Semantics for Conditional Literals via the SM Operator

Zachary Hansen[1][0000−0002−8447−4048]⋆ and
Yuliya Lierler[1][0000−0002−6146−623X]⋆

University of Nebraska Omaha, Omaha NE 68182, USA
{zachhansen,ylierler}@unomaha.edu

Abstract. Conditional literals are an expressive Answer Set Programming language construct supported by the solver CLINGO. Their semantics are currently defined by a translation to infinitary propositional logic, however, we develop an alternative characterization with the SM operator which does not rely on grounding. This allows us to reason about the behavior of a broad class of CLINGO programs/encodings containing conditional literals, without referring to a particular input/instance of an encoding. We formalize the intuition that conditional literals behave as nested implications, and prove the equivalence of our semantics to those implemented by CLINGO.

Keywords: ASP · Semantics · Conditional Literals

1 Introduction

Answer Set Programming (ASP) [13,14] is a widely utilized branch of logic programming that combines an expressive modeling language with efficient grounders and solvers. It has found a number of prominent applications since its inception, such as diagnostic AI and space shuttle decision support systems [1]. For various classes of logic programs, there are multiple equivalent ways to characterize their semantics [12]. Most of the semantics for non-propositional programs (used in the practice of ASP) are defined via grounding – a process of instantiating variables for passing constants. This often makes it difficult to reason about parts of logic programs in isolation. The SM operator [7] is one of the few approaches to interpreting logic programs without reference to grounding. In this approach, a logic program is viewed as an abbreviation for a first-order sentence. The semantics of a program are defined by means of an application of the SM operator to the program, which results in a second-order formula. The Herbrand models of this formula coincide with the answer sets of the considered logic program. In this work, we extend the class of programs to which the SM operator is applicable by providing a translation to first-order formulas for constructs known as "conditional" literals. We then illustrate that the newly defined semantics via the SM operator for programs with conditional literals coincides with that by

⋆ These authors contributed equally.

Harrison, Lifschitz, and Yang (2014). In that work, the authors captured the meaning of programs with conditional literals via grounding/transformation of a logic program into infinitary propositional formulas. Importantly, the answer set system CLINGO [9] obeys their semantics.

Rules with conditional literals are common in so-called meta-programming [11], where reification of constructs is utilized to build ASP-based reasoning engines that may go beyond the ASP paradigm itself. The meta-programming technique is well illustrated in [11] by means of multiple examples, where conditional literals are widespread. For instance, Kaminski et al. (2021) use this technique in the implementation of optimization statements, reasoning about actions, reasoning about preferences, and guess-and-check programming. Here, we showcase the utility of conditional literals on the well-studied graph (k) coloring problem. This problem has a simple specification: *For an undirected graph, the k-coloring problem assigns one of k colors to each vertex such that no two vertices connected by an edge share a color.* It is an NP-complete problem to decide if a given graph admits a k-coloring for any $k \geq 3$. This problem can be elegantly encoded in a few lines of ASP code. First consider the encoding in Listing 1.1, where $color(I;J)$ and $vtx(V;W)$ abbreviate expressions $color(I);color(J)$ and $vtx(V);vtx(W)$, respectively. This is an instructional encoding by Lierler (2017, Section 5) modulo the changes in predicate names.

Listing 1.1. 3-coloring problem encoding.

```
1 {asg(V,I)} :- vtx(V); color(I).
2 :- not asg(V,r); not asg(V,g); not asg(V,b); vtx(V).
3 :- asg(V,I); asg(V,J); I != J; vtx(V); color(I;J).
4 :- asg(V,I); asg(W,I); vtx(V;W); color(I); edge(V,W).
```

Given an instance of a graph, the program in Listing 1.1 assigns colors from the set $\{r, g, b\}$ to its vertices. Yet, however concise and self-explanatory this solution is, it lacks elaboration tolerance. It is restricted to the color names hard-coded into the program, and it only solves the 3-coloring problem as opposed to the k-coloring problem. Conditional literals provide us with a convenient means to address this shortcoming. Consider the encoding in Listing 1.1 with line 2 replaced by

```
:- not asg(V,I) : color(I); vtx(V).        (1)
```

where expression $not\ asg(V,I) : color(I)$ constitutes a conditional literal. The original rule in line 2 forbade solutions that did not assign any of the three colors to a vertex. In rule (1), the conditional literal is satisfied when no colors are assigned to a given vertex. In the sequel, we refer to the program consisting of the rules in lines 1, 3, and 4 of Listing 1.1 and rule (1) as the *k-coloring* encoding. Note how the *k-coloring* encoding is agnostic to the naming and number of colors, providing us with a truly elaboration tolerant solution for the k-coloring problem.

The remainder of this paper is organized as follows. Section 2 starts by presenting the syntax of logic programs considered in this paper. These programs contain conditional literals and we call them conditional programs. Section 3 continues by defining a translation from conditional programs to first-order formulas, and uses the SM operator to provide their semantics. Section 4 describes

the semantics of conditional programs using infinitary propositional logic. In Section 5 we connect these two characterizations of conditional programs, formally illustrating that they coincide. Thus, the SM-based semantics introduced here for conditional programs captures the behavior of CLINGO. In Section 6, we illustrate the utility of the SM-based semantics by arguing the correctness of the *k-coloring* program.

2 Syntax of Conditional Programs

In this section, we introduce a fragment of the input language of CLINGO with conditional literals. A *term* is a variable, or an object constant, or an expression of the form $f(\mathbf{t})$, where f is a function constant of arity k and $\mathbf{t}$ is a k-tuple of terms. When a term does not contain variables we call it *ground*. An *atomic formula* is either (i) an expression of the form $t = t'$ where t and t' are terms or (ii) an *atom* $p(t_1, \dots, t_n)$, where p is a predicate symbol of arity n and t_i $(1 \leq i \leq n)$ is a term; if $n = 0$, we omit the parentheses and write p (p is a propositional atom). A *basic literal* is an atomic formula optionally preceded by *not*; we identify a basic literal of the form *not* $t = t'$ with the expression $t \neq t'$. A *conditional literal* is an expression of the form $H : l_1, \dots, l_m$, where H is a basic literal or the symbol $\bot$ (denoting falsity) and $l_1, \dots, l_m$ is a nonempty list of basic literals. We often abbreviate such an expression as $H : \mathbf{L}$.

A *(conditional logic) program* is a finite set of *rules* of the form

$$H_1 \mid \cdots \mid H_m \leftarrow B_1; \dots; B_n. \tag{2}$$

$(m, n \geq 0)$, where each H_i, B_i is a basic or conditional literal; if $m = 0$ then we identify the *head* of the rule (left hand side of rule operator $\leftarrow$) with $\bot$. The right hand side of the rule operator $\leftarrow$ is called the *body*. We call a rule, where $m = 1$ and $n = 0$ a *fact*. We consider rules of the form $\{p(\mathbf{t})\} \leftarrow B_1; \dots; B_n$ to be shorthand for $p(\mathbf{t}) \mid \mathit{not}\ p(\mathbf{t}) \leftarrow B_1; \dots; B_n$, where p is a predicate constant of arity k and $\mathbf{t}$ is a k-tuple of terms.

Let $\sigma = (\mathcal{O}, \mathcal{F}, \mathcal{P})$ be a signature of a first-order language, where $\mathcal{O}$ is the set of object constants, $\mathcal{F}$ is the set of function constants of non-zero arity, and $\mathcal{P}$ is the set of predicate constants; by $\mathcal{G}_\sigma$ we denote the set of all ground terms that one may construct from the sets $\mathcal{O}$ and $\mathcal{F}$ of σ. For example, take $\mathcal{O} = \{a\}$ and $\mathcal{F} = \{f/1\}$ in some σ: then, $\mathcal{G}_\sigma = \{a, f(a), f(f(a)), \dots\}$. It is customary in logic programming that a program defines its signature implicitly, yet here it is convenient to make it explicit. For a program Π, we refer to its signature as a triple $(\mathcal{O}_\Pi, \mathcal{F}_\Pi, \mathcal{P}_\Pi)$, where $\mathcal{O}_\Pi$, $\mathcal{F}_\Pi$, and $\mathcal{P}_\Pi$ contain all the object constants, function symbols of non-zero arity, and predicate constants occurring in Π, respectively. To simplify the notation, we use $\mathcal{G}_\Pi$ to denote $\mathcal{G}_{(\mathcal{O}_\Pi, \mathcal{F}_\Pi, \mathcal{P}_\Pi)}$.

3 Semantics via the SM Operator

We now propose a syntactic transformation ϕ from logic programs to first-order sentences. The majority of this translation is implicitly described in [7, Section

2.1], where rules are viewed as an alternative notation for particular types of first-order sentences. We extend these ideas to programs with conditional literals.

First, we define some required concepts. A variable is *global* in a conditional literal $H : \mathbf{L}$ if it occurs in H but not in $\mathbf{L}$. Given a rule, all variables occurring in basic literals outside of conditional literals are considered *global*. A variable is *global in a rule* if it is global in one or more of the rule's literals. For example, in rule (1), variable V is global whereas variable I is not. Let R be a rule of the form (2) such that at least one H_i or B_i in the rule is a conditional literal $\mathcal{L}$. Let $\mathbf{v}$ be the list of variables occurring in $\mathcal{L}$. Let $\mathbf{z}$ be the list of the global variables in R. Then we call $\mathbf{x} = \mathbf{v} \setminus \mathbf{z}$ the *local* variables of $\mathcal{L}$ within R.

To transform a rule R of the form (2) into a first order sentence, we define a translation $\phi_{\mathbf{z}}$, where $\mathbf{z}$ is the list of global variables occurring in R:

1. $\phi_{\mathbf{z}}(\bot)$ is $\bot$;
2. $\phi_{\mathbf{z}}(A)$ is A for an atomic formula A;
3. $\phi_{\mathbf{z}}(not\ A)$ is $\neg\phi_{\mathbf{z}}A$ for an atomic formula A;
4. $\phi_{\mathbf{z}}(\mathbf{L})$ is $\phi_{\mathbf{z}}(l_1) \wedge \cdots \wedge \phi_{\mathbf{z}}(l_m)$ for a list $\mathbf{L}$ of basic literals;
5. for a conditional literal $H : \mathbf{L}$ occurring in the body of R with local variables $\mathbf{x}$, $\phi_{\mathbf{z}}(H : \mathbf{L})$ is $\forall\mathbf{x}\,(\phi_{\mathbf{z}}(\mathbf{L}) \rightarrow \phi_{\mathbf{z}}(H))$;
6. for a conditional literal $H : \mathbf{L}$ occurring in the head of R with local variables $\mathbf{x}$, $\phi_{\mathbf{z}}(H : \mathbf{L})$ is $\exists\mathbf{x}\big(\,(\phi_{\mathbf{z}}(\mathbf{L}) \rightarrow \phi_{\mathbf{z}}(H)) \wedge \neg\neg\phi_{\mathbf{z}}(\mathbf{L})\big)$.

Recall rule (1) containing the conditional literal $not\ asg(V, I) : color(I)$. Variable V is the only global variable of that rule, whereas variable I is local within this conditional literal. Hence, ϕ_V turns this conditional literal into formula $\forall i\,(color(i) \rightarrow \neg asg(v, i))$, where in accordance with the convention of first-order logic we turn variables into lower case.

We now define the translation ϕ on rules and programs as follows:

1. for every rule R of the form (2), its translation $\phi(R)$ is the formula

$$\forall\mathbf{z}(\phi_{\mathbf{z}}(B_1) \wedge \cdots \wedge \phi_{\mathbf{z}}(B_n) \rightarrow \phi_{\mathbf{z}}(H_1) \vee \cdots \vee \phi_{\mathbf{z}}(H_m)),$$

   where $\mathbf{z}$ is the list of the global variables of R;
2. for every program Π, its translation $\phi(\Pi)$ is the first-order sentence formed by the conjunction of $\phi(R)$ for every rule R in Π.

As a result, the rules of the *k-coloring* conditional program discussed in the Introduction are identified with the following sentences by translation ϕ:

$$\forall vi\big((vtx(v) \wedge color(i)) \rightarrow asg(v, i) \vee \neg asg(v, i)\big) \tag{3}$$

$$\forall v\big(\,(\forall i(color(i) \rightarrow \neg asg(v, i)) \wedge vtx(v)) \rightarrow \bot\big) \tag{4}$$

$$\forall vij\big((asg(v, i) \wedge asg(v, j) \wedge i \neq j \wedge vtx(v) \wedge color(i) \wedge color(j)) \rightarrow \bot\big) \tag{5}$$

$$\forall viw\big((asg(v, i) \wedge asg(w, i) \wedge vtx(v; w) \wedge color(i) \wedge edge(v, w)) \rightarrow \bot\big) \tag{6}$$

where $vtx(v; w)$ abbreviates $vtx(v) \wedge vtx(w)$. The first-order sentence corresponding to the *k-coloring* program consists of the conjunction of formulas (3-6). We refer to this first-order sentence as KC.

We now review the operator SM following Ferraris, Lee, and Lifschitz (2011). The symbols $\bot, \wedge, \vee, \rightarrow, \forall$, and $\exists$ are viewed as primitives. The formulas $\neg F$

and $\top$ are abbreviations for $F \to \bot$ and $\bot \to \bot$, respectively. If p and q are predicate symbols of arity n then $p \leq q$ is an abbreviation for the formula $\forall \mathbf{x}(p(\mathbf{x}) \to q(\mathbf{x}))$, where $\mathbf{x}$ is a tuple of variables of length n. If $\mathbf{p}$ and $\mathbf{q}$ are tuples $p_1, \ldots, p_n$ and $q_1, \ldots, q_n$ of predicate symbols then $\mathbf{p} \leq \mathbf{q}$ is an abbreviation for the conjunction $(p_1 \leq q_1) \wedge \cdots \wedge (p_n \leq q_n)$, and $\mathbf{p} < \mathbf{q}$ is an abbreviation for $(\mathbf{p} \leq \mathbf{q}) \wedge \neg(\mathbf{q} \leq \mathbf{p})$. We apply the same notation to tuples of predicate variables in second-order logic formulas. If $\mathbf{p}$ is a tuple of predicate symbols $p_1, \ldots, p_n$ (not including equality), and F is a first-order sentence then $\mathrm{SM}_{\mathbf{p}}[F]$ (called the *stable model operator with intensional predicates* $\mathbf{p}$) denotes the second-order sentence $F \wedge \neg\exists \mathbf{u}(\mathbf{u} < \mathbf{p}) \wedge F^*(\mathbf{u})$, where $\mathbf{u}$ is a tuple of distinct predicate variables $u_1, \ldots, u_n$, and $F^*(\mathbf{u})$ is defined recursively:

- $p_i(\mathbf{t})^*$ is $u_i(\mathbf{t})$ for any tuple $\mathbf{t}$ of terms;
- F^* is F for any atomic formula F that does not contain members of $\mathbf{p}$;
- $(F \wedge G)^*$ is $F^* \wedge G^*$;
- $(F \vee G)^*$ is $F^* \vee G^*$;
- $(F \to G)^*$ is $(F^* \to G^*) \wedge (F \to G)$;
- $(\forall x F)^*$ is $\forall x F^*$;
- $(\exists x F)^*$ is $\exists x F^*$.

We define the semantics of conditional logic programs using the SM operator. We note that if $\mathbf{p}$ is the empty tuple then $\mathrm{SM}_{\mathbf{p}}[F]$ is equivalent to F. We call an interpretation a $\mathbf{p}$-*stable model* of F when it is a model of $\mathrm{SM}_{\mathbf{p}}[F]$. For a conditional logic program Π and a Herbrand interpretation I over the signature $(\mathcal{O}_\Pi, \mathcal{F}_\Pi, \mathcal{P}_\Pi)$, I is an *answer set* of Π when I is a $\mathcal{P}_\Pi$-stable model of $\phi(\Pi)$.

As is customary, the concept of an answer set is defined for Herbrand interpretations. It is common to identify Herbrand interpretations with the set of ground atoms that are evaluated to true by this interpretation. When convenient, we follow this convention. It is worth noting that dropping the word Herbrand from the definition of an answer set allows us to extend the notion of an answer set/stable model to non-Herbrand interpretations following, for example, the tradition of [7,15]. Also, the provided definitions allow us to consider $\mathbf{p}$-stable models of conditional programs, where $\mathbf{p}$ is a tuple of predicate symbols in Π to characterize interesting properties of conditional programs. We articulate the utility of $\mathbf{p}$-stable models in Section 6. In that section we extend the *k-coloring* program with a sample set of facts encoding an instance of the *k-coloring* problem and argue how the answer sets of the resulting program correspond to the solutions of this instance.

4 Semantics via Infinitary Propositional Logic

Programs with conditional literals were first formalized by (i) their reduction to infinitary (propositional logic) formulas [10] and (ii) utilizing the definition of a stable model for such formulas introduced by Truszczyński (2012). We refer the reader to Definition 1 in [15] for the details on what constitutes a stable model for an infinitary formula as its details are not required in understanding the content of this paper. We now review the syntax of an infinitary formula and

the details of the translation of conditional programs to infinitary formulas by Harrison, Lifschitz, and Yang (2014).

Let A be a set of ground atoms. We define sets $\mathcal{F}_0, \mathcal{F}_1, \dots$ by induction:

- $\mathcal{F}_0 = A$;
- $\mathcal{F}_{i+1}$ adds to $\mathcal{F}_i$ all expressions $\mathcal{H}^\wedge, \mathcal{H}^\vee$ for all subsets $\mathcal{H}$ of $\mathcal{F}_i$, and $F \to G$ for all $F, G \in \mathcal{F}_i$.

The set of *infinitary formulas* over A is defined as $\cup_{i=0}^{\infty}\mathcal{F}_i$. $\{F, G\}^\wedge$ can be written as $F \wedge G$, and $\{F, G\}^\vee$ can be written as $F \vee G$. The symbols $\bot$ and $\top$ will be understood as abbreviations for $\emptyset^\vee$ and $\emptyset^\wedge$, respectively. Expression $\neg F$ is understood as $F \to \bot$.

Harrison, Lifschitz, and Yang (2014) define the semantics of programs with conditional literals using a translation τ that transforms rules of a given program into infinitary propositional formulas. It is worth noting that they allow a broader syntactic class of rules than we consider here. For instance, rules with aggregates are allowed; these rules are outside the scope of this paper, but we refer interested readers to [4] for a review of how the SM operator can define semantics for programs with aggregates. Here, we restrict our review of τ to conditional programs. The translation τ is summarized below using the following notation: if t is a term, $\mathbf{x}$ is a tuple of variables, and $\mathbf{r}$ is a tuple of terms of the same length as $\mathbf{x}$, then $[t]^{\mathbf{x}}_{\mathbf{r}}$ (equivalently, $t^{\mathbf{x}}_{\mathbf{r}}$) is the term obtained from t by substituting $\mathbf{x}$ by $\mathbf{r}$. We use similar notation for other expressions such as literals or their lists, e.g., we may write $[l_1, \dots, l_m]^{\mathbf{x}}_{\mathbf{r}}$ which stands for $[l_1]^{\mathbf{x}}_{\mathbf{r}}, \dots, [l_m]^{\mathbf{x}}_{\mathbf{r}}$ and $[H : \mathbf{L}]^{\mathbf{x}}_{\mathbf{r}}$ which stands for $H^{\mathbf{x}}_{\mathbf{r}} : \mathbf{L}^{\mathbf{x}}_{\mathbf{r}}$. A conditional literal or a rule is *closed* if it contains no global variables. $|\mathbf{x}|$ denotes the number of elements in a list $\mathbf{x}$.

To transform a closed rule R into an infinitary propositional formula w.r.t. a set $\mathcal{G}$ of ground terms, translation τ is defined as follows:

1. $\tau(\bot)$ is $\bot$;
2. $\tau(A)$ is A for a ground atom A;
3. $\tau(t_1 = t_2)$ is $\top$ if t_1 is identical to t_2, and $\bot$ otherwise, for ground terms t_1, t_2;
4. τ ($not\ A$) is $\neg\tau A$;
5. $\tau(\mathbf{L})$ is $\tau(l_1) \wedge \dots \wedge \tau(l_m)$ for a list $\mathbf{L}$ of basic literals;
6. for a closed conditional literal $H : \mathbf{L}$ occurring in the body of rule R, $\tau(H : \mathbf{L})$ is the conjunction of the formulas $\tau(\mathbf{L}^{\mathbf{x}}_{\mathbf{r}}) \to \tau(H^{\mathbf{x}}_{\mathbf{r}})$ where $\mathbf{x}$ is the list of variables occurring in the conditional literal, over all tuples of ground terms $\mathbf{r} \in \mathcal{G}^{|\mathbf{x}|}$ (recall that $\mathcal{G}^n$ denotes the Cartesian product $\mathcal{G} \times \dots \times \mathcal{G}$ of length n);
7. for a closed conditional literal $H : \mathbf{L}$ occurring in the head of rule R, $\tau(H : \mathbf{L})$ is the disjunction of formulas $\big(\tau(\mathbf{L}^{\mathbf{x}}_{\mathbf{r}}) \to \tau(H^{\mathbf{x}}_{\mathbf{r}})\big) \wedge \neg\neg\tau(\mathbf{L}^{\mathbf{x}}_{\mathbf{r}})$ where $\mathbf{x}$ is the list of variables occurring in the conditional literal, over all tuples of ground terms $\mathbf{r} \in \mathcal{G}^{|\mathbf{x}|}$;
8. for a closed rule r of form (2), $\tau(r)$ is $\tau B_1 \wedge \dots \wedge \tau B_n \to \tau H_1 \vee \dots \vee \tau H_m$.

Now, we formalize the rule instantiation process from [10]. Let $\mathbf{z}$ denote the global variables of rule R. By $inst_{\mathcal{G}}(R)$ we denote the *instantiations* of rule R w.r.t. a set $\mathcal{G}$ of ground terms, i.e., $inst_{\mathcal{G}}(R) = \{R^{\mathbf{z}}_{\mathbf{u}} \mid \mathbf{u} \in \mathcal{G}^{|\mathbf{z}|}\}^\wedge$. Clearly, every rule $r \in inst_{\mathcal{G}}(R)$ is closed, as is each (conditional) literal occurring in r.

Let Π be a conditional program. For a rule R in Π, its translation is defined as $\tau(R) = \{\tau(r) \mid r \in inst_{\mathcal{G}_\Pi}(R)\}^\wedge$. Similarly, $\tau(\Pi) = \{\tau(R) \text{ w.r.t. } \mathcal{G}_\Pi \mid R \in \Pi\}^\wedge$. A set of ground atoms constructed over the signature of Π forms a CLINGO *answer set* of a program Π when it is a stable model of $\tau(\Pi)$ in the sense of Definition 1 by Truszczyński (2012).

5 Connecting Two Semantics of Conditional Programs

In this section our goal is to connect our proposed semantics for conditional (logic) programs via the SM operator (Section 3) to the semantics defined for such programs in [10] (Section 4). For that purpose we review some of the details of [15] that help us to construct an argument for the formal relationship between the considered semantics for conditional programs.

Truszczynski (2012) provides a definition of stable models for first-order sentences. These models may be Herbrand and non-Herbrand interpretations. He defines *the grounding of a sentence F w.r.t. interpretation I*, denoted by $gr_I(F)$, as a transformation of F into infinitary propositional formulas over a given signature [15, Section 3]. If we restrict our attention to Herbrand interpretations, we may note that: *For arbitrary Herbrand interpretations I_1 and I_2 of a first-order sentence F, $gr_{I_1}(F)$ is identical to $gr_{I_2}(F)$*. Thus, we drop the subscript I from the definition of $gr_I(F)$ when we review this concept.

Let σ be a signature and let I be a Herbrand interpretation over σ. For a ground term c in $\mathcal{G}_\sigma$, we use c to denote both this ground term and its respective domain element in I, i.e., $c = c^I$. Let F be a first-order sentence over σ. The *grounding of F (w.r.t. σ)*, denoted by $gr(F)$, is defined recursively, mapping F into an infinitary propositional formula:

1. $gr(\bot) = \bot$;
2. $gr(A) = A$ for a ground atom A;
3. $gr(t_1 = t_2)$ is $\top$ if t_1 is identical to t_2, and $\bot$ otherwise, for ground terms t_1, t_2;
4. If $F = G \vee \ldots \vee H$, then $gr(F) = gr(G) \vee \cdots \vee gr(H)$;
5. If $F = G \wedge \ldots \wedge H$, then $gr(F) = gr(G) \wedge \cdots \wedge gr(H)$;
6. If $F = G \rightarrow H$, then $gr(F) = gr(G) \rightarrow gr(H)$;
7. If $F = \exists \mathbf{x} G(\mathbf{x})$, then $gr(F) = \{gr(G(\mathbf{u})) \mid \mathbf{u} \in \mathcal{G}_\sigma^{|\mathbf{x}|}\}^\vee$;
8. If $F = \forall \mathbf{x} G(\mathbf{x})$, then $gr(F) = \{gr(G(\mathbf{u})) \mid \mathbf{u} \in \mathcal{G}_\sigma^{|\mathbf{x}|}\}^\wedge$.

We now observe a key property relating gr, ϕ, and τ transformations that is essential in connecting the SM-based semantics proposed for conditional programs and the infinitary logic-based semantics reviewed in the previous section.

Theorem 1 (Syntactic Identity). *For any conditional logic program Π containing at least one object constant, $gr\big(\phi(\Pi)\big)$ is identical to $\tau(\Pi)$.*

By Theorem 5 from [15], it follows that for a first-order sentence F, Herbrand interpretations of F are answer sets of F if and only if they are stable models of $gr(F)$ in the sense of Definition 1 by Truszczyński (2012). The following

theorem is an immediate consequence of this formal result and the Theorem on Syntactic Identity.

Theorem 2 (Main Theorem). *For any conditional logic program Π containing at least one object constant and any Herbrand interpretation I over $(\mathcal{O}_\Pi, \mathcal{F}_\Pi, \mathcal{P}_\Pi)$, the following conditions are equivalent:*
- *I is an answer set of Π as defined in Section 3;*
- *I is a* CLINGO *answer set of Π as defined in Section 4.*

The remainder of this section presents auxiliary results required in constructing the proof of the Theorem on Syntactic Identity, followed by the theorem's proof. The following lemma captures basic equivalences between ϕ, function composition $gr \circ \phi$, and τ transformations.

Lemma 1. *Let $\mathbf{z}$ be a list of variables. Then, the following equivalences hold:*
1. $\phi_{\mathbf{z}}(\bot) = \tau(\bot)$;
2. $gr(\phi_{\mathbf{z}}\bot) = \tau(\bot)$;

Let A be an atom, $\mathbf{x}$ be a list that includes all variables in A, and $\mathbf{r}$ be a list of ground terms of the same length as $\mathbf{x}$. Then, the following equivalences hold:
3. $\phi_{\mathbf{z}}(A^{\mathbf{x}}_{\mathbf{r}}) = \tau(A^{\mathbf{x}}_{\mathbf{r}})$;
4. $\phi_{\mathbf{z}}(not\ A^{\mathbf{x}}_{\mathbf{r}}) = \tau(not\ A^{\mathbf{x}}_{\mathbf{r}})$.
5. $gr(\phi_{\mathbf{z}}(A^{\mathbf{x}}_{\mathbf{r}})) = \tau(A^{\mathbf{x}}_{\mathbf{r}})$;
6. $gr(\phi_{\mathbf{z}}(not\ A^{\mathbf{x}}_{\mathbf{r}})) = \tau(not\ A^{\mathbf{x}}_{\mathbf{r}})$.

Let A be an atomic formula of the form $t_1 = t_2$, $\mathbf{x}$ be a list that includes all variables in A, and $\mathbf{r}$ be a list of ground terms of the same length as $\mathbf{x}$. Then, the following equivalences hold:
7. $gr(\phi_{\mathbf{z}}(A^{\mathbf{x}}_{\mathbf{r}})) = \tau(A^{\mathbf{x}}_{\mathbf{r}})$;
8. $gr(\phi_{\mathbf{z}}(not\ A^{\mathbf{x}}_{\mathbf{r}})) = \tau(not\ A^{\mathbf{x}}_{\mathbf{r}})$.

Let $\mathbf{L}$ be a list of basic literals, $\mathbf{x}$ be a list that includes all variables in $\mathbf{L}$, and $\mathbf{r}$ be a list of ground terms of the same length as $\mathbf{x}$. The equivalence below holds:
9. $gr\big(\phi_{\mathbf{z}}(\mathbf{L}^{\mathbf{x}}_{\mathbf{r}})\big) = \tau(\mathbf{L}^{\mathbf{x}}_{\mathbf{r}})$.

Proof. Equivalences 1-5, 7 follow immediately from the definitions of ϕ, gr, and τ transformations (and preceding equivalences, e.g., proof of equivalence 2 takes into account equivalence 1). The claim of equivalences 6 and 8 relies on equivalences 5 and 7, respectively, and is supported by the following chain

$$gr(\phi_{\mathbf{z}}(not\ A^{\mathbf{x}}_{\mathbf{r}})) = gr(\neg\phi_{\mathbf{z}}(A^{\mathbf{x}}_{\mathbf{r}})) = gr(\phi_{\mathbf{z}}(A^{\mathbf{x}}_{\mathbf{r}}) \to \bot) =$$
$$gr(\phi_{\mathbf{z}}(A^{\mathbf{x}}_{\mathbf{r}})) \to gr(\bot) = \tau(A^{\mathbf{x}}_{\mathbf{r}}) \to \bot = \neg\tau(A^{\mathbf{x}}_{\mathbf{r}}) = \tau(not\ A^{\mathbf{x}}_{\mathbf{r}}).$$

Equivalence 9 follows immediately from the definitions of ϕ, gr, and τ transformations and equivalences 5-8. The remaining lemmas of this section capture less trivial equivalences between $gr \circ \phi$ and τ transformations.

Lemma 2. *Let Π be a conditional logic program containing at least one object constant. Let B be a literal in the body of a rule R in Π, where $\mathbf{z}$ is the list of global variables occurring in R and $\mathbf{u}$ is a list of ground terms of the same length as $\mathbf{z}$. Then, $gr\Big(\phi_{\mathbf{z}}([B]^{\mathbf{z}}_{\mathbf{u}})\Big)$ is identical to $\tau([B]^{\mathbf{z}}_{\mathbf{u}})$ w.r.t. $\mathcal{G}_\Pi$.*

Proof. For the case when B is a basic literal, the claim immediately follows from Lemma 1. What remains to be shown is that it is also the case when B is a conditional literal of the form $H : \mathbf{L}$. Take $\mathbf{x}$ to denote the set of local variables in B. Per condition 5 of the definition of $\phi_{\mathbf{z}}$, $\phi_{\mathbf{z}}(H : \mathbf{L}) = \forall \mathbf{x} \left(\phi_{\mathbf{z}}(\mathbf{L}) \rightarrow \phi_{\mathbf{z}}(H)\right)$.

$$
\begin{aligned}
gr\big(\phi_{\mathbf{z}}([H : \mathbf{L}]^{\mathbf{z}}_{\mathbf{u}})\big) &= gr\Big(\forall \mathbf{x}\big(\phi_{\mathbf{z}}(\mathbf{L}^{\mathbf{z}}_{\mathbf{u}}(\mathbf{x})) \rightarrow \phi_{\mathbf{z}}(H^{\mathbf{z}}_{\mathbf{u}}(\mathbf{x}))\big)\Big) \\
&= \{gr(\phi_{\mathbf{z}}(\mathbf{L}^{\mathbf{zx}}_{\mathbf{uv}}) \rightarrow \phi_{\mathbf{z}}(H^{\mathbf{zx}}_{\mathbf{uv}})) \mid \mathbf{v} \in \mathcal{G}_{\Pi}^{|\mathbf{x}|}\}^{\wedge} \\
&= \{gr(\phi_{\mathbf{z}}(\mathbf{L}^{\mathbf{zx}}_{\mathbf{uv}})) \rightarrow gr(\phi_{\mathbf{z}}(H^{\mathbf{zx}}_{\mathbf{uv}})) \mid \mathbf{v} \in \mathcal{G}_{\Pi}^{|\mathbf{x}|}\}^{\wedge} \\
&= \{\tau(\mathbf{L}^{\mathbf{zx}}_{\mathbf{uv}}) \rightarrow \tau(H^{\mathbf{zx}}_{\mathbf{uv}}) \mid \mathbf{v} \in \mathcal{G}_{\Pi}^{|\mathbf{x}|}\}^{\wedge} \\
&= \tau([H : \mathbf{L}]^{\mathbf{z}}_{\mathbf{u}})
\end{aligned}
$$

Condition 8 of the definition of the gr transformation allows us to move from the first line to the second in the chain above. Condition 6 of that definition allows us to move from the second line to the third. Lemma 1 provides us with grounds to move from the third to the fourth line. The final step is due to condition 6 of the τ transformation definition.

Lemma 3. *Let Π be a conditional logic program containing at least one object constant. Let H be a literal in the head of a rule R in Π, where $\mathbf{z}$ is the list of global variables occurring in R and $\mathbf{u}$ is a list of ground terms of the same length as $\mathbf{z}$. Then, $gr\Big(\phi_{\mathbf{z}}([H]^{\mathbf{z}}_{\mathbf{u}})\Big)$ is identical to $\tau([H]^{\mathbf{z}}_{\mathbf{u}})$ w.r.t. $\mathcal{G}_{\Pi}$.*
The proof of Lemma 3 is similar in structure to the proof of Lemma 2.

Lemma 4. *Let Π be a conditional logic program containing at least one object constant. For any rule R in Π, $gr\big(\phi(R)\big)$ is identical to $\tau(R)$ w.r.t. $\mathcal{G}_{\Pi}$.*

Proof. Let $\mathbf{z}$ be the list of the global variables of R.

$$
\begin{aligned}
gr\big(\phi(R)\big) &= gr\Big(\forall \mathbf{z}\big(\phi_{\mathbf{z}}(B_1) \wedge \cdots \wedge \phi_{\mathbf{z}}(B_n) \rightarrow \phi_{\mathbf{z}}(H_1) \vee \cdots \vee \phi_{\mathbf{z}}(H_m)\big)\Big) \\
&= \{gr\Big(\phi_{\mathbf{z}}([B_1]^{\mathbf{z}}_{\mathbf{u}}) \wedge \cdots \wedge \phi_{\mathbf{z}}([B_n]^{\mathbf{z}}_{\mathbf{u}}) \rightarrow \phi_{\mathbf{z}}([H_1]^{\mathbf{z}}_{\mathbf{u}}) \vee \cdots \vee \phi_{\mathbf{z}}([H_m]^{\mathbf{z}}_{\mathbf{u}})\Big) \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mid \mathbf{u} \in \mathcal{G}_{\Pi}^{|\mathbf{z}|}\}^{\wedge} \\
&= \{gr\Big(\phi_{\mathbf{z}}([B_1]^{\mathbf{z}}_{\mathbf{u}})\Big) \wedge \cdots \wedge gr\Big(\phi_{\mathbf{z}}([B_n]^{\mathbf{z}}_{\mathbf{u}})\Big) \\
&\qquad\qquad \rightarrow gr\Big(\phi_{\mathbf{z}}([H_1]^{\mathbf{z}}_{\mathbf{u}})\Big) \vee \cdots \vee gr\Big(\phi_{\mathbf{z}}([H_m]^{\mathbf{z}}_{\mathbf{u}})\Big) \mid \mathbf{u} \in \mathcal{G}_{\Pi}^{|\mathbf{z}|}\}^{\wedge} \\
&= \{\tau([B_1]^{\mathbf{z}}_{\mathbf{u}}) \wedge \cdots \wedge \tau([B_n]^{\mathbf{z}}_{\mathbf{u}}) \rightarrow \tau([H_1]^{\mathbf{z}}_{\mathbf{u}}) \vee \cdots \vee \tau([H_m]^{\mathbf{z}}_{\mathbf{u}}) \mid \mathbf{u} \in \mathcal{G}_{\Pi}^{|\mathbf{z}|}\}^{\wedge} \\
&= \{\tau(R^{\mathbf{z}}_{\mathbf{u}}) \mid \mathbf{u} \in \mathcal{G}_{\Pi}^{|\mathbf{z}|}\}^{\wedge} = \{\tau(r) \mid r \in inst_{\mathcal{G}_{\Pi}}(R)\}^{\wedge} \\
&= \tau(R)
\end{aligned}
$$

Lemmas 2 and 3 provide grounds for the fourth equality in the chain. The remainder follows from the definitions of gr, ϕ and τ transformations. The case when the head of the rule is $\bot$ follows the same lines.

The following equality follows immediately from Lemma 4 and constitutes a proof of the Theorem on Syntactic Identity:

$$gr\big(\phi(\Pi)\big) = \{gr(\phi(R)) \mid R \in \Pi\}^{\wedge} = \{\tau(R) \mid R \in \Pi\}^{\wedge} = \tau(\Pi).$$

6 Arguing Correctness of the k-coloring problem

In this section, we apply the *verification methodology for logic programs* proposed in [2] to the *k-coloring* encoding containing conditional literals. This methodology consists of four steps:

1. Decompose the informal description of the problem into independent (natural language) statements.
2. Fix the vocabulary/predicate constants used to represent the problem and its solutions.
3. Formalize the specification of the statements as a logic (modular) program.
4. Construct a "metaproof" in natural language for the correspondence between the constructed program and the informal description of the problem.

An instance of the k-coloring problem is a triple $\langle V, E, C\rangle$, where

– $\langle V, E\rangle$ is a graph with vertices V and edges $E \subseteq V \times V$, and
– C is a set of labels named *colors*, whose cardinality is k.

A solution to the *k-coloring* problem is

K1 a function $\widetilde{asg} : V \longrightarrow C$ such that
K2 every edge $(a, b) \in E$ satisfies condition $\widetilde{asg}(a) \neq \widetilde{asg}(b)$.

We view statements **K1** and **K2** as the formalization of Step 1. In fact, this formalization of Step 1 follows the lines by Fandinno, Hansen, and Lierler (2022), who considered another encoding of *k-coloring* problem (containing aggregates) and argued its correctness.

We fix predicate constants $vtx/1$, $edge/2$, $color/1$, $asg/2$ to represent the *k-coloring* problem and its solutions. In particular, predicate constants $vtx/1$, $edge/2$, and $color/1$ are used to encode a specific instance of the problem; whereas predicate constant $asg/2$ is used to encode the function $\widetilde{asg}$. Formally, we call a binary relation $\mathbf{r}$ *functional* when for all pairs (a_1, b_1), (a_2, b_2) in $\mathbf{r}$, if $a_1 = a_2$, then $b_1 = b_2$. Clearly, functional relations can be used to *encode* functions in an intuitive manner, where each pair (a, b) in functional relation $\mathbf{r}$ suggests a mapping from element a to element b. In other words, $asg/2$ will encode a functional relation meant to capture mapping $\widetilde{asg}$ that forms a solution to the considered instance of the *k-coloring* problem. This constitutes Step 2.

Splitting Theorem [8] is a fundamental result that allows us to uncover the internal structure of a logic program. For example, consider the context of the *k-coloring* program. Using the ϕ-transformation, we identify this program with sentence KC that is the conjunction of formulas (3-6). By K_1 we denote the conjunction of formulas (3-5) and by K_2 we denote formula (6). The Splitting Theorem tells us that

$$\mathrm{SM}_{asg}[KC] \equiv \mathrm{SM}_{asg}[K_1] \wedge K_2. \tag{7}$$

Within this verification methodology, Step 3 can be implemented by considering $\mathrm{SM}_{asg}[K_1]$ and K_2 as two modules of a logic (modular) program that formalize statements **K1** and **K2**, respectively. We now make this claim precise by stating formal results that culminate in capturing Step 4. Intuitively, the following Lemmas 5 and 6 state that modules $\mathrm{SM}_{asg}[K_1]$ and K_2 formalize statements **K1** and **K2**, respectively.

Lemma 5. *Let I be an Herbrand interpretation such that $\langle vtx^I, edge^I, color^I\rangle$ forms an instance of the k-coloring problem. Then, $I \models \mathrm{SM}_{asg}[K_1]$ if and only if relation asg^I encodes a function from vtx^I to $color^I$.*

Lemma 6. *Let I be an Herbrand interpretation such that $\langle vtx^I, edge^I, color^I\rangle$ forms an instance of the k-coloring problem and asg^I encodes a function $\widetilde{asg}$ from vtx^I to $color^I$. Then, $I \models K_2$ if and only if every edge $(a, b) \in edge^I$ satisfies condition $\widetilde{asg}(a) \neq \widetilde{asg}(b)$.*

By equivalence (7) and the two preceding lemmas the following theorem immediately follows. This theorem can be seen as a proof of correctness for the *k-coloring* encoding.

Theorem 3. *Let I be an Herbrand interpretation such that $\langle vtx^I, edge^I, color^I\rangle$ forms an instance of the k-coloring problem. Then, $I \models \mathrm{SM}_{asg}[KC]$ if and only if $(asg/2)^I$ encodes a function that forms a solution to the considered instance.*

Due to space constraints, we omit the proofs of the formal results of this section. We refer the reader to similar arguments, namely, the proofs by Cabalar, Fandinno, and Lierler (2020) when they argue correctness of logic program modules for the Hamiltonian Cycle problem, and the proofs by Fandinno, Hansen, and Lierler (2022) when they do the same for the Traveling Salesman problem and an alternative encoding of the *k-coloring* problem.

We note that in practical settings answer set systems accept instances of problems, typically encoded as sets of facts. For instance, the set Π_G of facts

$$\texttt{vtx(a). vtx(b). edge(a,b). color(g). color(b). color(r).} \tag{8}$$

corresponds to the following instance of the 3-coloring problem:

$$\langle \{a, b\}, \{(a, b)\}, \{g, b, r\}\rangle. \tag{9}$$

Consider module $\mathrm{SM}_{vtx,edge,color}[\phi(\Pi_G)]$. It captures the considered set of facts. For a Herbrand model I of this module, the extension $\langle vtx^I, edge^I, color^I\rangle$ forms (9). This claim trivially follows from Theorem on Completion [8]. Note that the program composed of the facts in Π_G and the *k-coloring* rules has answer sets that are the Herbrand models of formula $\mathrm{SM}_{vtx,edge,color,asg}[\phi(\Pi_G) \wedge KC]$. By the Splitting Theorem, it is equivalent to $\mathrm{SM}_{vtx,edge,color}[\phi(\Pi_G)] \wedge \mathrm{SM}_{asg}[KC]$. By Theorem 3 we may immediately conclude that any answer set of a program composed of the facts in Π_G and rules in the *k-coloring* program is such that the extension of predicate constant $asg/2$ encodes a solution to instance (9) of the 3-coloring problem.

The previous paragraph illustrates an important idea stemming from [3]. Given a problem P and a signature σ_P to encode it, in place of discussing the specifics of encoding of a particular instance of problem P, we may associate Herbrand interpretations over σ_P satisfying some conditions with this instance. This way we may speak of an encoding for problem P in separation from details on how an instance for this problem is encoded. The statement of the final theorem illustrates this idea. For example, the following rules `vtx(X):-edge(X,Y).` `vtx(Y):-edge(X,Y).` can be used to replace the first two facts in (8) to encode the same instance of the *k-coloring* problem. As long as we may associate this new encoding of an instance with an Herbrand interpretation capturing that instance from the perspective of the claim of Theorem 3, we can claim the correctness of the resulting ASP program composed of a newly encoded instance and the *k-coloring* encoding. This is true whenever the Splitting Theorem supports modularization of a program as illustrated.

Conclusions, Future Work, Acknowledgements

In this paper we present semantics for conditional programs that do not refer to grounding. These semantics demonstrate that conditional literals represent nested implications within rules. The benefits of this contribution are threefold. First, it has pedagogical value. The nested implication approach provides a simple characterization of conditional literals, supplying students with an intuitive perspective on their behavior. Our work provides rigorous support for this previously informal intuition. Second, conditional literals are a step towards developing ASP rules with complex, nested bodies that are closer to classical logic languages. This makes the language more expressive. Finally, we have broadened the class of ASP programs that can be formally verified without referring to grounding. For instance, the final section of this paper illustrates the use of the proposed semantics by arguing the correctness of the *k-coloring* encoding. In the Introduction we mentioned how conditional literals are omnipresent in meta-programming. The users of meta-programming may now apply similar ideas in constructing proofs of correctness for their formalizations.

We also note that for so called tight conditional programs [8], our characterization provides a way to associate such programs with classical first-order logic formulas by means of the Theorem on Completion [5, Section A.3]. This fact forms a theoretical foundation for a possible extension of the software verification tool ANTHEM [6] to programs with conditional literals. This tool allows its users to formally and automatically verify the correctness of tight logic programs (without conditional literals). The *k-coloring* program presented in this paper is tight and the suggested extension of ANTHEM would therefore be applicable to it. Implementing the corresponding extension in ANTHEM is a direction of future work.

Acknowledgements The work was partially supported by NSF grant 1707371. We are grateful to Jorge Fandinno, Vladimir Lifschitz, and Miroslaw Truszczynski for valuable discussions and comments on this paper.

References

1. Brewka, G., Eiter, T., Truszczyński, M.: Answer set programming at a glance. Communications of the ACM **54**(12), 92–103 (Dec 2011). https://doi.org/10.1145/2043174.2043195
2. Cabalar, P., Fandinno, J., Lierler, Y.: Modular answer set programming as a formal specification language. Theory and Practice of Logic Programming **20**(5), 767–782 (2020)
3. Fandinno, J., Hansen, Z., Lierler, Y.: Arguing correctness of asp programs with aggregates (2022), accepted to LPNMR-22
4. Fandinno, J., Hansen, Z., Lierler, Y.: Axiomatization of aggregates in answer set programming. In: Proceedings of the Thirty-six National Conference on Artificial Intelligence (AAAI'22). AAAI Press (2022)
5. Fandinno, J., Lifschitz, V.: Verification of locally tight programs (2022), http://www.cs.utexas.edu/users/ai-labpub-view.php?PubID=127938
6. Fandinno, J., Lifschitz, V., Lühne, P., Schaub, T.: Verifying tight logic programs with anthem and vampire. Theory and Practice of Logic Programming **20**(5), 735–750 (2020)
7. Ferraris, P., Lee, J., Lifschitz, V.: Stable models and circumscription. Artificial Intelligence **175**(1), 236–263 (2011)
8. Ferraris, P., Lee, J., Lifschitz, V., Palla, R.: Symmetric splitting in the general theory of stable models. In: Boutilier, C. (ed.) Proceedings of the Twenty-first International Joint Conference on Artificial Intelligence (IJCAI'09). pp. 797–803. AAAI/MIT Press (2009)
9. Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., Thiele, S.: A user's guide to `gringo`, `clasp`, `clingo`, and `iclingo`, http://potassco.org
10. Harrison, A., Lifschitz, V., Yang, F.: The semantics of gringo and infinitary propositional formulas. In: Baral, C., De Giacomo, G., Eiter, T. (eds.) Proceedings of the Fourteenth International Conference on Principles of Knowledge Representation and Reasoning (KR'14). AAAI Press (2014)
11. Kaminski, R., Romero, J., Schaub, T., Wanko, P.: How to build your own ASP-based system?! Theory and Practice of Logic Programming p. 1–63 (2021). https://doi.org/10.1017/S1471068421000508
12. Lifschitz, V.: Thirteen definitions of a stable model. In: Fields of Logic and Computation: Essays Dedicated to Yuri Gurevich on the Occasion of his 70th Birthday (2010)
13. Marek, V.W., Truszczyński, M.: Stable Models and an Alternative Logic Programming Paradigm, pp. 375–398. Springer Berlin Heidelberg, Berlin, Heidelberg (1999)
14. Niemelä, I.: Logic programs with stable model semantics as a constraint programming paradigm. Annals of Mathematics and Artificial Intelligence **25**(3), 241–273 (Nov 1999). https://doi.org/10.1023/A:1018930122475
15. Truszczyński, M.: Connecting first-order ASP and the logic FO(ID) through reducts. In: Erdem, E., Lee, J., Lierler, Y., Pearce, D. (eds.) Correct Reasoning: Essays on Logic-Based AI in Honour of Vladimir Lifschitz, Lecture Notes in Computer Science, vol. 7265, pp. 543–559. Springer-Verlag (2012)