contain a; so for each of these we add a negative example saying that no answer set of $B \cup H$ is allowed to not contain a (an equivalent condition).

Each negative example a in cautious induction says that every answer set of $B \cup H$ must not contain a; so for each of these we add a negative example saying that no answer set of $B \cup H$ is allowed to contain a (again, an equivalent condition).

Property 4.7. Given any Cautious Induction task $T = \langle B, \{e_1^+, \ldots, e_n^+\}, \{e_1^-, \ldots, e_m^-\} \rangle$ (where, as specified by the definition, the examples are atoms).

The cautious induction task can be mapped to a Learning from Answer Sets task with one (empty) positive example and $n + m$ negative examples.

The Learning from Answer Sets task is written:

$ILP_{LAS}(\langle B, \emptyset, \{\langle \emptyset, \{e_1^+\} \rangle, \ldots, \langle \emptyset, \{e_n^+\} \rangle, \{\langle \{e_1^-\}, \emptyset \rangle, \ldots, \langle \{e_m^-\}, \emptyset \rangle\} \rangle\}, \emptyset \rangle)$.

The inductive solutions of this task are exactly the inductive solutions of the cautious task.