Relevance-driven Evaluation of Modular Nonmonotonic Logic Programs

Minh Dao-Tran, Thomas Eiter, Michael Fink, Thomas Krennwallner

KBS Group, Institute of Information Systems, Vienna University of Technology

LPNMR 2009 — September 17, 2009

Knowledge-Based Systems Group

Modularity in Logic Programming

- Splitting set [Lifschitz and Turner, 1994]
- Generalized Quantifiers [Eiter *et al.*, 1997]
- Templates [Ianni *et al.*, 2003]
- Macros [Baral *et al.*, 2006]
- Smodels program modules [Oikarinen and Janhunen, 2008], DLP-functions [Janhunen *et al.*, 2007]
- Incremental modularity [Gebser *et al.*, 2008]
- Modular Nonmonotonic Logic Programs
  [D_, Eiter, Fink, Krennwallner, 2009]

Modular Nonmonotonic Logic Programs

- Analogies to ordinary programming languages
  - Function prototype
  - Formal parameters
  - Function body
  - Function calls
  - Call by reference/value

Modular Nonmonotonic Logic Programs

- Analogies to ordinary programming languages
  - Function prototype – Module name
  - Formal parameters – Formal input lists
  - Function body – Rules
  - Function calls – Module calls
  - Call by reference/value – Module instances

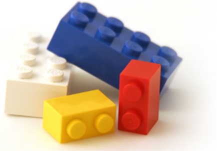
Modular Nonmonotonic Logic Programs

- Analogies to ordinary programming languages
  - Function prototype – Module name
  - Formal parameters – Formal input lists
  - Function body – Rules
  - Function calls – Module calls
  - Call by reference/value – Module instances
- Some features:
  - Mutually recursive calls between modules
  - Semantics based on instantiations

Modular Nonmonotonic Logic Programs

- Analogies to ordinary programming languages
  - Function prototype – Module name
  - Formal parameters – Formal input lists
  - Function body – Rules
  - Function calls – Module calls
  - Call by reference/value – Module instances
- Some features:
  - Mutually recursive calls between modules
  - Semantics based on instantiations
- Naive evaluation is infeasible in practice

Evaluation of MLPs



Modular Nonmonotonic Logic Programs

Stratification & Splitting for MLPs

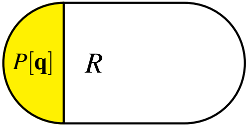
Algorithm

Syntax of MLPs

Module atoms $P[p_1, \ldots, p_k].o(t_1, \ldots, t_\ell)$

Rules: $\alpha_1 \vee \cdots \vee \alpha_k \leftarrow \beta_1, \ldots, \beta_j, \text{not } \beta_{j+1}, \ldots, \text{not } \beta_m$

Modules $m = (P[\mathbf{q}], R)$



Modular Nonmonotonic Logic Programs $\mathbf{P} = (m_1, \ldots, m_n)$

Example: Odd/Even

$\mathbf{P} = (m_1, m_2, m_3)$, where $m_1 = (P_1, R_1)$, $m_2 = (P_2[q_2], R_2)$, $m_3 = (P_3[q_3], R_3)$.

$$R_1 = \{q(a). \quad q(b). \quad ok \leftarrow P_2[q].even.\}$$

$$R_2 = \left\{ \begin{array}{lcl} q_2'(X) \vee q_2'(Y) & \leftarrow & q_2(X), q_2(Y), \\ & & X \neq Y. \\ skip_2 & \leftarrow & q_2(X), \text{not } q_2'(X). \\ even & \leftarrow & \text{not } skip_2. \\ even & \leftarrow & skip_2, P_3[q_2'].odd. \end{array} \right\}$$

$$R_3 = \left\{ \begin{array}{lcl} q_3'(X) \vee q_3'(Y) & \leftarrow & q_3(X), q_3(Y), \\ & & X \neq Y. \\ skip_3 & \leftarrow & q_3(X), \text{not } q_3'(X). \\ odd & \leftarrow & skip_3, P_2[q_3'].even. \end{array} \right\}$$

$main()$:
$n := |q|$
if $even(n)$ **then return** ok

$even(n)$:
$n' := n - 1$
if $n' < 0$ **then return** true
if $n' = 0$ **then return** false
if $odd(n')$ **then return** true
else return false

$odd(n)$:
$n' := n - 1$
if $even(n')$ **then return** true
else return false

Example: Odd/Even

$\mathbf{P} = (m_1, m_2, m_3)$, where $m_1 = (P_1, R_1)$, $m_2 = (P_2[q_2], R_2)$, $m_3 = (P_3[q_3], R_3)$.

$R_1 = \{q(a).\quad q(b).\quad ok \leftarrow P_2[q].even.\}$

$$R_2 = \left\{ \begin{array}{rcl} q_2'(X) \vee q_2'(Y) & \leftarrow & q_2(X), q_2(Y), \\ & & X \neq Y. \\ skip_2 & \leftarrow & q_2(X), \text{not } q_2'(X). \\ even & \leftarrow & \text{not } skip_2. \\ even & \leftarrow & skip_2, P_3[q_2'].odd. \end{array} \right\}$$

$$R_3 = \left\{ \begin{array}{rcl} q_3'(X) \vee q_3'(Y) & \leftarrow & q_3(X), q_3(Y), \\ & & X \neq Y. \\ skip_3 & \leftarrow & q_3(X), \text{not } q_3'(X). \\ odd & \leftarrow & skip_3, P_2[q_3'].even. \end{array} \right\}$$

$main()$:
$n := |q|$
if $even(n)$ **then return** ok

$even(n)$:
$n' := n - 1$
if $n' < 0$ **then return** true
if $n' = 0$ **then return** false
if $odd(n')$ **then return** true
else return false

$odd(n)$:
$n' := n - 1$
if $even(n')$ **then return** true
else return false

Example: Odd/Even

$\mathbf{P} = (m_1, m_2, m_3)$, where $m_1 = (P_1, R_1)$, $m_2 = (P_2[q_2], R_2)$, $m_3 = (P_3[q_3], R_3)$.

$R_1 = \{q(a). \quad q(b). \quad ok \leftarrow P_2[q].even.\}$

$$R_2 = \left\{ \begin{array}{rcl} q_2'(X) \vee q_2'(Y) & \leftarrow & q_2(X), q_2(Y), \\ & & X \neq Y. \\ skip_2 & \leftarrow & q_2(X), \text{not } q_2'(X). \\ even & \leftarrow & \text{not } skip_2. \\ even & \leftarrow & skip_2, P_3[q_2'].odd. \end{array} \right\}$$

$$R_3 = \left\{ \begin{array}{rcl} q_3'(X) \vee q_3'(Y) & \leftarrow & q_3(X), q_3(Y), \\ & & X \neq Y. \\ skip_3 & \leftarrow & q_3(X), \text{not } q_3'(X). \\ odd & \leftarrow & skip_3, P_2[q_3'].even. \end{array} \right\}$$

$main()$:
$n := |q|$
if $even(n)$ **then return** ok

$even(n)$:
$n' := n - 1$
if $n' < 0$ **then return** true
if $n' = 0$ **then return** false
if $odd(n')$ **then return** true
else return false

$odd(n)$:
$n' := n - 1$
if $even(n')$ **then return** true
else return false

Example: Odd/Even

$\mathbf{P} = (m_1, m_2, m_3)$, where $m_1 = (P_1, R_1)$, $m_2 = (P_2[q_2], R_2)$, $m_3 = (P_3[q_3], R_3)$.

$$R_1 = \{q(a). \quad q(b). \quad ok \leftarrow P_2[q].even.\}$$

$$R_2 = \left\{ \begin{array}{lcl} q_2'(X) \vee q_2'(Y) & \leftarrow & q_2(X), q_2(Y), \\ & & X \neq Y. \\ skip_2 & \leftarrow & q_2(X), \text{not } q_2'(X). \\ even & \leftarrow & \text{not } skip_2. \\ even & \leftarrow & skip_2, P_3[q_2'].odd. \end{array} \right\}$$

$$R_3 = \left\{ \begin{array}{lcl} q_3'(X) \vee q_3'(Y) & \leftarrow & q_3(X), q_3(Y), \\ & & X \neq Y. \\ skip_3 & \leftarrow & q_3(X), \text{not } q_3'(X). \\ odd & \leftarrow & skip_3, P_2[q_3'].even. \end{array} \right\}$$

$main()$:
$n := |q|$
if $even(n)$ **then return** ok

$even(n)$:
$n' := n - 1$
if $n' < 0$ **then return** true
if $n' = 0$ **then return** false
if $odd(n')$ **then return** true
else return false

$odd(n)$:
$n' := n - 1$
if $even(n')$ **then return** true
else return false

Example: Odd/Even

$\mathbf{P} = (m_1, m_2, m_3)$, where $m_1 = (P_1, R_1)$, $m_2 = (P_2[q_2], R_2)$, $m_3 = (P_3[q_3], R_3)$.

$$R_1 = \{q(a). \quad q(b). \quad ok \leftarrow P_2[q].even.\}$$

$$R_2 = \left\{ \begin{array}{rcl} q'_2(X) \vee q'_2(Y) & \leftarrow & q_2(X), q_2(Y), \\ & & X \neq Y. \\ skip_2 & \leftarrow & q_2(X), \text{not } q'_2(X). \\ even & \leftarrow & \text{not } skip_2. \\ even & \leftarrow & skip_2, P_3[q'_2].odd. \end{array} \right\}$$

$$R_3 = \left\{ \begin{array}{rcl} q'_3(X) \vee q'_3(Y) & \leftarrow & q_3(X), q_3(Y), \\ & & X \neq Y. \\ skip_3 & \leftarrow & q_3(X), \text{not } q'_3(X). \\ odd & \leftarrow & skip_3, P_2[q'_3].even. \end{array} \right\}$$

$main()$:
$n := |q|$
if $even(n)$ **then return** ok

$even(n)$:
$n' := n - 1$
if $n' < 0$ **then return** true
if $n' = 0$ **then return** false
if $odd(n')$ **then return** true
else return false

$odd(n)$:
$n' := n - 1$
if $even(n')$ **then return** true
else return false

Example: Odd/Even

$\mathbf{P} = (m_1, m_2, m_3)$, where $m_1 = (P_1, R_1)$, $m_2 = (P_2[q_2], R_2)$, $m_3 = (P_3[q_3], R_3)$.

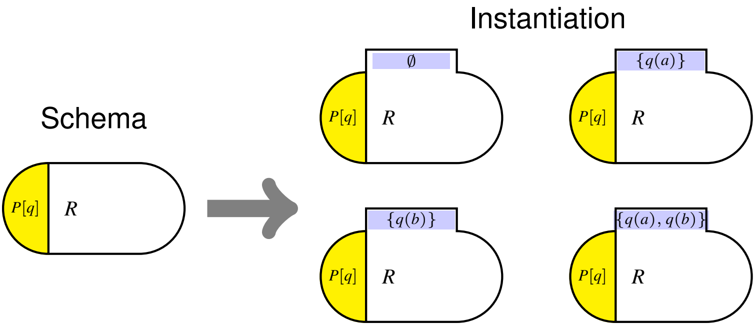
$$R_1 = \{q(a). \quad q(b). \quad ok \leftarrow P_2[q].even.\}$$

$$R_2 = \left\{ \begin{array}{rcl} q_2'(X) \vee q_2'(Y) & \leftarrow & q_2(X), q_2(Y), \\ & & X \neq Y. \\ skip_2 & \leftarrow & q_2(X), \text{not } q_2'(X). \\ even & \leftarrow & \text{not } skip_2. \\ even & \leftarrow & skip_2, P_3[q_2'].odd. \end{array} \right\}$$

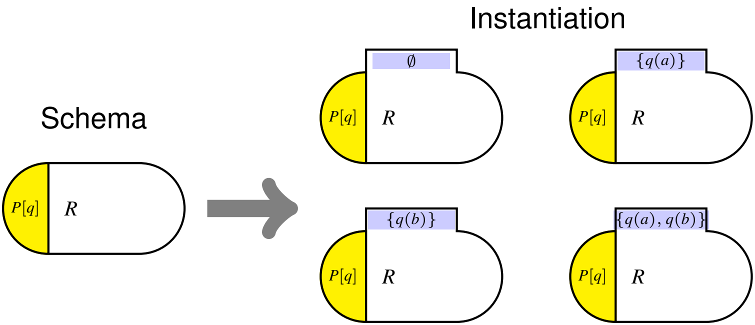
$$R_3 = \left\{ \begin{array}{rcl} q_3'(X) \vee q_3'(Y) & \leftarrow & q_3(X), q_3(Y), \\ & & X \neq Y. \\ skip_3 & \leftarrow & q_3(X), \text{not } q_3'(X). \\ odd & \leftarrow & skip_3, P_2[q_3'].even. \end{array} \right\}$$

M
$M_1/\emptyset : \{ok, q(a), q(b)\}$
$M_2/\{q_2(a), q_2(b)\} : \left\{ \begin{array}{l} even, skip_2, \\ q_2(a), q_2(b), q_2'(b) \end{array} \right\}$
$\vdots$
$M_2/\emptyset : \{even\}$
$\vdots$
$M_3/\{q_3(b)\} : \{odd, skip_3, q_3(b)\}$
$\vdots$

Instantiations of modules

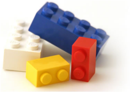


Instantiations of modules

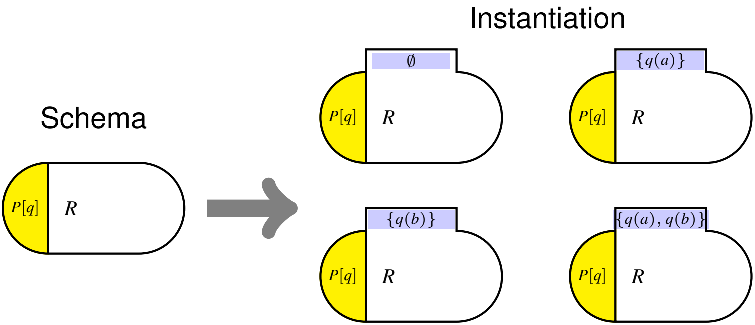


Value call: $P[S]$, with input $S \subseteq HB_{\mathbf{P}}|_{\mathbf{q}}$

$VC(\mathbf{P})$: set of all value calls $P[S]$ in $\mathbf{P}$



Instantiations of modules



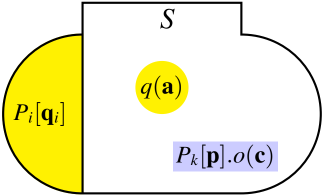
Value call: $P[S]$, with input $S \subseteq HB_{\mathbf{P}}|_{\mathbf{q}}$

$VC(\mathbf{P})$: set of all value calls $P[S]$ in $\mathbf{P}$

$I_{\mathbf{P}}(P[S]) = R \cup S$ is an instantiation of $m = (P[\mathbf{q}], R)$ with $S \subseteq HB_{\mathbf{P}}|_{\mathbf{q}}$

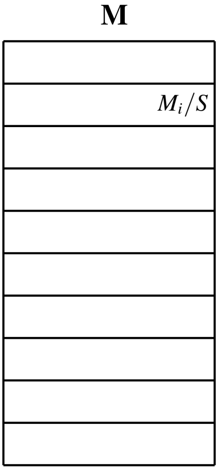
The rule base $I(\mathbf{P}) = (I_{\mathbf{P}}(P_i[S]) \mid P_i[S] \in VC(\mathbf{P}))$ is the instantiation of $\mathbf{P}$

Interpretation and Models

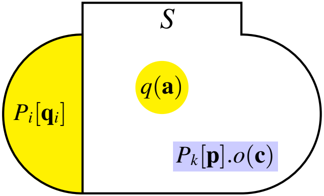


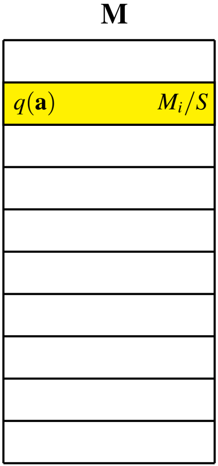
Instantiation: $I(\mathbf{P}) = (I_{\mathbf{P}}(P_i[S]) \mid P_i[S] \in VC(\mathbf{P}))$

Interpretation: $\mathbf{M} = (M_i/S \mid P_i[S] \in VC(\mathbf{P}))$



Interpretation and Models



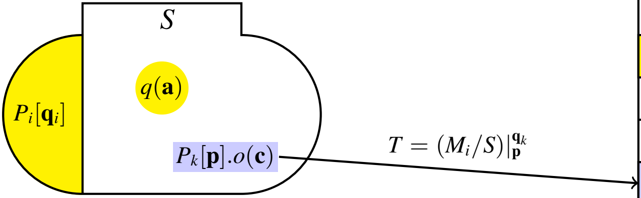


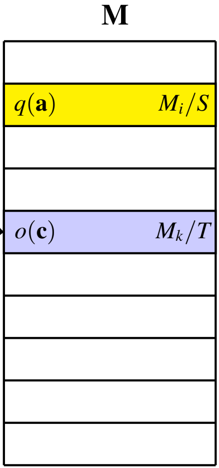
Instantiation: $I(\mathbf{P}) = (I_{\mathbf{P}}(P_i[S]) \mid P_i[S] \in VC(\mathbf{P}))$

Interpretation: $\mathbf{M} = (M_i/S \mid P_i[S] \in VC(\mathbf{P}))$

$\mathbf{M}, P_i[S] \models q(\mathbf{a})$ iff $q(\mathbf{a}) \in M_i/S$

Interpretation and Models





Instantiation: $I(\mathbf{P}) = (I_{\mathbf{P}}(P_i[S]) \mid P_i[S] \in VC(\mathbf{P}))$

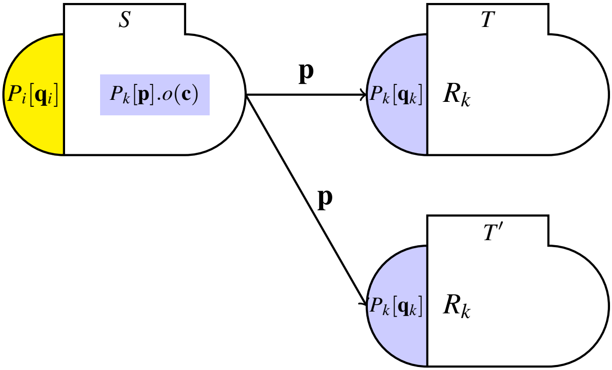
Interpretation: $\mathbf{M} = (M_i/S \mid P_i[S] \in VC(\mathbf{P}))$

$\mathbf{M}, P_i[S] \models q(\mathbf{a})$ iff $q(\mathbf{a}) \in M_i/S$

$\mathbf{M}, P_i[S] \models P_k[\mathbf{p}].o(\mathbf{c})$ iff $o(\mathbf{c}) \in M_k/T$

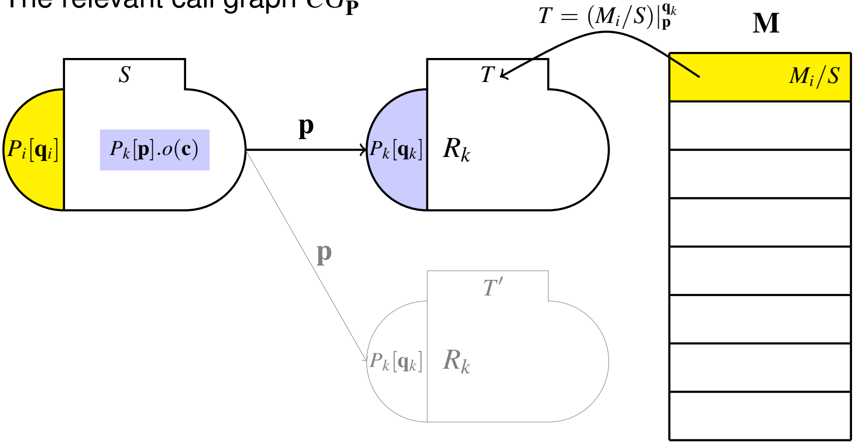
Call graphs and Relevant call graphs

The call graph $CG_{\mathbf{P}}$

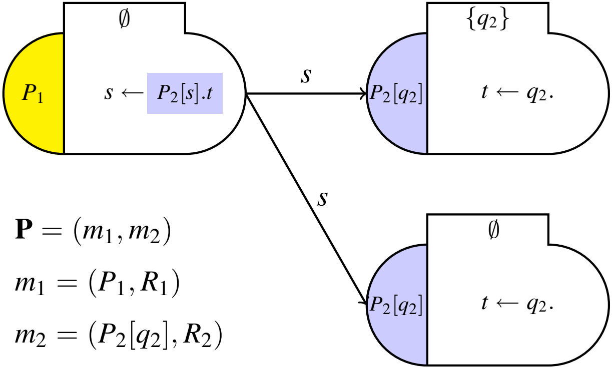


Call graphs and Relevant call graphs

The relevant call graph $CG_{\mathbf{P}}^{\mathbf{M}}$



Example



$\mathbf{P} = (m_1, m_2)$

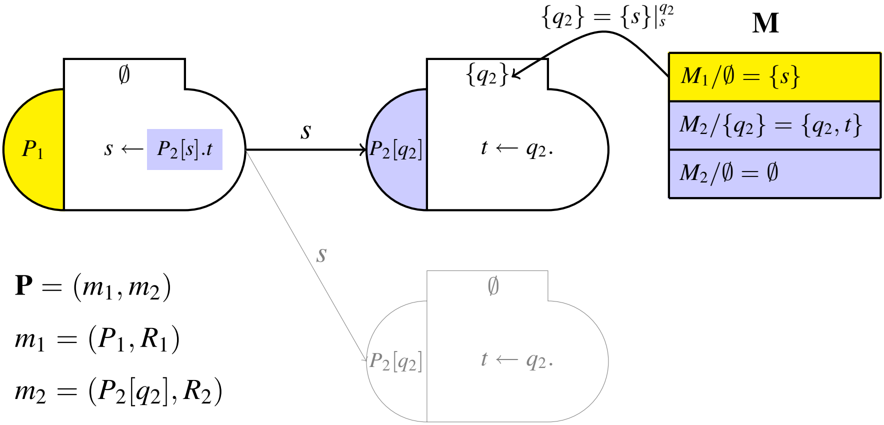
$m_1 = (P_1, R_1)$

$m_2 = (P_2[q_2], R_2)$

$R_1 = \{s \leftarrow P_2[s].t.\}$

$R_2 = \{t \leftarrow q_2.\}$

Example



$\mathbf{P} = (m_1, m_2)$

$m_1 = (P_1, R_1)$

$m_2 = (P_2[q_2], R_2)$

$R_1 = \{s \leftarrow P_2[s].t.\}$

$R_2 = \{t \leftarrow q_2.\}$

Generalization of the FLP-reduct to MLPs

Given an interpretation $\mathbf{M}$.

Let C be a set of value calls s.t. $V(CG_{\mathbf{P}}^{\mathbf{M}}) \subseteq C \subseteq VC(\mathbf{P})$

Generalization of the FLP-reduct to MLPs

Given an interpretation $\mathbf{M}$.

Let C be a set of value calls s.t. $V(CG_{\mathbf{P}}^{\mathbf{M}}) \subseteq C \subseteq VC(\mathbf{P})$

The FLP-reduct of $\mathbf{P}$ at $P[S]$ wrt. $\mathbf{M}$ and C is the rule set

$$f\mathbf{P}(P[S])^{\mathbf{M},C} = \begin{cases} \{r \in I_{gr(\mathbf{P})}(P[S]) \mid \mathbf{M}, P[S] \models B(r)\} & \text{if } P[S] \in C, \\ I_{gr(\mathbf{P})}(P[S]) & \text{otherwise.} \end{cases}$$

Generalization of the FLP-reduct to MLPs

Given an interpretation $\mathbf{M}$.

Let C be a set of value calls s.t. $V(CG_{\mathbf{P}}^{\mathbf{M}}) \subseteq C \subseteq VC(\mathbf{P})$

The FLP-reduct of $\mathbf{P}$ at $P[S]$ wrt. $\mathbf{M}$ and C is the rule set

$$f\mathbf{P}(P[S])^{\mathbf{M},C} = \begin{cases} \{r \in I_{gr(\mathbf{P})}(P[S]) \mid \mathbf{M}, P[S] \models B(r)\} & \text{if } P[S] \in C, \\ I_{gr(\mathbf{P})}(P[S]) & \text{otherwise.} \end{cases}$$

The FLP-reduct of $\mathbf{P}$ wrt. $\mathbf{M}$ and C is the rule base

$$f\mathbf{P}^{\mathbf{M},C} = (f\mathbf{P}(P[S])^{\mathbf{M},C} \mid P[S] \in VC(\mathbf{P})).$$

Generalization of the FLP-reduct to MLPs

Given an interpretation $\mathbf{M}$.

Let C be a set of value calls s.t. $V(CG_{\mathbf{P}}^{\mathbf{M}}) \subseteq C \subseteq VC(\mathbf{P})$

The FLP-reduct of $\mathbf{P}$ at $P[S]$ wrt. $\mathbf{M}$ and C is the rule set

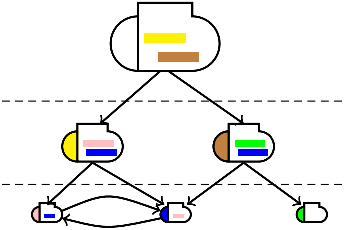
$$f\mathbf{P}(P[S])^{\mathbf{M},C} = \begin{cases} \{r \in I_{gr(\mathbf{P})}(P[S]) \mid \mathbf{M}, P[S] \models B(r)\} & \text{if } P[S] \in C, \\ I_{gr(\mathbf{P})}(P[S]) & \text{otherwise.} \end{cases}$$

The FLP-reduct of $\mathbf{P}$ wrt. $\mathbf{M}$ and C is the rule base

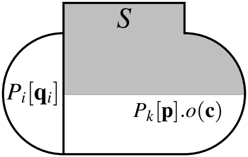
$$f\mathbf{P}^{\mathbf{M},C} = (f\mathbf{P}(P[S])^{\mathbf{M},C} \mid P[S] \in VC(\mathbf{P})).$$

$\mathbf{M}$ is an answer set of $\mathbf{P}$ wrt C for $\mathbf{M}$ iff $\mathbf{M}$ is a minimal model of $f\mathbf{P}^{\mathbf{M},C}$

MLP stratifications at two different levels

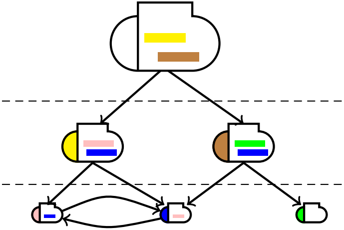


The global level along the relevant call graph

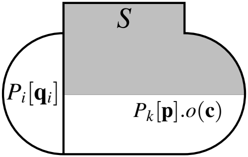


The local level (inside module instances) based on the (instance) dependency graph

MLP stratifications at two different levels



The global level along the relevant call graph

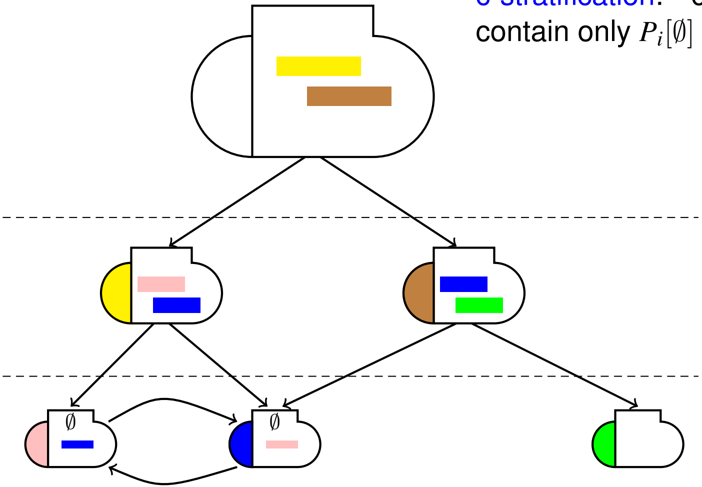


The local level (inside module instances) based on the (instance) dependency graph

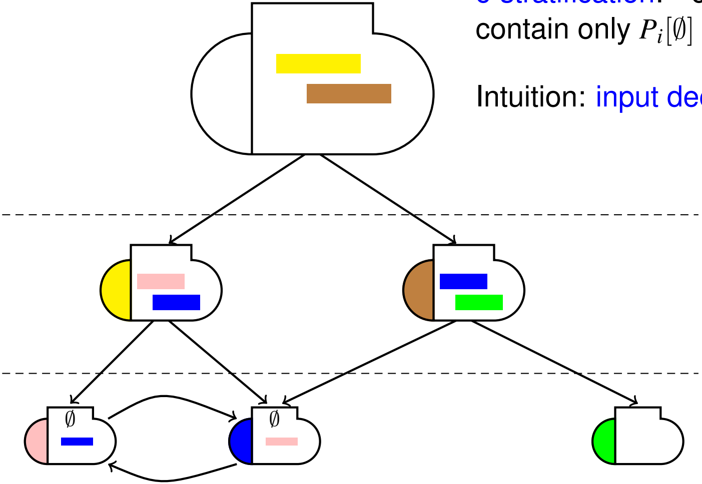
Given an interpretation $\mathbf{M}$ ...

call-stratification: (globally) split module layers

c-stratification: cycles in $CG_{\mathbf{P}}^{\mathbf{M}}$ contain only $P_i[\emptyset]$ nodes



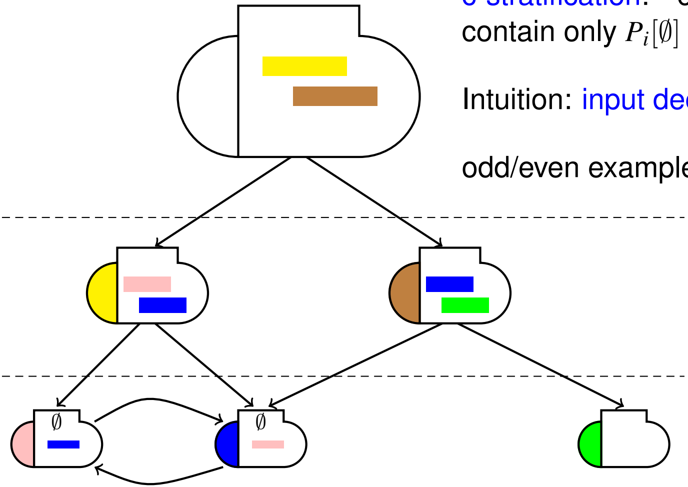
call-stratification: (globally) split module layers



c-stratification: cycles in $CG_{\mathbf{P}}^{\mathbf{M}}$ contain only $P_i[\emptyset]$ nodes

Intuition: input decreasing

call-stratification: (globally) split module layers



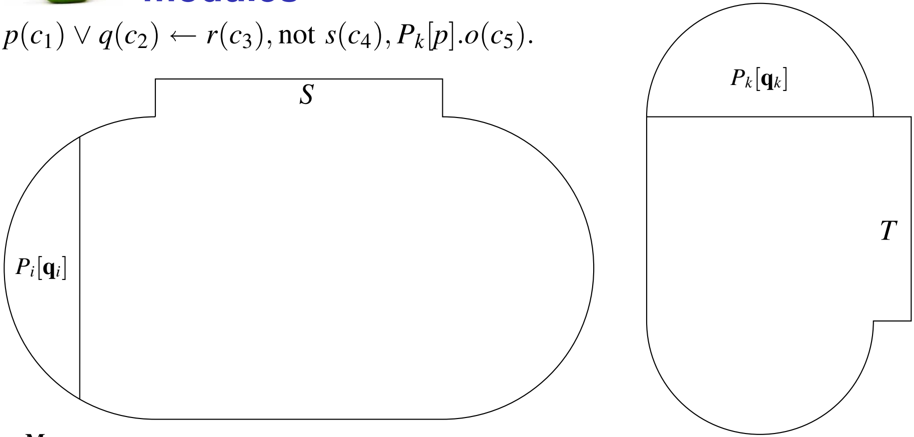
c-stratification: cycles in $CG_{\mathbf{P}}^{\mathbf{M}}$ contain only $P_i[\emptyset]$ nodes

Intuition: input decreasing

odd/even example: c-stratified

input-stratification: (locally) split inside modules

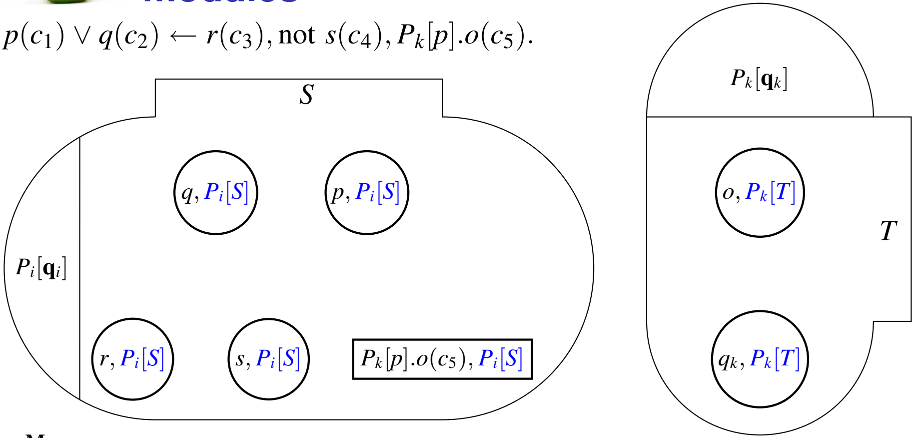
$p(c_1) \lor q(c_2) \leftarrow r(c_3), \text{not } s(c_4), P_k[p].o(c_5).$



$G_{\mathbf{P}}^{\mathbf{M}}$: the instance dependency graph

input-stratification: (locally) split inside modules

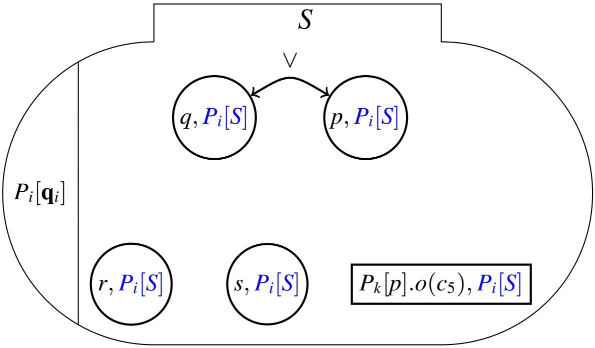
$p(c_1) \vee q(c_2) \leftarrow r(c_3), \text{not } s(c_4), P_k[p].o(c_5).$

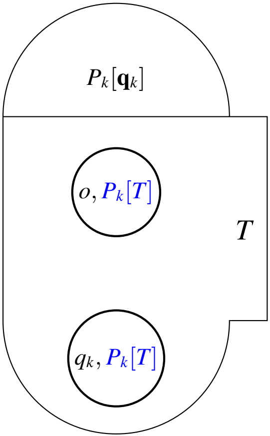


$G^{\mathbf{M}}_{\mathbf{P}}$: the instance dependency graph

input-stratification: (locally) split inside modules

$p(c_1) \vee q(c_2) \leftarrow r(c_3), \text{not } s(c_4), P_k[p].o(c_5).$

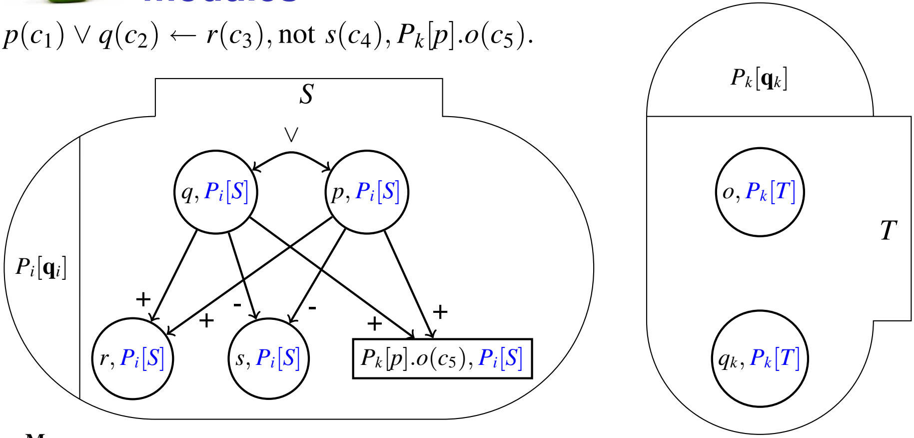




$G_{\mathbf{P}}^{\mathbf{M}}$: the instance dependency graph

input-stratification: (locally) split inside modules

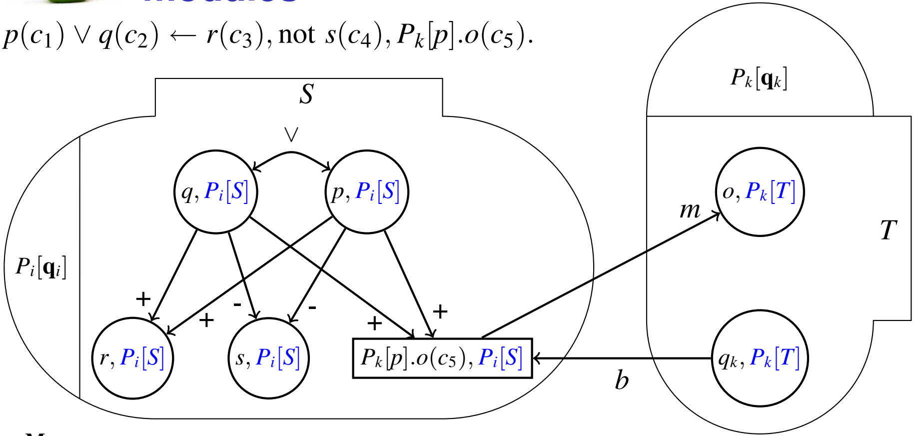
$p(c_1) \vee q(c_2) \leftarrow r(c_3), \text{not } s(c_4), P_k[p].o(c_5).$



$G_{\mathbf{P}}^{\mathbf{M}}$: the instance dependency graph

input-stratification: (locally) split inside modules

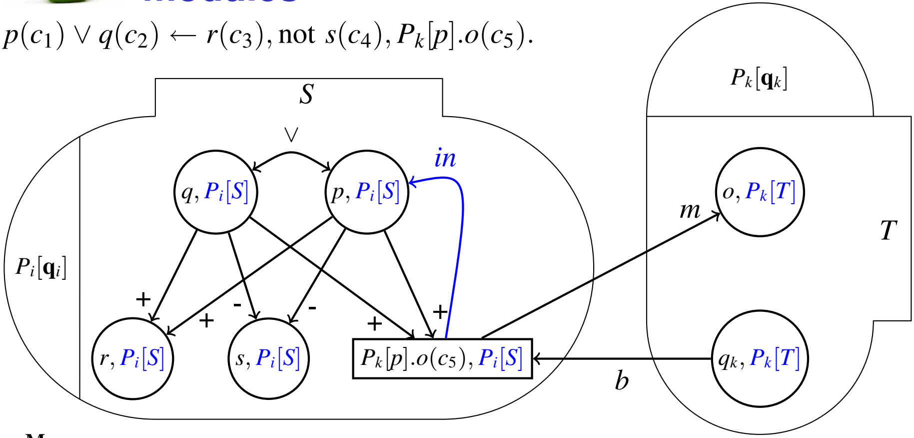
$p(c_1) \vee q(c_2) \leftarrow r(c_3), \text{not } s(c_4), P_k[p].o(c_5).$



$G_{\mathbf{P}}^{\mathbf{M}}$: the instance dependency graph

input-stratification: (locally) split inside modules

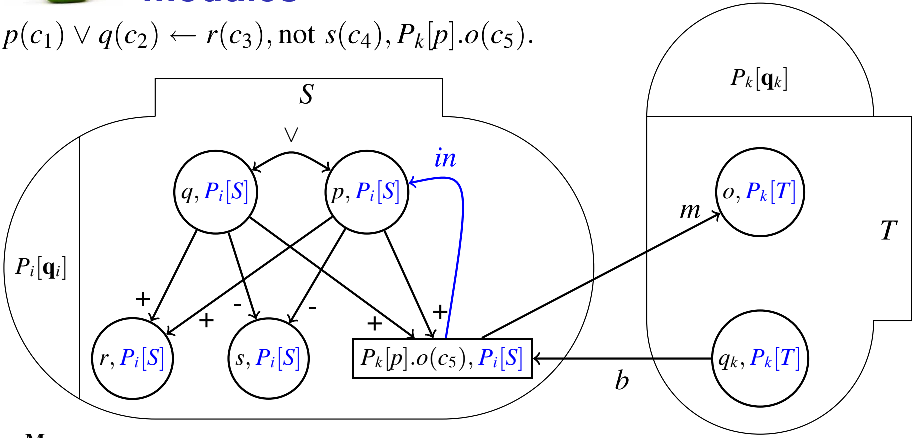
$p(c_1) \vee q(c_2) \leftarrow r(c_3), \text{not } s(c_4), P_k[p].o(c_5).$



$G_{\mathbf{P}}^{\mathbf{M}}$: the instance dependency graph

input-stratification: (locally) split inside modules

$p(c_1) \vee q(c_2) \leftarrow r(c_3), \text{not } s(c_4), P_k[p].o(c_5).$

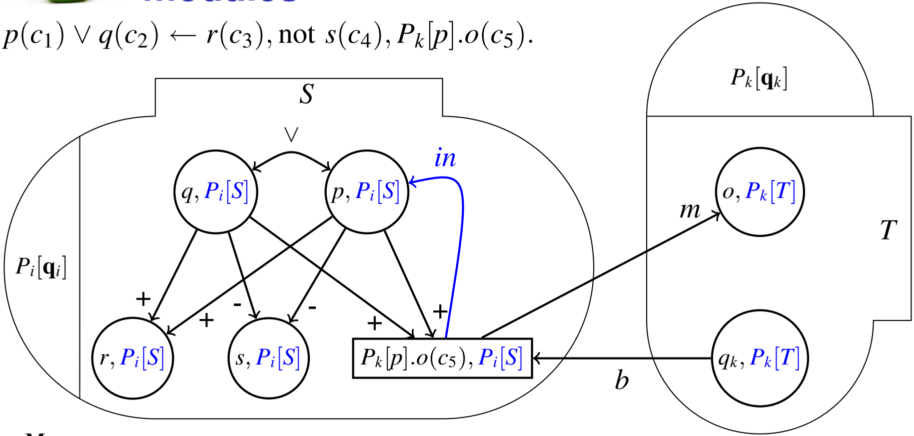


$G^{\mathbf{M}}_{\mathbf{P}}$: the instance dependency graph

i-stratification: cycles with *in*-edges in $G^{\mathbf{M}}_{\mathbf{P}}$ contain only $(X, P_i[\emptyset])$ nodes $\Rightarrow$ labels ill_i

input-stratification: (locally) split inside modules

$p(c_1) \vee q(c_2) \leftarrow r(c_3), \text{not } s(c_4), P_k[p].o(c_5).$



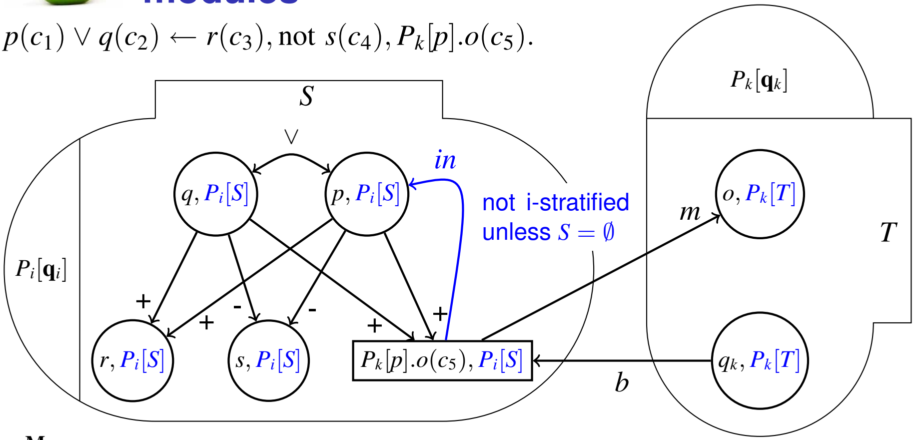
$G_{\mathbf{P}}^{\mathbf{M}}$: the instance dependency graph

i-stratification: cycles with *in*-edges in $G_{\mathbf{P}}^{\mathbf{M}}$ contain only $(X, P_i[\emptyset])$ nodes $\Rightarrow$ labels ill_i

odd/even example: i-stratified

input-stratification: (locally) split inside modules

$p(c_1) \vee q(c_2) \leftarrow r(c_3), \text{not } s(c_4), P_k[p].o(c_5).$



$G_{\mathbf{P}}^{\mathbf{M}}$: the instance dependency graph

i-stratification: cycles with *in*-edges in $G_{\mathbf{P}}^{\mathbf{M}}$ contain only $(X, P_i[\emptyset])$ nodes $\Rightarrow$ labels ill_i

odd/even example: i-stratified

Extending splitting sets to MLPs

- Given
  - $\mathbf{P}$: an MLP
  - R: set of ground rules
  - $\mathbf{p} = p_1, \ldots, p_\ell$: list of predicate names
- $def(\mathbf{p}, R) = \{p_i(\mathbf{d}) \mid \exists r \in R, p_i(\mathbf{d}) \in H(r), p_i \in \mathbf{p}\}$

Extending splitting sets to MLPs

- Given
  - $\mathbf{P}$: an MLP
  - R: set of ground rules
  - $\mathbf{p} = p_1, \ldots, p_\ell$: list of predicate names
- $def(\mathbf{p}, R) = \{p_i(\mathbf{d}) \mid \exists r \in R, p_i(\mathbf{d}) \in H(r), p_i \in \mathbf{p}\}$
- A splitting set of R is a set $U \subseteq HB_{\mathbf{P}}$ s.t.
  - for any $r \in R$, if $H(r) \cap U \neq \emptyset$ then $at(r) \subseteq U$
  - for each $P_k[\mathbf{p}].o(\mathbf{c}) \in U$ then $def(\mathbf{p}, R) \subseteq U$

Extending splitting sets to MLPs

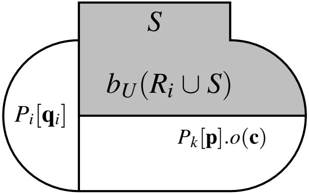
- Given
  - $\mathbf{P}$: an MLP
  - R: set of ground rules
  - $\mathbf{p} = p_1, \ldots, p_\ell$: list of predicate names
- $def(\mathbf{p}, R) = \{p_i(\mathbf{d}) \mid \exists r \in R, p_i(\mathbf{d}) \in H(r), p_i \in \mathbf{p}\}$
- A splitting set of R is a set $U \subseteq HB_{\mathbf{P}}$ s.t.
  - for any $r \in R$, if $H(r) \cap U \neq \emptyset$ then $at(r) \subseteq U$
  - for each $P_k[\mathbf{p}].o(\mathbf{c}) \in U$ then $def(\mathbf{p}, R) \subseteq U$
- U is an input splitting set of R for $\alpha = P_k[\mathbf{p}].o(\mathbf{c})$ iff $\alpha \notin U$ and $def(\mathbf{p}, R) \subseteq U$

Extending splitting sets to MLPs

- Given
  - $\mathbf{P}$: an MLP
  - R: set of ground rules
  - $\mathbf{p} = p_1, \ldots, p_\ell$: list of predicate names
- $def(\mathbf{p}, R) = \{p_i(\mathbf{d}) \mid \exists r \in R, p_i(\mathbf{d}) \in H(r), p_i \in \mathbf{p}\}$
- A splitting set of R is a set $U \subseteq HB_{\mathbf{P}}$ s.t.
  - for any $r \in R$, if $H(r) \cap U \neq \emptyset$ then $at(r) \subseteq U$
  - for each $P_k[\mathbf{p}].o(\mathbf{c}) \in U$ then $def(\mathbf{p}, R) \subseteq U$
- U is an input splitting set of R for $\alpha = P_k[\mathbf{p}].o(\mathbf{c})$ iff $\alpha \notin U$ and $def(\mathbf{p}, R) \subseteq U$
- Bottom $b_U(R) = \{r \in R \mid H(r) \cap U \neq \emptyset\}$

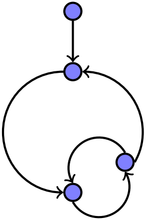
Extending splitting sets to MLPs

- Given
  - $\mathbf{P}$: an MLP
  - R: set of ground rules
  - $\mathbf{p} = p_1, \ldots, p_\ell$: list of predicate names

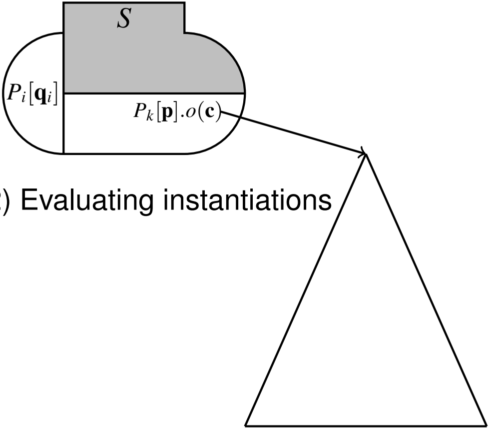


- $def(\mathbf{p}, R) = \{p_i(\mathbf{d}) \mid \exists r \in R, p_i(\mathbf{d}) \in H(r), p_i \in \mathbf{p}\}$
- A splitting set of R is a set $U \subseteq HB_{\mathbf{P}}$ s.t.
  - for any $r \in R$, if $H(r) \cap U \neq \emptyset$ then $at(r) \subseteq U$
  - for each $P_k[\mathbf{p}].o(\mathbf{c}) \in U$ then $def(\mathbf{p}, R) \subseteq U$
- U is an input splitting set of R for $\alpha = P_k[\mathbf{p}].o(\mathbf{c})$ iff $\alpha \notin U$ and $def(\mathbf{p}, R) \subseteq U$
- Bottom $b_U(R) = \{r \in R \mid H(r) \cap U \neq \emptyset\}$

Top-down evaluation algorithm

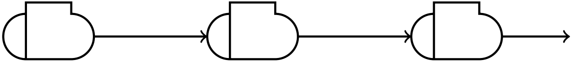


(1) Detecting cycles



(2) Evaluating instantiations

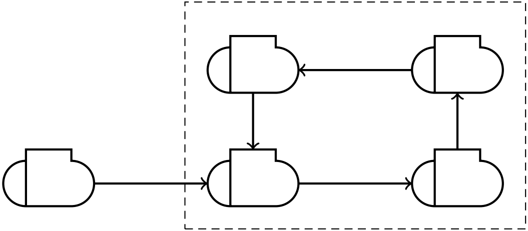
Detecting cycles



Done by maintaining a “path” of sets of visited value calls.

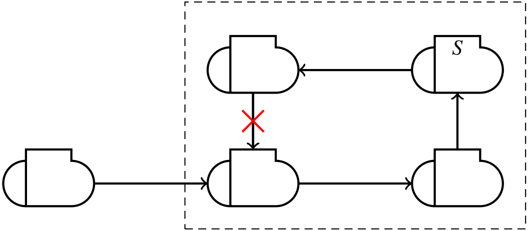
Each element of “path” is initialized with a single value call.

Detecting cycles



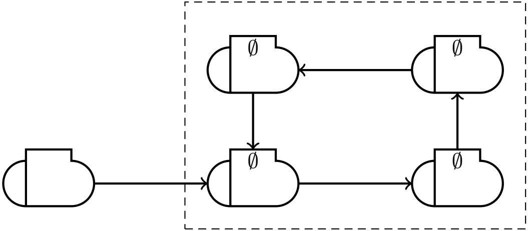
When a cycle is detected, there are two cases:

Detecting cycles



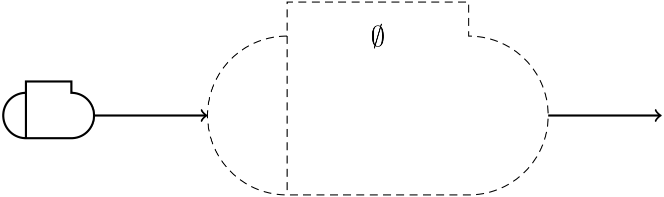
(1) A member of the cycle has non-empty input, we backtrack.

Detecting cycles



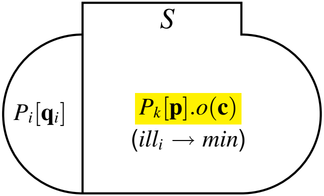
(2) All members of the cycle have empty input

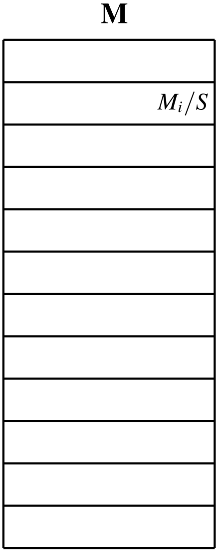
Detecting cycles



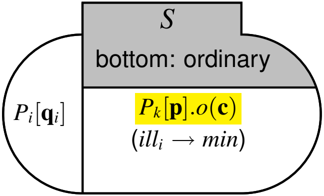
We combine all value calls of the cycle and treat the result as a new "virtual" value call.

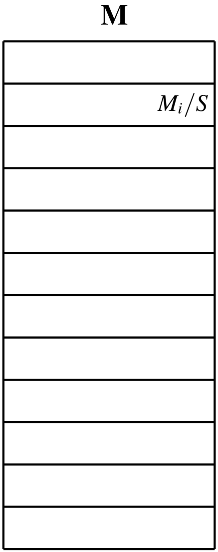
Evaluating module instances



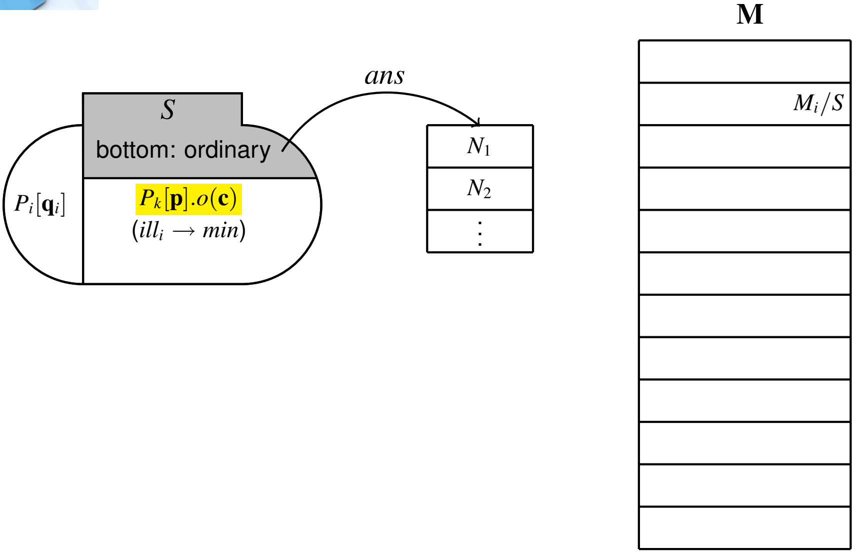


Evaluating module instances

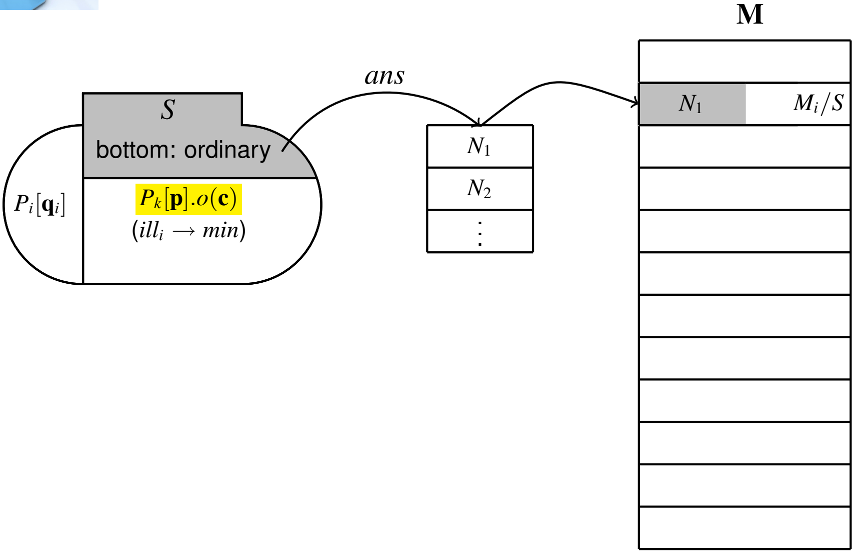




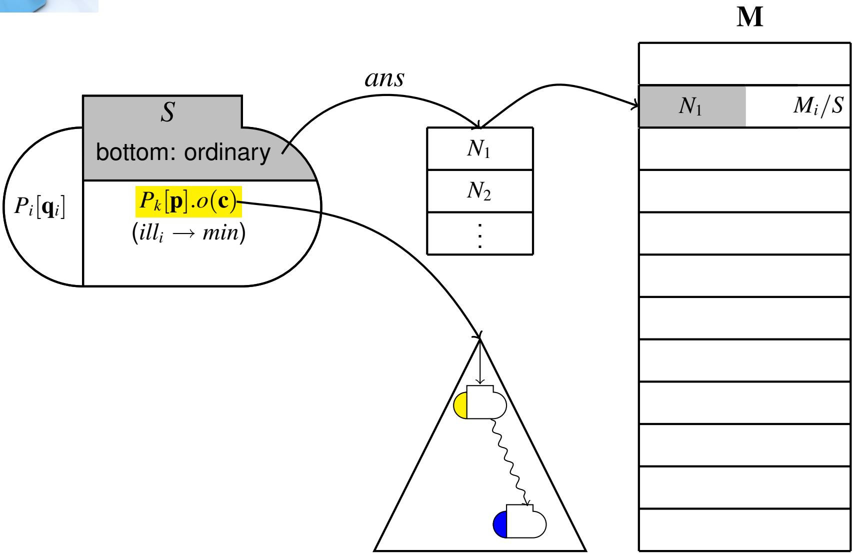
Evaluating module instances



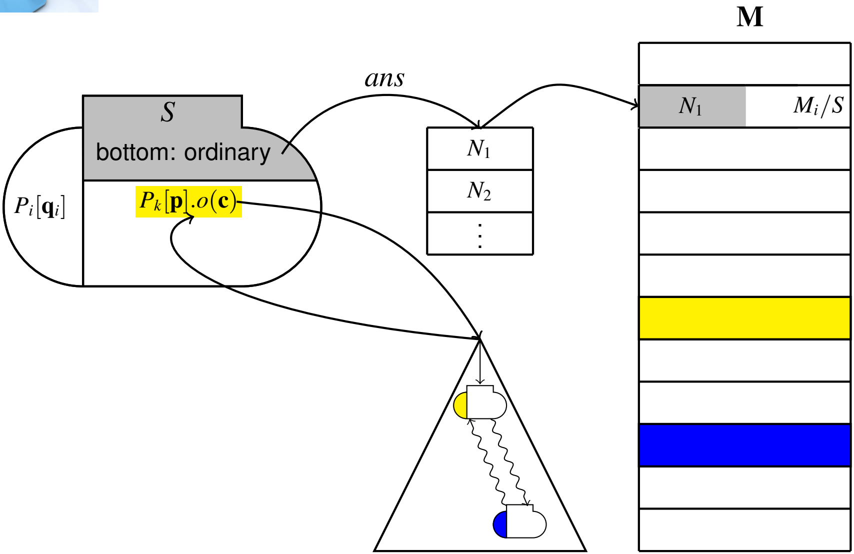
Evaluating module instances



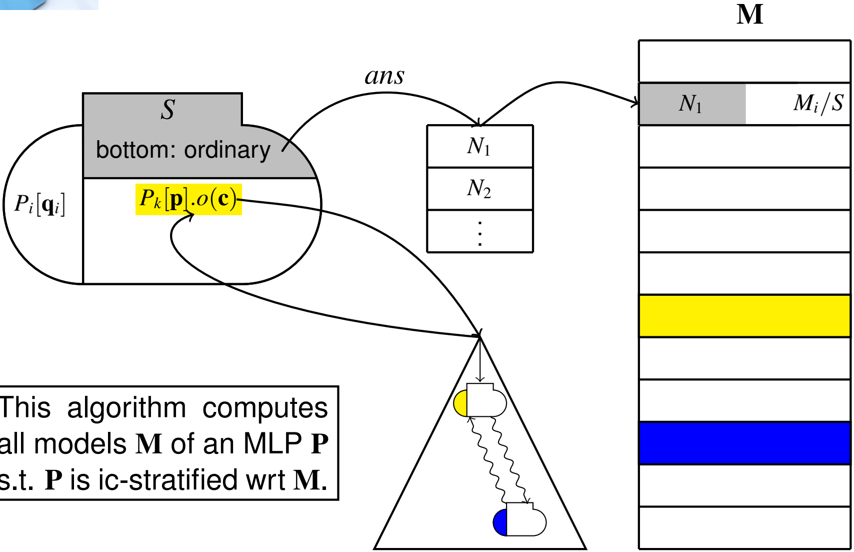
Evaluating module instances



Evaluating module instances



Evaluating module instances



This algorithm computes all models $\mathbf{M}$ of an MLP $\mathbf{P}$ s.t. $\mathbf{P}$ is ic-stratified wrt $\mathbf{M}$.

Conclusion

We investigated

- Different notions of stratification for MLPs
  - call-stratification (global - along the relevant call graph)
  - input-stratification (local - inside modul instances)
- Generalize the Splitting theorem for ic-stratified MLPs
- A top-down algorithm for evaluating ic-stratified MLPs

Details in the paper:

- i-stratification at the schematic level
- safety condition

Future work

- Extension on stratification
- Implementation

References I

Chitta Baral, Juraj Dzifcak, and Hiro Takahashi.
Macros, Macro calls and Use of Ensembles in Modular Answer Set Programming.
In *Proceedings of the 22th International Conference on Logic Programming (ICLP 2006)*, number 4079 in LNCS, pages 376–390. Springer, 2006.

Minh Dao-Tran, Thomas Eiter, Michael Fink, and Thomas Krennwallner.
Modular Nonmonotonic Logic Programming Revisited.
In Patricia M. Hill and David S. Warren, editors, *25th International Conference on Logic Programming (ICLP 2009), Pasadena, California, USA, July 14–17, 2009*, volume 5649 of *LNCS*, pages 145–159. Springer, July 2009.

References II

Thomas Eiter, Georg Gottlob, and Helmuth Veith.
Modular Logic Programming and Generalized Quantifiers.
In *Proceedings of the 4th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-1997)*, volume 1265 of *LNCS*, pages 290–309. Springer, 1997.

Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Max Ostrowski, Torsten Schaub, and Sven Thiele.
Engineering an incremental asp solver.
In *International Conference on Logic Programming (ICLP 2008)*, pages 190–205. Springer, 2008.

References III

- Giovambattista Ianni, Guiseppe Ielpa, Adriana Pietramala, and Maria Carmela Santoro.
  Answer Set Programming with Templates.
  In *Proceedings of the 2nd International Answer Set Programming Workshop (ASP'03)*, CEUR Workshop Proceedings. CEUR WS, 2003.
- Tomi Janhunen, Emilia Oikarinen, Hans Tompits, and Stefan Woltran.
  Modularity Aspects of Disjunctive Stable Models.
  In *Proceedings of the 9th International Conference on Logic Programming and Nonmonotonic Reasoning*, volume 4483 of *LNCS*, pages 175–187. Springer, May 2007.

References IV

Vladimir Lifschitz and Hudson Turner.
Splitting a Logic Program.
In *Proceedings of the 11th International Conference on Logic Programming (ICLP 1994)*, pages 23–37. MIT Press, June 1994.

Emilia Oikarinen and Tomi Janhunen.
Achieving compositionality of the stable model semantics for Smodels programs.
Theory and Practice of Logic Programming, 8(5–6):717–761, November 2008.