Lecture 10: Modularity Aspects

Outline

- Stratification
- Module architecture for ASP
- Compositional semantics
- Modularizing weak equivalence

1. STRATIFICATION

- The number of stable models varies from program to program.
- This is quite natural given the modelling philosophy of ASP: a strict correspondence of answer sets and solutions is sought for.
- The semantics of a positive program P is uniquely determined by the least model $\mathrm{LM}(P)$. Likewise, the well-founded model $\mathrm{WFM}(P)$ assigns a unique set of literals with a normal program P.
- These observations raise the question whether the existence of a unique stable model can be guaranteed under any circumstances.
- This is a property of *stratified programs* to be explored next.

Dependency Graphs

Definition. The *dependency graph* $\mathrm{DG(P)}$ of an `smodels` program P is $\langle \mathrm{Hb}(P), \leq_1 \rangle$ where $b \leq_1 a$ holds for $a, b \in \mathrm{Hb}(P)$ if and only if (i)

1. there is a basic rule $a \leftarrow B, {\sim}C \in P$,
2. there is a choice rule $\{A\} \leftarrow B, {\sim}C \in P$ such that $a \in A$,
3. there is a cardinality rule $a \leftarrow l\,\{B, {\sim}C\} \in P$, or
4. there is a weight rule $a \leftarrow l\,[B = w_B, {\sim}C = v_C] \in P$,

and $b \in B \cup C$, *or*

(ii) $b = a$ and $a \in A$ for some choice rule $\{A\} \leftarrow B, {\sim}C \in P$.

Remark. The *positive* dependency graph $\mathrm{DG^+(P)}$ of P is defined analogously but using only positive dependencies.

Strongly Connected Components

- The overall *dependency relation* $\leq$ $(\subseteq \mathrm{Hb}(P)^2)$ is the reflexive and transitive closure $(\leq_1)^*$ of the immediate dependency relation $\leq_1$.
- Thus $a \leq b$ holds if and only if there is a sequence $a_1, \ldots, a_n$ of atoms from $\mathrm{Hb}(P)$ such that $n > 0$ and $a = a_1 \leq_1 \ldots \leq_1 a_n = b$.

Definition. A *strongly connected component* (SCC) of a dependency graph $\mathrm{DG(P)} = \langle \mathrm{Hb}(P), \leq_1 \rangle$ is a *maximal* subset S of $\mathrm{Hb}(P)$ such that $a \leq b$ and $b \leq a$ for every $a, b \in S$.

Example. The dependency graph $\mathrm{DG(P)}$ of the `smodels` program

$$
\begin{array}{lll}
a \leftarrow b. & b \leftarrow c. & c \leftarrow a. \\
\{a,b,c\} \leftarrow d, {\sim}e. & d \leftarrow {\sim}e. & e \leftarrow {\sim}d.
\end{array}
$$

has strongly connected components $S_1 = \{a,b,c\}$ and $S_2 = \{d,e\}$.

Stratified Programs

- The strongly connected components of $\mathrm{DG}(\mathrm{P})$ determine sets of atoms which are *recursively defined* in terms of the rules of P.
- A dependency $c \leq_1 a$ in $\mathrm{DG}(\mathrm{P})$ is *negative* iff $\sim c$ appears in a negative body $\sim C$, or $c = a$ appears in the head of a choice rule.

Definition. A program P is *stratified* iff the strongly connected components of $\mathrm{DG}(\mathrm{P})$ do not involve negative dependencies.

Proposition. A stratified `smodels` program P has a unique stable model M such that $M = \mathrm{WFM}(P) \cap \mathrm{Hb}(P)$.

Remark. The stratifiability of a program can be decided in linear time because the strongly connected components of $\mathrm{DG}(\mathrm{P})$ can be computed in time linear with respect to $||P||$ (Tarjan's algorithm).

Example

Consider a normal program P consisting of the following rules:

$$a \leftarrow b, \sim c. \qquad b \leftarrow a, \sim d. \qquad c \leftarrow \sim e.$$
$$d \leftarrow e. \qquad e \leftarrow a.$$

1. The program is not stratified because $\mathrm{DG}(\mathrm{P})$ has a single SCC $S = \mathrm{Hb}(P) = \{a,b,c,d,e\}$ involving negative dependencies.
2. If the last rule is dropped, the resulting program P' is stratified because $\mathrm{DG}(\mathrm{P}')$ has SCCs $S_1 = \{e\}$, $S_2 = \{d\}$, $S_3 = \{c\}$, and $S_4 = \{a,b\}$—not involving negative dependencies.

Remark. The computation of the unique $M = \{c\} \in \mathrm{SM}(P')$ can be done in a *modular* fashion using an order of SCCs which is compatible with $\mathrm{DG}(\mathrm{P})$—such as S_1, S_2, S_3, S_4: $\sim e$, $\sim d$, c, $\sim a$, $\sim b$.

2. MODULE ARCHITECTURE FOR ASP

- Modular program development has a number of advantages:
  1. It enforces a good programming style by giving extra structure for programs (sets of rules in ASP).
  2. The semantics of programs is easier to grasp and potentially complex details can be hidden inside modules.
  3. The task of programming is naturally divided into subtasks that can be delegated for a team of programmers.
- In the sequel, a module architecture originally proposed for PROLOG programs [Gaifman and Shapiro, 1988], is tailored to the case of `smodels` programs under stable model semantics.

Modules for `smodels` Programs

- Given a set of rules R, we write $\mathrm{Hb}(R)$ and $\mathrm{Head}(R)$ for the sets of atoms that appear in R and in the heads of rules of R, respectively.

Definition. A *program module* $\mathbb{P}$ is a quadruple $\langle R, I, O, H \rangle$ where

1. I, O, and H are distinct sets *input*, *output*, and *hidden* atoms, respectively, and
2. R is a set of rules such that

$$\mathrm{Hb}(R) \subseteq \mathrm{Hb}(\mathbb{P}) = I \cup O \cup H, \text{ and } \mathrm{Head}(R) \cap I = \emptyset.$$

Example. Verify these requirements for an `smodels` program module

$$\mathbb{P} = \langle \{a \leftarrow \sim b.\ \ b \leftarrow \sim a, \sim c.\ \ \}, \{c\}, \{a\}, \{b\} \rangle.$$

Hebrand Bases and Interpretations

- The Herbrand base $\mathrm{Hb}(\mathbb{P})$ of $\mathbb{P} = \langle R, I, O, H \rangle$ partitions into
  1. $\mathrm{Hb}_{\mathrm{i}}(\mathbb{P}) = I$ (input atoms),
  2. $\mathrm{Hb}_{\mathrm{o}}(\mathbb{P}) = O$ (output atoms),
  3. $\mathrm{Hb}_{\mathrm{v}}(\mathbb{P}) = I \cup O$ (*visible* atoms), and
  4. $\mathrm{Hb}_{\mathrm{h}}(\mathbb{P}) = H$ (hidden atoms).
- An *interpretation* $M \subseteq \mathrm{Hb}(\mathbb{P})$, which determines the true atoms of $\mathrm{Hb}(\mathbb{P})$, has analogous projections with respect to these sets:

$$M_{\mathrm{i}},\ M_{\mathrm{o}},\ M_{\mathrm{v}} = M_{\mathrm{i}} \cup M_{\mathrm{o}}, \text{ and } M_{\mathrm{h}}.$$

- The idea is that the visible part is accessible by other modules.

Example: Graph Colouring

A program module for 3-colouring a graph having at most n nodes:

O_n:	$\{\mathsf{r}(x), \mathsf{g}(x), \mathsf{b}(x) \mid 1 \le x \le n\}$
R_n:	$\{\{\mathsf{r}(x), \mathsf{g}(x), \mathsf{b}(x)\} \leftarrow \mathsf{node}(x).\ \mid 1 \le x \le n\} \cup$
	$\{\mathsf{f} \leftarrow \mathsf{node}(x), \sim\mathsf{r}(x), \sim\mathsf{g}(x), \sim\mathsf{b}(x), \sim\mathsf{f}.\ \mid 1 \le x \le n\} \cup$
	$\{\mathsf{node}(x) \leftarrow \mathsf{edge}(x,y).\ \mid 1 \le x \le n\} \cup$
	$\{\mathsf{node}(y) \leftarrow \mathsf{edge}(x,y).\ \mid 1 \le x \le n\} \cup$
	$\{\mathsf{f} \leftarrow \mathsf{edge}(x,y), \mathsf{r}(x), \mathsf{r}(y), \sim\mathsf{f}.\ \mid 1 \le x < y \le n\} \cup$
	$\{\mathsf{f} \leftarrow \mathsf{edge}(x,y), \mathsf{g}(x), \mathsf{g}(y), \sim\mathsf{f}.\ \mid 1 \le x < y \le n\} \cup$
	$\{\mathsf{f} \leftarrow \mathsf{edge}(x,y), \mathsf{b}(x), \mathsf{b}(y), \sim\mathsf{f}.\ \mid 1 \le x < y \le n\}$
I_n :	$\{\mathsf{edge}(x,y) \mid 1 \le x < y \le n\}$

$\Longrightarrow$ Atoms in $H_n = \{\mathsf{f}\} \cup \{\mathsf{node}(x) \mid 1 \le x \le n\}$ are hidden.

Interpreting Input Atoms within the Reduct

Definition. For a module $\mathbb{P} = \langle R, I, O, H \rangle$ and an interpretation $M \subseteq \mathrm{Hb}(P)$ determining the input M_{i} for $\mathbb{P}$, the reduct $R^{M,I}$ contains:

1. For each basic rule $a \leftarrow B, \sim C \in R$ satisfying $M \models (B \cap I) \cup \sim C$, the reduced rule $a \leftarrow (B \setminus I)$.
2. For each choice rule $\{A\} \leftarrow B, \sim C \in R$ satisfying $M \models (B \cap I) \cup \sim C$ and for each head atom $a \in A \cap M$, the rule $a \leftarrow (B \setminus I)$.
3. For each cardinality rule $a \leftarrow l\,\{B, \sim C\} \in R$, the reduced rule $a \leftarrow l'\,\{(B \setminus I)\}$ with $l' = \min(0, l - |B \cap I \cap M| - |C \setminus M|)$.
4. For each weight rule $a \leftarrow l\,[B = w_B, \sim C = v_C] \in R$, the reduced rule $a \leftarrow l'\,[(B \setminus I) = w_{(B \setminus I)}]$ with

$$l' = \min(0, l - \textstyle\sum_{b \in B \cap I \cap M} w_b - \sum_{c \in (C \setminus M)} v_c).$$

Stable Semantics for Program Modules

Definition. An interpretation $M \subseteq \mathrm{Hb}(\mathbb{P})$ is a *stable model* of a program module $\mathbb{P} = \langle R, I, O, H \rangle$ having an input interface $\mathrm{Hb}_{\mathrm{i}}(\mathbb{P})$ iff

$$M \setminus I = \mathrm{LM}(R^{M,I}).$$

Example. Verify the set of stable models

$$\mathrm{SM}(\mathbb{P}) = \{\emptyset,\ \{a\},\ \{b\},\ \{a,b\},\ \{a,c\},\ \{b,c\}\}$$

for the `smodels` program module $\mathbb{P}$ illustrated below:

$\{a,b\}$
$\{a,b\} \leftarrow \sim c. \quad a \leftarrow c, \sim b. \quad b \leftarrow c, \sim a.$
$\{c\}$

3. COMPOSITIONAL SEMANTICS

- The principle of *compositionality*: the semantics of an entire theory should be a function of the semantics of its components.
- This is true for classical propositional theories:

$$\mathrm{CM}(T_1 \cup T_2) = \mathrm{CM}(T_1) \bowtie \mathrm{CM}(T_2)$$

where $\mathrm{CM}(T) = \{M \subseteq \mathrm{Hb}(P) \mid M \models T\}$ and the operator $\bowtie$ which combines *compatible* models will be defined next.

- Unfortunately, logic programs under stable model semantics do not have an analogous property for arbitrary *unions* of programs.
- Thus more attention has to be paid to circumstances under which programs, or modules introduced so far, can be joined together.

Compatibility of Models

Consider two propositional theories T_1 and T_2:

- We say that interpretations $M_1 \subseteq \mathrm{Hb}(T_1)$ and $M_2 \subseteq \mathrm{Hb}(T_2)$ are *compatible* if and only if $M_1 \cap \mathrm{Hb}(T_2) = M_2 \cap \mathrm{Hb}(T_1)$.
- If M_1 and M_2 are compatible, then $M_1 = (M_1 \cup M_2) \cap \mathrm{Hb}(T_1)$ and symmetrically $M_2 = (M_1 \cup M_2) \cap \mathrm{Hb}(T_2)$.

Definition. Given sets of interpretations $A_1 \subseteq \mathbf{2}^{\mathrm{Hb}(T_1)}$ and $A_2 \subseteq \mathbf{2}^{\mathrm{Hb}(T_2)}$ for propositional theories T_1 and T_2, the *natural join* of A_1 and A_2 is

$$A_1 \bowtie A_2 = \{M_1 \cup M_2 \mid M_1 \in A_1,\ M_2 \in A_2 \text{ and } M_1 \cap \mathrm{Hb}(T_2) = M_2 \cap \mathrm{Hb}(T_1)\}.$$

Composing Programs from Modules

- We say that $\mathbb{P}_1 = \langle R_1, I_1, O_1, H_1 \rangle$ and $\mathbb{P}_2 = \langle R_2, I_2, O_2, H_2 \rangle$ *respect the module interface* of each other if and only if

$$(I_1 \cup O_1 \cup H_1) \cap H_2 = \emptyset,\ (I_2 \cup O_2 \cup H_2) \cap H_1 = \emptyset, \text{ and } O_1 \cap O_2 = \emptyset.$$

Definition. The *composition* of program modules $\mathbb{P}_1 = \langle R_1, I_1, O_1, H_1 \rangle$ and $\mathbb{P}_2 = \langle R_2, I_2, O_2, H_2 \rangle$ respecting module interfaces of each other is

$$\mathbb{P}_1 \oplus \mathbb{P}_2 = \langle R_1 \cup R_2, (I_1 \setminus O_2) \cup (I_2 \setminus O_1), O_1 \cup O_2, H_1 \cup H_2 \rangle.$$

Example. Verify interface conditions for the following composition:

$\{a\}$		$\{b\}$		$\{a,b\}$
$a \leftarrow c.\quad c \leftarrow {\sim}b.$	$\oplus$	$b \leftarrow {\sim}a.$	$=$	$a \leftarrow c.\quad c \leftarrow {\sim}b.\quad b \leftarrow {\sim}a.$
$\{b\}$		$\{a\}$		$\emptyset$

Counter-Example for $\oplus$

- The interface conditions involved in the definition $\oplus$ are not sufficient to guarantee the compositionality of stable semantics.

Example. Let us analyze the composition illustrated below

$\{a\}$		$\{b\}$		$\{a,b\}$
$a \leftarrow b.$	$\oplus$	$b \leftarrow a.$	$=$	$a \leftarrow b.\quad b \leftarrow a.$
$\{b\}$		$\{a\}$		$\emptyset$

in more detail. Now, we have $\mathrm{SM}(\mathbb{P}_1) = \{\emptyset, \{a,b\}\} = \mathrm{SM}(\mathbb{P}_2)$ but $\mathrm{SM}(\mathbb{P}_1 \oplus \mathbb{P}_2) = \{\emptyset\}$ differs from $\mathrm{SM}(\mathbb{P}_1) \bowtie \mathrm{SM}(\mathbb{P}_2) = \{\emptyset, \{a,b\}\}$.

Joins of Program Modules

- In the preceding example, the key issue is that a and b are positively interdependent and hence false in the least model.
- The compositionality of stable semantics is achieved if the creation of such dependencies is pre-empted in program composition.

Definition. Modules $\mathbb{P}_1$ and $\mathbb{P}_2$, for which $\mathbb{P}_1 \oplus \mathbb{P}_2$ is defined, are *mutually dependent* if there is an SCC S in $\mathrm{DG}^+(\mathbb{P}_1 \oplus \mathbb{P}_2)$ such that

$$S \cap \mathrm{Hb}_o(\mathbb{P}_1) \neq \emptyset \text{ and } S \cap \mathrm{Hb}_o(\mathbb{P}_2) \neq \emptyset.$$

If there is no such S, we say that the *join* $\mathbb{P}_1 \sqcup \mathbb{P}_2 = \mathbb{P}_1 \oplus \mathbb{P}_2$ is defined.

Example. In the preceding example, the join is not defined because of the strongly connected component $S = \{a, b\}$ involved in the positive dependency graph $\mathrm{DG}^+(\{\mathrm{a} \leftarrow \mathrm{b}.\ \ \mathrm{b} \leftarrow \mathrm{a}.\ \ \})$.

Module Theorem

Theorem. Let $\mathbb{P}_1$ and $\mathbb{P}_2$ be two program modules such that $\mathbb{P}_1 \sqcup \mathbb{P}_2$ is defined. Then $\mathrm{SM}(\mathbb{P}_1 \sqcup \mathbb{P}_2) = \mathrm{SM}(\mathbb{P}_1) \bowtie \mathrm{SM}(\mathbb{P}_2)$.

Example. Consider the following composition of modules:

$$\begin{array}{|c|}\hline \{a\} \\ \hline a \leftarrow {\sim}b. \\ a \leftarrow b. \\ \hline \{b\} \\ \hline \end{array} \oplus \begin{array}{|c|}\hline \{b\} \\ \hline b \leftarrow {\sim}a. \\ \hline \{a\} \\ \hline \end{array} = \begin{array}{|c|}\hline \{a, b\} \\ \hline a \leftarrow {\sim}b. \quad a \leftarrow b. \\ b \leftarrow {\sim}a. \\ \hline \emptyset \\ \hline \end{array}$$

1. The SCCs of $\mathrm{DG}^+(\mathbb{P}_1 \oplus \mathbb{P}_2)$ are $S_1 = \{a\}$ and $S_2 = \{b\}$.
2. The sets $\mathrm{SM}(\mathbb{P}_1) = \{\{a\}, \{a, b\}\}$ and $\mathrm{SM}(\mathbb{P}_2) = \{\{a\}, \{b\}\}$.
3. Thus $\mathrm{SM}(\mathbb{P}_1) \bowtie \mathrm{SM}(\mathbb{P}_2) = \{\{a\}\} = \mathrm{SM}(\mathbb{P}_1 \sqcup \mathbb{P}_2)$.

Computing Stable Models for Module

- The definition of stable models for a program module $\mathbb{P}$ covers all interpretations $M_i \subseteq \mathrm{Hb}(\mathbb{P})$.
- The context of $\mathbb{P}$ determines which of them come into effect.
- The set of stable models $\mathrm{SM}(\mathbb{P})$ can be computed by attaching $\mathbb{P}$ to a general context that creates all input interpretations for $\mathbb{P}$.

Proposition. Let $\mathbb{P} = \langle R, I, O, H \rangle$ be a program module and $\mathbb{G}_I = \langle \{\{I\}.\ \}, \emptyset, I, \emptyset \rangle$ the respective *input generator*. Then

$$\mathrm{SM}(\mathbb{P}) = \mathrm{SM}(\mathbb{P} \sqcup \mathbb{G}_I).$$

Example. In an earlier example, the set of stable models $\mathrm{SM}(\mathbb{P}) = \mathrm{SM}(\mathbb{P} \sqcup \mathbb{G}_{\{c\}})$ is essentially generated by the set of rules $\{\{a, b\} \leftarrow {\sim}c.\ \ a \leftarrow c, {\sim}b.\ \ b \leftarrow c, {\sim}a.\ \ \{c\}.\ \ \}$ having no input atoms.

4. MODULARIZING WEAK EQUIVALENCE

- The computation of $\mathrm{SM}(\mathbb{P})$ for a module $\mathbb{P}$ is based on an input generator $\mathbb{G}_I$ that acts as the most general context for $\mathbb{P}$.
- The equivalence of modules can be addressed in the same way:

  Do $\mathbb{P}$ and $\mathbb{Q}$ have the same stable models in all possible contexts?
- The role of hidden atoms must be addressed at this point.
- The notion of *visible equivalence* stems from the modelling philosophy of ASP as well as the user's perspective:
  1. The number of stable models—that correspond to the solutions of the problem—should be the same.
  2. The visible parts of stable models—as observed by the user of an answer set solver—should be the same.

Visible/Modular Equivalence

Definition. The *visible* and *modular* equivalence of program modules $\mathbb{P}$ and $\mathbb{Q}$, denoted by $\mathbb{P} \equiv_{\mathrm{v}} \mathbb{Q}$ and $\mathbb{P} \equiv_{\mathrm{m}} \mathbb{Q}$, are defined as follows:

1. $\mathbb{P} \equiv_{\mathrm{v}} \mathbb{Q}$ if and only if $\mathrm{Hb}_{\mathrm{v}}(\mathbb{P}) = \mathrm{Hb}_{\mathrm{v}}(\mathbb{Q})$ and there is a bijection $f : \mathrm{SM}(\mathbb{P}) \rightarrow \mathrm{SM}(\mathbb{Q})$ such that for all $M \in \mathrm{SM}(\mathbb{P})$,
$$M \cap \mathrm{Hb}_{\mathrm{v}}(\mathbb{P}) = f(M) \cap \mathrm{Hb}_{\mathrm{v}}(\mathbb{Q}).$$
2. $\mathbb{P} \equiv_{\mathrm{m}} \mathbb{Q}$ if and only if $\mathrm{Hb}_{\mathrm{i}}(\mathbb{P}) = \mathrm{Hb}_{\mathrm{i}}(\mathbb{Q})$ and $\mathbb{P} \equiv_{\mathrm{v}} \mathbb{Q}$.

Theorem. Let $\mathbb{P}, \mathbb{Q}$, and $\mathbb{R}$ be program modules such that $\mathbb{P} \sqcup \mathbb{R}$ and $\mathbb{Q} \sqcup \mathbb{R}$ are defined. If $\mathbb{P} \equiv_{\mathrm{m}} \mathbb{Q}$, then $\mathbb{P} \sqcup \mathbb{R} \equiv_{\mathrm{m}} \mathbb{Q} \sqcup \mathbb{R}$.

Remark. The converse does not hold in general, i.e., $\mathbb{P} \sqcup \mathbb{R} \equiv_{\mathrm{m}} \mathbb{Q} \sqcup \mathbb{R}$ (equivalence in a specific context $\mathbb{R}$) might well not imply $\mathbb{P} \equiv_{\mathrm{m}} \mathbb{Q}$.

Example

Module $\mathbb{P}$:

$\{a, b\}$
$\{a\} \leftarrow c.$ $\{b\} \leftarrow \sim c.$
$\{c\}$

$\equiv_{\mathrm{m}}$

Module $\mathbb{Q}$:

$\{a, b\}$
$a \leftarrow c, \sim d. \quad d \leftarrow c, \sim a.$ $b \leftarrow \sim c, \sim e. \quad e \leftarrow \sim c, \sim b.$
$\{c\}$

The modular equivalence of the modules $\mathbb{P}$ and $\mathbb{Q}$ illustrated above is based on the following correspondence of stable models mediated by f:

$$\begin{array}{rcccc} \mathrm{SM}(\mathbb{P}): & \{c\} & \{a,c\} & \{\} & \{b\} \\ f: & \downarrow & \downarrow & \downarrow & \downarrow \\ \mathrm{SM}(\mathbb{Q}): & \{d,c\} & \{a,c\} & \{e\} & \{b\} \end{array}$$

Modules having Enough Visible Atoms

- In the worst case, the verification of $\equiv_{\mathrm{v}}$ and $\equiv_{\mathrm{m}}$ can be highly complex (a counting problem is involved in general).
- Hidden atoms tend to increase the complexity of the problem.

Definition. The *hidden part* of a module $\mathbb{P} = \langle R, I, O, H \rangle$ is $\mathbb{P}_{\mathrm{h}} = \langle R_{\mathrm{h}}, I \cup O, H, \emptyset \rangle$ where R_{h} contains rules of R defining atoms in H (the heads of rules are projected with respect to H).

Definition. A program module $\mathbb{P} = \langle R, I, O, H \rangle$ has *enough visible atoms* if and only if for each $N \subseteq \mathrm{Hb}_{\mathrm{v}}(\mathbb{P}) = I \cup O$, $\mathrm{SM}(\mathbb{P}_{\mathrm{h}}) = \{M\}$ where $M \cap (I \cup O) = N$.

Remark. If $\mathrm{Hb}_{\mathrm{h}}(\mathbb{P}) = \emptyset$, then the module $\mathbb{P}$ has enough visible atoms.

Translation-Based Verification

- The translation-based method for the verification of weak equivalence $P \equiv Q$ can be generalized for modules.
- The relation $\mathbb{P} \equiv_{\mathrm{m}} \mathbb{Q}$ coincides with $R_P \equiv R_Q$ for the respective rule sets, if $\mathrm{Hb}_{\mathrm{i}}(\mathbb{P}) = \mathrm{Hb}_{\mathrm{i}}(\mathbb{Q}) = \emptyset$ and $\mathrm{Hb}_{\mathrm{h}}(\mathbb{P}) = \mathrm{Hb}_{\mathrm{h}}(\mathbb{Q}) = \emptyset$.

Theorem. Let $\mathbb{P}$ and $\mathbb{Q}$ be two compatible `smodels` program modules having the *EVA property*, i.e., enough visible atoms. Then

$$\mathbb{P} \equiv_{\mathrm{m}} \mathbb{Q} \text{ iff } \mathrm{SM}(\mathrm{EQT}(\mathbb{P}, \mathbb{Q})) = \mathrm{SM}(\mathrm{EQT}(\mathbb{Q}, \mathbb{P})) = \emptyset.$$

Remarks. If $\mathrm{Hb}_{\mathrm{i}}(\mathbb{P}) = I = \mathrm{Hb}_{\mathrm{i}}(\mathbb{Q}) \neq \emptyset$, then the stable models $\mathrm{EQT}(\mathbb{P}, \mathbb{Q})$ are determined using the respective input generator $\mathbb{G}_I$.

Moreover, if $\mathbb{P} \sqcup \mathbb{R}$ and $\mathbb{Q} \sqcup \mathbb{R}$ are defined, then $\mathrm{EQT}(\mathbb{P} \sqcup \mathbb{R}, \mathbb{Q} \sqcup \mathbb{R})$ and $\mathrm{EQT}(\mathbb{P}, \mathbb{Q}) \sqcup \mathbb{R}$ have the same stable models (if any).

Tool Support

- The current `smodels` system does not distinguish input atoms.
- For now, the working definition is that input atoms have a name, i.e., are visible, but do not have any defining rules.
- The join $\sqcup$ operator of `smodels` programs has been implemented as a *linker* called `lpcat` (option flag `-m` indicates modules).

```
$ lparse p.lp > p.sm; lparse q.lp > q.sm
$ lpcat -m p.sm q.sm | igen | smodels 0
```

  In the pipeline, `igen` adds an input generator to the program
- The translator for equivalence checking, i.e., `lpeq`, supports the verification of modular equivalence (option flag `-m`).

```
$ lpeq -m p.sm q.sm | lpcat - r.sm | igen | smodels 1
$ lpeq -m q.sm p.sm | lpcat - r.sm | igen | smodels 1
```


OBJECTIVES

- You are able to form a (positive) dependency graph for a given logic program and exploit it in the computation of stable models.
- You understand the limitations of stable model semantics in view obtaining a compositional semantics for ASP.
- You are able to relate the notion of modular equivalence with weak and strong equivalence—as regards strength and *abstract properties* such as congruence.
- You are familiar with the basic tools for linking `smodels` program modules (`lpcat`) and verifying their equivalence (`lpeq`).

TIME TO PONDER

Consider program modules $\mathbb{P} = \langle R, I, O, H \rangle$ for which the hidden part $\mathbb{P}_{\mathrm{h}} = \langle R_{\mathrm{h}}, I \cup O, H, \emptyset \rangle$ is essentially a stratified program, i.e., R_{h} is stratified when reduced with respect to an interpretation $N \subseteq I \cup O$.

- Prove that modules of this kind have enough visible atoms.
- Provide an example of a program module which is not stratified in this sense but still has the EVA property.