Logic programming

Introducing GOLOG and FLUX

Michael Ruster

Research lab – Summer term 2014

University Koblenz-Landau

Outline

- Situation Calculus
- GOLOG
- FLUX
- Conclusion

Outline

- Situation Calculus
- GOLOG
- FLUX
- Conclusion

Situation calculus

- McCarthy and Hayes (1969), Reiter (1991)
- First-order logic
- Dynamic world encoded with second-order logic

Situation calculus
Elements

- **Fluents**
  - Model properties of the world
- **Actions**
  - Execution of actions may change the world
- **Situations**
  - Are a history of action executions
  - There exists one initial situation s_0

Situation calculus
Fluents

- **Relational fluents**
  - When evaluated can be true or false
  - e.g. $\mathrm{hasCoffee}(p, s)$
    with p being a person and s a situation
- **Functional fluents**
  - May return a value
  - e.g. $\mathrm{location}(p, s)$ returns coordinates (x, y)

Situation calculus
Actions (1/2)

- **Action precondition axioms**
  - Describe when an action is executable with the predicate $\mathrm{Poss}(a, s)$
  - For example:

$$\mathrm{Poss}(\mathrm{pourCoffee}(p), s) \Leftrightarrow \neg \mathrm{hasCoffee}(p, s)$$

- Executing an action alters the situation:

$$\mathrm{do}(a, s) \to s'$$

Situation calculus
Actions (2/2)

- **Action effect axioms**
  - Describe the effects of actions on the world
  - For example:

$$\mathrm{Poss}(\mathrm{pourCoffee}(p), s)$$
$$\rightarrow \mathrm{hasCoffee}(p, \mathrm{do}(\mathrm{pourCoffee}(p), s))$$

  - **Frame problem**: What are the non-effects of actions? Has $\mathrm{location}(p, s)$ changed?

Situation calculus
Successor state axiom (1/2)

- Defining for every fluent how every action may or may not affect it: $\mathcal{O}(A * F)$
- Instead define for every action all effects it can have (Reiter (1991)): $\mathcal{O}(A * E)$

Situation calculus
Successor state axiom (2/2)

$$\mathrm{Poss}(a, s) \rightarrow \left[\mathrm{F}(\mathrm{do}(a, s)) \right.$$
$$\left. \Leftrightarrow \gamma_{\mathrm{F}}^{+}(a, s) \vee \mathrm{F}(s) \wedge \neg\gamma_{\mathrm{F}}^{-}(a, s)\right]$$

$\mathrm{F}(\mathrm{do}(a, s))$ = fluent is true after action

$\gamma_{\mathrm{F}}^{+}(a, s)$ = action made fluent true

$\mathrm{F}(s) \wedge \neg\gamma_{\mathrm{F}}^{-}(a, s)$ = fluent was true beforehand and is unaffected by action

Situation calculus
Example

$$\text{Poss}(\text{pourCoffee}(p), s) \Leftrightarrow \neg\text{hasCoffee}(p, s)$$

$$\text{Poss}(\text{sing}) \Leftrightarrow \top$$

$$\begin{aligned}\text{Poss}(a, s) \rightarrow \Big[&\text{hasCoffee}(p, \text{do}(a, s)) \\ &\Leftrightarrow [a = \text{pourCoffee}(p)] \\ &\quad \vee [\text{hasCoffee}(p, s) \wedge a \neq \text{pourCoffee}(p)]\Big]\end{aligned}$$

Outline

- Situation Calculus
- GOLOG
- FLUX
- Conclusion

GOLOG

- Builds on situation calculus
- Adds complex actions like
  - Loops
  - Conditions and tests
  - Non-deterministic procedures
- **Regression-based**: deciding whether an action is executable is only possible after looking at all previous actions

GOLOG
Example (1/2)

- Extend the situation calculus fluents, action preconditions and successor state axiom with GOLOG procedures:

$$\textbf{proc}\ \texttt{pourSOCoffee}\ (\boldsymbol{\pi} p)\ [\neg \mathrm{hasCoffee}(p)\mathbf{?};\ \mathrm{pourCoffee}(p)]\ \textbf{endProc}$$

$$\textbf{proc}\ \texttt{control}\ [\textbf{while}\ (\exists p)\neg \mathrm{hasCoffee}(p)\ \textbf{do}\ \texttt{pourSOCoffee}(p)\ \textbf{endWhile}];\ \mathrm{sing}\ \textbf{endProc}.$$

GOLOG
Example (2/2)

- Initial configuration:

$$\neg \text{hasCoffee}(p, s_0) \Leftrightarrow p = \text{Miriam} \vee p = \text{Sergey}.$$

- Two possible results:

$$s = \text{do}\Big(\text{sing}, \text{do}\big(\text{pourCoffee}(\text{Miriam}),$$
$$\text{do}(\text{pourCoffee}(\text{Sergey}), s_0)\big)\Big)$$
$$s = \text{do}\Big(\text{sing}, \text{do}\big(\text{pourCoffee}(\text{Sergey}),$$
$$\text{do}(\text{pourCoffee}(\text{Miriam}), s_0)\big)\Big)$$

GOLOG
Problems

- Complete knowledge in initial situation assumed
- Internal reactions on sensed action and acting on it is missing
- Exogenous events not handled
- Reasoning takes exponentially longer over time due to being regression-based

Outline

- Situation Calculus
- GOLOG
- FLUX
- Conclusion

FLUX

- Uses fluent calculus
- Supports incomplete descriptions of the world
- Offers a solution to knowledge reasoning and sensing
- Reasoning is linear in the size of a state representation

FLUX
States

- A state z is a set of fluents $f_1, \ldots, f_n$ denoted as: $z = f_1 \circ \ldots \circ f_n$
- There is only one state in a situation
- The world can be in the same state in multiple situations
- Agents have their own state representing their knowledge $\mathrm{KState}(s, z)$
  - The agent knows that z holds in s

FLUX
State update axiom

- Solves frame problem
- Defines effects of an action as the difference between the state before and after the action
- Uses negative and positive effects of actions ϑ^- and ϑ^+ respectively
  - Both are macros for finite states

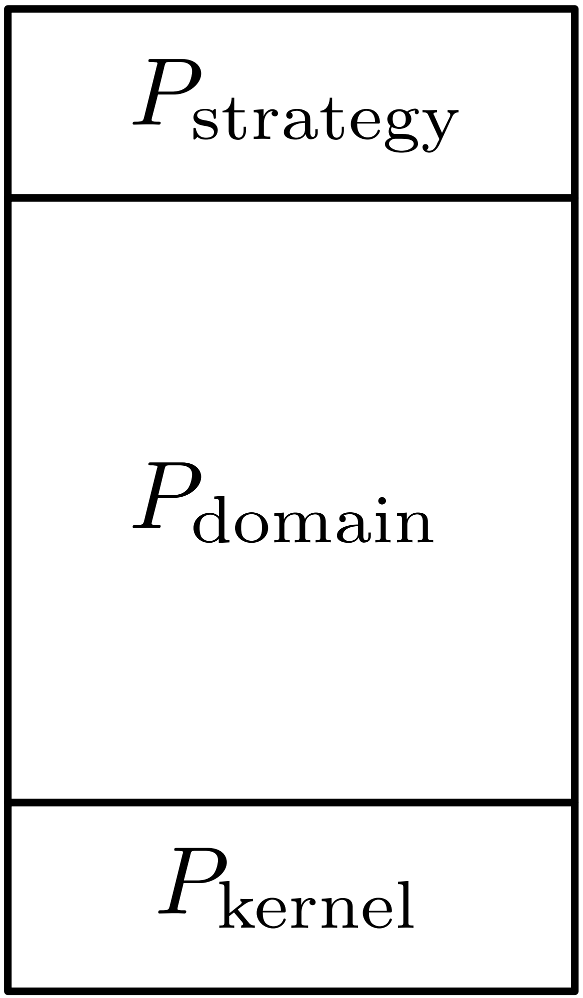
FLUX

Constraint solver

- Constraints model negative and disjunctive state knowledge
- Constraint solver uses **constraint handling rules** to rewrite constraints into simpler ones until they are solved
- $H_1, \ldots, H_m \Leftrightarrow G_1, \ldots, G_k \mid B_1, \ldots, B_n$
  - When the guard can be derived
  - The head gets replaced by the body

FLUX

Program structure



- Programmer defined intended agent behaviour
- Domain encodings
  - Initial knowledge state & domain constraints
  - Action precondition axioms & state update axioms
- Constraint system and constraint solver

FLUX
Example (1/3)

```
perform(sing, []).
poss(sing, Z) :- all_holds(hasCoffee(_), Z).
state_update(Z, sing, Z, []).

perform(pourCoffee(P), []).
poss(pourCoffee(P), Z) :-
      member(P,[miriam,sergey]),
      not_holds(hasCoffee(P), Z).
state_update(Z1, pourCoffee(P), Z2, []) :-
      update(Z1, [hasCoffee(P)], [], Z2).
```

FLUX

Example (2/3)

```
main_loop(Z) :-
   poss(sing, Z)
      -> execute(sing, Z, Z);
   poss(pourCoffee(P), Z)
      -> execute(pourCoffee(P), Z, Z1),
         main_loop(Z1);
   false.
```

FLUX
Example (3/3)

```
init(Z0) :-
        not_holds(hasCoffee(miriam), Z0),
        not_holds(hasCoffee(sergey), Z0).
```

The final state will be:

```
Z = [hasCoffee(sergey), hasCoffee(miriam)]
```

Outline

- Situation Calculus
- GOLOG
- FLUX
- Conclusion

Conclusion

- GOLOG does not satisfy our demands without modifications and extensions
- FLUX is applicable to a multi-agent scenario as shown e.g. by Schiffel and Thielscher (2007)

Questions?

References

- 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 27, 359–380.
- McCarthy, J., Hayes, P., 1968. Some philosophical problems from the standpoint of artificial intelligence. Stanford University USA.
- Levesque, H.J., Reiter, R., Lesperance, Y., Lin, F., Scherl, R.B., 1997. GOLOG: A logic programming language for dynamic domains. The Journal of Logic Programming 31, 59–83.

References

- Papataxiarhis, V., 2006. Situation Calculus.
- Thielscher, M., 2005. FLUX: A logic programming method for reasoning agents. Theory and Practice of Logic Programming 5, 533–565.
- Levesque, H.J., Reiter, R., Lesperance, Y., Lin, F., Scherl, R.B., 1997. GOLOG: A logic programming language for dynamic domains. The Journal of Logic Programming 31, 59–83.
- Thielscher, M., 1999. From situation calculus to fluent calculus: State update axioms as a solution to the inferential frame problem. Artificial intelligence 111, 277–299.

References

- Schiffel, S., Thielscher, M., 2006. Reconciling situation calculus and fluent calculus, in: AAAI. pp. 287–292.
- Frühwirth, T., 1998. Theory and practice of constraint handling rules. The Journal of Logic Programming 37, 95–138.
- Schiffel, S., Thielscher, M., 2007. Multi-agent FLUX for the gold mining domain (system description), in: Computational Logic in Multi-Agent Systems. Springer, pp. 294–303.