Lecture 4: Further Primitives

Outline

1. Syntactic extensions
2. Choice rules
3. Cardinality rules
4. Weight rules
5. The `smodels` system

1. SYNTACTIC EXTENSIONS

- The expressiveness of normal programs can be enhanced by introducing new syntactic primitives to the language.
- Any proper definition of a syntactic extension must address
  1. how the syntax of programs is generalized, and
  2. how the extension is covered by the stable model semantics.
- A way to address the second item is to provide a suitable translation for removing the new syntax viewed as sugar.

Example. *Extended programs* are obtained from normal ones by the introduction of *classical negation*, denoted by "$\neg$", in addition to default negation, denoted by "$\sim$".

Extended Programs in a Nutshell

- A *literal* is either positive (an atom a) or negative ($\neg a$).
- A *default literal* is formed from an ordinary literal using default negation: a, $\neg a$, $\sim a$, and $\sim\neg a$.
- Atoms are partitioned in three categories: *true* (a and $\sim\neg a$), *false* ($\neg a$ and $\sim a$), and *undefined*/*unknown* ($\sim a$ and $\sim\neg a$).

Definition. An *extended program* P is a set of rules of the form

$$l \leftarrow l_1, \ldots, l_n, \sim l_{n+1}, \ldots, \sim l_{n+m}.$$

where l and $l_1,\ldots,l_{n+m}$ belong to the *literal base*

$$\mathrm{Lit}(P) = \mathrm{Hb}(P) \cup \{\neg a \mid a \in \mathrm{Hb}(P)\}.$$

Answer Sets

Definition. A consistent set of literals $L \subseteq \mathrm{Lit}(P)$ is an *answer set* of an extended program P iff L is the *least set of literals* closed under

$$P^L = \{l \leftarrow l_1, \ldots, l_n \mid l \leftarrow l_1, \ldots, l_n, \sim l_{n+1}, \ldots, \sim l_{n+m} \in P \text{ and } l_{n+1} \notin L, \ldots, l_{n+m} \notin L\}.$$

Example. Consider an extended program P having the following rules:

$$\begin{array}{ll} \text{Flies} \leftarrow \text{Bird}, \sim\neg\text{Flies}. & \text{Bird}. \\ \neg\text{Flies} \leftarrow \text{Penguin}. & \neg\text{Flies} \leftarrow \text{Oily}. \end{array}$$

The respective unique answer sets of P and $Q = P \cup \{\text{Oily}.\}$ are

$$\{\text{Bird}, \text{Flies}\} \text{ and } \{\text{Bird}, \text{Oily}, \neg\text{Flies}\}.$$

Desiderata for Compilation

- There is trade-off between two basic ways of treating syntactic extensions when an ASP system is implemented:
  1. The support for syntactic extensions is integrated directly to the search engine in order to boost the search of answer sets.
  2. Expressions that involve syntactic extensions are compiled away in order to simplify the design of the search engine.
- The feasibility of compilation depends much on the complexity of the transformation required to remove a particular extension.
- For instance, transformations that are *linear time* and *modular* (applicable rule-by-rule) provide a good basis for compilation.

Translating Extended Programs

An extended program is transformed into a normal one as follows:

1. A new atom $\overline{a}$ is introduced for each atom $a \in \mathrm{Hb}(P)$.
2. A constraint $f \leftarrow a, \overline{a}, \sim f$ is introduced for each atom $a \in \mathrm{Hb}(P)$. Here $f \notin \mathrm{Hb}(P)$ can be a joint new atom for all such rules.
3. Literals are translated according to $\mathrm{Tr_N}(a) = a$ and $\mathrm{Tr_N}(\neg a) = \overline{a}$.
4. An extended rule $l \leftarrow l_1, \ldots, l_n, \sim l_{n+1}, \ldots, \sim l_{n+m}$ is translated into $\mathrm{Tr_N}(l) \leftarrow \mathrm{Tr_N}(l_1), \ldots, \mathrm{Tr_N}(l_n), \sim\mathrm{Tr_N}(l_{n+1}), \ldots, \sim\mathrm{Tr_N}(l_{n+m})$.

Theorem. (Correctness of the transformation) A consistent set of literals $L \subseteq \mathrm{Lit}(P)$ is an answer set of an extended program P iff $\mathrm{Tr_N}(L) = \{\mathrm{Tr_N}(l) \mid l \in L\}$ is an answer set of $\mathrm{Tr_N}(P)$.

2. CHOICE RULES

- We concentrate on syntactic extensions to (propositional) normal programs next and abandon extended programs for a while.
- A *choice rule* is an expression of the form
  $$\{a_1, \ldots, a_h\} \leftarrow b_1, \ldots, b_n, \sim c_1, \ldots, \sim c_m.$$
  where each a_i, b_j, and c_k is an atom.
- Intuitively, if the rule body is satisfied, we can choose any subset of the atoms mentioned in the *head* $\{a_1, \ldots, a_h\}$ to be true.
- Given a model candidate $M \subseteq \mathrm{Hb}(P)$, a reduced rule $a \leftarrow b_1, \ldots, b_n$ is included in the reduced program P^M iff $a \in \{a_1, \ldots, a_h\}$, $M \models \sim c_1, \ldots, \sim c_m$, and $M \models a$.

Representing Choices

- As suggested by their name, choice rules lend themselves to expressing various kinds of *choices* involved in applications.
- However, the minimality of stable models is no longer guaranteed in the presence of choice rules.

Example. Program $P = \{\{a\} \leftarrow \sim b.\ \}$ has two stable models $M_1 = \emptyset$ and $M_2 = \{a\}$ so that $M_1 \subseteq M_2$. Note that $P^{M_1} = \emptyset$ and $P^{M_2} = \{a.\ \}$.

Example. In our preceding example, the choice of goodies is nicely expressed in terms of a choice rule $\{\text{Cake}, \text{Bun}, \text{Cookie}\}$.

For now, the exclusive choice between coffee and tea must be accompanied by constraints (written below without $\mathrm{F} \leftarrow \sim\mathrm{F}$):

$$\{\text{Coffee}, \text{Tea}\}. \quad \leftarrow \text{Coffee}, \text{Tea}. \quad \leftarrow \sim\text{Coffee}, \sim\text{Tea}.$$

Translating Choice Rules

Choice rules can be removed from a program P as follows:

1. A new atom $\overline{a}$ is introduced for each atom $a \in \mathrm{Head}(P)$, i.e., those having a *head occurrence* in some choice rule of P.
2. A choice rule $\{a_1,\ldots,a_h\} \leftarrow b_1,\ldots,b_n,\sim c_1,\ldots,\sim c_m$ can be translated into $2h+1$ rules

$$a_1 \leftarrow b, \sim\overline{a_1}. \quad \ldots \quad a_h \leftarrow b, \sim\overline{a_h}.$$
$$\overline{a_1} \leftarrow \sim a_1. \quad \ldots \quad \overline{a_h} \leftarrow \sim a_h.$$
$$b \leftarrow b_1,\ldots,b_n,\sim c_1,\ldots,\sim c_m$$

where $b \in \mathrm{Hb}(P)$ is a new atom specific to this rule.

Theorem. An interpretation $M \subseteq \mathrm{Hb}(P)$ is a stable model of a program P iff $M \cup \{\overline{a} \mid a \in \mathrm{Head}(P) \setminus M\}$ is a stable model of $\mathrm{Tr}_{\mathrm{N}}(P)$.

3. CARDINALITY RULES

- A *default literal* is either an atom a or its default negation $\sim a$.
- A *cardinality rule* is an expression of the form
  $$a \leftarrow l\,\{b_1,\ldots,b_n,\sim c_1,\ldots,\sim c_m\}.$$
  where a, each b_j, and each c_k is an atom.
- The idea behind the rule is that if the number of satisfied default literals in the rule body is at least l, then the head a is true.
- Thus l acts as a *lower bound* in the rule.

Example. In our delicacy example, having at least one of the goodies can be formalized succinctly by a cardinality rule

$$\text{Some} \leftarrow 1\,\{\text{Cake}, \text{Bun}, \text{Cookie}\}.$$

Semantics of Cardinality Rules

- Given a model candidate $M \subseteq \mathrm{Hb}(P)$, the reduct P^M contains
  $$a \leftarrow l'\,\{b_1,\ldots,b_n\}$$
  with a revised lower bound $l' = \max(0, l - |\{c_1,\ldots,c_m\} \setminus M|)$.
- However, such rules are not encountered in positive programs.
- A positive cardinality rule $a \leftarrow l\,\{b_1,\ldots,b_n\}$ in a program P is satisfied in an interpretation $I \subseteq \mathrm{Hb}(P)$ iff
  $$l \leq |\{b_i \mid M \models b_i\}| \text{ implies } M \models a.$$
- Previous results about least models generalize for this class of programs, i.e., programs with *positive* cardinality rules.

Definition. An interpretation $M \subseteq \mathrm{Hb}(P)$ is a stable model of a normal program P with cardinality rules iff $M = \mathrm{LM}(P^M)$.

Making Choices of Specific Cardinality

- It is easy to incorporate *upper bounds* into cardinality rules: a rule of the form $a \leftarrow l\,\{b_1,\ldots,b_n,\sim c_1,\ldots,\sim c_m\}\,u$ stands for
  $$b \leftarrow l\,\{b_1,\ldots,b_n,\sim c_1,\ldots,\sim c_m\}.$$
  $$c \leftarrow u+1\,\{b_1,\ldots,b_n,\sim c_1,\ldots,\sim c_m\}.$$
  $$a \leftarrow b, \sim c.$$
- The meaning of a choice $l\,\{a_1,\ldots,a_h\}\,u \leftarrow b_1,\ldots,b_n,\sim c_1,\ldots,\sim c_m$ with lower and upper bounds l and u is given by
  $$b \leftarrow b_1,\ldots,b_n,\sim c_1,\ldots,\sim c_m.$$
  $$\{a_1,\ldots,a_h\} \leftarrow b. \quad c \leftarrow l\,\{a_1,\ldots,a_h\}\,u.$$
  $$\leftarrow b, \sim c.$$

Examples. $1\,\{\text{Coffee}, \text{Tea}\}\,1$. $\quad 1\,\{\text{Cake}, \text{Bun}, \text{Cookie}\}\,2$.

Translations Back and Forth

- A normal rule $a \leftarrow b_1, \ldots, b_n, \sim c_1, \ldots, \sim c_m$ is equivalent to a cardinality rule $a \leftarrow n+m\,\{b_1, \ldots, b_n, \sim c_1, \ldots, \sim c_m\}$.
- A cardinality rule $a \leftarrow l\,\{d_1, \ldots, d_n\}$ where $d_1, \ldots, d_n$ are default literals and $l \leq n$ can be rewritten as a set of normal rules:
  1. A condition $n\,\{d_1, \ldots, d_n\}$ is replaced by $d_1, \ldots, d_n$.
  2. A condition $0\,\{d_1, \ldots, d_n\}$ is dropped altogether.
  3. If $d_1 = b$, the cardinality rule is replaced by $a \leftarrow b, l-1\,\{d_2, \ldots, d_n\}$ and $a \leftarrow \sim b, l\,\{d_2, \ldots, d_n\}$.
  4. If $d_1 = \sim c$, the cardinality rule is replaced by $a \leftarrow \sim c, l-1\,\{d_2, \ldots, d_n\}$; $a \leftarrow \sim \overline{c}, l\,\{d_2, \ldots, d_n\}$, and $\overline{c} \leftarrow \sim c$.

☞ An exponential translation results in the worst case.

4. WEIGHT RULES

- A *weight rule* is an expression of the form

$$a \leftarrow l\,[b_1 = w_1, \ldots, b_n = w_n, \sim c_1 = v_1, \ldots, \sim c_m = v_m].$$

  where $w_1, \ldots, w_n$ and $v_1, \ldots, v_m$ are *weights* (natural numbers) associated with the respective default literals in the rule body.
- The number l acts as a *lower bound* for a sum of weights

$$\mathrm{WS}_M(b_1 = w_1, \ldots, b_n = w_n, \sim c_1 = v_1, \ldots, \sim c_m = v_m) = \textstyle\sum_{M \models b_i} w_i + \sum_{M \models \sim c_j} v_j$$

  that can be evaluated with respect to any interpretation M.
- Intuitively, the head a must be true if the sum of weights associated with satisfied default literals is at least l.

Semantics of Weight Rules

- Given a model candidate $M \subseteq \mathrm{Hb}(P)$, the reduct P^M contains

$$a \leftarrow l'\,[b_1 = w_1, \ldots, b_n = w_n]$$

  with a revised lower bound

$$l' = \max(0, l - \mathrm{WS}_M(\sim c_1 = v_1, \ldots, \sim c_m = v_m)).$$

- A positive weight rule $a \leftarrow l\,[b_1 = w_1, \ldots, b_n = w_n]$ is satisfied in an interpretation $I \subseteq \mathrm{Hb}(P)$ iff

$$l \leq \mathrm{WS}_M(b_1 = w_1, \ldots, b_n = w_n) \text{ implies } M \models a.$$

- Stable models, i.e., answer sets, generalize for weight programs in analogy to cardinality programs.

Modelling with Weight Rules

- Many applications involve numerical measures such as *prices*, *capacities*, etc. that have to be limited in a way or another.
- Weight rules provide a flexible way to formalize such limits.
- The unit and assignment of weights can vary from rule to rule.

Example. Consider the following program containing a weight rule:

$\{\text{Tea}, \text{Espresso}, \text{Cappucino}\}.\quad \{\text{Cake}, \text{Bun}, \text{Cookie}\}.\quad \{\text{TakeAway}\}$

$\text{Broke} \leftarrow 6\,[\text{Tea} = 1, \text{Espresso} = 2, \text{Cappucino} = 3,$
$\quad \text{Cake} = 3, \text{Bun} = 2, \text{Cookie} = 1, \sim\text{TakeAway} = 1].$

$\text{F} \leftarrow \text{Broke}, \sim\text{F}.$

$\Longrightarrow$ For instance, interpretations $M = \{\text{Espresso}, \text{Bun}\}$ and $N = \{\text{Espresso}, \text{Cake}, \text{TakeAway}\}$ are stable.

5. THE SMODELS SYSTEM

- The `smodels` system is an implementation of ASP based on normal rules, cardinality rules, and weight rules.
- The system consists of two main components: the grounder `lparse` (v. 1.0.17) and the search engine `smodels` (v. 2.32).

$$P \Rightarrow \boxed{\texttt{lparse}} \Rightarrow \mathrm{Gnd}(P) \Rightarrow \boxed{\texttt{smodels}} \Rightarrow M_1,\ M_2,\ \ldots$$

- In addition to removing variables, the front-end `lparse` is responsible for *partial evaluation* and *simplification* tasks.
- In UNIX-like environments, the system is run as a pipeline

  `lparse program.lp | smodels 1`

  where `1` is the number of models to be computed (`0` means all).

Internal Representation

- The output of lparse is based on a simplified language which provides programs with an internal (numeric) representation.
- Such an intermediate format enables the development of other ASP systems parallel to the `smodels` system.
- There are tools to handle programs in this format such as `lplist` (symbolic representation) and `len` (count atoms, rules, and length).

```
1 2 1 1 3
0
2 a
3 b
0 B+
0
B-
1
0
1
```

Example. The lines shown above on the right are produced by the command line "`echo 'a:-not b.' | lparse -dall`".

Cardinality/Weight Constraint Programs

- The forms of choice, cardinality, and weight rules introduced so far correspond to those used in the internal representation.
- In fact, the grounder of the system admits a more general syntax

$$l_0\, E_0\, u_0 \leftarrow l_1\, E_1\, u_1,\ \ldots,\ l_n\, E_n\, u_n.$$

  where each E_i is a cardinality/weight expression as above.
- The semantics of such rules can be understood from a translation

$$\{A_0\} \leftarrow b. \quad f \leftarrow b, \sim b_0, \sim f. \quad f \leftarrow b, c_0, \sim f.$$

$$b \leftarrow b_1, \ldots, b_n, \sim c_1, \ldots, \sim c_n.$$

$$b_i \leftarrow l_i E_i. \quad c_i \leftarrow (u_i + 1)E_i. \quad (\text{for } 0 \leq i \leq n)$$

  where A_0 is the set of positive default literals in E_0.

Guiding the Search of Answer Sets

- *Compute statements* allow the selection of answer sets to be computed by the `smodels` system:

$$\text{compute } \{b_1, \ldots, b_n, \sim c_1, \ldots, \sim c_m\}.$$

- It is also possible to *optimize* answer sets using optimization statements that resemble weight rules for default literals $d_1, \ldots, d_n$:

$$\text{minimize } \{d_1 = w_1, \ldots, d_n = w_n\}.$$

$$\text{maximize } \{d_1 = w_1, \ldots, d_n = w_n\}.$$

- The goal is to minimize/maximize the respective weight sum.
- If several optimization statements are specified, they are interpreted lexicographically (the first is most significant).

Syntactic Extensions in `lparse`

The front-end `lparse` has features that support concise encodings:

1. Range specifications like `node(1..10)` are allowed.
2. Several instances of the same predicate can be merged into one such as `queen(1,6; 2,3; 3,7; 4,4; 5,1; 6,8; 7,2; 8,5)`.
3. *Literal sets* are used to condense choices and rule bodies:
   `1 { in(X,Y):edge(X,Y) } 1 :- node(X).`
4. Values of constants can be assigned using option `-c`.
5. Classical negation is enabled with option flag `--true-negation`.

Consult the user's manual for *arithmetical operations* and more!

SuDoku Puzzles Revisited

Example. Even the very short program that solved SuDoku puzzles can be condensed using the special features of `lparse`:

```
number(1..9).
border(1;4;7).
region(X,Y) :- border(X;Y).

1 { value(X,Y,N):number(X;Y):X1<=X:X<=X1+2:Y1<=Y:Y<=Y1+2 } 1
:- number(N), region(X1,Y1).

:- 2 {value(X,Y,N):number(N)}, number(X;Y).
:- 2 {value(X,Y,N):number(Y)}, number(N;X).
:- 2 {value(X,Y,N):number(X)}, number(N;Y).
```


OBJECTIVES

- You know a number of syntactic extensions to normal programs and understand their semantics intuitively as well as by definition and/or via syntactic transformations.
- You are able to check/calculate stable models for simple programs involving choice rules, cardinality rules, and weight rules.
- You are able to formalize simple constraint programming problems using the language supported by the front-end `lparse`.
- You have tried out the `smodels` system in practise.

TIME TO PONDER

A translation from cardinality rules into normal rules was presented above (see slide 13).

- Can you think of a more succinct transformation for this purpose?
- What kind arguments are needed to prove your approach correct?