Towards a Paraconsistent Approach to Actions in Distributed Information-Rich Environments

Łukasz Białek
Barbara Dunin-Kęplicz
Andrzej Szałas
(Institute of Informatics, University of Warsaw, Poland)

IDC'2017

The Structure of Talk

- Motivations.
- Goals.
- The ActLog language.
- Tractability of the approach.
- Some examples.
- Conclusions.

Inconsistency robustness (C. Hewitt, 2008)

> *"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."*

L. Bertossi, A. Hunter, T. Schaub, 2005

> *"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."*

Inconsistency robustness (C. Hewitt, 2008)

"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."

L. Bertossi, A. Hunter, T. Schaub, 2005

"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."

Motivations: Action Languages

Distributed information sources

Frequently deliver information contradicting one another.

Contemporary action languages

Not prepared to work with inconsistent information.

Motivations: Action Languages

Distributed information sources

Frequently deliver information contradicting one another.

Contemporary action languages

Not prepared to work with inconsistent information.

Typical action specification

`{preconditions}` `action` `{postconditions}`

↑ enable `action`

↑ `action`'s effects

Example: a part of a rescue robot specification

`{safe-path(X,Y),in(X),X≠Y}` `go-to(X,Y)` `{¬in(X),in(Y)}`

What if:

- Multiple sensors deliver inconsistent information about `safe-path(X,Y)`?
- There is no information about `safe-path(X,Y)`?
- A victim might be located in place `Y`?

Typical action specification

`{preconditions}` `action` `{postconditions}`

↑ enable `action`

↑ `action`'s effects

Example: a part of a rescue robot specification

`{safe-path(X,Y),in(X),X≠Y}` `go-to(X,Y)` `{¬in(X),in(Y)}`

What if:

- Multiple sensors deliver inconsistent information about `safe-path(X,Y)`?
- There is no information about `safe-path(X,Y)`?
- A victim might be located in place `Y`?

Typical action specification

`{preconditions}` `action` `{postconditions}`

↑ enable `action`

↑ `action`'s effects

Example: a part of a rescue robot specification

`{safe-path(X,Y),in(X),X≠Y}` `go-to(X,Y)` `{¬in(X),in(Y)}`

What if:

- Multiple sensors deliver inconsistent information about `safe-path(X,Y)`?
- There is no information about `safe-path(X,Y)`?
- A victim might be located in place `Y`?

Typical action specification

`{preconditions} action {postconditions}`

↑ enable `action` ↑ `action`'s effects

Example: a part of a rescue robot specification

`{safe-path(X,Y),in(X),X≠Y} go-to(X,Y) {¬in(X),in(Y)}`

What if:

- Multiple sensors deliver inconsistent information about `safe-path(X,Y)`?
- There is no information about `safe-path(X,Y)`?
- A victim might be located in place `Y`?

Typical action specification

`{preconditions}` `action` `{postconditions}`

↑ enable `action`

↑ `action`'s effects

Example: a part of a rescue robot specification

`{safe-path(X,Y),in(X),X≠Y}` `go-to(X,Y)` `{¬in(X),in(Y)}`

What if:

- Multiple sensors deliver inconsistent information about `safe-path(X,Y)`?
- There is no information about `safe-path(X,Y)`?
- A victim might be located in place `Y`?

Needed a concise specification of:

$\underbrace{\{\texttt{safe-path(X,Y)}}_{\textit{true}}, \underbrace{\texttt{in(X)}, X \neq Y\}}_{\textit{true}}$ go-to(X,Y)$\{\underbrace{\neg\texttt{in(X)}}_{\textit{true}}, \underbrace{\texttt{in(Y)}\}}_{\textit{true}}$

$\underbrace{\{\texttt{safe-path(X,Y)}}_{\textit{inconsistent}}, \underbrace{\texttt{in(X)}, X \neq Y\}}_{\textit{true}}$ go-to(X,Y)$\{\underbrace{\neg\texttt{in(X)}}_{\textit{true}}, \underbrace{\texttt{in(Y)}}_{\textit{inconsistent}}\}$

$\underbrace{\{\texttt{safe-path(X,Y)}}_{\textit{unknown}}, \underbrace{\texttt{in(X)}, X \neq Y\}}_{\textit{true}}$ go-to(X,Y)$\{\underbrace{\neg\texttt{in(X)}}_{\textit{true}}, \underbrace{\texttt{in(Y)}}_{\textit{unknown}}\}$

Motivations: Example – continued

Needed a concise specification of:

$\underbrace{\{\texttt{safe-path(X,Y)}}_{\textit{true}}, \underbrace{\texttt{in(X)}, X \neq Y\}}_{\textit{true}}$ go-to(X,Y)$\{\underbrace{\neg\texttt{in(X)}}_{\textit{true}}, \underbrace{\texttt{in(Y)}}_{\textit{true}}\}$

$\underbrace{\{\texttt{safe-path(X,Y)}}_{\textit{inconsistent}}, \underbrace{\texttt{in(X)}, X \neq Y\}}_{\textit{true}}$ go-to(X,Y)$\{\underbrace{\neg\texttt{in(X)}}_{\textit{true}}, \underbrace{\texttt{in(Y)}}_{\textit{inconsistent}}\}$

$\underbrace{\{\texttt{safe-path(X,Y)}}_{\textit{unknown}}, \underbrace{\texttt{in(X)}, X \neq Y\}}_{\textit{true}}$ go-to(X,Y)$\{\underbrace{\neg\texttt{in(X)}}_{\textit{true}}, \underbrace{\texttt{in(Y)}}_{\textit{unknown}}\}$

Needed a concise specification of:

$\underbrace{\texttt{\{safe-path(X,Y)}}_{true}, \underbrace{\texttt{in(X)}, X \neq Y}_{true}\}$ go-to(X,Y)$\{\underbrace{\neg\texttt{in(X)}}_{true}, \underbrace{\texttt{in(Y)}}_{true}\}$

$\underbrace{\texttt{\{safe-path(X,Y)}}_{inconsistent}, \underbrace{\texttt{in(X)}, X \neq Y}_{true}\}$ go-to(X,Y)$\{\underbrace{\neg\texttt{in(X)}}_{true}, \underbrace{\texttt{in(Y)}}_{inconsistent}\}$

$\underbrace{\texttt{\{safe-path(X,Y)}}_{unknown}, \underbrace{\texttt{in(X)}, X \neq Y}_{true}\}$ go-to(X,Y)$\{\underbrace{\neg\texttt{in(X)}}_{true}, \underbrace{\texttt{in(Y)}}_{unknown}\}$

A formal language ActLog allowing for:

- Concise rule-based specification of actions and their effects.
- Flexibility in evaluation of formulas in distributed paraconsistent and paracomplete belief bases.
- Tractability of computing actions' preconditions and effects.
- Practical expressiveness: all (and only) actions with effects computable in deterministic polynomial time can be expressed.

Małuszyński, Szałas, Vitória

- Truth values: $\boldsymbol{t}$ (true), $\boldsymbol{f}$ (false), $\boldsymbol{i}$ (inconsistent), $\boldsymbol{u}$ (unknown);
- Truth ordering: $\boldsymbol{f} < \boldsymbol{u} < \boldsymbol{i} < \boldsymbol{t}$;
- Negation: $\neg\boldsymbol{f} \stackrel{\text{def}}{=} \boldsymbol{t}$, $\neg\boldsymbol{u} \stackrel{\text{def}}{=} \boldsymbol{u}$, $\neg\boldsymbol{i} \stackrel{\text{def}}{=} \boldsymbol{i}$, $\neg\boldsymbol{t} \stackrel{\text{def}}{=} \boldsymbol{f}$;
- Conjunction: $(s \wedge t) \stackrel{\text{def}}{=} \min\{s, t\}$ (wrt. $<$);
- Disjunction: $(s \vee t) \stackrel{\text{def}}{=} \max\{s, t\}$ (wrt. $<$).

4ql

- Introduced by Małuszyński and Szałas, 2010.
- Uses truth values ***t***, ***f***, ***i***, ***u***.
- Allows for paraconsistent and non-monotonic reasoning.
- Ensures tractable query evaluation.
- Captures all tractable queries.

4QL^{Bel}

- Introduced by Białek, Dunin-Kęplicz and Szałas, 2017.
- Extends 4ql by belief operator plus some other operators.
- Paraconsistent, non-monotonic and doxastic reasoning.
- Retains tractable query evaluation.
- Captures all tractable queries.

Rule Languages 4ql and 4QL$^{\text{Bel}}$

4ql

- Introduced by Małuszyński and Szałas, 2010.
- Uses truth values $\boldsymbol{t}$, $\boldsymbol{f}$, $\boldsymbol{i}$, $\boldsymbol{u}$.
- Allows for paraconsistent and non-monotonic reasoning.
- Ensures tractable query evaluation.
- Captures all tractable queries.

4QL$^{\text{Bel}}$

- Introduced by Białek, Dunin-Kęplicz and Szałas, 2017.
- Extends 4ql by belief operator plus some other operators.
- Paraconsistent, non-monotonic and doxastic reasoning.
- Retains tractable query evaluation.
- Captures all tractable queries.

Belief base

A set $\{C_1, \ldots, C_k\}$, where C_i $(i = 1, \ldots, k)$ are sets of ground literals representing different (e.g., alternative) views on the world.

Note: Sets C_i $(i = 1, \ldots, k)$ may be:

- inconsistent (containing $p(\bar{a})$ and $\neg p(\bar{a})$, for some p, $\bar{a}$);
- incomplete (containing neither $p(\bar{a})$ nor $\neg p(\bar{a})$, for some p, $\bar{a}$).

Action

A belief base transformer (rather than a state transformer).

Belief base

A set $\{C_1, \ldots, C_k\}$, where C_i $(i = 1, \ldots, k)$ are sets of ground literals representing different (e.g., alternative) views on the world.

Note: Sets C_i $(i = 1, \ldots, k)$ may be:

- inconsistent (containing $p(\bar{a})$ and $\neg p(\bar{a})$, for some p, $\bar{a}$);
- incomplete (containing neither $p(\bar{a})$ nor $\neg p(\bar{a})$, for some p, $\bar{a}$).

Action

A belief base transformer (rather than a state transformer).

ActLog: Action Specification

```
1 action name(x̄):
2 |   preconditions:
3 |   |   α(x̄)
4 |   postconditions:
5 |   |   add:
6 |   |   |   β⁺(x̄)
7 |   |   remove:
8 |   |   |   β⁻(x̄)
9 end.
```

- $\alpha(\bar{x})$ is an arbitrary formula of $4QL^{Bel}$ (the precondition of action name);
- $\beta^+(\bar{x}), \beta^-(\bar{x})$ are $4QL^{Bel}$ programs – effects of action name;
- $\beta^+(\bar{x})$ – the set of literals to add, and $\beta^-(\bar{x})$ – to remove.

ActLog: Action Specification

```
1 action name(x̄):
2 |   preconditions:
3 |   |   α(x̄)
4 |   postconditions:
5 |   |   add:
6 |   |   |   β⁺(x̄)
7 |   |   remove:
8 |   |   |   β⁻(x̄)
9 end.
```

- $\alpha(\bar{x})$ is an arbitrary formula of $4QL^{Bel}$ (the precondition of action `name`);
- $\beta^+(\bar{x}), \beta^-(\bar{x})$ are $4QL^{Bel}$ programs – effects of action `name`;
- $\beta^+(\bar{x})$ – the set of literals to add, and $\beta^-(\bar{x})$ – to remove.

Computing Action Effects

Input: action instance $\texttt{name}(\bar{a})$, belief base Δ;
Result: belief base Δ', the effect of $\texttt{name}(\bar{a})$ on Δ;

1 **set** $\Delta' = \emptyset$
2 **foreach** $C \in \Delta$ **do**
3 **if** action $\texttt{name}(\bar{a})$ is executable on C **then**
4 **set** $M^+ = \emptyset$; **set** $M^- = \emptyset$
5 **add** the consequeces of $\beta^+(\bar{a}) \cup C$ **to** M^+
6 **add** the consequeces of $\beta^-(\bar{a}) \cup C$ **to** M^-
7 **add** the set $(C \cup M^+) \setminus M^-$ **to** Δ'
8 **else**
9 **add** the set C **to** Δ'
10 **end**
11 **end**

Notation

For any ActLog specification of an action $\texttt{name}(\bar{x})$:

- $\#D$: the sum of sizes of all domains in the specification;
- $\#M$: the number of $4\mathrm{QL}^{\mathrm{Bel}}$ modules occurring in the specification.

For belief base Δ:

- $\#\Delta$: the number of all literals appearing in Δ (note that $\#\Delta$ is polynomial in the size of $\#D$).

In real-world applications, $\#M$ and $\#D$ are manageable by the hardware/database systems used, so is $\#\Delta$.

Theorem 1

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

Theorem 2

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

Theorem 1

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

Theorem 2

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

Robot `ID` moves from place `X` to place `Y`

```
1 action goTo(ID,X,Y):
2 |   preconditions:
3 |   |   safe-path(X,Y)∈{t,i,u}∧position(ID,X)∧X ≠ Y
4 |   postconditions:
5 |   |   add:
6 |   |   |   position(ID,Y) :- safe-path(X,Y).
7 |   |   remove:
8 |   |   |   position(ID,X):- safe-path(X,Y)≠f.
9 end.
```

Robot ID pours liquid L1 on place X

```
1  action pourL1(ID, X):
2      preconditions:
3          status(ID, ready)∧type(ID, ground)∧position(ID, X)∧
4          Bel(humidity(X,rain)) ∈ {t, i } ∧
5          Bel(pressure(X, v_P)∧ temperature(X, v_T)∧
6              concentration1(X, v_C1)) ∧
7          acceptable(v_P, v_T)∧ v_C1 ≥ e_1
8      postconditions:
9          add:
10             checkNeeded(X).
11 end.
```

ActLog

- Is a novel language designed for specifying actions in distributed information-rich environments.
- Competes with other approaches by providing comfortable tools for handling inconsistency and ignorance, ensuring tractability.
- Permits to evaluate belief operators and other formulas on global and local (distributed) belief bases, so is suitable for applications in distributed intelligent systems.
- Acts on belief bases, not solely on sets of literals.
- In the presented version deals with atomic deterministic actions only but will be extended to composite actions.

ActLog

- Is a novel language designed for specifying actions in distributed information-rich environments.
- Competes with other approaches by providing comfortable tools for handling inconsistency and ignorance, ensuring tractability.
- Permits to evaluate belief operators and other formulas on global and local (distributed) belief bases, so is suitable for applications in distributed intelligent systems.
- Acts on belief bases, not solely on sets of literals.
- In the presented version deals with atomic deterministic actions only but will be extended to composite actions.

ActLog

- Is a novel language designed for specifying actions in distributed information-rich environments.
- Competes with other approaches by providing comfortable tools for handling inconsistency and ignorance, ensuring tractability.
- Permits to evaluate belief operators and other formulas on global and local (distributed) belief bases, so is suitable for applications in distributed intelligent systems.
- Acts on belief bases, not solely on sets of literals.
- In the presented version deals with atomic deterministic actions only but will be extended to composite actions.

ActLog

- Is a novel language designed for specifying actions in distributed information-rich environments.
- Competes with other approaches by providing comfortable tools for handling inconsistency and ignorance, ensuring tractability.
- Permits to evaluate belief operators and other formulas on global and local (distributed) belief bases, so is suitable for applications in distributed intelligent systems.
- Acts on belief bases, not solely on sets of literals.
- In the presented version deals with atomic deterministic actions only but will be extended to composite actions.

ActLog

- Is a novel language designed for specifying actions in distributed information-rich environments.
- Competes with other approaches by providing comfortable tools for handling inconsistency and ignorance, ensuring tractability.
- Permits to evaluate belief operators and other formulas on global and local (distributed) belief bases, so is suitable for applications in distributed intelligent systems.
- Acts on belief bases, not solely on sets of literals.
- In the presented version deals with atomic deterministic actions only but will be extended to composite actions.