# Controlling Semi-Automatic Systems with FLUX

Michael Thielscher

Dresden University of Technology
mit@inf.tu-dresden.de

The programming of agents that are capable of reasoning about their actions is a major application of logic programming in Artificial Intelligence. FLUX is a recent, constraint logic programming-based method for designing reasoning agents that can sense, act, and plan under incomplete information [3]. An agent program in this language consists of a background theory, which endows the agent with the necessary knowledge of its actions, along with a high-level acting and planning strategy. The reasoning facilities are provided by a constraint solver, which is formally based on the action theory of the fluent calculus and its solution to the frame problem under incomplete states.

We have extended FLUX by a method that allows agents to reason, plan and act in semi-automatic environments, in which actions can initiate whole chains of indirect or delayed effects. Our approach addresses complex issues such as simultaneous, additive changes of state variables under incomplete information. As a case study, we have developed a control program for a complex dynamic environment of a steam boiler. A model of a real system, the domain involves uncertainty in form of varying output of the water pumps, which requires the agent to reason and plan under incomplete information. Moreover, simple actions in this system, such as turning off a pump, may trigger a whole chain of indirect effects: Another pump may automatically change to high capacity, which in turn may cause a valve to open, etc. Furthermore, every one of these changes has a specific effect on the water level in the boiler, which in turn affects the quantity of steam that is produced.

As an innovative and versatile way of modelling additive fluents and delayed effects, we introduce the notion of a **momentum fluent**. Such a fluent describes a property of the momentum in a physical system rather than a static property. An action may cause several momentum fluents to become true, each of which represents a specific contribution to the same fluent (such as the water level in the steam boiler). Additive fluents can thus be modelled. In a similar fashion, an action which has a delayed effect can be specified as bringing about a momentum fluent that eventually causes the actual delayed effect. Having triggered this effect, the momentum fluent itself may either automatically terminate, or continue to hold in case of recurring effects. On the other hand, the agent may have at its disposal the intervening action of terminating the momentum fluent before it produces the delayed effect.

The ramifications of an action in a semi-automatic environment are the consequence of causal connections among the components. In general, the occurrence of an indirect effect depends on the both the current state and the ef-

fects that have already been caused. To specify the causal relations among the components of a dynamic system, we have extended FLUX by the predicate `causes(Z1,P1,N1,Z2,P2,N2)`. Its intuitive meaning is that if positive and negative effects `P1` and `N1`, respectively, have just occurred in state `Z1`, then this causes an automatic update to state `Z2` with positive and negative effects `P2` and `N2`, respectively.

On the basis of the individual causal relationships of a domain, the indirect consequences of an action are inferred as "causal chains." To this end, the causal relationships together are viewed as a graph in which each node represents a state and each edge represents the transformation from one state into another as a consequence of a single indirect effect. Sinks in this graph are nodes which have no outgoing edges, thus representing a stable state that admits no (further) indirect effect. To infer all indirect effects of an action one therefore has to find a sink, starting in the node which represents the status of the environment after the direct effect of the respective action. To this end, we have defined the new FLUX predicate `ramify(Z1,P,N,Z2)` for updating a (possibly incomplete) state `Z1` by positive and negative effects `P` and `N`, respectively, and then automatically leading to a sink `Z2` through a chain of causally triggered transitions.

The formal underpinnings of our method are given by a solution to the ramification problem in the action theory of fluent calculus [2]. Using the notion of causal propagation, this extensive solution accounts for mutually dependent components (such as connected controls of pumps) and multiple changes of state variables (such as needed for additive changes). In comparison with previous approaches of modelling semi-automatic environments using solutions to the ramification problem [4, 1], our method addresses complex issues such as simultaneous, additive changes of state variables and delayed effects. Moreover, our solution has been embedded in programming language for agents that reason about their actions and sensor information and that can plan under incomplete information.

The full paper as well as the FLUX program for steam boiler control are available for download at our web site

`fluxagent.org`

## References


1. S. McIlraith. An axiomatic solution to the ramification problem (sometimes). *Artificial Intelligence*, 116(1–2):87–121, 2000.
2. M. Thielscher. Ramification and causality. *Artificial Intelligence*, 89(1–2):317–364, 1997.
3. M. Thielscher. FLUX: A Logic Programming Method for Reasoning Agents. *Theory and Practise of Logic Programming*, 2004.
4. R. Watson. An application of action theory to the space shuttle. In *Proc. of PADL*, vol. 1551 of *LNCS*, 290–304, 1998.