Disjunctive Answer Set Programming

CMPUT 325 LAB Winter 2023

Spencer Killen

March 17, 2023

An ASP program consists of a series of rules. Atoms are globally named symbols that appear within rules. Here we describe disjunctive answer set semantics [1].

Example Rule:

```
foo, blah :- baz, bar, not jam, not can, not bread.
```

Each rule has the following parts:

1. the head ("foo, blah" in above). Can contain 0 or more atoms.
2. the positive body ("baz, bar" in above). Can contain 0 or more atoms.
3. the negative body ("not jam, not can, not bread"). Can contain 0 or more atoms, each preceded by "not'"

Note that Clingo allows you inter interleave atoms in the positive and negative body of rules. E.g.,

```
:- not a, b, not j.
```

Is a valid rule.

Rather than "running" an answer set program, we try to find a true/false assignment to all atoms that appear in the program. Assignments are treated as sets where the set contains only the atoms that are assigned true. Such a truth assignment is called a **model** of a program if all rules are satisfied by the assignment. The following describes rule satisfaction:

- The positive body of a rule is satisfied by an answer set if all atoms in the body are **true**. If there are no atoms in the positive body, then it is automatically satisfied.
- The negative body of a rule is satisfied by an answer set if all of its atoms are **false**.
- A rule is satisfied if either:

- Either the rule's positive or negative body is not satisfied.
- The rule's body is satisfied and one or more atoms in the head of the rule is true (Note: if the head has 0 atoms this is not possible)

A model of a program is called an **answer set** if it's minimal, i.e. there is no other model of the program that is $\subseteq$ smaller (An assignment is treated as a set containing the atoms assigned true). **NOTE:** a model X may be an answer set even if there exists another model Y with a fewer number of atoms, so long there is some atom that is true in Y that is not true in X.

Another way of looking at answer sets / minimality: if we can change some of the true atoms in an answer set to be false, then there must be some rule that is no longer satisfied as a result. This is the same as saying that every atom must be justified by a rule. When performing such a "minimality check", negative rule bodies are given special treatment: When we change true atoms to be false, negative bodies are only satisfied if they were satisfied before we made the change.

Example:

```
a.
a, b :- not c.
```

There are 3 atoms, so there are 8 possible assignments that could be answer sets. We go through each possible assignment using pairs (T, F) where T is the set of atoms assigned true, and F is the set of atoms assigned false.

1. $(\{b\}, \{a, c\})$, $(\{c\}, \{a, b\})$, $(\{b, c\}, \{a\})$, and $(\emptyset, \{a, b, c\})$: All cases where "a" is false. The program is not satisfied because the first rule "a"'s body is satisfied (it is empty), but "a" (from the head of the rule) is not true.

2. $(\{a\}, \{b, c\})$: This is an answer set. If we try to make "a" false, then the program is not satisfied (See 1).

3. $(\{a, b\}, \{c\})$: This is not an answer set because it is not minimal (See 2, we can make b false).

4. $(\{a, b, c\}, \emptyset)$: This is not an answer set: We can make b and c false, and all the rules will be satisfied. Note that when we change "c" to be false, the negative body of the rule "a, b :- not c." does not become satisfied.

Another example:

```
a :- not b.
b :- not a.
```

1. $(\emptyset, \{a, b\})$: Not an answer set. Both rule bodies are satisfied therefore a and b must be true.

2. $(\{a, b\}, \emptyset)$: Not an answer set. We can make both a and b false and all rules are satisfied (Remember that previously unsatisfied negative bodies do not become satisfied during the minimality check)

3. $(\{a\}, \{b\})$: This is an answer set: if we make a false, the negative body of the rule "a :- not b" remains satisfied and requires "a" to be true.

4. $(\{b\}, \{a\})$: This is an answer set: same reason as above using the rule "b :- not a" instead.

Another example:

```
a, b.
b :- a.
```

1. $(\{a, b\}, \emptyset)$: Not an answer set. If we change "b" to be false, then all rules are satisfied.

2. $(\emptyset, \{a, b\})$: Not an answer set. The body of the rule ("a, b.") is satisfied (because it is empty). So either "a" or "b" (or both) must be true.

3. $(\{a\}, \{b\})$: Not an answer set, "b" must be true because of the rule "b :- a."

4. $(\{b\}, \{a\})$: An answer set.

Last example:

```
:- a.
a, b.
```

Here "a" cannot be true, because that would satisfy the body of the rule ":- a." and there is no way to satisfy the head of the rule.

References

[1] T. C. Przymusinski, "Stable semantics for disjunctive programs," *New Generation Computing*, vol. 9, no. 3, pp. 401–424, Aug. 1991.