Towards a Temporal Account of Contrary-to-Duty Constraints over Complex Actions in the Situation Calculus

Jens Claßen , **James P. Delgrande**
School of Computing Science, Simon Fraser University, Burnaby, BC, Canada
jens_classen@sfu.ca, jim@cs.sfu.ca

Abstract

With the advent of artificial agents in everyday life, it is important that these agents are guided by social norms and moral guidelines. Notions of obligation, permission, and the like have traditionally been studied in the field of Deontic Logic, where deontic assertions generally refer to what an agent should or should not do; that is they refer to *actions*. In Artificial Intelligence, the Situation Calculus is (arguably) the best known and most studied formalism for reasoning about action and change. In this paper, we further investigate the integration of these two areas, particularly addressing so-called contrary-to-duty (CTD) scenarios. For this purpose, we present a new logic based on Lakemeyer and Levesque's modal Situation Calculus variant $\mathcal{ES}$ that we modify to express properties about programs from the action language GOLOG, extended by new constructs for negated programs and their joint execution. We use this formalism to discuss three different approaches to CTD scenarios. First, we show it to be expressive enough to fully capture Meyer's dynamic deontic logic $\mathrm{PD_eL}$, and hence corresponding solutions for CTDs. Second, we demonstrate how our previous approach to tackle CTDs in terms of defeasible conditionals over a restricted set of GOLOG programs can be represented as well, along with a method to compile them directly into the Situation Calculus action theory. Finally, we extend the language of these conditionals to include a simple notion of intention, which allows to describe CTDs not only in terms of actions that will follow immediately, but that the agent has committed to execute at some time in the foreseeable future. All in all, the contribution of the paper is thus an approach that is substantially more general than previous approaches, and is able to handle CTDs in a flexible manner.

1 Introduction

With artificial agents playing an ever-greater role in our daily lives, there has been increasing interest in researching ways to ensure that such agents act ethically and subject their actions to social norms, in particular where they interact with humans or operate in shared environments. One possible approach is to formalize relevant notions such as obligation, permission and prohibition in a logical language, which traditionally has been the subject of study in the field of Deontic Logic (von Wright 1951; Gabbay et al. 2013).

Probably the best researched system of deontic logic is *Standard Deontic Logic* (SDL), a variant of the modal logic KD (Chellas 1980), where a modal operator $\mathbf{O}\phi$ expresses that "ϕ is obligatory" or "it ought to be that ϕ", permission is defined as its dual ($\mathbf{P}\phi = \neg\mathbf{O}\neg\phi$), and prohibition as the negation of permission ($\mathbf{F}\phi = \neg\mathbf{P}\phi$). Semantically, accessible worlds correspond to worlds that are in a certain sense *ideal*, and obligatory/permitted/forbidden is whatever is true in all/some/no accessible worlds.

While simple and elegant, SDL is also somewhat weak, and yields some unintuitive consequences, which have been traditionally referred to as "paradoxes" in the literature. One particular class of such paradoxes is concerned with so-called *contrary-to-duty* (CTD) obligations, usually given in the form of conditional exhortations that state what ought to be (done) if a certain other obligation is neglected. A well-known example scenario is due to Chisholm (1963), and can be phrased as follows:

1. You ought to help your neighbour.
2. If you help your neighbour you should tell them.
3. If you don't help your neighbour, you shouldn't tell them.
4. You don't help your neighbour.

Intuitively, these statements are consistent, independent from another, and lead to the conclusion that one shouldn't tell the neighbour one will come to help. However, different possible encodings in SDL all either lead to an inconsistency, or that one of the statements can be derived from the others. It was later recognized (Hansson 1969) that the problem lies in representing these statements through monadic deontic modalities and material implications, and that it rather requires *dyadic* obligations such as $\mathbf{O}(\mathit{tell}/\mathit{help})$ to express systems of *defeasible* conditionals. Semantically, the latter do not merely distinguish ideal from non-ideal worlds, but rank possible worlds according to some preference relation, allowing for differing "degrees of ideality". For example, worlds in which we don't go to help the neighbour but tell them we are coming are ranked worse than those where we don't go, but at least don't tell them we intend to come, even though both cases are not ideal.

Another observation about the Chisholm scenario is that there is a temporal aspect to it: If we are *going to* help, then we ought to tell them *beforehand*. Furthermore, here deontic modalities apply to actions ("ought-to-do") rather than propositions ("ought-to-be"). While some authors simply used propositions to represent actions, in his seminal article, von Wright (1951) originally introduced deontic modalities as applying to action types. He argued that a suitable deontic logic needs to be built upon the foundation of a more general theory of action (von Wright 1963). Essentially, when reasoning about obligations and permissions applying to actions, we have to take into consideration that actions have preconditions and effects that result in various forms of interaction and interdependency between them, and so it makes sense to formalize these notions. For example, helping the neighbour may require having the necessary supplies to do so, which may necessitate other actions, such as buying supplies at the hardware store.

Deontic action logic is an active area of research, and notable approaches to use such formalisms for tackling CTDs include (Bartha 1999), which uses *stit* ("see to it that") semantics (Horty 2001), and (Meyer, Dignum, and Wieringa 1994), which is based on Meyer's (1988) deontic dynamic logic $\mathrm{PD_eL}$. While (Bartha 1999) extends the aforementioned idea to assign degrees of ideality to possible histories (rather than worlds), a problem with stit is that actions do not have proper names or types, but are described purely through their effects, making it difficult to deal with deontic constraints over complex actions. This is not an issue in dynamic logic, but the approach to CTDs suggested in (Meyer, Dignum, and Wieringa 1994) is somewhat rudimentary in that rankings among alternatives are not inferred "automatically" by means of some non-monotonic mechanism, but need to be encoded "manually" by the domain designer.

In a recent paper (Claßen and Delgrande 2020), we proposed to tackle CTDs over actions by integrating deontic notions into what is (arguably) the best known and most studied formalism for reasoning about action and change, namely the Situation Calculus (McCarthy and Hayes 1969; Reiter 2001), together with the agent programming language GOLOG (Levesque et al. 1997) that is defined on top of it. Among other things, we proposed to express dyadic obligations as defeasible conditionals over complex actions (i.e., programs of GOLOG), and understand them as *deontic constraints* that the agent has to consider when planning its actions. These conditionals would then again induce a ranking of differing "degrees of ideality", but over situations (i.e., action sequences) instead of possible worlds. Moreover, we showed that these constraints can then be "compiled away" into the action theory, so that after a preprocessing step, no additional reasoning machinery is required for planning under such deontic constraints. A limitation was that for conditionals we considered a very restricted fragment of GOLOG programs that only admit single actions, one of the reasons being that the approach requires a notion of *negated* actions and programs, e.g. to express "not helping the neighbour", which is not trivial in the general case. Another limiting assumption we made is that the action the agent is "going to do" (e.g., helping) will follow immediately after the one it is currently deliberating about (e.g., telling).

In this paper, we address some of these issues and explore a more unified view on contrary-to-duty constraints over actions. For this purpose, in Section 2, we propose a new logic called $\mathcal{ESGL}$ that is based on an extension (Claßen and Lakemeyer 2008) of Lakemeyer and Levesque's (2010) modal Situation Calculus variant $\mathcal{ES}$, which we modify to express properties about a fragment of GOLOG programs, now including a more sophisticated notion of action negation as proposed by Meyer (1988). While the classical Situation Calculus is defined axiomatically over Tarskian structures, the modal variant we employ here uses a special semantics that renders many formal definitions and proofs easier, while retaining all benefits such as Reiter's (1991) solution to the frame problem. In particular, this is helpful for defining the new negation operator, where we shift from a macro-based definition of GOLOG (Levesque et al. 1997) to a transition-based semantics (De Giacomo, Lespérance, and Levesque 2000). Moreover, different from previous definitions, our semantics uses linear-time traces rather than branching-time tree models, which further simplifies the treatment. We use the new formalism to discuss three different approaches to CTDs. First, in Section 3 we show it to be expressive enough to fully capture Meyer's dynamic deontic logic $\mathrm{PD_eL}$, and hence corresponding solutions for CTDs. Second, in Section 4 we demonstrate how our previous approach to tackle CTDs in terms of defeasible conditionals over a restricted set of GOLOG programs can equally be represented. Finally, in Section 5 we extend the language of conditionals to include a simple notion of intention, which allows to describe CTDs not only in terms of actions that will follow immediately, but that the agent has committed to execute at some time in the foreseeable future. The overall contribution of this paper is hence an approach that is substantially more general than previous works, allowing for a flexible modelling of, among other things, CTDs in the style of Chisholm's paradox.

2 The Logic $\mathcal{ESGL}$

In this section we present the formal definition of the logic $\mathcal{ESGL}$. It is based on Lakemeyer and Levesque's (2010) logic $\mathcal{ES}$, a modal variant of the (epistemic) situation calculus, where instead of situation terms, modal operators $[t]\phi$ ("ϕ is true after action t") and $\Box\phi$ ("ϕ holds after any sequence of actions") are used to talk about future states of affairs. Our new logic is a variant of Claßen and Lakemeyer's (2008; 2013) extension $\mathcal{ESG}$, which, among other things, extends the $[\cdot]$ operator to take a program (or complex action) δ as argument, where δ is from a subset of the agent programming language GOLOG (Levesque et al. 1997). The latter includes both deterministic programming constructs such as while loops and if conditionals, and non-deterministic ones such as non-deterministic branching and iteration.

While the main purpose of $\mathcal{ESG}$ was the verification of GOLOG programs, our focus of interest in this paper is representing and reasoning about deontic properties. The new logic $\mathcal{ESGL}$ we propose differs from $\mathcal{ESG}$ in two aspects: For one, we extend the set of GOLOG programming constructs by negation ($\overline{\delta}$) and joint execution ($\delta_1 \times \delta_2$) of programs, which will allow to express deontic constraints as conditionals over GOLOG programs. For another, instead of interpreting formulas over branching-time, tree-shaped models as is done in the situation calculus and $\mathcal{ES}$, we will use linear-time models called traces. The latter not only is somewhat simpler, but, as we will see, helps in interpreting the new constructs in a similar fashion as in Meyer's (1988) dynamic deontic logic, thus inheriting many of its desirable features. Note though that this section solely deals with interpretating programs and their properties, and that deontic notions will only be introduced and discussed in the subsequent sections.

2.1 Syntax

The language is a first-order modal dialect with equality and sorts of type *object*, *action* and *number*. It includes countably infinitely many standard names for each of the sorts, denoted by $\mathcal{N}_O$, $\mathcal{N}_A$, and $\mathcal{N}_N$ respectively, allowing for a substitutional interpretation of quantification. Also included are both fluent and rigid predicate and function symbols. Fluents vary as the result of actions, but rigids do not. The logical connectives are $\wedge$, $\neg$, $\forall$, together with the modal operator $\langle\delta\rangle$, where δ may be any program expression, as defined below. Other connectives like $\vee$, $\supset$, $\subset$, $\equiv$, and $\exists$ are used as the usual abbreviations, and terms and formulas are built from these primitives in the usual way, including basic arithmetic operations and relations for numbers.

We read $\langle\delta\rangle\phi$ as "ϕ holds after some execution of program δ" and define its dual $[\delta]\phi$ as abbreviation for $\neg\langle\delta\rangle\neg\phi$, where $\Box\phi$ (read: "ϕ holds after any sequence of actions") in turn stands for $[\top]\phi$. The set of *programs* Δ is given by:

$$\delta ::= \; t \mid \phi? \mid \delta_1;\delta_2 \mid \overline{\delta} \mid \delta_1+\delta_2 \mid \pi x.\delta \mid \delta^* \qquad (1)$$

in which t can be any action term (including a variable), ϕ a formula, and x a variable. We thus consider a set of programs given by primitive actions t, test conditions $\phi?$, sequence $\delta_1;\delta_2$, action negation $\overline{\delta}$, nondeterministic branching $\delta_1+\delta_2$, nondeterministic choice of argument ("pick") $\pi x.\delta$, and nondeterministic iteration δ^*. In addition, we define joint execution $\delta_1\times\delta_2$ as abbreviation for $\overline{\overline{\delta_1}+\overline{\delta_2}}$, the empty program *nil* as TRUE?, the universal action $\top$ as $\pi a.\, a^*$, and failure $\bot$ as FALSE?. For any expression (formula, term, program,...) β, we use β^x_t to denote the result of simultaneously replacing all free occurrences of variable x by term t. We call a formula without $\Box$ and $[\cdot]$ a *fluent formula*, and one without free variables a *sentence*.

2.2 Semantics

Intuitively, a *trace* τ will be used to determine, at any point in time $k\in\mathbb{N}$, (a) what values the (fluent and rigid) predicates and functions take and (b) what action will be executed next. For the latter, we simply assume that there is a distinguished functional fluent $\aleph$ of sort action with this special meaning, but that we otherwise treat like any other function symbol. More precisely, let

- $\mathcal{N}$ denote the set of all standard names,
- $\mathcal{P}_F$ the set of all primitive sentences $R(n_1,\ldots,n_m)$, where R is a (fluent or rigid) predicate symbol and all the n_i are standard names, and
- $\mathcal{P}_T$ the set of all primitive terms $g(n_1,\ldots,n_m)$, where g is a (fluent or rigid) function symbol and all the n_i are standard names.

Then a trace $\tau\in\mathcal{T}$ is any mapping

$$\tau:\mathbb{N}\times\mathcal{P}_F\to\{0,1\} \qquad\qquad \tau:\mathbb{N}\times\mathcal{P}_T\to\mathcal{N}$$

that preserves sorts, interprets arithmetic operations and relations in the usual way, and satisfies the rigidity constraint: if g is a rigid function or predicate symbol, then for all k and k', $\tau[k,g(n_1,\ldots,n_k)] = \tau[k',g(n_1,\ldots,n_k)]$. The *progression* of a trace τ by k time points is the trace $\tau^{(k)}$ where for all $l\in\mathbb{N}$ and all $\beta\in\mathcal{P}_F\cup\mathcal{P}_T$,

$$\tau^{(k)}[l,\beta] = \tau[k+l,\beta].$$

We extend the idea of co-referring standard names to arbitrary ground terms as follows. Given a variable-free term t and a trace τ, we define $|t|_\tau$ (read: the co-referring standard name for t given τ) by:

1. If $t\in\mathcal{N}$, then $|t|_\tau = t$;
2. $|h(t_1,\ldots,t_k)|_\tau = \tau[0,h(n_1,\ldots,n_k)]$, if $n_i = |t_i|_\tau$.

Truth of a sentence ϕ wrt. a trace τ is then given by:

1. $\tau\models F(t_1,\ldots,t_k)$ iff $\tau[0,F(|t_1|_\tau,\ldots,|t_k|_\tau)]=1$;
2. $\tau\models(t_1=t_2)$ iff $|t_1|_\tau$ and $|t_2|_\tau$ are identical;
3. $\tau\models\phi\wedge\psi$ iff $\tau\models\phi$ and $\tau\models\psi$;
4. $\tau\models\neg\phi$ iff $\tau\not\models\phi$;
5. $\tau\models\forall x.\phi$ iff $\tau\models\phi^x_n$ for all $n\in\mathcal{N}_x$;
6. $\tau\models\langle\delta\rangle\phi$ iff $\tau\in\|\delta;\phi?\|$.

Above, $\mathcal{N}_x$ refers to the set of standard names of the same sort as x. A sentence is *satisfiable* if some τ exists with $\tau\models\phi$. When Σ is a set of sentences and ϕ a sentence, we write $\Sigma\models\phi$ (read: "Σ logically entails ϕ") to mean that for every τ, if $\tau\models\beta$ for every $\beta\in\Sigma$, then also $\tau\models\phi$. Finally, we write $\models\phi$ (read: "ϕ is valid") to mean $\{\}\models\phi$.

The interpretation of programs as required in rule 6 is defined by mutual induction. Let a *configuration* $\mathsf{c}=\langle\tau,\delta\rangle$ consist of a trace $\tau\in\mathcal{T}$ (intuitively describing the current and future states of the world) and a program $\delta\in\Delta$ (intuitively what remains to be executed). Then the *final configurations* $\mathcal{F}$ are the least set given by the rules shown in Fig. 2, and for every action name n, the transition relation $\xrightarrow{n}$ among configurations is the least set satisfying the rules shown in Fig. 1. For arbitrary action sequences z, we define the reflexive and transitive closure of $\xrightarrow{n}$ inductively as:

- $\mathsf{c}\xrightarrow{\langle\rangle}\mathsf{c}'$ iff $\mathsf{c}=\mathsf{c}'$;
- $\mathsf{c}\xrightarrow{nz}\mathsf{c}'$ iff there is some c'' such that $\mathsf{c}\xrightarrow{n}\mathsf{c}''$ and $\mathsf{c}''\xrightarrow{z}\mathsf{c}'$.

The *traces admitted by program* δ are then given by

$$\|\delta\| \doteq \{\tau \mid \langle\tau,\delta\rangle\xrightarrow{z}\langle\tau',\delta'\rangle,\ \langle\tau',\delta'\rangle\in\mathcal{F}\} \qquad (2)$$

The interpretation of formulas is standard in the sense that atomic formulas that are not in the scope of some $[\cdot]$ or $\langle\cdot\rangle$ operator are evaluated at the first time point of the trace (rule 1), and the Boolean connectives are defined as usual. For quantification (rule 5), we follow the substitutional interpretation of (Lakemeyer and Levesque 2010) in which a formula $\forall x P(x)$ holds just in case $P(x)$ is true for every instantiation of x by a standard name of the same sort.

Probably the most noteworthy difference to previous logics is in rule 6: A trace satisfies $\langle\delta\rangle\phi$ just in case it is one of the traces admitted by the program that executes δ and

(T1) $\langle \tau, n \rangle \xrightarrow{n} \langle \tau', nil \rangle$, if $\tau' = \tau^{(1)}$ and $\tau[0, \aleph] = n$;

(T2) $\langle \tau, \delta_1; \delta_2 \rangle \xrightarrow{n} \langle \tau', \gamma; \delta_2 \rangle$, if $\langle \tau, \delta_1 \rangle \xrightarrow{n} \langle \tau', \gamma \rangle$;

(T3) $\langle \tau, \delta_1; \delta_2 \rangle \xrightarrow{n} \langle \tau', \delta' \rangle$,
if $\langle \tau, \delta_1 \rangle \in \mathcal{F}$ and $\langle \tau, \delta_2 \rangle \xrightarrow{n} \langle \tau', \delta' \rangle$;

(T4) $\langle \tau, \delta_1 + \delta_2 \rangle \xrightarrow{n} \langle \tau', \delta' \rangle$,
if $\langle \tau, \delta_1 \rangle \xrightarrow{n} \langle \tau', \delta' \rangle$ or $\langle \tau, \delta_2 \rangle \xrightarrow{n} \langle \tau', \delta' \rangle$;

(T5) $\langle \tau, \pi x.\delta \rangle \xrightarrow{n} \langle \tau', \delta' \rangle$,
if $\langle \tau, \delta^x_m \rangle \xrightarrow{n} \langle \tau', \delta' \rangle$ for some $m \in \mathcal{N}_x$;

(T6) $\langle \tau, \delta^* \rangle \xrightarrow{n} \langle \tau', \gamma; \delta^* \rangle$, if $\langle \tau, \delta \rangle \xrightarrow{n} \langle \tau', \gamma \rangle$;

(T7) $\langle \tau, \overline{m} \rangle \xrightarrow{n} \langle \tau', nil \rangle$,
if $\langle \tau, n \rangle \xrightarrow{n} \langle \tau', nil \rangle$ and $n \neq m \in \mathcal{N}_A$;

(T8) $\langle \tau, \overline{\delta_1; \delta_2} \rangle \xrightarrow{n} \langle \tau', \delta' \rangle$, if $\langle \tau, \overline{\delta_1} \rangle \xrightarrow{n} \langle \tau', \delta' \rangle$;

(T9) $\langle \tau, \overline{\delta_1; \delta_2} \rangle \xrightarrow{n} \langle \tau', \gamma; \overline{\delta_2} \rangle$, if $\langle \tau, \delta_1 \rangle \xrightarrow{n} \langle \tau', \gamma \rangle$;

(T10) $\langle \tau, \overline{\delta_1; \delta_2} \rangle \xrightarrow{n} \langle \tau', \delta' \rangle$,
if $\langle \tau, \delta_1 \rangle \in \mathcal{F}$ and $\langle \tau, \overline{\delta_2} \rangle \xrightarrow{n} \langle \tau', \delta' \rangle$;

(T11) $\langle \tau, \overline{\delta_1 + \delta_2} \rangle \xrightarrow{n} \langle \tau', \delta_1' \times \delta_2' \rangle$,
if $\langle \tau, \overline{\delta_1} \rangle \xrightarrow{n} \langle \tau', \delta_1' \rangle$ and $\langle \tau, \overline{\delta_2} \rangle \xrightarrow{n} \langle \tau', \delta_2' \rangle$;

(T12) $\langle \tau, \overline{\delta_1 + \delta_2} \rangle \xrightarrow{n} \langle \tau', \delta' \rangle$,
if $\langle \tau, \overline{\delta_1} \rangle \xrightarrow{n} \langle \tau', \delta' \rangle$ and $\langle \tau, \overline{\delta_2} \rangle \in \mathcal{F}$;

(T13) $\langle \tau, \overline{\delta_1 + \delta_2} \rangle \xrightarrow{n} \langle \tau', \delta' \rangle$,
if $\langle \tau, \overline{\delta_1} \rangle \in \mathcal{F}$ and $\langle \tau, \overline{\delta_2} \rangle \xrightarrow{n} \langle \tau', \delta' \rangle$;

(T14) $\langle \tau, \overline{\overline{\delta}} \rangle \xrightarrow{n} \langle \tau', \delta' \rangle$, if $\langle \tau, \delta \rangle \xrightarrow{n} \langle \tau', \delta' \rangle$;

(T15) $\langle \tau, \overline{\pi x.\, \delta} \rangle \xrightarrow{n} \langle \tau', \delta' \rangle$,
if for all $n \in \mathcal{N}_x$, $\langle \tau, \delta^x_n \rangle \in \mathcal{F}$ or $\langle \tau, \delta^x_n \rangle \xrightarrow{n} \langle \tau', \delta' \rangle$.

Figure 1: Transition rules for programs

(F1) $\langle \tau, \phi? \rangle \in \mathcal{F}$ if $\tau \models \phi$;

(F2) $\langle \tau, \delta_1; \delta_2 \rangle \in \mathcal{F}$ if $\langle \tau, \delta_1 \rangle \in \mathcal{F}$ and $\langle \tau, \delta_2 \rangle \in \mathcal{F}$;

(F3) $\langle \tau, \delta_1 + \delta_2 \rangle \in \mathcal{F}$ if $\langle \tau, \delta_1 \rangle \in \mathcal{F}$ or $\langle \tau, \delta_2 \rangle \in \mathcal{F}$;

(F4) $\langle \tau, \pi x.\delta \rangle \in \mathcal{F}$ if $\langle \tau, \delta^x_n \rangle \in \mathcal{F}$ for some $n \in \mathcal{N}_x$;

(F5) $\langle \tau, \delta^* \rangle \in \mathcal{F}$;

(F6) $\langle \tau, \overline{\phi?} \rangle \in \mathcal{F}$ if $\tau \not\models \phi$;

(F7) $\langle \tau, \overline{\delta_1; \delta_2} \rangle \in \mathcal{F}$ if $\langle \tau, \overline{\delta_1} \rangle \in \mathcal{F}$ or $\langle \tau, \delta_1; \overline{\delta_2} \rangle \in \mathcal{F}$;

(F8) $\langle \tau, \overline{\delta_1 + \delta_2} \rangle \in \mathcal{F}$ if $\langle \tau, \overline{\delta_1} \rangle \in \mathcal{F}$ and $\langle \tau, \overline{\delta_2} \rangle \in \mathcal{F}$;

(F9) $\langle \tau, \overline{\overline{\delta}} \rangle \in \mathcal{F}$ if $\langle \tau, \delta \rangle \in \mathcal{F}$;

(F10) $\langle \tau, \overline{\pi x.\delta} \rangle \in \mathcal{F}$ if for all $n \in \mathcal{N}_x$, $\langle \tau, \overline{\delta^x_n} \rangle \in \mathcal{F}$.

Figure 2: Finality rules for programs

afterwards tests for ϕ. Note that while a trace is infinite, we require that a successful execution of a program consists of finitely many transition steps leading to a final configuration (2). Our semantics hence follows a similar intuition as the one presented by Meyer (1988), where a terminating program δ corresponds to all traces that start with a sequence of actions compatible with δ, and that afterwards continue indefinitely with the execution of arbitrary actions. A program such as $knock; open(door)$ can be viewed as a constraint on traces τ to satisfy $\tau[0, \aleph] = knock$ and $\tau[1, \aleph] = open(door)$, without saying anything about how to proceed afterwards (e.g., entering the door or not).

The transition semantics shown above is very similar to the one presented in (Claßen and Lakemeyer 2008; Claßen 2013), which in turn is based on the one for CONGOLOG (De Giacomo, Lespérance, and Levesque 2000), but with the modification that tests are interpreted as *conditions* (rule (F1)) rather than *transitions*. Due to the use of linear-time traces instead of branching-time worlds, transition rule (T1) for primitive actions differs in that it is required that the executed action n is actually the one scheduled to be executed next according to trace τ; in a tree-shaped world w this additional requirement would not be necessary since there is a successor situation for every possible action.

The most obvious change is that Figures 1 and 2 contain additional rules for negated programs, and hence provide a semantics for both negation and joint action. In the next section, we explore some properties and show in what sense defining the new constructs in this fashion is reasonable.

2.3 Basic Action Theories

We can formulate action theories in a similar fashion as in the classical Situation Calculus for encoding dynamic domains. Formally:

Definition 1 (Basic Action Theory). *A* basic action theory *(BAT)* $\Sigma = \Sigma_0 \cup \Sigma_{post}$ *is a set of formulas consisting of:*

1. Σ_0, the initial theory, *a finite set of fluent sentences describing the initial state of the world;*

2. Σ_{post}, *a finite set of* successor state axioms *(SSAs) incorporating Reiter's (1991) solution to the frame problem for encoding action effects:*[1]

$$\Box[a]F(\vec{x}) \equiv \gamma_F \tag{3}$$

$$\Box[a]f(\vec{x}) = y \equiv \gamma_f \tag{4}$$

Here it is assumed that one axiom of the form (3) is included for each relational fluent F relevant to the application domain, and one of the form (4) for each functional fluent f relevant to the application domain, and where γ_F is a fluent formula with free variables a and $\vec{x}$, and γ_f a fluent formula with free variables among a, $\vec{x}$, and y.

For simplicity, we don't include a precondition axiom into the BAT. Note that this is without loss of generality when dealing with programs due to the fact that a formula ϕ being

[1]Free variables are understood as universally quantified from the outside, $[t]$ has higher precedence than the logical connectives, and $\Box$ has lower precedence. So $\Box[a]F(\vec{x}) \equiv \gamma_F$ abbreviates $\forall a, \vec{x}.\Box(([a]F(\vec{x})) \equiv \gamma_F)$.

a precondition of action t can be represented by using the program expression $\phi?; t$ in its place.

2.4 Properties

We first note some properties related to joint action and sequence. In what follows, for any two programs δ_1 and δ_2, let $\delta_1 \equiv \delta_2$ stand for $\|\delta_1\| = \|\delta_2\|$.

Proposition 1.

(P1) $\delta_1 \times (\delta_1; \delta_2) \;\equiv\; \delta_1; \delta_2$

(P2) $\bot; \delta \;\equiv\; \bot$

Property (P1) exemplifies the aforementioned understanding of programs as constraints on the possible future courses of action, where after completing a program δ, infinitely many arbitrary actions will follow. The program $\delta_1; \delta_2$ hence constitutes an additional constraint on the set of traces admitted by δ_1 only. (Meyer 1988) gives the example that "opening the door" together with "opening the door and then leaving" is the same as "opening the door and then leaving". Property (P2) is due to the fact that the set of traces admitted by program $\bot$ is already empty.

It should also be noted that the shift from tree-shaped worlds to linear-time traces comes at a loss of expressiveness, and has the effect that $\mathcal{ESGL}$ is not (directly) comparable to $\mathcal{ES}$ or $\mathcal{ESG}$. A deeper analysis is beyond the scope of this paper, but to illustrate the point, consider the question whether a BAT Σ entails the formula $\neg\Box\phi$. Here, it means that along *every* trace consistent with Σ, there will sooner or later come a point where ϕ does not hold. In $\mathcal{ES}$ and $\mathcal{ESG}$, it means that in every world consistent with Σ there is *some* path where ϕ will be false at some point, which is a much weaker condition. However, it can be argued that for the purpose of planning, it is sufficient to look at the projection problem, which is to decide whether $\Sigma \models [\delta]\psi$ for some fluent formula ψ and some program δ that only mentions fluent formulas as tests, and it can be shown that for this class of reasoning tasks, the logics coincide.

Moreover, it is true that branching-time structures have traditionally been favoured in the deontic logic literature, the main reason being that there must be the possibility to violate a norm, as otherwise, if the future were not open, there would be nothing to reason about in terms of deontic properties. However, as we will see, our linear-time semantics equally allows for this possibility due to the fact that when reasoning about deontic constraints, we will consider *sets* of linear traces, some of which may violate certain deontic constraints, and others don't. Intuitively, this is therefore no real restriction as any tree-shaped branching-time model can be understood as a representation of a set of paths (i.e., traces).

3 Relation to $\mathrm{PD_eL}$

In this section we argue that the definitions and extensions presented in the previous section are reasonable in the sense that among other things, they capture Meyer's (1988) dynamic deontic logic $\mathrm{PD_eL}$. It is based on Anderson's (1958) proposal of reducing deontic logic to alethic modal logic by using a distinguished propositional variable V that intuitively represents a "bad state" or the violation of a norm. For dynamic logic, one defines

$$\mathbf{F}\delta \;\doteq\; [\delta]V \tag{5}$$

saying that an action δ is forbidden if its execution leads to a violation. Permission and obligation can then be defined in terms of prohibition as usual:

$$\mathbf{P}\delta \;\doteq\; \neg\mathbf{F}\delta \quad \text{and} \quad \mathbf{O}\delta \;\doteq\; \mathbf{F}\overline{\delta} \tag{6}$$

Obviously, this requires the action algebra to include an operator for negating actions in order to represent "ought-to-do" obligations $\mathbf{O}\delta$. As it turns out, the question of how to define the negation of a complex action is far from trivial[2]. For $\mathrm{PD_eL}$, Meyer presents the following five axioms as desiderata that he argues must "reasonably hold" for $\overline{\delta}$:

(N1) $\overline{\overline{\delta_1}} \;\equiv\; \delta_1$

(N2) $\overline{\delta_1; \delta_2} \;\equiv\; \overline{\delta_1} + \delta_1; \overline{\delta_2}$

(N3) $\overline{\delta_1 + \delta_2} \;\equiv\; \overline{\delta_1} \times \overline{\delta_2}$

(N4) $\overline{\delta_1 \times \delta_2} \;\equiv\; \overline{\delta_1} + \overline{\delta_2}$

(N5) $\overline{\phi \to \delta_1/\delta_2} \;\equiv\; \phi \to \overline{\delta_1}/\overline{\delta_2}$

The most interesting of these properties is probably (N2). It says that there are exactly two possible ways of executing the negation of a sequential program $\delta_1; \delta_2$: Either do something next that is "not δ_1", or if doing δ_1, then do something afterwards that is "not δ_2". The last property defines the negation of a conditional action ("if ϕ then δ_1 else δ_2"), which we can define in GOLOG by means of

$$\phi \to \delta_1/\delta_2 \;\doteq\; [\phi?; \delta_1] + [\neg\phi?; \delta_2] \tag{7}$$

With the definition presented in the previous section, we get:

Proposition 2. *(N1)–(N5) are valid in* $\mathcal{ESGL}$.

Meyer's action algebra does not include tests, the Kleene star, or pick operators ($\mathrm{PD_eL}$ is propositional). While for the pick operator, which essentially behaves like an existential quantifier, there is no obvious dual, we note that our transition semantics is compatible for tests and iteration in the following sense:

(N6) $\overline{\phi?} \;\equiv\; \neg\phi?$

(N7) $\overline{\delta^*} \;\equiv\; \bot$

(N6) follows from (N5) and the fact that $\phi? \equiv \phi \to \mathit{nil}/\bot$, using $\overline{\bot} \equiv \mathit{nil}$ and $\overline{\mathit{nil}} \equiv \bot$. (N7) makes sense when considering the expansion law for the Kleene star

$$\delta^* \;\equiv\; \mathit{nil} + \delta; \delta^*$$

as then

$$\overline{\delta^*} \equiv \overline{\mathit{nil} + \delta; \delta^*} \equiv \overline{\mathit{nil}} \times \overline{\delta; \delta^*} \equiv \bot \times \overline{\delta; \delta^*} \equiv \bot.$$

Based on (N1)–(N5), Meyer proposes the system $\mathrm{PD_eL}$ as given by the axioms and inference rules shown in Figure 3. He argues that this is sufficient to entail many important theorems of deontic logic (the paper lists 36 of them) when the deontic modalities are understood according to (5) and (6). We note that $\mathcal{ESGL}$ subsumes $\mathrm{PD_eL}$ as follows, assuming that $\mathit{duration}(\delta)$ denotes the maximal length of action sequences admitted by δ:

[2]See (Claßen and Delgrande 2020, Section 3.2) for a brief discussion. Alternatives to Meyer's definition were proposed e.g. by van der Meyden (1996) and Broersen (2004a).

Axioms

(PC) all tautologies of propositional logic

($\Box\supset$) $[\delta](\phi_1 \supset \phi_2) \supset [\delta]\phi_1 \supset [\delta]\phi_2$

(;) $[\delta_1 \ ; \ \delta_2]\phi \equiv [\delta_1][\delta_2]\phi$

($+$) $[\delta_1 + \delta_2]\phi \equiv [\delta_1]\phi \wedge [\delta_2]\phi$

($\times$) $[\delta_1 \times \delta_2]\phi \subset [\delta_1]\phi \vee [\delta_2]\phi$

(provided $\mathit{duration}(\delta_1) = \mathit{duration}(\delta_2)$)

($\rightarrow$) $[\phi \rightarrow \delta_1/\delta_2]\psi \equiv (\phi \supset [\delta_1]\psi) \wedge (\neg\phi \supset [\delta_2]\psi)$

($\Diamond$) $\langle\delta\rangle\phi \equiv \neg[\delta]\neg\phi$

($\overline{;}$) $[\overline{\delta_1 \ ; \ \delta_2}]\phi \equiv [\overline{\delta_1}]\phi \wedge [\delta_1][\overline{\delta_2}]\phi$

($\overline{+}$) $[\overline{\delta_1 + \delta_2}]\phi \subset [\overline{\delta_1}]\phi \vee [\overline{\delta_2}]\phi$

(provided $\mathit{duration}(\delta_1) = \mathit{duration}(\delta_2)$)

($\overline{\times}$) $[\overline{\delta_1 \times \delta_2}]\phi \equiv [\overline{\delta_1}]\phi \wedge [\overline{\delta_2}]\phi$

($\overline{\rightarrow}$) $[\overline{\phi \rightarrow \delta_1/\delta_2}]\psi \equiv (\phi \supset [\overline{\delta_1}]\psi) \wedge (\neg\phi \supset [\overline{\delta_2}]\psi)$

($\overline{}$) $[\overline{\overline{\delta}}]\phi \equiv [\delta]\phi$

($\bot$) $[\bot]\phi$

Rules

(MP) From ϕ and $\phi \supset \psi$ infer ψ.

(N) From ϕ infer $[\delta]\phi$.

Figure 3: The system PDeL

Proposition 3. *In $\mathcal{ESGL}$, axioms (PC)–($\bot$) are valid, and inference rules (MP) and (N) are sound.*

We remark that under our transition semantics, programs in general do not constitute a Boolean algebra (Figure 4):

Proposition 4. *Axioms (B1) – (B10) of Boolean algebras are valid in $\mathcal{ESGL}$, but axioms (B11) and (B12) are not.*

This means that while the usual laws of associativity, commutativity, neutral elements, distributivity, and idempotency apply, the complement does not always behave as expected. A simple counterexample for (B12) is the program $\delta = (a + a; b); c$, where a, b and c are primitive actions. A trace τ that executes $\langle a, b, c\rangle$ as its first three actions (i.e. $\tau[0,\aleph] = a$, $\tau[1,\aleph] = b$, $\tau[2,\aleph] = c$) is an execution of *both* δ and its negation:

$$\langle \tau, (a + a; b); c\rangle \xrightarrow{a} \langle \tau^{(1)}, b; c\rangle \xrightarrow{b} \langle \tau^{(2)}, c\rangle \xrightarrow{c} \langle \tau^{(3)}, \mathit{nil}\rangle$$
$$\langle \tau, \overline{(a + a; b); c}\rangle \xrightarrow{a} \langle \tau^{(1)}, \overline{c}\rangle \xrightarrow{b} \langle \tau^{(2)}, \mathit{nil}\rangle$$

Intuitively, this is due to property (N2): One way of executing the negation of a sequence $\delta_1; \delta_2$ is to execute δ_1, followed by the negation of δ_2. Here, doing action a is one way of executing $(a + a; b)$, and doing action b is one way of executing the negation of action c. While this behaviour may (or may not) be undesirable, note that this is already possible in Meyer's original system $\mathrm{PD_eL}$ (and does not conflict with any results concerning deontic properties). To avoid it, one would have to include counterparts of (B11) and (B12) as additional axioms for $\mathrm{PD_eL}$. The appendix in (Meyer 1988)

(B1) $(\delta_1 + \delta_2) + \delta_3 \equiv \delta_1 + (\delta_2 + \delta_3)$ ($+$ is associative)

(B2) $\delta_1 + \delta_2 \equiv \delta_2 + \delta_1$ ($+$ is commutative)

(B3) $\delta_1 + \bot \equiv \delta_1$ ($\bot$ is the neutral element wrt $+$)

(B4) $\delta_1 + (\delta_2 \times \delta_3) \equiv (\delta_1 + \delta_2) \times (\delta_1 + \delta_3)$ ($+$ distrib.)

(B5) $\delta_1 + \delta_1 \equiv \delta_1$ ($+$ is idempotent)

(B6) $(\delta_1 \times \delta_2) \times \delta_3 \equiv \delta_1 \times (\delta_2 \times \delta_3)$ ($\times$ is associative)

(B7) $\delta_1 \times \delta_2 \equiv \delta_2 \times \delta_1$ ($\times$ is commutative)

(B8) $\delta_1 \times \top \equiv \delta_1$ ($\top$ is the neutral element wrt $\times$)

(B9) $\delta_1 \times (\delta_2 + \delta_3) \equiv (\delta_1 \times \delta_2) + (\delta_1 \times \delta_3)$ ($\times$ distrib.)

(B10) $\delta_1 \times \delta_1 \equiv \delta_1$ ($\times$ is idempotent)

(B11) $\delta_1 + \overline{\delta_1} \equiv \top$ (complement wrt $+$)

(B12) $\delta_1 \times \overline{\delta_1} \equiv \bot$ (complement wrt $\times$)

Figure 4: Axioms of a Boolean algebra

provides the definition for a semantics satisfying these additional properties, but is (arguably) more involved than what we present here. In particular, it requires to consider *sets* of traces, rather than traces, as the semantical domain.

3.1 Representing the Chisholm Scenario

In (Meyer, Dignum, and Wieringa 1994) and (Meyer, Wieringa, and Dignum 1998), the authors suggest to address contrary-to-duty obligations by extending the formalism to include multiple violation atoms $V_1, V_2, V_3, \ldots$ and use accordingly indexed deontic modalities. The Chisholm scenario, for example, could then be expressed as follows:

$$\mathbf{O}_1 h \tag{8}$$
$$\mathbf{F}_2(\overline{t}; h) \tag{9}$$
$$\mathbf{F}_3(t; \overline{h}) \tag{10}$$

saying that one ought help the neighbour, that it is forbidden to not tell and then help, and that is also prohibited to tell followed by not helping. The fact that one actually does not go to help cannot be represented explicitly because of dynamic logic being about hypothetical reasoning in the form of "*if* a certain action is taken, *then* a certain result is obtained."

With the additional assumption that violations persist (by including $V_i \supset [\delta]V_i$ as an additional axiom schema), it is now possible to reason about sub-ideal states in terms of which norms have been violated. For example, telling followed by helping will result in V_1 (the first norm is still violated because we didn't help immediately as next action), but telling and not helping in $V_1 \wedge V_2 \wedge V_3$, so the former should be preferred over the latter.

There are multiple drawbacks to this approach. First, a preference relation among states with different violations has to be defined explicitly. While this arguably allows for a certain flexibility, e.g. to say that some violations are more severe than others, the number of combinations to be considered grows exponentially with the number of violation atoms, i.e., constraints. Second, the authors "admit that it would be far nicer to have a representation closer to the natural language representation, but this would call for a non-trivial extension of PD$_e$L, in which one can also reason 'backward' directly." Third, notice that even in the intuitively ideal case where the agent tells and actually helps, constraint 8 will cause a violation due to the fact that helping was not the immediate next action. What is missing is a notion of an agent *intending* to help in the foreseeable future. We will address these issues in the following sections.

4 Simple Temporal Conditionals

In this section we show that $\mathcal{ESGL}$ is also capable of capturing the approach presented in (Claßen and Delgrande 2020), where deontic constraints are expressed as conditionals over (a restricted set of) GOLOG programs. Specifically, here we are interested in conditionals of the temporal kind that allow to represent scenarios such as the Chisholm set. The set of GOLOG programs in question is as follows:

Definition 2 (Guarded-Action Fragment)**.** *The set of* guarded actions *is given by the following grammar:*

$$\gamma \ ::= \ t \mid \pi x.\gamma \mid \phi?;\gamma$$

The guarded-action fragment *is then given by*

$$\delta \ ::= \ \gamma \mid \overline{\delta} \mid \delta + \delta \mid \delta \times \delta$$

where γ is a guarded action.

Guarded actions hence are primitive actions, possibly preceded by a sequence of picks and test conditions, and the guarded-action fragment is all their Boolean combinations. An important special case is the "wildcard" action $\star \doteq \pi a.a$. We note that

Proposition 5. *The guarded-action fragment is a Boolean algebra with $\star$ as neutral element wrt $\times$.*

This means the laws shown in Figure 4 are valid for this restricted set if we substitute $\star$ for $\top$. Moreover, note that for such programs, joint execution distributes over sequence:

Proposition 6. *Let $\delta_1, \ldots, \delta_4$ be of the guarded-action fragment. Then* $(\delta_1;\delta_2) \times (\delta_3;\delta_4) \equiv (\delta_1 \times \delta_3);(\delta_2 \times \delta_4)$.

We then use programs from the guarded-action fragment to express deontic constraints as described below.

Definition 3 (Temporal Conditionals)**.** *A* temporal deontic conditional *is an expression that is of the form*

$$\delta \Rightarrow_a \gamma \quad or \quad \delta \Rightarrow_b \gamma$$

where δ and γ are from the guarded-action fragment. We read $\delta \Rightarrow_a \gamma$ as "if committed to doing δ, the agent ought to do γ afterwards*", and $\delta \Rightarrow_b \gamma$ as "*… before.*" This definition includes the special case of unconditional constraints where $\delta = \star$. The* materialization *of a rule is given by*

$$\mathfrak{M}(\delta \Rightarrow_a \gamma) \ \doteq \ \overline{\delta};\star + \star;\gamma$$
$$\mathfrak{M}(\delta \Rightarrow_b \gamma) \ \doteq \ \star;\overline{\delta} + \gamma;\star$$

For a finite set of rules $\rho = \{r_1, \ldots, r_k\}$ we understand $\mathfrak{M}(\rho)$ as $\mathfrak{M}(r_1) \times \cdots \times \mathfrak{M}(r_k)$, where $\mathfrak{M}(\emptyset) = \star;\star$.

A set of such defeasible conditionals now induces a ranking over traces using a construction similar to the one for *rational closure* (Kraus, Lehmann, and Magidor 1990):

Definition 4 (Ranking)**.** *Given a finite set ρ of temporal conditionals over programs and a set of traces e, a* ranking *of the rules in ρ wrt e is given by*

$$\begin{aligned}\rho^e_0 \ &= \ \rho \\ \rho^e_{i+1} \ &= \ \{(\delta \Rightarrow_a \gamma) \in \rho^e_i \mid e \cap \|\mathfrak{M}(\rho^e_i) \times (\delta;\star)\| = \emptyset\} \cup \\ &\qquad \{(\delta \Rightarrow_b \gamma) \in \rho^e_i \mid e \cap \|\mathfrak{M}(\rho^e_i) \times (\star;\delta)\| = \emptyset\}\end{aligned}$$

Rules $r \in \rho^e_{i+1}$ are called exceptional *wrt ρ^e_i. For every $\tau \in e$, the* rank assigned by ρ wrt e *then is*

$$\mathsf{Rank}(\tau, e, \rho) = \min\{i \mid \tau \in \|\mathfrak{M}(\rho^e_i)\|\}.$$

The cumulative rank assigned by e *to any time point $k \in \mathbb{N}$ is given by the sum of all ranks from times 0 up to k:*

$$\mathsf{CRank}(\tau, e, \rho, k) = \sum_{i=0}^{k} \mathsf{Rank}(\tau^{(i)}, e^{(i)}, \rho)$$

where $e^{(i)} = \{\tau^{(i)} \mid \tau \in e\}$.

Here we follow (Claßen and Delgrande 2020) for aggregating ranks over time points by simply summing them up. Intuitively, this means that a trace will be ranked as less ideal the more "bad" actions are performed in it. In particular, there is no way of undoing a bad act, and any further bad deed makes the course of action less and less ideal.

4.1 Representing the Chisholm Scenario

Using simple temporal constraints, the first three statements of the Chisholm scenario can be expressed as:

$$\star \ \Rightarrow_a \ \mathit{help} \tag{11}$$
$$\mathit{help} \ \Rightarrow_b \ \mathit{tell} \tag{12}$$
$$\overline{\mathit{help}} \ \Rightarrow_b \ \overline{\mathit{tell}} \tag{13}$$

The first rule states that generally, the agent ought to go help the neighbours. The second one means that when the agent intends to go and help, it should tell the neighbours immediately before. If on the other hand, says the third rule, the agent does not intend to go and help, it ought not tell them. Again, the fourth statement of the Chisholm set cannot be represented explicitly due to reasoning in this formalism being purely hypothetical.

Assume that $e = \mathcal{T}$ is the set of all traces. In the following, let h stand for the action term help, and t for tell. Materializing $\rho_0 = \{(11), (12), (13)\}$ then yields:

$$\begin{aligned}\mathfrak{M}((11)) \ &= \ (\overline{\star};\star) + (\star;h) &&\equiv (\star;h) \\ \mathfrak{M}((12)) \ &= \ &&(\star;\overline{h}) + (t;\star) \\ \mathfrak{M}((13)) \ &= \ (\star;\overline{\overline{h}}) + (\overline{t};\star) &&\equiv (\star;h) + (\overline{t};\star)\end{aligned}$$

Recall that $\mathfrak{M}(\rho_0)$ is given by the conjunction of these three expressions. Since according to Proposition 5, the usual distributive laws apply, we can "multiply" them out. Observe that $(\star;h)$ is incompatible with $(\star;\overline{h})$, and that $(t;\star)$ contradicts with $(\overline{t};\star)$. The result is hence equivalent to $(\star;h) \times (t;\star)$, which in turn can be simplified to $(t;h)$ using Propositions 5 and 6. We thus get

$$\begin{aligned}\|(t;h) \times (\star;\star)\| = \|t;h\| &&\neq \emptyset \\ \|(t;h) \times (\star;h)\| = \|t;h\| &&\neq \emptyset \\ \|(t;h) \times (\star;\overline{h})\| = \|\bot\| &&= \emptyset\end{aligned}$$

Because rule (13) is the only exceptional one, we obtain $\rho_1 = \{(13)\}$ and $\mathfrak{M}(\rho_1) \equiv \star; h + \bar{t}; \star$. The rule is obviously not exceptional with itself, so $\rho_2 = \emptyset$, hence $\mathfrak{M}(\rho_2) = \star; \star$. We thus end up with

$$\mathfrak{M}(\rho_0) \equiv t; h, \quad \mathfrak{M}(\rho_1) \equiv \star; h + \bar{t}; \star, \quad \mathfrak{M}(\rho_2) \equiv \star; \star$$

which induces the following ranking:

$$\mathsf{Rank}(\tau, e, \rho) = \begin{cases} 0, & \tau[0, \aleph] = \mathit{tell} \text{ and } \tau[1, \aleph] = \mathit{help} \\ 1, & \tau[0, \aleph] \neq \mathit{tell} \\ 2, & \tau[0, \aleph] = \mathit{tell} \text{ and } \tau[1, \aleph] \neq \mathit{help} \end{cases}$$

4.2 Compiling Conditionals into BATs

In (Claßen and Delgrande 2020), we also showed how deontic constraints can be compiled into the action theory, so that after a preprocessing step, no special (non-monotonic) reasoning machinery is needed for planning under deontic constraints. The basic idea is to use a new function fluent ideal that represents the degree of ideality of the current situation. For this purpose, we include

$$\mathit{ideal} = 0, \tag{14}$$

into the initial theory Σ_0 of our BAT. The value of this fluent may increase due to actions, as per the SSA

$$\Box[a]\mathit{ideal} = \mathit{ideal} + \mathit{bad}(a) \tag{15}$$

that we include into Σ_{post}. The potential increase is determined by another fluent $\mathit{bad}(a)$ that expresses how "bad" an action a is, and that we define further below. First, to keep track of which programs mentioned in deontic constraints have been executed previously, we introduce finitely many additional fluent predicates $\mathit{Did}(\gamma)$, where for each one Σ_0 contains the axiom

$$\neg \mathit{Did}(\gamma) \tag{16}$$

and Σ_{post} contains the SSA

$$\Box[a]\mathit{Did}(\gamma) \equiv \mathfrak{C}[\gamma, a]. \tag{17}$$

The right-hand side of the SSA uses the compilation operator $\mathfrak{C}$ whose definition is given below:

Definition 5.

1. $\mathfrak{C}[\alpha, a] = (a = \alpha)$
2. $\mathfrak{C}[\phi?; \delta, a] = \phi \wedge \mathfrak{C}[\delta, a]$
3. $\mathfrak{C}[\pi v.\delta, a] = \exists v.\ \mathfrak{C}[\delta, a]$
4. $\mathfrak{C}[\overline{\delta}, a] = \neg\mathfrak{C}[\delta, a]$, *if δ is a guarded action*
5. $\mathfrak{C}[\delta_1 + \delta_2, a] = \mathfrak{C}[\delta_1, a] \vee \mathfrak{C}[\delta_2, a]$
6. $\mathfrak{C}[\delta_1 \times \delta_2, a] = \mathfrak{C}[\delta_1, a] \wedge \mathfrak{C}[\delta_2, a]$
7. $\mathfrak{C}[\gamma; \delta, a] = \mathit{Did}(\gamma) \wedge \mathfrak{C}[\delta, a]$

With this operator, we can now define an axiom for bad. Suppose that we determined a ranking as shown previously, then for a finite number of rule sets $\rho_0, \ldots, \rho_k$, we obtained their materialized counterparts

$$\mathfrak{M}(\rho_0) \equiv \delta_0, \quad \mathfrak{M}(\rho_1) \equiv \delta_1, \quad \ldots \quad \mathfrak{M}(\rho_k) \equiv \delta_k$$

where each δ_i is a program from the guarded-action fragment. We then define the badness of action a as the *minimal* index i whose δ_i admits a:

$$\mathit{bad}(a) = b \equiv \bigvee_{i=0}^{k} (b = i) \wedge \mathfrak{C}[\delta_i, a] \ \wedge \bigwedge_{j=0}^{i-1} \neg\mathfrak{C}[\delta_j, a] \tag{18}$$

Proposition 7. *Let Σ be a BAT, ρ a set of simple temporal constraints, Σ_ρ be the result of extending Σ with axioms (14) – (18), and $e = \{\tau \mid \tau \models \Sigma_\rho\}$. For any $\tau \in e$ and $k \in \mathbb{N}$,*

$$\mathsf{CRank}(\tau, e, \rho, k) = d \quad \textit{iff} \quad \tau^{(k)} \models (\mathit{ideal} = d).$$

In the Chisholm example we obtain, after simplifications,

$$\begin{aligned} \mathit{bad}(a) = b \ \equiv \ & b = 0 \wedge \mathit{Did}(\mathit{tell}) \wedge a = \mathit{help} \ \vee \\ & b = 1 \wedge \mathit{Did}(\overline{\mathit{tell}}) \ \vee \\ & b = 2 \wedge \mathit{Did}(\mathit{tell}) \wedge a \neq \mathit{help} \end{aligned} \tag{19}$$

where the SSAs for $\mathit{Did}(\mathit{tell})$ and $\mathit{Did}(\overline{\mathit{tell}})$ are given by

$$\Box[a]\mathit{Did}(\mathit{tell}) \ \equiv \ a = \mathit{tell} \tag{20}$$

$$\Box[a]\mathit{Did}(\overline{\mathit{tell}}) \ \equiv \ a \neq \mathit{tell} \tag{21}$$

5 Intentional Conditionals

One shortcoming of the approaches discussed in the previous sections is that it is assumed that one action under consideration will follow immediately after the other. This is obviously not a realistic assumption for many practical scenarios. For example, helping the neighbour might necessitate other actions, such as buying supplies at the hardware store. In this section, we hence explore the idea of including a notion of intention, represented by temporal modalities over programs with a finite horizon. Specifically, for any δ from the guarded-action fragment and $k \geq 1$, we will use $\Box_k\delta$ to say "during k steps, always δ", defined through

$$\Box_k\delta \ \doteq \ \delta^k \ \doteq \ \underbrace{\delta; \cdots; \delta}_{k \text{ times}}, \tag{22}$$

and $\Diamond_k\delta$ to express "within k steps, eventually δ", given by

$$\Diamond_k\delta \ \doteq \ \delta + \star; \delta + \star; \star; \delta + \cdots + \underbrace{\star; \cdots; \star}_{k-1 \text{ times}}; \delta. \tag{23}$$

We note that the two operators are indeed duals:

Proposition 8.

(N8) $\overline{\Box_k\delta} \ \equiv \ \Diamond_k\overline{\delta}$

(N9) $\overline{\Diamond_k\delta} \ \equiv \ \Box_k\overline{\delta}$

Definition 6 (Intentional Guarded-Action Fragment). *The* intentional guarded-action fragment *is given by*

$$\delta \ ::= \ \gamma \mid \Box_k\gamma \mid \Diamond_k\gamma \mid \overline{\delta}$$

where γ is from the guarded-action fragment.

Definition 7 (Intentional Conditionals). *An* intentional deontic conditional *is an expression of the form*

$$\delta \Rightarrow \gamma$$

where δ and γ are programs of the intentional guarded-action fragment. The materialization *of a rule is given by*

$$\mathfrak{M}(\delta \Rightarrow \gamma) \ \doteq \ \overline{\delta} + \gamma$$

For a finite set $\rho = \{r_1, \ldots, r_k\}$ we understand $\mathfrak{M}(\rho)$ as $\mathfrak{M}(r_1) \times \cdots \times \mathfrak{M}(r_k)$ as before, but using $\mathfrak{M}(\emptyset) = \top$.

Definition 8 (Intentional Situation Ranking). *Given a finite set ρ of intentional conditionals and a set of traces e, an* intentional ranking *of the rules in ρ wrt e is given by*

$$\rho_0^e \ = \ \rho$$

$$\rho_{i+1}^e \ = \ \{(\delta \Rightarrow \gamma) \in \rho_i^e \mid e \cap \|\mathfrak{M}(\rho_i^e) \times \delta\| = \emptyset\}$$

$\mathsf{Rank}(\tau, e, \rho)$ *and* $\mathsf{CRank}(\tau, e, \rho, k)$ *are exactly as in Def. 4.*

5.1 Representing the Chisholm Scenario

Suppose we want to apply a finite horizon of $k \geq 2$. The first three statements of the Chisholm scenario could be expressed as:

$$\star \;\Rightarrow\; \Diamond_k help \tag{24}$$

$$\Diamond_k help \;\Rightarrow\; tell \tag{25}$$

$$\overline{\Diamond_k help} \;\Rightarrow\; \overline{tell} \tag{26}$$

Assume again that $e = \mathcal{T}$ is the set of all traces, and let h and t abbreviate $help$ and $tell$, respectively. Materializing $\rho_0 = \{(24), (25), (26)\}$ then yields:

$$\begin{aligned}
\mathfrak{M}((24)) &= \overline{\star} + \Diamond_k h && \equiv \Diamond_k h \\
\mathfrak{M}((25)) &= \overline{\Diamond_k h} + t && \equiv \Box_k \overline{h} + t \\
\mathfrak{M}((26)) &= \overline{\overline{\Diamond_k h}} + \overline{t} && \equiv \Diamond_k h + \overline{t}
\end{aligned}$$

Again, we determine the product of these expressions and apply the distributive law. Observe that $\Diamond_k h$ is incompatible with $\Box_k \overline{h}$, and that t contradicts with $\overline{t}$. The result is hence equivalent to $\Diamond_k h \times t$. We thus get

$$\begin{aligned}
&\|(\Diamond_k h \times t) \times \star\| && = \|\Diamond_k h \times t\| && \neq \emptyset \\
&\|(\Diamond_k h \times t) \times \Diamond_k h\| && = \|\Diamond_k h \times t\| && \neq \emptyset \\
&\|(\Diamond_k h \times t) \times \Box_k \overline{h}\| && = \|\bot\| && = \emptyset
\end{aligned}$$

Similar to before, we obtain $\rho_1 = \{(26)\}$ and $\rho_2 = \emptyset$, hence

$$\mathfrak{M}(\rho_0) \equiv \Diamond_k h \times t, \;\; \mathfrak{M}(\rho_1) \equiv \Diamond_k h + \overline{t}, \;\; \mathfrak{M}(\rho_2) \equiv \top$$

which induces the following ranking:

$$\mathsf{Rank}(\tau, e, \rho) = \begin{cases} 0, & \tau[0,\aleph] = tell \text{ and} \\ & \tau[i,\aleph] = help \text{ for some } 1 \leq i \leq k \\ 1, & \tau[0,\aleph] \neq tell \\ 2, & \tau[0,\aleph] = tell \text{ and} \\ & \tau[i,\aleph] \neq help \text{ for all } 1 \leq i \leq k \end{cases}$$

Note that for $k = 2$, we get exactly the same behaviour as with simple temporal conditionals in the previous section.

5.2 Compiling Conditionals into BATs

The compilation method works on intentional conditionals as well. The only thing we have to ensure is that negation is only applied to program expressions from the guarded-action fragment. For this purpose, we introduce the two following rules in addition to ones stated in Definition 5:

8. $\mathfrak{C}[\overline{\Box_k \delta}, a] = \mathfrak{C}[\Diamond_k \overline{\delta}, a]$
9. $\mathfrak{C}[\overline{\Diamond_k \delta}, a] = \mathfrak{C}[\Box_k \overline{\delta}, a]$

In the non-negated case, the existing rules can be applied using (22) and (23). For the Chisholm example, we get for horizon $k = 3$, again after simplifications:

$$\begin{aligned}
&bad(a) = b \;\equiv \\
&\qquad b = 0 \wedge (Did(tell + tell; \star)) \wedge a = help \vee \\
&\qquad b = 1 \wedge (Did(\overline{tell} + \overline{tell}; \star) \vee \\
&\qquad b = 2 \wedge (Did(tell + tell; \star)) \wedge a \neq help
\end{aligned} \tag{27}$$

with the additional SSAs

$$\Box[a] Did(tell + tell; \star) \;\equiv\; a = tell \vee Did(tell) \tag{28}$$

$$\Box[a] Did(\overline{tell} + \overline{tell}; \star) \;\equiv\; a \neq tell \vee Did(\overline{tell}) \tag{29}$$

6 Discussion

The Chisholm paradox has received a great deal of attention in the literature. To name but a few, works in the early 1980s (van Eck 1982; Loewer and Belzer 1983) presented solution proposals based on temporal extensions of deontic logic, identifying the scenario's temporal nature as a vital aspect that SDL is unable to appropriately represent. Van der Torre and Tan (1998) argued that these were still insufficient for Chisholm's original set, as this requires conditionalization where the antecedent (going to help) refers to a later point in time than the consequent (telling). They go on to present a formalization based on *stit* ("see to it that") semantics (Horty 2001), modified to include a preference relation over histories. While these kinds of analyses yielded valuable theoretical insights into the nature of the problem, the proposed formalisms do not lend themselves well to practical implementations, e.g. due to the fact that actions in stit – other than planning languages or action formalisms such as the Situation Calculus – do not have proper names or types, but are described purely through their effects.

In this paper, we proposed a new formalism for reasoning about contrary-to-duty scenarios based on a modal variant of the Situation Calculus that allows to express postconditions for complex actions in the form of programs from the GOLOG agent language. By employing a special semantics based on linear-time traces rather than branching-time tree models, we could integrate non-trivial notions of action negation and joint execution of programs. We showed that the approach is more general than two existing ones due to Meyer (1988) and Claßen and Delgrande (2020), and presented a third, more expressive alternative involving a simple notion of intention.

This line of research is work in progress, and there are many avenues for future work. On the technical side, it could be argued that having a program semantics where the entire set of programs constitutes a Boolean algebra is desirable, so as to be able to apply all the "usual" laws to such expressions. It is conceivable to do this by adopting a definition more similar to the one of (Meyer 1988), albeit at the cost of being less simple, and less close to the original transition semantics of GOLOG.

Regarding expressivity, besides supporting a larger fragment of GOLOG programs in deontic conditionals, it would be interesting to explore more sophisticated notions of intentions to formulate constraints. While it is certainly possible to come up with infinitary versions of the temporal operators $\Box_k$ and $\Diamond_k$, in most cases it seems reasonable to apply a certain form of deadline. The latter may be of a temporal nature (e.g., the neighbour needs our help on the same day), and so for a more realistic representation we could incorporate an explicit, quantitative notion of time, instead of just crudely counting the number of actions. However, a more general approach could be to allow for arbitrary conditions as deadlines, for instance by adopting an approach due to Broersen (2004b) that uses operators $M(\rho \leq \delta)$ to express the motivation to achieve ρ before δ becomes true (e.g., have the neighbour's roof fixed before it starts raining).

Finally, it will be interesting to combine the notions of actions and obligations we considered here not only with intentions, but also beliefs, and study their interplay. This is similar to how BOID architectures (Broersen et al. 2001) have been proposed to generalize beliefs, desires, intention (Bratman 1987) by including obligations and norms.

Acknowledgements

We gratefully acknowledge financial support from the Natural Sciences and Engineering Research Council of Canada and thank the referees for their valuable comments.

References

Anderson, A. R. 1958. A reduction of deontic logic to alethic modal logic. *Mind* 67(265):100–103.

Bartha, P. 1999. Moral preference, contrary-to-duty obligation and defeasible oughts. *Norms, logics and information systems: new studies in deontic logic and computer science* 93–108.

Bratman, M. 1987. *Intentions, Plans, and Practical Reason*. Cambridge University Press.

Broersen, J. M.; Dastani, M.; Hulstijn, J.; Huang, Z.; and van der Torre, L. W. N. 2001. The BOID architecture: conflicts between beliefs, obligations, intentions and desires. In *Proc. AGENTS 2001*, 9–16. ACM Press.

Broersen, J. M. 2004a. Action negation and alternative reductions for dynamic deontic logics. *Journal of Applied Logic* 2(1):153–168.

Broersen, J. M. 2004b. On the logic of 'being motivated to achieve rho, before delta'. In *Proc. JELIA 2004*, 334–346. Springer.

Chellas, B. F. 1980. *Modal Logic: An Introduction*. Cambridge University Press.

Chisholm, R. M. 1963. Contrary-to-duty imperatives and deontic logic. *Analysis* 24(2):33–36.

Claßen, J., and Delgrande, J. 2020. Dyadic obligations over complex actions as deontic constraints in the situation calculus. In *Proc. KR 2020*, 253–263. ijcai.org.

Claßen, J., and Lakemeyer, G. 2008. A logic for non-terminating Golog programs. In *Proc. KR 2008*, 589–599. AAAI Press.

Claßen, J. 2013. *Planning and Verification in the Agent Language Golog*. Ph.D. Dissertation, Department of Computer Science, RWTH Aachen University.

De Giacomo, G.; Lespérance, Y.; and Levesque, H. J. 2000. ConGolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence* 121(1–2):109–169.

Gabbay, D.; Horty, J.; Parent, X.; van der Meyden, R.; and van der Torre, L. 2013. *Handbook of Deontic Logic and Normative Systems*. College Publications.

Hansson, B. 1969. An analysis of some deontic logics. *Noûs* 3(4):373–398.

Horty, J. F. 2001. *Agency and Deontic Logic*. Oxford University Press.

Kraus, S.; Lehmann, D. J.; and Magidor, M. 1990. Nonmonotonic reasoning, preferential models and cumulative logics. *Artificial Intelligence* 44(1-2):167–207.

Lakemeyer, G., and Levesque, H. J. 2010. A semantic characterization of a useful fragment of the situation calculus with knowledge. *Artificial Intelligence* 175(1):142–164.

Levesque, H. J.; Reiter, R.; Lespérance, Y.; Lin, F.; and Scherl, R. B. 1997. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming* 31(1–3):59–83.

Loewer, B., and Belzer, M. 1983. Dyadic deontic detachment. *Synthese* 54(2):295–318.

McCarthy, J., and Hayes, P. 1969. Some philosophical problems from the standpoint of artificial intelligence. In *Machine Intelligence 4*. New York: American Elsevier. 463–502.

Meyer, J.-J. C.; Dignum, F.; and Wieringa, R. J. 1994. The paradoxes of deontic logic revisited: A computer science perspective. Technical Report UU-CS-1994-38, Utrecht University.

Meyer, J. C.; Wieringa, R. J.; and Dignum, F. 1998. The role of deontic logic in the specification of information systems. In *Logics for Databases and Information Systems (Proceedings of the the Dagstuhl Seminar 9529: Role of Logics in Information Systems, 1995)*, 71–115. Kluwer Academic Publishers.

Meyer, J. C. 1988. A different approach to deontic logic: deontic logic viewed as a variant of dynamic logic. *Notre Dame Journal of Formal Logic* 29(1):109–136.

Reiter, R. 1991. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy* 359–380.

Reiter, R. 2001. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press.

van der Meyden, R. 1996. The dynamic logic of permission. *Journal of Logic and Computation* 6(3):465–479.

van der Torre, L. W. N., and Tan, Y. 1998. The temporal analysis of Chisholm's paradox. In *Proc. AAAI 1998*, 650–655. AAAI Press.

van Eck, J. A. 1982. A system of temporally relative modal and deontic predicate logic and its philosophical applications. *Logique et Analyse* 25(99):249–290.

von Wright, G. H. 1951. Deontic logic. *Mind* 60(237):1–15.

von Wright, G. H. 1963. *Norm and action: a logical enquiry*. Routledge and Kegan Paul.