Annals of Mathematics and Artificial Intelligence (2019) 86:231–255
https://doi.org/10.1007/s10472-019-09627-9

A paraconsistent approach to actions in informationally complex environments

Łukasz Białek[1] · Barbara Dunin-Kęplicz[1] · Andrzej Szałas[1,2]

Published online: 31 May 2019

Abstract
Contemporary systems situated in real-world open environments frequently have to cope with incomplete and inconsistent information that typically increases complexity of reasoning and decision processes. Realistic modeling of such informationally complex environments calls for nuanced tools. In particular, incomplete and inconsistent information should neither trivialize nor stop both reasoning or planning. The paper introduces ACTLOG, a rule-based four-valued language designed to specify actions in a paraconsistent and paracomplete manner. ACTLOG is an extension of $4QL^{Bel}$, a language for reasoning with paraconsistent belief bases. Each belief base stores multiple world representations. In this context, ACTLOG's action may be seen as a belief bases' transformer. In contrast to other approaches, ACTLOG actions can be executed even when the underlying belief base contents is inconsistent and/or partial. ACTLOG provides a nuanced action specification tools, allowing for subtle interplay among various forms of nonmonotonic, paraconsistent, paracomplete and doxastic reasoning methods applicable in informationally complex environments. Despite its rich modeling possibilities, it remains tractable. ACTLOG permits for composite actions by using sequential and parallel compositions as well as conditional specifications. The framework is illustrated on a decontamination case study known from the literature.

Keywords Action languages · Paraconsistent reasoning · Paracomplete reasoning · Doxastic reasoning · Belief bases

Supported by the Polish National Science Centre grant 2015/19/B/ST6/02589, the ELLIIT Network Organization for Information and Communication Technology, and the Swedish Foundation for Strategic Research FSR (SymbiKBot Project).

✉ Andrzej Szałas
andrzej.szalas@mimuw.edu.pl

Łukasz Białek
lukasz.bialek@mimuw.edu.pl

Barbara Dunin-Kęplicz
keplicz@mimuw.edu.pl

[1] Institute of Informatics, University of Warsaw, Warsaw, Poland

[2] Department of Computer and Information Science, Linköping University, Linköping, Sweden

Mathematics Subject Classification (2010) 03B42 · 03B53 · 68N17 · 68T27 · 97R40

1 Actions in informationally complex environments

Reasoning about actions and change is an essential ingredient of AI systems. Throughout the years a variety of advanced solutions has been introduced, developed, verified and used in this field. Despite a broad and intensive research (see, e.g., [40] and references there), the issue of inconsistent information has rarely been addressed in this context. However, in informationally complex environments, due to the heterogeneity of distributed information sources of diverse quality and credibility, *inconsistent* and *incomplete information* (further abbreviated in this paper by 3i) is a common phenomenon. Therefore, inconsistency and incompleteness tolerance lies at the heart of our approach. Essentially, this attitude is shared by many researchers who addressed related issues in other application areas. In particular, the importance of addressing inconsistencies in a robust manner is emphasized in [28] (see also [29]), where *inconsistency robustness* is phrased as:

> "information system performance in the face of continually pervasive inconsistencies – a shift from the previously dominant paradigms of inconsistency denial and inconsistency elimination attempting to sweep them under the rug."

For a related discussion see also [4] or an overview in [3] where the authors point out that:

> "inconsistency is useful in directing reasoning, and instigating the natural processes of argumentation, information seeking, multi-agent interaction, knowledge acquisition and refinement, adaptation, and learning."

The ultimate goal of our research is to develop a planning system that is rich enough to cope with $3i$. An important subgoal, which we address here, is to develop actions' specification language, adequate for reasoning and planning in informationally complex and sometimes defective environments. Planning as a key ingredient of intelligent systems, in particular multiagent systems, has been intensively developed, with its roots in early seventies of the previous century. The seminal planner, STRIPS, introduced already in 1971 [22], initiated the classical approach to automated planning, further developed by many followers. On the modern level, STRIPS was lacking means for dealing with inconsistencies and gaps in knowledge. That does not mean that these issues have been neglected in the field. Even though paraconsistent approaches have been proposed (see, e.g., [18, 47]), in majority of contemporary planners, inconsistent or missing knowledge is projected into the two-valued classical setting. Such a projection is typically performed with the use of nonmonotonic techniques or other heuristics. The key of our approach, which is also rooted in STRIPS, is to find a language that is expressive enough to explicitly deal with $3i$ in all phases of planning. This includes resolving inconsistencies whenever it is necessary. For this reason different context-sensitive strategies of resolving conflicts maybe applied or constructed. Notice, that solving this problem in general terms is not possible. Disambiguation methods are highly context, application-dependent and individualized. They include strategies, like:

- "killing inconsistencies at the root": to solve them as soon as possible;
- "living with inconsistency": to postpone disambiguation to the last possible moment (or even forever);
- solving inconsistency whenever relevant information appears.

When building applications dealing with pervasive information gaps and gluts, it is crucial to design knowledge completion and disambiguation in accordance with the recognized needs and the requirements of the application in question. Along these lines, action specification languages call for nuanced but possibly simple and uniform tools supporting a rich repertoire of related techniques.

Since the inception of knowledge representation and planning, beliefs have usually been modeled via various combinations of multi-modal logics [15, 20], nonmonotonic logics [40], probabilistic reasoning [54] or fuzzy reasoning [58], just to mention some of them. However, most of those approaches either lack tools for handling $3i$ or are too complex for real-world applications. This motivated a total shift in perspective, presented in [12, 13]. Rather than reasoning in modal logics or other complex formalisms, a tractable approach based on querying paraconsistent belief bases has been introduced there. It has further been developed in [5]. In order to achieve the required expressiveness and modeling convenience, next to truth and falsity two additional truth values: $\mathfrak{i}$ (*inconsistent*) and $\mathfrak{u}$ (*unknown*) have been employed.

In summary, we aim to define a formal language ActLog for specifying actions in informationally complex environments, enjoying the following features:

- *concise rule-based specification* of actions and their effects in the presence of $3i$;
- *flexibility* in evaluation of formulas in distributed paraconsistent belief bases;
- *tractability* of computing actions' preconditions and resulting belief bases;
- *practical expressiveness* meaning that all actions (and only such) with preconditions and effects computable in deterministic polynomial time can be specified in ActLog.

ActLog belongs to the 4QL family of four-valued, rule-based languages. It builds on $4QL^{Bel}$ [5], which, in turn, extends the 4QL rule language [34–36]. While 4QL already permits to flexibly resolve/disambiguate $3i$ at any level of reasoning, $4QL^{Bel}$ includes means for doxastic reasoning by specifying paraconsistent belief bases and referring to them in rules. Specifically, the paper continues a line of research initiated in [6] by:

- extending the ActLog language with composite actions, in particular providing a novel semantics for parallel composition;
- providing many new examples of actions' specifications;
- extending the tractability results which now apply to composite actions, too.

The paper is structured in the following manner. First, in Section 2, we recall the background formalism used in ActLog and including the "plain" four-valued logic $\mathcal{R}_4$, its doxastic extension $\mathcal{R}_4^{Bel}$ and the $4QL^{Bel}$ rule-language. Next, in Section 3, we introduce the ActLog action specification language with atomic actions (Section 3.1) and composite actions (Section 3.2) using sequential and parallel compositions and the conditional specification. Section 3.3 provides results concerning tractability of the approach. In Section 4 we provide a decontamination case study illustrating ActLog. Section 5 discusses related work. Finally, Section 6 concludes the paper.

2 The background formalism

Let us now introduce logical formalisms used in the paper: $\mathcal{R}_4$, $\mathcal{R}_4^{Bel}$ and $4QL^{Bel}$. We recall them in a structured manner, "layer by layer":

- Section 2.1 recalls $\mathcal{R}_4$, the basic logic with the first-order syntax and a four-valued semantics of propositional connectives, quantifiers, and an additional *inspection operator*;
- Section 2.2 recalls $\mathcal{R}_4^{\mathrm{Bel}}$, a four-valued doxastic extension of $\mathcal{R}_4$ introducing operators for reasoning with beliefs and belief bases;
- Section 2.3 recalls $4\mathrm{QL}^{\mathrm{Bel}}$, a four-valued rule-based language, based on $\mathcal{R}_4^{\mathrm{Bel}}$ and providing a tractable reasoning engine.

2.1 The basic four-valued logic

The four-valued logic $\mathcal{R}_4$ has originally been introduced in [37] (see also [34, 36, 50, 55]). Below we present its main features and motivations behind its design choices.

2.1.1 Syntax of $\mathcal{R}_4$

The syntax of $\mathcal{R}_4$ is an extension of the syntax of classical first-order logic (see Table 1), where we assume the set of truth values $\mathfrak{t}$ (true), $\mathfrak{i}$ (inconsistent), $\mathfrak{u}$ (unknown), $\mathfrak{f}$ (false), constants *Const*, variables *Var* and relation symbols *Rel*. By convention, constant and relation symbols are denoted by names beginning with a small letter and variables beginning with a capital letter. Note that the only non-classical formulas, listed in the last line of Table 1, involve an *inspection operator* '$\in$'. Intuitively, the formula $\alpha \in T$ is $\mathfrak{t}$ when the truth value of α is in the set of truth values T.

An occurrence of variable X in a formula α is called *bound* if it occurs in the scope of a quantifier $\forall X$ or $\exists X$. An occurrence of a variable is *free* in formula α if it is not bound in α. A *literal* is an '$\langle AtomicFormula \rangle$' or '$\neg\langle AtomicFormula \rangle$'. A *ground literal* is a literal not containing variables. A *ground formula* is a formula without free variables.

2.1.2 Semantics of $\mathcal{R}_4$

The logic $\mathcal{R}_4$ uses four truth values, two classical values: $\mathfrak{t}$, $\mathfrak{f}$, and two non-classical ones: $\mathfrak{i}$ and $\mathfrak{u}$. The values $\mathfrak{i}$ and $\mathfrak{u}$, are introduced to indicate:

- the presence of inconsistent evidence supporting both truth and falsity of the formula;
- the lack of information needed to assign a truth value to the formula.

Let us start with the semantics of negation, applied to truth values:

$$\neg\mathfrak{t} \overset{\mathrm{def}}{=} \mathfrak{f},\ \ \neg\mathfrak{f} \overset{\mathrm{def}}{=} \mathfrak{t},\ \ \neg\mathfrak{i} \overset{\mathrm{def}}{=} \mathfrak{i},\ \ \neg\mathfrak{u} \overset{\mathrm{def}}{=} \mathfrak{u}. \tag{1}$$

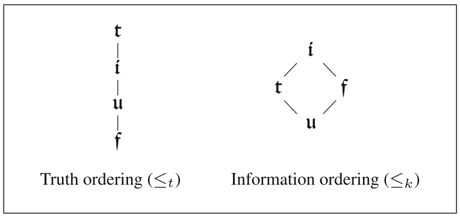
The negation behaves classically on the classical truth values. For non-classical ones, it behaves like traditional negation in three- and four-valued logics:

- the value $\mathfrak{u}$ of a formula α indicates that the actual truth value of α is unknown, so the value of $\neg\alpha$ is unknown, too;

Table 1 Syntax of $\mathcal{R}_4$

$\langle AtomicFormula \rangle$	::=	$r(u_1, \ldots, u_k)$, where $r \in Rel$, $u_1, \ldots, u_k \in Const \cup Var$;
$\langle Formula \rangle$	::=	$\langle AtomicFormula \rangle \mid \neg \langle Formula \rangle \mid$
		$\langle Formula \rangle \wedge \langle Formula \rangle \mid \langle Formula \rangle \vee \langle Formula \rangle \mid$
		$\langle Formula \rangle \rightarrow \langle Formula \rangle \mid$
		$\forall X \langle Formula \rangle \mid \exists X \langle Formula \rangle \mid$
		$\langle Formula \rangle \in T$, where $\emptyset \neq T \subseteq \{\mathfrak{t}, \mathfrak{f}, \mathfrak{i}, \mathfrak{u}\}$.

Fig. 1 Orderings on truth values



– the value $\mathfrak{i}$ of a formula α indicates that the actual truth value of α is claimed to be both true and false at the same time, so consequently $\neg\alpha$ is claimed false and true at the same time, being inconsistent, too.

By convention, we remove double negations: $\neg\neg\alpha$ is always identified with α.

Basic semantical structures we consider are finite sets of ground literals, further called $3i$*-worlds*. Each such a set provides a specific, possibly inconsistent set of facts about a given reality. More complex semantical structures used in this paper are *belief bases*, being finite sets of $3i$-worlds representing complementary or alternative views on a given reality. We shall discuss them in Section 2.3. Since the full language, involving beliefs, is evaluated in belief bases, in order to keep the presentation uniform, every $3i$-world w is identified with a one-element belief base $\{w\}$.[1]

The semantics of $\mathcal{R}_4$ formulas uses $\{w\}$ and an *assignment* $v : \mathit{Var} \longrightarrow \mathit{Const}$ assigning domain values to variables. If ℓ is a literal, by $\ell(v)$ we mean the ground literal obtained from ℓ by substituting each variable x occurring in ℓ by $v(x)$.

Definition 1 The *truth value* of a literal ℓ wrt an assignment v and a singleton belief base $\{w\}$, denoted by $\ell(w, v)$, is defined as follows:

$$\ell(w,v) \stackrel{\text{def}}{=} \begin{cases} \mathfrak{t} \ \text{ if } \ell(v) \in w \text{ and } (\neg\ell(v)) \notin w; \\ \mathfrak{i} \ \text{ if } \ell(v) \in w \text{ and } (\neg\ell(v)) \in w; \\ \mathfrak{u} \ \text{ if } \ell(v) \notin w \text{ and } (\neg\ell(v)) \notin w; \\ \mathfrak{f} \ \text{ if } \ell(v) \notin w \text{ and } (\neg\ell(v)) \in w. \end{cases}$$

Example 1 Consider the following $3i$-world:

$$w = \{\mathit{safe}(r_1), \neg\mathit{safe}(r_2), \mathit{safe}(r_3), \neg\mathit{safe}(r_3)\} \tag{2}$$

In w, the truth value of $\mathit{safe}(r_1)$ is $\mathfrak{t}$, of $\mathit{safe}(r_2)$ is $\mathfrak{f}$, of $\mathit{safe}(r_3)$ is $\mathfrak{i}$, and of $\mathit{safe}(r_4)$ (not present in w) is $\mathfrak{u}$.

The semantics of our framework is based on two different orderings, shown in Fig. 1. The first ordering, $\leq_t$, called the *truth ordering*, is used to evaluate formulas of $\mathcal{R}_4$ and the second one, $\leq_k$, the *information ordering*, is used to provide meaning to the belief operator.

The truth ordering reflects the truth level of a formula α:

– the value $\mathfrak{t}$ indicates that the available evidence pieces support the truth of α and no contradicting evidence weakens this support;
– the value $\mathfrak{i}$ contains "less true" than $\mathfrak{t}$: though it indicates a support for the truth of α, the support is weakened by a contradicting evidence;

[1]To simplify the notation we sometimes omit brackets {} and write w rather than $\{w\}$.

- the value $\mathfrak{u}$ contains "less true" than $\mathfrak{i}$: it indicates no evidence for the truth of α;
- the value $\mathfrak{f}$ contains "less true" than $\mathfrak{u}$: it not only lacks a support for the truth of α but, moreover, expresses the contrary.

The information ordering, when analyzed bottom up, reflects the process of fusing beliefs about (ground) formula α:

- initially no pieces of evidence are gathered, so there is no support for the truth nor for the falsity of α, so the status of α is unknown, represented by the value $\mathfrak{u}$;
- next, in the course of evidence acquisition, one may obtain pieces supporting only the truth of α or only its falsity (assigning to α the truth value $\mathfrak{t}$ or $\mathfrak{f}$, respectively);
- finally, if evidence for both: truth and falsity of α are gathered, its truth value becomes $\mathfrak{i}$.

Remark 1 The use of two different orderings on truth values is rather typical in application areas we deal with. A known framework for such orderings is based on the use of *bilattices* introduced by [25] (see also [23, 24, 26]). However, the linear truth ordering with negation defined by (1) does not fit the bilattices framework. Specifically, the requirement:

$$t_1 \leq_t t_2 \text{ implies} \neg t_2 \leq_t \neg t_1$$

is violated for $t_1 = \mathfrak{u}$ and $t_2 = \mathfrak{i}$.

The semantics of propositional connectives, quantifiers and inspection expressions is given in Table 2. Traditionally, the semantics of disjunction is given by the maximum and of conjunction – by the minimum wrt truth ordering.

Implication is defined classically:

$$\alpha \rightarrow \beta \stackrel{\text{def}}{=} \neg\alpha \vee \beta. \tag{3}$$

The universal quantifier generalizes conjunction and the existential quantifier generalizes disjunction.

Remark 2 The connectives $\neg, \wedge, \vee, \rightarrow$ behave classically on classical truth values $\mathfrak{t}$, $\mathfrak{f}$. When the set of truth values is restricted to $\mathfrak{t}$, $\mathfrak{f}$,$\mathfrak{u}$ or to $\mathfrak{t}$,$\mathfrak{f}$,$\mathfrak{i}$, the resulting logic is the well-known three-valued logic of Kleene, where the non-classical truth value represents

Table 2 The semantics of $\mathcal{R}_4$ formulas, where $\Delta = \{w\}$, v is an assignment of constants to variables, min, max are respectively minimum and maximum wrt truth ordering and $\alpha(X/a)$ denotes the formula obtained from α by substituting all free occurrences of variable X by constant a

- If α is a literal then $\alpha(\Delta, v)$ is defined in Definition 1;
- $(\neg\alpha)(\Delta, v) \stackrel{\text{def}}{=} \neg(\alpha(\Delta, v))$;
- $(\alpha \wedge \beta)(\Delta, v) \stackrel{\text{def}}{=} \min\{\alpha(\Delta, v), \beta(\Delta, v)\}$;
- $(\alpha \vee \beta)(\Delta, v) \stackrel{\text{def}}{=} \max\{\alpha(\Delta, v), \beta(\Delta, v)\}$;
- $(\alpha \rightarrow \beta)(\Delta, v) \stackrel{\text{def}}{=} \max\{\neg\alpha(\Delta, v), \beta(\Delta, v)\}$;
- $(\forall X\alpha(X))(\Delta, v) \stackrel{\text{def}}{=} \min\limits_{a \in Const} \{(\alpha(X/a)(\Delta, v)\}$;
- $(\exists X\alpha(X))(\Delta, v) \stackrel{\text{def}}{=} \max\limits_{a \in Const} \{(\alpha(X/a)(\Delta, v)\}$;
- $(\alpha \in T)(\Delta, v) \stackrel{\text{def}}{=} \begin{cases} \mathfrak{t} \text{ when } \alpha(L, v) \in T & \text{(here '}\in\text{' denotes set membership),} \\ \mathfrak{f} \text{ otherwise.} \end{cases}$

indefinitness, which is commonly accepted in modeling lack of knowledge (respectively, inconsistency) in the three-valued approaches.

Remark 3 One of commonly considered four-valued logics is the logic of Belnap [2]. It's information ordering is the $\leq_k$, while the truth ordering is $\leq_k$ "rotated right", with $\mathfrak{f}$ being its bottom, $\mathfrak{t}$ – its top and $\mathfrak{u}$, $\mathfrak{i}$ being its intermediate, incomparable elements. Thus, in particular, in Belnap's logic $\mathfrak{i} \vee \mathfrak{u} = \mathfrak{t}$ and $\mathfrak{i} \wedge \mathfrak{u} = \mathfrak{f}$ which violates intuitions we want to preserve:

– the disjunction should only be true when at least one of its disjuncts is true;
– the conjunction should only be false when at least one of its conjuncts is false.

For example, assume there are two paths, p_1 and p_2, between some given places. If a robot has inconsistent information whether p_1 is passable and no information whether p_2 is passable, in Belnap's logic we obtain:

$$passable(p_1) \vee passable(p_2) = \mathfrak{t}; \tag{4}$$

$$passable(p_1) \wedge passable(p_2) = \mathfrak{f}. \tag{5}$$

Both outcomes are questionable. In $\mathcal{R}_4$ the value of the disjunction (4) is $\mathfrak{i}$, and the value of the conjunction (5) is $\mathfrak{u}$.

2.2 Doxastic extension of $\mathcal{R}_4$

In this section we recall the approach of [5, 14] in a possibly compact, yet comprehensive manner.[2] In particular, we show an extension of $\mathcal{R}_4$ by the two operators for expressing beliefs. This extension is further denoted by $\mathcal{R}_4^{\mathrm{Bel}}$.

2.2.1 Syntax of $\mathcal{R}_4^{\mathrm{Bel}}$

Let us first introduce belief bases.

Definition 2 By a *belief base over* a set of constants *Const* we understand any finite set Δ consisting of $3i$-worlds over *Const*.

Recall that each $3i$-world in a belief base represents a possibly incomplete and/or inconsistent view of the world. For example, a belief base can consist of three $3i$-worlds: the first one containing facts based on measurements received from a ground robot's sensor platform, the second one containing facts extracted from a drone's camera video stream and the third one representing views provided by ground operators.

As regards belief operators, the syntax of $\mathcal{R}_4^{\mathrm{Bel}}$ is given in Table 3. It extends the syntax given in Table 1 by clauses for the following operators:

– $\mathrm{Bel}_\Delta(\alpha)$, expressing beliefs related to belief bases (indicated by Δ);
– $\varphi(\Delta).(\alpha)$, allowing one to evaluate Bel()-free formulas in belief bases: here φ is a mapping transforming a belief base into a single $3i$-world. In general, φ occurring in

[2]For a detailed description of belief bases and belief structures, see [5, 12–14]. Belief bases in $4\mathrm{QL}^{\mathrm{Bel}}$ have been presented in [5].

Table 3 Syntax of $\mathcal{R}_4^{\text{Bel}}$, where Δ is a belief base and φ is a mapping transforming a belief base into a (single) $3i$-world; if φ is not specified explicitly, it is by default assumed to be $\varphi(\Delta) \overset{\text{def}}{=} \bigcup \Delta$, i.e., $\Delta\alpha \overset{\text{def}}{=} (\bigcup \Delta)\alpha$

$$\langle Formula \rangle \quad ::= \quad \varphi(\Delta).(\langle Formula \rangle) \mid \mathrm{Bel}_{\Delta}\big(\langle Formula \rangle\big)$$

$\varphi(\Delta).()$ is an arbitrary (but tractable) belief fusion method, intended as a means to combine information included in $3i$-worlds of Δ. For example, $\varphi(\Delta)$ may be $\bigcup\limits_{D\in\Delta} D$ or $\bigcap\limits_{D\in\Delta} D$ (further abbreviated by $\bigcup \Delta$ and $\bigcap \Delta$, respectively).

2.2.2 Semantics of $\mathcal{R}_4^{\text{Bel}}$

In Table 4 we extend the semantics of $\mathcal{R}_4$ to all formulas of $\mathcal{R}_4^{\text{Bel}}$. Note that for nested Bel() operators, one starts evaluation with the innermost one.

Example 2 Let a belief base Δ consists of two $3i$-worlds:

$$\{safe(r_1), \neg safe(r_2), safe(r_3), \neg safe(r_3)\},\ \{safe(r_2), \neg safe(r_4)\}. \tag{6}$$

Then, for example,

- $\mathrm{Bel}_{\Delta}(safe(r_1))$ is $\mathfrak{t}$, $\mathrm{Bel}_{\Delta}(safe(r_2))$ as well as $\mathrm{Bel}_{\Delta}(safe(r_3))$ are $\mathfrak{i}$, $\mathrm{Bel}_{\Delta}(safe(r_4))$ is $\mathfrak{f}$, and $\mathrm{Bel}_{\Delta}(safe(r_2) \vee \neg safe(r_2))$ is $\mathfrak{t}$;
- for $i = 1, \ldots, 4$, $\Delta.(safe(r_i))$ is as above, but $\Delta.(safe(r_2) \vee \neg safe(r_2))$ is $\mathfrak{i}$.

2.3 Representing and querying belief bases

The logic $\mathcal{R}_4^{\text{Bel}}$ offers means for general paraconsistent and paracomplete reasoning about beliefs. Recall that we aim to develop a tractable framework for action specification. Therefore we need a suitable language to represent and query belief bases. As shown in [5], a suitable candidate is the 4QL$^{\text{Bel}}$ language.[3] 4QL$^{\text{Bel}}$ is an extension of 4QL. Though the full definition of 4QL$^{\text{Bel}}$ is available in [5], for clarity we recall the most important constructs of the language. The language inherits a fair amount of elements from the 4QL language [34, 36, 50], including basic program syntax and semantics. The 4QL$^{\text{Bel}}$ program consists of modules, structured as shown in Module 1. Sections **domains** and **relations** are used to specify domains of arguments and signatures of relations used in rules.

Module 1 Syntax of 4QL$^{\text{Bel}}$ modules.

```
module moduleName:
    domains:
    relations:
    rules:
    facts:
end.
```

[3] An open source experimental interpreter of 4QL$^{\text{Bel}}$ is available via `4ql.org`.

Table 4 Semantics of $\mathcal{R}_4^{\mathrm{Bel}}$, where Δ is a belief base, v assigns constants to variables, α is a formula without Bel() operators, LUB denotes the least upper bound wrt the information ordering (see Fig. 1)

- $\alpha(\Delta, v) \overset{\text{def}}{=} \alpha(\bigcup \Delta, v)$;
- $(\mathrm{Bel}_{\Delta}(t))(v) \overset{\text{def}}{=} t$, for $t \in \{\mathfrak{t}, \mathfrak{f}, \mathfrak{i}, \mathfrak{u}\}$;
- $\big(\mathrm{Bel}_{\Delta}(\alpha)\big)(v) \overset{\text{def}}{=} \textsc{Lub}\{\alpha(\{w\}, v) \mid w \in \Delta\}$;
- $(\varphi(\Delta).(\alpha))(v) \overset{\text{def}}{=} \alpha(\{\varphi(\Delta)\}, v)$.

4QL$^{\mathrm{Bel}}$ rules, specified in the section **rules** have the following form, where $\langle Formula \rangle$ is an arbitrary formula of the logic presented in Section 2.2

$$\langle Literal \rangle : -\langle Formula \rangle. \tag{7}$$

Facts, specified in the **facts** section, are rules with the empty $\langle Formula \rangle$ part (being $\mathfrak{t}$). In such cases we simply write $\langle Literal \rangle$.

A *model* w of a 4QL$^{\mathrm{Bel}}$ module m is a $3i$-world, not necessarily minimal, such that for every rule (thus also every fact) of the form (7) in m and every assignment v of constants appearing in m to free variables of m,

- whenever $(\langle Formula \rangle)(\{w\}, v) = \mathfrak{t}$, the conclusion $v(\langle Literal \rangle)$ is in w, and
- whenever $v(\langle Formula \rangle) = \mathfrak{i}$, the conclusion $v(\langle Literal \rangle)$ as well as its negation $\neg v(\langle Literal \rangle)$ are in w.

The above conditions reflect a generalization of the Shepherdson's implication [46].

The semantics of a module is given by its *well-supported* model. Intuitively, a model of m is *well-supported* when all literals of m are justified by reasoning starting from facts of m and using rules of m.[4] Importantly, for every 4QL module a well-supported model exists and is uniquely determined. Therefore, each 4QL$^{\mathrm{Bel}}$ module can be identified with a $3i$-world. That way:

$$\text{4QL}^{\text{Bel}} \text{ modules have a very important role as a tool for concise and uniform specification of } 3i - \text{worlds.} \tag{8}$$

A 4QL$^{\mathrm{Bel}}$ *program* is a finite set of 4QL$^{\mathrm{Bel}}$ modules. Its semantics is given by a set of well-supported models of its modules.

One can query modules using traditional remote calls' notation: $m.\alpha$, where m is a module name and α is a formula. The meaning of $m.\alpha$ is the (four-valued) relation defined as the answer to the query expressed by α, evaluated in m.[5]

Belief bases, as defined in the current paper, are specified in 4QL$^{\mathrm{Bel}}$ as in Belief Base 2.

Belief Base 2 Syntax of belief bases.

```
beliefs beliefBaseName:
    worlds:
        // list of 4QL^Bel module names specifying 3i-worlds
end.
```

[4]Well-supportedness does not entail minimality. This is an intended feature of our approach since in many contexts minimality is not desired [11, 21, 34, 44, 49].

[5]Acyclicity of references among modules is required (needed for tractability of computing queries).

As shown in [5], computing well-supported models contained in belief bases as well as querying them using 4QL$^{\mathrm{Bel}}$ formulas can be done in deterministic polynomial time wrt the number of constants occurring in the belief base.

3 The ActLog language

Let us now extend 4QL$^{\mathrm{Bel}}$ towards specifying actions. Our approach reflects the general idea of action definition. As a novelty, an ActLog action is a belief bases transformer: a state of the environment, expressed as a belief base, is transformed by an action into the resulting belief base. Next, the use of 4QL$^{\mathrm{Bel}}$ to represent actions' effects ensures their concise representation which is one of our important goals. Finally, due to tractability results for 4QL$^{\mathrm{Bel}}$ [5], effects of actions can be computed in a tractable manner.

All back-end operations like reasoning management is handled by 4QL$^{\mathrm{Bel}}$.

3.1 Atomic actions

Let us start from defining actions' specification. The syntax is presented as Action 3, where:

- `act` is the action name and $\bar{x}$ are its parameters;
- $\alpha(\bar{x})$ is an arbitrary formula of 4QL$^{\mathrm{Bel}}$, called the *precondition* of action `act`;
- $\beta^{+}(\bar{x})$, $\beta^{-}(\bar{x})$ are 4QL$^{\mathrm{Bel}}$ rules, representing effects of action `act` by specifying sets of literals to be added ($\beta^{+}(\bar{x})$) and to be removed ($\beta^{-}(\bar{x})$);
- it is assumed that α, β^{+} and β^{-} contain no free variables other than those in $\bar{x}$.

By an *instance* of action `act`$(\bar{x})$ we mean `act`$(\bar{a})$, where $\bar{a}$ is a tuple of constants.

Action 3 Syntax of actions in ActLog.

```
action act($\bar{x}$):
    preconditions:
        $\alpha(\bar{x})$
    postconditions:
        add:
            $\beta^{+}(\bar{x})$
        remove:
            $\beta^{-}(\bar{x})$
end.
```

Recall that one of our goals is to achieve concise specifications of actions' pre- and postconditions. The following example illustrates how this feature is achieved in ActLog'.

Example 3 Assume that the following properties of action `move(ID,X,Y)` are to be expressed, where 'ID' is a robot's identifier, 'safe-path(X,Y)' indicates whether the path from 'X' to 'Y' is safe, and 'in(ID,X)' states that the robot 'ID' is in the place 'X':

1. **when** 'safe-path(X,Y)' is true, and 'in(ID,X)', 'X$\neq$Y' are true
   then `move(ID,X,Y)` results in a state where '¬in(ID,X)' and 'in(ID,Y)' are true;
2. **when** 'safe-path(X,Y)' is inconsistent, and 'in(ID,X)', 'X$\neq$Y' are true
   then `move(X,Y)` results in a state where '¬in(ID,X)' is true
   and 'in(ID,Y)' is inconsistent;

3. **when** ‘safe-path(X,Y)’ is unknown, and ‘in(ID,X)’, ‘X≠Y’ are true
   then `move(X,Y)` results in a state where ‘¬in(ID,X)’ is true
   and ‘in(ID,Y)’ is unknown.

Action 4 provides a concise specification of points 1–3 in ActLog.

Action 4 A concise specification of points 1–3 in ActLog.

```
action move(ID,X,Y):
    preconditions:
        safe-path(X,Y) $\in \{\mathfrak{t}, \mathfrak{i}, \mathfrak{u}\} \wedge$ in(ID,X)=$\mathfrak{t}$ $\wedge$ X≠Y
    postconditions:
        add:
            in(ID, Y) :– safe-path(X,Y).
            ¬in(ID,X).
        remove:
            in(ID, X).
end.
```

It is also important to notice that rules in action specification may use operators like, e.g., $\text{Bel}_{\Delta}()$, referring to belief bases. This allows one to deal with distributed belief bases. In this paper belief bases are known from the context, so we sometimes omit the subscript indicating a belief base.

Let us now define the ActLog’s semantics formally.

Definition 3 Tuples $\langle a_1, \ldots, a_k \rangle$, $\langle b_1, \ldots, b_l \rangle$ consisting of variables and/or constants are called *compatible* if $k = l$ and, for $i = 1, \ldots, k$, at least one of a_i, b_i is a variable or both $a_i, b_i \in Const$ and $a_i = b_i$.

Given a $3i$-world w, specification expressed as Action 3 and a tuple of constants $\bar{a}$ compatible with $\bar{x}$, the action $\texttt{act}(\bar{a})$ is *executable* on w when its precondition $\alpha(\{w\}, v) = \mathfrak{t}$, where v assigns constants $\bar{a}$ to variables $\bar{v}$, respectively.

An action is *executable* on a belief base Δ, if it is executable on at least one $w \in \Delta$.

Remark 4 Note that in preconditions of actions (formula α of Action 3) one can use any formula of the form defined in Tables 1–3, in particular involving the Bel() operator as well as the operator ‘$\in T$’, permitting to react to inconsistency and lack of knowledge. Therefore an action can be executed when the state is inconsistent and/or some/all literals are unknown. Running actions in such circumstances is a unique feature of ActLog.

When action $\texttt{act}(\bar{a})$ is executed, it transforms its input belief base Δ into the resulting belief base Δ' as shown in Algorithm 5, where Δ' represents *effects of* action $\texttt{act}(\bar{a})$ on Δ. The algorithm iterates through the $3i$-worlds in Δ. Recall that $3i$-worlds in Δ represent different views on the world. If the considered action is executable on a given $3i$-world $w \in \Delta$ then the contents of w is considered to be actual, so is added to both β^+ and β^- and literals to be added (respectively, removed) are computed and added to (respectively subtracted from) w and the resulting world is added to Δ'. If the action is not executable on w, the $3i$-world w itself is not affected by the action, so is added to Δ' unchanged.

Algorithm 5 Computing effects of actions.

```
Input:
  – action $\mathtt{act}(\bar{a})$, specified as Action 3, where $\bar{a}$ is a tuple of constants;
  – belief base $\Delta$;
Output: belief base $\Delta'$ representing effects of executing $\mathtt{act}(\bar{a})$ on $\Delta$;
1  set $\Delta' = \emptyset$
2  foreach $w \in \Delta$ do
3      if action $\mathtt{act}(\bar{a})$ is executable on $w$ then
4          set $u^+ = \emptyset$; set $u^- = \emptyset$;
5          compute the well-supported model of $\beta^+(\bar{a}) \cup w$ adding to $u^+$
             each literal obtained as a conclusion of a rule of $\beta^+(\bar{a})$;
6          compute the well-supported model of $\beta^-(\bar{a}) \cup w$ adding to $u^-$
             each literal obtained as a conclusion of a rule of $\beta^-(\bar{a})$;
7          add the set $\big((w \cup u^+) \setminus u^-\big)$ to $\Delta'$
8      else
9          add the set $w$ to $\Delta'$
10     end
11 end
```

Remark 5 Notice that u^+ and u^- computed in Algorithm 5 contain conclusions of rules (thus facts, too) specified in the action. These conclusions can be (and typically are) computed taking the contents of the underlying belief base into account. However, the contents of the belief base should not be automatically "imported" as action effects. If this was allowed, it would be difficult control actions' specifications. For example, belief bases may contain literals involving relations unknown for the actions' designer. Such literals could become part of actions' effects even though the actions do not affect them.

The following example illustrates the use of Algorithm 5.

Example 4 Let `move` be the action specified as Action 4 in Example 6 and let the input belief base be:

$$\Delta = \{\{\text{safe-path(a,b)}, \neg\text{safe-path(a,b)}, \neg\text{in(rob,a)}\} \tag{9}$$
$$\{\neg\ \text{safe-path(a,b)}, \text{in(rob,a)}\}\}\,. \tag{10}$$

After executing the action `move(rob,a,b)` we obtain:

$$\Delta' = \{\{\text{safe-path(a,b)}, \neg\text{safe-path(a,b)}, \neg\ \text{in(rob,a)}, \text{in(rob,b)}, \neg\text{in(rob,b)}\} \tag{11}$$
$$\{\neg\ \text{safe-path(a,b)}, \text{in(rob,a)}\}\}\,. \tag{12}$$

Since 'safe-path(a,b)' is inconsistent in the $3i$-world (9), rule in Line 6 of Action 4 makes 'in(rob,b)' inconsistent in (11). The action is not executable on (10), so this world is added to Δ' without any changes (as the world (12)).

3.2 Composite actions

Composite actions' specifications are important in applications. Apart from simplifying actions' specification, they can allow for more efficient plan building. Namely, their use as templates frequently occurring in a given application can substantially reduce the branching factor when searching for plans by avoiding explorations of useless branches. For example the sequence 'locate-lift-move', consisting of three atomic actions, can be used in planning without the necessity to construct this sequence during the planing process. For further discussion of performance gains see Remark 6 (page 250).

For simplicity, we concentrate on sequential and parallel compositions, and if-then-else operator only. First, these operations do not increase the number of $3i$-worlds within belief bases. Second, their use does not violate tractability of the approach.

Composite actions are specified as shown in Action 6, where γ is an expression consisting of atomic actions (with parameters), built using ';' (sequential composition), $\Rightarrow$ (conditional 'if-then-else') and '||'(parallel composition).

Action 6 Syntax of composite actions in ACTLOG.

```
action act ($\bar{x}$):
    composite:
        $\gamma(\bar{x})$
end.
```

The syntax of composite actions' expressions (γ in Action 6) is given in Table 5.

We assume that arguments of actions in γ belong to arguments $\bar{x}$ of the action act and we disallow recursion. To formally define this requirement, for a set of actions' specifications consider a *reference graph* $\langle V, E\rangle$ where V is a set of nodes labeled by action names and $(n_1, n_2) \in E$ iff n_2 occurs in n_1's **composite** section. In ACTLOG we only allow actions' specifications whose reference graph is acyclic.

Operators ';', '$\Rightarrow$' and '||' transform belief bases into belief bases. Given a belief base Δ, and action act, by act(Δ) we denote the belief base representing effects of act. While the semantics of ';' and '$\Rightarrow$' is rather standard, let us explain our approach to '||'. When there are no conflicts between actions act$_1$ and act$_2$, their parallel composition act$_1$||act$_2$ simply adds to $3i$-worlds literals determined by act$_1$ or by act$_2$ and removes literals determined by act$_1$ or by act$_2$. However, in the case of conflict (e.g., act$_1$ attempts to add a literal ℓ and at the same time act$_2$ attempts to remove it), we solve it by assuming that ℓ is inconsistent in the resulting $3i$-world. Of course, using 4QL^{Bel} one can later disambiguate such conflicts, e.g., taking into account the relative strengths of actions (if known).

Before providing formal semantics, let us illustrate the intended meaning of the introduced operators.

Example 5 Let the actions of pouring water and lighting fire are given as Actions 7-8, where the parameter 'O' indicates the object subject to the actions.

Table 5 Syntax of composite actions' expressions, where $\langle AtomicAction\rangle$ represents atomic actions, as defined in Section 3.1, referenced by their names and arguments

$$\begin{array}{lcl} \langle Composite\rangle & ::= & \langle AtomicAction\rangle \mid \langle Composite\rangle\,;\langle Composite\rangle \mid \\ & & \langle Formula\rangle \Rightarrow \langle Composite\rangle \,/\, \langle Composite\rangle \mid \\ & & \langle Composite\rangle \,||\, \langle Composite\rangle \end{array}$$

Action 7 The action of pouring water.

```
action pour-water(O):
    postconditions:
        add:
            wet(O).
            ¬flammable(O).
        remove:
            flammable(X) :– wet(X).
            ¬wet(O).
            on-fire(O).
end.
```

Action 8 The action of lighting fire.

```
action light-fire(O):
    postconditions:
        add:
            on-fire(O) :– flammable(O).
            ¬wet(X) :– on-fire(X).
        remove:
            ¬on-fire(O).
            ¬flammable(X) :– on-fire(X).
end.
```

Consider a belief base consisting of a single $3i$-world w = {flammable(o1), ¬ wet(o1)}.

1. Action `a1=(pour-water;light-fire)(o1)`, run on w, starts with `pour-water(o1)`, transforming w into w'={wet(o1), ¬ flammable(o1)}. The action `light-fire(o1)`,executed next, does not change w' so w' remains the result of `a1`.
2. Action `a2=(light-fire;pour-water)(o1)`, run on w, starts with `light-fire(o1)`, transforming w into w''={flammable(o1), on-fire(o1), ¬wet(o1)}. The next action, `pour-water(o1)`, executed on w'' again returns w'.
3. `a3=flammable(X)⇒light-fire(X)/pour-water(X)(o1)` results in w'' since its condition, `flammable(o1)` is t in w;
4. Action `a4=(pour-water || light-fire)(o1)`, run on w, executes both `pour-water(o1)` and `light-fire(o1)` at the same time. Table 6 contains u^+, u^- computed by Algorithm 5 for these actions.

   Actions `pour-water(o1)` and `light-fire(o1)` have conflicting effects on literals:

Table 6 Effects of `pour-water(o1)` and `light-fire(o1)`

	u^+	u^-
`pour-water(o1)`	wet(o1), ¬ flammable(o1)	flammable(o1), ¬ wet(o1), on-fire(o1)
`light-fire(o1)`	on-fire(o1), ¬ wet(o1)	¬ on-fire(o1), ¬ flammable(o1)

- ¬flammable(o1): added by `pour-water(o1)` and removed by `light-fire(o1)`;
- on-fire(o1), ¬wet(o1): removed by `pour-water(o1)` and added by `light-fire(o1)`.

The inconsistent effects are reflected by inconsistency of corresponding literals. The resulting world will then consist of literals: flammable(o1), ¬ flammable(o1), on-fire(o1), ¬ on-fire(o1), wet(o1), ¬ wet(o1).

Note that in parallel composition $\texttt{act}_1||\texttt{act}_2$ both actions are executed when their preconditions are both true. If this is not the case, one or none of $\texttt{act}_1$, $\texttt{act}_2$ is executed. To make sure that both actions are actually executed, one can use conditional specification with the condition being the conjunction of preconditions of $\texttt{act}_1$ and preconditions of $\texttt{act}_2$.

The semantics of action instances is given in Table 7.

3.3 Tractability of the approach

For any ActLog specification of an action $\texttt{act}(\bar{x})$, by $\#D$ we denote the sum of sizes of all domains in the specification, $\#L$ stands for the sum of lengths of composite actions' specifications and by $\#M$ we denote the number of 4QL^{Bel} modules occurring in the specification. For belief base Δ, by $\#\Delta$ we denote the number of all literals appearing in Δ. Note that $\#\Delta$ is polynomial in the size of $\#D$ (the size of relations is constant). In real-world applications, $\#M$ as well as $\#D$ are manageable by the hardware/database systems used, so is $\#\Delta$.

The following theorems can be proved similarly to analogous results for 4QL [34–36] and 4QL^{Bel} [5].

Theorem 1 *Let Δ be a belief base. For every* ActLog *specification of action $\texttt{act}(\bar{x})$ and a tuple of constants $\bar{a}$, compatible with $\bar{x}$, the preconditions and effects of $\texttt{act}(\bar{a})$ can be computed in deterministic polynomial time in* $\max\{\#D, \#L, \#P, \#\Delta\}$.

Proof (Sketch) First assume that `act` is an atomic action. The preconditions of $\texttt{act}(\bar{a})$ are expressed by a 4QL^{Bel} formula whose evaluation on a belief base is deterministic polynomial [5]. Computing the effects of `act` requires to iterate through $3i$-worlds in the belief base Δ. In each iteration zero or two well supported models are computed which requires

Table 7 Semantics of actions' instances

- For any atomic action instance $\texttt{act}(\bar{a})$, $\texttt{act}(\bar{a})(\varDelta) \stackrel{\text{def}}{=} \varDelta'$, where $\varDelta'$ is defined by Algorithm 5;
- $(\texttt{act}_1;\texttt{act}_2)(\bar{a})(\varDelta) \stackrel{\text{def}}{=} \texttt{act}_2(\bar{a})(\texttt{act}_1(\bar{a})(\varDelta))$;
- $(F \Rightarrow \texttt{act}_1/\texttt{act}_2)(\bar{a})(\varDelta) \stackrel{\text{def}}{=} \begin{cases} \texttt{act}_1(\bar{a})(\varDelta) & \text{when } F(\bar{a})(\varDelta) = \mathbf{t}; \\ \texttt{act}_2(\bar{a})(\varDelta) & \text{otherwise;} \end{cases}$
- $(\texttt{act}_1||\texttt{act}_2)(\bar{a})(\varDelta) \stackrel{\text{def}}{=} \{$
  $\big(u^+_{\texttt{act}_1(\bar{a})}(w) \cup u^+_{\texttt{act}_2(\bar{a})}(w)\big) \setminus \big(u^-_{\texttt{act}_1(\bar{a})}(w) \cup u^-_{\texttt{act}_2(\bar{a})}(w)\big) \cup$
  $\{\ell, \neg\ell \mid \ell \in (u^+_{\texttt{act}_1(\bar{a})}(w) \cap u^-_{\texttt{act}_2(\bar{a})}(w)) \cup (u^-_{\texttt{act}_1(\bar{a})}(w) \cap u^+_{\texttt{act}_2(\bar{a})}(w))\}$
  $\mid w \in \varDelta\}$;

where, for $i \in \{1, 2\}$, $u^+_{\texttt{act}_i(\bar{a})}$ and $u^-_{\texttt{act}_i(\bar{a})}$ refer to sets of literals computed in Algorithm 5 applied to action $\texttt{act}_i(\bar{a})$.

deterministic polynomial time in $\#D$ [5, 36] The number of worlds is not greater than $\#\Delta$, so altogether deterministic polynomial time (in $\max\{\#D, \#P, \#\Delta\}$) suffices.

If `act` is a composite action, recursive procedure based on clauses given in Table 7, can be executed. The recursion depth is limited by $\#L$ and each recursion step requires either constant time or (when atomic action is reached), deterministic polynomial time, as above. Altogether deterministic polynomial time (in $\max\{\#D, \#L, \#P, \#\Delta\}$) suffices as well. □

Theorem 2 ACTLOG *captures deterministic polynomial time over linearly ordered domains. That is, every atomic action with polynomially computable preconditions and effects can be expressed in* ACTLOG.

Proof (Sketch) To prove the theorem, a technique similar to one given in [35] can be applied. That is, as shown there, all stratified DATALOG$^{\neg}$ programs can be emulated in 4QL. It is well-known that over linearly ordered domains, stratified DATALOG$^{\neg}$ captures PTIME [1], so 4QL does, too. The same holds for 4QL$^{\mathrm{Bel}}$, being an extension of 4QL.

Let `act` be an action with polynomially computable preconditions and effects. Then such preconditions and effects can be expressed in stratified DATALOG$^{\neg}$, so in 4QL$^{\mathrm{Bel}}$, too. Since formulas specifying preconditions (like α in Action 3) may refer to 4QL$^{\mathrm{Bel}}$ modules, they can express any polynomially computable preconditions. The effects are expressed by 4QL$^{\mathrm{Bel}}$ programs (like β^{+}, β^{-} in Action 3), so obviously capture all polynomially computable effects as well. □

4 A decontamination case study

Let us illustrate our approach by sample actions' specifications related to a scenario originally introduced in [16].

4.1 The scenario

Assume that a contamination has been detected in a grid-shaped area and a clean-up mission is to be started. When the contamination is too strong in a given cell, an evacuation has to be launched there. Each cell of the grid is characterized by the following features:

- poison concentration level with possible values `safe`, `dangerous` and `explosive`. When the concentration of the poison is high enough and weather conditions are adverse, then an explosive state occurs. In such a case, evacuation has to be initiated immediately, followed by a rescue mission after the explosion;
- current weather conditions given by temperature and pressure (expressed by integers), as well as humidity with possible values: `rain`, `normal` and `dry`.

When the situation in a cell is safe then no action is required. Otherwise, when the situation is unsafe or the safety of a cell cannot be determined, relevant actions have to be applied according to the following rules:

- **when** safe poison concentration: **then** unconditionally: situation recognition;
- **when** dangerous poison concentration **then**:
    - **when** humidity rain: spread a decontamination powder and then pour a liquid catalyst;
    - **when** humidity normal or dry: pour the liquid catalyst from the air;

- **when** explosive poison concentration **then**:
    - before explosion: evacuation;
    - after explosion: rescue action.

We assume that a sufficient number of neutralizing ground robots and drones is available. Each robot and drone is equipped with sensors measuring poison concentration, temperature, pressure and humidity. The goal is to make all cells in the area safe.

4.2 Sample actions in the case study

Actions will refer to the relations described in Table 8. We assume that these relations are provided by the underlying belief base.

The most basic ground robot's activity depends on moving from one place to another. Action 9 provides its specification.

Action 9 Ground robot R moves to the cell C.

```
action goTo(R,C):
    preconditions:
        $\mathrm{safe}(C) \in \{\mathfrak{f}, \mathfrak{i}, \mathfrak{u}\} \wedge \mathrm{status}(R, \mathrm{ready}) \wedge \mathrm{type}(R, \mathrm{ground})$
    postconditions:
        add:
            $\mathrm{status}(R, \mathrm{occupied}).$
            $\mathrm{position}(R, C).$
        remove:
            $\mathrm{status}(R, \mathrm{ready}).$
            $\mathrm{position}(R, C') \text{ :- } \mathrm{place}(C') \wedge C' \neq C.$
end.
```

Note that the action can be executed only when its preconditions are true. This may happen when its input belief base entails safe(C)=$\mathfrak{f}$, or contains inconsistent information as to the safety of cell C or C's safety is unknown, which happens when safe(C) $\in \{\mathfrak{f}, \mathfrak{i}, \mathfrak{u}\}$ is true.

Table 8 Relations used in the case study

- place(C), indicating that cell C belongs to the considered area;
- concentration(C, PC), indicating the poison's concentration level PC in cell C;
- safe(C), indicating that cell C is decontaminated;
- temperature(C, T), indicating the temperature level T in cell C;
- pressure(C, P), indicating the pressure level P in cell C;
- humidity(C, H), indicating the humidity level H in cell C;
- position(R, C), indicating that robot R is in the cell C;
- type(R, T), indicating the robot's R type, where T∈{ground, uav};
- status(R, S), indicating the current status S of robot R, where S∈{ready, occupied};
- acceptable(D, P, T), indicating that pressure P and temperature T are suitable for applying the decontamination method D;
- airSupportNeeded(C), indicating a need for air support in cell C;
- catalystNeeded(C), indicating a need to use a catalyst in cell C;
- checkNeeded(C), indicating a need for a final check in cell C;
- rescueNeeded(C), indicating a need for an after-explosion rescue in C.

Consider the belief base Δ consisting of the following two $3i$-worlds:

- B1={place(1), place(2), place(3), status(r1, ready), position(r1, 2), type(r1, ground), safe(1), ¬ safe(1) };
- B2={place(1), place(2), place(3), status(r1, ready), position(r1, 2), type(r1, ground), safe(1)}.

In B1, the value of safe(1) is $\mathfrak{i}$, the values of status(r1, ready), type(r1, ground) are $\mathfrak{t}$, and airSupportNeeded(1) and catalystNeeded(1) are $\mathfrak{u}$ making the precondition of `goto(r1,1)` true. Therefore the action is executable on B1.

In B2, the values of safe(1), status(r1,ready) and type(r1,ground) are $\mathfrak{t}$, making the precondition of `goto(r1,1)` false. Therefore the action is not executable on B2.

The effects of executing action `goto(r1,1)` on Δ is Δ' ={B1′, B2}, where:

$$\begin{aligned} \text{B1}' = \{&\text{place(1), place(2), place(3), status(r1, occupied),}\\ &\text{position(r1, 1), type(r1, ground), safe(1), } \neg \text{ safe(1)}\}. \end{aligned}$$

Action 10 specifies an action of flying to a given position.

Action 10 Robot `R` flies to the cell `C`.

```
action flyTo(R, C):
    preconditions:
        airSupportNeeded(C) ∧ status(R, ready) ∧ type(R, uav)
    postconditions:
        the same as in Action 9
end.
```

In the scenario we have two neutralization actions specified as Action 11 (the decontamination powder spreading) and Action 12 (the catalyst pouring).

Action 11 Robot `R` spreads decontamination powder on the cell `C`.

```
action spreadPowder(R, C):
    preconditions:
        position(R, C) ∧ status(R, occupied) ∧ type(R, ground) ∧
        Bel(humidity(C, rain) in {𝔱, 𝔦 }) ∧
        Bel(pressure(C, P) ∧ temperature(C, T) ∧ concentration(C, dangerous)) ∧
        acceptable(powderdPlusCatalyst, P, T)
    postconditions:
        add:
            catalystNeeded(X).
end.
```

Action 12 Robot `R` pours catalyst on cell `C`.

```
action pourCatalyst(R, C):
    preconditions:
        position(R, C) $\wedge$ status(R, occupied) $\wedge$ type(R, ground) $\wedge$ catalystNeeded(C) $\wedge$
        Bel(pressure(C, P) $\wedge$ temperature(C, V)) $\wedge$
        acceptable(catalystAfterPowder, P, T)
    postconditions:
        add:
            checkNeeded(C).
            status(R, ready).
        remove:
            status(ID, occupied).
            catalystNeeded(X).
end.
```

Action 13 is to be performed under dry and normal weather conditions when air support is needed and the catalyst should be sprayed from the air.

Action 13 Robot `R` calls air support for the cell `C`.

```
action callAirSupport(R, C):
    preconditions:
        position(R, C) $\wedge$ status(R, occupied) $\wedge$ type(R, ground) $\wedge$
        (Bel(humidity(C, normal)$\in\{\mathbf{t}, \mathbf{i}\}$) $\vee$
          Bel(humidity(X, dry)$\in\{\mathbf{t}, \mathbf{i}\}$)) $\wedge$
        Bel(pressure(C, P) $\wedge$ temperature(C, T) $\wedge$ concentration(C,dangerous) $\wedge$
        acceptable(catalystAfterPowder, P, T)
    postconditions:
        add:
            airSupportNeeded(C).
            status(R, ready).
        remove:
            status(R, occupied).
end.
```

When air support is called, the robot is free to find another unhandled cell. Later, after the decontamination, a (possibly the same) ground robot will return to the cell to verify its safety.

The air-decontamination action is formalized as Action 14.

Action 14 Drone `ID` sprays liquid `L1` over place `X`.

```
action spray(ID, X):
    preconditions:
        airSupportNeeded(X) $\wedge$ position(ID, X) $\wedge$ status(ID, occupied) $\wedge$ type(ID, uav)
    postconditions:
        add:
            checkNeeded(X).
            status(ID, ready).
        remove:
            status(ID, occupied).
            airSupportNeeded(X).
end.
```

Each ground robot can perform a check by moving to a cell and comparing sensor readings with the current norm values (Action 15).

Action 15 Robot `R` validates safety of cell `C`.

```
action checkCell(R, C):
    preconditions:
        position(R, C) ∧ status(R, occupied) ∧ type(R, ground) ∧ checkNeeded(C)
    postconditions:
        add:
            safe(C) :– Bel(concentration(C,safe)).
            ¬safe(C) :– ¬Bel(concentration(C,safe)).
            status(R, ready).
        remove:
            status(R, occupied).
            checkNeeded(C).
end.
```

Note that a cell safety is inconsistent when concentration sensor's readings are contradictory. Generally, inconsistencies may be produced as the output of action's postconditions. Using this property, inconsistencies may be passed between belief bases.

4.3 Composite actions in the case study

Composite actions allow one to specify complex procedures rather than planning them properly from scratch. Although the planner could eventually find appropriate ordering of actions, hinting the typical solutions may significantly reduce planning time and resources. An example of a composite action is given as Action 16.

Action 16 A composite action for a complex decontamination procedure.

```
action decontaminateFromAir(R, UAV, C):
    composite:
        callAirSupport(R, C);
        flyTo(UAV, C);
        spray(UAV, C);
        checkCell(C, R)
end.
```

Remark 6 Action 16 illustrates the performance gain with a composite action. The action consists of a sequential composition of four atomic actions and each one has six possible actions (`goTo`, `flyTo`, `spreadPowder`, `pourCatalyst`, `callAirSupport` and `checkCell`) that might be tried before selecting a proper one for decontamination. Altogether, this gives $6^4 = 1\,296$ possible actions' sequences checks to construct this plan.

When Action 16 is introduced, the number of checks can be reduced to just one for the composite action. Also the branching factor in planning is decreased. Clearly, the performance gain in more complex real-world scenarios may be more spectacular.

Action 17 specifies a parallel action. A catalyst can be poured simultaneously with the powder spread which may save the time spent on the whole decontamination process.

Action 17 Simple parallel action.

```
action decontaminateWithCatalyst(R1, R2, C):
    composite:
        spreadPowder(R1, C) || pourCatalyst(R2, C)
end.
```

Finally, conditional composite actions can be easily used in the scenario. Decontamination actions in Section 4.2 contain humidity checks to select the appropriate method. Action 18 is such a higher-level action. It also demonstrates the possibility of providing additional preconditions for the entire composite action. Observe that each of atomic actions included in a composite action can change the environment and affect preconditions of other atomic actions (and, e.g., break their sequential execution).

Action 18 A single action dealing with all humidity conditions.

```
action unifiedDecontamination(R, C):
    preconditions:
        position(R, C) $\wedge$ status(R, occupied) $\wedge$ type(R, ground) $\wedge$
        concentration(C, dangerous)
    composite:
        Bel(humidity(C, rain)) $\Rightarrow$
            decontaminateWithCatalyst(R, C)/callAirSupport(R, C)
end.
```

All actions can be freely composed to achieve the desired specification. Action 19 provides a specification of this kind.

Action 19 More advanced action for two agents.

```
action decontaminateAndValidate(R1, R2, C1, C2, C3):
    composite:
        unifiedDecontamination(R1, C1);
        (goTo(R1, C2) || checkCell(R2, C1) );
        goTo(R2, C3)
end.
```

5 Related work

Theories of action and change have been intensively investigated during the past decades (see books [40, 43, 45, 48, 53] and references there). Below we concentrate on the most relevant results.

Though ACTLOG is influenced by the STRIPS formalism [22], it is more general. While STRIPS uses classical logic as the specification language, our approach is based on a non-classical four-valued formalism, allowing for inconsistencies, ignorance and doxastic reasoning. While STRIPS actions are state transformers, in ActLog they transform belief bases representing possible alternatives and non-determinism in a complexitywise controlled manner. Moreover, unlike in SRIPS, ActLog's actions' effects are expressed by rules capturing PTIME specifications.

After STRIPS, a great deal of attention has been devoted to the reasoning about action and change. The main formalisms developed in this area are Situation Calculus (SC), Fluent Calculus (FC), Event Calculus (EC) and Temporal Action Logic (TAL). SC, introduced in [38], has been intensively studied and developed [32, 42, 43]. The main concepts in SC are actions, fluents and situations. Actions are domain elements, situations are sequences of actions and fluents are features whose values may change over time. As an implementation tool built over SC, the GOLOG logic programming language has been developed [33]. The FC formalism is a variant of SC, where situations and states are separated: situations represent the history while states represent the current state of the world [51, 53]. A constraint logic programming framework based on the FC has been designed [52]. In the EC formalism [30], actions (events) and fluents are considered. Fluents are evaluated in time points. EC, restricted to Horn clauses with negation, can be run in Prolog. For an exhaustive presentation of EC see [40]. A comprehensive approach to temporal action specification based on Temporal Action Logic (TAL), together with a forward chaining planner, has been developed in a series of papers – see, e.g., [7–9]. TAL-based composite actions with constraints are investigated in [10]. Though these formalisms allow for composite actions and address incomplete information, none of them attacks $3i$-related phenomena in a comprehensive manner. In particular, no tools for handling and disambiguating inconsistent information are provided.

Since early 1980s, plans more complex than sequences of actions have been considered. SIPE (System for Interactive Planning and Execution Monitoring [56]) with its later successor SIPE-2 [57] consider plans to be acyclic graphs with actions executed in parallel or sequentially. This planning system explicitly supports parallelism and conditionals which is also one of our goals to achieve. However our planning mechanism supports $3i$ environments while SIPE-based systems recognize general uncertainty of information represented by action's likeliness-of-success parameter. In our opinion, ActLog ensures higher level of freedom in defining actions and modeling realistic environments. Later, composite actions were investigated in many works, e.g., in [19, 27, 39]. Parallel action compositions have been used in the SC, FC, EC, TAL frameworks, and also, e.g., in [41] (determining which actions can be executed in parallel), [31] (developing a planning architecture with parallel action executions) or [17] (supporting parallel actions prepared especially for IPC-4 planning contest).

Unlike other approaches, ActLog offers a uniform framework allowing for tractable forms of paracomplete, paraconsistent, and doxastic reasoning. While guaranteeing tractability of reasoning and computing actions' effects, it is expressive enough to capture all tractable actions' specifications and underlying reasoning processes.

6 Conclusions

The paper presents a rule-based language ActLog developed for specifying actions in informationally complex and possibly defective environments. ActLog complements other approaches by providing rich and comfortable tools for handling inconsistency and ignorance in a tractable manner. Moreover, the involved agents can have their own belief bases or share beliefs in a group. The language permits to evaluate belief operators on arbitrary belief bases, not necessarily on the global one. This supports contemporary approaches to individual and group reasoning.

Planning in situated systems is a complex issue, substantially affecting their performance. To overcome this complexity, predefined plan skeleton libraries are typically being used rather than planning from the first principles. However, plan libraries are applicable when the environment and goals are recognized at least to some extent. When agents explore unknown environments, planning from scratch may turn out necessary: the predefined composite actions as plans' building blocks, may reduce the complexity of planning.

We have illustrated ActLog with a scenario adapted form the literature. We demonstrated that the generated plans may result in unknown or inconsistent results being still valuable: in situations where other frameworks fail, ActLog may deliver a feasible plan to be monitored and updated during its execution. This is especially important in critical/rescue situations.

Summing up, ActLog provides a nuanced action specification tools, allowing for subtle interplay among various forms of nonmonotonic, paraconsistent, paracomplete and doxastic reasoning methods applicable in informationally complex environments.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of databases. Addison-Wesley Pub. Co., Reading (1996)
2. Belnap, N.: How a Computer Should Think. In: Ryle, G. (ed.) Contemporary Aspects of Philosophy, pp. 30–55. Oriel Press, Stocksfield (1977)
3. Bertossi, L., Hunter, A., Schaub, T.: Introduction to inconsistency tolerance. In: Bertossi et al. (ed.). Inconsistency Tolerance, LNCS, pp. 1–14
4. Bertossi, L., Hunter, A., Schaub, T. (eds.): Inconsistency Tolerance, LNCS, vol. 3300. Springer, Berlin (2005)
5. Białek, Ł., Dunin-Kęplicz, B., Szałas, A.: Rule-Based Reasoning with Belief Structures. In: Kryszkiewicz, M., Appice, A., Ślęzak, D., Rybiński, H., Skowron, A., RaŚ, Z. (eds.) Foundations of Intelligent Systems, Proceedings of ISMIS Conference. LNAI, vol. 10352, pp. 229–239. Springer (2017)
6. Białek, Ł., Dunin-Kęplicz, B., Szałas, A.: Towards a Paraconsistent Approach to Actions in Distributed Information-Rich Environments. In: Ivanović, M., Bădică, C., Dix, J., Jovanović, Z., Malgeri, M., Savić, M. (eds.) Proceedings of IDC - Intelligent Distributed Computing XI. Studies in Computational Intelligence, vol. 737, pp. 49–60. Springer (2017)
7. Doherty, P., Kvarnström, J.: TALplanner: A temporal logic based forward chaining planner. Ann. Math. Artif. Intell. **30**, 119–169 (2001)
8. Doherty, P., Kvarnström, J.: TALplanner: A temporal logic-based planner. AI Mag. **22**(3), 95–102 (2001)
9. Doherty, P., Kvarnström, J.: The Handbook of Knowledge Representation. In: Lifschitz, V., Van Harmelen, F., Porter, F. (eds.), pp. 709–757. Elsevier (2008)
10. Doherty, P., Kvarnström, J., Szałas, A.: Temporal Composite Actions with Constraints. In: Brewka, G., Eiter, T., Mcilraith, S. (eds.) Proceedings of 13Th International Conference KR: Principles of Knowledge Representation and Reasoning, pp. 478–488. AAAI Press (2012)
11. Doherty, P., Szałas, A.: Stability, supportedness, minimality and Kleene Answer Set Programs. In: Eiter, T., Strass, H., Truszczyński, M., Woltran, S. (eds.) Advances in Knowledge Representation, Logic Programming, and Abstract Argumentation, LNCS, vol. 9060, pp. 125–140. Springer International Publishing (2015)

12. Dunin-Kęplicz, B., Szałas, A.: Epistemic Profiles and Belief Structures. In: Proceedings of KES-AMSTA 2012: Agents and Multi-Agent Systems: Technologies and Applications. LNCS, vol. 7327, pp. 360–369. Springer (2012)
13. Dunin-Kęplicz, B., Szałas, A.: Taming complex beliefs. Trans. Comput. Collective Intell. XI LNCS **8065**, 1–21 (2013)
14. Dunin-Kęplicz, B., Szałas, A.: Indeterministic Belief Structures. In: Jezic, G., Kusek, M., Lovrek, I., Howlett, J., Lakhmi, J. (eds.) Agent and Multi-Agent Systems: Technologies and Applications: Proceedings of 8th International Conference KES-AMSTA, pp. 57–66. Springer (2014)
15. Dunin-Kęplicz, B., Verbrugge, R.: Teamwork in Multi-Agent systems. a formal approach. Wiley, New York (2010)
16. Dunin-Kęplicz, B., Verbrugge, R., Ślizak, M.: TeamLog in action: a case study in teamwork. Comput. Sci. Inf. Syst. **7**(3), 569–595 (2010)
17. Edelkamp, S., Hoffmann, J.: PDDL2: The language for the classical part of the 4th international planning competition. In: Proceedings of the 4th International Planning Competition (2004)
18. Eiter, T., Faber, W., Leone, N., Pfeifer, G., Polleres, A.: Planning under Incomplete Knowledge. In: Lloyd, J., Dahl, V., Furbach, U., Kerber, M., Lau, K.K., Palamidessi, C., Pereira, L., Sagiv, Y., Stuckey, P. (eds.) Proceedings of Computational Logic: 1St International Conference, pp. 807–821. Springer (2000)
19. Eiter, T., Faber, W., Pfeifer, G.: Declarative Planning and Knowledge Representation in an Action Language. In: Sugumaran, V. (ed.) Intelligent Information Technologies: Concepts, Methodologies, Tools, and Applications, pp. 192–221. IGI Global (2008)
20. Fagin, R., Halpern, J., Moses, Y., Vardi, M.: Reasoning about knowledge the. MIT Press, Cambridge (2003)
21. Ferraris, P., Lifschitz, V.: On the Minimality of Stable Models. In: Balduccini, M., Son, T. (eds.) Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning. LNCS, vol. 6565, pp. 64–73. Springer (2011)
22. Fikes, R.E., Nilsson, N.J.: STRIPS: a new approach to the application of theorem proving to problem solving. In: Proceedings of the 2Nd International Joint Conference on Artificial Intelligence, pp. 608–620. IJCAI'71, Morgan Kaufmann Publishers Inc. (1971)
23. Fitting, M.: Bilattices are Nice Things. In: Proceedings of Philog Conference on Self-Reference. The Danish Network for Philosophical Logic and Its Applications, Copenhagen (2002)
24. Fitting, M.C.: Bilattices in Logic Programming. In: Epstein, G. (ed.) 20Th International Symposium on Multiple-Valued Logic, pp. 238–247. IEEE CS Press, Los Alamitos (1990)
25. Ginsberg, M.: Multi-Valued Logics. In: 5Th National Conference on AI Proceedings of AAAI-86. pp. 243–247 (1986)
26. Ginsberg, M.: Multivalued logics: a uniform approach to reasoning in AI. Comput. Intell. **4**, 256–316 (1988)
27. Giunchiglia, E., Lee, J., Lifschitz, V., Mc-Cain, N., Turner, H.: Nonmonotonic causal theories. Artif. Intell. **153**(1-2), 49–104 (2004)
28. Hewitt, C.: Formalizing common sense for scalable inconsistency-robust information integration using Direct Logic reasoning and the actor model. arXiv:0812.4852 (2008)
29. Hewitt, C., Woods, J. (eds.): Inconsistency Robustness. College Publications (2015)
30. Kowalski, R., Sergot, M.: A logic-based calculus of events. N. Gener. Comput. **4**(1), 67–95 (1986)
31. Lever, J., Richards, B.: parcPlan: a Planning Architecture with Parallel Actions, Resources and Constraints. In: Raś, Z.W., Zemankova, M. (eds.) Methodologies for Intelligent Systems, pp. 213–222. Springer Berlin Heidelberg, Berlin (1994)
32. Levesque, H., Pirri, F., Reiter, R.: Foundations for the situation calculus. Electron. Trans. AI **2**(3-4), 159–178 (1998)
33. Levesque, H., Reiter, R., Lespérance, Y., Lin, F., Scherl, R.: GOLOG: a logic programming language for dynamic domains. J. Log. Program. **31**, 59–84 (1997)
34. Małuszyński, J., Szałas, A.: Living with Inconsistency and Taming Nonmonotonicity. In: De Moor, O., Gottlob, G., Furche, T., Sellers, A. (eds.) Datalog Reloaded. LNCS, vol. 6702, pp. 384–398. Springer (2011)
35. Małuszyński, J., Szałas, A.: Logical foundations and complexity of 4QL, a query language with unrestricted negation. J. Appl. Non-Class. Log. **21**(2), 211–232 (2011)
36. Małuszyński, J., Szałas, A.: Partiality and Inconsistency in Agents' Belief Bases. In: Barbucha, D., Le, M., Howlett, R., Jain, L. (eds.) KES-AMSTA. Frontiers in Artificial Intelligence and Applications, vol. 252, pp. 3–17. IOS Press (2013)

37. Małuszyński, J., Szałas, A., Vitória, A.: Paraconsistent Logic Programs with Four-Valued Rough Sets. In: Chan, C.C., Grzymala-Busse, J., Ziarko, W. (eds.) Proceedings of 6Th International Conference on Rough Sets and Current Trends in Computing (RSCTC 2008). LNAI, vol. 5306, pp. 41–51 (2008)
38. McCarthy, J., Laboratory, S.A.I.: Situations, Actions, and Causal Laws. Memo (Stanford Artificial Intelligence Project), Stanford University, AI Project (1963)
39. Mcilraith, S., Fadel, R.: Planning with Complex Actions. In: Proceedings NMR'02, pp. 356–364 (2002)
40. Mueller, E.: Commonsense reasoning. An Event Calculus Based Approach. Morgan Faufmann, San Mateo (2006)
41. Regnier, P., Fade, B.: Complete Determination of Parallel Actions and Temporal Optimization in Linear Plans of Action. In: European Workshop on Planning, pp. 100–111. Springer, Berlin (1991)
42. Reiter, R.: The Frame Problem in the Situation Calculus: a Simple Solution (Sometimes) and a Completeness Result for Goal Regression. In: Lifshitz, V. (ed.) Artificial Intelligence and Mathematical Theory of Computation: Papers in Honour of John Mccarthy, pp. 359–380. Academic Press Professional Inc. (1991)
43. Reiter, R.: Knowledge in action: Logical foundations for specifying and implementing dynamical systems. MIT Press, Cambridge (2001)
44. Sakama, C., Inoue, K.: An alternative approach to the semantics of disjunctive logic programs and deductive databases. J. Autom. Reason. **13**(1), 145–172 (1994)
45. Sandewall, E.: Features and Fluents: The Representation of Knowledge about Dynamical Systems, vol. 1 Clarendon Press (1994)
46. Shepherdson, J.: Negation in Logic Programming. In: Minker, J. (ed.) Foundations of Deductive Databases and Logic Programming, pp. 19–88, Morgan Kaufmann (1988)
47. Shieber, S.M.: Solving Problems in an Uncertain World. Bachelor's thesis, Harvard College (1981)
48. Shoham, Y.: Reasoning about change: Time and causation from the standpoint of artificial intelligence. MIT Press, Cambridge (1987)
49. Soininen, T., Niemelä, I.: Developing a Declarative Rule Language for Applications in Product Configuration. In: Gupta, G. (ed.) Proceedings of PADL'99. LNCS, vol. 1551, pp. 305–319. Springer (1999)
50. Szałas, A.: How an agent might think. Log. J. IGPL **21**(3), 515–535 (2013)
51. Thielscher, M.: Introduction to the fluent calculus. Electron. Trans. AI **2**(3-4), 179–192 (1998)
52. Thielscher, M.: FLUX: a logic programming method for reasoning agents. Theory Pract. Log. Programm. **5**(4-5), 533–565 (2005)
53. Thielscher, M.: Reasoning robots: The art and science of programming robotic agents. Springer, Berlin (2011)
54. Thrun, S., Burgard, W., Fox, D.: Probabilistic robotics (intelligent robotics and autonomous agents). The MIT Press, Cambridge (2005)
55. Vitória, A., Małuszyński, J., Szałas, A.: Modeling and reasoning with paraconsistent rough sets. Fund. Inf. **97**(4), 405–438 (2009)
56. Wilkins, D.E.: Domain-independent planning representation and plan generation. Artif. Intell. **22**(3), 269–301 (1984)
57. Wilkins, D.E., Myers, K.L., Lowrance, J.D., Wesley, L.P.: Planning and reacting in uncertain and dynamic environments. J. Exper. Theor. Artif. Intell. **7**(1), 121–152 (1995)
58. Zadeh, L.: Fuzzy sets. Inf. Control **8**, 333–353 (1965)