

Preface

The creation of intelligent robots is surely one of the most exciting and challenging goals of Artificial Intelligence. A robot is, first of all, nothing but an inanimate machine with motors and sensors. In order to bring life to it, the machine needs to be programmed so as to make active use of its hardware components. This turns a machine into an autonomous robot. Since about the mid nineties of the past century, robot programming has made impressive progress. State-of-the-art robots are able to orient themselves and move around freely in indoor environments or negotiate difficult outdoor terrains, they can use stereo vision to recognize objects, and they are capable of simple object manipulation with the help of artificial extremities.

At a time where robots perform these tasks more and more reliably, we are ready to pursue the next big step, which is to turn autonomous machines into **reasoning robots**. A reasoning robot exhibits higher cognitive capabilities like following complex and long-term strategies, making rational decisions on a high level, drawing logical conclusions from sensor information acquired over time, devising suitable plans, and reacting sensibly in unexpected situations. All of these capabilities are characteristics of human-like intelligence and ultimately distinguish truly intelligent robots from mere autonomous machines.

What are Robotic Agents?

A fundamental paradigm of Artificial Intelligence says that higher intelligence is grounded in a mental representation of the world and that intelligent behavior is the result of correct reasoning with this representation. A **robotic agent** is a high-level control program for a robot—or, for that matter, for a proactive software agent—in which such mental models are employed to draw logical conclusions about the world. Intelligent robots need this technique for a variety of purposes:

- Reasoning about the current state.

What follows from the current sensor input in the context of the world model?

- Reasoning about action preconditions.

Which actions are currently possible?

- Reasoning about effects.

What holds after an action has been taken?

- Planning.

What needs to be done in order to achieve a given goal?

- Intelligent troubleshooting.

What went wrong and why, and what could be done to recover?

Research on how to design an automatic system for reasoning about actions has a long history in Artificial Intelligence. The earliest formal model for the ability of humans to solve problems by reasoning has been the so-called situation calculus, whose roots go back to the early sixties. In the late sixties this model has been used to build an automatic problem solver. However, this first implementation did not scale up beyond domains with a small state space and just a few actions because it suffered from what soon became a classic in Artificial Intelligence, the so-called frame problem. In a nutshell, the challenge is to describe knowledge of effects of actions in a succinct way so that an automatic system can efficiently update an internal world model upon the performance of an action. The frame problem has haunted researchers for many years, and only in the early nineties the first satisfactory solutions have emerged. These formal models for reasoning about actions are now being developed into actual programming languages and systems for the design of robotic agents.

One successful approach to the frame problem is provided by a formalism known as **fluent calculus**. This book is concerned with this model of rational thought as a way to specify mental models of dynamic worlds and to reason about actions on the basis of these models. Very recently the calculus has evolved into the programming method and system **FLUX**, which supports the problem-driven, top-down design of robotic agents with the cognitive capabilities of reasoning, planning, and intelligent troubleshooting.

Why Fluent Calculus?

Fluent calculus originates in the classical situation calculus. It provides the formal underpinnings for an effective and computationally efficient solution to the fundamental frame problem. To this end, fluent calculus extends situation calculus by the basic notion of a state, which allows to define effects very naturally in terms of how an action changes the state of the world. Based on classical predicate logic, fluent calculus is a very versatile formalism, which captures a variety of phenomena that are crucial for robotic agents, such as incomplete knowledge, nondeterministic actions, imprecise sensors and effectors, and indirect effects of actions as well as unexpected failures when an action is performed in the real, unpredictable world.

Why FLUX?

FLUX is a Prolog-based method for programming robotic agents based on the expressive action theory of fluent calculus. It comprises a way of encoding incomplete world models along with a technique for updating these models according to a declarative specification of the acting and sensing capabilities of a robot. Using a powerful constraint solver, a generic base system provides general reasoning facilities, so that the agent programmer can focus on specifying the application domain and designing the intended high-level behavior. Allowing for concise programs and supporting modularity, FLUX is eminently suitable for programming complex strategies for robotic agents. Thanks to a restricted expressiveness and a sound but incomplete inference engine, the system exhibits excellent computational behavior. In particular, it scales up well to long-term control thanks to the underlying principle of “progression,” which means to continually update a (possibly incomplete) world model upon the performance of an action. Appealing to a declarative programming style, FLUX programs are easy to write, understand, and maintain. The book includes the details of the base system written in Prolog, so that it can be easily adapted and extended according to one’s own needs.

Further Aspects

This introductory motivation would not be complete without the admission that the current state-of-the-art in research on reasoning robots still poses fundamentally unsolved problems. Maybe the most crucial issue is the interaction between cognitive and low-level control of a robot, in particular the question about the origin of the symbols, that is, the names for individuals and categories that are being used by a robotic agent. In this book we take a pragmatic, top-down approach, which requires the designer of a system to predefine the grounding of the symbols in the perceptual data coming from the sensors of a robot. Ultimately, a truly intelligent robot must be able to handle this symbol grounding problem by itself in a more flexible and adaptive manner. Another important issue is the lack of self-awareness and true autonomy. The reasoning robots considered in this book are able to follow complex strategies and they are capable of devising and executing their own plans. Still both the intended behavior and the boundaries for the plans are determined by the programmer. Ultimately, a truly self-governing robot must be able to reflect and adapt its behavior when facing new and unforeseen situations.

What’s in this Book?

This book provides an in-depth and uniform treatment of fluent calculus and FLUX as a mathematical model and programming method for robotic agents. As theory and system unfold, the agents will become capable of dealing with incomplete world models, which require them to act cautiously under uncertainty; they will be able to explore unknown environments by logically reasoning about

sensor inputs; they will plan ahead some of their actions and react sensibly to action failure.

The book starts in Chapter 1 with an introduction to the axiomatic formalism of fluent calculus as a method both for specifying internal models of dynamic environments and for updating these models upon the performance of actions. Based on this theory, the logic programming system FLUX is introduced in Chapter 2 as a method for writing simple robotic agents. The first two chapters are concerned with the special case of agents having complete knowledge of all relevant properties of their environment. In Chapters 3 and 4, theory and system are generalized to the design of intelligent agents with **incomplete knowledge** and which therefore have to act under uncertainty. Programming these agents relies on a formal account of knowledge given in Chapter 5, by which conditions in programs are evaluated on the basis of what an agent knows rather than what actually holds. Chapter 6 is devoted to **planning** as a cognitive capability that greatly enhances flexibility and autonomy of agents by allowing them to mentally entertain the effect of different action sequences before choosing one that is appropriate under the current circumstances. Chapters 7 and 8 are both concerned with the problem of **uncertainty** due to the fact that effects of actions are sometimes unpredictable, that state properties in dynamic environments may undergo changes unnoticed by an agent, and that the sensors and effectors of robots are always imprecise to a certain degree. Chapter 9 deals with a challenge known as the **ramification problem**. It arises in complex environments where actions may cause chains of indirect effects, which the agent needs to take into account when maintaining its internal world model. Chapter 10 is devoted to an equally important challenge known as the **qualification problem**. It arises in real-world applications where the executability of an action can never be predicted with absolute certainty since unexpected circumstances, albeit unlikely, may at any time prevent the successful performance of an action. The solution to this problem enables agents to react to unexpected failures by generating possible explanations and revising their intended course of actions accordingly. Finally, Chapter 11 describes a **system architecture** in which robotic agents are connected to a low-level, reactive control layer of a robot. The appendix contains a short user manual for the FLUX system.

The only prerequisite for understanding the material in this book is basic knowledge of standard first-order logic. Some experience with programming in Prolog might also be helpful. Sections marked with * contain technical details that may be skipped at first reading or if the reader's main interest is in programming. The book has its own webpage at

www.fluxagent.org

where the FLUX system and all example programs in this book are available for download.

Acknowledgments

This book would not have been possible without the help of many people who contributed in various ways to its success. The author wants to especially thank Wolfgang Bibel, Matthias Fichtner, Norman Foo, Michael Gelfond, Axel Großmann, Sandra Großmann, Birgit Gruber, Yi Jin, Matthias Knorr, Markus Krötzsch, Yves Martin, Maurice Pagnucco, Martin Pitt, Ray Reiter, Erik Sandewall, and Stephan Schiffel.