FO(C) and Related Modelling Paradigms

Bogaerts Bart[1] and **Vennekens Joost**[1] and **Denecker Marc**[1] and **Van den Bussche Jan**[2]

Abstract. Recently, C-LOG was introduced as a language for modelling causal processes. Its formal semantics has been defined together with introductory examples, but the study of this language is far from finished. In this paper, we compare C-LOG to other declarative modelling languages. More specifically, we compare to first-order logic (FO), and argue that C-LOG and FO are orthogonal and that their integration, FO(C), is a knowledge representation language that allows for clear and succinct models. We compare FO(C) to E-disjunctive logic programming with the stable semantics, and define a fragment on which both semantics coincide. Furthermore, we discuss object-creation in FO(C), relating it to mathematics, business rules systems, and data base systems.

1 Introduction

Previous work introduced C-LOG [3], an expressive language construct to describe causal processes, and FO(C), its integration with classical logic. In that work, it is indicated that C-LOG shows similarities to many other languages and it is suggested that C-LOG could serve as a tool to study the semantical relationship between these languages. In this paper, we take the first steps for such a study: we discuss the relationship of FO(C) with other paradigms and through this discussion, provide a comprehensive overview of the informal semantics of FO(C).

C-LOG and FO are syntactically very similar, but semantically very different languages. In this paper we formalise the semantical relationship between C-LOG and FO, and argue how their integration, FO(C), is a rich language in which knowledge can be represented succinctly and clearly.

We explain how modelling in FO(C) relates to the “generate, define, and test” methodology used in answer set programming. We discuss how FO(C) relates to disjunctive logic programs with existential quantification in rule heads [16], both informally and formally, and we identify a subset of E-disjunctive logic programs on which stable semantics corresponds to the FO(C) semantics. We also discuss four important knowledge representation constructs that FO(C) adds with respect to E-disjunctive logic programs: *nested rules* (in fact, arbitrary nesting of expressions), *dynamic choice*, *object creation*, and *a more modular semantics*.

Furthermore, we discuss object-creation in related paradigms. One of those discussed paradigms is the field of deductive databases, where extensions of Datalog have been defined. In [2], rules with existentially quantified head variables are used for object creation. It is remarkable to see how the same extension of logic programs is used sometimes (e.g., in [16]) for selection, and sometimes (e.g., in [2]) for object-creation. Consider for example a rule

$$\forall X : \exists Y : P(X,Y) \text{:-}\, q(X).$$

Viewing this rule as a rule in an E-disjunctive logic program, it corresponds to the C-LOG expression

$$\mathbf{All}\, X[q(X)] : \mathbf{Select}\, Y[\mathbf{t}] : P(X,Y)$$

while in case this same rule occurs in a LogicBlox [7] specification, it corresponds to the C-LOG expression

$$\mathbf{All}\, X[q(X)] : \mathbf{New}\, Y : P(X,Y).$$

The explicit distinction C-LOG makes between object-creation and selection is necessary for studying the relationship between these languages.

The rest of this paper is structured as follows. In Section 2 we give preliminaries, including the syntax and informal semantics of C-LOG. In Sections 3 and 4, we focus on the creation-free fragment of C-LOG, i.e., on expressions without the **New**-operator: first, we compare C-LOG to FO and discuss the integration of these two; afterwards, we compare C-LOG to E-disjunctive logic programs. In Section 5, we discuss object-creation in C-LOG by providing simple intuitive examples and relating the **New**-operator to other languages with similar forms of object-creation. We conclude in Section 6.

2 C-LOG

We assume familiarity with the basics of first-order logic. Vocabularies, formulas, and terms are defined as usual. We use **t** for truth and **f** for falsity. $\sigma^{\mathcal{I}}$ denotes the interpretation of symbol σ in structure $\mathcal{I}$. *Domain atoms* are atoms of the form $P(\overline{d})$ where the d_i are domain elements. We use restricted quantifications [13], e.g., in FO, these are formulas of the form $\forall x[\psi] : \varphi$ or $\exists x[\psi] : \varphi$, meaning that φ holds for all (resp. for a) x such that ψ holds. The above expressions are syntactic sugar for $\forall x : \psi \Rightarrow \varphi$ and $\exists x : \psi \land \varphi$, but such a reduction is not possible for other restricted quantifiers in C-LOG. We call ψ the *qualification* and φ the *assertion* of the restricted quantifications. From now on, let Σ be a relational vocabulary, i.e., Σ consists only of predicate, constant and variable symbols.

In what follows we briefly repeat the syntax and informal semantics of C-LOG. For more details and an extensive overview of the formal semantics of C-LOG, we refer to [3].

2.1 Syntax of C-LOG

Definition 2.1. Causal effect expressions *(CEE) are defined inductively as follows:*

- *if $P(\overline{t})$ is an atom, then $P(\overline{t})$ is a CEE,*

[1] Department of Computer Science, KU Leuven, Belgium email: firstname.name@cs.kuleuven.be

[2] Hasselt University & transnational University of Limburg, Belgium email: jan.vandenbussche@uhasselt.be

- *if φ is an FO formula and C' is a CEE, then $C' \leftarrow \varphi$ is a CEE,*
- *if C_1 and C_2 are CEEs, then C_1 **And** C_2 is a CEE,*
- *if C_1 and C_2 are CEEs, then C_1 **Or** C_2 is a CEE,*
- *if x is a variable, φ is a first-order formula and C' is a CEE, then* $\mathbf{All}\, x[\varphi] : C'$ *is a CEE,*
- *if x is a variable, φ is a first-order formula and C' is a CEE, then* $\mathbf{Select}\, x[\varphi] : C'$ *is a CEE,*
- *if x is a variable and C' is a CEE, then* $\mathbf{New}\, x : C'$ *is a CEE.*

We call a CEE an *atom-expression* (respectively *rule-*, **And**-, **Or**-, **All**-, **Select**- or **New**-*expression*) if it is of the corresponding form. We use $\mathbf{All}\, \overline{x}[\varphi] : C$ as an abbreviation for $\mathbf{All}\, x_1[\mathbf{t}] : \ldots \mathbf{All}\, x_n[\varphi] : C$ and similar for **Select**-expressions. We call a predicate symbol P *endogenous* in C if P occurs as the symbol of a (possibly nested) atom-expression in C, i.e., if P occurs in C but not only in first-order formulas. All other symbols are called *exogenous* in C. An occurrence of a variable x is *bound* in a CEE if it occurs in the scope of a quantification over that variable ($\forall x$, $\exists x$, $\mathbf{All}\, x$, $\mathbf{Select}\, x$, or $\mathbf{New}\, x$) and *free* otherwise. A variable is *free* in a CEE if it has free occurrences. A *causal theory*, or C-LOG *theory* is a CEE without free variables. We often represent a causal theory as a set of CEEs; the intended causal theory is the **And**-conjunction of these CEEs.

2.2 Informal Semantics of C-LOG

In this section, we discuss the informal semantics of CEEs. We repeat the driving principles on a simple example—one without non-determinism—and discuss more complex expressions afterwards.

Driving Principles

Following the philosophy of [15], the semantics of C-LOG is based on two principles that are common in causal modelling. The first is the distinction between *endogenous* and *exogenous* properties, i.e., those whose value is determined by the causal laws in the model and those whose value is not, respectively [12]. The second is the *default-deviant* assumption, used also by, e.g., [8, 10]. The idea here is to assume that each endogenous property of the domain has some "natural" state, that it will be in whenever nothing is acting upon it. For ease of notation, C-LOG identifies the default state with falsity, and the deviant state with truth. For example, consider the following simplified model of a bicycle, in which a pair of gear wheels can be put in motion by pedalling:

$$Turn(BigGear) \leftarrow Pedal. \tag{1}$$
$$Turn(BigGear) \leftarrow Turn(SmallGear). \tag{2}$$
$$Turn(SmallGear) \leftarrow Turn(BigGear). \tag{3}$$

Here, $Pedal$ is exogenous, while $Turn(BigGear)$ and $Turn(SmallGear)$ are endogenous. The semantics of this causal model is given by a straightforward "execution" of the rules. The domain starts out in an initial state, in which all endogenous atoms have their default value *false* and the exogenous atom $Pedal$ has some fixed value. If $Pedal$ is true, then the first rule is applicable and may be fired ("$Pedal$ causes $Turn(BigGear)$") to produce a new state of the domain in which $Turn(BigGear)$ now has its deviant value *true*. In this way, we construct the following sequence of states (we abbreviate symbols by their first letter):

$$\{P\} \rightarrow \{P, T(B)\} \rightarrow \{P, T(B), T(S)\} \tag{4}$$

In general, given a causal theory Δ, a causal process is a sequence of intermediate states, starting from the default state such that, at each state, the effects described by Δ take place. This notion of causal process is based on the following principles:

- The principle of *sufficient causation* states that if the precondition to a causal law is satisfied, then the event that it triggers must eventually happen. For example, the process described in (4) cannot stop after the first step: there is a cause for $Turn(SmallGear)$, hence this should eventually happen.
- The principle of *universal causation* states that all changes to the state of the domain must be triggered by a causal law whose precondition is satisfied. For example, the small gear can only turn if the big gear turns.
- The principle of *no self-causation* states that nothing can happen based on itself. E.g., if rule (1) would be excluded from the causal theory, the gears cannot start rotating by themselves.

Complex Expressions

A structure is a model of a causal theory Δ if it is the final states of a causal processes described by Δ. In order to define these processes correctly, one should know the events that take place in every state. We call the set of those events the *effect set* of the causal theory. There are two kinds of effects that can be described by a causal theory: 1) flipping an atom from its default to its deviant state and 2) creating a new domain element. We now explain in a compositional way what the effect set of a causal theory is in a given state of affairs, which we represent as usual by a structure.

The effect of an atom-expressions A is that A is flipped to its deviant state. A conditional effect, i.e., a rule expression, causes the effect set of its head if its body is satisfied in the current state, and nothing otherwise. The effect set described by an **And**-expression is the union of the effect sets of its two subexpressions; an **All**-expression $\mathbf{All}\, x[\varphi] : C'$ causes the union of all effect sets of $C'(x)$ for those x's that satisfy φ. An expression C_1 **Or** C_2 non-deterministically causes either the effect set of C_1 or the effect set of C_2; a **Select**-expression $\mathbf{Select}\, x[\varphi] : C'$ causes the effect set of C' for a non-deterministically chosen x that satisfies φ. An object-creating CEE $\mathbf{New}\, x : C'$ causes the creation of a new domain element n and the effect set of $C'(n)$.

Example 2.2. American citizenship can be obtained in several ways. One way is passing the naturalisation test. Another way is by playing the "Green Card Lottery", where each year a number of lucky winners are randomly selected and granted American citizenship. We model this as follows:

$$\mathbf{All}\, p[Apply(p) \wedge PassedTest(p)] : American(p)$$
$$(\mathbf{Select}\, p[Play(p)] : American(p)) \leftarrow Lottery.$$

The first CEE describes the "normal" way to become an American citizen; the second rule expresses that one winner is selected among everyone who plays the lottery. If $\mathcal{I}$ is a structure in which $Lottery$ holds, due to the non-determinism, there are many possible effect sets of the above CEE, namely the sets $\{American(p) \mid p \in Apply^{\mathcal{I}} \wedge p \in PassedTest^{\mathcal{I}}\} \cup \{American(d)\}$ for some $d \in Play^{\mathcal{I}}$.

Models of this causal theory are structures such that everyone who applies and passes the test is American, in case the lottery happens, one random person who played the lottery is American as well, and such that no-one else is American. The principle of sufficient causation guarantees a form of closed world assumption: you can only become American if there is a rule that causes you to obtain this nationality. The two CEEs are considered independent: the winner could be one of the people that obtained it through standard application, as well as someone else.

Note that in the above, there is a great asymmetry between $Play(p)$, which occurs as a qualification of **Select**-expression, and $American(p)$, which occurs as a caused atom. This means that the effect will never cause atoms of the form $Play(p)$, but only atoms of the form $American(p)$. This is one of the cases where the qualification of an expression cannot simply be eliminated.

Example 2.3. Hitting the "send" button in your mail application causes the creation of a new package containing a specific mail. That package is put on a channel and will be received some (unknown) time later. As long as the package is not received, it stays on the channel. In C-Log, we model this as follows:

$$\begin{aligned}
&\mathbf{All}\, m,t[Mail(m) \wedge HitSend(m,t)] : \mathbf{New}\, p :\\
&\quad Pack(p)\, \mathbf{And}\, Cont(p,m)\, \mathbf{And}\, OnCh(p,t+1)\, \mathbf{And}\\
&\quad \mathbf{Select}\, d[d>0] : Received(p,t+d)\\
&\mathbf{All}\, p,t[Pack(p) \wedge OnCh(p,t) \wedge \neg Received(p,t)] :\\
&\quad OnCh(p,t+1)
\end{aligned}$$

Suppose an interpretation $HitSend^{\mathcal{I}} = \{(MyMail, 0)\}$ is given. A causal process then unfolds as follows: it starts in the initial state, where all endogenous predicates are false. The effect set of the above causal effect in that state consists of 1) the creation of one new domain element, say $\_p$, and 2) the caused atoms $Pack(\_p)$, $Cont(\_p, MyMail)$, $OnCh(\_p, 1)$ and $Received(\_p, 7)$, where instead of 7, we could have chosen any number greater than zero. Next, it continues, and in every step t, before receiving the package, an extra atom $OnCh(p, t+1)$ is caused. Finally, in the seventh step, no more atoms are caused; the causal process ends. The final state is a model of the causal theory.

2.3 FO(C)

First-order logic and C-Log have a straightforward integration, FO(C). Theories in this logic are sets of FO sentences and causal theories. A model of such a theory is a structure that satisfies each of its expressions (each of its CEEs and sentences). An illustration is the mail protocol from Example 2.3, which we can extend with the "observation" that at at some time, two packages are on the channel:

$$\exists t, p_1, p_2[p_1 \neq p_2] : OnCh(p_1,t) \wedge OnCh(p_2,t).$$

Models of this theory represent states of affairs where at least once two packages are on the channel simultaneously. This entirely differs from **And**-conjoining our CEE with

$$\mathbf{Select}\, t, p_1, p_2[p_1 \neq p_2] : OnCh(p_1,t)\, \mathbf{And}\, OnCh(p_2,t).$$

The resulting CEE would have unintended models in which two packages suddenly appear on the channel for no reason.

3 C-Log, FO, and FO(C)

There is an obvious syntactical correspondence between FO and creation-free C-Log (C-Log without **New**-expressions): **And** corresponds to $\wedge$, **Or** to $\vee$, $\leftarrow$ to $\Leftarrow$, **All** to $\forall$, and **Select** to $\exists$. As already mentioned above, expressions in C-Log have an entirely different meaning than the corresponding FO expression. A C-Log expression describes a process in which more and more facts are caused, while an FO expression describes a truth. For example

$$P\, \mathbf{Or}\, Q$$

describes a process that picks either P or Q and makes one of them true, hence its models are structures in which exactly one of the two holds. On the other hand, the FO sentence

$$P \vee Q$$

has more models, namely also one in which both hold. We generalise this observation:

Theorem 3.1. *Let Δ be a creation-free causal theory over Σ and $\mathcal{T}_\Delta$ the corresponding FO theory (the theory obtained from Δ by replacing* **All** *by $\forall$,* **Select** *by $\exists$,* **Or** *by $\vee$,* **And** *by $\wedge$, and $\leftarrow$ by $\Leftarrow$). Then for every Σ-structure $\mathcal{I}$, if $\mathcal{I} \models \Delta$, then also $\mathcal{I} \models \mathcal{T}_\Delta$.*

The reverse does not always hold: there is no obvious way to translate an FO formula to a C-Log expression, since C-Log has no negation. In some cases, it is possible to find an inverse transformation, for example for positive (negation-free) FO theories. This would yield a constructive way to create models for a positive FO theory, which is not a surprising, nor a very interesting result; another constructive way to get a model of such a theory would be to make everything true. But it is interesting to view C-Log theories as a constructive way to create a certain structure. This shows that modelling in C-Log is orthogonal to modelling in FO. In FO, by default everything is open, every atom can be true or false arbitrarily. Every constraint removes worlds from the set of possible worlds. In C-Log on the other hand, all endogenous symbols are by default false. Adding extra rules to a C-Log theory can result in more models (when introducing extra non-determinism), or modify worlds. In some cases, one of the approaches is more natural than the other.

Consider for example a steel oven scheduling problem. For every block of steel, we should find a time t to put that block in the oven and at time $t+D$, where D is some fixed delay, we take the block out. In C-Log this is modelled as

$$\mathbf{All}\, b[Block(b)] : \mathbf{Select}\, t[\mathbf{t}] : In(b,t)\, \mathbf{And}\, Out(b, t+D),$$

but to model this in FO we would get one similar constraint together with several constraints guaranteeing uniqueness:

$$\begin{aligned}
&\forall b[Block(b)] : \exists t : In(b,t) \wedge Out(b,t+D)\\
&\forall b,t,t'[Block(b)] : In(b,t) \wedge In(b,t') \Rightarrow t = t'\\
&\forall b,t,t'[Block(b)] : Out(b,t) \wedge Out(b,t') \Rightarrow t = t'\\
&\forall x : (\exists t : In(x,t) \vee Out(x,t)) \Rightarrow Block(x)
\end{aligned}$$

Here, the approach in C-Log is much more natural, as in this example it is clear how to construct a model, whereas to model it in FO, we should analyse all properties of models. On the other hand, if we extend this example with a constraint that no two blocks can enter the oven at the same time, this is easily expressible in FO:

$$\neg\exists t, b, b'[b \neq b'] : In(b,t) \wedge In(b',t),$$

while this is not naturally expressible in C-Log. This shows the power of FO(C), the integration of FO and C-Log. For example, the entire above scheduling problem would be modelled in FO(C) as follows (where we use "{" and "}" to separate the C-Log theory from the FO sentences).

$$\begin{aligned}
&\Big\{\ \mathbf{All}\, b[Block(b)] : \mathbf{Select}\, t[\mathbf{t}] : In(b,t)\, \mathbf{And}\, Out(b,t+D)\ \Big\}\\
&\neg\exists t, b, b'[b \neq b'] : In(b,t) \wedge In(b',t)
\end{aligned}$$

This is much more readable and much more concise than any pure C-LOG or FO expression that expresses the same knowledge. As can be seen, the integration of the orthogonal languages FO and C-LOG, FO(C) provides a great modelling flexibility.

4 FO(C) and ASP

The methodology from the previous section is very similar to the "generate, define, and test" (GDT) methodology used in Answer Set Programming (ASP). In that methodology, "generate" and "define" are constructive modules of ASP programs that describe which atoms can be true, while the "test" module corresponds to first-order sentences that constrain solutions. In [6], it has been argued that GDT programs correspond to FO(ID) theories. Furthermore, in [3], we showed that FO(ID) is syntactically and semantically a sublanguage of FO(C). Here, we argue that a more general class of ASP programs can be seen as FO(C) theories.

E-disjunctive programs [16] are finite sets of rules of the form:

$$\forall \overline{x} : \exists \overline{y} : \alpha_1; \dots ; \alpha_m \text{ :- } \beta_1, \dots, \beta_k, \text{not}\, \gamma_1, \dots, \text{not}\, \gamma_n. \quad (5)$$

where the α_i, β_i and γ_i are atoms and variables in $\overline{y}$ only occur in the α_i. Given a structure $\mathcal{M}$, we define $\mathcal{M}^-$ as the literal set

$$\{\neg\alpha \mid \alpha \text{ is a domain atom on } dom(\mathcal{M}) \text{ and } \mathcal{M} \not\models \alpha\}.$$

A structure $\mathcal{M}$ is a stable model of E-disjunctive program $\mathcal{P}$ (denoted $\mathcal{M} \models \mathcal{P}$) if $\mathcal{M}$ is a minimal set X satisfying the condition: for any rule $r \in \mathcal{P}$ and any variable assignment η, if the literal set $X \cup \mathcal{M}^-$ logically entails $body(r)\eta$, then for some assignment θ, and for some α in the head of r, $(\alpha\eta|_{\overline{x}})\theta \in X$. A rule rule of the form (5) is called a *constraint* if $m = 0$.

Definition 4.1. *Let $\mathcal{P}$ be an E-disjunctive program. The* corresponding FO(C)*-theory is the theory $\mathcal{T}_\mathcal{P}$ with as* C-LOG *expression the* **And***-conjunction of all expressions*

$$\mathbf{All}\, \overline{x}[\beta_1 \wedge \dots \wedge \neg\gamma_n] : \mathbf{Select}\, \overline{y}[\mathbf{t}] : \alpha_1\, \mathbf{Or}\, \dots\, \mathbf{Or}\, \alpha_m$$

such that there is a rule of the form (5) with $m > 0$ in $\mathcal{P}$. $\mathcal{T}_\mathcal{P}$ has as FO part all sentences

$$\forall \overline{x} : \neg(\beta_1 \wedge \dots \wedge \beta_k \wedge \neg\gamma_1 \wedge \dots \wedge \neg\gamma_n)$$

such that there is a constraint of the form (5) in $\mathcal{P}$ and the constraints $\forall \overline{x} : \neg P(\overline{x})$ for symbols P that do not occur in the head of any rule in $\mathcal{P}$.

The last type of constraint is a technical detail: in ASP, all symbols are endogenous, while in C-LOG, this is only the case for predicates occurring in "the head of rules".

The above syntactical correspondence does not always correspond to a semantical correspondence. Intuitively, an E-disjunctive rule r (roughly) means the following: if the body of r holds for an instantiation of $\overline{x}$, then we select one instantiation of the $\overline{y}$ and one disjunct; that disjunct is caused to be true for that instantiation. But, globally the selection should happen in such a way that the final model is minimal. For example the program

$$\{p. \qquad p; q.\}$$

only has one stable model, namely $\{p\}$. The intuition behind it is that the first rule causes p to be true, and hence compromises the choice in the second rule. As p already holds, the global minimality condition ensures that the second rule is obliged to choose p as well, if possible. When we slightly modify the above program, by adding a constraint:

$$\{p. \qquad p; q. \qquad \text{:- not}\, q.\}$$

suddenly, q can (and should) be chosen by the second rule, as $\{p\}$ no longer is a model of this theory. The above illustrates that there is a great interdependency between different rules and between rules and constraints: adding an extra rule or constraint changes the meaning of other rules. Below, we identify a fragment of E-disjunctive ASP in which this dependency is not too strong, and we show that for this fragment, the stable model semantics equals the FO(C) semantics. In order to do so, we introduce the following concepts:

Definition 4.2. *Let δ be a domain atom and r a rule in the form of (5). Suppose η is a variable assignment of the variables $\overline{x}$ and $\overline{y}$. We say that δ* occurs in *r at i for η if $\alpha_i\eta = \delta$. We say that δ* occurs in *r if there exist and i and an η such that r occurs at i for η.*

Definition 4.3. *We call a rule* disjunctive *if $\overline{y}$ is not the empty tuple or if $m > 1$.*

Definition 4.4. *An E-disjunctive program $\mathcal{P}$ is called* causally independent *if for every domain atom δ one of the following holds*

- *δ occurs only in non-disjunctive rules, or*
- *there are at most one rule r, one i, and one η such that δ occurs in r at i for η.*

The above condition states that domain atoms occurring in heads of disjunctive rules, cannot occur multiple times in rule heads. Intuitively, this guarantees that different choices do not interfere.

Theorem 4.5. *Let $\mathcal{P}$ be a causally independent E-disjunctive program without recursion over negation and $\mathcal{T}_\mathcal{P}$ the corresponding* FO(C) *theory. For every structure $\mathcal{I}$, $\mathcal{I} \models \mathcal{P}$ if and only if $\mathcal{I} \models \mathcal{T}_\mathcal{P}$.*

In Theorem 4.5, there is one extra condition on causally independent ASP programs to be equivalent to the corresponding FO(C) theory, namely that it does not contain recursion over negation, i.e., there are no rules of the form

$$p \text{ :- not}\, p'. \qquad p' \text{ :- not}\, p.$$

This kind of pattern does not occur very often. It has already been argued in [6] that in practical applications recursion over negation is only used to "open" the predicate p, i.e., to encode that it can have arbitrary truth value. In this case, the predicate p' would not be used in the rest of the theory. This can as well be done with a rule

$$p; p'.$$

This last rule is equivalent to the above two in causally independent programs (or, if p and p' do not occur in other rule heads). In FO(C), we could either add the disjunctive rule, or simply omit this rule, since exogenous predicates are open anyway.

As already stated above, in case an ASP program is not causally independent, semantics might differ. However, we do have

Theorem 4.6. *Let $\mathcal{P}$ be any E-disjunctive program without recursion over negation and $\mathcal{T}_\mathcal{P}$ be the corresponding* FO(C) *theory. For every structure $\mathcal{I}$, if $\mathcal{I} \models \mathcal{P}$ then also $\mathcal{I} \models \mathcal{T}_\mathcal{P}$.*

The reverse does not hold, since C-LOG does not impose a global minimality condition. The difference in semantics is illustrated in the American Lottery example, which we resume below.

In the above, we argued that for many practical applications of E-disjunctive programs, semantics of FO(C) corresponds to the stable model semantics. This raises the question of relevance of FO(C). From a knowledge representation perspective, FO(C) adds several useful constructs with respect to E-disjunctive logic programs. Among these are nested rules (in fact, arbitrary nesting of expressions), dynamic choice, object creation, and a more modular semantics.

Nested causal rules occur in many places, for example, one could state that the electrician causes a causal link between a button and a light, e.g.,

$$(light \leftarrow button) \leftarrow electrician.$$

We found similar nested rules in [11]. Of course, for simple examples this can also be expressed compactly in ASP, e.g. by

$$light \text{ :- } electrician, button.$$

but when causes and effects are more complex, translating them requires the introduction of auxiliary predicates, diminishing the readability of the resulting program.

Dynamic choices occur in many practical applications. Consider the following situation: a robot enters a room, opens some of the doors in this room, and then leaves by one of the doors that are open. The robot's leaving corresponds to a non-deterministic choice between a *dynamic* set of alternatives, which is determined by the robot's own actions, and therefore cannot be hard-coded into the head of a rule. In C-LOG, we would model this last choice as

$$\mathbf{Select}\, x[open(x)] : leave(x).$$

To model this in an E-disjunctive logic program, we need an extra auxiliary predicate, thus reducing readability:

$$\begin{aligned} &\exists X : chosen(X).\\ &\forall X : leave(X) \text{ :- } chosen(X).\\ &\forall X \text{ :- } chosen(X); \text{not}\, open(X). \end{aligned}$$

Modularity of the semantics has already been discussed above: The causal independency condition on ASP programs guarantees similar modularity. However, when the causal independency condition is violated, semantics of ASP programs are often less clear. Let us reconsider Example 2.2. The E-disjunctive program

$$\begin{aligned} &\exists X : american(X) \text{ :- } lottery.\\ &\forall X : american(X) \text{ :- } passtest(X). \end{aligned}$$

is similar to

$$\begin{aligned} &(\mathbf{Select}\, x[\mathbf{t}] : american(x)) \leftarrow lottery\\ &\mathbf{All}\, x[passtest(x)] : american(x) \end{aligned}$$

Semantically, the first imposes a minimality condition: the lottery is always won by a person succeeding the test, if there exists one. On the other hand, in C-LOG the two rules are independent, and models might not be minimal. In this example, it is the latter that is intended. This illustrates modularity of C-LOG. The rule $(\mathbf{Select}\, x[\mathbf{t}] : american(x)) \leftarrow lottery$ means that one person is selected randomly and caused to be American. Adding other rules does not change the meaning of this rule; causal effects do not interfere.

Object-creation in C-LOG is discussed in the next section.

5 Object-creation in C-LOG

Object creation is available in C-LOG through the $\mathbf{New}$-operator. Like every language construct in C-LOG, the informal interpretation of an expression

$$\mathbf{New}\, x : P(x) \leftarrow \varphi$$

is defined in terms of causal processes. The above expression states that φ causes the creation of a new element and that for that new element, P is caused. Object-creation is also subject to the principles of sufficient causation, universal causation and no self-causation. In order to apply these principles, the domain of a structure is partitioned into two parts: the *initial* elements are those whose existence is not governed by the causal theory, they are exogenous and the *created* elements are those created by expressions in the causal theory, i.e., they are endogenous. For created elements, their default value is *not existing* and their deviant value is existing. Thus, at the start of a causal process, only the initial elements exist, as soon as the preconditions of a $\mathbf{New}$-expressions are satisfied, an element is added to the domain. The principle of no self-causation takes these default and deviant values into account: an object cannot be created based on its own existence. Consider for example the following causal theory:

$$\begin{aligned} &\mathbf{Select}\, x[\mathbf{t}] : P(x)\\ &(\mathbf{New}\, y : Q(y)) \leftarrow \exists x : P(x)\\ &\mathbf{Select}\, x[\mathbf{t}] : R(x) \end{aligned}$$

The first and last expressions select one object randomly and cause P (respectively R) to hold for that object. The second expression creates a new element conditionally, only if there is at least one element satisfying P. In this example, the element selected for the first expression cannot be the one created in the second. $\mathbf{Select}$-operators can only select existing elements and the object created in the second expression can only be created after the selection in the first rule, after there is some object satisfying P. For the last expression, any element can be selected. Hence, this causal theory has no models with only one domain element. A structure $\mathcal{I}$ with domain $\{A, B\}$ and with $P^{\mathcal{I}} = \{A\}$ and $Q^{\mathcal{I}} = R^{\mathcal{I}} = \{B\}$ is a model of the above causal theory. In this case, B is the unique created element, and A is initial, i.e., A is assumed to exist before the described causal process takes place. This illustrates that the $\mathbf{New}$-operator is more than simply a $\mathbf{Select}$ together with unique name axioms: its semantics is really integrated in the underlying causal process. The behaviour of $\mathbf{New}$-expressions can be simulated using $\mathbf{Select}$-expressions if we make the two parts of the domain (initial and created elements) explicit and conditionalise all quantifications. A detailed discussion of this transformation is out of the scope of this paper.

Object creation occurs in many fields, of which we discuss some below.

5.1 Object-Creation in Database Systems

Object-creation has been studied intensively in the field of deductive databases. In [2], various extensions of Datalog are considered, resulting in non-deterministic semantics for queries and updates. One of the studied extensions is object creation. This is implemented for example in the LogicBlox system [7]. An example from the latter paper is the rule:

$$President(p), presidentOf[c] = p \leftarrow Country(c).$$

which means that for every country c, a new (anonymous) "derived entity" of type $President$ is created. Of course, the president of a country is not a new person, but the president is new with respect to the database, which does not contain any persons yet. Such rules with (implicit) existentially quantified head variables correspond to $\mathbf{New}$-expressions. Here, it would translate to

$$\mathbf{All}\, c[Country(c)] : \mathbf{New}\, p : Pres(p)\, \mathbf{And}\, presOf(c,p).$$

This shows that in some rule-based paradigms, an existentially quantified head-variable corresponds to object-creation ($\mathbf{New}$), while in other rule-based paradigms, such as ASP, we saw that an existentially quantified head variable corresponds to a selection. The relation between these paradigms has, to the best of our knowledge, not yet been studied thoroughly. We believe that FO(C), which makes an explicit distinction between selection and object-creation, is an interesting tool to study this relationship. This is future work.

Other Datalog extensions with forms of object creation exist. For example [14] discusses a version with creation of sets and compares its expressivity with simple object creation.

Object, creation also occurs in other database languages, such as for example the query language while$_{\mathrm{new}}$ in [1]. An expression

$$\texttt{while R do (P = new Q)}$$

in that language corresponds to a CEE.

$$\mathbf{All}\, t[R(t)] : \mathbf{All}\, \overline{x}[\mathbf{t}] : \mathbf{New}\, y : P(\overline{x}, y, t+1) \leftarrow Q(\overline{x}, t).$$

In fact C-LOG can simulate the entire language while$_{\mathrm{new}}$, but details of this fall out of the scope of this paper.

5.2 Object-Creation in Mathematics

Object-creation also occurs in mathematics. The set of all natural numbers can be thought of as the set obtained by a process that first creates one element (zero) and for every element in this set, adds another element (its successor). In C-LOG, the above natural language sentences can be modelled as follows

$$\mathbf{New}\, x : (Nat(x)\, \mathbf{And}\, Zero(x))$$
$$\mathbf{All}\, x[Nat(x)] : \mathbf{New}\, y : (Nat(y)\, \mathbf{And}\, Succ(x,y)).$$

Models of the above theory are exactly those structures interpreting $Nat, Zero, Succ$ as the natural numbers, zero and the successor function (modulo isomorphism).

5.3 Object-Creation in Business Rules Systems

Business Rules [5] engines are widely used in the industry. One big drawback of these systems is their inability to perform multiple forms of reasoning. For example, banks might use a Business Rules engine to decide whether someone is eligible for a loan. This approach can be very efficient, but as soon as one is not only interested in the above question, but also in explanations, or suggestions about what to change in order to become eligible, the application should be redesigned. Previous attempts to translate Business Rules applications into a logic with a Tarskian model semantics have been made in [9]. The conclusion of this study was that for such a transformation, we need object creation . We believe that C-LOG provides a suitable form of object-creation for this purpose. As an illustration, the JBoss manual [4] contains the following rule:

```
when Order( customer == null )
then insertLogical(new ValidationResult(
   validation.customer.missing ));
```

This rule means that if an order is created without customer, a new *ValidationResult* is created with the message that the customer is missing. This can be translated to C-LOG as follows:

$$\mathbf{All}\, y[Order(y) \land NoCustumer(y)] :$$
$$\mathbf{New}\, x : ValidationR(x)\, \mathbf{And}\, Message(x, \text{“\ldots”}).$$

A more thorough study of the relationship between the operational semantics of Business Rules systems and the semantics of C-LOG is a topic for future work.

6 Conclusion

In this paper we compared FO(C) to other modelling paradigms. We discussed the semantical relationship between C-LOG and FO. We identified a fragment of E-disjunctive logic programs for which the stable model semantics corresponds to the semantics of FO(C), and argued how FO(C) enriches such programs with several useful modelling constructs. Furthermore, we argued that the object-creation in FO(C) corresponds to the object creation in many related language. Besides technical relationship between these languages, we believe that this discussion also provides insights in the semantics of FO(C).

References

[1] Serge Abiteboul, Richard Hull, and Victor Vianu, *Foundations of Databases*, Addison-Wesley, 1995.

[2] Serge Abiteboul and Victor Vianu, ‘Datalog extensions for database queries and updates’, *J. Comput. Syst. Sci.*, **43**(1), 62–124, (1991).

[3] Bart Bogaerts, Joost Vennekens, Marc Denecker, and Jan Van den Bussche, ‘C-Log: a knowledge representation language of causality’, Technical Report CW 656, Departement of Computer Science, Katholieke Universiteit Leuven, (2014).

[4] P. Browne, *JBoss Drools Business Rules*, From technologies to solutions, Packt Publishing, Limited, 2009.

[5] Business Rules Group, ‘Defining Business Rules ∼ What Are They Really?’, Technical report, (2000).

[6] Marc Denecker, Yuliya Lierler, Miroslaw Truszczynsky, and Joost Vennekens, ‘A Tarskian informal semantics for answer set programming’, in *Technical Communications of the 28th International Conference on Logic Programming,*, eds., Agostino Dovier and Vitor Santos Costa, pp. 277–289. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, (2012).

[7] Todd J. Green, Molham Aref, and Grigoris Karvounarakis, ‘Logicblox, platform and language: A tutorial’, in *Datalog*, eds., Pablo Barceló and Reinhard Pichler, volume 7494 of *LNCS*, pp. 1–8. Springer, (2012).

[8] N. Hall, ‘Two concepts of causation’, in *Causation and Counterfactuals*, (2004).

[9] Pieter Van Hertum, Joost Vennekens, Bart Bogaerts, Jo Devriendt, and Marc Denecker, ‘The effects of buying a new car: an extension of the IDP knowledge base system’, *TPLP*, **13**(4-5-Online-Supplement), (2013).

[10] C. Hitchcock, ‘Prevention, preemption, and the principle of sufficient reason’, *Philosophical review*, **116**(4), (2007).

[11] Robert A. Kowalski and Fariba Sadri, ‘Towards a logic-based unifying framework for computing’, *CoRR*, **abs/1301.6905**, (2013).

[12] J. Pearl, *Causality: Models, Reasoning, and Inference*, Cambridge University Press, 2000.

[13] G. Preyer and G. Peter, *Logical Form and Language*, Clarendon Press, 2002.

[14] J. Van den Bussche and J. Paredaens, ‘The expressive power of complex values in object-based data models’, *Information and Computation*, **120**, 220–236, (1995).

[15] Joost Vennekens, Marc Denecker, and Maurice Bruynooghe, ‘CP-logic: A language of causal probabilistic events and its relation to logic programming’, *Theory and Practice of Logic Programming*, **9**(3), 245–308, (2009).

[16] Jia-Huai You, Heng Zhang, and Yan Zhang, ‘Disjunctive logic programs with existential quantification in rule heads’, *Theory and Practice of Logic Programming*, **13**, 563–578, (7 2013).