Productive Corecursion in Logic Programming

Yue Li

School of Mathematical and Computer Sciences
Heriot-Watt University
Joint research with Ekaterina Komendantskaya

30 Aug 2017

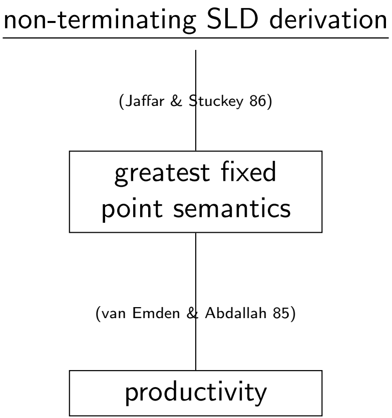
Outline

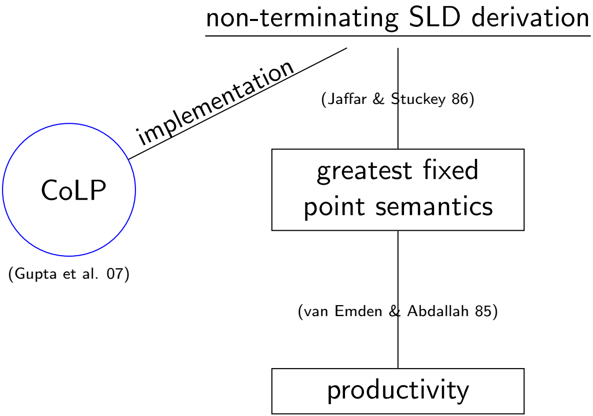
1. Motivation
   - Overview of motivation
   - Background knowledge for understanding motivation
   - Problem description
2. Productive Corecursion
   - Loop detection rule review
   - Productivity guarantee
3. Conclusion
4. Future Work & Implementation

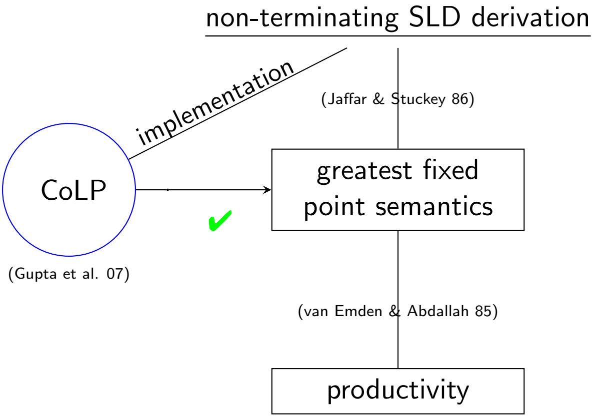
non-terminating SLD derivation

(Jaffar & Stuckey 86)

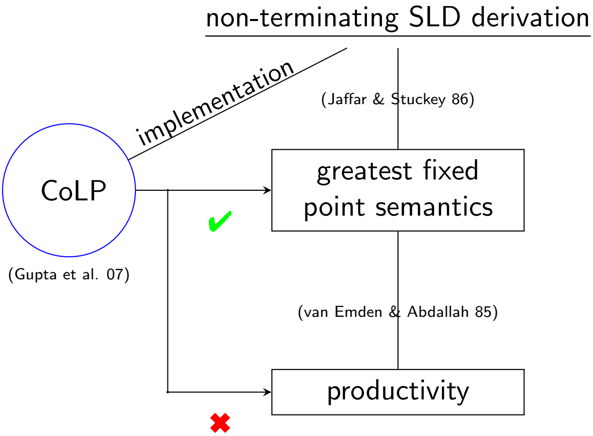
greatest fixed point semantics


non-terminating SLD derivation
(Jaffar & Stuckey 86)
greatest fixed point semantics
(van Emden & Abdallah 85)
productivity


non-terminating SLD derivation
implementation
(Jaffar & Stuckey 86)
CoLP
(Gupta et al. 07)
greatest fixed point semantics
(van Emden & Abdallah 85)
productivity

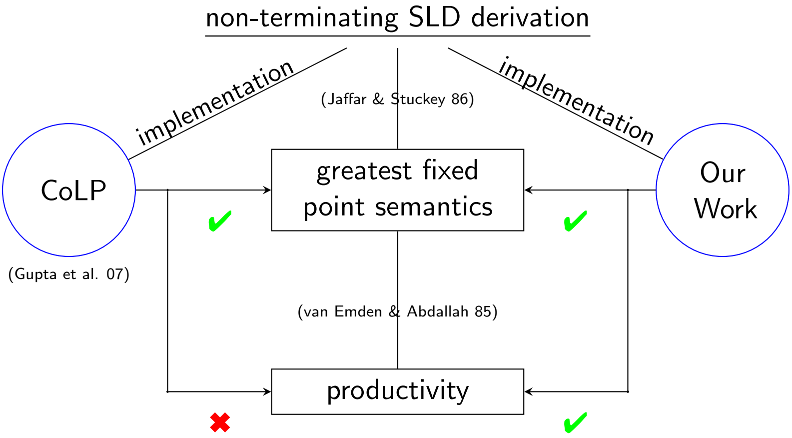

non-terminating SLD derivation
implementation
(Jaffar & Stuckey 86)
CoLP
greatest fixed point semantics
(Gupta et al. 07)
(van Emden & Abdallah 85)
productivity

✔: sound



✔: sound

✖: not sound



✔: sound
✖: not sound

Definition (Syntax of definite clause logic) (Lloyd 87)

Definition (Syntax of definite clause logic) (Lloyd 87)

Term:=Constant | Variable | Functor (<List of Terms>)

Definition (Syntax of definite clause logic) (Lloyd 87)

Term:=Constant | Variable | Functor (<List of Terms>)

Definite clause:=Term $\leftarrow$ Set of Terms

Definition (Syntax of definite clause logic) (Lloyd 87)

Term:=Constant | Variable | Functor (<List of Terms>)

Definite clause:=Term $\leftarrow$ Set of Terms

Goal clause:=List of Terms

Definition (Syntax of definite clause logic) (Lloyd 87)

Term:=Constant | Variable | Functor (<List of Terms>)
Definite clause:=Term $\leftarrow$ Set of Terms
Goal clause:=List of Terms
Program:=Set of Definite clauses

Definition (Fixed point semantics) (Lloyd 87)

Given a logic program,

Definition (Fixed point semantics) (Lloyd 87)

Given a logic program,

the least fixed point is the smallest set closed forward under the program.

Definition (Fixed point semantics) (Lloyd 87)

Given a logic program,

the least fixed point is the smallest set closed forward under the program.

the greatest fixed point is the largest set closed backward under the program.

Definition (Fixed point semantics) (Lloyd 87)

Given a logic program,

the least fixed point is the smallest set closed forward under the program.

the greatest fixed point is the largest set closed backward under the program.

Example

Definition (Fixed point semantics) (Lloyd 87)

Given a logic program,

the least fixed point is the smallest set closed forward under the program.

the greatest fixed point is the largest set closed backward under the program.

Example

nat(0)

Definition (Fixed point semantics) (Lloyd 87)

Given a logic program,

the least fixed point is the smallest set closed forward under the program.

the greatest fixed point is the largest set closed backward under the program.

Example

nat(0)
nat(s(X)) $\leftarrow nat(X)$

Definition (Fixed point semantics) (Lloyd 87)

Given a logic program,

the least fixed point is the smallest set closed forward under the program.

the greatest fixed point is the largest set closed backward under the program.

Example

nat(0)
nat(s(X)) $\leftarrow nat(X)$
The least fixed point is {nat(0), nat(s(0)), nat(s(s(0))), ...}.

Definition (Fixed point semantics) (Lloyd 87)

Given a logic program,

the least fixed point is the smallest set closed forward under the program.

the greatest fixed point is the largest set closed backward under the program.

Example

nat(0)
nat(s(X)) $\leftarrow$ $nat(X)$
The least fixed point is {nat(0), nat(s(0)), nat(s(s(0))), ...}.
The greatest fixed point is {nat(0), nat(s(0)), nat(s(s(0))), ...} $\cup$ {nat(s(s(...)))}.

Definition (Fixed point semantics) (Lloyd 87)

Given a logic program,

the least fixed point is the smallest set closed forward under the program.

the greatest fixed point is the largest set closed backward under the program.

Example

nat(0)
nat(s(X)) $\leftarrow nat(X)$
The least fixed point is {nat(0), nat(s(0)), nat(s(s(0))), ...}.
The greatest fixed point is {nat(0), nat(s(0)), nat(s(s(0))), ...} $\cup$ {nat(s(s(...)))}.

Formulae computed by non-terminating derivations are in greatest fixed points. (Jaffar & Stuckey 86; van Emden & Abdallah 85)

Definition (Productivity)
(LP: van Emden & Abdallah 86; Komendantskaya et al. 16; FP: Sijtsma 89; Endrullis et al. 08)

Definition (Productivity)
**(LP: van Emden & Abdallah 86; Komendantskaya et al. 16;
FP: Sijtsma 89; Endrullis et al. 08)**

A productive non-terminating derivation does useful computations while looping rather than just looping.

Definition (Productivity)
(LP: van Emden & Abdallah 86; Komendantskaya et al. 16;
FP: Sijtsma 89; Endrullis et al. 08)

A productive non-terminating derivation does useful computations while looping rather than just looping.

Example

nat(X) $\leftarrow$ nat(X)

Definition (Productivity)
(LP: van Emden & Abdallah 86; Komendantskaya et al. 16;
FP: Sijtsma 89; Endrullis et al. 08)

A productive non-terminating derivation does useful computations while looping rather than just looping.

Example

$\mathrm{nat}(X) \leftarrow \mathrm{nat}(X)$
has non-productive derivation:

Definition (Productivity)
(LP: van Emden & Abdallah 86; Komendantskaya et al. 16;
FP: Sijtsma 89; Endrullis et al. 08)

A productive non-terminating derivation does useful computations while looping rather than just looping.

Example

$\text{nat}(X) \leftarrow \text{nat}(X)$
has non-productive derivation:

$$\begin{array}{c} \text{nat}(X) \\ \downarrow \\ \text{nat}(X) \\ \downarrow \\ \text{nat}(X) \\ \vdots \end{array}$$

Definition (Productivity)
(LP: van Emden & Abdallah 86; Komendantskaya et al. 16;
FP: Sijtsma 89; Endrullis et al. 08)

A productive non-terminating derivation does useful computations while looping rather than just looping.

Example

nat(X) ← nat(X)
has non-productive derivation:

$$\begin{array}{c} \text{nat(X)} \\ \downarrow \\ \text{nat(X)} \\ \downarrow \\ \text{nat(X)} \\ \vdots \end{array}$$

Example

nat(s(X)) ← nat(X)

Definition (Productivity)
(LP: van Emden & Abdallah 86; Komendantskaya et al. 16;
FP: Sijtsma 89; Endrullis et al. 08)

A productive non-terminating derivation does useful computations while looping rather than just looping.

Example

nat(X) $\leftarrow$ nat(X)
has non-productive derivation:

$$\begin{array}{c} \text{nat(X)} \\ \downarrow \\ \text{nat(X)} \\ \downarrow \\ \text{nat(X)} \\ \vdots \end{array}$$

Example

nat(s(X)) $\leftarrow$ nat(X)
computes the first limit ordinal
nat(s(s(. . .))):

Definition (Productivity)
(LP: van Emden & Abdallah 86; Komendantskaya et al. 16; FP: Sijtsma 89; Endrullis et al. 08)

A productive non-terminating derivation does useful computations while looping rather than just looping.

Example

nat(X) ← nat(X)
has non-productive derivation:

$$\begin{array}{c} \text{nat}(X) \\ \downarrow \\ \text{nat}(X) \\ \downarrow \\ \text{nat}(X) \\ \vdots \end{array}$$

Example

nat(s(X)) ← nat(X)
computes the first limit ordinal
nat(s(s(. . .))):

$$\begin{array}{c} \text{nat}(X) \\ \downarrow\ X \mapsto s(X_2) \\ \text{nat}(X_2) \\ \downarrow\ X_2 \mapsto s(X_3) \\ \text{nat}(X_3) \\ \vdots \end{array}$$

Now consider finite implementation of non-terminating SLD derivations.

Since regular formulae have cyclic derivations, finding a cycle (loop) is suffice for knowing the whole derivation.

Definition (Gupta et al. 07)

CoLP = SLD resolution + loop detection rule.

Definition (Loop detection rule) (Gupta et al. 07)

A goal succeeds if it unifies with its ancestor goal.

Theorem (Coinductive soundness of coLP) (Gupta et al. 07)

Now consider finite implementation of non-terminating SLD derivations.
Since regular formulae have cyclic derivations, finding a cycle (loop) is suffice for knowing the whole derivation.

Definition (Gupta et al. 07)

CoLP = SLD resolution + loop detection rule.

Definition (Loop detection rule) (Gupta et al. 07)

A goal succeeds if it unifies with its ancestor goal.

Theorem (Coinductive soundness of coLP) (Gupta et al. 07)

Now consider finite implementation of non-terminating SLD derivations.
Since regular formulae have cyclic derivations, finding a cycle (loop) is suffice for knowing the whole derivation.

Definition (Gupta et al. 07)

CoLP = SLD resolution + loop detection rule.

Definition (Loop detection rule) (Gupta et al. 07)

A goal succeeds if it unifies with its ancestor goal.

Theorem (Coinductive soundness of coLP) (Gupta et al. 07)

Now consider finite implementation of non-terminating SLD derivations.

Since regular formulae have cyclic derivations, finding a cycle (loop) is suffice for knowing the whole derivation.

Definition (Gupta et al. 07)

CoLP = SLD resolution + loop detection rule.

Definition (Loop detection rule) (Gupta et al. 07)

A goal succeeds if it unifies with its ancestor goal.

Theorem (Coinductive soundness of coLP) (Gupta et al. 07)

Now consider finite implementation of non-terminating SLD derivations.

Since regular formulae have cyclic derivations, finding a cycle (loop) is suffice for knowing the whole derivation.

Definition (Gupta et al. 07)

CoLP = SLD resolution + loop detection rule.

Definition (Loop detection rule) (Gupta et al. 07)

A goal succeeds if it unifies with its ancestor goal.

Theorem (Coinductive soundness of coLP) (Gupta et al. 07)

Now consider finite implementation of non-terminating SLD derivations.

Since regular formulae have cyclic derivations, finding a cycle (loop) is suffice for knowing the whole derivation.

Definition (Gupta et al. 07)

CoLP = SLD resolution + loop detection rule.

Definition (Loop detection rule) (Gupta et al. 07)

A goal succeeds if it unifies with its ancestor goal.

Theorem (Coinductive soundness of coLP) (Gupta et al. 07)

Successful coLP derivations only compute formulae in greatest fixed points.

Example (CoLP at work)

Example (CoLP at work)

$nat(s(X)) \leftarrow nat(X)$ defines the first limit ordinal $s(s(\ldots))$.

Example (CoLP at work)

$nat(s(X)) \leftarrow nat(X)$ defines the first limit ordinal $s(s(\ldots))$. We compare coLP and SLD derivation for goal $nat(X)$.

Example (CoLP at work)

nat(s(X)) $\leftarrow$ nat(X) defines the first limit ordinal s(s(...)). We compare coLP and SLD derivation for goal nat(X).

SLD derivation
(non-terminating)

Example (CoLP at work)

nat(s(X)) ← nat(X) defines the first limit ordinal s(s(...)). We compare coLP and SLD derivation for goal nat(X).

SLD derivation
(non-terminating)

G_0 nat(X)

$\downarrow$ $X \mapsto s(X_2)$

G_1 nat(X_2)

$\downarrow$ $X_2 \mapsto s(X_3)$

G_2 nat(X_3)

$\vdots$

Example (CoLP at work)

nat(s(X)) ← nat(X) defines the first limit ordinal s(s(...)). We compare coLP and SLD derivation for goal nat(X).

SLD derivation (non-terminating)

G_0 nat(X)

$\downarrow$ X $\mapsto s(X_2)$

G_1 nat(X_2)

$\downarrow$ $X_2 \mapsto s(X_3)$

G_2 nat(X_3)

$\vdots$

CoLP derivation (terminating)

Example (CoLP at work)

nat(s(X)) $\leftarrow$ nat(X) defines the first limit ordinal s(s(...)). We compare coLP and SLD derivation for goal nat(X).

SLD derivation (non-terminating)

G_0 nat(X)
$\downarrow$ X $\mapsto s(X_2)$
G_1 nat(X_2)
$\downarrow$ $X_2 \mapsto s(X_3)$
G_2 nat(X_3)
$\vdots$

CoLP derivation (terminating)

G_0 nat(X)
$\downarrow$ X $\mapsto s(X_2)$
G_1 nat(X_2)
$\downarrow$ $X_2 \mapsto X$ (G_1 unifies G_0)
G_2 $\square$

Example (CoLP at work)

nat(s(X)) $\leftarrow$ nat(X) defines the first limit ordinal s(s(...)). We compare coLP and SLD derivation for goal nat(X).

SLD derivation (non-terminating)

G_0 nat(X)
$\downarrow$ X $\mapsto s(X_2)$
G_1 nat(X_2)
$\downarrow$ $X_2 \mapsto s(X_3)$
G_2 nat(X_3)
$\vdots$

CoLP derivation (terminating)

G_0 nat(X)
$\downarrow$ X $\mapsto s(X_2)$
G_1 nat(X_2)
$\downarrow$ $X_2 \mapsto X$ (G_1 unifies G_0)
G_2 $\square$

SLD derivation computes s(s(...)) by accumulating

Example (CoLP at work)

nat(s(X)) $\leftarrow$ nat(X) defines the first limit ordinal s(s(...)). We compare coLP and SLD derivation for goal nat(X).

SLD derivation (non-terminating)

G_0 nat(X)

$\downarrow$ X $\mapsto s(X_2)$

G_1 nat(X_2)

$\downarrow$ $X_2 \mapsto s(X_3)$

G_2 nat(X_3)

$\vdots$

CoLP derivation (terminating)

G_0 nat(X)

$\downarrow$ X $\mapsto s(X_2)$

G_1 nat(X_2)

$\downarrow$ $X_2 \mapsto X$ (G_1 unifies G_0)

G_2 $\square$

SLD derivation computes s(s(...)) by accumulating $X \mapsto s(X_2)$,

Example (CoLP at work)

nat(s(X)) ← nat(X) defines the first limit ordinal s(s(...)). We compare coLP and SLD derivation for goal nat(X).

SLD derivation (non-terminating)	CoLP derivation (terminating)
G_0 nat(X)	G_0 nat(X)
$\downarrow$ $X \mapsto s(X_2)$	$\downarrow$ $X \mapsto s(X_2)$
G_1 nat(X_2)	G_1 nat(X_2)
$\downarrow$ $X_2 \mapsto s(X_3)$	$\downarrow$ $X_2 \mapsto X$ (G_1 unifies G_0)
G_2 nat(X_3) $\vdots$	G_2 $\square$

SLD derivation computes s(s(...)) by accumulating $X \mapsto s(X_2)$, $X_2 \mapsto s(X_3)$, ...

Example (CoLP at work)

nat(s(X)) $\leftarrow$ nat(X) defines the first limit ordinal s(s(...)). We compare coLP and SLD derivation for goal nat(X).

SLD derivation (non-terminating)	CoLP derivation (terminating)
G_0 nat(X)	G_0 nat(X)
$\downarrow$ X$\mapsto s(X_2)$	$\downarrow$ X$\mapsto s(X_2)$
G_1 nat(X_2)	G_1 nat(X_2)
$\downarrow$ $X_2 \mapsto s(X_3)$	$\downarrow$ $X_2 \mapsto X$ (G_1 unifies G_0)
G_2 nat(X_3) $\vdots$	G_2 $\Box$

SLD derivation computes s(s(...)) by accumulating $X \mapsto s(X_2)$, $X_2 \mapsto s(X_3)$, ... CoLP derivation computes s(s(...)) by circular binding

Example (CoLP at work)

nat(s(X)) $\leftarrow$ nat(X) defines the first limit ordinal s(s(...)). We compare coLP and SLD derivation for goal nat(X).

SLD derivation (non-terminating)

$$G_0 \quad \mathrm{nat}(X)$$
$$\downarrow \; X \mapsto s(X_2)$$
$$G_1 \quad \mathrm{nat}(X_2)$$
$$\downarrow \; X_2 \mapsto s(X_3)$$
$$G_2 \quad \mathrm{nat}(X_3)$$
$$\vdots$$

CoLP derivation (terminating)

$$G_0 \quad \mathrm{nat}(X)$$
$$\downarrow \; X \mapsto s(X_2)$$
$$G_1 \quad \mathrm{nat}(X_2)$$
$$\downarrow \; X_2 \mapsto X \; (G_1 \text{ unifies } G_0)$$
$$G_2 \quad \square$$

SLD derivation computes s(s(...)) by accumulating $X \mapsto s(X_2)$, $X_2 \mapsto s(X_3)$, ... CoLP derivation computes s(s(...)) by circular binding X$\mapsto s(X)$

However, coLP does not take good care of productivity . . .

Assume some successful coLP derivation that computes an infinite formula.

Problem 1: *It is not guaranteed that there exists a corresponding non-terminating SLD derivation.*

e.g. For program p(f(X),X) ← p(X,X) and goal p(f(X),X), coLP computes p(f(f(. . .)),f(f(. . .))) but here is no non-terminating SLD derivation.

CoLP derivation

G_0 : p(f(X),X)

↓

G_1 : p(X,X)

↓ $X \mapsto f(f(\ldots))$ by unifying G_1 with G_0

G_2 : □

However, coLP does not take good care of productivity . . .

Assume some successful coLP derivation that computes an infinite formula.

Problem 1: *It is not guaranteed that there exists a corresponding non-terminating SLD derivation.*

e.g. For program p(f(X),X) ← p(X,X) and goal p(f(X),X), coLP computes p(f(f(. . .)),f(f(. . .))) but here is no non-terminating SLD derivation.

CoLP derivation

G_0 : p(f(X),X)

↓

G_1 : p(X,X)

↓ $X \mapsto f(f(\ldots))$ by unifying G_1 with G_0

G_2 : $\square$

However, coLP does not take good care of productivity . . .

Assume some successful coLP derivation that computes an infinite formula.

Problem 1: *It is not guaranteed that there exists a corresponding non-terminating SLD derivation.*

e.g. For program p(f(X),X) ← p(X,X) and goal p(f(X),X), coLP computes p(f(f(. . .)),f(f(. . .))) but here is no non-terminating SLD derivation.

CoLP derivation

G_0 : p(f(X),X)

↓

G_1 : p(X,X)

↓ $X \mapsto f(f(\ldots))$ by unifying G_1 with G_0

G_2 : □

However, coLP does not take good care of productivity . . .

Assume some successful coLP derivation that computes an infinite formula.

Problem 1: *It is not guaranteed that there exists a corresponding non-terminating SLD derivation.*

e.g. For program p(f(X),X) $\leftarrow$ p(X,X) and goal p(f(X),X), coLP computes p(f(f(. . .)),f(f(. . .))) but here is no non-terminating SLD derivation.

CoLP derivation

G_0 : p(f(X),X)

$\downarrow$

G_1 : p(X,X)

$\downarrow$ $X \mapsto f(f(\ldots))$ by unifying G_1 with G_0

G_2 : $\square$

However, coLP does not take good care of productivity . . .

Assume some successful coLP derivation that computes an infinite formula.

Problem 1: *It is not guaranteed that there exists a corresponding non-terminating SLD derivation.*

e.g. For program p(f(X),X) ← p(X,X) and goal p(f(X),X), coLP computes p(f(f(. . .)),f(f(. . .))) but here is no non-terminating SLD derivation.

CoLP derivation

$$G_0 : \ \text{p(f(X),X)}$$
$$\downarrow$$
$$G_1 : \ \text{p(X,X)}$$
$$\downarrow \ X \mapsto f(f(\ldots)) \text{ by unifying } G_1 \text{ with } G_0$$
$$G_2 : \ \square$$

However, coLP does not take good care of productivity ...

Assume some successful coLP derivation that computes an infinite formula.

Problem 1: *It is not guaranteed that there exists a corresponding non-terminating SLD derivation.*

e.g. For program p(f(X),X) ← p(X,X) and goal p(f(X),X), coLP computes p(f(f(...)),f(f(...))) but here is no non-terminating SLD derivation.

SLD derivation

G_0 : p(f(X),X)

↓

G_1 : p(X,X)

† G_1 does not unifiy head of clause

However, coLP does not take good care of productivity . . .

Assume some successful coLP derivation that computes an infinite formula.

Problem 1: *It is not guaranteed that there exists a corresponding non-terminating SLD derivation.*

e.g. For program p(f(X),X) ← p(X,X) and goal p(f(X),X), coLP computes p(f(f(. . .)),f(f(. . .))) but here is no non-terminating SLD derivation.

SLD derivation

G_0 : p(f(X),X)

↓

G_1 : p(X,X)

† G_1 does not unifiy head of clause

However, coLP does not take good care of productivity ...

Assume some successful coLP derivation that computes an infinite formula.

Problem 2: *There exists a corresponding non-terminating SLD derivation but it computes a different formula.*

e.g. For program q(f(X),Y) ← q(X,h(Y)) and goal q(f(X),Y), coLP computes q(f(f(...)),h(h(...))) but the corresponding non-terminating SLD derivation computes q(f(f(...)), Y).

CoLP derivation

G_0 : q(f(X),Y)

↓

G_1 : q(X,h(Y))

↓ $X \mapsto f(f(\ldots)), Y \mapsto h(h(\ldots))$ by unifying G_1 with G_0

G_2 : □

However, coLP does not take good care of productivity . . .

Assume some successful coLP derivation that computes an infinite formula.

Problem 2: *There exists a corresponding non-terminating SLD derivation but it computes a different formula.*

e.g. For program q(f(X),Y) ← q(X,h(Y)) and goal q(f(X),Y), coLP computes q(f(f(. . .)),h(h(. . .))) but the corresponding non-terminating SLD derivation computes q(f(f(. . .)), Y).

CoLP derivation

G_0 : q(f(X),Y)

↓

G_1 : q(X,h(Y))

↓ $X \mapsto f(f(\ldots)), Y \mapsto h(h(\ldots))$ by unifying G_1 with G_0

G_2 : $\square$

However, coLP does not take good care of productivity . . .

Assume some successful coLP derivation that computes an infinite formula.

Problem 2: *There exists a corresponding non-terminating SLD derivation but it computes a different formula.*

e.g. For program q(f(X),Y) ← q(X,h(Y)) and goal q(f(X),Y), coLP computes q(f(f(. . .)),h(h(. . .))) but the corresponding non-terminating SLD derivation computes q(f(f(. . .)), Y).

CoLP derivation

G_0 : q(f(X),Y)

↓

G_1 : q(X,h(Y))

↓ $X \mapsto f(f(\ldots)), Y \mapsto h(h(\ldots))$ by unifying G_1 with G_0

G_2 : $\square$

However, coLP does not take good care of productivity . . .

Assume some successful coLP derivation that computes an infinite formula.

Problem 2: *There exists a corresponding non-terminating SLD derivation but it computes a different formula.*

e.g. For program q(f(X),Y) ← q(X,h(Y)) and goal q(f(X),Y), coLP computes q(f(f(. . .)),h(h(. . .))) but the corresponding non-terminating SLD derivation computes q(f(f(. . .)), Y).

CoLP derivation

G_0 : q(f(X),Y)

↓

G_1 : q(X,h(Y))

↓ $X \mapsto f(f(\ldots)), Y \mapsto h(h(\ldots))$ by unifying G_1 with G_0

G_2 : $\square$

However, coLP does not take good care of productivity . . .
Assume some successful coLP derivation that computes an infinite formula.
Problem 2: *There exists a corresponding non-terminating SLD derivation but it computes a different formula.*
e.g. For program q(f(X),Y) ← q(X,h(Y)) and goal q(f(X),Y), coLP computes q(f(f(. . .)),h(h(. . .))) but the corresponding non-terminating SLD derivation computes q(f(f(. . .)), Y).

SLD derivation (non-terminating)

$$
\begin{array}{rl}
G_0 : & \mathrm{q(f(X),Y)} \\
 & \downarrow \\
G_1 : & \mathrm{q(X,h(Y))} \\
 & \downarrow \; X \mapsto f(X_2) \\
G_2 : & \mathrm{q}(X_2,\mathrm{h(h(Y))}) \\
 & \downarrow \; X_2 \mapsto f(X_3) \\
G_3 : & \mathrm{q}(X_3,\mathrm{h(h(h(Y)))}) \\
 & \vdots
\end{array}
$$

However, coLP does not take good care of productivity . . .

Assume some successful coLP derivation that computes an infinite formula.

Problem 2: *There exists a corresponding non-terminating SLD derivation but it computes a different formula.*

e.g. For program q(f(X),Y) ← q(X,h(Y)) and goal q(f(X),Y), coLP computes q(f(f(. . .)),h(h(. . .))) but the corresponding non-terminating SLD derivation computes q(f(f(. . .)), Y).

SLD derivation (non-terminating)

G_0 : q(f(X),Y)

↓

G_1 : q(X,h(Y))

↓ $X \mapsto f(X_2)$

G_2 : q(X_2,h(h(Y)))

↓ $X_2 \mapsto f(X_3)$

G_3 : q(X_3,h(h(h(Y))))

⋮

Our question

Our question

Can we have an implementation of infinite SLD derivation,

Our question

Can we have an implementation of infinite SLD derivation, that is not only sound regarding greatest fixed points,

Our question

Can we have an implementation of infinite SLD derivation,
that is not only sound regarding greatest fixed points,
but also sound regarding productivity?

Our question

Can we have an implementation of infinite SLD derivation, that is not only sound regarding greatest fixed points, but also sound regarding productivity?

Our answer is affirmative.

If two formulae unify *without* occurs check and produce an infinite regular formula, If two formulae unify *without* occurs check and produce an infinite regular formula, we should standardize them apart, If two formulae unify *without* occurs check and produce an infinite regular formula, we should standardize them apart, then try to unify the standardized version *with* occurs check.

If two formulae unify *without* occurs check and produce an infinite regular formula, we should standardize them apart, then try to unify the standardized version *with* occurs check. The following are possible results.

If two formulae unify *without* occurs check and produce an infinite regular formula, we should standardize them apart, then try to unify the standardized version *with* occurs check. The following are possible results.

Case 1 They do not unify.

If two formulae unify *without* occurs check and produce an infinite regular formula, we should standardize them apart, then try to unify the standardized version *with* occurs check. The following are possible results.

Case 1 They do not unify. e.g. p(f(X),X) and p(X,X);

If two formulae unify *without* occurs check and produce an infinite regular formula, we should standardize them apart, then try to unify the standardized version *with* occurs check. The following are possible results.

Case 1 They do not unify. e.g. p(f(X),X) and p(X,X); p(f(X),X) and p(X_1, X_1).

If two formulae unify *without* occurs check and produce an infinite regular formula, we should standardize them apart, then try to unify the standardized version *with* occurs check. The following are possible results.

Case 1 They do not unify. e.g. p(f(X),X) and p(X,X); p(f(X),X) and p(X_1, X_1).

Case 2 One is a variant or instance of the other.

If two formulae unify *without* occurs check and produce an infinite regular formula, we should standardize them apart, then try to unify the standardized version *with* occurs check. The following are possible results.

Case 1 They do not unify. e.g. p(f(X),X) and p(X,X); p(f(X),X) and p(X_1, X_1).

Case 2 One is a variant or instance of the other. e.g. nat(X) and nat(s(X));

If two formulae unify *without* occurs check and produce an infinite regular formula, we should standardize them apart, then try to unify the standardized version *with* occurs check. The following are possible results.

Case 1 They do not unify. e.g. p(f(X),X) and p(X,X); p(f(X),X) and p(X_1, X_1).

Case 2 One is a variant or instance of the other. e.g. nat(X) and nat(s(X)); nat(X) and nat(s(X_1)).

If two formulae unify *without* occurs check and produce an infinite regular formula, we should standardize them apart, then try to unify the standardized version *with* occurs check. The following are possible results.

Case 1 They do not unify. e.g. p(f(X),X) and p(X,X); p(f(X),X) and p(X_1, X_1).

Case 2 One is a variant or instance of the other. e.g. nat(X) and nat(s(X)); nat(X) and nat(s(X_1)).

Case 3 Otherwise.

If two formulae unify *without* occurs check and produce an infinite regular formula, we should standardize them apart, then try to unify the standardized version *with* occurs check. The following are possible results.

Case 1 They do not unify. e.g. p(f(X),X) and p(X,X); p(f(X),X) and p(X_1, X_1).

Case 2 One is a variant or instance of the other. e.g. nat(X) and nat(s(X)); nat(X) and nat(s(X_1)).

Case 3 Otherwise. e.g. q(f(X),Y) and q(X,h(Y));

If two formulae unify *without* occurs check and produce an infinite regular formula, we should standardize them apart, then try to unify the standardized version *with* occurs check. The following are possible results.

Case 1 They do not unify. e.g. p(f(X),X) and p(X,X); p(f(X),X) and p(X_1, X_1).

Case 2 One is a variant or instance of the other. e.g. nat(X) and nat(s(X)); nat(X) and nat(s(X_1)).

Case 3 Otherwise. e.g. q(f(X),Y) and q(X,h(Y)); q(f(X),Y) and q(X_1,h(Y_1)).

If two formulae unify *without* occurs check and produce an infinite regular formula, we should standardize them apart, then try to unify the standardized version *with* occurs check. The following are possible results.

Case 2 One is a variant or instance of the other. e.g. nat(X) and nat(s(X)); nat(X) and nat(s(X_1)).

To guarantee soundness regarding productivity, we propose using the following loop detection rule

To guarantee soundness regarding productivity, we propose using the following loop detection rule

Definition (Our loop detection rule)

A goal succeeds if it's a variant of its ancestor goal.

To guarantee soundness regarding productivity, we propose using the following loop detection rule

Definition (Our loop detection rule)

A goal succeeds if it's a variant of its ancestor goal.

instead of

Definition (Loop detection rule) (Gupta et al. 07)

A goal succeeds if it unifies with its ancestor goal.

We also characterized a class of logic programs whose non-terminating SLD derivations, if any, are guaranteed to be productive.

Definition (Rewriting for LP) (Komendantskaya et al. 15)

Rewriting is a special case of SLD resolution, where the selected subgoal is an instance of the chosen program clause's head.

Definition (Rewriting for LP) (Komendantskaya et al. 15)

Rewriting is a special case of SLD resolution, where the selected subgoal is an instance of the chosen program clause's head.

Example (Rewriting)

- From goal nat(X) and clause nat(X) $\leftarrow$ nat(X), derive nat(X).

Definition (Rewriting for LP) (Komendantskaya et al. 15)

Rewriting is a special case of SLD resolution, where the selected subgoal is an instance of the chosen program clause's head.

Example (Rewriting)

- From goal nat(s(X)) and clause nat(Y)$\leftarrow$ nat(s(Y)), derive nat(s(s(Y))).

Definition (Rewriting for LP) (Komendantskaya et al. 15)

Rewriting is a special case of SLD resolution, where the selected subgoal is an instance of the chosen program clause's head.

Theorem (Productivity Guarantee)

A logic program that is

Definition (Rewriting for LP) (Komendantskaya et al. 15)

Rewriting is a special case of SLD resolution, where the selected subgoal is an instance of the chosen program clause's head.

Theorem (Productivity Guarantee)

A logic program that is

1. *terminating for rewriting, and*

Definition (Rewriting for LP) (Komendantskaya et al. 15)

Rewriting is a special case of SLD resolution, where the selected subgoal is an instance of the chosen program clause's head.

Theorem (Productivity Guarantee)

A logic program that is

1. *terminating for rewriting, and*
2. *free from existential variables,*

Definition (Rewriting for LP) (Komendantskaya et al. 15)

Rewriting is a special case of SLD resolution, where the selected subgoal is an instance of the chosen program clause's head.

Theorem (Productivity Guarantee)

A logic program that is

1. *terminating for rewriting, and*
2. *free from existential variables,*

is guaranteed to be productive for its non-terminating SLD derivations, if any.

Definition (Rewriting for LP) (Komendantskaya et al. 15)

Rewriting is a special case of SLD resolution, where the selected subgoal is an instance of the chosen program clause's head.

Theorem (Productivity Guarantee)

A logic program that is

1. *terminating for rewriting, and*
2. *free from existential variables,*

is guaranteed to be productive for its non-terminating SLD derivations, if any.

Why termination for rewriting plays a role?

Definition (Rewriting for LP) (Komendantskaya et al. 15)

Rewriting is a special case of SLD resolution, where the selected subgoal is an instance of the chosen program clause's head.

Theorem (Productivity Guarantee)

A logic program that is

1. *terminating for rewriting, and*
2. *free from existential variables,*

is guaranteed to be productive for its non-terminating SLD derivations, if any.

Why termination for rewriting plays a role?

$$Assume\ SLD\ derivation\ is\ non\text{-}terminating,$$

Definition (Rewriting for LP) (Komendantskaya et al. 15)

Rewriting is a special case of SLD resolution, where the selected subgoal is an instance of the chosen program clause's head.

Theorem (Productivity Guarantee)

A logic program that is

1. *terminating for rewriting, and*
2. *free from existential variables,*

is guaranteed to be productive for its non-terminating SLD derivations, if any.

Why termination for rewriting plays a role?

> *Assume SLD derivation is non-terminating,*
> *where all rewriting terminates,*

Definition (Rewriting for LP) (Komendantskaya et al. 15)

Rewriting is a special case of SLD resolution, where the selected subgoal is an instance of the chosen program clause's head.

Theorem (Productivity Guarantee)

A logic program that is

1. *terminating for rewriting, and*
2. *free from existential variables,*

is guaranteed to be productive for its non-terminating SLD derivations, if any.

Why termination for rewriting plays a role?

> *Assume SLD derivation is non-terminating,*
> *where all rewriting terminates,*
> *then it is guaranteed that*

Definition (Rewriting for LP) (Komendantskaya et al. 15)

Rewriting is a special case of SLD resolution, where the selected subgoal is an instance of the chosen program clause's head.

Theorem (Productivity Guarantee)

A logic program that is

1. *terminating for rewriting, and*
2. *free from existential variables,*

is guaranteed to be productive for its non-terminating SLD derivations, if any.

Why termination for rewriting plays a role?

> *Assume SLD derivation is non-terminating,*
> *where all rewriting terminates,*
> *then it is guaranteed that*
> *there are infinitly many productive SLD resolution steps.*

Definition (Rewriting for LP) (Komendantskaya et al. 15)

Rewriting is a special case of SLD resolution, where the selected subgoal is an instance of the chosen program clause's head.

Theorem (Productivity Guarantee)

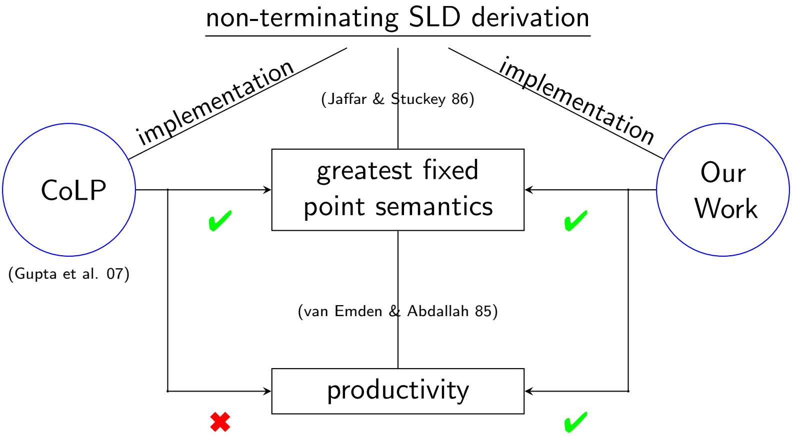
A logic program that is

1. *terminating for rewriting, and*
2. *free from existential variables,*

is guaranteed to be productive for its non-terminating SLD derivations, if any.

Theorem (our main result: Productivity Semi-decision)

Productivity is semi-decidable for programs characterized above, by SLD resolution combined with our loop detection rule.



✔: sound

✖: not sound

- We had to change the loop detection rule.

- We had to change the loop detection rule.
- Put conditions on clauses.

- We had to change the loop detection rule.
- Put conditions on clauses.
- These kinds of clauses characterize a rich class of productive corecursion in LP.

- We had to change the loop detection rule.
- Put conditions on clauses.
- These kinds of clauses characterize a rich class of productive corecursion in LP.

Example (number streams) (Gupta et al. 07)

Streams of natural numbers, e.g. 3 1 4 1 5 9 2 . . . ,

- We had to change the loop detection rule.
- Put conditions on clauses.
- These kinds of clauses characterize a rich class of productive corecursion in LP.

Example (number streams) (Gupta et al. 07)

Streams of natural numbers, e.g. 3 1 4 1 5 9 2 . . . , are defined by the corecursive clause nats([X|S]) $\leftarrow$ nat(X),nats(S).

- We had to change the loop detection rule.
- Put conditions on clauses.
- These kinds of clauses characterize a rich class of productive corecursion in LP.

Example (number streams) (Gupta et al. 07)

Streams of natural numbers, e.g. 3 1 4 1 5 9 2 . . . , are defined by the corecursive clause nats([X|S]) $\leftarrow$ nat(X),nats(S).

Example (increasing stream) (Simon et al. 06)

Streams of consecutive numbers, e.g. 1 2 3 . . . or 99 100 101 . . . ,

- We had to change the loop detection rule.
- Put conditions on clauses.
- These kinds of clauses characterize a rich class of productive corecursion in LP.

Example (number streams) (Gupta et al. 07)

Streams of natural numbers, e.g. 3 1 4 1 5 9 2 . . . , are defined by the corecursive clause nats([X|S]) $\leftarrow$ nat(X),nats(S).

Example (increasing stream) (Simon et al. 06)

Streams of consecutive numbers, e.g. 1 2 3 . . . or 99 100 101 . . . , are defined by the corecursive clause from(X,[X|T]) $\leftarrow$ from(s(X),T).

- Programs that contain existential variables?

- Programs that contain existential variables?

Example (Fibonacci stream) (Komendantskaya et al. 15)

Streams of Fibonacci numbers, e.g. 1 1 2 3 5 8 . . . or 10 4 14 18 32 . . . ,

- Programs that contain existential variables?

Example (Fibonacci stream) (Komendantskaya et al. 15)

Streams of Fibonacci numbers, e.g. 1 1 2 3 5 8 ... or 10 4 14 18 32 ..., are defined by a corecursive clause that has an existential variable.
fibs(X,Y,[X|S]) $\leftarrow$ add(X,Y,Z), fibs(Y,Z,S).

- Programs that contain existential variables?
- Practical application in

- Programs that contain existential variables?
- Practical application in
  - type inference in programming languages, and

- Programs that contain existential variables?
- Practical application in
  - type inference in programming languages, and
  - internet programming

Implementation is available at
GitHub / coalp / Productive-Corecursion

Thanks!