Implementing a Layered Models based Query-Solving Engine

L. M. Pereira A. M. Pinto

CENTRIA — Artificial Intelligence Center
Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa

Outline

Outline

Outline

Outline

Outline

Motivation

- NLPs are commonly used for KRR
- The current standard 2-v semantics (Stable Models) does not deal appropriately with all NLPs

Example

$$
\begin{aligned}
\mathit{beach} &\leftarrow \mathit{not\ mountain} \\
\mathit{mountain} &\leftarrow \mathit{not\ travel} \\
\mathit{travel} &\leftarrow \mathit{not\ beach}, \mathit{passport\_ok} \\
\mathit{passport\_ok} &\leftarrow \mathit{not\ expired\_passport} \\
\mathit{expired\_passport} &\leftarrow \mathit{not\ passport\_ok}
\end{aligned}
$$

- What is the semantics of this program? SMs say the only solution is $\{\mathit{expired\_passport}, \mathit{mountain}\}$
- What if the passport is ok? SMs say: "No semantics!"

Motivation

- NLPs are commonly used for KRR
- The current standard 2-v semantics (Stable Models) does not deal appropriately with all NLPs

Example

$$
\begin{aligned}
\mathit{beach} &\leftarrow \mathit{not\ mountain} \\
\mathit{mountain} &\leftarrow \mathit{not\ travel} \\
\mathit{travel} &\leftarrow \mathit{not\ beach}, \mathit{passport\_ok} \\
\mathit{passport\_ok} &\leftarrow \mathit{not\ expired\_passport} \\
\mathit{expired\_passport} &\leftarrow \mathit{not\ passport\_ok}
\end{aligned}
$$

- What is the semantics of this program? SMs say the only solution is $\{\mathit{expired\_passport}, \mathit{mountain}\}$
- What if the passport is ok? SMs say: "No semantics!"

Motivation

- NLPs are commonly used for KRR
- The current standard 2-v semantics (Stable Models) does not deal appropriately with all NLPs

Example

$$
\begin{aligned}
\mathit{beach} &\leftarrow \mathit{not\ mountain} \\
\mathit{mountain} &\leftarrow \mathit{not\ travel} \\
\mathit{travel} &\leftarrow \mathit{not\ beach}, \mathit{passport\_ok} \\
\mathit{passport\_ok} &\leftarrow \mathit{not\ expired\_passport} \\
\mathit{expired\_passport} &\leftarrow \mathit{not\ passport\_ok}
\end{aligned}
$$

- What is the semantics of this program? SMs say the only solution is $\{\mathit{expired\_passport}, \mathit{mountain}\}$
- What if the passport is ok? SMs say: "No semantics!"

Motivation

- NLPs are commonly used for KRR
- The current standard 2-v semantics (Stable Models) does not deal appropriately with all NLPs

Example

$$
\begin{aligned}
\mathit{beach} &\leftarrow \mathit{not\ mountain} \\
\mathit{mountain} &\leftarrow \mathit{not\ travel} \\
\mathit{travel} &\leftarrow \mathit{not\ beach}, \mathit{passport\_ok} \\
\mathit{passport\_ok} &\leftarrow \mathit{not\ expired\_passport} \\
\mathit{expired\_passport} &\leftarrow \mathit{not\ passport\_ok}
\end{aligned}
$$

- What is the semantics of this program? SMs say the only solution is $\{\mathit{expired\_passport}, \mathit{mountain}\}$
- What if the passport is ok? SMs say: "No semantics!"

Motivation

- NLPs are commonly used for KRR
- The current standard 2-v semantics (Stable Models) does not deal appropriately with all NLPs

Example

$$
\begin{aligned}
\mathit{beach} &\leftarrow \mathit{not\ mountain} \\
\mathit{mountain} &\leftarrow \mathit{not\ travel} \\
\mathit{travel} &\leftarrow \mathit{not\ beach}, \mathit{passport\_ok} \\
\mathit{passport\_ok} &\leftarrow \mathit{not\ expired\_passport} \\
\mathit{expired\_passport} &\leftarrow \mathit{not\ passport\_ok}
\end{aligned}
$$

- What is the semantics of this program? SMs say the only solution is $\{\mathit{expired\_passport}, \mathit{mountain}\}$
- What if the passport is ok? SMs say: "No semantics!"

Outline

Background Notions (i)

Definition

A Logic Rule r has the general form

$$A \leftarrow B_1, \ldots, B_n, not\ C_1, \ldots, not\ C_m$$

where A, the B_i and the C_j are atoms.
We say A Directly Depends on each B_i and on each C_j.
Dependency being the transitive closure of Direct Dependency.

Definition

A NLP P is a (possibly infinite) set of ground Logic Rules

A non ground rule stands for the whole collection of its ground instances

Background Notions (i)

Definition

A Logic Rule r has the general form

$$A \leftarrow B_1, \ldots, B_n, not\ C_1, \ldots, not\ C_m$$

where A, the B_i and the C_j are atoms.
We say A Directly Depends on each B_i and on each C_j.
Dependency being the transitive closure of Direct Dependency.

Definition

A NLP P is a (possibly infinite) set of ground Logic Rules

A non ground rule stands for the whole collection of its ground instances

Background Notions (i)

Definition

A Logic Rule r has the general form

$$A \leftarrow B_1, \ldots, B_n, not\ C_1, \ldots, not\ C_m$$

where A, the B_i and the C_j are atoms.
We say A Directly Depends on each B_i and on each C_j.
Dependency being the transitive closure of Direct Dependency.

Definition

A NLP P is a (possibly infinite) set of ground Logic Rules

A non ground rule stands for the whole collection of its ground instances

Background Notions (ii)

Definition

Two atoms ***a*** and ***b*** are in a loop iff they depend on each other.

$$\begin{aligned} \mathit{passport\_ok} &\leftarrow \mathit{not\ expired\_passport} \\ \mathit{expired\_passport} &\leftarrow \mathit{not\ passport\_ok} \end{aligned}$$

passport_ok and *expired_passport* are in an Even Loop Over default Negation (ELON).

$$\begin{aligned} \mathit{beach} &\leftarrow \mathit{not\ mountain} \\ \mathit{mountain} &\leftarrow \mathit{not\ travel} \\ \mathit{travel} &\leftarrow \mathit{not\ beach}, \mathit{passport\_ok} \end{aligned}$$

beach, *mountain*, and *travel* are in one same OLON.

Background Notions (ii)

Definition

Two atoms ***a*** and ***b*** are in a loop iff they depend on each other.

$$\begin{aligned} \textit{passport\_ok} &\leftarrow \textit{not expired\_passport} \\ \textit{expired\_passport} &\leftarrow \textit{not passport\_ok} \end{aligned}$$

passport_ok and *expired_passport* are in an Even Loop Over default Negation (ELON).

$$\begin{aligned} \textit{beach} &\leftarrow \textit{not mountain} \\ \textit{mountain} &\leftarrow \textit{not travel} \\ \textit{travel} &\leftarrow \textit{not beach}, \textit{passport\_ok} \end{aligned}$$

beach, *mountain*, and *travel* are in one same OLON.

Background Notions (ii)

Definition

Two atoms ***a*** and ***b*** are in a loop iff they depend on each other.

$$\begin{aligned} \textit{passport\_ok} &\leftarrow \textit{not expired\_passport} \\ \textit{expired\_passport} &\leftarrow \textit{not passport\_ok} \end{aligned}$$

passport_ok and *expired_passport* are in an Even Loop Over default Negation (ELON).

$$\begin{aligned} \textit{beach} &\leftarrow \textit{not mountain} \\ \textit{mountain} &\leftarrow \textit{not travel} \\ \textit{travel} &\leftarrow \textit{not beach}, \textit{passport\_ok} \end{aligned}$$

beach, *mountain*, and *travel* are in one same OLON.

Background Notions (iii)

- Usual notion of stratification says: atom ***a*** is in a stratum above of ***b*** iff ***a*** depends on ***b***
- Consequence: both even and odd loops are *unstratifiable*
- Usual notion of support says: an atom ***a*** is supported in a model M iff **all** the literals of at least one body of a rule for ***a*** are true in M

Background Notions (iii)

- Usual notion of stratification says: atom ***a*** is in a stratum above of ***b*** iff ***a*** depends on ***b***
- Consequence: both even and odd loops are *unstratifiable*
- Usual notion of support says: an atom ***a*** is supported in a model M iff **all** the literals of at least one body of a rule for ***a*** are true in M

Background Notions (iii)

- Usual notion of stratification says: atom ***a*** is in a stratum above of ***b*** iff ***a*** depends on ***b***
- Consequence: both even and odd loops are *unstratifiable*
- Usual notion of support says: an atom ***a*** is supported in a model *M* iff **all** the literals of at least one body of a rule for ***a*** are true in *M*

Layering — Intuitive Idea (i)

- Layering is a generalization of stratification that tackles all kinds of loops
- Layering says:
  - two atoms ***a*** and ***b*** are in the same layer iff they are in a loop with each other
  - if atom ***a*** depends on ***b*** and they are not in a loop with each other, then ***a*** is *strictly above* ***b***
- Consequence: all NLPs are *layerable* (a loop's rules are in the same layer)
- Layered notion of support says: an atom ***a*** is supported in a model *M* iff there is some rule for ***a*** whose body literals that are *strictly below* ***a*** are true in *M*

Layering — Intuitive Idea (i)

- Layering is a generalization of stratification that tackles all kinds of loops
- Layering says:
  - two atoms ***a*** and ***b*** are in the same layer iff they are in a loop with each other
  - if atom ***a*** depends on ***b*** and they are not in a loop with each other, then ***a*** is *strictly above* ***b***
- Consequence: all NLPs are *layerable* (a loop's rules are in the same layer)
- Layered notion of support says: an atom ***a*** is supported in a model *M* iff there is some rule for ***a*** whose body literals that are *strictly below* ***a*** are true in *M*

Layering — Intuitive Idea (i)

- Layering is a generalization of stratification that tackles all kinds of loops
- Layering says:
  - two atoms ***a*** and ***b*** are in the same layer iff they are in a loop with each other
  - if atom ***a*** depends on ***b*** and they are not in a loop with each other, then ***a*** is *strictly above* ***b***
- Consequence: all NLPs are *layerable* (a loop's rules are in the same layer)
- Layered notion of support says: an atom ***a*** is supported in a model *M* iff there is some rule for ***a*** whose body literals that are *strictly below* ***a*** are true in *M*

Layering — Intuitive Idea (i)

- Layering is a generalization of stratification that tackles all kinds of loops
- Layering says:
  - two atoms ***a*** and ***b*** are in the same layer iff they are in a loop with each other
  - if atom ***a*** depends on ***b*** and they are not in a loop with each other, then ***a*** is *strictly above* ***b***
- Consequence: all NLPs are *layerable* (a loop's rules are in the same layer)
- Layered notion of support says: an atom ***a*** is supported in a model *M* iff there is some rule for ***a*** whose body literals that are *strictly below* ***a*** are true in *M*

Layering — Intuitive Idea (i)

- Layering is a generalization of stratification that tackles all kinds of loops
- Layering says:
  - two atoms ***a*** and ***b*** are in the same layer iff they are in a loop with each other
  - if atom ***a*** depends on ***b*** and they are not in a loop with each other, then ***a*** is *strictly above* ***b***
- Consequence: all NLPs are *layerable* (a loop's rules are in the same layer)
- Layered notion of support says: an atom ***a*** is supported in a model *M* iff there is some rule for ***a*** whose body literals that are *strictly below* ***a*** are true in *M*

Layering — Intuitive Idea (i)

- Layering is a generalization of stratification that tackles all kinds of loops
- Layering says:
  - two atoms ***a*** and ***b*** are in the same layer iff they are in a loop with each other
  - if atom ***a*** depends on ***b*** and they are not in a loop with each other, then ***a*** is *strictly above* ***b***
- Consequence: all NLPs are *layerable* (a loop's rules are in the same layer)
- Layered notion of support says: an atom ***a*** is supported in a model *M* iff there is some rule for ***a*** whose body literals that are *strictly below* ***a*** are true in *M*

Layering — Intuitive Idea (ii)

Example

$$
\begin{aligned}
\mathit{beach} &\leftarrow \mathit{not\ mountain} \\
\mathit{mountain} &\leftarrow \mathit{not\ travel} \\
\mathit{travel} &\leftarrow \mathit{not\ beach}, \mathit{passport\_ok}
\end{aligned}
$$

$$
\begin{aligned}
\mathit{passport\_ok} &\leftarrow \mathit{not\ expired\_passport} \\
\mathit{expired\_passport} &\leftarrow \mathit{not\ passport\_ok}
\end{aligned}
$$

Layering — Intuitive Idea (ii)

Example

$$
\begin{array}{rcll}
\mathit{beach} & \leftarrow & \mathit{not\ mountain} & \\
\mathit{mountain} & \leftarrow & \mathit{not\ travel} & \\
\mathit{travel} & \leftarrow & \mathit{not\ beach}, \mathit{passport\_ok} & \\
 & & & \\
\mathit{passport\_ok} & \leftarrow & \mathit{not\ expired\_passport} & \text{Layer 1} \\
\mathit{expired\_passport} & \leftarrow & \mathit{not\ passport\_ok} &
\end{array}
$$

Layering — Intuitive Idea (ii)

Example

$$
\begin{array}{rcll}
\mathit{beach} & \leftarrow & \mathit{not\ mountain} & \\
\mathit{mountain} & \leftarrow & \mathit{not\ travel} & \text{Layer 2} \\
\mathit{travel} & \leftarrow & \mathit{not\ beach}, \mathit{passport\_ok} & \\
 & & & \\
\mathit{passport\_ok} & \leftarrow & \mathit{not\ expired\_passport} & \text{Layer 1} \\
\mathit{expired\_passport} & \leftarrow & \mathit{not\ passport\_ok} &
\end{array}
$$

How to Compute Layered Models — Iterative Method

1. Compute the Layers $L_1, \ldots, L_n$ of P
2. Compute a Minimal Model M_1 of layer L_1
3. For each layer L_{i+1}
   1. Reduce L_{i+1} by M_i, i.e. remove body literals true in M_i
   2. Compute a Minimal Model M_{i+1} of $L_{i+1} \cup M_i$

Such a Model M_n is a Layered Model of P

How to Compute Layered Models — Iterative Method

1. Compute the Layers $L_1, \ldots, L_n$ of P
2. Compute a Minimal Model M_1 of layer L_1
3. For each layer L_{i+1}
   1. Reduce L_{i+1} by M_i, i.e. remove body literals true in M_i
   2. Compute a Minimal Model M_{i+1} of $L_{i+1} \cup M_i$

Such a Model M_n is a Layered Model of P

How to Compute Layered Models — Iterative Method

1. Compute the Layers $L_1, \ldots, L_n$ of P
2. Compute a Minimal Model M_1 of layer L_1
3. For each layer L_{i+1}
   1. Reduce L_{i+1} by M_i, i.e. remove body literals true in M_i
   2. Compute a Minimal Model M_{i+1} of $L_{i+1} \cup M_i$

Such a Model M_n is a Layered Model of P

How to Compute Layered Models — Iterative Method

1. Compute the Layers $L_1, \ldots, L_n$ of P
2. Compute a Minimal Model M_1 of layer L_1
3. For each layer L_{i+1}
   1. Reduce L_{i+1} by M_i, i.e. remove body literals true in M_i
   2. Compute a Minimal Model M_{i+1} of $L_{i+1} \cup M_i$

Such a Model M_n is a Layered Model of P

How to Compute Layered Models — Iterative Method

1. Compute the Layers $L_1, \ldots, L_n$ of P
2. Compute a Minimal Model M_1 of layer L_1
3. For each layer L_{i+1}
   1. Reduce L_{i+1} by M_i, i.e. remove body literals true in M_i
   2. Compute a Minimal Model M_{i+1} of $L_{i+1} \cup M_i$

Such a Model M_n is a Layered Model of P

How to Compute Layered Models — Iterative Method

1. Compute the Layers $L_1, \ldots, L_n$ of P
2. Compute a Minimal Model M_1 of layer L_1
3. For each layer L_{i+1}
   1. Reduce L_{i+1} by M_i, i.e. remove body literals true in M_i
   2. Compute a Minimal Model M_{i+1} of $L_{i+1} \cup M_i$

Such a Model M_n is a Layered Model of P

Layered Models of the Example

Example

$$
\begin{aligned}
b &\leftarrow \mathit{not}\ m \\
m &\leftarrow \mathit{not}\ t \\
t &\leftarrow \mathit{not}\ b\ ,\ p\_ok
\end{aligned}
$$

$$
\begin{aligned}
p\_ok &\leftarrow \mathit{not}\ exp\_p \\
exp\_p &\leftarrow \mathit{not}\ p\_ok
\end{aligned}
$$

Layered Models of the Example

Example

$$
\begin{array}{rcl}
b & \leftarrow & not\ m \\
m & \leftarrow & not\ t \\
t & \leftarrow & not\ b\ , p\_ok
\end{array}
$$

$$
\begin{array}{rcll}
p\_ok & \leftarrow & not\ exp\_p & M_{1_1} = \{p\_ok\} \\
exp\_p & \leftarrow & not\ p\_ok &
\end{array}
$$

Layered Models of the Example

Example

$$
\begin{array}{rcl}
b & \leftarrow & not\ m \\
m & \leftarrow & not\ t \\
t & \leftarrow & not\ b\ , p\_ok
\end{array}
$$

$$
\begin{array}{rcll}
p\_ok & \leftarrow & not\ exp\_p & M_{1_1} = \{p\_ok\} \\
exp\_p & \leftarrow & not\ p\_ok & M_{1_2} = \{exp\_p\}
\end{array}
$$

Layered Models of the Example

Example

$$
\begin{array}{rcl}
b & \leftarrow & not\ m \\
m & \leftarrow & not\ t \\
t & \leftarrow & not\ b\ , p\_ok
\end{array}
$$

$$
\begin{array}{rcll}
p\_ok & \leftarrow & not\ exp\_p & M_{1_1} = \{p\_ok\} \\
exp\_p & \leftarrow & not\ p\_ok &
\end{array}
$$

Layered Models of the Example

Example

$$
\begin{array}{rcl}
b & \leftarrow & not\ m \\
m & \leftarrow & not\ t \\
t & \leftarrow & not\ b
\end{array}
$$

$$
\begin{array}{rcl}
p\_ok & \leftarrow & not\ exp\_p \\
exp\_p & \leftarrow & not\ p\_ok
\end{array}
\qquad M_{1_1} = \{p\_ok\}
$$

Layered Models of the Example

Example

$$
\begin{array}{rcll}
b & \leftarrow & not\ m & M_{2_1} = \{b, m, p\_ok\} \\
m & \leftarrow & not\ t & \\
t & \leftarrow & not\ b &
\end{array}
$$

$$
\begin{array}{rcll}
p\_ok & \leftarrow & not\ exp\_p & M_{1_1} = \{p\_ok\} \\
exp\_p & \leftarrow & not\ p\_ok &
\end{array}
$$

Layered Models of the Example

Example

$$
\begin{array}{rcll}
b & \leftarrow & not\ m & M_{2_1} = \{b, m, p\_ok\} \\
m & \leftarrow & not\ t & M_{2_2} = \{m, t, p\_ok\} \\
t & \leftarrow & not\ b &
\end{array}
$$

$$
\begin{array}{rcll}
p\_ok & \leftarrow & not\ exp\_p & M_{1_1} = \{p\_ok\} \\
exp\_p & \leftarrow & not\ p\_ok &
\end{array}
$$

Layered Models of the Example

Example

$$
\begin{array}{rcll}
b & \leftarrow & not\ m & M_{2_1} = \{b, m, p\_ok\} \\
m & \leftarrow & not\ t & M_{2_2} = \{m, t, p\_ok\} \\
t & \leftarrow & not\ b & M_{2_3} = \{b, t, p\_ok\}
\end{array}
$$

$$
\begin{array}{rcll}
p\_ok & \leftarrow & not\ exp\_p & M_{1_1} = \{p\_ok\} \\
exp\_p & \leftarrow & not\ p\_ok &
\end{array}
$$

Layered Models of the Example

Example

$$
\begin{array}{rcl}
b & \leftarrow & not\ m \\
m & \leftarrow & not\ t \\
t & \leftarrow & not\ b\ , p\_ok
\end{array}
$$

$$
\begin{array}{rcll}
p\_ok & \leftarrow & not\ exp\_p & \\
exp\_p & \leftarrow & not\ p\_ok & M_{1_2} = \{exp\_p\}
\end{array}
$$

Layered Models of the Example

Example

$$m \leftarrow$$

$$\begin{array}{lll} p\_ok & \leftarrow & not\ exp\_p \\ exp\_p & \leftarrow & not\ p\_ok \end{array} \qquad M_{1_2} = \{exp\_p\}$$

Layered Models of the Example

Example

$$m \leftarrow \qquad M_{2_4} = \{m, exp\_p\}$$

$$\begin{array}{lll} p\_ok & \leftarrow & not\ exp\_p \\ exp\_p & \leftarrow & not\ p\_ok \end{array} \qquad M_{1_2} = \{exp\_p\}$$

Details about Properties

- Relevance: a semantics is relevant iff
  $\forall_a a \in Sem(P) \Leftrightarrow a \in Sem(Rel_P(a))$

  Only relevant semantics can benefit from top-down query-solving

- Cumulativity: a semantics is cumulative iff
  $\forall_{a,b}(a \in Sem(P) \wedge b \in Sem(P)) \Rightarrow a \in Sem(P \cup \{b\})$

  Only cumulative semantics can benefit from tabling methods

Properties of Layered Models Semantics

Property	SMs	LMs
Existence		
Relevance		
Cumulativity		
Classical Support		
Layered Support		

Properties of Layered Models Semantics

Property	SMs	LMs
Existence	No	
Relevance		
Cumulativity		
Classical Support		
Layered Support		

Properties of Layered Models Semantics

Property	SMs	LMs
Existence	No	Yes
Relevance		
Cumulativity		
Classical Support		
Layered Support		

Properties of Layered Models Semantics

Property	SMs	LMs
Existence	No	Yes
Relevance	No	
Cumulativity		
Classical Support		
Layered Support		

Properties of Layered Models Semantics

Property	SMs	LMs
Existence	No	Yes
Relevance	No	Yes
Cumulativity		
Classical Support		
Layered Support		

Properties of Layered Models Semantics

Property	SMs	LMs
Existence	No	Yes
Relevance	No	Yes
Cumulativity	No	
Classical Support		
Layered Support		

Properties of Layered Models Semantics

Property	SMs	LMs
Existence	No	Yes
Relevance	No	Yes
Cumulativity	No	Yes
Classical Support		
Layered Support		

Properties of Layered Models Semantics

Property	SMs	LMs
Existence	No	Yes
Relevance	No	Yes
Cumulativity	No	Yes
Classical Support	Yes	
Layered Support		

Properties of Layered Models Semantics

Property	SMs	LMs
Existence	No	Yes
Relevance	No	Yes
Cumulativity	No	Yes
Classical Support	Yes	No
Layered Support		

Properties of Layered Models Semantics

Property	SMs	LMs
Existence	No	Yes
Relevance	No	Yes
Cumulativity	No	Yes
Classical Support	Yes	No
Layered Support	No	

Properties of Layered Models Semantics

Property	SMs	LMs
Existence	No	Yes
Relevance	No	Yes
Cumulativity	No	Yes
Classical Support	Yes	No
Layered Support	No	Yes

Properties of Layered Models Semantics

Property	SMs	LMs
Existence	No	Yes
Relevance	No	Yes
Cumulativity	No	Yes
Classical Support	Yes	No
Layered Support	No	Yes

Moreover,

Properties of Layered Models Semantics

Property	SMs	LMs
Existence	No	Yes
Relevance	No	Yes
Cumulativity	No	Yes
Classical Support	Yes	No
Layered Support	No	Yes

Moreover,

- *LMs* $\supseteq$ *SMs* for any NLP

Properties of Layered Models Semantics

Property	SMs	LMs
Existence	No	Yes
Relevance	No	Yes
Cumulativity	No	Yes
Classical Support	Yes	No
Layered Support	No	Yes

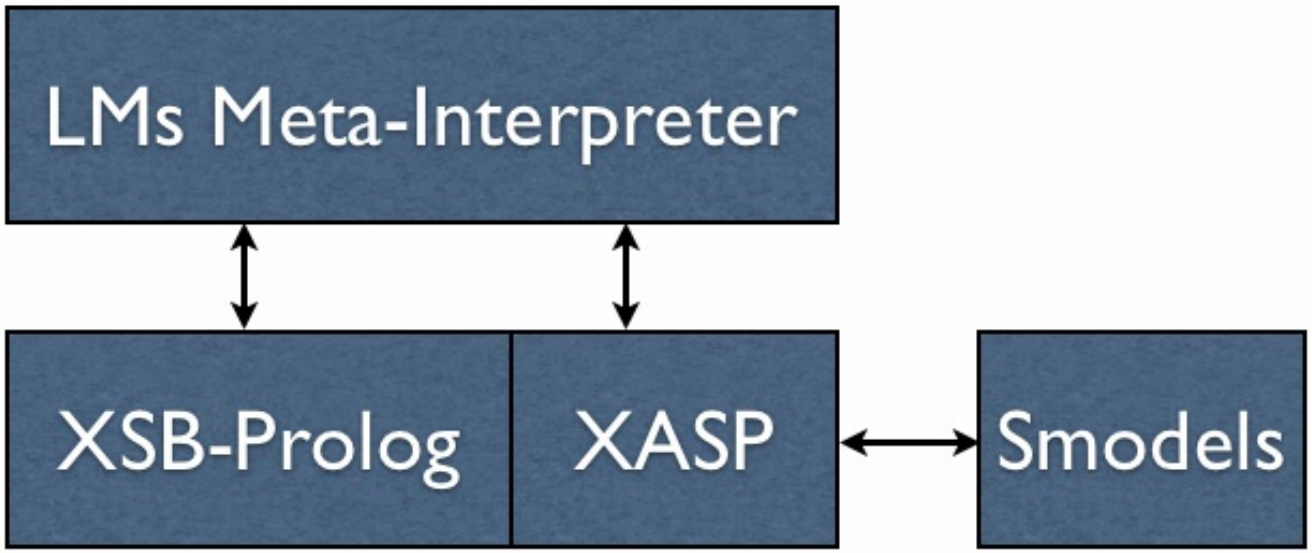
Moreover,

- $LMs \supseteq SMs$ for any NLP
- When the NLP has no OLONs, $LMs = SMs$

Outline

Implementation Architecture

Implementation is via a meta-interpreter written in XSB-Prolog. Takes advantage of XSB's XASP interface to Smodels.



Implementation's Query Solving Steps — `lmquery/2` predicate

- Load the KB in the input NLP
- Accept a query from the user
- Identify the part of the NLP relevant for the query (residuum)
- In a "top-down" (Prolog-like) fashion, traverse each call-graph branch in the residuum, and identify and solve the OLON (if there is any) along the branch
  Identifying and solving OLONs naturally respects the layering
- Send the relevant part of the NLP, along with the solved OLONs, to Smodels via XASP
- Get one solution from Smodels and present it to the user

Implementation's Query Solving Steps — `lmquery/2` predicate

- Load the KB in the input NLP
- Accept a query from the user
- Identify the part of the NLP relevant for the query (residuum)
- In a "top-down" (Prolog-like) fashion, traverse each call-graph branch in the residuum, and identify and solve the OLON (if there is any) along the branch
  Identifying and solving OLONs naturally respects the layering
- Send the relevant part of the NLP, along with the solved OLONs, to Smodels via XASP
- Get one solution from Smodels and present it to the user

Implementation's Query Solving Steps — `lmquery/2` predicate

- Load the KB in the input NLP
- Accept a query from the user
- Identify the part of the NLP relevant for the query (residuum)
- In a "top-down" (Prolog-like) fashion, traverse each call-graph branch in the residuum, and identify and solve the OLON (if there is any) along the branch
  Identifying and solving OLONs naturally respects the layering
- Send the relevant part of the NLP, along with the solved OLONs, to Smodels via XASP
- Get one solution from Smodels and present it to the user

Implementation's Query Solving Steps — `lmquery/2` predicate

- Load the KB in the input NLP
- Accept a query from the user
- Identify the part of the NLP relevant for the query (residuum)
- In a "top-down" (Prolog-like) fashion, traverse each call-graph branch in the residuum, and identify and solve the OLON (if there is any) along the branch
  Identifying and solving OLONs naturally respects the layering
- Send the relevant part of the NLP, along with the solved OLONs, to Smodels via XASP
- Get one solution from Smodels and present it to the user

Implementation's Query Solving Steps — `lmquery/2` predicate

- Load the KB in the input NLP
- Accept a query from the user
- Identify the part of the NLP relevant for the query (residuum)
- In a "top-down" (Prolog-like) fashion, traverse each call-graph branch in the residuum, and identify and solve the OLON (if there is any) along the branch
  Identifying and solving OLONs naturally respects the layering
- Send the relevant part of the NLP, along with the solved OLONs, to Smodels via XASP
- Get one solution from Smodels and present it to the user

Implementation's Query Solving Steps — `lmquery/2` predicate

- Load the KB in the input NLP
- Accept a query from the user
- Identify the part of the NLP relevant for the query (residuum)
- In a "top-down" (Prolog-like) fashion, traverse each call-graph branch in the residuum, and identify and solve the OLON (if there is any) along the branch
  Identifying and solving OLONs naturally respects the layering
- Send the relevant part of the NLP, along with the solved OLONs, to Smodels via XASP
- Get one solution from Smodels and present it to the user

OLON resolution in the Example

```
?- lmquery([beach], R).
```

OLON resolution in the Example

```
?- lmquery([beach], R).
```

beach ← *not mountain*

OLON resolution in the Example

```
?- lmquery([beach], R).
```

beach ← *not mountain*

mountain ← *not travel*

OLON resolution in the Example

```
?- lmquery([beach], R).
```

beach ← *not mountain*

mountain ← *not travel*

travel ← *not beach*, *passport_ok*

OLON is detected! Depends on *passport_ok*

OLON resolution in the Example

```
?- lmquery([beach], R).
```

beach ← *not mountain*

mountain ← *not travel*

travel ← *not beach*, *passport_ok*

OLON is detected! Depends on *passport_ok*

The OLON has 2 solutions for the query 'beach':

OLON resolution in the Example

```
?- lmquery([beach], R).
```

beach ← *not mountain*
mountain ← *not travel*
travel ← *not beach*, *passport_ok*
OLON is detected! Depends on *passport_ok*

The OLON has 2 solutions for the query 'beach':
beach ← *passport_ok* and *travel* ← *passport_ok*

Both alternative rules create classical support for *beach* which still depends on *passport_ok*

OLON resolution in the Example

```
?- lmquery([beach], R).
```

beach ← *not mountain*
mountain ← *not travel*
travel ← *not beach*, *passport_ok*
OLON is detected! Depends on *passport_ok*

The OLON has 2 solutions for the query 'beach':
beach ← *passport_ok* and *travel* ← *passport_ok*

Both alternative rules create classical support for *beach* which still depends on *passport_ok*

On backtracking each alternative rule is sent along with the rest of the residuum to Smodels.

Outline

Conclusions

- Stable Models are a very important step, but we can take another step forward
- Introduced new (layered) notion of support (extends the usual notion)
- New 2-valued semantics for NLPs
  - is conservative extension to SMs
  - guarantees model existence, enjoys relevance and cumulativity
- Implemented Query-Solving Engine

Conclusions

- Stable Models are a very important step, but we can take another step forward
- Introduced new (layered) notion of support (extends the usual notion)
- New 2-valued semantics for NLPs
  - is conservative extension to SMs
  - guarantees model existence, enjoys relevance and cumulativity
- Implemented Query-Solving Engine

Conclusions

- Stable Models are a very important step, but we can take another step forward
- Introduced new (layered) notion of support (extends the usual notion)
- New 2-valued semantics for NLPs
  - is conservative extension to SMs
  - guarantees model existence, enjoys relevance and cumulativity
- Implemented Query-Solving Engine

Conclusions

- Stable Models are a very important step, but we can take another step forward
- Introduced new (layered) notion of support (extends the usual notion)
- New 2-valued semantics for NLPs
  - is conservative extension to SMs
  - guarantees model existence, enjoys relevance and cumulativity
- Implemented Query-Solving Engine

Conclusions

- Stable Models are a very important step, but we can take another step forward
- Introduced new (layered) notion of support (extends the usual notion)
- New 2-valued semantics for NLPs
  - is conservative extension to SMs
  - guarantees model existence, enjoys relevance and cumulativity
- Implemented Query-Solving Engine

Conclusions

- Stable Models are a very important step, but we can take another step forward
- Introduced new (layered) notion of support (extends the usual notion)
- New 2-valued semantics for NLPs
  - is conservative extension to SMs
  - guarantees model existence, enjoys relevance and cumulativity
- Implemented Query-Solving Engine

Future Work

- Thorough complexity study of the Layered Models semantics
- Implement a full-fledged query answering Layered Models engine for first order NLPs at WAM level (already underway in cooperation with XSB)
- Implement abduction on top of Layered Models
- Implement constructive negation on top of Layered Models with abduction

Future Work

- Thorough complexity study of the Layered Models semantics
- Implement a full-fledged query answering Layered Models engine for first order NLPs at WAM level (already underway in cooperation with XSB)
- Implement abduction on top of Layered Models
- Implement constructive negation on top of Layered Models with abduction

Future Work

- Thorough complexity study of the Layered Models semantics
- Implement a full-fledged query answering Layered Models engine for first order NLPs at WAM level (already underway in cooperation with XSB)
- Implement abduction on top of Layered Models
- Implement constructive negation on top of Layered Models with abduction

Future Work

- Thorough complexity study of the Layered Models semantics
- Implement a full-fledged query answering Layered Models engine for first order NLPs at WAM level (already underway in cooperation with XSB)
- Implement abduction on top of Layered Models
- Implement constructive negation on top of Layered Models with abduction

References

- L. M. Pereira and A. M. Pinto, Revised Stable Models - a Semantics for LPs, Procs. EPIA'05, LNAI, 2005
- L. Castro, T. Swift and D. S. Warren, XASP: Answer Set Programming with XSB and Smodels, `http://xsb.sourceforge.net/packages/xasp.pdf`
- M. Gelfond and V. Lifschitz, The Stable Model Semantics for Logic Programming, ICLP/SLP, MIT Press, 1988
- F. Fages, Consistency of Clark's Completion and Existence of SMs, Methods of Logic in Comp. Sci., 1994, vol. 1

Demo

Let's see a demo
of the example!

Thank you!

FAQs

- Layering calculation complexity? Computing Layering = Finding SCCs = Quadratic
- Layer L_{i+1} reduction by M_i:
  - remove all rules from L_{i+1} with *not a* in the body where $a \in M_i$
  - remove all atoms in M_i from the bodies of rules of L_{i+1}
  - remove all the remaining *not x*
- Integrity Constraints (ICs) are now written as

  $$falsum \leftarrow Body\_of\_IC$$

  instead of

  $$falsum \leftarrow not\ falsum, Body\_of\_IC$$
- Queries with ICs must include *not falsum*, e.g.

```
lmQuery([beach,not falsum],R).
```

FAQs

- Layering calculation complexity? Computing Layering = Finding SCCs = Quadratic
- Layer L_{i+1} reduction by M_i:
  - remove all rules from L_{i+1} with *not a* in the body where $a \in M_i$
  - remove all atoms in M_i from the bodies of rules of L_{i+1}
  - remove all the remaining *not x*
- Integrity Constraints (ICs) are now written as

  $$falsum \leftarrow Body\_of\_IC$$

  instead of

  $$falsum \leftarrow not\ falsum, Body\_of\_IC$$

- Queries with ICs must include *not falsum*, e.g.

```
lmQuery([beach,not falsum],R).
```

FAQs

- Layering calculation complexity? Computing Layering = Finding SCCs = Quadratic
- Layer L_{i+1} reduction by M_i:
  - remove all rules from L_{i+1} with *not a* in the body where $a \in M_i$
  - remove all atoms in M_i from the bodies of rules of L_{i+1}
  - remove all the remaining *not x*
- Integrity Constraints (ICs) are now written as

  $$falsum \leftarrow Body\_of\_IC$$

  instead of

  $$falsum \leftarrow not\ falsum, Body\_of\_IC$$

- Queries with ICs must include *not falsum*, e.g.

```
lmQuery([beach,not falsum],R).
```

FAQs

- Layering calculation complexity? Computing Layering = Finding SCCs = Quadratic
- Layer L_{i+1} reduction by M_i:
  - remove all rules from L_{i+1} with *not a* in the body where $a \in M_i$
  - remove all atoms in M_i from the bodies of rules of L_{i+1}
  - remove all the remaining *not x*
- Integrity Constraints (ICs) are now written as

  $$falsum \leftarrow Body\_of\_IC$$

  instead of

  $$falsum \leftarrow not\ falsum, Body\_of\_IC$$

- Queries with ICs must include *not falsum*, e.g.

```
lmQuery([beach,not falsum],R).
```

FAQs

- Layering calculation complexity? Computing Layering = Finding SCCs = Quadratic
- Layer L_{i+1} reduction by M_i:
  - remove all rules from L_{i+1} with *not a* in the body where $a \in M_i$
  - remove all atoms in M_i from the bodies of rules of L_{i+1}
  - remove all the remaining *not x*
- Integrity Constraints (ICs) are now written as

  $$falsum \leftarrow Body\_of\_IC$$

  instead of

  $$falsum \leftarrow not\ falsum, Body\_of\_IC$$

- Queries with ICs must include *not falsum*, e.g.

```
lmQuery([beach,not falsum],R).
```

FAQs

- Layering calculation complexity? Computing Layering = Finding SCCs = Quadratic
- Layer L_{i+1} reduction by M_i:
  - remove all rules from L_{i+1} with *not a* in the body where $a \in M_i$
  - remove all atoms in M_i from the bodies of rules of L_{i+1}
  - remove all the remaining *not x*
- Integrity Constraints (ICs) are now written as

  $$falsum \leftarrow Body\_of\_IC$$

  instead of

  $$falsum \leftarrow not\ falsum, Body\_of\_IC$$

- Queries with ICs must include *not falsum*, e.g.

```
lmQuery([beach,not falsum],R).
```

FAQs

- Layering calculation complexity? Computing Layering = Finding SCCs = Quadratic
- Layer L_{i+1} reduction by M_i:
  - remove all rules from L_{i+1} with *not a* in the body where $a \in M_i$
  - remove all atoms in M_i from the bodies of rules of L_{i+1}
  - remove all the remaining *not x*
- Integrity Constraints (ICs) are now written as

  $$falsum \leftarrow Body\_of\_IC$$

  instead of

  $$falsum \leftarrow not\ falsum, Body\_of\_IC$$
- Queries with ICs must include *not falsum*, e.g.

```
lmQuery([beach,not falsum],R).
```