Structural Resolution with Co-inductive Loop Detection

Yue Li
School of Mathematical & Computer Sciences
Heriot-Watt University
Edinburgh, United Kingdom
yl55@hw.ac.uk

A way to combine co-SLD style loop detection with structural resolution was found and is introduced in this work, to extend structural resolution with co-induction. In particular, we present the operational semantics, called co-inductive structural resolution, of this novel combination and prove its soundness with respect to the greatest complete Herbrand model.

1 Introduction

Co-inductive logic programming extends traditional logic programming by enabling co-inductive reasoning to deal with infinite data structures, which has practical implication in different fields of computing such as model checking, planning as well as type inference. (see [13, 12, 1, 9])

One paradgim of co-inductive logic programming is co-inductive SLD resolution (co-SLD) (see [13, 12]), which combines loop detection with traditional SLD resolution, so that it can be used to reason co-inductively about infinite cyclic terms. An alternative co-inductive logic programming paradgim is co-algebraic logic programming (see [9]), which adopts co-algebraic semantics and uses structural resolution instead of SLD resolution as its key inference rule.

Structural resolution (see [8, 7]) features reliance on rewriting reduction to reveal the productivity, i.e. the ability to produce answers, of the interpreted logic program. Also it adopts a combination of rewriting with instantiation to achieve inductive soundness.

Both SLD resolution and structural resolution are inductively sound and complete with regard to the least Herbrand model (see [14, 8]), but, as mentioned earlier, they are extended in different ways to support co-induction. It is not clear whether co-SLD style loop detection can be used with structural resolution to do co-inductive reasoning, and if the answer is affirmative, will such "crossbred" semantics distinguish itself from its "parents" by any favorable feature?

In this paper we explore combining co-SLD style loop detection with structural resolution. Operational semantics for such combination is proposed, based on experiment, and we prove its soundness with respect to the greatest Herbrand model. It will be future work to make further comparison between co-inductive structural resolution with co-SLD.

The rest of the paper is arranged as follows. In Section 2 we introduce preliminary concepts that covers substitution and unification with rational trees, co-SLD and structural resolution. Information in this section will prepare us for further theory construction in later sections. In Section 3 we introduce the semantics for co-inductive structural resolution and prove its soundness. In Section 4 we have a review of related work. Section 5 concludes the paper and Appendix A presents the implementation.

Submitted to:
CoALP-Ty 2016

2 Preliminaries

We assume readers' understanding of standard definition of *first order term* and the modeling of (possibly infinite) terms by *trees*. Details about these definition can be found in [10, 4].

Our definition for *substitution with rational trees* (referred to as *substitution* for short hereinafter) inherits the principle of substitution with finite trees in inductive logic programming (see [10]) with regard to requirement on variable occurrence, but extends the terms that can be used to intantiate variables from finite trees to rational trees.

Definition 1 (Substitution)**.** A substitution is of the form $S = \{x_1/t_1, \ldots, x_n/t_n\}$ where $x_1, \ldots, x_n$ are distinct variables, $t_1, \ldots, t_n$ are rational terms, and $\forall\, i, j \in \{1, \ldots, n\}$, x_i does not occur in t_j. Moreover, ε always denotes empty substitution, which is $\emptyset$.

Example 1. Let substitution $\theta = \{x_1/f(f(f(\ldots))), x_2/g(x_3)\}$. Applying θ to term $t = p(x_1, x_2)$ yields $t\theta = p(f(f(f(\ldots))), g(x_3))$.

Substitution is a prerequisite concept for unification. Note that when (possibly infinite) rational trees are involved in unification, the standard approach to unification involves *systems of term equations*, and *transforms* that turn equation systems to *reduced form*, which is the output of unification algorithm.(see [15, 5], see also [11] for finite tree unification).

The reduced form equation system can be further *solved* in theory to obtain solution in the domain of rational trees (see [4]). Our definition for substitution intends to refer to the exact solution for each variable, even if the solution is an infinite rational tree, instead of referring to the unsolved reduced form of equation system as in [15, 5]. We believe that such treatment of substitution can emphsis the variables that are to be instantiated and will make the theory about co-inductive structure resolution easier to formulate and understand.

Definition 2 (Unification)**.** Given two rational terms t_1 and t_2, unification is the process of finding a substitution (unifier) θ such that $t_1\theta = t_2\theta$, i.e. applying θ separately to t_1 and t_2 yields the same tree. Such relation is denoted by $t_1 \sim_\theta t_2$.

We omit details of unification algorithm for rational trees and details of solving equation systems since these topics have standard treatment (see [11, 4, 3, 5, 15]).

Term matching is a concept closely related to unification, and is prerequisite for rewriting reduction in structral resolution (see [8, 7]). We extend applicable terms for matching from finite trees to rational trees by building the concept of rational tree term matching on the concept of rational tree unification.

Definition 3 (Term Matching)**.** Given two rational terms t_1, t_2 and their unifier σ, if σ also satisfies that $t_1\sigma = t_2$, then it is said that t_1 *subsumes* t_2. Such relation is denoted by $t_1 \prec_\sigma t_2$. σ is called *matcher* for t_1 against t_2.

The symbolic notation for unification ($\sim$) and matching ($\prec$) follows [8]. Note that ($\sim$) is a symmetric relation but ($\prec$) is not symmetric, and sometimes from $t_1 \prec t_2$ it can be confusing about which term subsumes which. A mnemonic tip for the meaning of $t_1 \prec t_2$ is to regard ($\prec$) as an open mouth, and derive the meaning "t_1 subsumes t_2" from "t_1 eats t_2".

Example 2. a) $p(x) \sim_\theta p(f(x))$ where $\theta = \{x/f(f(f(\ldots)))\}$. θ is not a matcher.

b) $p(x_1, x_1) \sim_\theta p(f(y_1), y_1)$ where $\theta = \{x_1/f(f(f(\ldots))), y_1/f(f(f(\ldots)))\}$. θ is not a matcher.

c) $p(x_1, x_2) \sim_\theta p(f(y_1), y_1)$ where $\theta = \{x_1/f(y_1), x_2/y_1\}$. θ is a matcher and $p(x_1, x_2) \prec_\theta p(f(y_1), y_1)$.

We then introduce operational semantics of the well-know SLD-resolution (see [10, 14]) as a precursor of co-SLD and of structural resolution.

Definition 4 (SLD-resolution)**.** Given a logic program P and goal

$$G = \leftarrow A_1, \ldots, A_n$$

if there exist in P a clause $B_0 \leftarrow B_1, \ldots, B_m$ (with freshly renamed variables), such that $B_0 \sim_\theta A_k$ for some $k \in \{1, \ldots, n\}$, then by SLD-resolution we derive

$$G' = \leftarrow (A_1, \ldots, A_{k-1}, B_1, \ldots, B_m, A_{k+1}, \ldots, A_n)\theta.$$

In the following definition of co-SLD we introduce a set S for each predicate A in a goal, where S records all previous goals that are relevant to the (co-inductive) proof of A. Predicates in S are also known as *co-inductive hypotheses*. (see [2, 12, 13])

Definition 5 (co-SLD)**.** Given a logic program P and goal

$$G = \leftarrow (A_1, S_1), \ldots, (A_n, S_n)$$

the next goal G' can be derived by one of the following two rules:

1. If there exist in P a clause $B_0 \leftarrow B_1, \ldots, B_m$ (with freshly renamed variables), such that $B_0 \sim_\theta A_k$ for some $k \in \{1, \ldots, n\}$, then let $S' = S_k \cup \{A_k\}$, we derive

$$G' = \leftarrow \big((A_1, S_1), \ldots, (A_{k-1}, S_{k-1}), (B_1, S'), \ldots, (B_m, S'), (A_{k+1}, S_{k+1}), \ldots, (A_n, S_n)\big)\theta.$$

2. (Loop Detection) If $A_k \sim_\theta B$ for some $k \in \{1, \ldots, n\}$ and some $B \in S_k$, we derive

$$G' = \leftarrow \big((A_1, S_1), \ldots, (A_{k-1}, S_{k-1}), (A_{k+1}, S_{k+1}), \ldots, (A_n, S_n)\big)\theta.$$

Definition 6 (co-SLD Derivation)**.** Let $G_0 = \leftarrow (A_1, \emptyset), \ldots, (A_m, \emptyset)$, G_n is empty goal for some $n \geq 0$. The finite sequence of goal reduction $G_0, \ldots, G_n$ using Definition 5 constitutes a co-SLD derivation.

Example 3. Consider this program:

$p \leftarrow q.$ $\qquad$ $q \leftarrow p.$ $\qquad$ $r \leftarrow.$

Some Boolean algebra analysis shows that $\big((p \leftarrow q) \wedge (q \leftarrow p)\big) \equiv \big((p \wedge q) \vee (\neg p \wedge \neg q)\big)$. So there are two models for the program: the least model $\{r\}$ and greatest model $\{p, q, r\}$.

Using co-SLD with goal $\leftarrow (p, \emptyset), (r, \emptyset)$, the derived goal is $\leftarrow (q, \{p\}), (r, \emptyset)$ by co-SLD rule 1, then the next goal is $\leftarrow (p, \{q, p\}), (r, \emptyset)$ by co-SLD rule 1, then the next goal is $\leftarrow (r, \emptyset)$ by co-SLD rule 2, and the next goal $\leftarrow$ (i.e. empty goal) by co-SLD rule 1. The goal $\leftarrow$ terminates the derivation (successfully), proving that p, r are in the greatest model.

Definition 7 (Structural Resolution)**.** Given a logic program P and goal

$$G = \leftarrow A_1, \ldots, A_n$$

the next goal G' is derived using one of the following two rules:

1. (*Rewriting Reduction*) If there exist in P a clause $B_0 \leftarrow B_1, \ldots, B_m$ (with freshly renamed variables), such that $B_0 \prec_\theta A_k$ for some $k \in \{1, \ldots, n\}$, then we derive

$$G' = \leftarrow (A_1, \ldots, A_{k-1}, B_1, \ldots, B_m, A_{k+1}, \ldots, A_n)\theta.$$

2. (*Substitution Reduction*) If there exist in P a clause $B_0 \leftarrow B_1,\ldots,B_m$ (with freshly renamed variables), such that $B_0 \sim_\theta A_k$ but not $B_0 \prec_\theta A_k$ for some $k \in \{1,\ldots,n\}$, then we derive

$$G' = \leftarrow (A_1,\ldots,A_n)\theta.$$

Remark. About rewriting reduction, notice that it is a special case of SLD-resolution, and since matcher θ only binds variables from the renamed clause $B_0 \leftarrow B_1,\ldots,B_m$ without binding variables from goal G, the derived goal G' can also be written as

$$G' = \leftarrow A_1,\ldots,A_{k-1},(B_1,\ldots,B_m)\theta,A_{k+1},\ldots,A_n.$$

About substitution reduction, notice that it is, by nature, *universal instantiation*.

Example 4. Consider the program:

$p(f(X)) \leftarrow q(X).$ $\quad$ $q(a) \leftarrow .$ $\quad$ $r(f(a)) \leftarrow .$

Given goal $\leftarrow p(X),r(X).$ the next goal is $\leftarrow p(f(X_1)),r(f(X_1)).$ by substitution reduction on $p(X)$ (with renamed clause $p(f(X_1)) \leftarrow q(X_1).$ and unifier $\theta_1 = \{X/f(X_1)\}$), then the next goal is $\leftarrow q(X_1),r(f(X_1)).$ by rewriting reduction on $p(f(X_1))$ (with renamed clause $p(f(X_2)) \leftarrow q(X_2).$ and matcher $\{X_2/X_1\}$), then the next goal is $\leftarrow q(a),r(f(a)).$ by substitution reduction on $q(X_1)$ (with unifier $\theta_2 = \{X_1/a\}$). Two more steps of rewriting reduction derive the empty goal $\leftarrow$ which terminates successfully the resolution and the computed substitution is the composition $\theta_1\theta_2 = \{X/f(a),X_1/a\}$.

For more details on structural resolution, see [8, 7, 9]. Next we introduce the combination of structural resolution and co-SLD style loop detection.

3 Co-inductive Structural Resolution

We introduce declarative and operational semantics of co-inductive structural resolution, indicating the origin of the semantics and prove the soundness of the operational semantics.

3.1 Declarative Semantics

The declarative semantics of co-inductive structural resolution is chosen to be the greatest fixed point over the complete Herbrand base (see [10, Chapter 4][15]), as for co-SLD (see [13, 12, 2]). In fact, it was a conjecture that co-inductive structural resolution is correct w.r.t. the greatest complete Herbrand model as co-SLD is since they share the same co-induction mechanism and their inductive components (structural resolution and SLD resolution, resp.) are both sound and complete w.r.t. the least Herbrand model (see [8]).

3.2 Operational Semantics

Definition 8 (Co-inductive Structural Resolution)**.** Given a logic program P and goal

$$G = \leftarrow (A_1,S_1),\ldots,(A_n,S_n)$$

the next goal G' can be derived by one of the following three rules:

1. If there exist in P a clause $B_0 \leftarrow B_1,\ldots,B_m$ (with freshly renamed variables), such that $B_0 \prec_\theta A_k$ for some $k \in \{1,\ldots,n\}$, then let $S' = S_k \cup \{A_k\}$, we derive

$$G' = \leftarrow (A_1,S_1),\ldots,(A_{k-1},S_{k-1}),(B_1\theta,S'),\ldots,(B_m\theta,S'),(A_{k+1},S_{k+1}),\ldots,(A_n,S_n).$$

2. If there exist in P a clause $B_0 \leftarrow B_1, \ldots, B_m$ (with freshly renamed variables), such that $B_0 \sim_\theta A_k$ but not $B_0 \prec_\theta A_k$ for some $k \in \{1, \ldots, n\}$, then we derive

$$G' = \leftarrow \big((A_1, S_1), \ldots, (A_n, S_n)\big)\theta.$$

3. (Loop Detection) If $A_k \sim_\theta B$ for some $k \in \{1, \ldots, n\}$ and some $B \in S_k$, we derive

$$G' = \leftarrow \big((A_1, S_1), \ldots, (A_{k-1}, S_{k-1}), (A_{k+1}, S_{k+1}), \ldots, (A_n, S_n)\big)\theta.$$

Remark. The operational semantics of co-inductive structural resolution is formulated according to the behaviour of an *experimental implementation* (see Appendix A) that is based on:

- The idea from co-SLD that a goal predicate succeeds if it unifies with a predicate from some previous goal, and
- The realization that in the context of structural resolution, unification between A and A', where A' is a predicate from a goal G' and A is a predicate from some previous goal G, is not *trivial* only if G' is not derived from G by substitution reduction on A, which instantiated A into A' (c.f. substitution reduction made in Example 4). Trivial unification can be avoided by not adding a goal predicate A to the set of hypotheses if the next goal is derived by substitution reduction on A.

Notice also that the Loop Detection rule for co-inductive structural resolution is the same as its counterpart in co-SLD, and rule-1 of co-inductive structural resolution is a special case of rule-1 of co-SLD.

Based on Definition 8, we define co-inductive structural resolution derivation.

Definition 9 (co-S-derivation)**.** A sequence $\mathscr{D} = G_0, G_1, \ldots, G_k, \ldots$ constitutes a co-inductive structural resolution derivation if all of the following conditions hold:

(i) $\mathscr{D}$ is finite, with G_n $(n \geq 0)$ as the last goal.

(ii) G_0 is of the form $\leftarrow (A_1, \emptyset), \ldots, (A_m, \emptyset)$. $(m \geq 0)$.

(iii) If $n > 0$, then $\forall i \in [0, n-1]$, G_{i+1} is derived from G_i according to Definition 8.

(iv) If $n > 1$, then $\forall i \in [0, n-2]$, the reductions $G_i \rightarrow G_{i+1}$ and $G_{i+1} \rightarrow G_{i+2}$ are *not both* by rule-2 in Definition 8, i.e. no consecutive application of rule-2.

(v) G_n is empty goal.

Definition 10 (Computed Substitution)**.** Given a co-S-derivation $\mathscr{D}$, let $\theta_1, \theta_2, \ldots, \theta_n$ be the sequence of unifiers computed in $\mathscr{D}$ due to application of rule-2 or rule-3, sorted in the same order as they were computed, their composition $\theta_1\theta_2\cdots\theta_n$ is called the computed substitution from $\mathscr{D}$.

We then see several examples of co-S-derivation over different programs and goals, which demonstrate different aspects of its rules. Example 5 emphasizes how co-inductive hypotheses are managed by rule-1. Example 6 focuses on the influence of matcher/unifier on goals, esp. on hypotheses, when using only rule-1,2. Example 7 comprehensively shows hypotheses management, unifier/matcher influence and loop detection, and all three rules are involved.

Example 5. Consider program:

$a \leftarrow a_1, a_2.$ $\quad a_1 \leftarrow b_1, b_2.$ $\quad b_1 \leftarrow c_1, c_2.$ $\quad c_1 \leftarrow .$ $\quad c_2 \leftarrow .$
$a_2 \leftarrow .$ $\quad b_2 \leftarrow .$

We see a co-S-derivation solely by using rule-1. In each goal, we always select the left most predicate to resolve.

Goal 1: $\leftarrow (a, \emptyset)$.

Goal 2: $\leftarrow (a_1, \{a\}), (a_2, \{a\})$.

Goal 3: $\leftarrow (b_1, \{a_1, a\}), (b_2, \{a_1, a\}), (a_2, \{a\})$.

Goal 4: $\leftarrow (c_1, \{b_1, a_1, a\}), (c_2, \{b_1, a_1, a\}), (b_2, \{a_1, a\}), (a_2, \{a\})$.

Goal 5: $\leftarrow (c_2, \{b_1, a_1, a\}), (b_2, \{a_1, a\}), (a_2, \{a\})$.

Goal 6: $\leftarrow (b_2, \{a_1, a\}), (a_2, \{a\})$.

Goal 7: $\leftarrow (a_2, \{a\})$.

Goal 8: $\leftarrow$.

Example 6. Consider program:

$p(s(X)) \leftarrow$. $\quad q(s(a)) \leftarrow$. $\quad r(X) \leftarrow p(X), q(X)$.

In the following co-S-derivation, the details involved in deriving Goal $(i+1)$ from Goal (i) is noted after Goal $(i+1)$ in the format of *(rule.clause.unifier)*, providing information on the reduction rule used, the program clause used, and the computed unifier/matcher. In each goal, we always select the left most predicate to resolve.

Goal 1: $\leftarrow \big(r(X), \emptyset\big)$.

Goal 2: $\leftarrow \big(p(X), \{r(X)\}\big), \big(q(X), \{r(X)\}\big)$. (rule-1. $r(X_1) \leftarrow p(X_1), q(X_1)$. $\{X_1/X\}$)

Goal 3: $\leftarrow \big(p(s(X_2)), \{r(s(X_2))\}\big), \big(q(s(X_2)), \{r(s(X_2))\}\big)$. (rule-2. $p(s(X_2)) \leftarrow$. $\theta_1 = \{X/s(X_2)\}$)

Goal 4: $\leftarrow \big(q(s(X_2)), \{r(s(X_2))\}\big)$. (rule-1. $p(s(X_3)) \leftarrow$. $\{X_3/X_2\}$)

Goal 5: $\leftarrow \big(q(s(a)), \{r(s(a))\}\big)$. (rule-2. $q(s(a)) \leftarrow$. $\theta_2 = \{X_2/a\}$)

Goal 6: $\leftarrow$. (rule-1. $q(s(a)) \leftarrow$.)

The answer to Goal 1 is given by computed substitution $\theta_1\theta_2 = \{X/s(a), X_2/a\}$.

Example 7. Consider program:

$p(s(X)) \leftarrow q(X)$. $\quad q(X) \leftarrow p(X), r(X)$. $\quad r(X) \leftarrow$.

In the following derivation, for each goal we always select the left most predicate to resolve.

Goal 1: $\leftarrow \big(q(X), \emptyset\big)$.

Goal 2: $\leftarrow \big(p(X), \{q(X)\}\big), \big(r(X), \{q(X)\}\big)$. (rule-1. $q(X_1) \leftarrow p(X_1), r(X_1)$. $\{X_1/X\}$)

Goal 3: $\leftarrow \big(p(s(X_2)), \{q(s(X_2))\}\big), \big(r(s(X_2)), \{q(s(X_2))\}\big)$. (rule-2. $p(s(X_2)) \leftarrow q(X_2)$. $\theta_1 = \{X/s(X_2)\}$)

Goal 4: $\leftarrow \big(q(X_2), \{p(s(X_2)), q(s(X_2))\}\big), \big(r(s(X_2)), \{q(s(X_2))\}\big)$. (rule-1. $p(s(X_3)) \leftarrow q(X_3)$. $\{X_3/X_2\}$)

Goal 5: $\leftarrow \big(r(s(s(s(\ldots)))), \{q(s(s(s(\ldots))))\}\big)$. (rule-3. $q(X_2) \sim_{\theta_2} q(s(X_2))$. $\theta_2 = \{X_2/s(s(s(\ldots)))\}$)

Goal 6: $\leftarrow$. (rule-1. $r(X_4) \leftarrow$. $\{X_4/s(s(s(\ldots)))\}$)

The answer to Goal 1 is given by computed substitution $\theta_1\theta_2 = \{X/s(s(s(\ldots))), X_2/s(s(s(\ldots)))\}$. Loop detection is used once for reduction from Goal 4 to Goal 5, and predicate $q(X_2)$ in Goal 4 is co-inductively proved.

Remark. Programs used in Example 5 6 and 7 are adapted from those that were originally designed to varify that rules and conditions formulated in Definition 8 and Definition 9 correctly describe the behaviour of the experimental implementation given in Appendix A, in the sense that when the same query is made, the trace of the implementation shall be the same as the trace of applying the definitions.

3.3 Soundness Proof

We establish that soundness of co-inductive structural resolution depends on the soundness of co-SLD. We will show that given a goal G, if there is a co-inductive structural resolution derivation for G, with computed substitution σ, then there is a co-SLD derivation for $G\sigma$, with computed substitution ε (i.e. the empty substitution). Therefore if co-SLD is sound, then $G\sigma\varepsilon = G\sigma$ is in the greatest complete Herbrand model, meaning that co-inductive structural resolution is also sound.

Definition 11 (Identity Reduction)**.** Given some goal G, the reduction from G to itself, is called identity reduction, denoted by

$$G \xrightarrow{\text{id}} G$$

Definition 12 (co-Rewrting-ID Resolution)**.** Given a program and some goal G, the next goal G' can be derived from G using one of following three rules:

1. The same as rule 1 in Definition 8.
2. Identity reduction.
3. The same as rule 3 in Definition 8.

Definition 13 (co-Rewriting-ID Derivation)**.** A sequence $\mathscr{D} = G_0, G_1, \ldots, G_k, \ldots$ constitutes a co-rewriting-id derivation if all of the following conditions hold:

(i) $\mathscr{D}$ is finite, with G_n $(n \geq 0)$ as the last goal.

(ii) G_0 is of the form $\leftarrow (A_1, \emptyset), \ldots, (A_m, \emptyset)$. $(m \geq 0)$.

(iii) If $n > 0$, then $\forall i \in [0, n-1]$, G_{i+1} is derived from G_i according to Definition 12.

(iv) If $n > 1$, then $\forall i \in [0, n-2]$, the reductions $G_i \rightarrow G_{i+1}$ and $G_{i+1} \rightarrow G_{i+2}$ are *not both* by identity reduction, i.e. no consecutive application of identity reduction.

(v) G_n is empty goal.

Theorem 1. *Given a program, for any goal G, if there is a co-rewriting-id derivation for G with computed substitution θ, then there is a co-SLD derivation for G with the same computed substitution θ.*

Proof. Suppose $\mathscr{D} = G_0, \ldots, G_n$ is a co-rewriting-id derivation for $G = G_0$ with computed substitution θ. By simultaneously removing from $\mathscr{D}$ all G_{i+1} $(i \in [0, n-1])$ such that $G_i \xrightarrow{\text{id}} G_{i+1}$, the resulting derivation $\mathscr{D}'$ constitutes a co-SLD derivation with computed substitution θ. Moreover, $\mathscr{D}'$ is a special case of co-SLD derivation since Definition 12-rule 1 is a special case of Definition 5-rule 1. □

Theorem 2. *Given a program, for any goal G, if there is a co-S-derivation for G with computed substitution σ, then there is a co-rewriting-id derivation for $G\sigma$ with computed substitution ε (the empty substitution).*

The proof of Theorem 2 is based on properties of co-inductive structural resolution rules, formulated in the following three lemmas.

Lemma 1 (Rewriting Preservation)**.** *Let*

$$G \xrightarrow[B]{\textit{rule-1}} G'$$

be a goal reduction using rule-1 (as defined in Definition 8), and B is the program clause involved in the reduction.

Then for any substitution σ, it holds that

$$G\sigma \xrightarrow[B]{\textit{rule-1}} G'\sigma$$

Proof. Assume

- $G = \leftarrow (A_1,S_1),\ldots,(A_k,S_k),\ldots,(A_n,S_n)$. and
- B has the form $B_0 \leftarrow B_1,\ldots,B_m$. $(m \geq 0)$ and
- $B_0 \prec_\gamma A_k$, for some $k \in [1,n]$.

By Definition 8, rule-1,

$$G' = \leftarrow (A_1,S_1),\ldots,(A_{k-1},S_{k-1}),(B_1\gamma,S'),\ldots,(B_m\gamma,S'),(A_{k+1},S_{k+1}),\ldots,(A_n,S_n). \tag{1}$$

where $S' = S_k \cup \{A_k\}$.

Since $B_0 \prec_\gamma A_k$ (by above assumption), it means (by Definition 3) that

$$B_0\gamma = A_k \tag{2}$$

then for all σ, if we apply σ to both sides of (2), we have $B_0\gamma\sigma = A_k\sigma$, which means (by associativity of substitution (see,e.g. [10, Sec. 4]) and Definition 3) that

$$B_0 \prec_{\gamma\sigma} A_k\sigma. \tag{3}$$

Now consider $G\sigma$, by notational convention,

$$G\sigma = \leftarrow (A_1\sigma,S_1\sigma),\ldots,(A_k\sigma,S_k\sigma),\ldots(A_n\sigma,S_n\sigma). \tag{4}$$

Because of (3) and (4), we can have reduction

$$G\sigma \xrightarrow[B]{\text{rule-1}} G'' \tag{5}$$

where, by Definition 8, rule-1,

$$G'' = \leftarrow (A_1\sigma,S_1\sigma),\ldots,(A_{k-1}\sigma,S_{k-1}\sigma),(B_1\gamma\sigma,S''),\ldots,(B_m\gamma\sigma,S''),(A_{k+1}\sigma,S_{k+1}\sigma),\ldots,(A_n\sigma,S_n\sigma). \tag{6}$$

where $S'' = S_k\sigma \cup \{A_k\sigma\}$.

Compare (1) and (6), we have, by notational convention,

$$G'' = G'\sigma \tag{7}$$

By (5) and (7), we reach the conclusion of Lemma 1. □

Definition 14 (Composition Notation)**.** Given a sequence of substituions $\theta_1,\theta_2,\ldots,\theta_n$, for all $k \in [1,n]$, let σ_k denote the composition $\theta_k\theta_{k+1}\cdots\theta_n$.

Example 8. Let $\theta_1,\theta_2,\theta_3,\theta_4$ be a sequence of 4 substitutions, then $\sigma_1 = \theta_1\theta_2\theta_3\theta_4$, $\sigma_2 = \theta_2\theta_3\theta_4$, $\sigma_3 = \theta_3\theta_4$ and $\sigma_4 = \theta_4$.

Lemma 2 (Instantiation Preservation)**.** *Let*

$$G \xrightarrow[\theta_k]{\textit{rule-2}} G'$$

be a goal reduction using rule-2 (as defined in Definition 8), where θ_k is the unifier involved in the reduction, and let

$$\sigma_k = \theta_k\theta_{k+1}\cdots\theta_n$$

for some $n > k$ and some (possibly ε) substitutions $\theta_{k+1},\ldots,\theta_n$.

Then

$$G\sigma_k \xrightarrow{id} G'\sigma_{k+1}$$

Proof. From the premise of Lemma 2, we have

$$G\theta_k = G' \tag{8}$$

Applying $\sigma_{k+1}(= \theta_{k+1}\cdots\theta_n)$ to both sides of (8) we have

$$G\theta_k\sigma_{k+1} = G'\sigma_{k+1} \tag{9}$$

Note that in (9)

$$\theta_k\sigma_{k+1} = \sigma_k$$

therefore by associativity of substitution (see, e.g. [10, Sec. 4]) (9) can be written as

$$G\sigma_k = G'\sigma_{k+1}$$

hence the identity reduction $G\sigma_k \xrightarrow{\text{id}} G'\sigma_{k+1}$. □

Lemma 3 (Loop Detection Preservation)**.** *Let*

$$G \xrightarrow[\theta_k]{\textit{rule-3}} G'$$

be a goal reduction using rule-3 (as defined in Definition 8), where θ_k is the unifier involved in the reduction, and let

$$\sigma_k = \theta_k\theta_{k+1}\cdots\theta_n$$

for some $n > k$ and some (possibly ε) substitutions $\theta_{k+1},\ldots,\theta_n$.

Then

$$G\sigma_k \xrightarrow[\varepsilon]{\textit{rule-3}} G'\sigma_{k+1}$$

Proof. Assume

$$G = \leftarrow (A_1,S_1),\ldots,(A_k,S_k),\ldots,(A_n,S_n). \tag{10}$$

and

$$A_k \sim_{\theta_k} B \tag{11}$$

for some $k \in [1,n]$ and some

$$B \in S_k \tag{12}$$

By Definition 8, rule-3,

$$G' = \leftarrow \big((A_1,S_1),\ldots,(A_{k-1},S_{k-1}),(A_{k+1},S_{k+1}),\ldots,(A_n,S_n)\big)\theta_k. \tag{13}$$

From (11) and Definition 2,

$$A_k\theta_k = B\theta_k \tag{14}$$

Applying $\sigma_{k+1}(= \theta_{k+1}\cdots\theta_n)$ to both sides of (14), we have $A_k\theta_k\sigma_{k+1} = B\theta_k\sigma_{k+1}$, then due to $\sigma_k = \theta_k\sigma_{k+1}$ and the associativity of substitution,

$$A_k\sigma_k = B\sigma_k \tag{15}$$

which means

$$A_k\sigma_k \sim_\varepsilon B\sigma_k \tag{16}$$

Consider $G\sigma_k$, which can be written as

$$G\sigma_k = \leftarrow (A_1\sigma_k, S_1\sigma_k), \ldots, (A_k\sigma_k, S_k\sigma_k), \ldots, (A_n\sigma_k, S_n\sigma_k) \tag{17}$$

Due to (12), it holds that

$$B\sigma_k \in S_K\sigma_k \tag{18}$$

From (16) and (18), $G\sigma_k$ as in (17) can be reduced using Definition 8 rule 3, resulting in

$$G'' = \leftarrow (A_1\sigma_k, S_1\sigma_k), \ldots, (A_{k-1}\sigma_k, S_{k-1}\sigma_k), (A_{k+1}\sigma_k, S_{k+1}\sigma_k), \ldots, (A_n\sigma_k, S_n\sigma_k)$$

which can be rewritten in a simpler form

$$G'' = \leftarrow \big((A_1, S_1), \ldots, (A_{k-1}, S_{k-1}), (A_{k+1}, S_{k+1}), \ldots, (A_n, S_n)\big)\sigma_k \tag{19}$$

The reduction of $G\sigma_k$ is denoted by

$$G\sigma_k \xrightarrow[\varepsilon]{\text{rule-3}} G'' \tag{20}$$

Comparing (13) with (19), we conclude, by associativity of substitution, that

$$G'' = G'\sigma_{k+1}$$

and with (20) we reach the conclusion of Lemma 3. □

Next we give an algorithm that outputs co-rewriting-id derivation, giving a co-S-derivation as input. This algorithm constitutes our proof of Theorem 2 and we will provide an example to demonstrate the algorithm at work.

Proof of Theorem 2. Given a goal $G = G_0$, assume $\mathscr{D} = G_0, \ldots, G_n$ is a co-S-derivation, and $\theta_1, \ldots \theta_m$ is the sequence of unifiers computed during derivation $\mathscr{D}$ due to the use of Definition 8 rule 2 or 3. Let $\theta_{m+1} = \varepsilon$ so $\sigma_1 = \theta_1, \ldots \theta_m\theta_{m+1}$ is the computed substitution for goal G.

Note our composition notation in Definition 14. Also Definition 8 is the default domain when we mention rule 1,2 or 3 in this proof. We build the co-rewriting-id derivation $\mathscr{D}'$ of $G\sigma_1$ using the following algorithm.

For each $i \in [0, n-1]$, *starting from* $i = 0$ *and in the ascending order of* i,

- If
$$G_i \xrightarrow[B]{\text{rule-1}} G_{i+1}$$
then write down, by Lemma 1,
$$G_i\sigma_x \xrightarrow[B]{\text{rule-1}} G_{i+1}\sigma_x$$
where
$$x = \begin{cases} 1 & \text{If } i = 0; \\ k & \text{If } i > 0 \text{ and } G_i\sigma_k \text{ is in } \mathscr{D}'. \end{cases}$$
- If
$$G_i \xrightarrow[\theta_k]{\text{rule-2}} G_{i+1}$$
then write down, by Lemma 2,
$$G_i\sigma_k \xrightarrow{\text{id}} G_{i+1}\sigma_{k+1}$$

- If
$$G_i \xrightarrow[\theta_k]{\text{rule-3}} G_{i+1}$$
then write down, by Lemma 3,
$$G_i\sigma_k \xrightarrow[\varepsilon]{\text{rule-3}} G_{i+1}\sigma_{k+1}$$

□

Example 9. Consider the specific co-S-derivation $\mathscr{D}$ for goal G:

$$(G = G_0) \xrightarrow[B_1]{rule-1} G_1 \xrightarrow[B_2]{rule-1} G_2 \xrightarrow[\theta_1]{rule-2} G_3 \xrightarrow[\theta_2]{rule-3} G_4 \xrightarrow[B_3]{rule-1} G_5 \xrightarrow[\theta_3]{rule-3} (G_6 = \leftarrow .)$$

The computed substitution is $\sigma_1 = \theta_1\theta_2\theta_3\theta_4$ where $\theta_4 = \varepsilon$. Using the algorithm we can get co-rewriting-id derivation $\mathscr{D}'$ for goal $G\sigma_1$:

$$(G\sigma_1 = G_0\sigma_1) \xrightarrow[B_1]{rule-1} G_1\sigma_1 \xrightarrow[B_2]{rule-1} G_2\sigma_1 \xrightarrow{id} G_3\sigma_2 \xrightarrow[\varepsilon]{rule-3} G_4\sigma_3 \xrightarrow[B_3]{rule-1} G_5\sigma_3 \xrightarrow[\varepsilon]{rule-3} (G_6\sigma_4 = \leftarrow .)$$

Theorem 3 (Soundness)**.** *Co-inductive structural resolution is sound with respect to the greatest complete Herbrand model. In other words, if a goal G has a co-S-derivation with computed substitution σ, then $G\sigma$ is in the greatest model.*

Proof. Given a program and some goal G, assume G has a co-S-derivation with computed substitution σ. By Theorem 2 $G\sigma$ has a co-rewriting-id derivation with computed substitution ε, then by Theorem 1 $G\sigma$ has a co-SLD derivation with computed substitution ε. Since co-SLD is sound with respect to the greatest complete Herbrand model, $G\sigma\varepsilon = G\sigma$ is the model. □

4 Related Work

A soundness proof of co-SLD, based on co-induction, is provided in [12], in which it was established by a lemma that if some goal G has co-SLD derivation with computed substitution σ, then $G\sigma$ also has a co-SLD derivation. This idea inspired the author to explore if a goal has a co-S-derivation, whether there is also a co-S-derivation for the same goal with computed answer applied.

Another soundness proof of co-SLD is given in [2], which is based on infinite tree logic programming theory formulated in [6]. The P-derivation based on a breadth first search of a possibly infinite SLD tree is proposed in [6] which is sound with respect to the greatest model. In [2] it was proved that SLD resolution with rational trees has a successful derivation for a goal G iff the P-derivation does. In the same paper the soundness of co-SLD is proved by showing that any co-SLD derivation can be unfolded into a P-derivation, therefore the soundness of co-SLD derivation is backed by the soundness of P-derivation. Our proof in this paper obviously is inspired such technique in [2], which relates different derivations and reuses previous results.

5 Conclusion and Discussion

We proposed a way to combine loop detection with structural resolution, resulting in semantics of co-inductive structural resolution. We have shown that soundness of co-inductive structural resolution depends on soundness of co-SLD.

As a by-product of our proof, we can use the same arguments to show that soundness of structural resolution depends on soundness of SLD resolution, because if a goal G has successful structural resolution derivation with computed substitution σ, then $G\sigma$ has a rewriting-id derivation, which is a special case of SLD derivation.

6 Acknowledgement

Thanks my supervisor Dr. Ekaterina Komendantskaya for her support and discussion.

References

[1] Davide Ancona, Andrea Corradi, Giovanni Lagorio & Ferruccio Damiani (2010): *Abstract Compilation of Object-Oriented Languages into Coinductive CLP(X): Can Type Inference Meet Verification?* In: *Formal Verification of Object-Oriented Software - International Conference, FoVeOOS 2010, Paris, France, June 28-30, 2010, Revised Selected Papers*, pp. 31–45, doi:10.1007/978-3-642-18070-5_3. Available at `http://dx.doi.org/10.1007/978-3-642-18070-5_3`.

[2] Davide Ancona & Agostino Dovier (2015): *A Theoretical Perspective of Coinductive Logic Programming*. *Fundam. Inform.* 140(3-4), pp. 221–246, doi:10.3233/FI-2015-1252. Available at `https://users.dimi.uniud.it/~agostino.dovier/PAPERS/FUIN15rid.pdf`.

[3] Alain Colmerauer (1985): *Prolog in 10 Figures*. *Commun. ACM* 28(12), pp. 1296–1310, doi:10.1145/214956.214958. Available at `http://doi.acm.org/10.1145/214956.214958`.

[4] Bruno Courcelle (1983): *Fundamental properties of infinite trees*. *Theoretical Computer Science* 25(2), pp. 95 – 169, doi:http://dx.doi.org/10.1016/0304-3975(83)90059-2. Available at `http://www.sciencedirect.com/science/article/pii/0304397583900592`.

[5] M.H. van Emden & J.W. Lloyd (1984): *A logical reconstruction of Prolog II*. *The Journal of Logic Programming* 1(2), pp. 143 – 149, doi:http://dx.doi.org/10.1016/0743-1066(84)90001-3. Available at `http://www.sciencedirect.com/science/article/pii/0743106684900013`.

[6] Joxan Jaffar & Peter J. Stuckey (1986): *Semantics of Infinite Tree Logic Programming*. *Theor. Comput. Sci.* 46(2-3), pp. 141–158. Available at `http://dl.acm.org/citation.cfm?id=21545.21547`.

[7] Patricia Johann, Ekaterina Komendantskaya & Vladimir Komendantskiy (2015): *Structural Resolution for Logic Programming*. *CoRR* abs/1507.06010. Available at `http://arxiv.org/abs/1507.06010`.

[8] Ekaterina Komendantskaya & Patricia Johann (2015): *Structural Resolution: a Framework for Coinductive Proof Search and Proof Construction in Horn Clause Logic*. *CoRR* abs/1511.07865. Available at `http://arxiv.org/abs/1511.07865`.

[9] Ekaterina Komendantskaya, John Power & Martin Schmidt (2016): *Coalgebraic logic programming: from Semantics to Implementation*. *Journal of Logic and Computation* 26(2), p. 745, doi:10.1093/logcom/exu026. Available at `+http://dx.doi.org/10.1093/logcom/exu026`.

[10] J. W. Lloyd (1987): *Foundations of Logic Programming; (2Nd Extended Ed.)*. Springer-Verlag New York, Inc., New York, NY, USA.

[11] Alberto Martelli & Ugo Montanari (1982): *An Efficient Unification Algorithm*. *ACM Trans. Program. Lang. Syst.* 4(2), pp. 258–282, doi:10.1145/357162.357169. Available at `http://doi.acm.org/10.1145/357162.357169`.

[12] Luke Simon (2006): *Extending Logic Programming with Coinduction*. Ph.D. thesis, The University of Texas at Dallas. Available at `http://www.utdallas.edu/~gupta/lukethesis.pdf`.

[13] Luke Simon, Ajay Mallya, Ajay Bansal & Gopal Gupta (2006): *Coinductive Logic Programming*, pp. 330–345. Springer Berlin Heidelberg, Berlin, Heidelberg, doi:10.1007/11799573_25.

[14] M. H. Van Emden & R. A. Kowalski (1976): *The Semantics of Predicate Logic As a Programming Language*. *J. ACM* 23(4), pp. 733–742, doi:10.1145/321978.321991. Available at `http://doi.acm.org/10.1145/321978.321991`.

[15] W. P. Weijland (1988): *Semantics for logic programs without occur check*, pp. 710–726. Springer Berlin Heidelberg, Berlin, Heidelberg, doi:10.1007/3-540-19488-6_152. Available at `http://dx.doi.org/10.1007/3-540-19488-6_152`.

A Implementation of Co-Inductive Structural Resolution

SWI-Prolog (Multi-threaded, 64 bits, Version 7.2.3)
http://www.swi-prolog.org

```
%-----------------------------------------------------------
clause_tree(true,_) :- !.
clause_tree((G,R), Hypo) :-
   !,                                                  % Note 1
   clause_tree(G, Hypo),
   clause_tree(R, Hypo).

clause_tree(A,Hypo) :- find_loop(A,Hypo).

clause_tree(A, Hypo) :-    % rewriting reduction
            unifying_and_matching_rule(A, Body),
            clause_tree(Body, [A|Hypo]).                % Note 2

clause_tree(A, Hypo) :-    % substitution reduction.
            unifying_but_matching_rule(A, _),
            clause_tree(A, Hypo).                       % Note 3

%-----------------------------------------------------------
find_loop(A,[B|_]) :- A = B.
find_loop(A,[_|C]) :- find_loop(A,C).

% choose clauses whose heads unifies with the goal,
% and specificly, matches the goal.
unifying_and_matching_rule(A, Body) :-
         copy_term(A,A_copy),                           % Note 4
         clause(A_copy,_,Ref),                          % Note 5
         clause(A1,_,Ref),                              % Note 6
         subsumes_term(A1,A),                           % Note 7
         clause(A,Body,Ref).                            % Note 8

% choose clauses whose head unifies with the goal,
% and specificly, does not match the goal.
unifying_but_matching_rule(A, Body) :-
        copy_term(A,A_copy),
        clause(A_copy,_,Ref),
        clause(A1,_,Ref),
        \+ subsumes_term(A1,A),
        clause(A,Body,Ref).
%-----------------------------------------------------------
```

Note 1 Clauses deal with mutually exclusive cases, hence the cuts.

Note 2 `A` is not instantiated by finding a matching clause.

Note 3 `A` is instantiated by finding a unifiying but matching clause.

Note 4 At run time variable `A` is bound to the current (atomic sub-)goal G, `A_copy` then is a variant $\acute{G}$ of G with fresh variables. Built-in `copy_term/2` is used to make a copy of G to use in the next procedure `clause(A_copy,_,Ref)` to search for a unifying rule without instantiating variables from G.

Note 5 At run time, this procedure finds some clause whose head unifies with the variant $\acute{G}$ of the current (atomic sub-)goal G and get the clause's reference number n that is bound to `Ref`. The term $\acute{G}$ bound to `A_copy` may be instantiated. The body of the found clause, which may be instantiated, is disgarded as indicated by "_".

Note 6 Use the reference number `Ref` to get a copy of the found clause by the previous procedure '`clause(A_copy,_,Ref)`'. Only the head, which is bound to `A1`, of the clause is needed for subsumes check, and the body of the clause is disgarded as shown by '_'.

Note 7 Term matching is checked by using built-in predicate `subsumes_term/2`, which does not instantiate variables. Any binding made for subsumes check will be undone by implementation of `subsumes_term/2`.

Note 8 If the subsumes check is passed by the found unifying clause, then use a fresh copy of this particular clause, as specified by `Ref`, to reduce the goal. Variables in the term t bound to `A` will not be instantiated because the clause head subsumes t as has been checked, but variables in the body of the clause, which is bound to `Body`, are instantiated by sub-terms from t.

```
% Example object programs
% -------------------------
% trace: clause_tree(a,[])
a  :- a1,a2.
a1 :- b1,b2.
b1 :- c1,c2.
a2.
b2.
c1.
c2.
%-------------------------
% trace: clause_tree(p(X),[]).
p(s(X)) :- p(X),p(X).  % non-linear co-recursion
%-------------------------
% trace: clause_tree(cond_f(X),[]).
cond_c(s(a)).
cond_e(s(_)).
cond_f(X) :- cond_e(X),cond_c(X).
%-------------------------
% trace: clause_tree(q(X),[]).
r(_).
p(s(X)) :- q(X).
q(X) :- p(X),r(X).
```