Knowledge Compilation of Logic Programs Using Approximation Fixpoint Theory

Bart Bogaerts and Guy Van den Broeck
Presenter: Joost Vennekens

ICLP 2015

Content

Content

Knowledge Compilation: What?

- Given:
  - Theory $\mathcal{T}$ in language $\mathcal{L}$.
- Find:
  - Equivalent theory $\mathcal{T}'$ in language $\mathcal{L}'$ with attractive properties.

Knowledge Compilation: What?

- Given:
  - Theory $\mathcal{T}$ in language $\mathcal{L}$.
- Find:
  - Equivalent theory $\mathcal{T}'$ in language $\mathcal{L}'$ with attractive properties. E.g., if $\mathcal{L}'$ is the language of SDDs: the following inference methods are polytime:
    - Validity checking,
    - Consistency checking,
    - Equivalence checking,
    - Model enumeration,
    - (Weighted) model counting.
    - . . .

Knowledge Compilation: Why?

- Compile once, evaluate often.
- Reuse (off-line) compilation.
- Reduction of problems in $\mathcal{L}$ to problems in $\mathcal{L}'$.

Knowledge Compilation: How?

- In this talk, $\mathcal{L}$ is the language of propositional logic programs (parametrised well-founded semantics)
- $\mathcal{L}'$ is the language of SDDs (Sentential Decision Diagrams) (but it could be any representation of propositional formulas)
- State of the art:
  - Transform logic program $\mathcal{P}$ to CNF (completion + loop breaking formulas)
  - Transform the CNF to an SDD

Knowledge Compilation: How?

- In this talk, $\mathcal{L}$ is the language of propositional logic programs (parametrised well-founded semantics)
- $\mathcal{L}'$ is the language of SDDs (Sentential Decision Diagrams) (but it could be any representation of propositional formulas)
- State of the art:
  - Transform logic program $\mathcal{P}$ to CNF (completion + loop breaking formulas)
  - Transform the CNF to an SDD
- Disadvantages
  - Many auxiliary variables are introduced
  - Expensive pre-processing (loop breaking) not always feasible

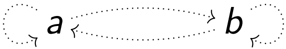
Example

$$\left\{ \begin{array}{l} R(x,y) \leftarrow E(x,y). \\ R(x,y) \leftarrow E(x,z) \wedge R(z,y). \end{array} \right\}$$

$$\begin{tikzcd} a \arrow[loop left, dotted] \arrow[r, bend left, dotted] & b \arrow[l, bend left, dotted] \arrow[loop right, dotted] \end{tikzcd}$$

Example

$$\left\{ \begin{array}{l} R(x,y) \leftarrow E(x,y). \\ R(x,y) \leftarrow E(x,z) \wedge R(z,y). \end{array} \right\}$$



$$R(a,a) \Leftarrow E(a,a) \vee (E(a,b) \wedge R(b,a)) \vee (E(a,a) \wedge R(a,a)).$$
$$R(a,b) \Leftarrow E(a,b) \vee (E(a,a) \wedge R(a,b)) \vee (E(a,b) \wedge R(b,b)).$$
$$R(b,b) \Leftarrow E(b,b) \vee (E(b,a) \wedge R(a,b)) \vee (E(b,b) \wedge R(b,b)).$$
$$R(b,a) \Leftarrow E(b,a) \vee (E(b,b) \wedge R(b,a)) \vee (E(b,a) \wedge R(a,a)).$$
$$R(a,a) \Rightarrow E(a,a) \vee (E(a,b) \wedge R(b,a) \wedge T_1) \vee (E(a,a) \wedge R(a,a) \wedge \mathbf{f}).$$
$$R(a,b) \Rightarrow E(a,b) \vee (E(a,a) \wedge R(a,b) \wedge \mathbf{f}) \vee (E(a,b) \wedge R(b,b) \wedge T_2).$$
$$R(b,b) \Rightarrow E(b,b) \vee (E(b,a) \wedge R(a,b) \wedge \neg T_2) \vee (E(b,b) \wedge R(b,b) \wedge \mathbf{f}).$$
$$R(b,a) \Rightarrow E(b,a) \vee (E(b,b) \wedge R(b,a) \wedge \mathbf{f}) \vee (E(b,a) \wedge R(a,a) \wedge \neg T_1).$$

Example

$$\left\{ \begin{array}{l} R(x,y) \leftarrow E(x,y). \\ R(x,y) \leftarrow E(x,z) \wedge R(z,y). \end{array} \right\}$$

$$\begin{tikzcd} a \arrow[loop left, dotted] \arrow[r, bend left, dotted] & b \arrow[l, bend left, dotted] \arrow[loop right, dotted] \end{tikzcd}$$

$$R(a,a) \Leftrightarrow E(a,a) \vee (E(a,b) \wedge E(b,a)).$$
$$R(a,b) \Leftrightarrow E(a,b).$$
$$R(b,a) \Leftrightarrow E(b,a).$$
$$R(b,b) \Leftrightarrow E(b,b) \vee (E(b,a) \wedge E(a,b)).$$

Situation

- IJCAI'15 "Anytime inference in probabilistic logic programs with $T_{\mathcal{P}}$-compilation" (Vlasselaer et al):
  - New *bottom-up* knowledge compilation for positive logic programs
  - Improved efficiency
  - Enables *approximate inference*
- This paper:
  - Generalisation for *generalised logic program under parametrised well-founded semantics*
  - Generalisation to algebraical setting (*Approximation Fixpoint Theory*)

Content

Reachability

$$\left\{\begin{array}{l} R(x,y) \leftarrow E(x,y). \\ R(x,y) \leftarrow E(x,z) \wedge R(z,y). \end{array}\right\}$$

Reachability

$$\left\{\begin{array}{l} R(x,y) \leftarrow E(x,y). \\ R(x,y) \leftarrow E(x,z) \wedge R(z,y). \end{array}\right\}$$

$$\begin{tikzcd} a \arrow[rr] & & b \arrow[dl] \\ & c & \end{tikzcd}$$

Reachability

$$\left\{ \begin{array}{l} R(x,y) \leftarrow E(x,y). \\ R(x,y) \leftarrow E(x,z) \wedge R(z,y). \end{array} \right\}$$

$$\begin{array}{ccc} a & \longrightarrow & b \\ & \swarrow & \\ c & & \end{array}$$

$$\begin{aligned} R(a,b) &= \mathbf{f} \\ R(b,a) &= \mathbf{f} \\ R(a,c) &= \mathbf{f} \\ R(c,a) &= \mathbf{f} \\ R(b,c) &= \mathbf{f} \\ R(c,b) &= \mathbf{f} \\ &\dots \end{aligned}$$

Reachability

$$\left\{ \begin{array}{l} R(x,y) \leftarrow E(x,y). \\ R(x,y) \leftarrow E(x,z) \wedge R(z,y). \end{array} \right\}$$

$$\begin{tikzcd} a \arrow[r] & b \arrow[dl] \\ c & \end{tikzcd}$$

$R(a,b) =$	**f**	**t**
$R(b,a) =$	**f**	**f**
$R(a,c) =$	**f**	**f**
$R(c,a) =$	**f**	**f**
$R(b,c) =$	**f**	**t**
$R(c,b) =$	**f**	**f**
$\dots$		

Reachability

$$\left\{\begin{array}{l} R(x,y) \leftarrow E(x,y). \\ R(x,y) \leftarrow E(x,z) \wedge R(z,y). \end{array}\right\}$$

$$\begin{array}{ccc} a & \longrightarrow & b \\ & c \swarrow & \end{array}$$

$R(a,b) =$	**f**	**t**	**t**
$R(b,a) =$	**f**	**f**	**f**
$R(a,c) =$	**f**	**f**	**t**
$R(c,a) =$	**f**	**f**	**f**
$R(b,c) =$	**f**	**t**	**t**
$R(c,b) =$	**f**	**f**	**f**
$\dots$			

Reachability

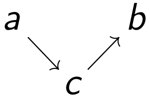
$$\left\{ \begin{array}{l} R(x,y) \leftarrow E(x,y). \\ R(x,y) \leftarrow E(x,z) \wedge R(z,y). \end{array} \right\}$$

$$\begin{tikzcd} a \arrow[dr] & & b \\ & c \arrow[ur] & \end{tikzcd}$$

$$\begin{aligned} R(a,b) &= && \mathbf{f} \\ R(b,a) &= && \mathbf{f} \\ R(a,c) &= && \mathbf{f} \\ R(c,a) &= && \mathbf{f} \\ R(b,c) &= && \mathbf{f} \\ R(c,b) &= && \mathbf{f} \\ &\ldots \end{aligned}$$

Reachability

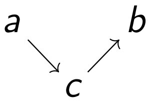
$$\left\{ \begin{array}{l} R(x,y) \leftarrow E(x,y). \\ R(x,y) \leftarrow E(x,z) \wedge R(z,y). \end{array} \right\}$$



$R(a,b) =$	**f**	**f**
$R(b,a) =$	**f**	**f**
$R(a,c) =$	**f**	**t**
$R(c,a) =$	**f**	**f**
$R(b,c) =$	**f**	**f**
$R(c,b) =$	**f**	**t**
$\dots$		

Reachability

$$\left\{ \begin{array}{l} R(x,y) \leftarrow E(x,y). \\ R(x,y) \leftarrow E(x,z) \wedge R(z,y). \end{array} \right\}$$



$R(a,b) =$	**f**	**f**	**t**
$R(b,a) =$	**f**	**f**	**f**
$R(a,c) =$	**f**	**t**	**t**
$R(c,a) =$	**f**	**f**	**f**
$R(b,c) =$	**f**	**f**	**f**
$R(c,b) =$	**f**	**t**	**t**
$\dots$			

Observations

- For every graph: similar process.
- Lots of overlap.
- Least fixpoint computation = sequence of interpretations.
- Idea: Generalise this. Execute this fixpoint computation once, symbolically.

Parametrised well-founded semantics

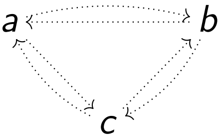
- $\mathcal{P}$: parametrised logic program
  - Σ_d: defined symbols
  - Σ_p: parameter symbols
- For every Σ_p-interpretation I: $\mathcal{P}$ defines the well-founded model of $\mathcal{P}$ in context I (a Σ_d-interpretation), denoted $WFM(\mathcal{P}, I)$
- $=$ Parametrised well-founded semantics

Symbolic Interpretations

- Σ_d-interpretation: mapping $\Sigma_d \to \{\mathbf{t}, \mathbf{f}\}$: state in the least fixpoint computation
- Symbolic Σ_d-interpretation $\mathcal{A}$: mapping $\Sigma_d \to \{\text{formulas over } \Sigma_p\}$ (modulo equivalence)
- Will be a state in a symbolic least fixpoint computation

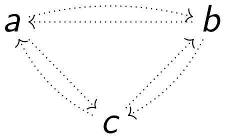
Symbolic Least Fixpoint Computation

$$\left\{ \begin{array}{l} R(x, y) \leftarrow E(x, y). \\ R(x, y) \leftarrow E(x, z) \wedge R(z, y). \end{array} \right\}$$



Symbolic Least Fixpoint Computation

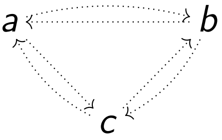
$$\left\{ \begin{array}{l} R(x,y) \leftarrow E(x,y). \\ R(x,y) \leftarrow E(x,z) \wedge R(z,y). \end{array} \right\}$$



$$\begin{aligned} R(a,b) &= \quad \mathbf{f} \\ R(b,a) &= \quad \mathbf{f} \\ R(c,a) &= \quad \mathbf{f} \\ &\dots \end{aligned}$$

Symbolic Least Fixpoint Computation

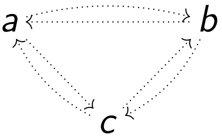
$$\left\{ \begin{array}{l} R(x,y) \leftarrow E(x,y). \\ R(x,y) \leftarrow E(x,z) \wedge R(z,y). \end{array} \right\}$$



$$\begin{aligned} R(a,b) &= \quad \mathbf{f} \quad E(a,b) \\ R(b,a) &= \quad \mathbf{f} \quad E(b,a) \\ R(c,a) &= \quad \mathbf{f} \quad E(c,a) \\ &\ldots \end{aligned}$$

Symbolic Least Fixpoint Computation

$$\left\{ \begin{array}{l} R(x,y) \leftarrow E(x,y). \\ R(x,y) \leftarrow E(x,z) \wedge R(z,y). \end{array} \right\}$$



$$\begin{array}{lllll}
R(a,b) = & \mathbf{f} & E(a,b) & E(a,b) \vee (E(a,c) \wedge E(c,a)) \vee \ldots \\
R(b,a) = & \mathbf{f} & E(b,a) & E(b,a) \vee (E(b,c) \wedge E(c,a)) \vee \ldots \\
R(c,a) = & \mathbf{f} & E(c,a) & E(c,a) \vee (E(c,b) \wedge E(b,a)) \vee \ldots \\
\quad \ldots & & &
\end{array}$$

Content

Negation

- Least fixpoint computation only works for monotone logic programs.
- What about negation?
- For the standard (non-parametrised) well-founded semantics: solved by Van Gelder, Ross and Schlipf (1991).

Well-founded model construction

$$\left\{\begin{array}{l} p \leftarrow q \vee s \\ q \leftarrow p \\ r \leftarrow \neg p \end{array}\right\}$$

Well-founded model construction

$$\left\{ \begin{array}{l} p \leftarrow q \vee s \\ q \leftarrow p \\ r \leftarrow \neg p \end{array} \right\}$$

$$s = \mathbf{t}$$

Well-founded model construction

$$\left\{ \begin{array}{l} p \leftarrow q \vee s \\ q \leftarrow p \\ r \leftarrow \neg p \end{array} \right\}$$

$$s = \mathbf{t}$$

$$\begin{aligned} p &= \mathbf{u} \\ q &= \mathbf{u} \\ r &= \mathbf{u} \\ &\dots \end{aligned}$$

Well-founded model construction

$$\left\{ \begin{array}{l} p \leftarrow q \vee s \\ q \leftarrow p \\ r \leftarrow \neg p \end{array} \right\}$$

$s = \mathbf{t}$

$p =$	$\mathbf{u}$	$\mathbf{t}$
$q =$	$\mathbf{u}$	$\mathbf{u}$
$r =$	$\mathbf{u}$	$\mathbf{u}$
$\ldots$		

Rule application

Well-founded model construction

$$\left\{ \begin{array}{l} p \leftarrow q \vee s \\ q \leftarrow p \\ r \leftarrow \neg p \end{array} \right\}$$

$$s = \mathbf{t}$$

$p =$	**u**	**t**	**t**
$q =$	**u**	**u**	**t**
$r =$	**u**	**u**	**u**
$\dots$			

Rule application

Well-founded model construction

$$\left\{ \begin{array}{l} p \leftarrow q \vee s \\ q \leftarrow p \\ r \leftarrow \neg p \end{array} \right\} \qquad s = \mathbf{t}$$

$$\begin{array}{lllll} p = & \mathbf{u} & \mathbf{t} & \mathbf{t} & \mathbf{t} \\ q = & \mathbf{u} & \mathbf{u} & \mathbf{t} & \mathbf{t} \\ r = & \mathbf{u} & \mathbf{u} & \mathbf{u} & \mathbf{f} \\ \ldots & & & & \end{array}$$

Rule application

Well-founded model construction

$$\left\{ \begin{array}{l} p \leftarrow q \vee s \\ q \leftarrow p \\ r \leftarrow \neg p \end{array} \right\}$$

$$s = \mathbf{f}$$

$$\begin{aligned} p &= \quad \mathbf{u} \\ q &= \quad \mathbf{u} \\ r &= \quad \mathbf{u} \\ &\dots \end{aligned}$$

Well-founded model construction

$$\left\{\begin{array}{l} p \leftarrow q \vee s \\ q \leftarrow p \\ r \leftarrow \neg p \end{array}\right\}$$

$$s = \mathbf{f}$$

$p =$	$\mathbf{u}$	$\mathbf{f}$
$q =$	$\mathbf{u}$	$\mathbf{f}$
$r =$	$\mathbf{u}$	$\mathbf{u}$
$\ldots$		

Unfounded set

Well-founded model construction

$$\left\{ \begin{array}{l} p \leftarrow q \vee s \\ q \leftarrow p \\ r \leftarrow \neg p \end{array} \right\}$$

$s = \mathbf{f}$

$p =$	**u**	**f**	**f**
$q =$	**u**	**f**	**f**
$r =$	**u**	**u**	**t**
...			

Rule application

Observations

- For every Σ_p-interpretation: similar process
- Lots of overlap
- Well-founded model computation $=$ sequence of partial interpretations
- Idea: Generalise this. Execute this fixpoint computation once, symbolically.

Parametrised well-founded semantics

- $\mathcal{P}$: parametrised logic program
  - Σ_d: defined symbols
  - Σ_p: parameter symbols
- For every Σ_p-interpretation I: $\mathcal{P}$ defines the well-founded model of $\mathcal{P}$ in context I (a Σ_d-interpretation), denoted $WFM(\mathcal{P}, I)$
- $=$ Parametrised well-founded semantics

Parametrised well-founded semantics

- $\mathcal{P}$: parametrised logic program
  - Σ_d: defined symbols
  - Σ_p: parameter symbols
- For every Σ_p-interpretation I: $\mathcal{P}$ defines the well-founded model of $\mathcal{P}$ in context I (a Σ_d-interpretation), denoted $WFM(\mathcal{P}, I)$
- $=$ Parametrised well-founded semantics
- Now: compute $PWFM(\mathcal{P})$

Symbolic Partial Interpretations

- Partial interpretation: mapping $\Sigma_d \to \{\mathbf{t}, \mathbf{f}, \mathbf{u}(, \mathbf{i})\}$: state in the well-founded model computation.
- Alternative representation: mapping $\Sigma_d \to \{\mathbf{t}, \mathbf{f}\} \times \{\mathbf{t}, \mathbf{f}\}$ (certain, possible)
  - $(\mathbf{t}, \mathbf{t}) = \mathbf{t}$
  - $(\mathbf{f}, \mathbf{f}) = \mathbf{f}$
  - $(\mathbf{f}, \mathbf{t}) = \mathbf{u}$
  - $(\mathbf{t}, \mathbf{f}) = \mathbf{i}$
- Symbolic partial interpretation:
  mapping $\Sigma_d \to \{\text{formulas over } \Sigma_p\} \times \{\text{formulas over } \Sigma_p\}$ (modulo equivalence)
- Will be a state in a symbolic well-founded model computation

Symbolic well-founded model construction

$$\left\{ \begin{array}{l} p \leftarrow q \vee s \\ q \leftarrow p \\ r \leftarrow \neg p \end{array} \right\}$$

Symbolic well-founded model construction

$$\left\{ \begin{array}{l} p \leftarrow q \vee s \\ q \leftarrow p \\ r \leftarrow \neg p \end{array} \right\}$$

$$\begin{aligned}
p_c &= \quad \mathbf{f} \\
p_p &= \quad \mathbf{t} \\
q_c &= \quad \mathbf{f} \\
q_p &= \quad \mathbf{t} \\
r_c &= \quad \mathbf{f} \\
r_p &= \quad \mathbf{t} \\
&\ldots
\end{aligned}$$

Symbolic well-founded model construction

$$\left\{ \begin{array}{l} p \leftarrow q \vee s \\ q \leftarrow p \\ r \leftarrow \neg p \end{array} \right\}$$

$$\begin{array}{lcr} p_c = & \mathbf{f} & \color{red}{\mathbf{f} \vee s} \\ p_p = & \mathbf{t} & \mathbf{t} \\ q_c = & \mathbf{f} & \mathbf{f} \\ q_p = & \mathbf{t} & \mathbf{t} \\ r_c = & \mathbf{f} & \mathbf{f} \\ r_p = & \mathbf{t} & \mathbf{t} \\ \ldots & & \end{array}$$

Rule application

Symbolic well-founded model construction

$$\left\{ \begin{array}{l} p \leftarrow q \vee s \\ q \leftarrow p \\ r \leftarrow \neg p \end{array} \right\}$$

$p_c =$	$\mathbf{f}$	s	s
$p_p =$	$\mathbf{t}$	$\mathbf{t}$	$\mathbf{t}$
$q_c =$	$\mathbf{f}$	$\mathbf{f}$	s
$q_p =$	$\mathbf{t}$	$\mathbf{t}$	$\mathbf{t}$
$r_c =$	$\mathbf{f}$	$\mathbf{f}$	$\mathbf{f}$
$r_p =$	$\mathbf{t}$	$\mathbf{t}$	$\neg s$
$\ldots$			

Rule application

Symbolic well-founded model construction

$$\left\{ \begin{array}{l} p \leftarrow q \vee s \\ q \leftarrow p \\ r \leftarrow \neg p \end{array} \right\}$$

$$\begin{array}{lllll} p_c = & \mathbf{f} & s & s & s \\ p_p = & \mathbf{t} & \mathbf{t} & \mathbf{t} & s \\ q_c = & \mathbf{f} & \mathbf{f} & s & s \\ q_p = & \mathbf{t} & \mathbf{t} & \mathbf{t} & s \\ r_c = & \mathbf{f} & \mathbf{f} & \mathbf{f} & \mathbf{f} \\ r_p = & \mathbf{t} & \mathbf{t} & \neg s & \neg s \\ \quad \ldots & & & & \end{array}$$

Unfounded set

Symbolic well-founded model construction

$$\left\{ \begin{array}{l} p \leftarrow q \vee s \\ q \leftarrow p \\ r \leftarrow \neg p \end{array} \right\}$$

$$\begin{array}{llllll}
p_c = & \mathbf{f} & s & s & s & s \\
p_p = & \mathbf{t} & \mathbf{t} & \mathbf{t} & s & s \\
q_c = & \mathbf{f} & \mathbf{f} & s & s & s \\
q_p = & \mathbf{t} & \mathbf{t} & \mathbf{t} & s & s \\
r_c = & \mathbf{f} & \mathbf{f} & \mathbf{f} & \mathbf{f} & \neg s \\
r_p = & \mathbf{t} & \mathbf{t} & \neg s & \neg s & \neg s \\
\ldots & & & & &
\end{array}$$

Rule application

Symbolic well-founded model

$$\left\{ \begin{array}{l} p \leftarrow q \vee s \\ q \leftarrow p \\ r \leftarrow \neg p \end{array} \right\}$$

Symbolic interpretation: $p \mapsto s, q \mapsto s, r \mapsto \neg s$

Knowledge Compilation

$$\left\{ \begin{array}{l} p \leftarrow q \vee s \\ q \leftarrow p \\ r \leftarrow \neg p \end{array} \right\}$$

Propositional theory: $(p \Leftrightarrow s) \wedge (q \Leftrightarrow s) \wedge (r \Leftrightarrow \neg s)$

Knowledge Compilation

$$\left\{\begin{array}{l} p \leftarrow q \vee s \\ q \leftarrow p \\ r \leftarrow \neg p \end{array}\right\}$$

Non-recursive logic program:

$$\left\{\begin{array}{l} p \leftarrow s. \\ q \leftarrow s. \\ r \leftarrow \neg s. \end{array}\right\}$$

Content

Background: Approximation Fixpoint Theory

- Algebraical theory
- Defines different types of fixpoints of lattice operators
  - Supported fixpoints
  - (Partial) stable fixpoints
  - Kripke-Kleene fixpoint
  - Well-founded fixpoint
- Captures semantics of many logical formalisms
  - Logic programming
  - Autoepistemic logic (Moore, 1985)
  - Default logic (Reiter, 1980)
  - Dung's argumentation frameworks (Dung, 1995)
  - Abstract dialectical frameworks (Brewka and Woltran, 2013)

(Denecker, Marek, Truszczyński, 2000)

Background: Approximation Fixpoint Theory

- Given:
  - Complete lattice $\langle L, \leq \rangle$
  - Bilattice $\langle L^2, \leq_p \rangle$
  - Lattice operator $O : L \to L$
  - Approximator $A : L^2 \to L^2$
- Define:
  - Supported fixpoint: fixpoint of O
  - A-Kripke-Kleene fixpoint: $\text{lfp}_{\leq_p} A$
  - Partial A-stable fixpoint: pair (x, y) such that $x = \text{lfp}(A(\cdot, y)_1)$ and $y = \text{lfp}(A(x, \cdot)_2)$
  - A-well-founded fixpoint: least precise partial A-stable fixpoint
  - A-stable fixpoint of O: fixpoint x of O such that (x, x) is a partial A-stable fixpoint

Background: Approximation Fixpoint Theory

- Given: (Logic programming)
  - Complete lattice $\langle L, \leq \rangle$ (Lattice of interpretations: $\langle 2^{\Sigma_d}, \subseteq \rangle$)
  - Bilattice $\langle L^2, \leq_p \rangle$ (Partial interpretations)
  - Lattice operator $O : L \to L$ ($T_{\mathcal{P}}$)
  - Approximator $A : L^2 \to L^2$ ($\Psi_{\mathcal{P}}$)
- Define:
  - Supported fixpoint: fixpoint of O (Supported model)(Clark, 1978)
  - A-Kripke-Kleene fixpoint: $\text{lfp}_{\leq_p} A$ (Kripke-Kleene semantics) (Fitting, 1985)
  - Partial A-stable fixpoint: pair (x, y) such that $x = \text{lfp}(A(\cdot, y)_1)$ and $y = \text{lfp}(A(x, \cdot)_2)$ (Partial stable model)
  - A-well-founded fixpoint: least precise partial A-stable fixpoint (Well-founded model) (Van Gelder, Ross and Schilpf, 1988)
  - A-stable fixpoint of O: fixpoint x of O such that (x, x) is a partial A-stable fixpoint (Stable model) (Gelfond and Lifschitz, 1988)

Background: Approximation Fixpoint Theory

- Well-founded induction (Denecker and Vennekens, 2007):
  - Algebraical generalisation of well-founded model computation
  - Constructive characterisation of the well-founded fixpoint
  - Sequence of bilattice elements (partial interpretations)
  - Transitions similar to “rule application” and “unfounded set computation”

Extending AFT

1. Studied the link between well-founded inductions of different operators
2. (For logic programming): defined symbolic versions of $T_{\mathcal{P}}$ and $\Psi_{\mathcal{P}}$ (called $\mathcal{T}_{\mathcal{P}}$ and $\mathit{\Psi}_{\mathcal{P}}$ respectively)
3. Applying (1): well-founded induction of $\mathit{\Psi}_{\mathcal{P}}$: fixpoint procedure to compute the parametrised well-founded model

Extending AFT

1. Studied the link between well-founded inductions of different operators
2. (For logic programming): defined symbolic versions of $T_{\mathcal{P}}$ and $\Psi_{\mathcal{P}}$ (called $\mathcal{T}_{\mathcal{P}}$ and $\mathit{\Psi}_{\mathcal{P}}$ respectively)
3. Applying (1): well-founded induction of $\mathit{\Psi}_{\mathcal{P}}$: fixpoint procedure to compute the parametrised well-founded model
4. Any-time algorithm for approximate inference

Theorem

Let $\mathcal{P}$ be a parametrised logic program with parametrised well-founded model $(\mathcal{A}, \mathcal{A})$. Let $(\mathcal{A}_i, \mathcal{A}'_i)_{i \leq \beta}$ be a well-founded induction of $\mathit{\Psi}_{\mathcal{P}}$. Then

$$WMC(\mathcal{A}_i, \varphi, W) \leq WMC(\mathcal{A}, \varphi, W) \leq WMC(\mathcal{A}'_i, \varphi, W).$$

Content

Advantages

Advantages of this approach for logic programming

- No auxiliary variables needed
- Preserves equivalence
- No loop-breaking preprocessing
- Any-time algorithm
- Works with any type of circuits ("propositional formulas modulo equivalence") that support bottom-up compilation

Advantages

Advantages of this approach for logic programming

- No auxiliary variables needed
- Preserves equivalence
- No loop-breaking preprocessing
- Any-time algorithm
- Works with any type of circuits ("propositional formulas modulo equivalence") that support bottom-up compilation

Advantages of using AFT

- Paves the way for knowledge compilation for autoepistemic logic, default logic, abstract argumentation, ...
- Knowledge compilation for Kripke-Kleene semantics
- Simple (but abstract) proofs of correctness

Conclusion

Main contributions:

- Lifted knowledge compilation principles to *general* logic programs
- Lifted knowledge compilation principles to *AFT*

Work in progress:

- Implementation and experiments
- Knowledge compilation for stable semantics