Supporting the exploratory nature of simulations in D-MASON

Gennaro Cordasco[1], Rosario De Chiara[2],
Fabio Fulgido[3], and Mario Fiore Vitale[3]

[1] Dipartimento di Psicologia
Seconda Università degli Studi di Napoli, Italy
gennaro.cordasco@unina2.it
[2] Centro Ricerca e Sviluppo
Poste Italiane, Italy
dechia24@posteitaliane.it
[3] ISISLab - Dipartimento di Informatica
Università di Salerno, Italy
{f.fulgido,mvitale86}@gmail.com

Abstract. Agent-Based Models (ABM) are a class of models which, by simulating the behaviors of multiple agents (i.e., independent actions, interactions and adaptation), aims to emulate and/or predict complex phenomena. The "emergence" of such complex phenomena is often computation intensive and requires tools, libraries and frameworks, capable of speeding up and facilitate the design of complex simulations.
In this paper we present new developments on D-MASON, that is a distributed version of MASON, a well-known and popular library for writing and running Agent-based Simulations.
The new developments are: a) a tool that allows the parallel exploration of the behavior parameter space; b) an infrastructure that improves the management of distributed simulations in terms of easy deployment of new simulations, automatic update, versioning control and distributed logging.

Keywords: Agent-Based Simulation, Distributed Systems, System Management.

1 Introduction

Agent-based modeling is a style of modeling in which multiple agents and their interactions with each other and their environment are explicitly represented in a program with the aims of understanding and generating complex phenomena. An agent-based model (ABM) consists of: a set of agents, a set of agent relationships and an environment to host the agents and where they take action. ABMs have become very popular in various research fields, such as biology[25], ecology, economics[1], political science, social science[15] etc.. A fundamental benefit of ABMs is the discovery and explanation of emergent behavior. Emergent behaviors of complex systems are patterns that do not depend on individual components of a system but are generated by their interaction to one another. For instance, emergent behavior is ubiquitous in biological systems (chemical

interactions among cells) and in groups of animals (flocking of birds and schooling of fish).

In [12] the very nature of simulations implemented by a computer program is described, such programs are involved in an exploratory process where the model is iteratively refined, as the problem it is meant to be simulated is better understood. For this reason the framework which is used must provide facilities to debug the model, support the re-runs of experiments and to the collection of the generated data.

A common approach for improving the efficiency of ABM is to distribute the overall computation on a number of *Logical Processors* (LPs). Parallelization and distribution of the computational load has two important motivations in the Agent-Based Modeling field:

- the *speed* running a model faster allows, in the same time, to make more thorough exploration of the (possible) complex behavior. An efficient simulator should allow to run experiments long enough to make emergent behaviours evident;
- the *size* of the models can be crucial to ensure that the emerging behavior is indeed shown and made evident; when models are too small, some of the effects might be hidden, not evident and/or emerge very late.

Several parallel implementation of ABMs frameworks have been proposed [5, 6, 8, 19].

A good parallel implementation should face several conflicting issues: (a) balance the overall load distribution among the LPs; (b) minimize the communication overhead due to the interdependencies existing between the tasks executed on the LPs; (c) synchronize the evolution of the simulation among the LPs that provide the computing power.

The general behaviors of a simulation associated to an ABM can be quite complex and therefore it is not possible to apply analytical methods to understand what is happening. This is particularly true when one looks at the parameters' space of an ABM: a common approach is to sample the parameter space in order to figure out the behavior of the model over a wide range of different conditions.

Speeding up such exploration is important for the researcher because the emergence of some phenomena might be evident just for some specific subsets of the parameters: small changes made to a single parameter may lead to a radical modification of the dynamics of the whole system.

In this paper we present a tool that increases the support for the exploratory nature of simulations by implementing a parallel behavioral parameter space exploration. Indeed a parallel architecture can be easily exploited by letting multiple simulations run at once with different parameters distribution on multiple machines [3].

An alternative to parallel behavioral parameter space exploration is proposed by the authors in [11]. They run a simulation and store information about its states and transitions between them. Under the assumption that multiple runs are often similar, they try to save computing time by reusing event computations done during the first simulation. This approach requires to build an *updateable* model for the simulation, along with a comparison predicate in order to evaluate if two states of a simulation are similar (i.e. applying the same transition to both leads to the same destination state, and therefore

the previously computed destination state can be reused) or different (a new destination state have to be computed and the simulation needs to be *updated*). This kind of framework seems to be effective for computing-intensive simulations but less adaptable to agent-based massive simulations, due to the difficulty to build a comparison predicate and to the large memory footprint needed to store the state of such massive simulation.

2 Distributed Simulation and D-MASON

Some aspects of ABMs implementation, like the parameter space exploration, can be effectively parallelizable. Other aspects, like running massive simulation (i.e., simulating a large number of agents) and/or simulations which deal with complex agents, that is, computationally intensive agents, are more difficult to be parallelizable. A common approach to the parallelization of massive simulations is just an instance of a more general problem of parallelizing a sequential computation by dividing it into smaller computations (subtasks) and assigning them to different processes for parallel executions. This two phases represent two key steps in the design of parallel algorithms [16]. The communication and the synchronization between the different subtasks are typically some of the greatest obstacles in getting good performances.

As an example a simple way to partition the whole computation into subtasks is to assign a fixed number of agents to each available LP. This approach named *agents partitioning* enables a balanced workload distribution but usually introduces a significant communication overhead (all–to–all communications are required to synchronize the simulation).

Other strategies which partition the work in a smarter way [7] have been proposed in order to reduce the time due to the communication and synchronization between processes.

Agent-Based Simulations are designed accordingly to a discrete-event paradigm where the simulation time is split in steps named *simulation steps*, often but not necessarily, of the same duration. So, as the simulation advances, the step number increases.

We used D-MASON [6] a general purpose framework for discrete-event ABMs. D-MASON enlists the aid of two or more LPs to carry out the simulation: each of the LPs is derived from MASON and is in charge of a portion of the set of the agents.

MASON [17, 18] toolkit is a discrete-event simulation core and visualization library written in Java, designed to be used for a wide range of ABMs. The toolkit is composed of two independent layers: the *simulation* layer and the *visualization* layer. The main reasons that suggested the development of a distributed version of MASON are:

- MASON is one of the most expressive and efficient library for ABMs (as reported by many reviews [2, 20, 22]);
- MASON architecture, that clearly separates visualization by simulation, greatly helped during the process of development of a distributed version of the library[17, 18];
- the significant amount of research and simulations already present in MASON, which can be easily ported to D-MASON, makes it particularly cost effective for the scientists.

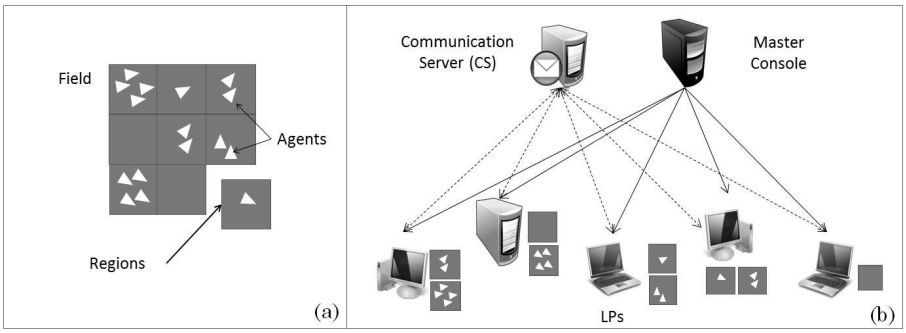

Fig. 1. The Architecture of D-MASON.

Considering that most ABMs are inspired by natural models where agents are placed in a bi-dimensional (or tri-dimensional) space (some examples are [10, 13, 14, 21, 24]) and agents' communications/interactions are limited to a delimited area of interest (AOI), D-MASON exploits a space partitioning approach: the space to be simulated (the field) is partitioned into regions (see Fig. 1.(a)). Each region, together with the agents it contains, is assigned to a LP; each LP is in charge of: simulating the agents that belong to the assigned region, handling the migration of agents between regions and managing the synchronization between neighboring regions (see Fig. 1.(b)).

Since the AOI extension of an agent is small compared with the size of a region it belongs to, the communication is often limited to local messages within the same LP. LPs in charge of the simulation of neighboring regions, are locally synchronized in order to let the simulation run consistently. LPs communicate by using a publish–subscribe mechanism design pattern: a multicast channel is assigned to each region.

D-MASON is released under a Free and Open Software license and is available at [9].

3 D-MASON enhancements

In this section we describe the extensions of the D-MASON system management. These extensions add new features to D-MASON aimed at improving the usability of the environment for the researcher who is designing a simulation. We extended D-MASON by adding two facilities: a tool for the parallel exploration of the behavior parameter space and an infrastructure that improves the management of distributed simulations.

3.1 Exploring parameter space

As observed in Section 1, the exploration of the parameters' space requires the execution of a large number of simulations with different parameters distributions [3, 23]. These executions are independent therefore can be safely carried out in parallel.

In the following we present a semi-automatic tool that supports such exploration. The tool performs three functionalities: selection of parameters, management of parameters and execution.

Selection of parameter. D-MASON allows the programmer to select, via Java *annotations*, which parameter of the simulation can be explored. The programmer can also annotate, for each parameter, its domain and/or its suggested value. A portion of an annotated source Java file is in Figure 2. In this specific case, both the parameters `width`, `height` and `numFlockers` are annotated using the keyword `@batch`. Moreover for the parameter `height` the programmer has also defined the domain ($100 - 300$) and the suggested value (250).

```
public class DFlockers extends DistributedState<Double2D>
{
    private static final long serialVersionUID = 1L;
    public DContinuous2D flockers;
    private static boolean isToroidal=true;
    @batch
    public double width = 150;
    @batch(
        domain = "100−300",
        suggestedValue = "250"
    )
    public double height = 150;
    @batch
    public int numFlockers = 20;
   ...
}
```

Fig. 2. A portion of a source file with annotations.

Management of parameters. D-MASON includes a tool, named *Batch Wizard*, which takes in input the `.jar` file of a specific D-MASON simulation and extracts the list of annotated parameters. Then, by using the application depicted in Figure 3, the user is able to define for each parameter the set of values to be analyzed (to explore the parameter space) and the number of runs to be executed for each fixed set of parameter (to evaluate the stability of the model). Each parameter can be explored in four different ways:

Fixed: the parameter has a single fixed value, which can be the one suggested by the programmer by annotations or another value chosen by the user.

By values: the user provides a set of values to be evaluated.

Range: the user defines a range of values by providing a lower endpoint, an increment and the upper endpoint.

Distribution: the values of the parameter are picked from a probability distribution. In this case the number of runs is used to pick different values. Three probability distributions are currently available: *uniform*, *normal* and *exponential*. For each probability distributions, its parameters (for instance, mean and standard deviation for the normal distribution) can also be selected by the user.

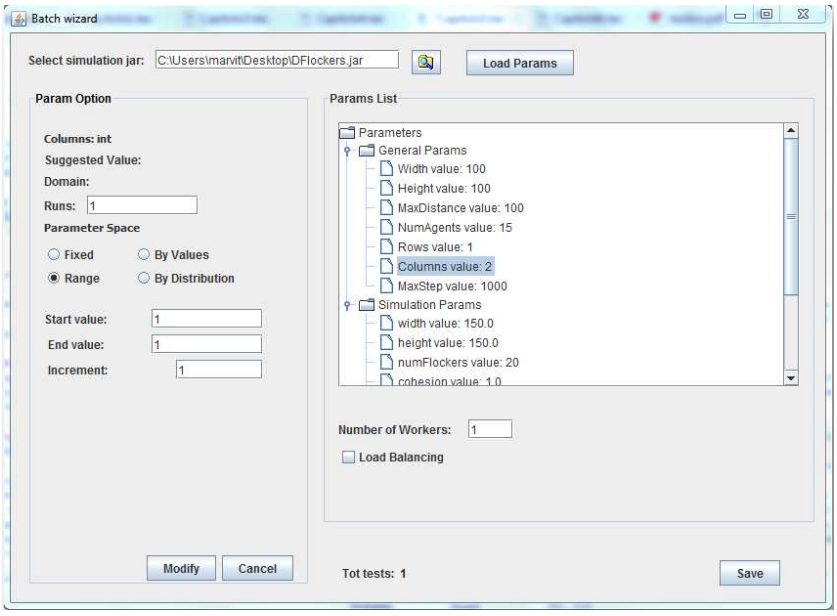

Fig. 3. The management of parameters via the Batch Wizard

The last two options (Range and Distribution) are available only for numeric parameters. During the setup phase of the simulation the tool provides to the user the number of tests necessary, for the exploration of the parameters, in the current configuration.

Other options are the number of LPs n_{LP} for each run and the possibility of enabling or not the load balancing functionality, see [4]. The output of the Batch Wizard is an XML file similar to the one in Figure 4.

```
<?xml version="1.0" encoding="UTF−8" ?>
<Batch>
  <simulationName>vampires.jar</simulationName>
  <neededWorkers>10</neededWorkers>
  <isBalanced>false</isBalanced>
  <simulationParams>
    <paramFixed name = "numRoosts" type = "int" runs = "10" mode = "fixed">
      <value>10</value>
    </paramFixed>
     <paramList name = "numBats" type = "int" runs = "5" mode = "list">
      <item>200</item>
      <item>2000</item>
    </paramList>
  ...
</Batch>
```

Fig. 4. An example of XML file generated by the Batch Wizard.

Execution. At the execution time the XML file described above is read and validated and the system prepares the sets of simulations to run. The system builds two sets: a *test list* which is the list of all the configurations to be executed, and a *LPs list* which is a partition of available LPs, such that each set in the list contains at least n_{LP} LPs. The user can choose whether the execution must be sequential or parallel. During a parallel batch execution each set of LPs, in the LPs list, executes at the same time a different test. However, different runs will use, concurrently, a single communication server (CS).

The execution of a batch is performed by several *Batch Executors* (on sequential batch executions a single Batch Executor is used). A Batch Executor is in charge of starting and stopping a single run of a simulation. Briefly, a batch executor manages a set of LPs and as soon as the execution of the simulation terminates, it picks another test from the test list and starts its execution (see Figure 5).

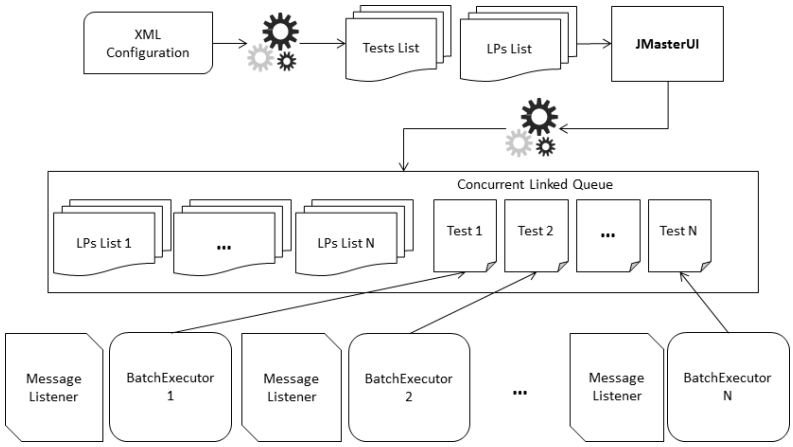

Fig. 5. Batch execution

3.2 A novel Management Infrastructure

We present now a novel infrastructure that improves the management of distributed simulations in terms of easy deployment of new simulations, automatic update, versioning control and distributed logging. The novel architecture is depicted in Figure 6. An FTP Server has been added to the system with the goal of enabling the exchange of files between the console (master) and the LPs. The use of the FTP server is explained in the following.

Deploy of new simulations and LP software update. First of all the FTP server is exploited for the deployment of new simulations. In the previous version of D-MASON (v. 2.0) , simulations were encoded within the D-MASON package, and consequently the development of a new simulation required to rebuild the entire package and restarting the entire system (i.e. Console, LPs and CS). In this enhanced version of D-MASON(v.

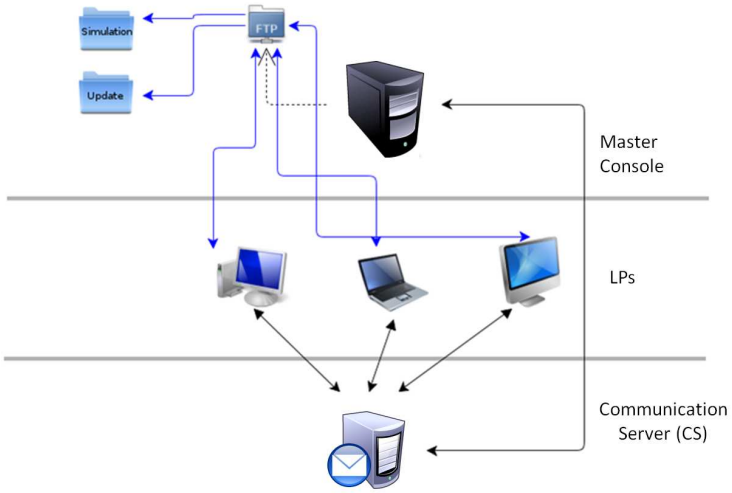

Fig. 6. The novel infrastructure

2.1), new simulations are compiled separately in a `.jar` file and are added to the system via the Console application (see Figure 7). The `.jar` file and a corresponding digest file are store into a specific directory of the FTP Server. Then the Console application sends, via the CS, a message to the LPs, containing the name of the simulation file, the address and the port of the FTP server. Each LP then check checks whether it already has that specific simulation and, if so, it checks, using the digest file, if it has the correct version. If the LP does not have the simulation file, or if the digest files do not match (that is, the new simulation file is an update of a previous version), then the LP download the simulation file from the FTP server and, in the case of update, it replaces the previous file.

A similar approach is used to update the software running on each LP. Since the upgrade processes affects only the LPs connected to the system, we have also developed a mechanism that allows each LP, during the connection to the system, to check whether there have been updates or new simulation files have been deployed. Moreover, before starting a new simulation the console checks that all the LPs are aligned to the same software version and the same simulation file.

Distributed Logging. The novel management infrastructure is also exploited to collect and aggregate the log files generated by each LP. When a simulation is completed, the Console sends a *gather* command to the LPs. When the LPs receive this command, they upload the log files on the FTP server, so that they can be aggregated into a single file that can be used for subsequent analysis.

4 Discussion and conclusion

This paper reports on an currently undergoing project, D-MASON, that has been developed with the purpose of speeding up the performances of MASON, a very well known and quite widespread framework for ABMs.

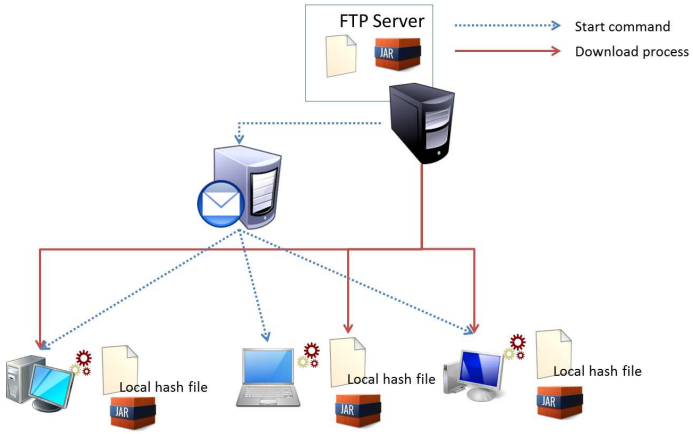

Fig. 7. Deploy of a new simulation.

This work has been motivated by the the need for a system management facility that is extremely important for the deployment, the tuning and the analysis of complex simulations on multiple machines.

We have shown a) a tool that allows the parallel exploration of the behavior parameter space, and b) a simple architecture that speeds the deployment and the analysis of distributed simulations.

References

1. Agents of Change. The economist, July 22nd, 2010.
2. Matthew Berryman. Review of Software Platforms for Agent Based Models. Technical Report DSTO-GD-0532, Australian Government, Department of Defence, 2008.
3. Benoît Calvez and Guillaume Hutzler. Parameter space exploration of agent-based models. In *Proceedings of the 9th international conference on Knowledge-Based Intelligent Information and Engineering Systems*, KES'05, pages 633–639, 2005.
4. Michele Carillo, Gennaro Cordasco, Rosario De Chiara, Francesco Raia, Vittorio Scarano, and Flavio Serrapica. Enhancing the Performances of D-MASON - A Motivating Example. In *SIMULTECH , pages 137-143. SciTePress*, pages 137–143, 2012.
5. Nicholson Collier and Michael North. Parallel agent-based simulation with Repast for High Performance Computing. *SIMULATION: Transactions of the Society for Modeling and Simulation International*, online:, 2012.
6. Gennaro Cordasco, Rosario De Chiara, Ada Mancuso, Dario Mazzeo, Vittorio Scarano, and Carmine Spagnuolo. Bringing together efficiency and effectiveness in distributed simulations: the experience with D-MASON. *SIMULATION: Transactions of The Society for Modeling and Simulation International*, online:, 2013.
7. Biagio Cosenza, Gennaro Cordasco, Rosario De Chiara, and Vittorio Scarano. Distributed Load Balancing for Parallel Agent-based Simulations. In *Proc. of the 19th Euromicro International Conference on Parallel, Distributed and Network-Based Computing (PDP 2011)*, 2011.

8. Christophe Deissenberg, Sander van der Hoog, and Herbert Dawid. Eurace: A massively parallel agent-based model of the european economy. *Applied Mathematics and Computation*, 204:541–552, 2008.
9. Distributed-Mason Project. http://www.isislab.it/projects/dmason/, 2011.
10. Piter Dykstra, Corinna Elsenbroich, Wander Jager, Gerard Renardel de Lavalette, and Rineke Verbrugge. Put your money where your mouth is: Dial, a dialogical model for opinion dynamics. *Journal of Artificial Societies and Social Simulation*, 16(3):4, 2013.
11. Steve L. Ferenci, Richard M. Fujimoto, Mostafa H. Ammar, Kalyan Perumalla, and George F. Riley. Updateable simulation of communication networks. In *Proceedings of the sixteenth workshop on Parallel and distributed simulation*, PADS '02, pages 107–114, Washington, DC, USA, 2002. IEEE Computer Society.
12. Nigel Gilbert and Klaus G Troitzsch. *Simulation for the Social Scientist*. Open University Press, 2005.
13. Max Hartshorn, Artem Kaznatcheev, and Thomas Shultz. The evolutionary dominance of ethnocentric cooperation. *Journal of Artificial Societies and Social Simulation*, 16(3):7, 2013.
14. Erez Hatna and Itzhak Benenson. The schelling model of ethnic residential dynamics: Beyond the integrated - segregated dichotomy of patterns. *Journal of Artificial Societies and Social Simulation*, 15(1):6, 2012.
15. Simon A. Levin Joshua M. Epstein and Series Editors Steven H. Strogatz, editors. *Generative Social Science: Studies in Agent-Based Computational Modeling*. Princeton University Press, 2007.
16. Zhiwei Xu Kai Hwang. *Scalable parallel computing: technology, architecture, programming*. WCB/McGraw-Hill, 1998.
17. Sean Luke, Claudio Cioffi-Revilla, Liviu Panait, and Keith Sullivan. MASON: A new multi-agent simulation toolkit. In *Proceedings of the SwarmFest Workshop, Ann Arbor (Michigan), USA. May 9-11*, 2004.
18. Sean Luke, Claudio Cioffi-Revilla, Liviu Panait, Keith Sullivan, and Gabriel Balan. MASON: A Multiagent Simulation Environment. *Simulation*, 81(7):517–527, 2005.
19. Mikola Lysenko and Roshan M. D'Souza. A framework for megascale agent based model simulations on graphics processing units. *Journal of Artificial Societies and Social Simulation*, 11(4):, 2008.
20. Robert Najlis, Marco A. Janssen, and Dawn C. Parkerx. Software tools and communication issues. In *Proc. Agent-Based Models of Land-Use and Land-Cover Change Workshop*, pages 17–30, 2001.
21. Amit Patel, Andrew Crooks, and Naoru Koizumi. Slumulation: An agent-based modeling approach to slum formations. *Journal of Artificial Societies and Social Simulation*, 15(4):2, 2012.
22. Steven F. Railsback, Steven L. Lytinen, and Stephen K. Jackson. Agent-based simulation platforms: Review and development recommendations. *Simulation*, 82:609–623, September 2006.
23. Attila Szab, Rajmund Bocsi, Gbor Ferschl, Balzs Blint, and Lszl Gulys. Experiments with complex agent-based systems using the meme tool: A case study. *ASME*, 2009.
24. Andrew White. An abstract model showing that the spatial structure of social networks affects the outcomes of cultural transmission processes. *Journal of Artificial Societies and Social Simulation*, 16(3):9, 2013.
25. William C. Wimsatt. *False models as means to truer theories*, pages 23 – 55. Oxford University Press, London, 1987 1987.