Do-Form: Enabling Domain Experts to use Formalised Reasoning

Manfred Kerber, Christoph Lange and Colin Rowat (editors)

AISB Convention 2013 • University of Exeter • 3rd–5th April, 2013

Foreword from the Convention Chairs

This volume forms the proceedings of one of eight co-located symposia held at the AISB Convention 2013 that took place 3rd-5th April 2013 at the University of Exeter, UK. The convention consisted of these symposia together in four parallel tracks with five plenary talks; all papers other than the plenaries were given as talks within the symposia. This symposium-based format, which has been the standard for AISB conventions for many years, encourages collaboration and discussion among a wide variety of disciplines. Although each symposium is self contained, the convention as a whole represents a diverse array of topics from philosophy, psychology, computer science and cognitive science under the common umbrella of artificial intelligence and the simulation of behaviour.

We would like to thank the symposium organisers and their programme committees for their hard work in publicising their symposium, attracting and reviewing submissions and compiling this volume. Without these interesting, high quality symposia the convention would not be possible.

Dr Ed Keedwell & Prof. Richard Everson
AISB 2013 Convention Chairs

Published by
The Society for the Study of Artificial Intelligence and the Simulation of Behaviour
http://www.aisb.org.uk

ISBN: 978-1-908187-32-1

Do-Form: Enabling Domain Experts to use Formalised Reasoning

Contents

1 Motivation **1**

2 Topics **2**

3 Programme **2**

4 Submission and Review Process **3**

5 Matchmaking **4**

6 Organisation **8**

1 Motivation

Do-Form is motivated by the long-term **vision** of making information systems dependable. In the past even mis-represented units of measurement caused fatal **engineering** disasters. In **economics**, the subtlety of issues involved in good auction design may have led to low revenues in auctions of public goods such as the 3G radio spectra. Similarly, banks' value-at-risk (VaR) models – the leading method of financial risk measurement – are too large and change too quickly to be thoroughly vetted by hand, the current state of the art; in the London Whale incident of 2012, JP Morgan claimed that its exposures were $67mn under one of its VaR models, and $129 under another one. Verifying a model's properties requires formally specifying them; for VaR models, any work would have to start with this most basic step, as regulators' current desiderata are subjective and ambiguous.

We believe that these problems can be addressed by representing the knowledge underlying such models and mechanisms in a formal, explicit, machine-verifiable way. Contemporary computer science offers a wide choice of knowledge representation languages well supported by verification tools. Such tools have been successfully applied, e.g., for verifying software that controls commuter rail or payment systems. Still, **domain experts** without a strong computer science background find it challenging to

choose the right tools and to use them. Do-Form aims at investigating ways to support them. Some problems can be addressed now, others will bring new challenges to computer science.

2 Topics

The call for papers listed the following topics of interest:

- for **domain experts**: what problems in application domains could benefit from better verification and knowledge management facilities? As possible fields we suggested:
  - Example 1 (economics): auctions, VaR, trading algorithms, market design
  - Example 2 (engineering): system interoperability, manufacturing processes, product classification
- for **computer scientists**: how to provide the right knowledge management and verification tools to domain experts without a computer science background?
  - wikis and blogs for informal, semantic, semiformal, and formal mathematical knowledge;
  - general techniques and tools for online collaborative mathematics;
  - tools for collaboratively producing, presenting, publishing, and interacting with online mathematics;
  - automation and computer-human interaction aspects of mathematical wikis;
  - ontologies and knowledge bases designed to support knowledge management and verification in application domains;
  - practical experiences, usability aspects, feasibility studies;
  - evaluation of existing tools and experiments;
  - requirements, user scenarios and goals.

3 Programme

The symposium is designed to bring domain experts and formalisers into close and fruitful contact with each other: domain experts will be able to present their fields and problems to formalisers; formalisers will be exposed to new and challenging problem areas. The programme combines talks, system demos, and tutorials to ensure close interaction among participants from both sides.

For the tutorials we are delighted to have three world-class economists, who will be presenting the following domains:

- **Matching markets** (**M. Utku Ünver**, Boston College): These include matching students to schools, interns to hospitals, and kidney donors to recipients. See the documentation for the 2012 Nobel Memorial Prize in Economic Sciences for more background information.[1]

- **Auctions** (**Peter Cramton**, University of Maryland): Peter has been working on auctions for Ofcom UK (4G spectrum auction), the UK Department of the Environment and Climate Change, and others – and most recently on the "applicant auctions" for the new top-level Internet domains issued by the ICANN.

- **Finance markets regulation** (**Neels Vosloo**, Financial Services Authority, UK): It is currently impossible for regulators to properly inspect risk management models. Test portfolios are a promising tool for identifying problems with risk management models. To what extent can techniques from mechanised reasoning automate some of the inspection process?

Further information about the speakers and their tutorials is available from the symposium homepage.

4 Submission and Review Process

We ran a novel two-stage submission process:

Stage 1: problem & tool ("nail & hammer") descriptions to be reviewed and matched with each other

Stage 2: regular extended abstracts, with a normal conference-like peer review

In stage 1 we solicited ...

- from **domain experts**: descriptions of canonical models and problems in their domain that might benefit from better verification and knowledge management facilities. Descriptions should focus on aspects of these models that domain users find particularly problematic, and suspect might be aided by formalisation tools

- from **computer scientists**: descriptions of formalisation, verification and knowledge management tools, with an emphasis on how they could be applied in a concrete real-world setting, or tailored to such application domains.

The symposium chairs, assisted by the PC members, reviewed and initially published commented versions of the stage 1 submissions on the symposium homepage (reproduced below), to provide orientation for stage 2. Where we identified matching problems and tools, we notified the respective authors. Excluding one withdrawal, we received seven stage 1 submissions, out of which two classified as nails, three as hammers, and two covered both aspects.

[1] `http://www.nobelprize.org/nobel_prizes/economics/laureates/2012/`

In stage 2, with a later deadline, we solicited regular submissions on any of the topics outlined initially. We encouraged submissions that specifically addressed topics identified in stage 1; for a tool description paper, this could, e.g., have been done by motivating the tool with a stage 1 problem, and sketching how the tool could, or will, be applied in this domain. We also encouraged the stage 1 authors to revise and expand their initial submissions. The referees were advised to judge submissions based on the PC's views of the likelihood of contributing to a better matching of hammers (formalisation and verification tools) to nails (domain problems).

We invited research and position papers, as well as tool and system descriptions, and finally formalised knowledge representations with human-readable annotations.

In addition to the revised and expanded versions of the stage 1 submissions, we accepted four new submissions in stage 2. These included two tool and system descriptions, one description of a formalised knowledge representation and one research paper.

The submission and review process was supported by the EasyChair system.

5 Matchmaking

To provide orientation for submission stage 2, we published for each stage 1 submission a summary of their hammer/nail aspects and gave advice on how they could be matched.[2] Several authors followed this advice and made explicit references to stage 1 submissions in their papers, using the submission numbers given below.

Submission 1 (nail)

SAsSy – Scrutable Autonomous Systems (Nava Tintarev, Nir Oren, Roman Kutlak, Matt Green, Judith Masthoff, Kees van Deemter and Wamberto Vasconcelos)

Failure to understand complex (e.g. distributed) autonomous systems may lead to unrealistic expectation or lack of trust (e.g. in dangerous environments), and thus limits their adoption. The authors discuss the importance of making them scrutable, i.e. adding to them a mechanism that can explain the alternatives for actions and its decisions, so that it can communicate with humans in an understandable way on a level that allows users to cope with potentially large amounts of data. Making autonomous systems scrutable rests on four foundations:

1. a representation for planning
2. human-understandable reasoning mechanisms
3. translating logical statements into natural language
4. techniques that enable the user to cope with a deluge of information

Each of them requires formalisation and has its own unique challenges, as does the integration of all of these into a single system.

[2] Most stage 1 submissions have subsequently been revised for stage 2.

The SAsSy project aims at developing a hammer for this nail but is in an early stage and could therefore (re)use existing hammers. Respondents to this work can be found on two different ends: non-trivial applications as well as contributions to the different parts (1–4). The authors have so far identified the following possible hammers, and studied relevant literature:

- 1. knowledge acquisition
- 1. and 2. argumentation theory
- 2. and 3. natural language generation
- 4. diagrams, user modelling, user evaluation

Submission 2 (nail and hammer)

Transparency of Environmental Computer Models (Martine de Vos, Jan Top, Willem Robert van Hage and Guus Schreiber)

The environment is complex to model: long-term processes, complex mechanisms, lots of variability, not everything well understood, lack of empirical data. Current computer models of this, as well as the process of their development, lack transparency: What is incorporated in the model and what not, how trustable are the results? This restricts their applicability and reusability by outsiders/non-developers. The following is needed to make them more transparent:

1. make their underlying conceptual model explicit
2. encourage model developers (i.e. domain experts) to do (1), using formalisation
3. encourage them by incentives

The following previously existing solutions are not yet completely sufficient:

- teaching best practice to modellers (when they don't see an urgent need for transparency, while society has this need)
- high-level annotation of scientific models and workflows, and provenance annotation of scientific data

The authors present the approach of explicitly describing the conceptual models themselves (concepts and relations) as ontologies, and using the latter to

- facilitate reviews of the model
- communicate the model and its underlying scientific knowledge

Respondents to this work can help to address requirements (1–3), or apply ontologies as a means of review and communication in other application settings.

Submission 3 (hammer)

A Vision of Collaborative Verification-Driven Engineering of Hybrid Systems (Stefan Mitsch, Grant Olney Passmore and Andre Platzer)

Hybrid systems with both discrete and continuous dynamics are an important model for real-world physical systems. The key challenge is how to ensure their correct functioning w.r.t. safety requirements. Promising techniques to ensure safety seem to be model-driven engineering to develop hybrid systems in a well-defined and traceable manner and formal verification to prove their correctness, forming the vision of verification-driven engineering. Despite the remarkable progress in automating formal verification of hybrid systems, the construction of proofs of complex systems often requires significant human guidance, since hybrid systems verification tools work over an undecidable theory and cover different fields of mathematics. It is thus not uncommon for verification teams to consist of many players with diverse expertise (e.g. on several fields of mathematics, verification, machine-supported theorem proving, etc.) This paper presents a verification-driven toolset for collaboratively engineering large-scale hybrid systems, which supports

- modeling hybrid systems
- exchanging and comparing models and proofs, and
- managing verification tasks

Particular strenghts include

- decomposition of hybrid systems (so that they can be verified by verifying properties of their subsystems),
- efficiently solving multivariate polynomial inequalities (by integrating existing provers; work in progress),
- collaborative development of models and proofs (in a generic language and in domain-specific ones)

Respondents to this work can offer new application domains.

Submission 4 (hammer)

Interacting with Ontologies and Linked Data through Controlled Natural Languages and Dialogues (Ronald Denaux, Vania Dimitrova and Anthony Cohn)

Ontologies play an important role in many different appication areas. Although the languages in which they can be speficied look superficially simple, domain expert have problems to apply them appropriately. This paper presents three steps to enable domain experts to build ontologies:

1. Rabbit, a controlled natural language frontend for OWL, which has been successfully tested with domain experts in cartography

2. ROO, an editor that helps domain experts to avoid modelling mistakes by guiding them through the ontology authoring process

3. a facility that enriches this editor with interactive feedback on logical consequences of new facts to be added to an ontology, i.e.:

   (a) the fact already is in the ontology

   (b) can be derived from it,

   (c) contradicts it,

   (d) is new (and if so whether anything follows from it)

   So far this makes authors aware of problems, but does not really yet help to resolve them.

4. a dialogue-based user interface for giving more differentiated feedback while the author is building an ontology (but useful beyond ontology authoring)

Respondents to this work could evaluate whether the tools presented here are really independent of the application domain, or try to scale the techniques to more expressive logics.

Submission 5 (hammer)

Explaining the Outcome of Knowledge-Based Systems; a discussion-based approach (Martin Caminada, Mikołaj Podlaszewski and Matt Green)

Nonmonotonic reasoning is a framework for analysing the construction of arguments based on rules of thumb which are subject to exceptions. This submission proposes such a discussion-based approach for explaining the outcome of knowledge-based systems that use nonmonotonic (defeasible) inference. An interactive Python-based frontend is available, which aids in the process of constructing arguments. A key feature is the ability not only to establish an argument, but also to give a (one hopes) explanation of the justification of the argument suitable for human users to digest.

Respondents could apply this technique in their respective domains; it seems particularly promising in the social sciences.

Submission 6 (hammer and nail)

Developing an Auction Theory Toolbox (Christoph Lange, Colin Rowat, Wolfgang Windsteiger and Manfred Kerber)

Auctions allocate trillions of dollars in goods and services every year. Auction design can have significant consequences, but its practice outstrips theory. The authors aim at advancing auction theory with help from mechanised reasoning. To that end they are developing a toolbox of formalised representations of key facts of auction theory, which will allow auction designers to have relevant properties of their auctions machine-checked. In the starting phase of this effort, they are investigating the suitability of different mechanised reasoning systems (Isabelle, Theorema, and TPTP) for

reproducing a key result of auction theory: Vickrey's celebrated 1961 theorem on the properties of second price auctions – the foundational result in modern auction theory. Based on their formalisation experience, they give tentative recommendations on what system to use for what purpose in auction theory, and outline further steps towards a complete auction theory toolbox.

Respondents could apply a similar toolbox building methodology in other domains, or could contribute additional mechanised reasoning know-how to the authors' auction theory toolbox building effort.

Submission 7 (nail)

Model Validation and Test Portfolios in Financial Regulation (Neels Vosloo)

Modern finance relies on software, whether to price assets ('valuation models') or to compute portfolios' riskiness ('capital models'). Given the quantities of money involved, errors in financial software can lead to tens or hundreds of millions of dollars of losses. At the same time, model validation and checking is typically performed using traditional, manual methods. This constellation of observations has led to high-level concern about whether the modern financial system is overly reliant on computational models, whose possibilities for failure are not well understood or controlled (q.v. the 2012 Beddington/Foresight report).

In the midst of this clearly very large domain area, Vosloo outlines a tractable starting point for the consideration of formal methods/mechanised reasoning: how can regulators develop minimal 'test portfolios', capable of efficiently ensuring that capital models incorporate relevant risk factors, benchmarking those capital models against each other, and stress testing them under a variety of market conditions.

Respondents could outline how to apply verification techniques to this problem.

6 Organisation

Do-Form had the following programme committee:

1. Bill Andersen, Highfleet, Baltimore, US
2. Rob Arthan, Lemma 1, Reading, UK
3. Christoph Benzmüller, Mathematics and Computer Science, Free University of Berlin, Germany
4. Peter Cramton, Economics, University of Maryland, US
5. James Davenport, Computer Science and Mathematical Sciences, University of Bath, UK
6. Michael Grüninger, Mechanical and Industrial Engineering, University of Toronto, Canada
7. Manfred Kerber, Computer Science, University of Birmingham, UK (**co-chair**)
8. Michael Kohlhase, Computer Science, Jacobs University Bremen, Germany
9. Christoph Lange, Computer Science, University of Birmingham, UK (**co-chair**)
10. Till Mossakowski, DFKI and University of Bremen, Germany
11. Colin Rowat, Economics, University of Birmingham, UK (**co-chair**)
12. Todd Schneider, Raytheon, Sterling, VA, US

13. Richard Steinberg, London School of Economics, UK
14. Geoff Sutcliffe, Computer Science, University of Miami, US
15. Theodore L. Turocy, Centre for Behavioural and Experimental Social Science, University of East Anglia, UK
16. Makarius Wenzel, Formal Testing and System Exploration, University of Paris Sud, France
17. Wolfgang Windsteiger, RISC / JKU Linz, Austria

The three chairs are running ForMaRE, an effort to apply Formal Mathematical Reasoning in Economics[3]. ForMaRE is supported by EPSRC grant EP/J007498/1 ("Formal Representation and Proof for Cooperative Games: A Foundation for Complex Social Behaviour").

Do-Form web page: `http://www.cs.bham.ac.uk/research/projects/formare/events/aisb2013/`

[3] `http://www.cs.bham.ac.uk/research/projects/formare/`

Author Index

B
Botta, Nicola 32
Bundy, Alan 57
C
Caminada, Martin 21
Caminati, Marco B. 51
Cohn, Anthony 18
D
Davis, Jared 43
de Vos, Martine 4
Denaux, Ronald 18
Deng, Liwei 57
Dimitrova, Vania 18
G
Green, Matt 1, 21
H
Hofmann, Mareen 32
I
Ionescu, Cezar 32
K
Kerber, Manfred 25
Kutlak, Roman 1
L
Lange, Christoph 25
M
Mandel, Antoine 32
Masthoff, Judith 1
McNeill, Fiona 57
Mitsch, Stefan 8
O
Oren, Nir 1
P
Passmore, Grant Olney 8
Platzer, Andre 8
Podlaszewski, Mikołaj 21
R
Rowat, Colin 25
S
Schreiber, Guus 4
Schupp, Sibylle 32
Smaill, Alan 57

T
Tintarev, Nava 1
Top, Jan 4
V
van Deemter, Kees 1
van Hage, Willem Robert 4
Vasconcelos, Wamberto 1
Vosloo, Neels 30
W
Windsteiger, Wolfgang 25

SAsSy — Scrutable Autonomous Systems

Nava Tintarev, Roman Kutlak, Nir Oren, Kees Van Deemter
Matt Green, Judith Masthoff, Wamberto Vasconcelos
University of Aberdeen, email: n.tintarev@abdn.ac.uk

Abstract. An autonomous system consists of physical or virtual systems that can perform tasks without continuous human guidance. Autonomous systems are becoming increasingly ubiquitous, ranging from unmanned vehicles, to robotic surgery devices, to virtual agents which collate and process information on the internet. Existing autonomous systems are opaque, limiting their usefulness in many situations. In order to realise their promise, techniques for making such autonomous systems scrutable are therefore required. We believe that the creation of such scrutable autonomous systems rests on four foundations, namely an appropriate planning representation; the use of a human understandable reasoning mechanism, such as argumentation theory; appropriate natural language generation tools to translate logical statements into natural ones; and information presentation techniques to enable the user to cope with the deluge of information that autonomous systems can provide. Each of these foundations has its own unique challenges, as does the integration of all of these into a single system.

1 Introduction

An *autonomous system* consists of physical or virtual systems that can perform tasks without continuous human guidance. Autonomous systems are becoming increasingly ubiquitous, ranging from unmanned vehicles, to robotic surgery devices, to virtual agents which collate and process information on the internet. Such systems can potentially replace humans in a variety of tasks which can be dangerous (such as refuelling a nuclear reactor), mundane (such as crop picking), or require superhuman precision (as in robotic surgery). Note that some of these tasks could be safety critical, in the sense that human life can be at risk if the autonomous system does not behave as intended. The reasoning processes driving an autonomous system can range from reactive mechanisms (potentially interacting to create complex behaviours c.f. [1]), to rule based systems such as [4, 12] which are widely studied in the agents literature to very complex formalisms capable of dealing with uncertainty, online planning, and the like (see [13] for an overview). While increasing reasoning complexity can enable an autonomous system to handle a wider range of situations, modelling and verifying the operation of such systems becomes increasingly difficult.

A *distributed autonomous system (DAS)* consists of multiple autonomous components (often referred to as *agents*), which can communicate with each other while cooperating or competing to achieve some set of goals. The complexities of autonomous systems, particularly when distributed, means that humans struggle to establish why a system chose to behave as it did, to identify what alternative actions the system considered, and to determine why these alternatives were not selected for execution by the system. In other words, such systems are *opaque*. Such opacity is exacerbated by the formal models typically used to drive the reasoning behaviour in such systems — a human (and particularly a non-expert) often struggles to communicate with an autonomous system. This lack of understanding can lead to unrealistic expectations of an autonomous system, or alternatively to a lack of trust in it, causing inefficiencies at best, and leading to dangerous outcomes in the worst cases. Such problems limit the adoption of these types of systems.

The SAsSy project[1] proposes to investigate computational mechanisms for providing transparency to humans regarding the internal workings of a DAS. More specifically, we seek to utilise formal argumentation techniques to explain (to a human) *why* some plan was chosen for execution by the system, and to allow the human to provide additional information which can be used to modify the plan. We also aim to identify the best ways to present the explanations (primarily as text, but also in diagrammatic form) to a human operator, based on extensive knowledge acquisition, user modelling and user evaluation.

2 Example

Let us provide a simple example from the railway domain that illustrates some of the issues. In this scenario multiple stakeholders have to agree on a plan for maintaining railway lines. We assume two actions are possible: `Move(equipment, from, to)` and `Repair(equipment, location)`. Repair is performed using equipment, or more specifically a crane c_1 or c_2, and the crane has to be moved to a location before repair of that location can take place. Two locations, a and b need to be repaired. Location b has to be repaired before location a (an example of a constraint; cannot be violated). Each of the stakeholders is represented by an agent and each agent has different preferences and requirements. In our scenario, agent α prefers not to use crane c_2 (an example of preference; can be violated). Now assume the following starting 'state of the world' S_0: Locations a and b need to be repaired, crane c_1 is at location a and crane c_2 is at location b. S_G is the goal state – the desired state of the world where both locations a and b are fixed and all constraints are fulfilled.

There are many plans that achieve the goal. Figure 1 shows the two simplest plans. Suppose the system recommends plan A and a human user has to decide whether to follow the plan.

The user may ask why the system chose plan A over plan B. Plan A contains more steps (an extra Move action). We note that the question can be asked and answered at different levels of granularity. For example, a user may request an explanation for the transition

[1] `http://www.abdn.ac.uk/ncs/computing/research/ark/projects/current/sassy/`

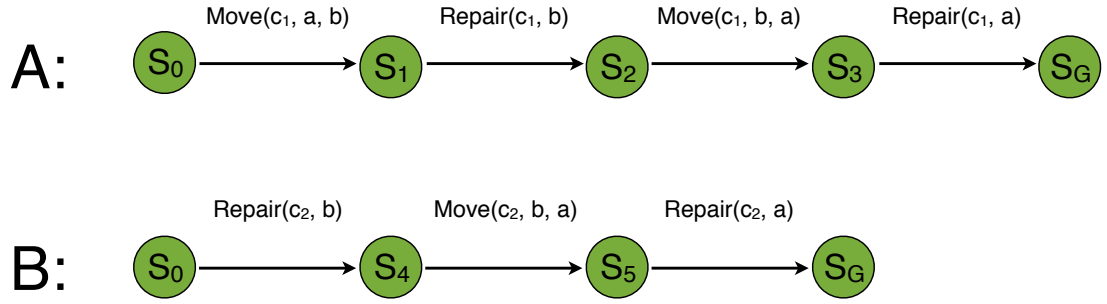


Figure 1. An example of two possible plans. A plan can be viewed as a sequence of transitions changing the state of the world. S_0 is the initial state and S_G is the final (goal) state. The plan can be verbalised as, e.g.:*"Move crane c_1 to location b and fix it. Then move the crane to location a and fix location a. "*

$s_0 \rightarrow s_1$ ("Why did you move c1 first?") or for the plan itself ("Why did you select a plan with more steps?"). It is also possible to argue against competing plans, or for the winning plan.

A possible explanation for the transition may be: *"So that c_1 could repair location b."* (why plan A) or *"Using c_2 at b goes against α's preference."* (why not plan B) . For the plan, the explanation may be: *"Because this means that we respect α's preference not to use crane c_2* (why plan A), or *"B involves the use of crane c_2, which goes against the preference of agent α."* (why not plan B).

We have presented a very simple example, that is limited to a single decision or choice point. A more complex plan might involve a large number of decisions (as well as a larger number of competing plans). However we believe it to be sufficient to illustrate the majority of challenges scrutability of plans and arguments bring forward. In the next section we elaborate on these challenges.

3 Challenges

Agents execute actions in order to achieve goals. Within a distributed system, the choice of actions to execute depends on both the environment and the actions of other agents. Goals usually require more than one action to be executed before they are achieved, and agents therefore generate *plans*, either individually, or through interactions with other agents. We therefore believe that in order to understand a distributed autonomous system, a human must be able to understand the plans within the system — including constraints, dependencies, and why a given plan may have been chosen over other plans. Given such an understanding, the human can *critique* the plans if necessary, providing the agents with additional information which they can use to modify their plans. The example in Section 2 described a scenario where a person was supported in making a choice between two plans where there is both a preference in terms of which resources to use and a constraint in terms of the order in which certain things can happen.

Recent research has investigated using formal argumentation to perform distributed planning [16]. Much of the literature in argumentation theory concerns itself with the *status* of an argument, that is, whether the argument licenses certain conclusions. This work assumes the existence of a static set of arguments, and computes an argument's status based on this set. Intuitively, arguments can be seen as a debate, or reasoning for and against a course of action. The psychology of human reasoning as validation for argumentation semantics is a largely unexplored area [11], but a recent strand of work takes its inspiration from human dialogue to find intelligible explanations of an argument's status [2]. A popular approach to the use of argumentation in planning requires the identification of appropriate *argumentation schemes* — common templates for argument — which are able to describe the planning process [6, 10].

Formal argument theory represents arguments using a logical language. In order to make arguments accessible to non-technical users, a system must be able to translate formulas of this formal logical language into natural language. *Natural Language Generation* (NLG) is the study of computer algorithms which produce understandable and appropriate texts in English or other human languages, from some underlying non-linguistic representation of information. Surprisingly little work in NLG has considered how such logical statements can be expressed in a clear and understandable manner. Substantial work on proof presentation exists, but this tends to focus on the proof structure (e.g., by omitting easily inferable information [5]) rather than the individual propositions. Despite useful early work in the 1980's [18], what is largely missing is an algorithm that converts complicated input formulas into more accessible forms. NLG researchers have typically simplified by departing from a fixed logical form in which crucial presentation decisions have already been made (e.g., [7, 15]). For example, suppose the system produces the following argument in support of an action A:

$$(\neg success \rightarrow \neg q) \;\&\; (\neg q \rightarrow \neg p) \;\&\; (\neg p \rightarrow \neg A).$$

This formula is unnecessarily complex, and existing NLG systems that express this "word-by-word" would tend to generate something like: *"If success is not achieved then q is false. If q is false then p is false. If p is false then Action A is not performed"*. A much simpler presentation would be possible, however, e.g. *"q guarantees success; p implies q; Action A results in p"*. To allow natural language to do this, however, the formula above first needs to be converted into the equivalent

$$(q \rightarrow success) \;\&\; (p \rightarrow q) \;\&\; (A \rightarrow p).$$

There are two challenges here: Firstly, one needs to find out what is an optimally understandable format for expressing a given proposition. This is an empirical question that can only be solved using extensive experimentation with human subjects. Secondly, one needs to find algorithms for actually finding that optimal form. This can be extremely challenging computationally, particularly if the logic is very expressive.[2] This is a famous open problem in NLG, known as

[2] Logical formulas are typically equivalent to infinitely many other formu-

the Logical Form Equivalence problem ([14], [17]). We shall attack the problem by relaxing the requirement that it is always necessary to find the unique *optimal* form, settling for heuristics that give good results in most situations. That is, find heuristics for expressions that are "good enough" rather than necessarily the "best" form. A simple example of a heuristic that might prove to go a long way is: *shorter formulas are easier to understand than longer ones*. Finding the most useful heuristics is an important challenge for the experimental work in this area.

Making the language understandable is a crucial one for system adoption. It could be possible to ask the system to reformulate a statement, or to confirm a given interpretation, as one would a human conversational partner. However, a system that requires too many additional interactions is less likely to be adopted by users.

Other Information Presentation (IP) issues that our application gives rise to include aggregation and summarisation of information. For example, if a large number, say n, of locations need to be repaired, it would be cumbersome to spell this it in n separate steps (*"First move to location 1 and repair it; then move to location 2 and repair it; ...; finally move to location n and repair it."*). It is much better to combine these events in one phrase (saying "each location") and to leave out any obvious information. For instance, the system might simply say *"Repair each location, starting with the nearest one and ending with the one furthest away"*, which aggregates the repairs actions and leaves out the 'move' actions because they are inferable.

Beyond the elucidation of an individual plan, it is important to be able to explain why one plan is superior to another, and this raises Information Presentation (IP) challenges of its own. Consider Figure 1, for example. In an initial interaction it may be suitable to give a high-level explanation of the choice between the plans A and B, as in *"A is proposed because we wish to respect α's preference not to use crane c_2"*. However, the user may also want to ask questions about particular actions, such as why a certain crane was used or moved first, and this may require more detailed explanations. Since different users may have different information requirements, the system should make use of adaptation and user modelling, to decide about the level of abstraction, and the level of detail, that is required in a given situation. Managers, for example, may only require a small amount of high-level information, whereas others may need to scrutinise the plan in more detail, for example to make sure they agree with the assumptions on which it is based. It also seems plausible that there is a need to combine multiple levels of abstraction. For example, a user may want to begin with a more high-level description for which they then request a more detailed description, or scrutinise aspects of.

Traditionally, diagrams have played an important role in argumentation, with diagrams expressing networks of arguments supporting and attacking each other [3]. These diagrams, however, become cluttered when networks are large and would benefit from aggregation and summarisation as well. Given that each modality has its own advantages [8, 9], we shall explore how language can complement these diagrams, for instance by placing text inside a diagram, or by using one modality at a meta-level with respect to the other. For example, a caption may be generated that highlights an important aspect of a diagram, for instance *"Several arguments attacked this claim, but each proved contentious"*.

Given that the dialogue occurs as dynamic process involving inter-dependent components, evaluation of these components is difficult, but critical. For example, it may be necessary to evaluate the scrutability on multiple levels of abstraction such as an entire plan, the argumentation in favour of a plan, and individual facts or arguments. The evaluation therefore forms a final challenge in this program of work.

las, and finding out if two formulas are equivalent is undecidable in many logics, or else computationally very expensive.

Acknowledgements

This research has been carried out within the project "Scrutable Autonomous Systems" (SAsSY), funded by the Engineering and Physical Sciences Research Council (EPSRC, UK), grant ref. EP/J012084/1.

REFERENCES

[1] Rodney A. Brooks, 'Intelligence without representation', *Artificial Intelligence*, **47**, 139–159, (1991).
[2] Martin Caminada and Mikolaj Podlaszewski, 'Grounded semantics as persuasion dialogue', in *COMMA*, eds., Bart Verheij, Stefan Szeider, and Stefan Woltran, volume 245 of *Frontiers in Artificial Intelligence and Applications*, pp. 478–485. IOS Press, (2012).
[3] Phan Minh Dung, 'On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games.', *Artificial Intelligence*, **77**, 321–257, (1995).
[4] Koen V. Hindriks, Frank S. De Boer, Wiebe Van Der Hoek, and John-Jules Ch. Meyer, 'Agent programming in 3apl', *Autonomous Agents and Multi-Agent Systems*, **2**(4), 357–401, (November 1999).
[5] Helmut Horacek, 'Generating inference-rich discourse through revisions of rst-trees', in *AAAI*, pp. 814–820, (1998).
[6] Rolando Medellin-Gasque, Katie Atkinson, Peter McBurney, and Trevor J. M. Bench-Capon, 'Arguments over co-operative plans', in *TAFA*, eds., Sanjay Modgil, Nir Oren, and Francesca Toni, volume 7132 of *Lecture Notes in Computer Science*, pp. 50–66. Springer, (2011).
[7] Yael Dahan Netzer, Michael Elhadad, and Ben Gurion, 'Generating determiners and quantifiers in hebrew', in *ACL workshop on Semitic Languages*, pp. 89–96, (1998).
[8] Jon Oberlander, Richard Cox, Padraic Monaghan, Keith Stenning, and Richard Tobin, 'Individual differences in proof structures following multimodal logic teaching', in *COGSCI*, pp. 201–206, (1996).
[9] M. Petre, 'Why looking isnt always seeing: Readership skills and graphical programming', *CACM*, **39**, 33–42, (1995).
[10] Luc De Raedt, Christian Bessière, Didier Dubois, Patrick Doherty, Paolo Frasconi, Fredrik Heintz, and Peter J. F. Lucas, eds. *ECAI. Including Prestigious Applications of Artificial Intelligence (PAIS) System Demonstrations Track*, volume 242 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2012.
[11] Iyad Rahwan, Mohammed Iqbal Madakkatel, Jean-Francois Bonnefon, Ruqiyabi Naz Awan, and Sherief Abdallah, 'Behavioural experiments for assessing the abstract argumentation semantics for reinstatement', in *Cognitive Science*, pp. 1483–1502, (2010).
[12] Anand S. Rao, 'AgentSpeak(L): BDI agents speak out in a logical computable language', in *MAAMAW '96: Proceedings of the Seventh European workshop on Modelling autonomous agents in a multi-agent world : agents breaking away*, pp. 42–55, Eindhoven, The Netherlands, (1996).
[13] S. Russell and P Norvig, *Artificial Intelligence: A Modern Approach (Second Edition)*, Prentice Hall, 2003.
[14] S. M. Shieber, 'The problem of logical form equivalence', *Computational Linguistics*, **19**, 179–190, (1993).
[15] Xiantang Sun and Chris Mellish, 'An experiment on free generation from single rdf triples', in *ENLG*, pp. 105–108, (2007).
[16] Yuqing Tang, Timothy J. Norman, and Simon Parsons, 'A model for integrating dialogue and the execution of joint plans', in *AAMAS (2)*, eds., Carles Sierra, Cristiano Castelfranchi, Keith S. Decker, and Jaime Simão Sichman, pp. 883–890. IFAAMAS, (2009).
[17] Kees van Deemter, 'Structured meanings in computational linguistics', in *13th International Conf. on Computational Linguistics (COLING-90)*, ed., H.Karlgren, pp. 85–89, Helsinki, Finland, (1990).
[18] Wahlster, Marburger, Jameson, and Busemann, 'Over-answering yes-no questions', in *8th Int. Joint Conference on Artificial Intelligence (IJCAI-83)*, ed., A.Bundy, pp. 643–646, Karlsruhe, Germany, (1983).

Transparency of Environmental Computer Models

Martine G. de Vos and **Jan Top** and **Willem Robert van Hage** and **Guus Schreiber** [1]

Abstract. Environmental computer models are considered essential tools in supporting environmental decision making, but their main value is that they allow a better understanding of our complex environment. Despite numerous attempts to promote good modelling practice, transparency of current environmental computer models is limited, which hinders progress in both science and policy making. An important cause is that the structure, meaning and context of environmental computer models is often not clear for other people than the model developers. In the proposed research project we would like to find out whether it is possible to increase the transparency of environmental computer models by making their underlying conceptual model explicit. In preliminary research we identified the following challenges: 1) many model developers are mainly focused on the computational instead of the descriptive aspects of computer models 2) many environmental modellers may not consider the lack of transparency a big problem nor do they see computer scientists as natural partners in cooperation. However, we think that both environmental and computer science could benefit from an interdisciplinary or even a totally integrated approach. We expect that experimenting with tools and methods from computer science could teach us important lessons on the practice of environmental modelling and hopefully guide us to this novel, integrated way of performing e-science.

1 Introduction

1.1 Environmental Computer Models

Current environmental issues have features that distinguish them from traditional scientific problems. They are universal in their scale and long-term in their impact, their mechanisms are complex, variable and not well understood and empirical data are scarce or inadequate [5, 18, 21]. In addition there is an urgent need to find strategies to cope with these issues and political pressure on the research community is high [21].

Environmental computer models are simplified and controllable representations of natural systems, developed by scientists. These models include knowledge and data on the key mechanisms and factors that explain the behaviour of natural systems in a certain context. Although it is hardly possible to validate the results of environmental computer models [13], they are essential tools in supporting environmental decision making by exploring the consequences of alternative policies or management scenarios [5, 18]. They are used to support important political decisions and national investments like the construction of dikes and the design of the future energy system. But the main value of environmental models is that they allow a better understanding of our complex environment [13].

[1] Computer Science, Network Institute, VU University Amsterdam, the Netherlands, email:{Martine.de.Vos—W.R.van.Hage—J.L.Top—Guus.Schreiber}@vu.nl

1.2 Problem Description

Both the developing process and the computer model itself need to be transparent, in order to enable stakeholders and colleague scientists to understand and use environmental computer models. They need to be able to trace model results and insights through the model structure to the underlying choices and assumptions made by the developer [13, 21]. Despite numerous attempts to promote good modelling practice, transparency of current environmental computer models is limited [18, 1] As a consequence: 1) model results and insights may be used in applications without respecting and discussing their underlying choices and assumptions, and 2) learning from model results and insights is difficult, which hinders progress in both science and policy making.

An important cause is that the structure, meaning and context of environmental computer models is often not clear for other people than the model developers [22]. In the development process modellers inevitably make choices on which processes and concepts to include and which to simplify or neglect [5, 21], but they do not make these assumptions explicit. This, in turn, is caused by the lack of short-term incentives for modellers to provide structure, meaning and context to their models, [2, 18, 7] and the size and complexity of their models [9].

2 Related work

Many authors in the field of environmental modelling advocate standardization of the modelling process and information, summarized to as Good Modelling Practice, to enhance transparency of environmental models [14, 5, 17]. The question is whether providing guidelines is sufficient, as the difficulty is not that the elements of Good Modelling Practice are not known or shared, but that the modelling community lacks the urge to act accordingly [1, 18, 16].

In recent years significant progress has been made in the semantic annotation of scientific models, data work flows and publications. In scientific model development ontologies are used to facilitate conceptualization and to achieve shared understanding among model developers and stakeholders [6]. Ontologies are also widely used to semantically annotate scientific models, datasets and publications, i.e., to connect measurements and terms to the identity of observable entities they quantify [11, 19, 8]. A higher level of abstraction that is being investigated is the semantic annotation of scientific practice as a whole. Annotation of work flows supports scientists to integrate and analyse data in a correct and meaningful way [20]. The open provenance model, PROV, developed by the W3C provenance working group [2] helps scientists to document and process provenance information to ensure reproducibility of their analyses [10].

[2] W3C Provenance Working Group,`http://www.w3.org/2011/prov/`

However, in the described annotation methods the models themselves remain largely black-boxes. As a consequence, we may miss out on valuable information on the developers' understanding and interpretation of the system of interest, which is captured in, for example, the used modelling paradigm, the chosen concepts and their interrelations, and the mathematical equations [22].

3 Approach

This research aims to enable developers and stakeholders to cooperate in the development and use of computer models, and to discuss real-world issues not only on the level of model results but also on the level of functioning of the corresponding natural system.

The main central concept of his study is 'transparency'. We define transparency of a computer model as the connections between model concepts, underlying datasets, related publications and knowledge of the model developer. A transparent computer model, with these connections in place, enables peers and stakeholders to 1) understand the knowledge captured in the computer model and 2) to trace back model results and insights through the model structure to this knowledge.

The second important concept of this study is 'conceptual model'. We define the conceptual model of environmental computer models as a knowledge level model [12] containing the concepts that are included, their definitions and their interrelations. The conceptual model represents the basic premises and knowledge about the working of the system being modelled [5] [14].

The main research question of this research is: *Is it possible to increase the transparency of environmental computer models by making their underlying conceptual model explicit?*

3.1 Preliminary results

We did preliminary research on representing the knowledge underlying environmental computer models. In two case studies on existing environmental computer models we manually reconstructed the underlying conceptual model and formally described it in an ontology.

In the first case study [3] we analysed a computational model that determines the energy use by Indian households. The model specifically addresses the socio economic factors influencing energy uses and includes knowledge on consumer behaviour, public health and sustainable development. We used the model documentation, the model source code and personal communication with the model developer to list and define the concepts and their interrelations and represented them in an OWL ontology [3]. We used this ontology in a peer reviewed model evaluation. Scientists representing different disciplines, viz., economics, sustainable development, energy and public health, were asked to determine if the model consisted of the right elements to achieve its goal, or that elements should be added or deleted. They were provided with a visual representation of the ontology (figure 1), i.e. a UML like diagram, and a glossary of the terms in the ontology, as well as the model documentation and source code. We found that the ontology helped these peers to obtain more information on the model and to gain more insight in its structure. However, they lacked time to get a clear overview of the model and were confused by the different sources of information. We concluded that a better balance between different types of model documentation and explicit links between them are needed to really improve the understanding of the model by the peers. An ontology could be useful in bridging the gap between formal documentation, like source code, and documentation in natural language, like reports and papers.

In the second case study [4] we analysed a spreadsheet model that enables policy analyses concerning the Dutch energy system. We studied the design of the tables and the formulas in the spreadsheets [4] and semantically characterized the underlying concepts and their interrelations (figure 2). We represented these as an instantiation of an existing ontology, the OM Ontology for units of Measure and related concepts [15], and verified our findings with the model developers. We found that the both the spreadsheet design and the formulas contain implicit knowledge about the semantics. The main concepts and their interrelations as we identified them in our ontology did not conflict with the developer's views. But we found that representing the conceptual model in an ontology represented a different perspective, as the developers were primarily focussed on the calculation work flow. The developers may see environmental computer models mainly as instruments to perform simulation studies, and therefore focus on the computational aspects, while we see them as tools to communicate scientific knowledge and therefore focus mainly on the descriptive aspects.

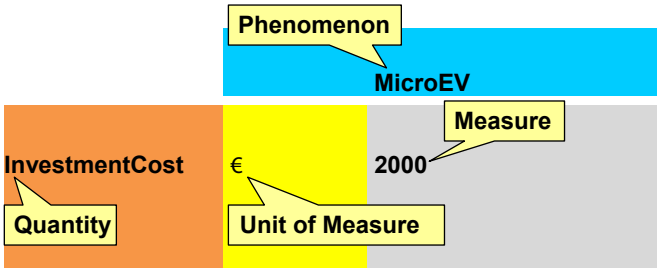


Figure 2. Example, in outline, of the semantic characterization of terms in a spreadsheet table.

3.2 Future work

In future work we intend to perform several case studies on existing environmental computer models. We intend to perform experiments with stakeholders to investigate to what extent reconstructing conceptual models is helpful in understanding and reusing these models. In more detail, we would like to test which form of (visual) presentation of the conceptual model works best to achieve transparency, which aspects of transparency are influenced and to what extent.

The reconstructions in our first case studies were performed manually. During the project we intend to investigate to what extent it is efficient and effective to automatize the process of reconstruction and visual presentation.

We also plan to analyse written (scientific) publications on environmental computer models and the results of their analyses. These publications are often the only way of access to computer models for stakeholders. We would like to find out to what extent it is possible to derive the underlying conceptual model from the written publication and to relate it to the actual content of the computer model.

[3] W3c Web Ontology Language, `http://www.w3.org/TR/owl-features/`

[4] Spreadsheet Examples,`http://semanticweb.cs.vu.nl/edesign/`

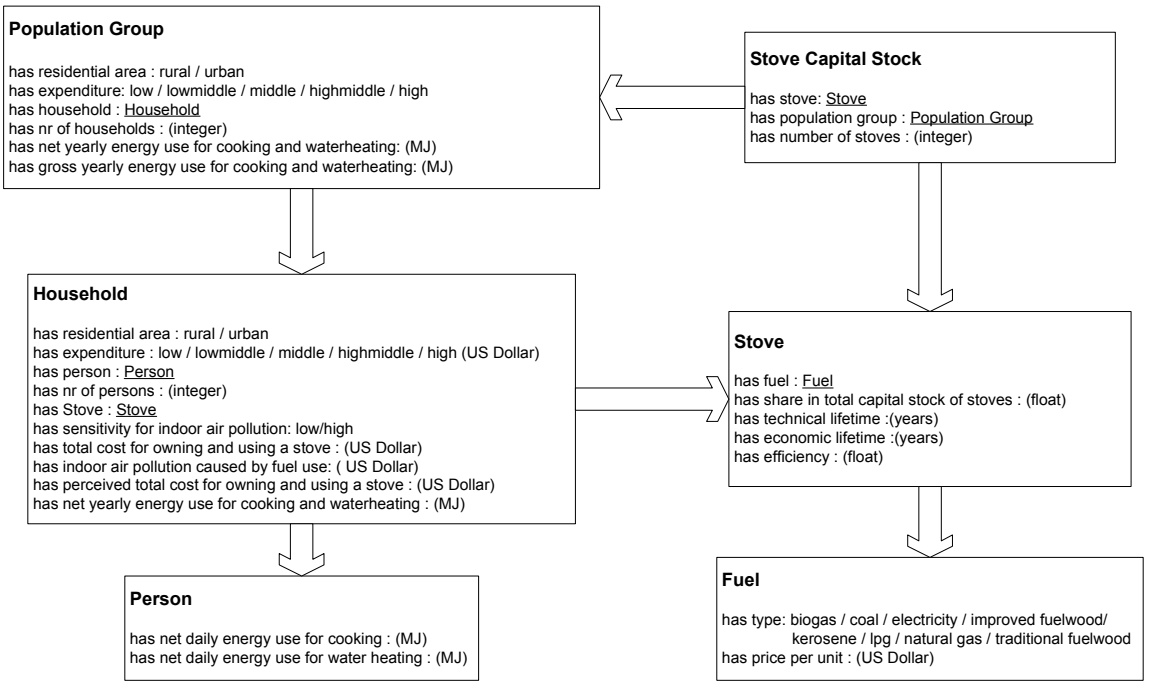


Figure 1. Simplified, visual representation of the ontology of a computational model that determines the energy use by Indian households.

4 Discussion

Our preliminary research gave rise to some additional questions and challenges. Considering our finding that model developers are mainly focused on the computational instead of the descriptive aspects of computer models, we could question whether ontologies are the right method of representing the underlying conceptual knowledge. Analysis and representation of the calculation work flow might also be an option to make the content of a computer model more explicit, but it is not clear to what extent it will contribute to the transparency of computer models. A combined approach is also possible, for example by relating formulas in calculation work flow to concepts in the ontology. An important benefit of ontologies is that they are formal representations and subsequently are amenable to computer processing. (Semi)Automatic analysis of model content and corresponding data and publications could be helpful to achieve transparency of environmental computer models in a more efficient and effective way.

Besides, the request for more transparency is mainly coming from society. Many environmental modellers may not consider the lack of transparency a big problem nor do they see computer scientists as natural partners in cooperation. We think that both environmental and computer science could benefit from an interdisciplinary or even a totally integrated approach. We expect that experimenting with tools and methods from computer science could teach us important lessons on the practice of environmental modelling and hopefully guide us to this novel, integrated way of performing e-science.

5 Matches with other submissions

We see some parallels between our study and submissions 1 and 7. The researchers of submission 1 aim to increase the transparency, which they call scrutability, of autonomous systems. They intend to develop a clear and understandable way to represent formal reasoning models in these systems to humans by translating them into natural language expressions. It would be interesting to compare their and our ways of reconstructing and presenting the conceptual knowledge underlying computer models. Furthermore, their approach could be applicable to the analysis and representation of the calculation work flow in environmental computer models.

The problems concerning computational models in the financial system described by researcher of submission 7 are quite similar to the problems we encounter with environmental computer models. The reliability of these models is questioned and there is a lack of suitable validation/verification techniques. We wonder whether (the lack of) transparency is an issue for these models. Should these models be understandable for non-experts? What type of assumptions and choices are made in these models and to what extent do they influence model results?

REFERENCES

[1] G.a. Alexandrov, D. Ames, G. Bellocchi, M. Bruen, N. Crout, M. Erechtchoukova, A. Hildebrandt, F. Hoffman, C. Jackisch, P. Khaiter, G. Mannina, T. Matsunaga, S.T. Purucker, M. Rivington, and L. Samaniego, 'Technical assessment and evaluation of environmental models and software: Letter to the Editor', *Environmental Modelling & Software*, **26**(3), 328–336, (March 2011).

[2] Nick Barnes, 'Publish your computer code: it is good enough.', *Nature*, **467**(7317), 753, (October 2010).

[3] M.G. De Vos, N Koenderink, B Van Ruijven, and J Top, 'The use of ontologies in peer reviews of Integrated Assessment Models', in *Proceedings of the iEMSs Fifth Biennial Meeting International Congress on Environmental Modelling and Software*, eds., David A. Swayne, Wanhong Yang, Alexey A. Voinov, Andrea Rizzoli, and Tatiana Filatova, pp. 1207–1214, (2010).

[4] M.G. De Vos, Willem Robert Van Hage, Jan Ros, and Guus Schreiber, 'Reconstructing Semantics of Scientific Models : a Case Study', in *Proceedings of the OEDW workshop on Ontology engineering in a data driven world, EKAW 2012*, Galway, Ireland, (2012).

[5] a Jakeman, R Letcher, and J Norton, 'Ten iterative steps in development and evaluation of environmental models', *Environmental Modelling & Software*, **21**(5), 602–614, (May 2006).

[6] S. Janssen, F. Ewert, Hongtao Li, I.N. Athanasiadis, J.J.F. Wien, O. Thérond, M.J.R. Knapen, I. Bezlepkina, J. Alkan-Olsson, a.E. Rizzoli, H. Belhouchette, M. Svensson, and M.K. van Ittersum, 'Defining assessment projects and scenarios for policy support: Use of ontology in Integrated Assessment and Modelling', *Environmental Modelling & Software*, **24**(12), 1491–1500, (December 2009).

[7] Kurt Kleiner, 'Data on demand', *Nature Climate Change*, **1**(April), (2011).

[8] Carla Geovana N. Macário, Sidney Roberto de Sousa, and Claudia Bauzer Medeiros, 'Annotating geospatial data based on its semantics', in *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems - GIS '09*, p. 81, New York, New York, USA, (2009). ACM Press.

[9] Zeeya Merali, 'Why scientific programming doesn't compute', *Nature*, **467**, 6–8, (2010).

[10] Luc Moreau, Ben Clifford, Juliana Freire, Joe Futrelle, Yolanda Gil, Paul Groth, Natalia Kwasnikowska, Simon Miles, Paolo Missier, Jim Myers, Beth Plale, Yogesh Simmhan, Eric Stephan, and Jan Van den Bussche, 'The Open Provenance Model core specification (v1.1)', *Future Generation Computer Systems*, **27**(6), 743–756, (June 2011).

[11] Roberto Navigli, Paola Velardi, Alessandro Cucchiarelli, and Francesca Neri, 'Quantitative and Qualitative Evaluation of the OntoLearn Ontology Learning System', in *Proceedings of the 20th international conference on Computational Linguistics*, (2004).

[12] Allen Newell, 'The knowledge level', *Artificial Intelligence*, **18**(1), 87–127, (January 1982).

[13] Naomi Oreskes, Kristin Shrader-Frechette, and Kenneth Belitz, 'Verification , Validation , and Confirmation of Numerical Models in the Earth Sciences', *Science*, **263**(5147), 641–646, (1994).

[14] J Refsgaard, 'Modelling guidelinesterminology and guiding principles', *Advances in Water Resources*, **27**(1), 71–82, (January 2004).

[15] H. Rijgersberg, M. Wigham, and J.L. Top, 'How semantics can improve engineering processes: A case of units of measure and quantities', *Advanced Engineering Informatics*, **25**(2), 276–287, (April 2011).

[16] Muir Russell, Geoffrey Boulton, Peter Clarke, David Eyton, and James Norton. The Independent Climate Change E-mails Review, 2010.

[17] Edward J. Jr. Rykiel, 'Testing ecological models: the meaning of validation', *Ecological Modelling*, **90**, (1996).

[18] Amelie Schmolke, Pernille Thorbek, Donald L DeAngelis, and Volker Grimm, 'Ecological models supporting environmental decision making: a strategy for the future.', *Trends in ecology & evolution*, **25**(8), 479–86, (August 2010).

[19] Tony C Smith and John G Cleary, 'Automatically linking MEDLINE abstracts to the Gene Ontology', in *Proc. ISMB 2003 BioLINK Text Data Mining SIG*, pp. 1–4, (2003).

[20] Jacek Sroka, Jan Hidders, Paolo Missier, and Carole Goble, 'A formal semantics for the Taverna 2 workflow model', *Journal of Computer and System Sciences*, **76**(6), 490–508, (September 2010).

[21] Jeroen P. van der Sluijs, 'A way out of the credibility crisis of models used in integrated environmental assessment', *Futures*, **34**(2), 133–146, (March 2002).

[22] Ferdinando Villa, Ioannis N. Athanasiadis, and Andrea Emilio Rizzoli, 'Modelling with knowledge: A review of emerging semantic approaches to environmental modelling', *Environmental Modelling & Software*, **24**(5), 577–587, (May 2009).

A Vision of Collaborative Verification-Driven Engineering of Hybrid Systems

Stefan Mitsch [1] and **Grant Olney Passmore** [2] and **André Platzer** [3]

Abstract. Hybrid systems with both discrete and continuous dynamics are an important model for real-world physical systems. The key challenge is how to ensure their correct functioning w.r.t. safety requirements. Promising techniques to ensure safety seem to be model-driven engineering to develop hybrid systems in a well-defined and traceable manner, and formal verification to prove their correctness. Their combination forms the vision of verification-driven engineering. Despite the remarkable progress in automating formal verification of hybrid systems, the construction of proofs of complex systems often requires significant human guidance, since hybrid systems verification tools solve undecidable problems. It is thus not uncommon for verification teams to consist of many players with diverse expertise. This paper introduces a verification-driven engineering toolset that extends our previous work on hybrid and arithmetic verification with tools for (i) modeling hybrid systems, (ii) exchanging and comparing models and proofs, and (iii) managing verification tasks. This toolset makes it easier to tackle large-scale verification tasks.

1 Introduction

Motivation Computers that control physical processes, thus forming so-called *cyber-physical systems* (CPS), are today pervasively embedded into our lives. For example, cars equipped with adaptive cruise control form a typical CPS, responsible for controlling acceleration on the basis of distance sensors. Further prominent examples can be found in many safety-critical areas, such as in factory automation, medical equipment, automotive, aviation, and railway industries. From an engineering viewpoint, CPSs can be described in a *hybrid* manner in terms of discrete control decisions (the cyber-part, e. g., setting the acceleration of a car) and in terms of differential equations modeling the entailed physical continuous dynamics (the physical part, e. g., motion) [28]. More advanced models include aspects of distributed hybrid systems [32] or stochasticity [31], but are not addressed in this paper.

Challenge The key challenge in engineering hybrid systems is the question of how to ensure their correct functioning in order to avoid incorrect control decisions w.r.t. safety requirements (e. g., a car with adaptive cruise control will never collide with a car driving ahead). Especially promising techniques to ensure safety seem to be model-driven engineering (MDE) to incrementally develop systems in a well-defined and traceable manner and formal verification to mathematically prove their correctness, together forming the vision of *verification-driven engineering* (VDE) [20]. Despite the remarkable progress in automating formal verification of hybrid systems, still many interesting and complex verification problems remain that are hard to solve in practice with a single tool by a single person.

Because hybrid systems are undecidable, hybrid systems verification tools work over an undecidable theory, and so verifying complicated systems within them often requires significant human guidance. This need for human guidance is true even for *decidable* theories utilized within hybrid systems verification [5], such as the first-order theory of nonlinear real arithmetic (also called the theory of *real closed fields* or **RCF**), a crucial component of real-world verification efforts. Though decidable, **RCF** is fundamentally infeasible (it is worst-case doubly exponential in the number of variables [6]), which poses a problem for the automated verification of hybrid systems. Much expertise is often needed to discharge arithmetical verification conditions in a reasonable amount of time and space, expertise requiring the use of deep results in real algebraic geometry. It is thus not uncommon for serious hybrid systems verification teams to consist of many players, some with expertise in control theory and dynamical systems, some in software engineering, some in mathematical logic, some in real algebraic geometry, and so on. Hence, well-established project management techniques to coordinate team members are crucial to achieve effective collaborative large-scale verification of hybrid systems. Successful examples of team-based large-scale verification of non-hybrid systems include the operating system kernel seL4 [19] in Isabelle/HOL and the Flyspeck project [15], and show, that indeed collaboration is key for proving large systems.

Vision This paper introduces a VDE toolset (including a backend deployment for project management and collaboration support) and sketches our vision on further enhancing this toolset. It applies proof decomposition in-the-large across multiple verification tools, basing on the completeness of differential dynamic logic ($\mathsf{d}\mathcal{L}$ [28, 33]), which is a real-valued first-order dynamic logic for hybrid programs, a program notation for hybrid systems. The VDE toolset $\mathsf{S}\varphi\mathsf{n}\mathsf{x}$ extends our previous work on the deductive verification tool KeYmaera [36] and on the nonlinear real arithmetic verification tools RAHD [26] and MetiTarski [27] with modeling tools for (i) modeling of hybrid systems in $\mathsf{d}\mathcal{L}$, (ii) exchanging and com-

[1]Carnegie Mellon University, Computer Science Department, 5000 Forbes Avenue, Pittsburgh, PA 15213, email: smitsch@cs.cmu.edu
[2]LFCS, Edinburgh and Clare Hall, Cambridge, 10 Crichton Street, Edinburgh, UK, email: grant.passmore@cl.cam.ac.uk [3]Carnegie Mellon University, Computer Science Department, 5000 Forbes Avenue, Pittsburgh, PA 15213, email: aplatzer@cs.cmu.edu

paring models and proofs via a central source repository, and (iii) exchanging knowledge and tasks through a project management backend.

Structure of the paper In the next section, we give an overview on related work. In Sect. 3 we introduce our architecture of a verification-driven engineering toolset, and describe implementation and features of its components, including a vision of further work. Section 4 introduces an autonomous robotic ground vehicle as application example. Finally, in Sect. 5 we conclude the paper with an outlook on real-world application of the toolset.

2 Related Work

Model-driven engineering in a collaborative manner has been successfully applied in the embedded systems community. Efforts, for instance, include transforming between different UML models and SysML [16], modeling in SysML and transforming these models to the simulation tool Orchestra [2], integration of modeling and simulation in Cosmic/Cadena [13], or modeling of reactive systems and integration of various verification tools in Syspect [10].

Recent surveys on verification methods for hybrid systems [1], modeling and analysis of hybrid systems [8], and modeling of cyber-physical systems [9], reveal that indeed many tools are available for modeling and analyzing hybrid systems, but in a rather isolated manner. Supporting collaboration on formal verification by distributing tasks among members of a verification team in a model-driven engineering approach has not yet been the focus. Although current verification tools for hybrid systems (e. g., PHAVer [11], SpaceEx [12]), as well as those for arithmetic (e. g., Z3 [7]) are accompanied by modeling editors of varying sophistication, they are not yet particularly well-prepared for collaboration either. Developments in collaborative verification of source code by multiple complementary static code checkers [4], modular model-checking (e. g., [22]), and extreme verification [17], however, indicate that this is indeed an interesting field. Most notably, usage of online collaboration tools in the Polymath project has led to an elementary proof of a special case of the density Hales-Jewett theorem [14].

3 The VDE Toolset Sφnx

In order to integrate different modeling and verification tools, the verification-driven engineering toolset Sφnx[1] proposed in this paper follows a model-driven architecture: metamodels for different modeling and proof languages form the basis for manipulating, persisting, and transforming models. The notion of a model here denotes an instance of a metamodel, i. e., it comprises models, proofs, and strategies. Following the definition of the OMG[2], a metamodel defines a language to formulate models: one example for a metamodel is the grammar of d$\mathcal{L}$, which, among others, defines language elements for non-deterministic choice, sequential composition, assignment, repetition, and differential equations. An example for a model is given in Sect. 4: it is a set of formulas, differential equations, and other d$\mathcal{L}$ language elements. It conforms to the grammar of d$\mathcal{L}$, and thus is an instance of the d$\mathcal{L}$ metamodel. Figure 1 gives an overview of the toolset architecture. As can be seen, the d$\mathcal{L}$, KeY, arithmetic, and arithmetic proof metamodels represent interfaces between tools and to the backend.

d$\mathcal{L}$ metamodel The hybrid modeling components (textual and graphical editors for d$\mathcal{L}$, as well as model comparison) manipulate models that conform to the d$\mathcal{L}$ metamodel. A transformation runtime transforms between models in d$\mathcal{L}$ and their textual form read by KeYmaera.

KeY proof metamodel The proof comparison component reads proofs that conform to the KeY proof metamodel. These proofs may either be closed ones (completed proofs, nothing else to be done) or partial proofs (to be continued). A transformation runtime transforms between proofs in KeY and their textual form as generated by KeYmaera.

Arithmetic metamodel Arithmetic editors (not yet implemented) manipulate arithmetic models. Again a transformation runtime transforms between models expressed in terms of the arithmetic metamodel and the corresponding textual input (e. g., SMT-LIB syntax [3]) as needed by arithmetic tools, such as RAHD, MetiTarski, or Z3.

Arithmetic proof metamodel Finally, the proof comparison component reads arithmetic proofs expressed in terms of the arithmetic proof metamodel, and transformed to and from the arithmetic tool's (textual) format by a transformation runtime.

Let us exemplify the toolset with a virtual walk-through a collaborative verification scenario. We begin with modeling a hybrid system using textual and graphical d$\mathcal{L}$ editors. As both operate on the same model, changes in either editor are reflected instantly in the other. The resulting model, which conforms to the d$\mathcal{L}$ metamodel, is transformed on-the-fly during editing by the transformation runtime to a textual input file, and loaded into KeYmaera. In KeYmaera, we apply various strategies for proving safety of our hybrid system model, but may get stuck at some difficult arithmetic problem. We mark the corresponding node in the partial proof and save it in KeYmaera's textual output format. The proof collaboration tool transforms the partial proof text file into a model of the partial proof. We persist the hybrid model and the model of the partial proof in the model and proof repository, respectively. Then we create a request for arithmetic verification (ticket) in the project management repository using the task planning component. The assignee of the ticket accesses the linked partial proof, and extracts an arithmetic verification model from the marked proof node. Then a transformation runtime creates the textual input for one of the arithmetic verification tools. In this tool, a proof for the ticket can be created, along with a proof strategy that documents the proof. Such a proof strategy is vital for replaying the proof later, and for detecting whether or not the arithmetic proof still applies if the initial model changed. Both proof and proof strategy, are imported into the proof collaboration tool and persisted to the corresponding repository. The ticket is closed, together with the node on the original proof (if the arithmetic proof is complete; otherwise, the progress made is reported back). We fetch the new proof model version from the repository and inspect it using the proof comparison component. Then we transform the proof model into its textual form, load KeYmaera and continue proving our hybrid system from where we left off, but now with one goal closed. In case the corresponding arithmetic prover is connected to

[1] `http://www.cis.jku.at/sphinx/` [2] `http://www.omg.org`

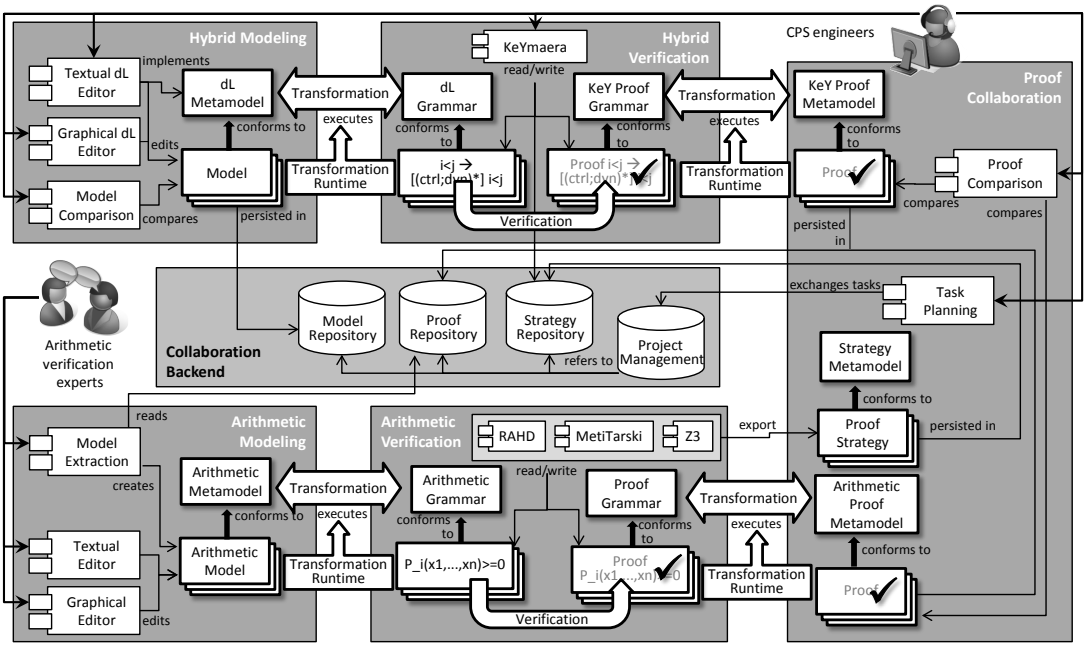


Figure 1: Overview of components in the verification-driven engineering toolset

KeYmaera, we could even load the proof strategy from the strategy repository and repeat it locally.

3.1 KeYmaera: Hybrid System Verification

KeYmaera[3] [36] is a verification tool for hybrid systems that combines deductive, real algebraic, and computer algebraic prover technologies. It is an automated and interactive theorem prover for a natural specification and verification logic for hybrid systems. KeYmaera supports differential dynamic logic (d$\mathcal{L}$) [28, 29, 30, 33], which is a real-valued first-order dynamic logic for hybrid programs, a program notation for hybrid systems. KeYmaera supports hybrid systems with nonlinear discrete jumps, nonlinear differential equations, differential-algebraic equations, differential inequalities, and systems with nondeterministic discrete or continuous input.

For automation, KeYmaera implements a number of automatic proof strategies that decompose hybrid systems symbolically in differential dynamic logic and prove the full system by proving properties of its parts [30]. This compositional verification principle helps scaling up verification, because KeYmaera verifies a big system by verifying properties of subsystems. Strong theoretical properties, including relative completeness results, have been shown about differential dynamic logic [28, 33] indicating how this composition principle can be successful.

KeYmaera implements fixedpoint procedures [34] that try to compute invariants of hybrid systems and differential invariants of their continuous dynamics, but may fail in practice. By completeness [33], this is the only part where KeYmaera's automation can fail in theory. In practice, however, also the decidable parts of dealing with arithmetic may become infeasible at some point, so that interaction with other tools or collaborative verification via Sφnx is crucial.

[3] `http://symbolaris.com/info/KeYmaera.html`

At the same time, it is an interesting challenge to scale to solve larger systems, which is possible according to completeness but highly nontrivial. For systems that are still out of reach for current automation techniques, the fact that completeness proofs are compositional can be exploited by interactively splitting parts of the hybrid systems proof off and investigating them separately within Sφnx. If, for instance, a proof node in arithmetic turns out to be infeasible within KeYmaera, this node could be verified using a different tool connected to Sφnx.

KeYmaera has been used successfully for verifying case studies from train control [37], car control [24], air traffic management [35], and robotic surgery [21]. These verification results illustrate how some systems can be verified automatically while others need more substantial user guidance. The KeYmaera approach is described in detail in a book [30].

In order to guide domain experts in modeling discrete and continuous dynamics of hybrid systems, the case studies, further examples, and their proofs are included in the KeYmaera distribution. When applying proof strategies manually by selection from the context menu in the interactive theorem prover, KeYmaera shows only the applicable ones sorted by expected utility. Preliminary collaboration features include marking and renaming of proof nodes, as well as extraction of proof branches as new subproblems. These collaboration features are used for interaction with the arithmetic verification tools and the collaboration backend described below.

3.2 Arithmetic Verification

Proofs about hybrid systems often require significant reasoning about multivariate polynomial inequalities, i.e., reasoning within the *theory of real closed fields* (**RCF**). Though **RCF** is decidable, it is fundamentally infeasible (hyper-exponential in the number of variables). It is not uncommon for hybrid system models to have tens or even hundreds of real variables,

and **RCF** reasoning is commonly the bottleneck for nontrivial verifications. Automatic **RCF** methods simply do not scale, and manual human expertise is often needed to discharge a proof's arithmetical subproblems.

RCF infeasibility is not just a problem for hybrid systems verification. Real polynomial constraints are pervasive throughout the sciences, and this has motivated a tremendous amount of work on the development of feasible proof techniques for various special classes of polynomial systems. In the context of hybrid systems verification, we wish to take advantage of these new techniques as soon as possible.

Given this fundamental infeasibility, how might one go about deciding large **RCF** conjectures? One approach is to develop a battery of efficient proof techniques for different practically useful fragments of the theory. For example, if an $\exists$ **RCF** formula can be equisatisfiably transformed into an $\wedge\vee$-combination of strict inequalities, then one can eliminate the need to consider any irrational real algebraic solutions when deciding the formula. Tools such as RAHD, Z3 and MetiTarski exemplify this heterogeneous approach to **RCF**, and moreover allow users to define *proof strategies* consisting of heuristic combinations of various specialised proof methods. When faced with a difficult new problem, one works to develop a proof strategy which can solve not only the problem at hand but also other problems sharing similar structure. Such strategies, though usually constructed by domain experts, can then be shared and utilised as automated techniques by the community at large.

3.3 Modeling and Proof Collaboration

In order to interconnect the variety of specialized verification procedures introduced above, Sφnx follows a model-driven engineering approach: it introduces metamodels for the included modeling and proof languages. These metamodels provide a clean basis for model creation, model comparison, and model transformation between the formats of different tools. This approach is feasible, since in principle many of those procedures operate over the theory **RCF**, or at least share a large portion of symbols and their semantics. One could even imagine that very same approach for exchanging proofs between different proof procedures, since proofs in RCF, in theory, can all be expressed in the same formal system. Currently, proofs in Sφnx are exchanged merely for the sake of being repeated in the original tool (although KeYmaera already utilizes many such tools and hence is able to repeat a wide variety of proofs).

In the case of textual languages, Sφnx uses the Eclipse Xtext[4] framework to obtain such metamodels directly from the language grammars (cf. Figure 2, obtained from the d$\mathcal{L}$ grammar [28]), together with other software artifacts, such as a parser, a model serializer, and a textual editor with syntax highlighting, code completion, and cross referencing.

These metamodels are the basis for creating models in d$\mathcal{L}$, as well as for defining transformations between d$\mathcal{L}$ and other modeling languages. The models in d$\mathcal{L}$ make use of mathematical terms, and are embedded in KeY files since KeYmaera uses the KeY [25] format for loading models and saving proofs. In the following sections, we introduce d$\mathcal{L}$ in more detail and describe the support for creating d$\mathcal{L}$ models and working on proofs in Sφnx.

[4] `www.eclipse.org/Xtext`

3.3.1 Differential Dynamic Logic

For specifying and verifying correctness statements about hybrid systems, we use *differential dynamic logic* d$\mathcal{L}$ [28, 30, 33], which supports *hybrid programs* as a program notation for hybrid systems. The syntax of hybrid programs is summarized together with an informal semantics in Table 1; it reflects the metamodel introduced in Figure 2. The sequential composition $\ll\alpha;\ \beta\gg$ expresses that β starts after α finishes (e. g., first let a car choose its acceleration, then drive with that acceleration). The non-deterministic choice $\ll\alpha \cup \beta\gg$ follows either α or β (e. g., let a car decide non-deterministically between accelerating and braking). The non-deterministic repetition operator $\ll\alpha^*\gg$ repeats α zero or more times (e. g., let a car choose a new acceleration arbitrarily often). Discrete assignment $\ll x := \theta\gg$ instantaneously assigns the value of the term θ to the variable x (e. g., let a car choose a particular acceleration), while $\ll x := *\gg$ assigns an arbitrary value to x (e. g., let a car choose any acceleration). $\ll x' = \theta\ \&\ F\gg$ describes a continuous evolution of x within the evolution domain F (e. g., let the velocity of a car change according to its acceleration, but always be greater than zero). The test $\ll?F\gg$ checks that a particular condition expressed by F holds, and aborts if it does not (e. g., test whether or not the distance to a car ahead is large enough). A typical pattern that involves assignment and tests, and which will be used subsequently, is to limit the assignment of arbitrary values to known bounds (e. g., limit an arbitrarily chosen acceleration to the physical limits of a car, as in $x := *; ?x \geq 0$). The deterministic choice $\ll$if(F) then α else $\beta\gg$ executes α if F holds, and β otherwise (e. g., let a car accelerate only when it is safe; brake otherwise). Finally, $\ll$while(F) do α elihw$\gg$ is a deterministic repetition that executes α as long as F holds.

To specify the desired correctness properties of the hybrid programs, differential dynamic logic (d$\mathcal{L}$) provides modal operators $[\alpha]$ and $\langle\alpha\rangle$, one for each hybrid program α. When ϕ is a d$\mathcal{L}$ formula (e.g., a simple arithmetic constraint) describing a safe state and α is a hybrid program, then the d$\mathcal{L}$ formula $[\alpha]\phi$ states that all states reachable by α satisfy ϕ. Dually, d$\mathcal{L}$ formula $\langle\alpha\rangle\phi$ expresses that there is a state reachable by the hybrid program α that satisfies d$\mathcal{L}$ formula ϕ. The set of d$\mathcal{L}$ formulas is generated by the following EBNF grammar (where $\sim \in \{<, \leq, =, \geq, >\}$ and θ_1, θ_2 are arithmetic expressions in $+, -, \cdot, /$ over the reals):

$$\phi ::= \theta_1 \sim \theta_2 \mid \neg\phi \mid \phi \wedge \psi \mid \forall x\phi \mid \exists x\phi \mid [\alpha]\phi \mid \langle\alpha\rangle\phi$$

Thus, besides comparisons $(<, \leq, =, \geq, >)$, d$\mathcal{L}$ allows one to express negations $(\neg\phi)$, conjunctions $(\phi \wedge \psi)$, universal $(\forall x\phi)$ and existential quantification $(\exists x\phi)$, as well as the already mentioned state reachability expressions $([\alpha]\phi, \langle\alpha\rangle\phi)$.

3.3.2 Creating Models

For creating models of hybrid and cyber-physical systems, Sφnx currently includes d$\mathcal{L}$ as generic modeling language. The concrete textual syntax and d$\mathcal{L}$ editor created from the d$\mathcal{L}$ metamodel is shown in Figure 3, together with a concrete graphical syntax and the KeYmaera prover attached through the console. In order to facilitate creation of textual models in d$\mathcal{L}$, Sφnx includes templates of common model artifacts (e. g., ODEs of linear and circular motion). These templates, when

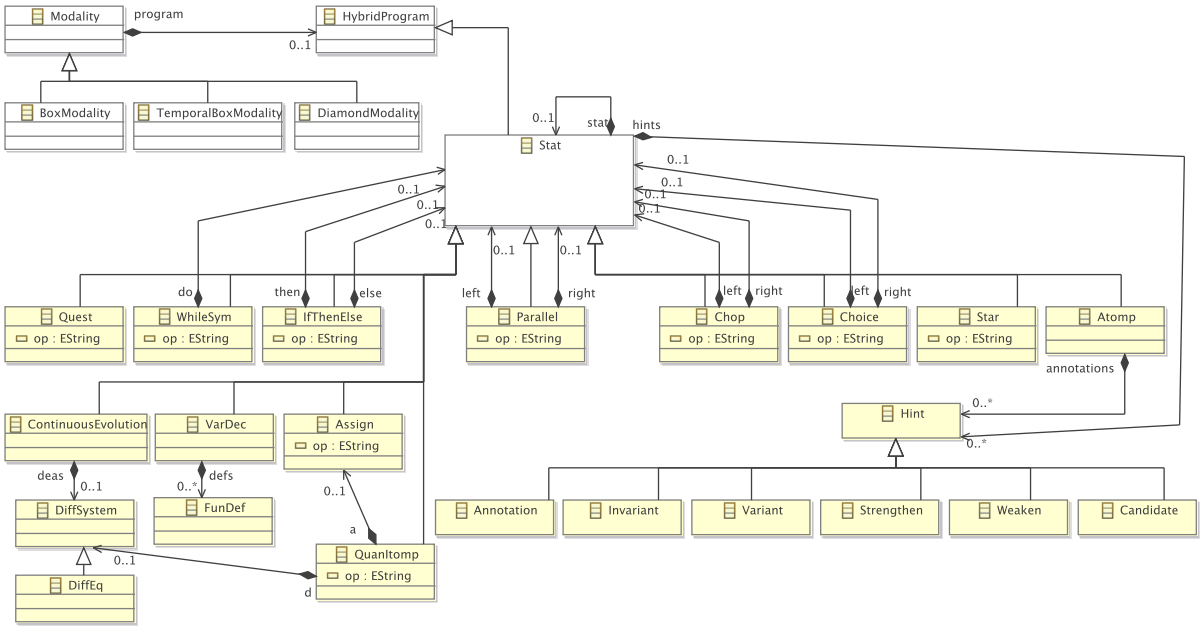


Figure 2: The dℒ metamodel extracted from the input grammar of KeYmaera

Table 1: Statements of hybrid programs

Statement	Metamodel element	Effect
$\alpha;\ \beta$	Chop	sequential composition, first performs α and then β afterwards
$\alpha\ \cup\ \beta$	Choice	nondeterministic choice, following either α or β
α^*	Star	nondeterministic repetition, repeating α $n \geq 0$ times
$x := \theta$	Assign (term)	discrete assignment of the value of term θ to variable x (jump)
$x := *$	Assign (wild card term)	nondeterministic assignment of an arbitrary real number to x
$\big(x_1' = \theta_1, \ldots,$	ContinuousEvolution	continuous evolution of x_i along differential equation system
$\quad x_n' = \theta_n\ \&\ F\big)$	DiffSystem	$x_i' = \theta_i$, restricted to maximum domain or invariant region F
$?F$	Quest	check if formula F holds at current state, abort otherwise
if(F) then α else β	IfThenElse	perform α if F holds, perform β otherwise
while(F) do α end	WhileSym	perform α as long as F holds
$[\alpha]\phi$	BoxModality	dℒ formula ϕ must hold after all executions of hybrid program α
$\langle\alpha\rangle\phi$	DiamondModality	dℒ formula ϕ must hold after at least one execution of hybrid program α

instantiated, allow in-place editing and automated renaming of the template constituents. As usual in the Eclipse platform, such templates can be easily extended and shared between team members.

Since generic modeling languages, such as dℒ for hybrid systems, tend to incur a steep learning curve, the Sφnx platform can be extended with dedicated domain-specific languages (DSL). Such DSLs should be designed to meet the vocabulary of a particular group of domain experts. They can be included into Sφnx in a similar fashion to the generic modeling language dℒ, i. e., in the form of Eclipse plugins that provide the DSL metamodel and the modeling editor. In order to be processable in a verification tool, such as KeYmaera, a model transformation specification (e. g., using the Atlas transformation language ATL [18]) from the DSL to the tool's modeling language (e. g., dℒ) must be provided by these plugins.

For modeling hybrid systems, an interesting opportunity for inspecting the behavior of such a system prior to verification is provided by Mathematica[5] 9, which is able to simulate and plot hybrid system behavior. Specifically, hybrid systems behavior can be plotted using a combination of *NDSolve* and *WhenEvent*. We envision transforming corresponding excerpts of dℒ to Mathematica for visualizing plots of the dynamic behavior and their hybrid programs over time in Sφnx.

[5] `www.wolfram.com/mathematica`

3.3.3 During the Proof

Collaboration support in Sφnx currently comprises model as well as proof comparison, both locally and with the model and proof repositories maintained in a central source code repository. For this, not only textual comparison is implemented, but also structural comparison of models expressed in terms of the dℒ metamodel, as well as of proofs expressed in terms of the KeY metamodel is supported (cf. Figure 4). Especially for collaboration, exchanging proofs and inspecting updates on partial proofs is vital. For example, highlighted changes between different versions of a partial proof lets one easily spot and adopt proof progress made by other team members, go back and forth between versions, and detect conflicts.

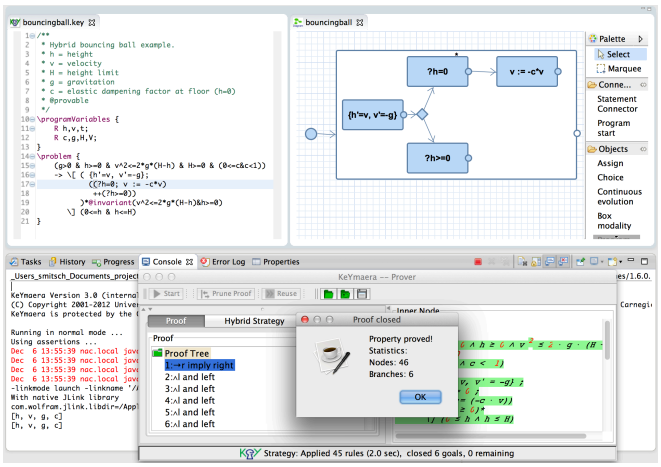


Figure 3: Textual and graphical syntax, proof in KeYmaera

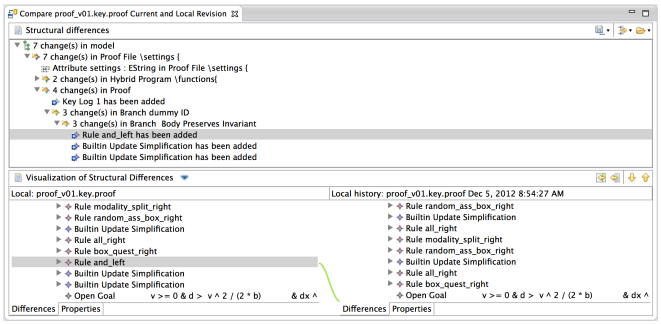


Figure 4: Comparison of the structure of two proof versions

To further facilitate knowledge and expertise exchange, specific unsolved subproblems of a proof (e. g., complex arithmetic problems) can be flagged in KeYmaera and extracted to other tools. Thereby, division of verification work is achieved. An open question, however, concerns the merging of the partial verification results into a coherent proof. In a first step, we plan to extend KeYmaera to let experts close proof branches in the same manner as it currently trusts mathematical solvers. To avoid unintentionally closing a proof goal with a proof that may no longer apply in case the original model changed, Sφnx checks that the statement remained the same (textually) when replaying an arithmetic proof. Later, actual proof certificates and proof strategies will be exchanged to further increase trust, and more sophisticated comparisons of proof goals are envisioned to better support replaying proofs.

3.4 The Collaboration Backend

The Sφnx modeling tool uses existing Eclipse plugins to connect to a variety of backend source code repositories and online project management tools. As source code repository we utilize Subversion[6] and the Eclipse plugin Subclipse[7]. Currently, Mylyn[8] and its connectors are used for accessing online project management tools (e. g., Bugzilla[9], Redmine[10], or any web-based tool via Mylyn's Generic Web templates connector) and exchanging tickets (i. e., requests for verification). These tickets are the organizational means for collaborating on verification problems and tasks within a working group. Exchange of models and proofs may then be conducted either by attaching files to tickets, or by linking tickets directly to models and proofs in the source code repository. In the latter case, one benefits from the model and proof comparison capabilities of Sφnx. Verification tools (currently KeYmaera), are linked to the modeling tool by implementing extensions to the Eclipse launch configuration. These extensions hook into the context menu of Eclipse (models in dℒ and proof files in KeY in our case) and, on selection, launch an external program.

As a vision of extending collaboration support, it is planned to integrate Wikis and other online collaboration tools (currently, we use Redmine both as project management repository and for knowledge exchange) for exchanging knowledge on proof tactics. Additionally, collaboration with experts outside the own organization can be fostered by linking to Web resources, such as MathOverflow[11] and Amazon Mechanical-Turk[12]. Especially interesting, in this respect, is the possibility to create a social-network-like expert platform. In such a platform, requests could be forwarded to those experts whose knowledge matches the verification problem best.

4 Application Example

With the increased introduction of autonomous robotic ground vehicles as consumer products—such as autonomous hovers and lawn mowers, or even accepting driverless cars on regular roads in California—we face an increased need for ensuring product safety not only for the good of our consumers, but also for the sake of managing manufacturer liability. One important aspect in building such systems is to make them scrutable, in order to mitigate unrealistic expectations and increase trust [38]. In the design stage of such systems, formal verification techniques ensure correct functioning w.r.t. some safety condition, and thus, increase trust. In the course of this, formal verification techniques can help to make assumptions explicit and thus clearly define what can be expected from the system under which circumstances.

We discuss a model of an autonomous robotic ground vehicle and its proof (to increase trust), and describe how we can derive bounds on the behavior of that vehicle. The sample autonomous robotic ground vehicle used in this paper operates on predefined tracks and, thus, cannot steer freely (i. e., single wheel drive with angular velocity zero). The control options of such vehicles are limited to choosing the value of acceleration and result in sequences of straight lines as trajectories. The trajectories are thus akin to those produced by our previous car models [23],[24].

In this example, navigation of autonomous robotic ground vehicles is considered safe, if such vehicles are able to stay within their assigned area (e. g., on a track) and do not actively crash with obstacles. Since we cannot guarantee reasonable behavior of obstacles, however, autonomous ground vehicles are allowed to passively crash (i. e., while obstacles might run into the robot, the robot will never move into a position where the obstacle could not avoid a collision).

[6] subversion.apache.org
[7] subclipse.tigris.org
[8] www.eclipse.org/mylyn
[9] www.bugzilla.org
[10] www.redmine.org
[11] mathoverflow.net
[12] www.mturk.com

Model 1 Single wheel drive without steering (one-dimensional robot navigation)

$$swd \equiv (ctrl; dyn)^* \tag{1}$$
$$ctrl \equiv (ctrl_r \parallel ctrl_o) \tag{2}$$
$$ctrl_r \equiv (a_r := -b) \tag{3}$$
$$\cup\ (?safe;\ a_r := *;\ ?-b \leq a_r \leq A) \tag{4}$$
$$\cup\ (?v_r = 0;\ a_r := 0;\ o_r := *;\ ?o_r^2 = 1) \tag{5}$$
$$safe \equiv x_{\underline{b}} + \frac{1-o_r}{2} \cdot \left(\frac{v_r^2}{2b} + \left(\frac{A}{b}+1\right) \cdot \left(\frac{A}{2} \cdot \varepsilon^2 + \varepsilon \cdot v_r\right)\right) < x_r < x_{\overline{b}} - \frac{1+o_r}{2} \cdot \left(\frac{v_r^2}{2b} + \left(\frac{A}{b}+1\right) \cdot \left(\frac{A}{2} \cdot \varepsilon^2 + \varepsilon \cdot v_r\right)\right) \tag{6}$$
$$\wedge\ \|x_r - x_o\| \geq \frac{v_r^2}{2b} + \left(\frac{A}{b}+1\right) \cdot \left(\frac{A}{2} \cdot \varepsilon^2 + \varepsilon \cdot v_r\right) + V \cdot \left(\varepsilon + \frac{v_r + A \cdot \varepsilon}{b}\right) \tag{7}$$
$$ctrl_o \equiv \left(?v_o = 0;\ o_o := *;\ ?o_o^2 = 1\right) \tag{8}$$
$$\cup\ (v_o := *;\ ?0 \leq v_o \leq V) \tag{9}$$
$$dyn \equiv (t := 0;\ x_r' = o_r \cdot v_r,\ v_r' = a_r,\ x_o' = o_o \cdot v_o,\ t' = 1\ \&\ v_r \geq 0 \wedge v_o \geq 0 \wedge t \leq \varepsilon) \tag{10}$$

4.1 Modeling

Model 1 shows the model of a hybrid system comprising the control choices of an autonomous robotic ground vehicle, the control choices of a moving obstacle, and the continuous dynamics of the system. The system represents the common controller-plant model: it repeatedly executes control choices followed by dynamics, cf. (1). The control of the robot is executed in parallel to that of the obstacle, cf. (2).

The robot has three options: It is always allowed to brake, as expressed by cf. (3) having no test condition. If its current state is safe (defined by (6)), then the robot may accelerate with any rate within its physical bounds, cf. (4). For this, we utilize the modeling pattern introduced above: we assign an arbitrary value to the robot's acceleration state ($a_r := *$), which is then restricted to any value from the interval $(-b, A)$ using a test ($?-b \leq a_r \leq A$). Finally, if the robot is stopped, it may choose to remain in its current spot and may or may not change its orientation while doing so, cf. (5). This is expressed again by arbitrary assignment with subsequent test: this time, the test $?o_r^2 = 1$, however, restricts the orientation value to either forwards or backwards ($o_r \in \{1, -1\}$).

For always remaining safely inside its area, the robot must account for (i) its own braking distance ($\frac{v^2}{2b}$), (ii) the distance it may travel with its current velocity ($\varepsilon \cdot v_r$) until it is able to initiate braking, and (iii) the distance needed to compensate the acceleration A that may have been chosen in the worst case, cf. (6). Note, that the safety margin applies to either the upper or the lower bound of the robot's area, depending on the robot's orientation: when driving forward (i. e., towards the upper bound), we do not need a safety margin towards the lower bound, and vice versa. This is expressed by the factors $\frac{1-o_r}{2}$ and $\frac{1+o_r}{2}$, which mutually evaluate to zero (e. g., $\frac{1-o_r}{2} = 0$ when driving forward with $o_r = 1$). The distance between the robot and the obstacle must be large enough to (i) allow the robot to brake to a stand-still, (ii) compensate its current velocity and worst-case acceleration, and (iii) account for the obstacle moving towards the robot with worst-case velocity V while the robot is still not stopped, cf. (7). Note, that w.r.t. the obstacle we have to be more conservative than towards the bounds, because we want to be able to come to a full stop even when the obstacle approaches the robot from behind.

The obstacle, essentially, has similar control options as the robot (with the crucial difference of not having to care about safety): it may either remain in a spot and possibly change its orientation (8), or choose any velocity up to V, cf. (9).

4.2 Verification

We verify the safety of acceleration and orientation choices as modeled in Model 1 above, using a formal proof calculus for $\mathsf{d}\mathcal{L}$ [28, 30]. The robot is safely within its assigned area and at a safe distance to the obstacle, if it is able to brake to a complete stop at all times[13]. The following condition captures this requirement as an invariant that we want to hold at all times during the execution of the model:

$$\begin{aligned} r \text{ stoppable } (o, b) \equiv\ & \|x_r - x_o\| \ \geq\ \frac{v_r^2}{2b} + \frac{v \cdot V}{b} \\ & \wedge\ x_{\underline{b}} + \frac{1-or}{2} \cdot \frac{v_r^2}{2b} < x_r < x_{\overline{b}} - \frac{1+or}{2} \cdot \frac{vr^2}{2b} \\ & \wedge\ v_r \geq 0 \wedge o_r^2 = 1 \\ & \wedge\ o_o^2 = 1 \wedge 0 \leq v_o \leq V \end{aligned}$$

The formula states that the distance between the robot to both the obstacle and the bounds is safe, if there is still enough distance for the robot to brake to a complete stop before it reaches either. Also, the robot must drive with positive velocity, the chosen directions of robot and obstacle must be either forwards ($o_r = 1$) or backwards ($o_r = -1$), and the obstacle must use only positive velocities up to V.

Theorem 1 (Safety of single wheel drive)**.** *If a robot is inside its assigned area and at a safe distance from the obstacle's*

[13] The requirement that the robot has to ensure an option for the obstacle to avoid a collision is ensured trivially, since the obstacle in this model can choose its velocity directly. In a more realistic model the obstacle would choose acceleration instead; then the robot had to account for the braking distance of the obstacle, too

position x_o initially, then it will not actively collide with the obstacle and stay within its area while it follows the `swd` *control model (Model 1), as expressed by the provable* $\mathsf{d}\mathcal{L}$ *formula:*

$$r \text{ stoppable } (o,b) \rightarrow [swd]\big((v>0 \rightarrow \|p_r - p_o\| > 0) \\ \wedge x_{\underline{b}} < x_r < x_{\overline{b}}\big)$$

We proved Theorem 1 using KeYmaera. With respect to making autonomous systems more scrutable, such a proof may help in a twofold manner: on the one hand, it may increase trust in the implemented robot (given the assumption that the actual implementation can be traced back to the abstract model). On the other hand, it makes the behavior of the robot more understandable. In this respect, the most interesting properties of the proven model are the definition of $safe$ and the invariant, which allow us to analyze design tradeoffs and tell us what is always true about the system regardless of its state. As an example, let us consider the distance between the robot and the obstacle that is considered safe: $\|x_r - x_o\| \geq \frac{v_r^2}{2b} + \big(\frac{A}{b}+1\big)\cdot\big(\frac{A}{2}\cdot\varepsilon^2 + \varepsilon\cdot v_r\big) + V\cdot\big(\varepsilon + \frac{v_r + A\cdot\varepsilon}{b}\big)$. This distance can be interpreted as the minimum distance that the robot's obstacle detection sensors are required to cover; it is a function of other robot design parameters (maximum velocity, braking power, worst-case acceleration, sensor/processor/actuator delay) and the parameters expected in the environment (obstacle velocity). $\|x_r - x_o\|$ can be optimized w.r.t. different aspects: for example, to find the most cost-efficient combination of components that still guarantees safety, to specify a safe operation environment given a particular robot configuration, or to determine time bounds for algorithm optimization.

With respect to the manual guidance and collaboration needed in such a proof, we had to apply knowledge in hybrid systems and in-depth understanding of the robot model to find a system invariant, which is the most important manual step in the proof above. We further used arithmetic interactions, such as the hiding of superfluous terms to reduce arithmetic complexity, transforming and replacing terms (e. g., substitute the absolute function with two cases, one for negative and one for positive values).

4.3 Model Variants and Proof Structure

Since it is hard to come up with a fully verifiable model that includes all the details right from the beginning, the models discussed in the previous section are the result of different modeling and verification variants. In the process of creating these models, different assumptions and simplifications were applied until we reached the version in Model 1. For example, one can make explicit restrictions on particular variables, such as first letting the robot start in a known direction (instead of an arbitrary direction). Such assumptions and simplifications, of course, are not without implications on the proof. While in some aspect a proof may become easier, it may become more laborious or more complex in another. In this section, we discuss five variants of the single wheel drive model (without obstacle) to demonstrate implications on the proof structure and on the entailed manual guidance needed to complete a proof in KeYmaera.

The following model variants are identical in terms of the behavior of the robot. However, assumptions on the starting direction were made in the antecedent of a provable $\mathsf{d}\mathcal{L}$ formula, and the starting direction as well as the orientation of the robot were explicitly distinguished by disjunction or nondeterministic choice, or implicitly encoded in the arithmetic, as described below.

Assumed starting direction, orientation by disjunction In the first variant, the robot is assumed to start in a known direction, specified in the antecedent of $o_r = 1 \ldots \rightarrow [swd](x_{\underline{b}} < x_r < x_{\overline{b}})$. Also, the orientation of the robot is explicitly distinguished by disjunction in $safe \equiv (o_r = -1 \wedge x_{\underline{b}} + \ldots < x_r) \vee (o_r = 1 \wedge x_r < x_{\overline{b}} - \ldots)$, and the robot had an explicit choice on turning during stand-still $(?v_r = 0;\ o_r := -o_r; \ldots) \cup (?v_r = 0; \ldots)$.

Orientation by arithmetic In the second variant, we kept the assumed starting direction of the first variant. However, the orientation by disjunction in the definition of $safe$ was replaced by using o_r as discriminator value encoded in the arithmetic, as in $safe \equiv x_{\overline{b}} - \frac{1+o_r}{2}\cdot(\ldots) < x_r < x_{\underline{b}} + \frac{1-o_r}{2}\cdot(\ldots)$.

Arbitrary starting direction by disjunction The third variant relaxes the assumption on the starting direction by introducing a disjunction of possible starting directions in the antecedent of the provable formula $(o_r = 1 \vee o_r = -1) \ldots \rightarrow [swd](x_{\underline{b}} < x_r < x_{\overline{b}})$.

Arbitrary starting direction by arithmetic The fourth variant replaces the disjunction in the antecedent by stating the two orientation options as $o_r^2 = 1$ in $o_r^2 = 1 \ldots \rightarrow [swd](x_{\underline{b}} < x_r < x_{\overline{b}})$.

Replace non-deterministic choice with arithmetic Finally, we replace the non-deterministic turning choice with $(?v_r = 0;\ o_r := *;\ ?o_r^2 = 1; \ldots)$.

Table 2 summarizes the proof structures of the five variants. Unsurprisingly—when considering the rules of the $\mathsf{d}\mathcal{L}$ proof calculus [29] as listed in Table 2—disjunctions in the antecedent ($\vee l$) or in tests of hybrid programs, as well as non-deterministic choices ($[\cup]$) increase the number of proof branches and with it the number of manual proof steps. The number of proof branches can be reduced, if we can replace disjunctions in the antecedent (but also conjunctions in the consequent) or non-deterministic choices in the hybrid program by an equivalent arithmetic encoding. Conversely, this means that some arithmetic problems can be traded for easier ones with additional proof branches.

5 Conclusion

In this paper, we gave a vision of a verification-driven engineering toolset including hybrid and arithmetic verification tools, and introduced modeling and collaboration tools with the goal of making formal verification of hybrid systems accessible to a broader audience. The current implementation features textual and graphical modeling editors, integration of KeYmaera as a hybrid systems verification tool, model and proof comparison, and connection to various collaboration backend systems. The VDE toolset is currently being tested in a collaborative verification setting between Carnegie Mellon University, the University of Cambridge, and the University of Edinburgh.

Table 2: Nodes, branches, and manual proof steps of variants

	Variant	Nodes	Branches	Manual steps	Avoids
(i)	Assumed starting direction, orientation by disjunction	387	34	24	
(ii)	Orientation by arithmetic	331	28	25	$(\vee l)$
(iii)	Arbitrary starting direction by disjunction	650	56	44	
(iv)	Arbitrary starting direction by arithmetic	185	17	22	$(\vee l)$
(v)	Replace non-deterministic choice	160	14	29	$([\cup])(\wedge r)$

$$\frac{\Gamma, \phi \vdash \Delta \quad \Gamma, \psi \vdash \Delta}{\Gamma, \phi \vee \psi \vdash \Delta}\ (\vee l) \qquad \frac{[a]\phi \wedge [b]\phi}{[a \cup b]\phi}\ ([\cup]) \qquad \frac{\Gamma \vdash \phi, \Delta \quad \Gamma \vdash \psi, \Delta}{\Gamma \vdash \phi \wedge \psi, \Delta}\ (\wedge r)$$

Acknowledgments

This material is based upon work supported by the National Science Foundation under NSF CAREER Award CNS-1054246, NSF EXPEDITION CNS-0926181, and under Grant Nos. CNS-1035800 and CNS-0931985, by DARPA under agreement number FA8750-12-2-0291, and by the US Department of Transportation's University Transportation Center's TSET grant, award# DTRT12GUTC11. Passmore was also supported by the UK's EPSRC [grants numbers EP/I011005/1 and EP/I010335/1]. Mitsch was also supported by the Austrian Federal Ministry of Transport, Innovation and Technology (BMVIT) under grant FFG FIT-IT 829589, FFG BRIDGE 838526, and FFG Basisprogramm 838181.

References

[1] Rajeev Alur, 'Formal verification of hybrid systems', in *Proceedings of the 11th International Conference on Embedded Software (EMSOFT)*, eds., Samarjit Chakraborty, Ahmed Jerraya, Sanjoy K. Baruah, and Sebastian Fischmeister, pp. 273–278. ACM, (2011).

[2] Manas Bajaj, Andrew Scott, Douglas Deming, Gregory Wickstrom, Mark De Spain, Dirk Zwemer, and Russell Peak, 'Maestro—a model-based systems engineering environment for complex electronic systems', in *Proceedings of the 22nd Annual INCOSE International Symposium*, Rome, Italy, (2012). INCOSE.

[3] Clark Barrett, Aaron Stump, and Cesare Tinelli. The SMT-LIB standard: Version 2.0. `http://smtlib.cs.uiowa.edu/papers/smt-lib-reference-v2.0-r12.09.09.pdf`, 2012. (last accessed: 2013-01-09).

[4] Maria Christakis, Peter Müller, and Valentin Wüstholz, 'Collaborative verification and testing with explicit assumptions', in *FM*, eds., Dimitra Giannakopoulou and Dominique Méry, volume 7436 of *LNCS*, 132–146, Springer, (2012).

[5] Pieter Collins and John Lygeros, 'Computability of finite-time reachable sets for hybrid systems', in *44th IEEE Conference on Decision and Control and European Control Conference (CDC-ECC)*, pp. 4688–4693. IEEE, (dec. 2005).

[6] James H. Davenport and Joos Heintz, 'Real quantifier elimination is doubly exponential', *J. Symb. Comput.*, **5**(1-2), 29–35, (February 1988).

[7] Leonardo De Moura and Nikolaj Bjørner, 'Z3: an efficient smt solver', in *TACAS*, pp. 337–340, Budapest, Hungary, (2008). Springer-Verlag.

[8] Bart De Schutter, W.P.M.H. Heemels, Jan Lunze, and Christophe Prieur, 'Survey of modeling, analysis, and control of hybrid systems', in *Handbook of Hybrid Systems Control – Theory, Tools, Applications*, eds., Jan Lunze and Françoise Lamnabhi-Lagarrigue, chapter 2, 31–55, Cambridge University Press, Cambridge, UK, (2009).

[9] Patricia Derler, Edward A. Lee, and Alberto Sangiovanni-Vincentelli, 'Modeling cyber-physical systems', *Proceedings of the IEEE*, **100**(1), 13 –28, (jan. 2012).

[10] Johannes Faber, Sven Linker, Ernst-Rüdiger Olderog, and Jan-David Quesel, 'Syspect - modelling, specifying, and verifying real-time systems with rich data', *International Journal of Software and Informatics*, **5**(1-2), 117–137, (2011).

[11] Goran Frehse, 'PHAVer: Algorithmic verification of hybrid systems past HyTech', in *Hybrid Systems: Computation and Control, 8th International Workshop, HSCC 2005, Zurich, Switzerland, March 9-11, 2005, Proceedings*, eds., Manfred Morari and Lothar Thiele, volume 3414 of *LNCS*, pp. 258–273. Springer, (2005).

[12] Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler, 'SpaceEx: Scalable verification of hybrid systems', in *CAV*, ed., Shaz Qadeer Ganesh Gopalakrishnan, LNCS. Springer, (2011).

[13] Aniruddha S. Gokhale, Krishnakumar Balasubramanian, Arvind S. Krishna, Jaiganesh Balasubramanian, George Edwards, Gan Deng, Emre Turkay, Jeff Parsons, and Douglas C. Schmidt, 'Model driven middleware: A new paradigm for developing distributed real-time and embedded systems', *Sci. Comput. Program.*, **73**(1), 39–58, (2008).

[14] Timothy Gowers and Michael Nielsen, 'Massively collaborative mathematics', *Nature*, **461**, 879–881, (2009).

[15] Thomas C. Hales, John Harrison, Sean McLaughlin, Tobias Nipkow, Steven Obua, and Roland Zumkeller, 'A revision of the proof of the Kepler conjecture', *Discrete & Computational Geometry*, **44**(1), 1–34, (2010).

[16] Matthew Clayton Hause and Francis Thom, 'An integrated MDA approach with SysML and UML', in *Proc. of the 13th Intl. Conference on Engineering of Complex Computer Systems*, ICECCS '08, pp. 249–254, Washington, DC, USA, (2008). IEEE Computer Society.

[17] Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, and Marco A. A. Sanvido, 'Extreme model checking', in *Verification: Theory and Practice, Essays Dedicated to Zohar Manna on the Occasion of His 64th Birthday*, ed., Nachum Dershowitz, volume 2772 of *LNCS*, pp. 332–358. Springer, (2003).

[18] Frédéric Jouault, Freddy Allilaire, Jean Bézivin, and Ivan Kurtev, 'ATL: A model transformation tool', *Sci. Comput. Program.*, **72**(1-2), 31–39, (2008).

[19] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood, 'seL4: formal verification of an OS kernel', in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pp. 207–220, New York, NY, USA, (2009). ACM.

[20] Fabrice Kordon, Jérôme Hugues, and Xavier Renault, 'From model driven engineering to verification driven engineering', in *Proc. of the 6th IFIP int. workshop on Software Technologies for Embedded and Ubiquitous Systems*, pp. 381–393. Springer, (2008).

[21] Yanni Kouskoulas, David Renshaw, André Platzer, and Peter Kazanzides, 'Certifying the safe design of a virtual fixture control algorithm for a surgical robot', in *HSCC*, eds., Calin Belta and Franjo Ivancic. ACM, (2013).

[22] Orna Kupferman and Moshe Y. Vardi, 'Modular model checking', in *Revised Lectures from the International Symposium on Compositionality: The Significant Difference*, COM-

POS'97, pp. 381–401, London, UK, (1998). Springer.
[23] Sarah M. Loos, André Platzer, and Ligia Nistor, 'Adaptive cruise control: Hybrid, distributed, and now formally verified', in *FM*, volume 6664 of *LNCS*, pp. 42–56. Springer, (2011).
[24] Stefan Mitsch, Sarah M. Loos, and André Platzer, 'Towards formal verification of freeway traffic control', in *Proc. of the 2nd Int. Conference on Cyber-Physical Systems (ICCPS)*, ed., Chenyang Lu, pp. 171–180. IEEE, (2012).
[25] Wojciech Mostowski, 'The KeY syntax', in *Verification of Object-Oriented Software. The KeY Approach*, eds., Bernhard Beckert, Reiner Hähnle, and Peter H. Schmitt, volume 4334 of *Lecture Notes in Computer Science*, 599–626, Springer Berlin Heidelberg, (2007).
[26] Grant Olney Passmore, *Combined Decision Procedures for Nonlinear Arithmetics, Real and Complex*, Ph.D. dissertation, University of Edinburgh, 2011.
[27] Grant Olney Passmore, Lawrence C. Paulson, and Leonardo Mendonça de Moura, 'Real algebraic strategies for MetiTarski proofs', in *AISC/MKM/Calculemus*, eds., Johan Jeuring, John A. Campbell, Jacques Carette, Gabriel Dos Reis, Petr Sojka, Makarius Wenzel, and Volker Sorge, volume 7362 of *Lecture Notes in Computer Science*, pp. 358–370. Springer, (2012).
[28] André Platzer, 'Differential dynamic logic for hybrid systems.', *J. Autom. Reas.*, **41**(2), 143–189, (2008).
[29] André Platzer, 'Differential-algebraic dynamic logic for differential-algebraic programs', *J. Log. Comput.*, **20**(1), 309–352, (2010).
[30] André Platzer, *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*, Springer, Heidelberg, 2010.
[31] André Platzer, 'Stochastic differential dynamic logic for stochastic hybrid programs', in *CADE*, pp. 431–445, (2011).
[32] André Platzer, 'A complete axiomatization of quantified differential dynamic logic for distributed hybrid systems', *Logical Methods in Computer Science*, **8**(4), 1–44, (2012). Special issue for selected papers from CSL'10.
[33] André Platzer, 'The complete proof theory of hybrid systems', in *LICS*, pp. 541–550. IEEE, (2012).
[34] André Platzer and Edmund M. Clarke, 'Computing differential invariants of hybrid systems as fixedpoints', *Formal Methods in System Design*, **35**(1), 98–120, (2009).
[35] André Platzer and Edmund M. Clarke, 'Formal verification of curved flight collision avoidance maneuvers: A case study', in *FM*, eds., Ana Cavalcanti and Dennis Dams, volume 5850 of *LNCS*, pp. 547–562. Springer, (2009).
[36] André Platzer and Jan-David Quesel, 'KeYmaera: A hybrid theorem prover for hybrid systems.', in *IJCAR*, eds., Alessandro Armando, Peter Baumgartner, and Gilles Dowek, volume 5195 of *LNCS*, pp. 171–178. Springer, (2008).
[37] André Platzer and Jan-David Quesel, 'European Train Control System: A case study in formal verification', in *ICFEM*, eds., Karin Breitman and Ana Cavalcanti, volume 5885 of *LNCS*, pp. 246–265. Springer, (2009).
[38] Nava Tintarev, Nir Oren, Kees Van Deemter, Roman Kutlak, Matt Green, Judith Masthoff, and Wamberto Vasconcelos, 'SAsSy—scrutable autonomous systems', in *to appear in: Proceedings of the 2013 Workshop on Enabling Domain Experts to use Formalised Reasoning (Do-Form)*, (2013).

Interacting with Ontologies and Linked Data through Controlled Natural Languages and Dialogues

Ronald Denaux, Vania Dimitrova and **Anthony G. Cohn** [1]

Abstract. This paper describes a suite of tools developed at the University of Leeds which aim to make it easier for domain experts to be involved in the creation and use of ontologies. The paper summarises the main features of the tools and gives a short summary of our evaluations and experiences using the tools with domain experts.

1 Expressing Knowledge through a Controlled Natural Language

At the core of our suite of tools for supporting domain experts when using ontologies is the use of a Controlled Natural Language (CNL) as the main way to express knowledge in a way that:

- is easy to understand and write by domain experts and
- can be automatically translated into a logical form, in particular an ontology language

We have adopted Rabbit, a CNL designed by the Ordnance Survey – the mapping agency of Great Britain – to enable domain experts (e.g. cartographers) to contribute to the development of ontologies for describing the Ordnance Survey's topographic data [7]. Although inspired by the topographical domain, Rabbit has been designed to be domain independent. The language is designed to be directly mappable to OWL, the web ontology language.

1.1 Tool Support

As part of our adoption of the Rabbit CNL, we have collaborated with the Ordnance Survey to build a **parser** for the language that translates textual inputs into OWL constructs. This parser uses Natural Language Processing (NLP) techniques to improve the recognition of Rabbit sentences. These techniques are used, for example, to recognise that concepts tend to be expressed as nouns (or noun phrases); similarly, relationships tend to be expressed as verbs. Another example of the use of NLP techniques is in recognising that the same concept may be expressed as either a singular noun or a plural noun.

An important aspect of the Rabbit parser is that, when the input text is not syntactically correct, it is often able to generate **error messages that are easy to understand** and help the domain expert to correct the input (see for example, Figure 1). We refer to [2] for more information about the Rabbit parser.

Finally, other Rabbit tool support we provide is **automatic translation of valid Rabbit sentences into OWL** constructs and **generation of Rabbit sentences** to show the contents of existing ontologies[2]. Since we have developed the tool support in house, we are able to adapt and extend the tools if necessary.

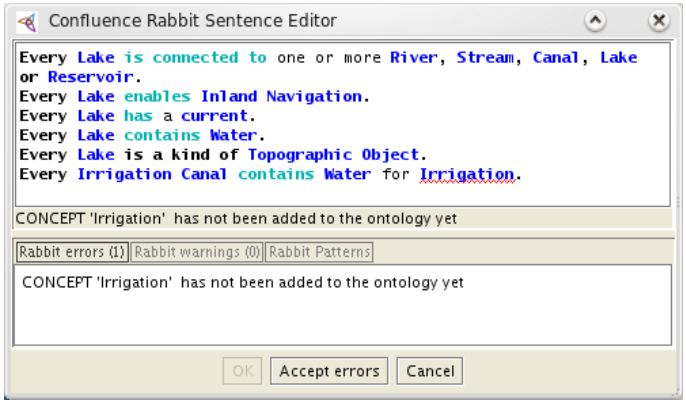


Figure 1. Rabbit Editor component showing various Rabbit sentences, syntax highlighting and error feedback.

1.2 Conclusion

The Ordnance Survey has studied how easy the Rabbit language is to understand [7] and to write [6] with positive results. In the next sections, we will discuss our experiences using the language to support ontology authoring and use of linked data.

Although we are using Rabbit as the CNL in our tools, there are other CNLs which provide comparable tool support and expressivity. We refer to [11] for a comparison to the main alternative CNLs for OWL.

2 Ontology Authoring Methodology Support for Domain Experts

After developing Rabbit, the Ordnance Survey found that domain experts were able to create ontologies using Rabbit, but these ontologies contained modelling errors due to the lack of knowledge about ontology modelling processes by domain experts [7]. The Ordnance Survey adapted existing ontology engineering methodologies to give the domain experts a central role in the ontology authoring process, and we developed tool support to guide domain experts through this methodology. This resulted in an ontology editor called ROO (Rabbit to OWL Ontology authoring).

[1] University of Leeds, England, email: r.denaux@leeds.ac.uk

[2] This only works if the ontologies provide labels for their classes and properties.

ROO provides an alternative user interface for Protege, one of the leading ontology editors. The main advantage of ROO is that it is designed for domain experts:

- knowledge is entered and presented using the Rabbit CNL
- the interface has been simplified to
  1. guide the user through the ontology authoring process: the main interface consists of three tabs, one for defining the scope and purpose of the ontology, a second tab for defining knowledge sources and a third tab to define the ontology concepts and relationships.
  2. avoid OWL specific terminology: for example by using the term 'concept' instead of the OWL specific 'class'. As another example, OWL provides options to use annotations with language tags and various XML types; ROO uses defaults for various annotations to simplify the input by domain experts.

Another feature of ROO for guiding domain experts through the ontology authoring methodology is the **Guide Dog**: when the domain expert is unsure about what to do next, the Guide Dog analyses the current ontology and suggests a next task to perform. Example tasks are: to define the purpose of the ontology, to add a natural language definition for a concept or to add a Rabbit sentence to define a concept.

We performed a comparative evaluation study comparing ROO to a similar CNL-based ontology authoring tool called ACE-View [9]. In the study, Geography and Environmental Studies students and researches at the University of Leeds were given a task to create an ontology about Hydrology and Water Polution. The results were encouraging, showing that domain experts with no previous exposure to ROO were able to create an initial ontology. The results validated our approach for providing tool support for editing CNL sentences and following an ontology authoring ontology [5, 3].

ROO is open source[3] and has more than a 1000 downloads. The ontology editor has successfully been used by several staff members and students at the University of Leeds to build ontologies. For example as part of an EU-project[4], ROO was used by a staff member of the Leeds University Business School to create an ontologies about Activity Theory and Interpersonal Communication.

3 Feedback about Logical Aspects of Ontology Authoring

While Rabbit enables domain experts to directly contribute their knowledge to an ontology and ROO guides them through the process of building ontologies, the problem of modelling errors in the resulting ontologies remained. In practice, this means that the contributions made by domain experts have to be evaluated and corrected by knowledge engineers who have training in the formal logics of ontology languages. However, finding and evaluating such ontology bugs can be difficult, especially if the knowledge engineer is not an expert of the domain at hand. To alleviate this problem, we added tool support for providing semantic feedback: **interactive feedback about the logical consequences of integrating a new fact into an existing ontology**.

In order to provide semantic feedback, we have defined a framework that extends the syntactic analysis performed by the Rabbit parser with an integration analysis based on various existing ontology reasoning services. When the domain expert enters a valid Rabbit sentence, the resulting new ontological fact is classified into one of the following categories:

- **the fact is already in the ontology**: this can be because the syntax of the sentence differs from the syntax used in the ontology;
- **the fact is implied by the ontology**: ontologies consist of a number of stated facts, but those facts can combine and imply new facts that have not been explicitly stated. Knowing about such inferences can be useful to domain experts to detect unwanted inferences but also to avoid redundancy in the ontology;
- **the fact is inconsistent with the ontology**: in this case the new fact contradicts other facts in the ontology. Domain experts typically are not aware of such logical contradiction, thus getting feedback helps them to become aware of existing parts of the ontology and to avoid introducing bugs into the ontology.
- **the fact is novel, but no 'relevant' implications can be found**: this case helps the domain expert to realise that the ontology needs to be more interconnected in order to make more inferences possible. The domain expert may also be expecting the system to make inferences, getting this type of feedback helps them to realise the limitations of the ontology language and its reasoning capabilities.
- **the fact is novel and has 'relevant' implications**: in this case, the system can provide a list of new implications that follow from the new fact. This helps the domain expert to become aware of existing facts in the ontology and to get a feeling for the reasoning capabilities of the ontology languages. Furthermore, it helps to avoid unwanted inferences.

An example input Rabbit sentence with the corresponding feedback is shown below:

Rabbit Input: *Every Teaching Hospital is a kind of Hospital.*

Axiom category [Novel Axiom with new Relevant Implications]

ROO Feedback: This assertion is **novel**: it has not been added to the ontology yet. This input **implies** 6 new relevant facts. Have a look at the list of implications to make sure you agree with the implications. If you do not agree, it may be that you are using the wrong terminology.
Check the new implications:

- *Every Teaching Hospital has footprint a Footprint.*
- *Organisation and Teaching Hospital are mutually exclusive.*
- *Training Centre and Teaching Hospital are mutually exclusive.*
- *Every Teaching Hospital is a kind of Topographic Object.*
- *Every Teaching Hospital is a kind of Place.*
- *Teaching Hospital and University (Institution) are mutually exclusive.*

A full description of the integration analysis performed as well as of the semantic feedback provided for each of the categories is provided in [4]. We have integrated the semantic feedback in ROO and we performed an evaluation to find out what novice users without prior knowledge in ontologies found about the feedback. We compared their impressions of the feedback with experienced knowledge engineers and we found that both novice users as well as experts find the interactive semantic feedback informative, timely and useful [4]. We are not aware of another semantic system that provides this type of information in a manner that is understandable to domain experts.

A current limitation of this tool support is that the feedback helps domain experts to become aware of possible problems, but it does not provide sufficient information to resolve the problems as this requires further analysis of the problem. Also, the evaluation study was performed in a controlled environment; we look forward to evalute the semantic feedback in a production-like context.

[3] The project site is `http://www.comp.leeds.ac.uk/confluence/`

[4] `http://imreal-project.eu/`

4 CNL-Based Dialogue Interface to Ontologies and Linked Data

The final tool in our suite to support domain experts is an **ontology and CNL-based dialogue manager**. The need for such a dialogue manager came from the realisation that CNL-based ontology authoring is similar to a dialogue where the domain expert is telling facts about the domain to the ontology editor, which responds with syntactic or semantic feedback as described above. While our experiences with such an approach for ontology authoring are promising, one limitation is that – depending on the situation – the feedback provided can be repetitive, overwhelming, redundant or insufficient. The dialogue manager enables us to define different feedback strategies for different situations in order to provide more appropriate feedback.

The dialogue manager is based on years of research in the field of dialogue systems [10] and can be seen as an extension to the syntactic analysis provided by the Rabbit parser and the semantic analysis described in Section 3. The dialogue manager adds a level of input analysis in which **each input by the domain expert is assigned an intention**: in ROO until now, the intention is assumed to be to inform the system about facts about the domain, but other intentions may be to query the system about existing facts in the ontology or to tell the system that the domain expert disagrees with an existing fact in the ontology. The dialogue manager also **keeps track of the dialogue history**, which enables the definition of alternative interaction strategies based on the previous inputs. For example, if the domain expert has informed the system about a new fact that is novel without 'relevant' implications, the dialogue manager can now elicit new facts that can make new inferences possible, as shown in turn 10 below:[5]:

1 ROO: Tell me a new fact to add to the ontology.
2 ROO: You can do one of the following:
- Enter a new fact or
- Enter *Skip* if you do not want to tell a new fact or
- Enter *Help* to receive a hint.

3 DE: Bus Line 95 has end point Bodington Hall.
4 ROO: This fact is indeed new.
5 ROO: Normally, entering a new fact enables me to make further inferences but I cannot do this in this case.
6 ROO: Can you tell me something that I should be able to infer?
7 ROO: You can do one of the following:
- Enter a new fact or
- Enter *Skip* if you prefer not to answer this question or
- Enter *Help* to receive a hint.

8 DE: Bus Line 95 stops near Bodington Hall.
9 ROO: Actually, "Bus Line 95 stops near Bodington Hall" is a new fact.
10 ROO: Tell me more general facts about `Bus Route` or `Hall of Residence` or `has end point` or `stops near`.

At the moment, the dialogue manager has been implemented and can be used to simulate the ontology authoring interaction in ROO, as well as to perform simple elicitations as shown above. In the near future we are planning to perform evaluations to determine the benefit of such a dialogue interface for ontology authoring.

4.1 Beyond Ontology Authoring

The dialogue manager makes it possible to establish dialogues for other purposes than ontology authoring. For example, in the context of the ImReal project[6], we are using the dialogue manager to assess the cultural exposure of learners based on countries they have visited. We are reusing a service that determines a list of visited countries of a user based on their Flickr and Twitter profiles[8].

Instead of letting the domain expert build an ontology, in this case we have extracted various ontologies from DBPedia [7] containing general facts about countries (currency, languages, income inequality and human development) as well as cultural facts, in particular gestures that occur in specific countries. We use these ontologies to generate a quiz that is presented to the user in the form of a dialogue. Based on the user's anwers, the dialogue attempts to determine how much cultural exposure the learner had to the visited countries. We encourage the reader to see [1] for more details on this dialogue, the data sources used and, in particular, a video showing an example dialogue session.

5 Conclusion

We have presented a suite of tools for enabling the interaction of domain experts with ontologies and linked data. Our line of research has focused on CNL-based interaction between domain experts and ontologies as well as interaction to facilitate ontology authoring. However, the various techniques described can be adapted to other tasks where domain experts need to interact with logic systems as shown with in our latest work on a dialogue for assessing the cultural exposure of learners, where we have successfully reused linked data (i.e. DBPedia) to drive the dialogue.

REFERENCES

[1] Ronald Denaux. Modelling User Cultural Exposure through Dialogues, 2012. `http://imash.leeds.ac.uk/services/CultaDis/` accessed on 10/12/2012.

[2] Ronald Denaux, Vania Dimitrova, Anthony Cohn, Catherine Dolbear, and Glen Hart, 'Rabbit to OWL: Ontology Authoring with a CNL-Based Tool', in *Controlled Natural Language*, ed., Norbert Fuchs, volume 5972 of *Lecture Notes in Computer Science*, 246–264, Springer Berlin / Heidelberg, (2010).

[3] Ronald Denaux, Catherine Dolbear, Glen Hart, Vania Dimitrova, and Anthony G. Cohn, 'Supporting domain experts to construct conceptual ontologies: A holistic approach', *Web Semantics: Science, Services and Agents on the World Wide Web*, **9**(2), 113–127, (July 2011).

[4] Ronald Denaux, Dhaval Thakker, Vania Dimitrova, and Anthony G. Cohn, 'Interactive Semantic Feedback for Intuitive Ontology Authoring', in *7th International Conference on Formal Ontology in Information Systems*, Graz, (2012).

[5] Vania Dimitrova, Ronald Denaux, Glen Hart, Catherine Dolbear, Ian Holt, and Anthony G. Cohn, 'Involving Domain Experts in Authoring OWL Ontologies', in *Proceedings of the 7th International Semantic Web Conference, ISWC*, pp. 1–16, (2008).

[6] Paula Engelbrecht, Glen Hart, and Catherine Dolbear, 'Talking rabbit: a user evaluation of sentence production', in *Proceedings of the 2009 conference on Controlled natural language*, pp. 56–64, (2010).

[7] G. Hart, M. Johnson, and C. Dolbear, 'Rabbit: Developing a control natural language for authoring ontologies', in *Proceedings of the 5th European Semantic Web Conference*, pp. 348–360. Springer, (2008).

[8] C Hauff and G J Houben, 'Geo-Location estimation of flickr images: social web based enrichment', *Advances in Information Retrieval*, 85–96, (2012).

[9] K. Kaljurand, 'ACE View an ontology and rule editor based on Attempto Controlled English', *5th International Workshop on OWL Experiences and Directions*, (2008).

[10] Staffan Larsson and David R Traum, 'Information state and dialogue management in the TRINDI dialogue move engine toolkit', *Natural Language Engineering*, **6**(3–4), 323–340, (2000).

[11] R. Schwitter, K. Kaljurand, A. Cregan, C. Dolbear, and G. Hart, 'A Comparison of three Controlled Natural Languages for OWL 1.1', in *4th OWL Experiences and Directions Workshop (OWLED DC)*, (2008).

[5] The example dialogue session is in the domain of Points of Interest in Leeds

[6] `http://imreal-project.eu/`

[7] `http://dbpedia.org`

Explaining the Outcome of Knowledge-Based Systems; a discussion-based approach

Martin Caminada[1] and **Mikołaj Podlaszewski**[2] and **Matt Green**[3]

Abstract. Many inferences made in everyday life are only valid in the absence of explicit counter information. This has led to the development of nonmonotonic logics. The kind of reasoning performed by these logics can be difficult to explain to the average end-user of a knowledge based system that implements them. Although the system can still give advice, it is hard for the user to assess the rationale behind this advice. In this paper we propose an argumentation approach that enables the advice to be assessed through an interactive dialogue with the system much like the discussion one might have with a colleague. The aim of thie dialogue is for the system to convince the user that the advice is well-founded.

1 NONMONOTONIC REASONING

Human-style common sense reasoning is inherently nonmonotonic. When new information becomes available, some of our previous beliefs and inferences might no longer be warranted. An often cited example of philosopher and AI researcher John Pollock is that from the fact that an object looks red one might reasonably infer that the object really is red. However, if one later obtains the additional information that the object was in fact illuminated by a red light, one should block the conclusion that the object really is red (unless one also has other reasons to believe so). This kind of reasoning contrasts strongly with the approach taken by formalisms like classical logic. Here, the notion of entailment is essentially monotonic, meaning that whenever one adds new facts, one can only obtain more (possibly the same) and never fewer conclusions.[4]

The need for nonmonotonic reasoning (NMR) comes from the fact that many inferences made in everyday life are defeasible. That is, they are only valid in the absence of explicit counter information. The need to accommodate this type of reasoning in formal logic has led to the field of nonmonotonic logics, of which Default Logic and Circumscription are some well-known examples.

One of the purposes of nonmonotonic logics was to be implemented in knowledge-based systems, which would then be able to assist its users in things like diagnosis and decision making. One of the difficulties, however, was that the kind of reasoning performed by nonmonotonic logics can be notoriously difficult to explain to the average end-user, who has no explicit background in how these formalisms function. Although the resulting system could still give advice, it would be hard for the user to assess how this advice came about, why it is indeed the right advice and whether any objections that the user might have are indeed taken into account. That is, the challenge would be for the system to *justify* its advice in a way that can actually be understood by the user. The failure to address this issue could be seen as one of the reasons why the field of nonmonotonic logics did not manage to come up with any widely used commercial applications.[5]

2 ARGUMENT-BASED REASONING

A new impulse was given in the 1990s, with the development of formalisms for argument-based reasoning (see for instance [17, 13, 14, 19]) which culminated in the landmark paper of Dung [6]. In this paradigm, an argument is essentially an aggregation of reasons that, when taken together, supports a particular conclusion. An argument can be attacked by other arguments (like the argument "the object is red because it looks red" is attacked by the argument "the object is illuminated by a red light, so the fact that it looks red is not a reason for it actually being red"). The idea is that, given the information that is available, one can construct the relevant arguments and examine which arguments attack which other arguments. The result can be visualised in a graph, in which the nodes represent arguments and the arrows represent the attack relation. Given such an *argumentation framework* (as this graph is called in [6]) one should then determine (using a formal criterion) which of the arguments should be accepted, rejected or abstained from having an explicit opinion about. As an example, consider the following hypothetical situation involving three arguments:
A: "We should give the patient aspirin, because he's in pain."
B: "We should not give him aspirin, because he's diabetic and existing research indicates that providing aspirin leads to complications for patients who are diabetic."
C: "The research on which this claim is based has been proven to be flawed and has been refuted by clinical evidence."

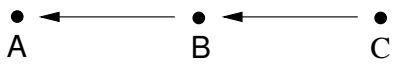


Figure 1. Simple argumentation framework

This situation is graphically depicted in the graph in Figure 1,

[1] University of Aberdeen, Scotland
[2] University of Luxembourg, Luxembourg
[3] University of Aberdeen, Scotland, email: mjgreen@abdn.ac.uk
[4] Formally, when Φ_1 and Φ_2 are sets of formulas in a particular logic of which Cn stands for the consequence relation, then monotonicity means that if $\Phi_1 \subseteq \Phi_2$ then $Cn(\Phi_1) \subseteq Cn(\Phi_2)$.
[5] One notable exception is Answer Set Programming (ASP). ASP, however, is mainly aimed at providing efficient computation for problems involving constraint satisfaction, instead of tackling the original NMR challenge of how to reason with rules of thumb that are subject to exceptions.

where the nodes represent arguments and the arrows represent attacks between the arguments. Here, the idea is that at least argument C should be accepted, because it is not attacked by any other argument. Argument B, however, should be rejected because it is attacked by an argument (C) that is accepted. Argument A is the most interesting case. Its only attacker (B) is rejected and therefore cannot be a valid ground anymore against the acceptance of A. Since there is no other argument attacking A, A should therefore be accepted. This is in line with the general approach of formal argumentation: an argument is accepted unless there are valid grounds against doing so. In this simple example it can be seen that the status of an argument depends on the status of its attackers, which in its turn depends on the status of their respective attackers, etc. However, in more complex graphs (especially those containing cycles) things are not that obvious. In that case, a formal criterion for acceptance and rejection (called an "argumentation semantics") is needed. An example of such a criterion is that of a *complete labelling* [1, 3]. Here, the idea is to assign each argument exactly one label, which can be `in` (indicating acceptance), `out` (indicating rejection) or `undec` (indicating that there are insufficient grounds for either acceptance or rejection).

Existing results in formal argumentation theory state that each graph ("argumentation framework") has at least one complete labelling (that is, an assignment of `in`, `out` and `undec` that satisfies the above three conditions). If the graph contains cycles, more than one complete labelling can exist. Informally, the concept of a complete labelling can be seen as a reasonable position one can take based on the conflicting information encoded in the argumentation framework.

The popularity of argument-based inference formalisms[6] can partly be explained by the facts that:

1. these have been shown to be powerful enough to model a wide range of existing formalisms for nonmonotonic reasoning (like Default Logic [16] and logic programming under various semantics [7, 8, 18]),
2. efficient proof procedures and algorithms are available, and
3. formal argumentation can be seen as a step forward to making formal nonmonotonic inference understandable to end-users

The traditional approach to formal argument-based inference consists of a three-step process. The first step is, given a particular knowledge base, to construct the relevant arguments and examine how they attack each other (that is, to construct the argumentation framework). The second step is to evaluate the resulting argumentation framework (for instance, to determine the complete labellings). The third step is then to examine what this means at the level of conclusions (recall that each argument has at least one conclusion). That is, for each complete labelling of arguments, one determines the associated complete labelling of conclusions (for instance by applying the procedure described in [20]).

One of the things that is missing in the above process is the dialectical aspect. The traditional argumentation process (as for instance formalised in ASPIC [2, 15, 11]) aims at putting all arguments on the table and then simply computing which of them should be accepted. However, in natural argumentation one also encounters the concepts of dialogue and discussion. Where do these concepts fit in when it comes to formal argumentation theory?

[6] Another application of argumentation theory can be found in the field of game theory (see [6] for details). However, in the current paper we focus on argumentation for (nonmonotonic) inference.

3 ARGUMENTATION AS DIALOGUE

A complete labelling can be achieved by assigning a single label to each argument. The following rules show the possible labels:

1. if the argument is labelled `in` (accepted) then all its attackers have to be labelled `out` (rejected)
2. if the argument is labelled `out` (rejected) then it has at least one attacker that is labelled `in` (accepted)
3. if the argument is labelled `undec` (abstained) then not all its attackers are labelled `out` (so there are insufficient grounds to accept it) and it doesn't have an attacker that is labelled `in` (so there are insufficient grounds to reject it)

A dialogue game consists of the following moves:

claim This is the first move in the dialogue. The proponent claims that a particular argument has to be labelled `in`. This creates a commitment that the proponent enters into.

why The opponent asks why a particular claim holds – why a particular argument has to be labelled a particular way

because A party explains why the label of a particular argument has to be the way it was earlier claimed to be.

concede With this move, a party concedes part of the statements uttered earlier by the other party

A dialogue takes place under the following rules:

- The proponent (P) and the opponent (O) take turns. Each turn of P consists of a single move: `claim` or `because`. O plays one or more moves in a turn. O's turn starts with an optional sequence of `concede` moves and finishes (when possible) with a single `why` move.
- P gets committed to arguments used in `claim` and `because` moves; O gets committed to arguments used in `concede` moves.
- P starts with `claim in` (A) where A is the main argument of the discussion: `claim` cannot be repeated later in the game.
- In consecutive turns P provides reasons for the directly preceding `why` ($\mathcal{L}$) move of O by moving `because` ($\mathcal{L}'$) where $\mathcal{L}'$ is a reason of $\mathcal{L}$. [7]
- P can play `because` only if the reason given does not contain any arguments already mentioned (i.e., in P's commitment store) but not yet accepted (i.e., not in O's commitment store). We call such arguments *open issues*.
- O addresses the most recent open issue $\mathcal{L}$ (`in`(A) or `out`(A)) in the discussion. If O is committed to reasons for $\mathcal{L}$ it must concede $\mathcal{L}$ otherwise O starts to question all reasons that O is not committed to with `why`.
- O can question with `why` only one argument at a time.
- The moves `claim`, `because` and `concede` can be played only if new commitments do not contradict a previous one.
- The discussion terminates when no more moves are possible. If O `conceded` the main argument then P wins, otherwise O wins.

Given the argumentation framework in Figure 2 the interaction between a proponent and an opponent may look as set out in Table 1. Recent research has indicated that it is perfectly possible to use the above-sketched dialogue as a basis for formal argumentation theory. The idea is that an argument is accepted iff it can be defended in rational (structured) discussion. The fully specified theory described in [4] together with the with associated implementation [5] allows

[7] A *reason* for $\{(A, \texttt{in})\}$ is $\{(B_1, \texttt{out}) \ldots (B_n, \texttt{out})\}$ where $B_1 \ldots B_n$ are all the attackers of A in the argumentation framework

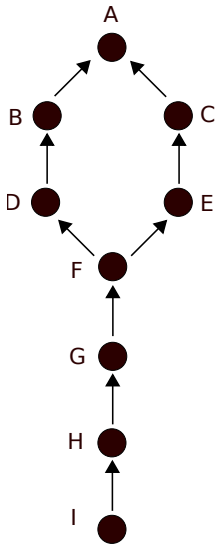


Figure 2. An argumentation framework with more than one path. Nodes represent arguments and arrows indicate attack relations. The dialogue in Table 1 shows how this argumentation framework is traversed, and how the status of argument A is determined by observing that it has to be labelled `in` by every complete labelling.

Table 1. Table shows the interaction between Proponent and Opponent for the argumentation framework in Figure 2

	Commitment stores			
	Proponent		Opponent	
move	in	out	in	out
P:`claim` `in` (A)	A	-	-	-
O:`why` `in` (A)	A	-	-	-
P:`because` `out` (B,C)	A	B,C	-	-
O:`why` `out` (B)	A	B,C	-	-
P:`because` `in` (D)	A,D	B,C	-	-
O:`why` `in` (D)	A,D	B,C	-	-
P:`because` `out` (F)	A,D	B,C,F	-	-
O:`why` `out` (F)	A,D	B,C,F	-	-
P:`because` `in` (G)	A,D,G	B,C,F	-	-
O:`why` `in` (G)	A,D,G	B,C,F	-	-
P:`because` `out` (H)	A,D,G	B,C,F,H	-	-
O:`why` `out` (H)	A,D,G	B,C,F,H	-	-
P:`because` `in` (I)	A,D,G,I	B,C,F,H	-	-
O:`concede` `in` (I)	A,D,G,I	B,C,F,H	I	-
O:`concede` `out` (H)	A,D,G,I	B,C,F,H	I	H
O:`concede` `in` (G)	A,D,G,I	B,C,F,H	I,G	H
O:`concede` `out` (F)	A,D,G,I	B,C,F,H	I,G	H,F
O:`concede` `in` (D)	A,D,G,I	B,C,F,H	I,G,D	H,F
O:`concede` `out` (B)	A,D,G,I	B,C,F,H	I,G,D	H,F,B
O:`why` `out` (C)	A,D,G,I	B,C,F,H	I,G,D	H,F,B
P:`because` `in` (E)	A,D,G,I	B,C,F,H	I,G,D	H,F,B
O:`concede` `in`(E)	A,D,G,I,E	B,C,F,H	I,G,D,E	H,F,B
O:`concede` `out` (C)	A,D,G,I,E	B,C,F,H	I,G,D,E	H,F,B,C
O:`concede` `in`(A)	A,D,G,I,E	B,C,F,H	I,G,D,E,A	H,F,B,C

participants to discuss whether a particular argument has to be accepted by every reasonable position (complete labelling) that can be taken based on the available information (argumentation framework). The rules of the structured discussion are such that the ability to win the discussion against a maximally sceptical opponent coincides with the argument in question being labelled `in` by each and every complete labelling of the argumentation framework. The structured discussion proposed in [4] is based on Mackenzie-style persuasion dialogue [9, 10], where one can apply moves like `claim`, `why`, `because` and `concede` as described above.

The associated implementation [5] uses a command-line interface, and is written in Python. The argumentation framework can either be loaded from a text file or entered manually. At the highest level, the user has eight commands at his disposal: `question`, `claim`, `load`, `save`, `af_cat`, `af_define`, and `quit`. With `question` the user asks the system about the status of a particular argument (say A). The system then responds either with `claim` `in`(A), meaning that A has to be labelled `in` by every complete labelling, with `claim` `out`(A), meaning that A has to be labelled `out` by every complete labelling or with `no commitment` A, meaning that neither is the case. In the first two cases, the associated `claim` move is the start of a persuasion dialogue as described in [4], which the user could choose to bypass by immediately conceding the main claim. When the user does a `claim` command, the system responds either by conceding (if it holds the claim that a particular argument has to be labelled `in` or `out` to be correct) or by holding a persuasion dialogue (if the system holds the claim to be incorrect). Although in the latter case, the discussion will in the end always be won by the system (since the ability to win the persuasion dialogue for a particular argument coincides with the argument being labelled `in` by every complete labelling of the argumentation framework [4]) the discussion might still lead the user to valuable insight about why his initial position was wrong. With the `load`, `save`, `af_cat` and `af_define` commands one respectively loads, saves, displays or manually defines an argumentation framework. The dialogue game follows the rules described in [4], with the exception that parties can terminate the dialogue at any point by conceding or withdrawing the main claim.

The source code (GPL) and other necessary files can be downloaded at the project page [8]. The plan is to keep developing it and integrate it with ArguLab [12]. Furthermore, we are currently working on a theory in which arguments are more than just abstract entities, but have an internal structure consisting of a number of reasons that collectively support a particular claim (conclusion). This would result in a richer formalism, and the resulting discussion could be more natural than is the case when (like in the above example) arguments are completely abstract.

4 CONCLUSION

In general, the ability to express formal inference as the ability to win a particular type of structured discussion can be helpful for providing explanation to end users about why the system derived a particular outcome. If the user disagrees with the system, then one would essentially do the same as when disagreeing with a colleague: start a discussion.

ACKNOWLEDGEMENTS

This work has been supported by the National Research Fund, Luxembourg (LAAMIcomp project) and by the Engineering and

[8] `http://code.google.com/p/pyafl/downloads/list`

Physical Sciences Research Council (EPSRC, UK), grant ref. EP/J012084/1 (SAsSy project).

REFERENCES

[1] M.W.A. Caminada, 'On the issue of reinstatement in argumentation', in *Logics in Artificial Intelligence; 10th European Conference, JELIA 2006*, eds., M. Fischer, W. van der Hoek, B. Konev, and A. Lisitsa, pp. 111–123. Springer, (2006). LNAI 4160.

[2] M.W.A. Caminada and L. Amgoud, 'On the evaluation of argumentation formalisms', *Artificial Intelligence*, **171**(5-6), 286–310, (2007).

[3] M.W.A. Caminada and D.M. Gabbay, 'A logical account of formal argumentation', *Studia Logica*, **93**(2-3), 109–145, (2009). Special issue: new ideas in argumentation theory.

[4] M.W.A. Caminada and M. Podlaszewski, 'Grounded semantics as persuasion dialogue', in *Computational Models of Argument - Proceedings of COMMA 2012*, eds., Bart Verheij, Stefan Szeider, and Stefan Woltran, pp. 478–485, (2012).

[5] M.W.A. Caminada and M. Podlaszewski, 'User-computer persuasion dialogue for grounded semantics', in *Proceedings of BNAIC 2012; The 24th Benelux Conference on Artificial Intelligence*, eds., Jos W.H.M. Uiterwijk, Nico Roos, and Mark H.M. Winands, pp. 343–344, (2012).

[6] P.M. Dung, 'On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games', *Artificial Intelligence*, **77**, 321–357, (1995).

[7] M. Gelfond and V. Lifschitz, 'The stable model semantics for logic programming', in *Proceedings of the 5th International Conference/Symposium on Logic Programming*, eds., R.A. Kowalski and K. Bowen, pp. 1070–1080. MIT Press, (1988).

[8] M. Gelfond and V. Lifschitz, 'Classical negation in logic programs and disjunctive databases', *New Generation Computing*, **9**(3/4), 365–385, (1991).

[9] J. D. Mackenzie, 'Question-begging in non-cumulative systems', *Journal of Philosophical Logic*, **8**, 117–133, (1979).

[10] J. D. Mackenzie, 'Four dialogue systems', *Studia Logica*, **51**, 567–583, (1990).

[11] S.J. Modhil and H. Prakken, 'A general account of argumentation with preferences', *Artificial Intellligence*, (2013). in press.

[12] M. Podlaszewski, Y. Wu, and M. Caminada, 'An implementation of basic argumentation components', in *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 3*, pp. 1307–1308, (2011).

[13] J.L. Pollock, 'How to reason defeasibly', *Artificial Intelligence*, **57**, 1–42, (1992).

[14] J.L. Pollock, *Cognitive Carpentry. A Blueprint for How to Build a Person*, MIT Press, Cambridge, MA, 1995.

[15] H. Prakken, 'An abstract framework for argumentation with structured arguments', Technical Report UU-CS-2009-019, Department of Information and Computing Sciences, Utrecht University, (2009).

[16] R. Reiter, 'A logic for default reasoning', *Artificial Intelligence*, **13**, 81–132, (1980).

[17] G.R. Simari and R.P. Loui, 'A mathematical treatment of defeasible reasoning and its implementation', *Artificial Intelligence*, **53**, 125–157, (1992).

[18] A. van Gelder, K.A. Ross, and J.S. Schlipf, 'The well-founded semantics for general logic programs.', *J. ACM*, **38**(3), 620–650, (1991).

[19] G.A.W. Vreeswijk, 'Abstract argumentation systems', *Artificial Intelligence*, **90**, 225–279, (1997).

[20] Y. Wu and M.W.A. Caminada, 'A labelling-based justification status of arguments', *Studies in Logic*, **3**(4), 12–29, (2010).

Developing an Auction Theory Toolbox

Christoph Lange[1] and **Colin Rowat**[2] and **Wolfgang Windsteiger**[3] and **Manfred Kerber**[4]

Abstract. Auctions allocate trillions of dollars in goods and services every year. Auction design can have significant consequences, but its practice outstrips theory. We seek to advance auction theory with help from mechanised reasoning. To that end we are developing a *toolbox* of formalised representations of key facts of auction theory, which will allow auction designers to have relevant properties of their auctions machine-checked. As a first step, we are investigating the suitability of different mechanised reasoning systems (Isabelle, Theorema, and TPTP) for reproducing a key result of auction theory: Vickrey's celebrated 1961 theorem on the properties of second price auctions – the foundational result in modern auction theory. Based on our formalisation experience, we give tentative recommendations on what system to use for what purpose in auction theory, and outline further steps towards a complete auction theory toolbox.

1 MOTIVATION

Auctions are a widely used mechanism for allocating goods and services (trillions of dollars each year[5]), perhaps second in importance only to market mechanisms. Auctions are used to allocate electromagnetic spectrum, airplane landing slots, bus routes, oil fields, bankrupt firms, internet domains [5], works of art, eBay items, as well as to establish exchange rates, treasury bill yields, and opening prices in stock exchanges. Auction design can have significant consequences. Klemperer attributed the low revenues gained by some governments when auctioning their 3G radio spectrum in 2000 (€20 per capita vs. €600 in other countries) to bad design [11].

Further, the practice of auction design outstrips theory, especially for more complex modern auctions, such as combinatorial auctions in which bids may be submitted on subsets of items (e.g. collections of spectrum, bus routes, landing slots). Important auctions often run 'in the wild' with few formal results [10].

We aim to advance auction theory with help from mechanised reasoning: representing the knowledge underlying auction mechanisms in a formal, explicit way, and verifying these formalisations using computer support. Mechanised reasoning has been successfully applied, e.g., for verifying software that controls commuter rail or payment systems [27]. It has also been applied in economics [9], particularly to social choice theory (cf., e.g., [7]) and game theory (cf., e.g., [22]). However, all of this work has been done by computer scientists, not by economists. The formalisation of (mathematical) theories and the application of mechanised reasoning tools remain novel to economics. Ultimately, we aim to make such techniques more familiar to auction theorists by providing them with a toolbox of basic auction theory formalisations, on top of which they can formalise and verify their own auction designs.

[1] School of Computer Science, University of Birmingham, UK
[2] Department of Economics, University of Birmingham, UK
[3] RISC, Hagenberg, Austria
[4] School of Computer Science, University of Birmingham, UK
[5] Extrapolated from the figure of $268.5 billion reported by the US National Auctioneers Association for 2008 [2].

2 REQUIREMENTS

From the perspective of a domain expert, i.e. an auction designer, the auction theory toolbox (ATT) should satisfy the following requirements:

D1 Provide ready-to-use formalisations of basic concepts of auction theory, including their definitions and their essential properties

D2 Allow for extension and application to custom-designed auctions without requiring expert knowledge of the underlying mechanised reasoning system

From a computer scientist's perspective, these requirements translate to the following, more technical ones:

C1 Identify the right language to formalise auction theory, i.e. a language that is sufficiently expressive for capturing relevant concepts, while supporting efficient proofs for the majority of relevant problems.

C2 Identify a mechanised reasoning system that allows for formalising auction designs in a way that is close to the textbook style economists are used to, and that facilitates reuse of any existing formalisations in the toolbox.

These two requirements cannot be treated independently of each other: a language that is adequate w.r.t. requirement **C1** may not be supported by any mechanised reasoning system that satisfies requirement **C2**.

3 APPROACH

We are building the ATT in parallel to identifying suitable languages and mechanised reasoning systems to avoid a chicken-and-egg problem. The right 'hammers' can only be identified when there are 'nails', i.e. concepts of the application domain to be formalised. In the case of a successful choice of language and system, these initial formalisations will form the core of the future toolbox.

3.1 The Nail: Vickrey's Theorem and Beyond

As the first nail, we chose William Vickrey's 1961 theorem on the properties of second price auctions of a single, indivisible good whose value is not publicly known. In such an auction, each participant submits a sealed bid; one of those with the highest bids wins, and pays the highest of the *remaining* bids; the losers pay nothing.[6] Vickrey proved that 'truth-telling' (i.e. submitting a bid equal to

[6] Wikipedia provides further background, including a discussion of variants used by eBay, Google and Yahoo! [25].

one's actual valuation of the good) was a *weakly dominant* strategy. This means that no bidder could do strictly better by bidding above or below its valuation *whatever* the other bidders did. Thus, the auction is also *efficient*, awarding the item to the bidder with the highest valuation. Vickrey was awarded economics' Nobel prize in 1996 for his work.

We have several reasons for starting our work by redoing an old proof: its formalisation will enable us to prove properties of contemporary related auctions as well; the underlying mathematical theory can be formalised in contemporary systems with reasonable effort; finally, we assume that domain experts being introduced to mechanised reasoning would rather trust this technology, which is new to them, if it is first applied to known results.

Eric Maskin collected high level versions of Vickrey's theorem and 12 others in his 2004 review [13] of Paul Milgrom's influential book on auction theory [14]. This review guides us in building the toolbox; here, however, we focus on Vickrey's theorem (restated as proposition 1 in Maskin's review), as it is the only one we have fully formalised so far.

3.2 Wielding the Hammer

We are formalising auction theory, starting with Vickrey's theorem, in three systems, which differ in logic, syntax and user experience. Before discussing these categories in detail, we discuss how we prepared ourselves for machine formalisation by refining the original paper source; we then introduce the three systems we are using.

3.2.1 Preparing the Paper Formalisation

Maskin's paper states Vickrey's theorem in two sentences of high-level text and proves it in another six sentences. On paper, we elaborated the theorem and its proof into a version that made the details explicit, resulting in eight definitions.[7] Mechanised reasoning systems generally require much more explicit statements than commonly found on paper: Automated theorem provers (cf. the detailed discussion in section 3.2.2) require them to find proofs without running out of search space, whereas proof checkers require proofs to be at a certain level of detail, which in turn requires detailed statements.

An initial attempt to formalise our elaborated proof distinguished cases on the basis of bidders' bids. This generated a multi-level case distinction with nine mostly straightforward leaf cases for Isabelle to check. While feasible, this was tedious, encouraging us to to re-write the cases on the basis of whether a participant won or lost the auction, whether by truthful bidding or otherwise. This resulted in four cases, once more largely straightforward.

We conjecture that such an extra step of elaborating the original paper source will typically be helpful before starting a machine formalisation. The history of proving Arrow's impossibility theorem on the non-existence of a fair aggregation of the preferences of a group of individuals, a central result of social choice theory, supports this claim: Nipkow's Isabelle formalisation [17] as well as Wiedijk's Mizar formalisation [24], both based on Geanakoplos' widely known paper version [6], required such elaborations where the paper source was imprecise. Beyond these formalisations, Tang and Lin obtained insights on the general structure of impossibility results in social choice theory only by studying and formalising a novel, induction-based proof [21].

[7] This 'paper formalisation' is available from the ATT homepage [12].

3.2.2 Choosing a Mechanised Reasoning System

In terms of *logic*, it is not immediately clear whether Vickrey's theorem is inherently a higher-order statement. It does make a statement about the maximum of an arbitrary number n of bids. Defining such an n-vector and proving essential properties of the maximum operation requires induction and thus goes beyond first-order logic. However, if one takes n-vectors and a maximum operation on them for granted, the rest of the formalisation does not require higher-order logic. That is, first-order logic suffices to formalise the concepts that are actually relevant from the perspective of the application domain: single good auctions, second price auctions, and the statement of Vickrey's theorem. First-order logic has the computational advantage that its statements are semi-decidable, and that sound and complete calculi exist. In practice, this means that a number of efficient automated theorem provers are available.

In terms of *syntax*, we assume that our domain experts, i.e. auction designers, will prefer a language that looks like the textbook mathematics they are used to, rather than one that has the flavour of a programming language. We assume that a typed language will support them in defining types for domain concepts (such as bids) and avoiding mistakes in formalisation.

In terms of *user experience*, one has to distinguish between *fully automated proving*, where systems are given a theorem and a knowledge base and they try to automatically find a proof of the theorem w.r.t. the given knowledge base, and *interactive proving*, where the author has to write a proof and have it checked interactively by the system. There are systems that confine themselves to one of these paradigms, but there are also systems that try to combine them.

We roughly describe the features of the languages and systems that we are using:

Isabelle/HOL [8]: higher-order logic (typed), supported by the Isabelle interactive theorem prover, accessible via the ProofGeneral [1] and jEdit [23] text editor interfaces

Theorema [26]: first-order logic with set theory[8], implemented as an add-on package for the Mathematica computer algebra system with its document-oriented notebook interface

TPTP FOF [19, 18]: untyped first-order logic, a machine-oriented language supported by several automated theorem provers, accessible for human authors and developers via the many-sorted first-order logic language CASL[9] [4], which the Hets interface [15] can translate to TPTP and send to provers.[10]

4 STATE, AND EXPERIENCES SO FAR

We performed the first complete formalisation of Vickrey's theorem in Isabelle.[11] We redid the same formalisation in CASL (to obtain TPTP) and thus showed that first-order logic is sufficiently expressive for Vickrey's theorem. Here, the full proof is still work in progress. Finally, we are redoing the formalisation once more in Theorema 2.0. The purpose of redoing the formalisation from scratch is to understand the specific advantages and disadvantages of the different

[8] The Theorema language is actually based on higher-order logic, but in our case we use FOL + set theory.

[9] CASL has a few higher-order features, such as inductive datatypes.

[10] TPTP has a typed (sorted) first-order form (TFF) as well [20]. As Hets is not currently capable of translating CASL to TPTP TFF, we are not using the latter. Note that CASL is more expressive than TPTP TFF in that it supports subsorts.

[11] There was no specific conceptual reason for starting with Isabelle.

languages and systems and to obtain a formalisation that is as idiomatic as possible.[12] The formalisations are available from the ATT homepage [12].

4.1 Theory Structure

Formalising further theorems from Maskin's review paper, some of which make statements about similar types of auctions, should reuse as much of the Vickrey formalisation as possible. To enable this, we have therefore organised them into modular theories. Four of them are essential from the domain perspective:

Single good auctions (basic concepts): the auction as a mechanism that maps bids to an outcome (i.e. uniquely allocating the good to one participant, and defining the participants' payments).
Properties single good auctions may have: efficiency and weak dominance of a given bidding strategy (cf. section 3.1).
Second price auctions: the definition (cf. section 3.1) and some lemmas one can infer from them, e.g. that if there is exactly one highest bidder, that bidder wins; functions to compute the payoff of a winner and a loser.
Vickrey's theorem: truthful bidding in a second price auction is a weakly dominant and efficient strategy.

4.2 Library Coverage

All of our formalisations are structured in terms of the theories listed above.[13] Depending on the system, we had to provide further mathematical foundations as additional theories. Theorema has a built-in tuple datatype which we used to formalise vectors of bids, and supports a maximum operation on such tuples. CASL supports inductive datatypes, and its standard library provides a number of them, including arrays [3, 16], but there is no built-in n-argument maximum operation. The Isabelle/HOL standard library provides a *Max* operation on finite sets; however, given Isabelle's functional programming style syntax, we found it most intuitive to model our own vectors as functions $\mathbb{N} \to \mathbb{R}$ evaluated for arguments up to a given n. Thin wrappers make the set maximum operator usable for these vectors and prove the properties required subsequently.

4.3 Level of Detail Required by the Machine

Even the elaborated paper version introduced in section 3.2.1 turned out to be insufficient for direct machine formalisation. The Isabelle formalisation of the four theories listed above comprises 4 additional, auxiliary definitions, and 7 auxiliary lemmas. We estimate that a similar number of auxiliary statements will be required to guide the automated provers of Theorema and those that support TPTP. On the other hand, our initial steps of extending the formalisation beyond Vickrey's theorem suggest that the auxiliary material makes it easier to formalise further notions.

[12] Hets is, for example, capable of translating CASL to Isabelle, but the resulting Isabelle code would not make use of higher-order features other than inductive datatypes.
[13] In Theorema, which does not support a formal notion of theory, we chose to use notebook sections for structuring. Once further reuse will be required, we may change this to *archives*: collections of definitions, theorems, etc., which a notebook can load for reuse.

4.4 User-friendliness of Input Syntax

While we have not yet collected feedback from actual domain experts, we assume that they will find Theorema's input syntax most accessible. The two-dimensional symbolic notation of Mathematica notebooks is similar to textbook notation; additionally, large parts of our target audience are familiar with Mathematica already. The appearance of Isabelle and CASL is closer to programming languages; however, both allow for defining operators with a custom 'mixfix' notation. The ProofGeneral and jEdit interfaces of Isabelle display the operators built into the language or defined in the standard library in a one-dimensional approximation of textbook style, using the appropriate Unicode characters. Finally, TPTP's pure ASCII syntax is not extensible by custom symbols.

4.5 Interactive or Automated?

Our tentative verdict on interactive vs. automated approaches is that it does not matter: What matters, instead, is that the systems give good error messages, which allow the user to tell where exactly the formalisation is wrong or insufficient, and why exactly the system failed to check or to find a proof. Given that the user provides a proof that proceeds in sufficiently small steps, the jEdit interface for Isabelle gives localised feedback on errors. When Theorema tries to prove a theorem, it develops a structured textbook-style proof at a configurable level of verbosity. The user can navigate it via a tree view and thus quickly identify where Theorema failed to proceed. For CASL, Hets itself performs a type check, and otherwise relies on the output of the theorem provers it invokes.

4.6 Community Support

Community support is another criterion that facilitates adoption of a system. The user communities of the systems considered here mainly comprise computer scientists and therefore may not be ready to answer questions that a domain expert with little computer science background may have, as they often assume previous knowledge about mechanised reasoning, mathematical formalisation, and the specific reasoning approach underlying the respective system. Therefore we simply compare the sizes of the communities for now, assuming that domain experts can expect better assistance from a larger community. Isabelle is developed by multiple institutions and has a large user community; more than 100 posts per month are made to its user mailing list. CASL has been standardised in an international effort and has been the subject of several hundred scientific publications but does not currently have a functional mailing list. Hets is mainly developed and used within a single institution; its user mailing list receives less than 10 posts per month. TPTP has been the subject of more than a thousand publications but does not have a mailing list. Theorema does not have a mailing list either and has so far been developed as closed source software within a single institution, but this is expected to change soon with the open source release of Theorema 2.0.

5 CONCLUSION AND OUTLOOK

Auctions allocate trillions of dollars in goods and services every year, but still it is not well understood how to design a 'good' auction. Our auction theory toolbox (ATT) currently formalises key results about single good second price auctions, a simple, well-understood type of auction. It does so in a modular way, which makes us confident that

we will be able to build formalisations of more complex and new auction types on top of the existing core, ultimately enabling computer-supported verification of auction designs. We are planning to put this hammer right into the hand of domain experts, i.e. auction designers. To that end we are, at the same time as we are building the toolbox, identifying those mechanised reasoning languages and systems whose user experience is closest to the domain experts' mindset.

The logics, languages and systems studied so far satisfy our technical requirements **C1** and, thanks to their support for modular theories, **C2** as well. Our Isabelle formalisation, the most complete one so far, satisfies the domain expert's requirement **D1** for second price auctions but needs to be expanded to other types of auctions. An assessment of how well the ATT satisfies requirement **D2** can only be made once there is a larger core ATT, on top of which we – or rather: auction designers themselves – will have formalised new auctions.

ACKNOWLEDGEMENTS

We would like to thank Makarius Wenzel for considerably improving the Isabelle formalisation of Vickrey's theorem, Till Mossakowski for his help with the CASL formalisation and with using Hets, Marco B. Caminati for a productive exchange of ideas concerning his Mizar formalisation of Vickrey's theorem, as well as Geoff Sutcliffe for his input on TPTP.

References

[1] David Aspinall et al. *Proof General*. 5th Nov. 2012. URL: http://proofgeneral.inf.ed.ac.uk/ (visited on 2012-12-10).

[2] National Auctioneers Association. *Auctions: The Past, Present and Future*. 2010. URL: http://www.sarasotabestauctions.com/resources/History_of_Auctions_3.10.10.ppt.

[3] M. Bidoit and Peter D. Mosses. *CASL — the Common Algebraic Specification Language: User Manual*. LNCS 2900. Springer Verlag, 2004.

[4] *CASL*. CoFI. 12th Feb. 2008. URL: http://www.informatik.uni-bremen.de/cofi/wiki/index.php/CASL (visited on 2012-12-10).

[5] Cramton Associates. *Applicant Auction for Top-Level Domains*. 2012. URL: http://applicantauction.com (visited on 2012-12-05).

[6] John D. Geanakoplos. 'Three brief proofs of Arrow's impossibility theorem'. *Economic Theory* **26**(1) (2005), pp. 211–215.

[7] Christian Geist and Ulle Endriss. 'Automated search for impossibility theorems in social choice theory: ranking sets of objects'. *Journal of Artificial Intelligence Research* **40** (2011), pp. 143–174.

[8] *Isabelle*. 5th May 2012. URL: http://isabelle.in.tum.de (visited on 2012-09-14).

[9] Manfred Kerber, Christoph Lange and Colin Rowat. *An economist's guide to mechanized reasoning or My computer just proved 84 impossibility theorems*. Ed. by Kenneth Judd. Invited lecture at the Initiative for Computational Economics summer school. 25th July 2012. URL: http://cs.bham.ac.uk/research/projects/formare/pubs/ice2012/2012-07-25-ice-mech-reas-econ.pdf.

[10] Paul Klemperer. 'The product-mix auction: a new auction design for differentiated goods'. *Journal of the European Economic Association* **8**(2–3) (2010), pp. 526–536.

[11] Paul Klemperer. 'What Really Matters in Auction Design'. *Journal of Economic Perspectives* **16**(1) (2002), pp. 169–189.

[12] Christoph Lange et al. *Auction Theory Toolbox*. 13th Jan. 2013. URL: http://www.cs.bham.ac.uk/research/projects/formare/code/auction-theory/ (visited on 2013-01-13).

[13] Eric Maskin. 'The unity of auction theory: Milgrom's master class'. *Journal of Economic Literature* **42**(4) (2004), pp. 1102–1115. URL: http://scholar.harvard.edu/files/maskin/files/unity_of_auction_theory.pdf.

[14] Paul Milgrom. *Putting auction theory to work*. Churchill lectures in economics. Cambridge University Press, 2004.

[15] Till Mossakowski. *Hets: the Heterogeneous Tool Set*. URL: http://www.dfki.de/cps/hets (visited on 2012-12-10).

[16] Peter D. Mosses, ed. *CASL Reference Manual*. LNCS 2960. Springer Verlag, 2004.

[17] Tobias Nipkow. 'Social choice theory in HOL: Arrow and Gibbard-Satterthwaite'. *Journal of Automated Reasoning* **43**(3) (2009), pp. 289–304.

[18] Geoff Sutcliffe. 'The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0'. *Journal of Automated Reasoning* **43**(4) (2009), pp. 337–362.

[19] Geoff Sutcliffe and Christian Suttner. *The TPTP Problem Library for Automated Theorem Proving*. URL: http://www.tptp.org (visited on 2012-12-10).

[20] Geoff Sutcliffe et al. 'The TPTP Typed First-order Form with Arithmetic'. In: *Proceedings of the 18th International Conference on Logic for Programming Artificial Intelligence and Reasoning*. (Merida, Venezuela). Ed. by Nikolaj Bjørner and Andrei Voronkov. Lecture Notes in Artificial Intelligence 7180. Springer-Verlag, 2012, pp. 406–419.

[21] Pingzhong Tang and Fangzhen Lin. 'Computer-aided proofs of Arrow's and other impossibility theorems'. *Artificial Intelligence* **173**(11) (2009), pp. 1041–1053.

[22] Pingzhong Tang and Fangzhen Lin. 'Discovering theorems in game theory: two-person games with unique pure Nash equilibrium payoffs'. *Artificial Intelligence* **175**(14–15) (2011), pp. 2010–2020.

[23] Makarius Wenzel. 'Isabelle/jEdit – a Prover IDE within the PIDE framework'. In: *Intelligent Computer Mathematics*. Conferences on Intelligent Computer Mathematics (CICM) (Bremen, Germany, 9th–14th July 2012). Ed. by Johan Jeuring et al. LNAI 7362. Berlin and Heidelberg: Springer Verlag, 2012, pp. 468–471. arXiv: 1207.3441 [cs.LO].

[24] Freek Wiedijk. 'Formalizing Arrow's theorem'. *Sādhanā* **34**(1) (2009), pp. 193–220.

[25] Wikimedia Foundation, ed. *Vickrey auction. From Wikipedia, the free encyclopedia*. 15th Nov. 2012. URL: http://en.wikipedia.org/w/index.php?title=Vickrey_auction&oldid=523230741 (visited on 2012-12-05).

[26] Wolfgang Windsteiger. 'Theorema 2.0: A Graphical User Interface for a Mathematical Assistant System'. In: 10th UITP workshop (at CICM 2012) (Bremen, Germany, 11th July 2012). Ed. by Cezary Kaliszyk and Christoph Lüth. 2012, pp. 1–8. URL: http://www.informatik.uni-bremen.de/uitp12/.

[27] Jim Woodcock et al. ‘Formal method: practice and experience’. *ACM Computing Surveys* **41**(4) (2009), pp. 1–40.

Model Validation and Test Portfolios in Financial Regulation

Neels Vosloo[1]

Abstract. We describe the problem of test portfolio selection and construction that arises in the context of regulators' use of test portfolios to validate and benchmark the models that financial firms use to determine their regulatory capital requirements. We provide an example and discuss whether Mechanised Reasoning might be used to help provide a more structured approach to this problem.

1 INTRODUCTION

Regulators allow the use of financial firms' own ("internal") models for calculation of regulatory capital requirements for several risk types and business areas – the focus of this note is on market risk in the trading book, i.e. fluctuations in value of assets held with "trading intent". Such assets are required to be re-valued ("marked to market") daily, it is therefore implicitly assumed that a reliable market price is available every business day, and market risk is broadly defined as the uncertainty in the future value of the product due to changes in market prices.

Regulated Firms with trading books are required to hold capital against this risk, to absorb "unexpected" losses due to market moves, and under the under the so-called Basel accords are allowed to use their own ("internal") models to calculate this capital requirement, subject to regulatory approval. This option is mainly used by large investment banks, and the markets and investment banking divisions of large financial groups.

Regulators' model approval approach typically relies on firms' internal model validation and control functions, but this is increasingly being supplemented by hypothetical portfolio exercises (where several firms are given the same portfolios to evaluate), as concerns increase over variability of capital requirements and adequacy of risk capture following the experience of the most recent financial crisis. The importance of test portfolios will increase, as regulators are becoming more interested in the cross-industry benchmarking of models and capital – this is very likely to be reflected in future versions of regulatory requirements.

2 MODELS

The term "model" is used in two distinct ways in the present context: "valuation" models and "capital" models.

Valuation models are used to calculate the value of financial products given a state of the market – note that the values of complex financial instruments are calculated as functions of the (more readily observable) values of simpler instruments, also called "risk factors", for example currency exchange rates.

Capital models are used to calculate capital requirements, and are based on estimating a future distribution of market values (or equivalently, changes in market values from the current value, or "profit & loss") over a given time horizon, and calculating a metric (VaR, Expected Shortfall) from this distribution.

Capital models calculate future distributions of product valuations by first modelling a joint distribution of future values of risk factors, and then using appropriate valuation models for each product, taking the modelled risk factor distribution as input.

3 MODEL VALIDATION

For the validation of capital models, the set of financial products to which the model applies is assumed to be known; the values of these products are in turn dependent on a set of known risk factors.

Test portfolios for the model will therefore be defined either by subsets of the set of products in scope, or by subsets of the set of all risk factors, which will in turn define test portfolios of (potentially simplified) products.

As a simple example, consider a product scope of interest rate swaps and FX forwards. The associated risk factors are the FX spot rate and the interest rate curve, the latter represented by a set of interest rates at different maturities. A test portfolio for this scope could consist of some combination of swaps and FX forwards, or, when analysed in terms of risk factors, might consist of portfolios of FX forwards and zero coupon bonds, the latter having exposure to just one point on the curve.

The validation of capital models can usefully be thought of as having three distinct (although obviously related) objectives.

First, we need to verify that risk factors for individual traded products are appropriately defined, and that the valuation model used for the product *in the capital model* is sufficiently accurate. There are typically two issues in this regard: one, not all risk factors for the product are included in the capital model, and two, the capital model could be using an approximation of the product's valuation function (for efficiency and infrastructure reasons Taylor series approximations are often used). This aspect is typically verified by comparing the actual historical profit and loss (P&L) series for the product with the corresponding P&L calculated by the capital model, i.e. via a form of back testing.

Second, we need to verify that risk factor data used in the capital model is complete and sufficiently granular, i.e. risks that are similar but not the same should be recognised by the model. This aspect is typically verified through test portfolios consisting of simple strategies that give exposure to the risk factors being tested but have little or no exposure to other risk factors. The capital model should not give a zero capital requirement for such a strategy – this would indicate that the model does not distinguish between these risk factors, and therefore misses "basis risk".

Third, the impact of risk factor covariance and distributional assumptions should be tested, to establish whether the model will give sufficiently conservative capital levels for all plausible product combinations (portfolios) and market conditions (normal, stressed). This is determined by properties of the capital model, including the type of metric (e.g. VaR, Expected Shortfall), the amount of data used to calibrate the model, the framework (Historical, Monte Carlo, weighted or un-weighted).

The main test used for this aspect is – again – back testing of capital against historical P&L time series for test portfolios, ideally compared across different model types. The additional dimension in this instance is of course test portfolio selection.

4 EXAMPLES

Products and strategies run by a typical Credit Trading business can be used to illustrate the three aspects of model validation discussed above.

A "high-yield" corporate bond has a low credit rating and therefore a high probability of default. An appropriate valuation model will recognise the fact that, for large moves in the bond's credit spread, corresponding price moves should be limited so that the position cannot lose more than the assumed loss given default indicates. A capital model using only the credit spread delta ("CS01") to price the bond will not measure the risk of the position correctly, as it will linearly extrapolate price moves from spread moves.

A "negative basis trade" is a strategy where the trader owns a bond, and also buys protection on it via CDS, when the CDS spread is low relative to the bond's own credit spread. Many capital models use the CDS spread as a proxy for both the CDS and bond spreads, and will therefore not measure the risk of this strategy correctly.

The capital calculated for a portfolio of CDS contracts on highly correlated names will be highly sensitive to both small changes in position and small changes in risk factors (credit spreads), potentially giving an unstable capital measure.

5 CONCLUSIONS & FUTURE WORK

The crux of the test portfolio problem is that, while back testing individual products and strategies across different models and time periods arguably provide most of the information required to validate pricing and risk factor selection for the purpose of capital modelling, the "portfolio effect" means that not all portfolios are equally well capitalised at all times in all models.

This effect is more pronounced during periods of stress, as the usual relationships between products (and asset classes) break down.

All three aspects discussed in Section 3 are (or should be) individually validated by firms, using a range of approaches including test portfolios. Regulatory test portfolios necessarily need to test all three, interrelated, aspects. The problem is to design a set of portfolios that does this efficiently, and is further complicated by the fact that information on valuation models and data is typically limited.

There is also a significant additional problem: regulators typically cannot directly verify the correctness of the actual implementation of a firm's capital model, i.e. whether it is implemented according to documented specifications in software and hardware.

Given the particular requirements of regulatory test portfolios, we think that Mechanised Reasoning could provide model validators with new procedures to discover efficient test portfolios beyond traditional portfolio replication and optimisations techniques, particularly given that regulatory test portfolios will have a range of success criteria, or that the determination of success criteria might even be part of the test. This contrasts with other techniques where a desired level of risk, return, or exposure is typically first specified.

A contribution to an Auction Theory Toolbox through code and discussion

Marco B. Caminati

Abstract. I am contributing a further mechanization of Vickrey's theorem on weak equilibrium in second price auctions. Submission 6 of stage 1 is an invitation to discuss on early developments in formalizing a toolbox for auction theory, and I wish to add a further viewpoint. My formalization is in Mizar, which is nearer than other systems to the common mathematical language. Auctions being not yet present in the library, there is the chance of taking the first design decisions: one should put effort into making them suitable to build a large future library on them, and at the same time try his best to reduce as much material as possible to the most common mathematical objects, which are well supported in the existing library. I will illustrate how I faced this task.

1 What has been formalized

This AISB submission provides the Mizar formalization of a theorem by Vickrey about weak equilibrium in second price auctions. The mathematics is simply and clearly exposed in [3]. A quick summary follows. The initial data is a vector $\boldsymbol{b}$, containing the bids of each participant. Given this vector, the dynamics of the auction is simply modeled by assuming that each participant has, in his mind, a precise valuation of the auctioned good, which may or may not coincide with the amount of money he bids. The theorem in question says that, in this regard, the best strategy is to make them coincide, in the 'weak' sense: given a random $\boldsymbol{b}$, changing the bid of a participant to his valuation, the payoff of *that* participant does not decrease.

1.1 Defining the payoff to express the theorem

The best possible payoff for a single participant would be given by winning the whole auctioned good without paying anything, in which case it can be quantified by the subjective valuation v he deems the good worth. Given that, generally, any participant gets a fraction ranging from 0 (for losers) to 1 of the auctioned good, and that, of course, any decent auction scheme will impose at least to the winner(s) to disburse an amount of money, such payoff is defined by $vx - p$; here v is the valuation, x the fraction of the good obtained, and p the amount paid.

x and p depend on all the participants' bid, and as such each of them is a component of two distinct vectors having the same length as $\boldsymbol{b}$; $\boldsymbol{x}$ and $\boldsymbol{p}$ are respectively termed the *allocations* and the *payments*, and are calculated from $\boldsymbol{b}$ according to the auction algorithm. Thus, the theorem's wordy statement above can be put into this inequality:[1]

$$v \cdot \mathcal{X}^{II}_{\boldsymbol{b}}(i) - \mathcal{P}^{II}_{\boldsymbol{b}}(i) \leq v \cdot \mathcal{X}^{II}_{\boldsymbol{b}^v_i}(i) - \mathcal{P}^{II}_{\boldsymbol{b}^v_i}(i), \qquad (1)$$

where

1. $\mathcal{X}^{II}$ and $\mathcal{P}^{II}$ calculate, from the bids vector $\boldsymbol{b}$, the allocations and payments vectors respectively, according to the second price auction algorithm;
2. i indexes the considered participant;
3. $\boldsymbol{a}^y_i$ is the vector obtained by changing the i-th component of $\boldsymbol{a}$ into y.

2 An overview of Mizar

The Mizar project (http://www.mizar.org) delivers a few provisions:

1. The Mizar *language* permits to write formulas in first-order set theory which read close to common mathematical language. For example, the formula

   $$X \neq \emptyset \Longrightarrow \exists x (x \in X)$$

   is written

   ```
   X <> {} implies ex x st x in X;
   ```

   In addition to the few reserved words pertaining to the first-order alphabet of set theory, the language specifies grammar and reserved words to invoke the verifier (see point 2) and to exploit advanced features of the system.
2. The Mizar *verifier* (PC Mizar) is a piece of software certifying whether one such formula can be deduced (according to some formal system for classical logic, see sections 2.2.1 and 3.5 of [2]) from other given formulas, specified via the keyword `by` of the Mizar language:

   ```
   A1: x in X;
   A2: for y being set holds y in X\/Y iff
                          (y in X or y in Y);
   x in X\/Y by A1, A2;
   ```

3. The Mizar Mathematical Library (MML) builds on the components (1) and (2) above to provide a mass of Mizar language formulas certified, by Mizar verifier, to be derivable from a handful

[1] Consistently with the fact that a vector is a function with a special domain (compare this with the beginning of section 3), we indicate the i-th component of vector $\boldsymbol{a}$ as $\boldsymbol{a}(i)$, reserving the use of sub- and superscripts for other cases.

of set-theoretical axioms affine to ZFC (Zermelo-Fraenkel with the axiom of choice) . The set theory resulting from these axioms, Tarski-Grothendieck (TG), is an extension of ZFC, and more on it can be found in [5].

MML is made up of Mizar source files called *articles*, and its latest version is always browsable at http://mizar.uwb.edu.pl/version/current/mml/. In the following, we will be using typewriter font for referencing articles and results inside MML: for example, XBOOLE_1:4 denotes the fourth theorem appearing in the MML article xboole_1.miz, which is thus viewable at http://mizar.uwb.edu.pl/version/current/mml/xboole_1.miz. We will also adopt typewriter font for Mizar code, as already done in point (1) of the numbered list above.

3 Formalization choices

In this section we pass from common mathematical language of section 1.1 to the corresponding Mizar definitions and statements.

We have to encode the objects appearing in (1): $\mathcal{X}_{\boldsymbol{b}}^{II}$ and $\mathcal{P}_{\boldsymbol{b}}^{II}$. It turns out that a first convenient step is to relax the requirement on $\boldsymbol{b}$: we just ask that it is a relation (not even a function) rather than a natural-indexed vector. A second convenient choice is to restrict the bids to be natural (as opposed to rational or real) numbers. This is not a hindrance, since in real world currency is indeed a positive integer. These choices allow to give the wanted definitions in simple terms of very general, low-level (from a set-theoretical point of view) objects; given a relation R representing our 'vector' of bids:

1. `union rng R` is the highest bid. `rng R` is the range of `R`, and `union rng R` is the union of all the elements of the set `rng R`. This works thanks to our requirement that the bids are natural, because of the so-called Von Neumann encoding of ordinals, which means that 0 is represented as the empty set, 1 is the set $\{0\}$, 2 is the set $\{0,1\}$, and $n+1$ is the set $\{0,1,\ldots,n\}$.
2. Then `R"(union rng R)`, aptly named `topbiddersof R`, is the set of the participants having placed the highest bid. Indeed, `R"Y` is the Mizar rendition of the preimage of the set `Y` under the relation `R`. So that
3. `winnerof R`, defined as

   ```
   the Element of (R"(union rng R))
   ```

   is the winner of the auction. Note that, in Mizar, the construct `the Element of` refers to a fixed element of a set without actually specifying it, hence implicitly employing the axiom of choice. In the present case, this gives a very quick way to randomly extract a winner in case of more than one top-bidder.
4. At this point, `losersof R` is trivially defined as `dom R \ {winnerof R}`. Here, `dom R` is the domain of the relation `R` and `X\Y` is the set-theoretical difference of `X` and `Y`, also called the relative complement of `Y` in `X`.
5. Note that `R|(losersof R)` is still a bid 'vector': this is true exactly because we gave up the requirement of it being a *proper* vector (i.e., having a domain like $\{1,2,\ldots,n\}$). Hence, we can repeat the calculation as from point 1., and conclude that `union (rng (R|(losersof R))` is the highest non-winning bid, that is, the price to be paid by the winner, by definition of second-price auction. It is aptly named `priceof R`. Here, `R|X` is the restriction of the relation `R` to the set `X`.

Now the needed objects $\mathcal{P}_{\boldsymbol{b}}^{II}$ and $\mathcal{X}_{\boldsymbol{b}}^{II}$ are easy to express in Mizar (where, of course, the argument $\boldsymbol{b}$ becomes a generic relation `R`), respectively as

```
[:dom R,{0}:]+*[:{winnerof R},{priceof R}:]
```

and as

```
[:dom R,{0}:]+*[:{winnerof R},{1}:],
```

whose names are `R-pay` and `R-allocations`.

Only two further set-theoretical definitions are involved here: first, the cartesian product $X \times Y$, which in Mizar is written `[:X,Y:]`. Recalling that, in set theory, relations and functions *are* their own graph, this means that `[:dom R,{0}:]` is the function constantly evaluating to 0 on the whole domain of `R`, and `[:{winnerof R},{y}:]` is the function associating the single element of its domain, `winnerof R`, to the value `y`. Finally, `f +* g` 'pastes' the functions `f` and `g`, evaluating to `g` on the intersection of their domains.[2] So that `R-pay` is a function evaluating to `0` for all the participants, except the winner, for which it evaluates to `priceof R`. Similarly, `R-allocations` is a function evaluating to `0` for all the participants, except the winner, for which it evaluates to `1`. This last occurrence of `1` reveals the simplified case we limit ourselves to: that of an indivisible auctioned good (we had not mentioned this limitation yet).

Luckily, the object `+*` just introduced naturally permits to define also the last operation we need to pass from the definitions to the theorem statement: the single-component alteration introduced in point 3. on page 1. Indeed, $\boldsymbol{f}_x^y$ is easily given by `f +* [:{x},{y}:]`, so that (1) has been formalized and proved in Mizar as:

```
(v*(f-allocations.i)) - ((f-pay).i) <=
v*((f +* [:{i},{v}:])-allocations).i -
(f +* [:{i},{v}:])-pay.i,
```

where the reader only needs to know that `g.x` is the function `g` evaluated in the point `x`: it is the Mizar way of denoting $g(x)$. Of course, having been so slack with the requirements on the bids `R` implies that some additional hypotheses are needed. The first one is already present in the code snippet above, hinted by the change of `R` into `f`: the bids is no longer only a relation, but a function. The remaining ones make the theorem actually read as:

```
for f being Function st
(rng f c= NAT\{0} & rng f is finite
& dom f is non trivial) holds
(v*(f-allocations.i)) - ((f-pay).i) <=
v*((f +* [:{i},{v}:])-allocations).i
- (f +* [:{i},{v}:])-pay.i
```

A byproduct of this approach is that it makes the points of the proof in which each additional hypothesis is needed self-evident: for example, the requirement of the bids being a relation is to be strengthened into the one that they are a function exactly in lemma `Lm7` (referring to the full Mizar source or to its essential version on page 6). We conclude with few last clarifications: `NAT` is $\mathbb{N}$, `c=` is the set-theoretical inclusion (so that the `rng f c= NAT\{0}` means that `f` is $\mathbb{Z}^+$-valued), and a set is trivial iff it has cardinality smaller than 2.

Table 1 sums up some Mizar notation for reader's convenience.

[2] Note that this still results in a function.

Table 1. Summary of Mizar notations.

Mizar notation	description	blackboard rendition
`X c= Y`	set inclusion	$X \subseteq Y$
`X c< Y`	strict set inclusion	$X \subset Y$
`{}`	the empty set	$\emptyset$
`X\/Y, X/\Y`	pairwise union, intersection	$X \cup Y$, $X \cap Y$
`X\Y`	difference of sets	$X \backslash Y$
`union X`	arbitrary union	$\bigcup_{x \in X} x$
`[:X,Y:]`	cartesian product	$X \times Y$
`NAT`	natural numbers	$\mathbb{N}$
`R"X`	preimage of set X through relation R	$R^{-1}[X]$
`R|X`	restriction of R to X	$R\|_X$
`dom R, rng R`	domain, range of R	
`f.x`	f evaluated in x	$f(x)$
`f +* g`	pasting of functions	$f\|_{\mathrm{dom}\, f \setminus \mathrm{dom}\, g} \cup g$
`+, -, *, /`	arithmetic operations	$+, -, \cdot, /$
`b-allocations`	allocations vector	$\mathcal{X}_{\boldsymbol{b}}^{II}$
`b-pay`	payments vector	$\mathcal{P}_{\boldsymbol{b}}^{II}$

4 Principles behind the formalization choices

Some points of this formalization expose a clash between two needs: that of writing Mizar code in a novice-friendly, closer-to-natural-language style, versus that of reducing the bloat and the coding time; the latter goal implies exploiting the features of both the Mizar system and its foundational axioms to create lean and efficient code. While Mizar language is reputedly among the ones which most resemble common mathematical language [6], and thus one of the most accessible to a novice to the field of mechanized proving, a proficient Mizar user will easily have priorities leading him to sacrifice this aspect.

We point out three of them, as seen useful to discuss the present matters:

1. Evaluating which are the most convenient mathematical objects to reduce the examined objects to.
2. Escaping the typing system when it is a hindrance.
3. Reformulating statements in a style more liable to automations, though possibly farther from natural language.

Let us deepen each point in a dedicated section.

4.1 Which objects to base on?

One of the referees stated a poignant observation which can be a very good starting point for this discussion:

> Aspects of the foundations of Mizar (which is a version of untyped set-theory with 'soft' type system on top) are leaking into the application: e.g., properties of natural numbers that happen to be encoded in the manner of von Neumann are used to express the idea of the maximum of a set. Do mathematicians do this in real life?

In a sense, yes: if they accept ZF as a foundation (as most do), they do such things all the time, albeit generally without thinking (or even knowing) about the underlying set-theoretical machinery. Indeed, the point in introducing all the 'higher level' encodings (as Von Neumann ordinals) is to be able to hide the complexities of ZF modeling, while keeping the comforts given by a solid, time-tested foundation as ZF, and this is usually very convenient in traditional mathematics. When doing *mechanized* mathematics, things change a bit, though: reasoning in terms of the underlying sets of course does not change the essence of the mathematical objects, and permits to get rid of typing when it is a hindrance (see section 4.2). Moreover, doing a formalization in terms of the encoded mathematical objects rather than in terms of the encoding sets usually prevents from taking advantage of the range of canned proofs in the library, which is unavoidably broader in the latter case. Even when a result is not available in the existing library, proving it for sets rather than for the particular objects one is dealing with is of course overall more desirable.

Thus how one encodes the new objects he need is no longer a mere matter of style; it is crucial to how effective and maintainable a formalization will be. The time initially spent to figure out how to translate the involved mathematical object into those already well supported (which typically means low-level with respect to the particular foundations) is a labor usually well repaid, both in terms of how easy the work will result and of the work's impact with regard to the global usefulness of the system's library. In the present case, as already illustrated, we used few general objects, like `union`, `\`, `"`, `|`, `/\`, `[:,:]`, `rng`, `+*`; this reflects in a somewhat long introductory section of `vickrey.miz`, containing general lemmas regarding those objects which are of wider interest. As a bottom line, considerations of a software engineering flavor as the ones above suggest that, in mechanized proving, there are occasions in which, opposedly to standard mathematics, it is convenient forgetting 'higher level' encodings and working on the underlying entities (sets, in the case of set theory).

As an example, let us dwell back on the definition of `priceof R` (see point 5 of page 2):

```
func priceof R equals
union rng (R | losersof R);
```

`union` is a universal set-theoretical operations, but in our cases it does exactly what we need, i.e. taking the maximum, thus obtaining the wanted second price. We focus on this last step, where two crucial choices emerge: first, rather than using the operator `max`, which Mizar provides, we use `union`. It should be noted that in our case the two operate in exactly the same way, with the big difference that `max` is only defined on *numerical* sets: this would imply constraining `R|losersof R` to be numerical-valued, which limits generality and invariably leads to additional work in the proofs involving this object. This work would be spent on something that is mathematically irrelevant, since `max` and `union` do exactly the same: we are discarding the typing because it is limiting us, in this case. Additionally, we would be obliged to import all the definitions regarding `max` and its operands, while `union` is such a basic operation that it needs much less dependencies.

Implicit in such a definition is the second choice: that of limiting the bids to natural numbers. Indeed, the identification of `max` and `union` only holds for *natural* numbers. This is not accidental: natural numbers are simple objects in most formal frameworks, and hence admit simple encodings in term of low-level objects. Thus, this is a symptom of a more general issue: integer, rational and real numbers present escalating complexity in definitions and proofs. Correspondingly, assessing which of these numerical families one will base future code should occur early in the endeavour. In the present case, the first consideration was that currency has always an atomic quantum even on financial exchanges where 'sub-pennying' is allowed. The second consideration was that even if, for some currently unforeseen reason, the case of fractional or even continuous 'currency'

were needed, definitions and theorems could be generalized by embedding them into the correspondent enlargements.

4.2 How much typing?

Contrary to most other systems, typing has no foundational role in Mizar (and in set theory). Assigning a type to each term one talks about was an early response to the foundational crisis of naive set theory. However, although type systems subsequently found extremely widespread application in programming language design to catch errors in software, in its original goal they were largely displaced by ZF set theory, which fixed naive set theory in other ways.

But while programming languages are used to produce executables, formalization languages are used to verify correctness mathematics: correspondingly, the role of a type system changes. In the case of Mizar, this role is twofold: to make the text read more natural (e.g., to be able to write `n is natural number` instead of the equivalent, plainly set-theoretical statement `n in NAT`) and to embed several automation mechanisms in it. When these two aspects are not top-priority, typing should be reduced to a minimum. For a concrete example, let us go back once again to the definition of `priceof`, and assume we ignored the considerations above, conceding a more immediate and typed definition:

```
let R be REAL-valued Relation;
func priceof R equals max rng (R|loserof R);
```

This version would probably be slightly more intelligible than ours for a newcomer, because it saves him to be warned about the proviso that taking the union over finitely many natural numbers equates to taking the maximum. On the other hand, to make such a definition be accepted by Mizar, one first of all has to look for and import all the results making all the types right: one has to know that

- `R|losersof R` is still `REAL-valued`
- the range of a `REAL-valued` is a subset of `REAL`, and thus
- `max` can be applied

This burden is faced every time one invokes `priceof R` in subsequent proofs, while it is utterly bypassed by breaking the abstraction layer and looking down at the sets embodying it.

4.3 Automatically accepted statements

Due to how automations work in Mizar, often there are mathematically equivalent statements presenting different amounts of justification needed to have them accepted by the checker. Usually, the most natural and readable rendition (i.e., the one closest to natural language and easiest to digest for a layman) is not the one minimizing the justifying effort.

We review here few concrete cases, while a general treatment of these kinds of custom exploitation of Mizar automation mechanisms is exposed in [1]. The simplest and most common case is that of statements about the equality of two sets. The equality symbol, =, can be rendered through the set theoretical operation `\+\` and the attribute `empty`, via the result `FOMODEL0:29`:

```
for X, Y being set holds
X \+\ Y is empty iff X=Y;
```

This means that for every theorem whose statement has the form

```
B1: term1 = term2;                (2)
```

one can produce a translation like

```
term1 \+\ term2 is empty;         (3)
```

This latter version presents the advantage of being usable without justification in subsequent proofs, while the former ones require the user to explicitly refer to the label `B1`. This implies consuming a substantial amount of time to locate this label inside the huge MML. For this reason, in the present formalization, the second form prevails over the first, which is nonetheless much more immediate to read. For example, one finds

```
dom f \/ dom g \+\ dom (f +* g) ={};
```

in lieu of

```
dom(f +* g)=dom f \/ dom g by FUNCT_4:def 1;
```

along with more intricate schemes (all documented in [1]) which permit to save lookups to MML at the expense of clarity (because the formula reads less natural *and* because an explicit justification is missing). On the other hand, these developer's shortcut schemes tend to be applied to such trivial passages that the reader, once he is aware of the base idea, should have no problem with them.

5 Formal proof

The Mizar source is organized into three thematic sections[3]: the first one, as mentioned in section 4.1, contains those results which could be induced down to low-level objects, thanks to the approach discussed in section 4.1. For example, `union {x} = x;`, or

```
y<>0 implies
(x=z iff ([:X,{0}:]+*[:{z},{y}:]).x<>0);
```

which says that the function constantly zero except in a point z yields a non-zero value exactly when evaluated in z.

Given their triviality, this whole first section can be omitted when illustrating the proof. The second sections defines a second price auctions and the related concepts, also stating few simple characterizing properties. Third section contains the significant results. Stripping off the proofs, second and third section fit on one single page, attached here (page 6) for the reader to have an overview of the proof design. This make sense because each single Mizar lemma results quite elementary, having proof never exceeding thirty lines.

As already remarked, in most theorems the hypotheses on the bids 'vector', denoted with `R` or `f`, are restrictions which restore some very natural properties we gave up in name of generality and simplicity of definitions. They are usually implicitly assumed in standard treatments, and require, for example, that the bids are numerical, that they are finite and finitely many, or that the participants are more than one.

The key result is `Lm21`. The subsequent theorem, `Lm20`, is introduced only to translate the hypotheses of `Lm21` on the bids in a more direct and usable form, thus producing `Lm22`.

[3] The Mizar keyword `begin` permits starting a new section for the writer's convenience; it is ignored by the verifier.

6 Informal proof

The proof mechanism is slightly different from the ones found in [3, 4]. Here, the cases are parsed not on, e.g., whether the given participant i wins or loses in the original and modified auction. Rather, we focus on the sign of left-hand side (LHS) of inequality (1): if it is non positive, then (1) is obviously true because of lemma 1. Thus, we reduced to the case LHS being positive, which means that i is the winner of the auction $\boldsymbol{b}$ and that $v > \mathcal{P}^{II}_{\boldsymbol{b}}(i) > 0$. Then,

$$\mathcal{P}^{II}_{\boldsymbol{b}^v_i}(i) = \mathcal{P}^{II}_{\boldsymbol{b}}(i) > 0 \tag{4}$$

by lemma 2. This also implies the equality $\mathcal{X}^{II}_{\boldsymbol{b}^v_i}(i) = \mathcal{X}^{II}_{\boldsymbol{b}}(i) = 1$, which, reusing (4) with it, yields[4] LHS=RHS in (1).

This concludes the proof, leaving only the following couple of lemmas to be justified; in what follows, the winner and the number representing the price (given in Mizar by `winnerof` and `priceof`, respectively) of a second price auction starting with input $\boldsymbol{b}$ will be denoted with $\underline{\boldsymbol{b}}$ and $\overline{\boldsymbol{b}}$.

Lemma 1 (`Lm4`)**.** *RHS of* (1) *is not negative.*

Proof. Consider the function

$$\tau := (v - \overline{\boldsymbol{b}^v_i}) \cdot \mathcal{X}^{II}_{\boldsymbol{b}^v_i}.$$

The only possible point at which such a function can yield a negative value is $\underline{\boldsymbol{b}^v_i}$, and this happens if and only if $v < \overline{\boldsymbol{b}^v_i}$. Since RHS is $\tau(i)$, if thesis were false we should then assume $i = \underline{\boldsymbol{b}^v_i}$ and $v < \overline{\boldsymbol{b}^v_i}$, which leads to contradiction: $\overline{\boldsymbol{b}^v_i} \leq \boldsymbol{b}^v_i\left(\underline{\boldsymbol{b}^v_i}\right) = v$. □

Lemma 2 (`Lm5`)**.** *Changing the bid of the winner of a second price auction into a value strictly higher than the auction price does not change the payments vector:*

$$\epsilon > 0 \Longrightarrow \mathcal{P}^{II}_{\boldsymbol{b}^{\overline{\boldsymbol{b}}+\epsilon}_{\underline{\boldsymbol{b}}}} = \mathcal{P}^{II}_{\boldsymbol{b}}$$

Proof. As long as the winner bids strictly more than the second price, he will remain the winner, so that the allocation vector will not vary; moreover, changing the winner's bid will not change the second price, by definition. Hence the thesis, because the payments vector is the allocations vector scalar-multiplied by the price. □

The reader can compare these informal lemmas with their Mizar counterparts, whose labels are reported bracketed.

REFERENCES

[1] M.B. Caminati and G. Rosolini, 'Custom automations in mizar', *Journal of Automated Reasoning*, (2012).

[2] A. Grabowski, A. Korniłowicz, and A. Naumowicz, 'Mizar in a Nutshell', *Journal of Formalized Reasoning*, **3**(2), 153–245, (2010).

[3] M. Kerber, C. Lange, and C. Rowat, 'Vickrey auctions', *http://www.cs.bham.ac.uk/research/projects/formare/code/auction-theory/*, (2013).

[4] Eric Maskin, 'The unity of auction theory: Milgrom's masterclass', *Journal of Economic Literature*, **42**(4), 1102–1115, (2004).

[5] P. Rudnicki and A. Trybulec, 'On equivalents of well-foundedness', *Journal of Automated Reasoning*, **23**(3), 197–234, (1999).

[6] J. Urban, 'MizarMode–an integrated proof assistance tool for the Mizar way of formalizing mathematics', *Journal of Applied Logic*, **4**(4), 414–427, (2006).

[4] Actually, we just proved something stronger than inequality (1); i.e., that LHS $> 0 \Longrightarrow$ LHS $=$ RHS and LHS $< 0 \Longrightarrow$ LHS $<$ RHS

```
begin
:::definitions related to second price auctions, and basic facts

reserve x,y,z,X,Y,i for set, P,Q,R for Relation, f,g for Function, v for Nat;

definition let R;
func topbiddersof R -> Subset of dom R
equals R"{union rng R};
func winnerof R equals
the Element of topbiddersof R;
func losersof R -> Subset of dom R
equals dom R \ {winnerof R};
func priceof R equals
union rng (R | losersof R);
func priceof R -> Element of NAT equals
the Element of {priceof R}/\NAT;
func R-pay -> Function equals
[:dom R,{0}:]+*[:{winnerof R},{priceof R}:];
func R-allocations -> Function equals
[:dom R,{0}:]+*[:{winnerof R},{1}:];
end;

Lm16: priceof R=0 or priceof R=union rng (R|losersof R)

Lm8: (rng R<>{} & rng R c= NAT & rng R is finite)
implies winnerof R in topbiddersof R

Lm7: winnerof f in topbiddersof f
implies f.(winnerof f)=union rng f &
priceof f c= f.(winnerof f)

begin :::the core theorems

Lm3: R-pay.(winnerof R)=priceof R

Lm0: union rng (P|(dom P \ dom Q)) c< union rng Q
implies topbiddersof (P +*1 Q) = Q"{union rng Q}

Lm1: union rng (P | (dom P\{x})) c< X implies
topbiddersof (P +*1 [:{x},{X}:])={x}

Lm5: (priceof R<>0 & i=winnerof R & R-pay.i c< y)
implies (R +*1 [:{i},{y}:])-pay.i = R-pay.i

Lm4: f is NAT-valued & rng f is finite implies
(f+*[:{i},{v}:])-pay.i <= v*((f+*[:{i},{v}:])-allocations.i)

Lm21: :::Vickrey's theorem, version 1
f is NAT-valued & rng f is finite & priceof f <> 0 implies
(v*(f-allocations.i)) - ((f-pay).i) <=
v*((f +* [:{i},{v}:])-allocations).i - (f +* [:{i},{v}:])-pay.i

Lm20: (rng R c= NAT & rng R is finite & dom R is non trivial)
implies priceof R in rng R

Lm22: rng f c= NAT\{0} & rng f is finite & dom f is non trivial
implies :::Vickrey's theorem, version 2
(v*(f-allocations.i)) - ((f-pay).i) <=
v*((f +* [:{i},{v}:])-allocations).i - (f +* [:{i},{v}:])-pay.i
```

Mathematical specification of an agend-based model of exchange

Nicola Botta[1] and **Antoine Mandel**[2] and **Mareen Hofmann**[3] and **Sibylle Schupp**[4] and **Cezar Ionescu**[5]

1 Introduction

Agent-based computer models are becoming increasingly popular in economics, both as a toolbox for theoreticians [12] and for policy advise [7].

All science nowadays relies on computer models. For an interesting analysis of just how much, see "Simulation and its Discontents" [13]. But a distinguishing feature is that, in computational economics, computer models are often the only kind of models we have.

The situation is completely different from physics and engineering where computer models are secondary to mathematical ones. It is the latter which embody the theories and serve as object of discussion between scientists: computer code rarely makes its way in scientific publications.

In physics and engineering a computer model is judged to be correct if it faithfully implements its mathematical counterpart. Prominent computer model failures – from 1996 Ariane 5 flight 501 [1] to 2010 "flash crash" at the New York Stock Exchange – remind us that, sometimes, it is difficult to formulate exactly what counts as "faithful". But, at least in principle, mathematical models provide a "golden standard" for the correctness of computer models in physics and engineering.

But what is "golden standard" against which to judge the correctness of agent-based models for computational economics ? When can we trust the outcome of multi-agent simulations designed to study viable decarbonization options or to inform politicians on the effect of green building subsidies on unemployment ?

Such models are usually accompanied by a narrative, an informal description of the ideas behind their development. But a narrative is too blunt an instrument to help us decide whether a computer implementation of it is correct or not, whether the results of a simulation are trustworthy or flawed by programming errors.

In a nutshell, the current state of affairs in computational economics can be summed up as follows: we can limit ourselves to informal narratives, or we can use simulations of computer models which we do not understand [10] and whose correctness we cannot guarantee.

[1] Potsdam Institute for Climate Impact Research
[2] Centre d'économie de la Sorbonne, Université Paris 1
[3] Freie Universität Berlin
[4] Technische Univertität Hamburg-Harburg
[5] Potsdam Institute for Climate Impact Research

In agent-based simulations of financial markets, the deployment of proprietary toolkits and platforms makes this situation even worse.

1.1 Gintis' model

In 2006, Herbert Gintis [5] announced the discovery of a mechanism that would explain price formation and disequlibrium adjustment without requiring the presence of a central authority as is currently assumed in mainstream economics.

Gintis' results were, as he put it "empirical rather than theoretical: we have created a class of economies and investigated their properties for a range of parameters." They were obtained by computer simulations.

The results were relevant – understanding price formation is a long-standing puzzle in economic theory – and intriguing: which class of economies is actually specified by the model presented in [5] ? Can one express the properties investigated in [5] formally ?

In 2009 two groups of researchers, one at PIK, the other at Chalmers [4], independently attempted to do something which should perhaps be routine, but is hardly ever done: to reimplement the model and reproduce the results reported in [5].

After initial attempts failed and Gintis graciously provided the source code, both groups discovered several ways that his implementation diverged from the description in the paper, one of which could be called a "bug". Much more problematic, however, was the ambiguity left open by the narrative given in [5]. This is quite typical in computational economy and econophysics: scientists tend to believe that the mathematical equations and the narrative used to describe a model are sufficient specifications for the implementation of that model but this is rarely the case.

1.2 Our contribution

In [3] we proposed a functional framework for the specifications of computer models of exchange. The framework provides, among others, the notions of bilateral exchanges, bilateral trades, trading policies and games and dynamical models of exchange.

There, we addressed the question of how to describe and specify computer-based dynamical models of exchange in a language which is more accessible to non-programmers than

program listings and yet less ambiguous than narrative descriptions.

In this contribution, we apply that framework and derive a complete mathematical specification of Gintis' model [5][6].

We first derive a bona-fide specification of the model on the basis of [5] and of the model implementation kindly made available by the author. From this specification, we derive a first model reimplementation. This fails to reproduce the results reported in [5]. This is not really surprising: in the original model, exchanges between agents are rationed. In contrast, the results presented in [8] suggest that convergence of agent-specific prices towards the "special" equilibrium prices discovered in [5] depend on a "no strategic rationing condition".

Then, we show how the model specification can be relaxed for the corresponding implementation to reproduce the original results. The relaxed specification is consistent with the results reported in [8]. It suggests that trade resolving policies that yield convergence of agent-specific prices (towards equilibrium prices at large times) cannot grant convergence of allocations (towards equilibrium allocations, at fixed equilibrium prices).

2 Notation

In this section we introduce the basic notation used throughout this paper. The presentation is intentionally terse, details and motivations are discussed in [3].

We use A and G to denote finite sets of agents and goods.

Stocks are formulated as functions of type $G \to \mathbb{R}_{\geq 0}$. Prices are formulated as functions of type $G \to \mathbb{R}_{>0}$ and utilities are formulated as functions of type $(G \to \mathbb{R}_{\geq 0}) \to \mathbb{R}$. For convenience, we introduce the abbreviations $Q = G \to \mathbb{R}_{\geq 0}$, $P = G \to \mathbb{R}_{>0}$ and $U = Q \to \mathbb{R}$. Allocations and utility profiles are formulated as functions of type $A \to Q$ and $A \to U$, respectively.

We write $q : Q$ to posit that q is a stock that is, a function of type $G \to \mathbb{R}_{\geq 0}$. We denote function application by juxtaposition: if $x : A \to Q$, $x\ a : Q$ is the stock of $a : A$ according to x and $x\ a\ g : \mathbb{R}_{\geq 0}$ the quantity of $g : G$ according to $x\ a$.

This notation is standard in mathematics and computing science but not quite common in engineering and computational economics[7]. For a discussion of the advantages and disadvantaged of the two notations we refer the reader to [3].

In the next section we apply the framework presented in [3] to derive a complete mathematical specification of the model presented in [5]. Terms that denote notions specified in [3] are introduced in italics and the relevant sections of [3] are given in footnotes, e.g., *bilateral exchange*[8].

3 A mathematical specification of [5]

In a nutshell, the model presented in [5] is a *dynamical model of exchange*[9] for so-called private prices. These are agent-specific prices. At each iteration step, the agent-specific prices are updated according to an evolutionary algorithm. This is driven by a trading fitness. The trading fitness is measured in a *trading game*[9] which, in turn, is controlled by the actual prices. Strictly speaking, the trading game is not a model of exchange: in [5], agents are not only trading but also producing and consuming.

We formulate time-dependent agent-specific prices as a function $y : \mathbb{N} \to A \to P$. In the model, $y\ 0$ is given and $y\ 1, y\ 2, \ldots$ are computed by iterating an evolutionary algorithm [11]. This is obtained by folding a copy-mutate rule *cm* on a copy-mutate schedule:

$$\begin{aligned} y\ 0 &= y_0 \\ y\ (t+1) &= \text{fold}\ \ cm\ (y\ t)\ (cms\ t) \end{aligned} \tag{1}$$

In the above specification, $cms\ t$ is a random list of pairs

$$cms\ t\ :\ List\ ((A \times A) \times (G \to [0,1]))\ .$$

According to the signature of fold[10], *cm* maps agent-specific prices and pairs of type $(A \times A) \times (G \to [0,1])$ into agent-specific prices:

$$cm\ :\ (A \to P) \to (A \times A) \times (G \to [0,1]) \to (A \to P)$$

In the rest of this section, we refine (1) by putting forward specifications for *cm* and *cms*. We start by formulating the model setup. Then, we specify *cm* in terms of a trading fitness function $f : A \to \mathbb{R}$ and explain how random copy-mutate schedules are drawn. Finally, we specify the trading game which defines f. This completes the specification of the model presented in [5].

3.1 Model setup

The copy-mutate rule *cm* and the random schedules $cms\ t, t \in \mathbb{N}$ depend on a number of functions and parameters. These are:

- a *sector*[11] function s. Its specification (equation 20 of [3]) reads

$$\begin{aligned} &s : A \to G \\ &\forall\ g \in G\ \ s^{-1}\ g \neq \emptyset\ . \end{aligned}$$

- An initial allocation $x_0 : A \to Q$.
- A utility profile $u : A \to U$.
- A sector-to-sector *number of peers*[11] np. Its specification (equation (23) of [3]) reads

$$\begin{aligned} &np : G \to G \to \mathbb{N} \\ &\forall\ g, g' \in G\ \ np\ g\ g' \leq |s^{-1}\ g'|\ . \end{aligned}$$

[6] By complete specifications we mean specifications that describe the model entirely and from which model implementations can be derived without ambiguity.

[7] In economics, stocks, for instance, are often formulated in terms of vectors in $\mathbb{R}^n_{\geq 0}$ where $n = |G|$ and one writes $q \in \mathbb{R}^n_{\geq 0}$ and q_j instead of $q : G \to \mathbb{R}_{\geq 0}$ and $q\ g$

[8] Section 3.3 of [3].

[9] Section 3.6 of [3].

[10] fold is a polymorphic function. Its type depends on two parameters: $\text{fold} : (X \to Y \to X) \to X \to List\ Y \to X$. Fold reduces lists of arbitrary type Y to a single value of (another arbitrary) type X: it takes a binary function, an initial value (the accumulator), and a list. It first calls this binary function with the initial value and the rightmost element of the list, to obtain a new accumulator, and then repeats for the remaining elements; in the end, it returns one single value, e.g., $\text{fold}\ +\ 0\ [3,2,1] = (((0+1)+2)+3) = 6$. For a definition of fold see [3] or standard computing science textbooks.

[11] Section 3.5 of [3].

- A copy-mutate fraction $cmf \in [0,1] \subset \mathbb{R}$.
- A mutation probability $mp \in [0,1] \subset \mathbb{R}$.
- A mutation factor $mf \in (0,1) \subset \mathbb{R}$.
- A number of *trading rounds*[12] $n_r \in \mathbb{N}$.

The model is based on a number of specific assumptions. These are:

(a) The number of goods is greater than one: $|G| > 1$.

(b) Sectors are not empty and equally populated. The number of agents per sector is denoted by $n_a \in \mathbb{N}$:

$$\forall\ g \in G\ \ |s^{-1}\ g| = |A|/|G| = n_a > 0$$

(c) The sector-to-sector number of peers is constant. The number of peers is denoted by $n_p \in \mathbb{N}$:

$$\forall\ g, g' \in G\ \ np\ g\ g' = n_p$$

(d) The initial allocation x_0 is sector-wise constant. Moreover, in any sector, x_0 is different from zero only in the sector-specific good:

$$\forall\ a, a' \in A\ \ s\ a = s\ a'\ \Rightarrow\ x_0\ a = x_0\ a'$$
$$x_0\ a\ g \neq 0\ \Rightarrow\ g = s\ a\ .$$

(e) The utility profile is constant. The utility function of each agent is the Scarf utility function[13] with $\lambda = 1/|A|$:

$$\forall\ a \in A\ \ u\ a\ y = \min_{g \in G}\ (y\ g)\ /\ (w\ g) \tag{2}$$

$$\text{where}: w = \frac{1}{|A|} * \sum_{a \in A} x_0\ a \tag{3}$$

3.2 The copy-mutate rule

In the prices iteration (1), the copy-mutate rule *cm* instantiates a replicator dynamic [11]:

$$z' = cm\ z\ ((a_1, a_2), \xi)$$
$$\Rightarrow$$
$$z'\ a \neq z\ a\ \Rightarrow\ a = a_1 \vee a = a_2$$
$$\wedge$$
$$f\ a_1 < f\ a_2$$
$$\Rightarrow$$
$$z'\ a_2 = z\ a_2\ \wedge$$
$$z'\ a_1\ g = \begin{cases} z\ a_2\ g & \text{if } \xi\ g < mp \\ (z\ a_2\ g)/mf & \text{if } \xi\ g \geq mp \wedge \xi\ g < 0.5 \\ (z\ a_2\ g) * mf & \text{if } \xi\ g \geq mp \wedge \xi\ g \geq 0.5 \end{cases}$$
$$\wedge$$
$$f\ a_1 \geq f\ a_2$$
$$\Rightarrow$$
$$z'\ a_1 = z\ a_1\ \wedge$$
$$z'\ a_2\ g = \begin{cases} z\ a_1\ g & \text{if } \xi\ g < mp \\ (z\ a_1\ g)/mf & \text{if } \xi\ g \geq mp \wedge \xi\ g < 0.5 \\ (z\ a_1\ g) * mf & \text{if } \xi\ g \geq mp \wedge \xi\ g \geq 0.5 \end{cases}$$

[12] Section 3.5 of [3].
[13] Section 3.1 of [3].

As mentioned above, *cm* depends on a trading fitness f. Thus, the specification of *cm* is, strictly speaking, incorrect. While *mf*, *mp* are constant values provided by the model setup given in section 3.1, f is an undefined function of the system's *state*[9]. We specify the computation of f in section 3.4.

3.3 The copy-mutate schedule

Each copy-mutate schedule *cms* t is defined in terms of two random functions. The first one is an inverse numbering[14] of G

$$\gamma\ :\ [\,0, |G|\,) \to G\ .$$

There are $|G|!$ such numberings. The second random function provides, for each sector $g \in G$, $cmf * n_a$ random pairs[15]. The first element of each pair is itself a pair of agents in $s^{-1}\ g$. The second element are $|G|$ random numbers in $[0,1] \subset \mathbb{R}$

$$\beta\ :\ G \to [\,0, cmf * n_a\,) \to (A \times A) \times (G \to [0,1])\ .$$

Thus, $\beta\ g\ k$ is a partial function and fulfills the specification $\forall\ g \in G, \forall\ k \in [\,0, cmf * n_a\,)$[16]

$$\text{ran}\,(\beta\ g\ k)\ =\ \big((s^{-1}\ g) \times (s^{-1}\ g)\big) \times (G \to [0,1])\ .$$

Each draw is assumed to be equally probable. Given a draw (γ, β), the schedule is defined by the list comprehension[17]

$$[\ \beta\ (\gamma\ j)\ k\ |\ j \leftarrow [\,0, |G|\,),\ k \leftarrow [\,0, cmf * n_a\,)\ ]$$

Thus, the schedule is the list of pairs $\beta\ (\gamma\ j)\ k$ obtained by drawing j and k from $[\,0, |G|\,)$ and $[\,0, cmf * n_a\,)$, respectively. It consists of $|G| * (cmf * n_a)$ elements.

3.4 The trading fitness

The trading fitness $f : A \to \mathbb{R}$ is computed in a trading game of the kind formulated in section 3.6 of [3]:

$$f = h\ (\text{map}\ (\text{fold}\ (cp.tr)\ (f_0, x_0)\ )\ tss)\ . \tag{4}$$

In the above expression, $\text{map} : (X \to Y) \to List\ X \to List\ Y$ is a function that transforms lists: it takes a unary function and applies it element-wise to the list. For instance, map negate $[1,2,3] = [-1,-2,-3]$. The function $cp.tr$ represents the composition of *cp* and *tr*: $(cp.tr)\ a\ b = cp\ (tr\ a\ b)$. The target of h is equal to $A \to \mathbb{R}$. The game consists of n_r rounds. The elements of *tss* are random trading schedules,

[14] a numbering of a finite set X is a bijective function of type $X \to [\,0, |X|\,)$. It associates to each element of X exactly one number between 0 and $|X| - 1$. There are $|X|!$ such functions.
[15] For sake of simplicity, we just write $cmf * n_a$ to denote the natural number obtained by rounding the product between $cmf \in \mathbb{R}$ and $n_a \in \mathbb{N}$.
[16] we could make $\beta\ g\ k$ total by introducing dependent types. With dependent types, one could express the specification for $\beta\ g\ k$ through the type system. Applying dependent types for specifications is a natural approach but goes beyond the scope of this paper.
[17] Informally, comprehension on lists is analog to set comprehension. For instance $\{2 * i\ |\ i \in \{0, \ldots, n\}\}$ translates to $[2 * i\ |\ i \leftarrow [0..n]]$ on lists. This constructs the list of integers $[0, 2, \ldots, 2 * n]$ from $[0..n]$. The latter denotes the list of integers $0, 1, \ldots$ until n. The expression $a \leftarrow as$ is read "for a drawn from as". It represents the action of iterating over the elements of as in the order defined by as.

one schedule per round. The outcome of a round is a fitness-allocation pair:

$$h : List\ ((A \to \mathbb{R}) \times (A \to Q)) \to (A \to \mathbb{R})\ . \tag{5}$$

In every round, the initial fitness f_0 is zero for all agents:

$$\forall\ a \in A\ \ f_0\ a = 0\ .$$

The initial allocation x_0 (and the number of rounds n_r) are given by the model setup, see section 3.1. We specify the computation of f by giving h, cp, tr and by describing how the random schedules of tss are drawn.

3.4.1 The random schedules

Each schedule in tss is defined in terms of three random functions. The first one is an inverse numbering of G like the one introduced above for the copy-mutate schedules. For the sake of simplicity we call this function γ, too. There are $n_\gamma = |G|!$ such numberings. The second function provides $|G|$ inverse numbering, one for each sector

$$\alpha\ :\ G \to [\,0, n_a\,) \to A\ .$$

The idea is that $\alpha\ g$ is an inverse numbering of $s^{-1}\ g$. Therefore $\alpha\ g$ fulfills

$$\forall\ g \in G\ \ \mathrm{ran}\,(\alpha\ g)\ =\ s^{-1}\ g \tag{6}$$

There are $n_\alpha = (n_a!)^{|G|}$ such numberings. The third function gives, for each pair of sectors g, $g' \neq g$ and for each agent in $s^{-1}\ g$, a random draw of n_p agents in $s^{-1}\ g'$

$$\eta\ :\ G \to G \to A \to \mathbb{N} \to A\ .$$

Thus, $\eta\ g\ g'$ is a partial function and fulfills the specification[18]

$$\forall\ g, g' \in G\ \ g' \neq g\ \Rightarrow\ \mathrm{dom}\,(\eta\ g\ g')\ =\ s^{-1}\ g\ , \tag{7}$$

$$\forall\ g, g' \in G\ \ g' \neq g\ \Rightarrow\ \mathrm{ran}\,(\eta\ g\ g')\ =\ [\,0, n_a\,) \to s^{-1}\ g'\ . \tag{8}$$

There are $n_\eta = (n_a!/(n_a - n_p)!)^{n_\alpha * (|G|-1)^{|G|}}$ such random draws. Thus, at each round, there are $n_\gamma^{n_\alpha * n_\eta}$ ways to define a random trading schedule, each one represented by a draw (γ, α, η) satisfying (6)-(8). Each draw is assumed to be equally probable. Given a draw (γ, α, η), the schedule is defined by the list comprehension

$$\begin{aligned}[\big(\alpha\ (\gamma\ j)\ i, \eta\ (\gamma\ j)\ (\gamma\ j')\ (\alpha\ (\gamma\ j)\ i)\ k\big)\ \mid\ & j \leftarrow [\,0, |G|\,),\\ & j' \leftarrow [\,0, |G|\,),\\ & j' \neq j,\\ & i \leftarrow [\,0, n_a\,),\\ & k \leftarrow [\,0, n_p\,)\ ]\end{aligned}$$

[18] As for the random function β introduced for the specification of the copy-mutate schedules, we could make $\eta\ g\ g'$ total by introducing dependent types.

3.4.2 The trade function tr and the function cp

As mentioned above, the trading game (4) is not a model of exchange in the sense made clear in [3]: after a trade, the two agents may undergo a consumption-production step. This is represented by the function cp. This function is composed with an *extended elementary bilateral trade* tr[9]. We start with the specification of tr:

$$\begin{aligned}tr : (A \to \mathbb{R}) \times (A \to Q) \to A \times A \to\\ ((A \to \mathbb{R}) \times (A \to Q)) \times (A \times A)\end{aligned} \tag{9}$$

$$tr\ (f, x)\ (a_1, a_2) = ((f', x'), (a_1', a_2'))$$
$$\Rightarrow$$
$$f' = f\ \wedge\ a_1' = a_1\ \wedge\ a_2' = a_2 \tag{10}$$
$$x(\mathcal{X}_e\ a_1\ a_2\ g_2\ g_1)x' \tag{11}$$
$$(x'\ a_1 - x\ a_1)\ g_2 = \delta_1 \tag{12}$$
$$(x'\ a_2 - x\ a_2)\ g_1 = \delta_2 \tag{13}$$
$$(\delta_1, \delta_2) = trp\ p_1\ p_2\ (o_1, g_1)\ (d_1, g_2)\ (o_2, g_2)\ (d_2, g_1) \tag{14}$$
$$(o_1, d_1) = odp\ a_1\ p_1\ g_2 \tag{15}$$
$$(o_2, d_2) = odp\ a_2\ p_2\ g_1 \tag{16}$$
$$g_1 = s\ a_1 \tag{17}$$
$$g_2 = s\ a_2 \tag{18}$$
$$p_1 = y\ t\ a_1 \tag{19}$$
$$p_2 = y\ t\ a_2 \tag{20}$$

The types of the two arguments taken by tr, a fitness-allocation pair and a pair of agents, is dictated by equations (4) and (5). The result of applying tr to (f, x), (a_1, a_2) is a pair $((f, x'), (a_1, a_2))$, see (10). The new allocation, x', is related to x through an *elementary bilateral exchange* of goods $s\ a_2$ and $s\ a_1$. The specification of elementary bilateral exchanges is given in section 3.3 of [3]. One could replace equation (11) with specifications (11), (12) of [3][19] with $g_1 = s\ a_2$ and $g_2 = s\ a_1$.

Equations (12)-(20) are refinements of (11). They completely define the outcome of the trade x' in terms of the *offer and demand policies*[20] of a_1, a_2 – the function odp – and of an agent-independent trade-resolving policy trp. These functions depend on the actual allocation x, on the model setup functions u, x_0, s and on the actual prices $y\ t$ of the iteration in which the trading game is played. As we will see, trp and, therefore, the elementary bilateral trade tr are not symmetric. In general,

$$tr\ (f, x)\ (a_1, a_2) \neq tr\ (f, x)\ (a_2, a_1)\ .$$

In [5], the first agents in the pairs of a trading schedule are called demanders. The second agents are called responders. In the following, we will use the same terminology.

The offer and demand policy is *demand-based* and *value-preserving*[13]. With u, x_0 as in the model setup of section 3.1 and according to equation (8) of [3], the demand of a under

[19] Or, equivalently, with (9), (10) and (12) of [3] with $g_1 = s\ a_2$ and $g_2 = s\ a_1$.

[20] Section 3.4 of [3].

the agent-specific prices of the t-th iteration $y\ t$ is

$$\frac{(x_0\ a)\cdot(y\ t\ a)}{(\sum\limits_{a\in A} x_0\ a)\cdot(y\ t\ a)} * (\sum_{a\in A} x_0\ a)\ .$$

Thus, for fixed x_0, u (model setup) and time dependent prices $y\ t$ (we are considering a trading game at fixed prices), the excess demand profile for $x : A \to Q$ is

$$\begin{aligned} &ed\ x\ : A \to Q \\ &ed\ x\ a = \frac{(x_0\ a)\cdot(y\ t\ a)}{(\sum\limits_{a\in A} x_0\ a)\cdot(y\ t\ a)} * (\sum_{a\in A} x_0\ a)\ -\ (x\ a) \end{aligned}$$

The offer and demand policy is defined in terms of such excess demand

$$\begin{aligned} &odp : A \to P \to G \to \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0} \\ &g \neq (s\ a)\ \wedge\ odp\ a\ p\ g = (o, d)\ \Rightarrow \\ &d\ =\ \begin{cases} 0 & \text{if } ed\ x\ a\ g \leq 0 \\ ed\ x\ a\ g & \text{if } ed\ x\ a\ g > 0 \\ & \quad \wedge \\ & \quad (ed\ x\ a\ g) * \dfrac{p\ g}{p\ (s\ a)} \leq x\ a\ (s\ a) \\ (x\ a\ (s\ a)) * \frac{p\ (s\ a)}{p\ g} & \text{if } ed\ x\ a\ g > 0 \\ & \quad \wedge \\ & \quad (ed\ x\ a\ g) * \dfrac{p\ g}{p\ (s\ a)} > x\ a\ (s\ a) \end{cases} \\ &o\ =\ d * \frac{p\ g}{p\ (s\ a)} \end{aligned}$$

and by requiring the value of the offer to equal the value of the demand. Notice that $odp\ a\ g$ is only defined for $g \neq (s\ a)$[21] and fulfills specification (13) of [3].

The description of the trading mechanism given in [5] does not yield an unambiguous specification of *trp*. However, the author has provided a complete implementation of the model presented in [5] in Delphi. This is an extension of Object Pascal which provides primitives for implementing applications based on graphical user interfaces on Microsoft Windows operating systems. The implementation is consistent with the following specification:

$$\begin{aligned} &trp : P \to P \to \mathbb{R}_{\geq 0} \times G \to \mathbb{R}_{\geq 0} \times G \to \mathbb{R}_{\geq 0} \times G \to \\ &\qquad \mathbb{R}_{\geq 0} \times G \to \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0} \\ &g_1 \neq g_2\ \wedge\ trp\ p_1\ p_2\ (o_1, g_1)\ (d_1, g_2)\ (o_2, g_2)\ (d_2, g_1) = (\delta_1, \delta_2) \\ &\quad \Rightarrow \\ &(\delta_1, \delta_2)\ =\ \begin{cases} (0,0) & \text{if } o_1 * (p_2\ g_1) < d_1 * (p_2\ g_2) \\ (\delta_1', \delta_2') & \text{otherwise} \end{cases} \qquad (21) \\ &(\delta_1', \delta_2')\ =\ \begin{cases} (d_2 * \frac{p_1\ g_1}{p_1\ g_2}, d_2) & \text{if } d_2 < \delta_2'' \\ (\delta_1'', \delta_2'') & \text{otherwise} \end{cases} \qquad (22) \\ &(\delta_1'', \delta_2'')\ =\ \begin{cases} (o_2, o_2 * \frac{p_1\ g_2}{p_1\ g_1}) & \text{if } o_2 < \delta_1''' \\ (\delta_1''', \delta_2''') & \text{otherwise} \end{cases} \qquad (23) \\ &(\delta_1''', \delta_2''')\ =\ \begin{cases} (x_2\ g_2, x_2\ g_2 * \frac{p_1\ g_2}{p_1\ g_1}) & \text{if } x_2\ g_2 < d_1 \\ (d_1, o_1) & \text{otherwise} \end{cases} \qquad (24) \end{aligned}$$

[21] as discussed for the random functions which define the trading schedules, we could make *odp* total by introducing dependent types.

In section 4 we provide some experimental support for this claim. As anticipated, *trp* is not symmetric: if the value of the demander's offer meets or exceeds the value of its demand (according to the responder's prices), the outcome of the trade is liable to be the demander's offer and demand, see equation (21). For this to be the case, three further conditions have to be met.

The first one, equation (24), is a budget condition. It requires the amount of g_2 to be exchanged not to exceed the stock of the offerer. If this happens, the demander's demand is capped to the responder's stock. This ensures that, after exchange, the responder's stock of g_2 does not turn negative.

The second and the third are rationing constraints. They require the amount of g_2 exchanged not to exceed the offer (of g_2) of the responder, equation (23) and the amount of g_1 to be exchanged not to exceed the demand of the responder, equation equation (22). We will discuss the effects of these conditions on the dynamics of prices in section 4.

Notice that, with *trp* defined as above, the outcome of a trade does not, in general, fulfill specification (14) of [3]. On the other hand, *odp*, *trp* ensure that *tr* is value-preserving: if $p_1 = p_2$, the exchange of δ_1 and δ_2 does not modify the value of the stocks of both agents.

Let's now turn our attention to the consume-produce function *cp*. The type of *cp* can be inferred from the type of *tr* (9) and from equation (4):

$$cp : ((A \to \mathbb{R}) \times (A \to Q)) \times (A \times A) \to (A \to \mathbb{R}) \times (A \to Q)$$

The consume-produce function *cp* takes the outcome of *tr* — a fitness-allocation pair and a pair of agents — and returns a new fitness allocation pair. The rationale of *cp* is twofold.

On one hand, it accounts for successful trades — trades which have modified the stocks of the interacting agents — in the new trading fitness. Remember that all agents compute their offer and demand according to the same policy *odp*. In other words, two agents with equal initial and actual stocks and with the same (agent-specific) prices issue equal offers and demands. Thus, since initial allocations are sector-wise constant, the trading fitness is, within a given sector, a measure of the fitness of the agent-specific prices. This explains why prices, in [5] are sometimes referred to as *strategies*.

On the other hand, *cp* rewards successful agents by providing them with even more goods to trade (production). From the listing made available by Gintis, we deduce the specification:

$$\begin{aligned} &cp\ ((f, x), (a_1, a_2)) = (f', x') \\ &\quad \Rightarrow \\ &f'\ a \neq f\ a \Rightarrow a = a_1 \vee a = a_2 \\ &x'\ a \neq x\ a \Rightarrow a = a_1 \vee a = a_2 \\ &a \in \{a_1, a_2\} \Rightarrow f'\ a = f\ a + u\ a\ (x\ a) \qquad (25) \\ &a \in \{a_1, a_2\} \Rightarrow \\ &\quad x'\ a = \begin{cases} x_0\ a & \text{if } u\ a\ (x\ a) \geq 1 \\ (x\ a) * (1 - u\ a\ (x\ a)) & \text{otherwise} \end{cases} \qquad (26) \end{aligned}$$

This specification is puzzling in a number of ways. The new stocks — or, from a modeling perspective, production processes — depend on the utility in a very discontinuous fashion. Agents which have achieved levels of utility near but below one are forced to reduce their stocks — to consume —

proportionally to the achieved utilities. No production takes place. With little stocks left to trade, such agents are not likely to improve their utilities in later interactions. Their stocks will be further reduced albeit at increasingly slower rates. On the other hand, the stocks of agents which have managed to achieve utilities equal or greater than one are set back to their initial stocks. This resetting can be interpreted as a consumption-production step. After such step, agents enter the next trade with zero utility and full budget. It is not obvious (to us) what is the motivation behind the utility threshold and why such threshold is set to one. Notice, however, that, with x_0, u and λ as defined in the model setup 3.1, one has

$$(x,p)\mathcal{E}(x_0,u) \;\Rightarrow\; u\ a\ (x\ a) = \frac{(x'_0\ (s\ a)) * (p\ (s\ a))}{\frac{1}{|G|} * \sum_{g' \in G} (x'_0\ g') * (p\ g')} \quad (27)$$

where $x'_0 : G \to \mathbb{R}_{\geq 0}$ is the function that defines a sector-wise constant initial allocation x_0 which complies with assumption 3 from section 3.1:

$$x_0\ a\ g = \begin{cases} x'_0\ g & \text{if } g = s\ a \\ 0 & \text{otherwise .} \end{cases} \quad (28)$$

The numerical results presented in [5] suggest that, at large times, the prices

$$\begin{aligned} &\hat{p} : P \\ &\hat{p}\ g = \frac{1}{w\ g} = \frac{|A|}{\sum_{a \in A} (x_0\ a\ g)} = \frac{|G|}{x'_0\ g} \end{aligned} \quad (29)$$

are, under (1), more stable than any other prices. Specifically, the author has observed that, independently of the initial prices y_0, the price iteration tends to "converge", at long times, towards the "special" prices (29)[22]. For $p = \hat{p}$ equation (27) becomes

$$(x,p)\mathcal{E}(x_0,u) \;\Rightarrow\; u\ a\ (x\ a) = 1$$

Thus, in [5], the utility threshold in the consume-produce function cp has been set to match the (agent-independent) utility of equilibrium stocks under "special" prices.

3.4.3 The function h

The function h computes, for each agent, its average trading fitness

$$h\ [(f_1,x_1),\ldots,(f_{n_r},x_{n_r})]\ a = \frac{1}{n_r} * \sum_{k=1}^{k=n_r} f_k\ a\ . \quad (30)$$

Remember that, at the t-th prices iteration, the fitness achieved by an agent in a trading round is a random event: the set of interactions which take place in a round is defined by the random trading schedule of that round. At each round, the trading schedule is drawn from the set of all feasible trading schedules with equal probability. Moreover, each round is played starting from the same initial allocation x_0, zero fitness and constant agent-specific prices $y\ t$.

Therefore, $h\ a$ can be interpreted as an approximation of the expected trading round fitness obtained with the prices $y\ t\ a$.

[22] In [8], we study a simple class of models and suggest an explanation for this behavior.

4 Model implementation, numerical experiments

We have implemented the prices iteration (1) on the basis of the specification presented in the previous sections.

The implementation has been done in C++. It relies on standard libraries and on the collection of generic software components SC [2]. C++ is a statically typed programming language. Thus, when developing programs from specifications, one has to choose which specifications to enforce at compile time and which at run time.

In practice, the capabilities of expressing specifications at compile time are limited by the lack of explicit language support for higher order functions, constrained genericity and non-integer value genericity. Concept-based language extensions [6] are a promising approach but need to be practiced and substantiated with a better understanding of dependent types.

We have taken a pragmatic approach and enforced most specifications at run time. This has been done mainly through pre- and post-conditions clauses following a design-by-contract (DBC) [9] approach.

Of course, such approach can not guarantee that the implementation is correct that is, that it fulfills the specification. We turn back to this problem in section 4.2. In the rest of this section we discuss a number of results obtained with this implementation. All results have been obtained for 6 goods $G = \{g_1,\ldots,g_6\}$ and 600 agents $A = \{a_1,\ldots,a_{600}\}$. Thus, the number of sectors is equal to 6 and the number of agents per sector n_a is 100. The other parameters are defined according to section 3.1 and to the following setup:

- the sector function s is

$$s\ a_i \;=\; g_j,\ j = 1 + (i-1)/100, \quad i = 1,\ldots,600\ .$$

- The sector-to-sector number of peers, np, is 5.
- The initial allocation x_0 is defined by equation (28) with

$$x'_0\ g_j = \frac{|G|}{|A|} * (1+j), \quad j = 1,\ldots,6\ .$$

The correspondent special prices (29) normalized by g_1 are

$$\frac{\hat{p}\ g_j}{\hat{p}\ g_1} = \frac{2}{j+1}\ . \quad (31)$$

- The copy-mutate fraction cmf is 0.05.
- The mutation probability mp is 0.1.
- The mutation factor mf is 0.95.
- The number of trading rounds per prices iteration n_r is 10.

For any agent $a \in A$, the initial price $y_0\ a\ g_1$ is set to one. All other prices are drawn randomly from $(0,1]$ with uniform probability distribution.

The above setup is, to the best of our knowledge, the one used to compute the results shown in figures 1 and 2 of [5][23]. These results suggest that, under (1)

- the standard deviations of $y\ t$ rapidly decline in the first 1500 trading rounds that is, at 10 rounds per iteration, for t between 0 and 150.

[23] In [5] the mutation probability is reported to be 0.01 that is, one tenth of mp. However, the value used in the original Delphi implementation is equal to mp.

- at large times — for numbers of trading rounds (iterations) of the order of 10^5 (10^4) — the agent-specific prices tend to converge[24] towards the special prices (31).

In the next sections we compare these results with those obtained with our implementation. Two caveats are at place:

First, as it turns out, the dynamics of prices at large times depends critically on the conditions upon which the trading fitness f is incremented. In Gintis' original program, this is done only upon "successful" trades. The source code provided by Gintis shows that trades are taken to be be "successful" even when the amounts of goods actually exchanged by the two interacting agents are zero. This seems to be inconsistent. In our implementation, the trading fitness f is incremented if (and only if) an elementary bilateral trade between two agents yields non zero exchanges. More precisely, *cs* is executed only if *trp* returns non zero exchanges or, equivalently, if $x' \neq x$ in (11):

$$\delta_1 > 0 \ \vee \ \delta_2 > 0 \tag{32}$$

Second, in the original implementation exchanges are always rationed according to specifications (22) and (23). Rationing or, in other words, demand-limited trade resolving policies are certainly necessary[25] for sequences of elementary bilateral exchanges at fixed equilibrium prices to yield equilibrium allocations, see [3]. On the other hand, the analysis presented in [8] suggests that, at large times, rationing might prevent the convergence of agent-specific prices towards stochastically stable equilibrium prices. As anticipated, our implementation allows for rationing (and, in fact, production and consumption) to be disabled.

4.1 Short times price dynamics

In [5], the dynamics of prices at short times is described in terms of three standard deviations: a "cross-period standard deviation of mean private prices" and "inter-agent standard deviations of private producer and consumer prices".

Unfortunately, [5] does not define operational rules for computing these quantities and the vertical scale in figure 1 of [5] suggests that these quantities are not standard deviations or that the probability distribution for the initial prices is not uniform. Here, we describe the dynamics of prices at short times in terms of two mean squared deviation functions. The first one simply computes, for any price, the mean squared deviation over all agents for that (normalized) price:

$$msd : (A \to P) \to G \to \mathbb{R}$$

$$msd\ y\ g = \frac{\sum_{a \in A} \left(\frac{y\ a\ g}{y\ a\ g_1} - \mu\ y\ g \right)^2}{|A|}, \quad \mu\ y\ g = \frac{\sum_{a \in A} \frac{y\ a\ g}{y\ a\ g_1}}{|A|}$$

The second function computes, for any sector, the sector mean of the mean squared deviation of the prices of that sector:

$$msd' : (A \to P) \to G \to \mathbb{R}$$

$$msd'\ y\ g = \frac{\sum_{g' \in G} * \left(\frac{1}{|s^{-1}\ g|} * \sum_{a \in s^{-1}\ g} \left(\frac{y\ a\ g'}{y\ a\ g_1} - \mu'\ y\ g\ g' \right)^2 \right)}{|G| - 1}, \tag{33}$$

$$\mu'\ y\ g\ g' = \frac{1}{|s^{-1}\ g|} * \sum_{a \in s^{-1}\ g} \frac{y\ a\ g'}{y\ a\ g_1}\ .$$

Notice that, because of the normalization, the mean and the mean squared deviation of g_1 are 1 and 0 on any subset of $|A|$. This is why, in (33), the sum over $g' \in G$ is divided by $|G| - 1$ and not by $|G|$.

In figure 1 we report the graphs of $msd\ y$ (top) and $msd'\ y$ (bottom). Thus, the curves on the top are mean squared deviations (over all agents) of prices, one curve for each price. Since the prices are normalized, the mean squared deviation for g_1 is identically zero.

In contrast, the curves on the bottom are averages (over prices) of the mean squared deviations of the prices of a given sector, one curve for each sector. As one would expect, the sector-specific average mean squared deviations decrease faster than the mean squared deviations taken over all agents, especially at very short times. This is because the copy-mutate rule *cm* is applied sector-wise, see section 3.3.

Notice that the rates at which mean squared deviations decrease for increasing number of trading games are different for different prices (top) and sectors (bottom). We do not have an explanation for this behavior. Also, notice that the results of figure 1 seem qualitatively different from those reported in figure one of [5]. We come back to this point in the next section.

4.2 Large times price dynamics

In [5], the dynamics of prices at large times is described in terms of the relative deviations of average prices from the special prices (31). Such deviations are:

$$dev : (A \to P) \to G \to \mathbb{R}, \quad dev\ y\ g = -\frac{\mu\ y\ g - \frac{\hat{p}\ g}{\hat{p}\ g_1}}{\mu\ y\ g}$$

Again, $dev\ y\ g_1$ is identically zero. Figure 2 shows the mean squared deviations *msd* (top) and the deviations of the mean prices from the special prices *dev* (bottom) at large times. The results shown in figure 2 do not confirm those presented in [5]. While [5] does not report mean squared deviations at large times, the graphs of the deviations of the mean prices from the special prices on the right of figure 2 do not suggest that agent-specific prices tend to "converge" towards the special prices at large times.

Let us pause for a moment. We have derived a specification for the prices iteration (1) on the basis of [5] and of an original model implementation made available by the author. We have implemented (1) from such specification. Our numerical results fail to reproduce those presented in [5]. There are three logically possible reasons for such failure:

1. Our specification is a good description of the model presented in [5] but our implementation does not fulfill the specification, that is, it is not correct.

[24] of course, modulo stochastic mutations as effected through the copy-mutate rule *cm*.

[25] But, in general, not sufficient.

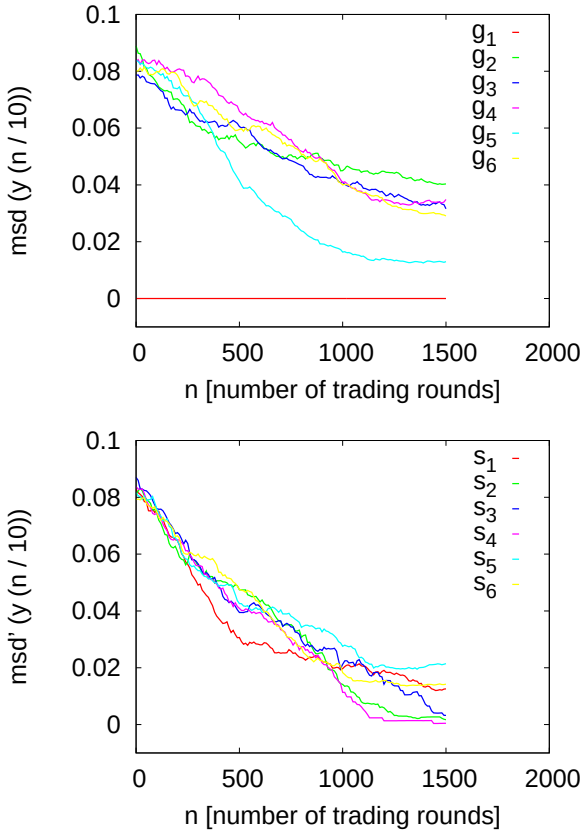


Figure 1. Standard setup: *msd* (top) and *msd′* (bottom).

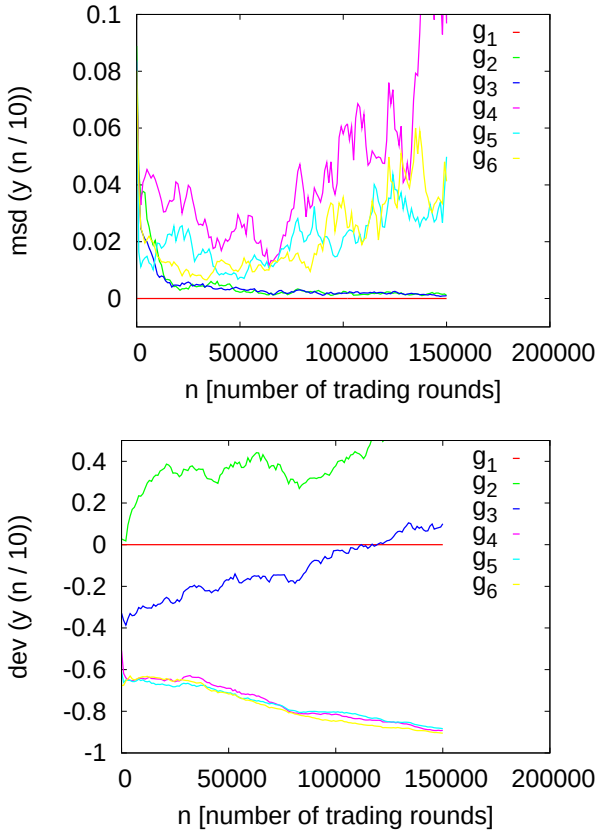


Figure 2. Standard setup: *msd* (top) and *dev* (bottom).

2. Or, our implementation is correct but our specification is not a good description of the model presented in [5].
3. Or, our model specification is a good description of the model presented in [5]. The results presented in [5], however, have been obtained with an implementation which does not fulfill such specification.

While 1 and 2 and 2 and 3 are mutually exclusive, it can of course be that both 1 and 3 are true: implementational errors never can be ruled out.

We can explain the inconsistencies between our numerical results and those presented in [5] by looking at the mechanisms that control the dynamics of prices at large times. We start by noticing that the results presented so far are robust with respect to perturbations of the copy-mutate rule *cm*[26] and focus the attention on the functions that define a trading game: *tr* and *cp*.

As anticipated above, the analysis presented in [8] suggests that, at large times, rationing might prevent the convergence of agent-specific prices towards stochastically stable equilibrium prices. Thus, an obvious experiment is to run a simulation in which rationing is disabled that is, (21)-(24) are replaced by

$$trp : P \to P \to \mathbb{R}_{\geq 0} \times G \to \mathbb{R}_{\geq 0} \times G \to \mathbb{R}_{\geq 0} \times G \to \mathbb{R}_{\geq 0} \times G \to \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0}$$

$$g_1 \neq g_2 \ \wedge \ trp\ p_1\ p_2\ (o_1, g_1)\ (d_1, g_2)\ (o_2, g_2)\ (d_2, g_1) = (\delta_1, \delta_2)$$

$$\Rightarrow$$

$$(\delta_1, \delta_2) = \begin{cases} (0,0) & \text{if } o_1 * (p_2\ g_1) < d_1 * (p_2\ g_2) \\ (\delta_1', \delta_2') & \text{otherwise} \end{cases} \tag{34}$$

$$(\delta_1', \delta_2') = \begin{cases} (x_2\ g_2, x_2\ g_2 * \frac{p_1\ g_2}{p_1\ g_1}) & \text{if } x_2\ g_2 < d_1 \\ (d_1, o_1) & \text{otherwise} \end{cases} \tag{35}$$

Except for this modification, the setup is the same as the of figure 2. Figure 3 shows the usual mean squared deviation (top) and relative deviations (bottom) graphs. As in [5] agent-specific prices appear to converge, at large times, towards the special prices. In fact, figure 3 bottom and figure 2 of [5] are quite similar. Before attempting at drawing any conclusions, let us consider the impact of two more controls on the dynamics of prices at large times.

First, remember that the consume-produce function *cp* is responsible for updating the trading fitness of interacting agents and for modeling consumption and production. As dis-

[26] of course, the speed at which the mean squared deviations of the prices decrease at short times and the amplitude of the high frequency oscillations in the graphs of *dev* depend on the values of cmf, mp and mf. In a number of numerical experiments (not reported here) designed to study how the dynamics of prices at large times depends on the values of cmf, mp and mf, however, we never observed small perturbations of these parameters (or of the function *cm*) to yield dramatic changes in the dynamics of prices. These results are consistent with those reported in [5].

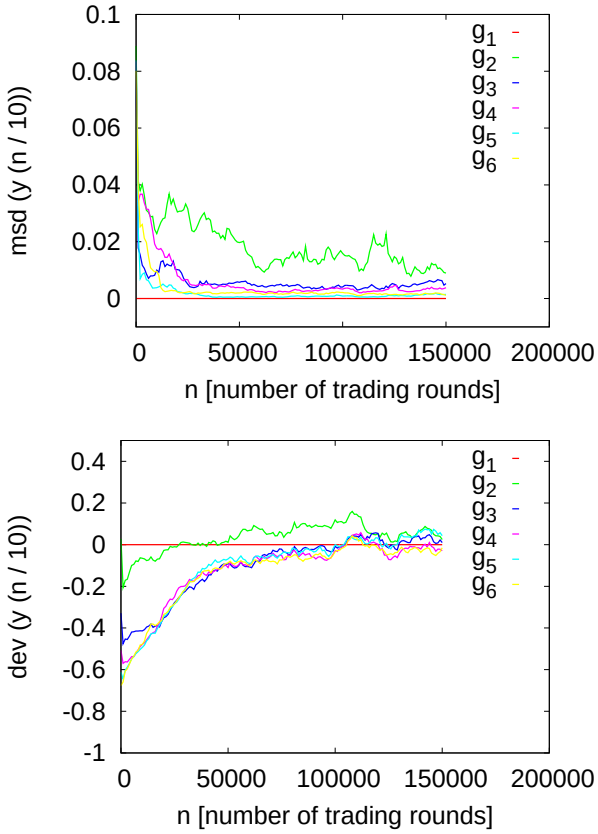


Figure 3. Standard setup but without rationing: *msd* (top) and *dev* (bottom).

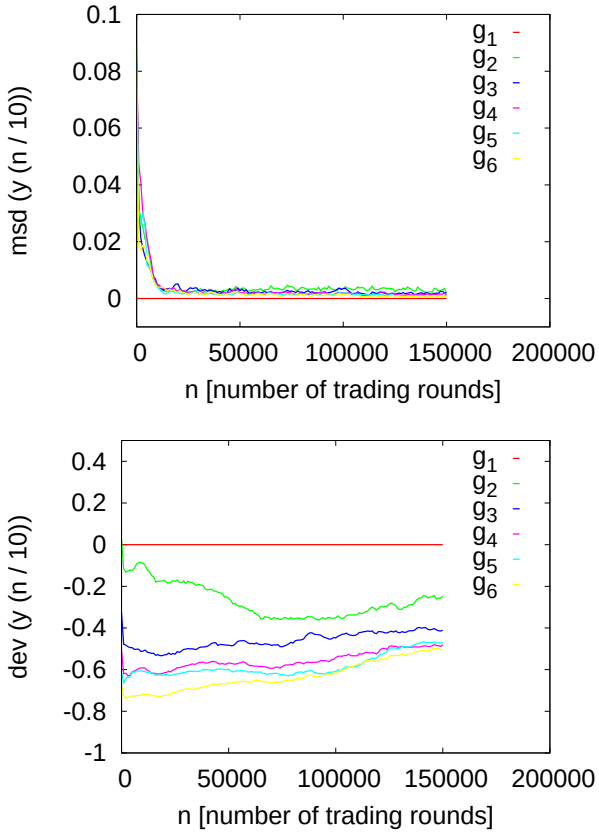


Figure 4. Standard setup but without consumption/production: *msd* (top) and *dev* (bottom).

cussed in section 3.4.2, consumption and production depend on a utility threshold which we find difficult to motivate. In a first experiment, we switch off consumption and production that is, we replace (26) with

$$a \in \{a_1, a_2\} \Rightarrow x'\ a = x\ a$$

Figure 4 reports the results obtained for the same setup of figure 2 but without consumption and production. The effect of switching off consumption and production is clearly visible in the graphs of the mean squared deviation of prices. As the number of iterations increases, the mean squared deviation of prices decrease remarkably. The prices, of course, tend to stabilize (again, modulo stochastic mutations) but not around the special prices (31): the deviations of the mean prices from the special prices (figure 2, bottom) do not tend towards zero as the number of iterations increases. On the other hand, disabling both consumption and production and rationing brings back the large times behavior of figure two with, not surprisingly, significantly lower mean squared deviations, see figure 5. Second, the conditions upon which the fitness of the interacting agents is incremented — the notion of "successful" trade discussed above — has of course an impact on the result of a trading game, the agents trading fitness f. A natural question is therefore whether such conditions critically affect the dynamics of prices at large times. Figures 6, 7 shows the results obtained with the same setups of figures 2, 3 but with modified conditions for incrementing the trading fitness. Specifically, in figures 6, 7 the trading fitness of the two agents entering an elementary bilateral trade is incremented if

$$o_2 > 0 \wedge d_2 > 0 \wedge x_2\ g_2 > 0 \wedge o_1 * (p_2\ g_1) \geq d_1 * (p_2\ g_2) \quad (36)$$

This condition can be easily derived from the original implementation. It is necessary for a non-trivial exchange to take place but, of course, not sufficient. Thus, the modified condition weakens our specification. A comparison between figure 2 and 6 suggests that, when bilateral trades are rationed, the conditions upon which the fitness of the interacting agents is incremented affect the dynamics of prices at large times significantly. In contrast, when bilateral trades are not rationed, the same conditions appear to impact the dynamics of prices less severely, see figures 3 and 7.

We have discussed the outcome of numerical experiments on the dynamics of prices at large times in terms of the graphs of *msd* and *dev* during the first 15000 iterations or 150000 trading games. Each computation, however, has been carried out for 1500000 iterations (15000000 trading games). The results (not shown here) confirm the observations done for the first 15000 iterations.

5 Conclusions, future work

We have applied the functional framework proposed in [3] to derive a "bona fide" mathematical specification of the model discussed in [5].

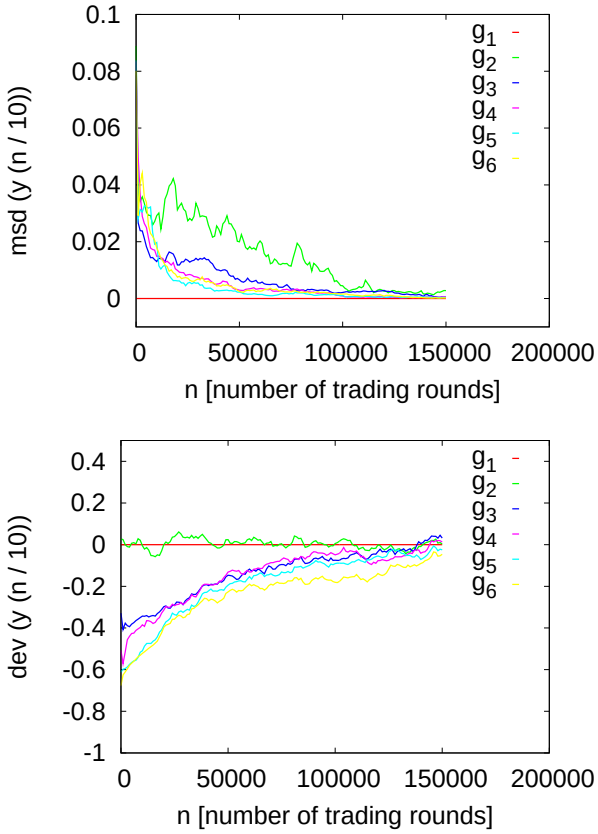


Figure 5. Standard setup but without rationing and consumption/production: *msd* (top) and *dev* (bottom).

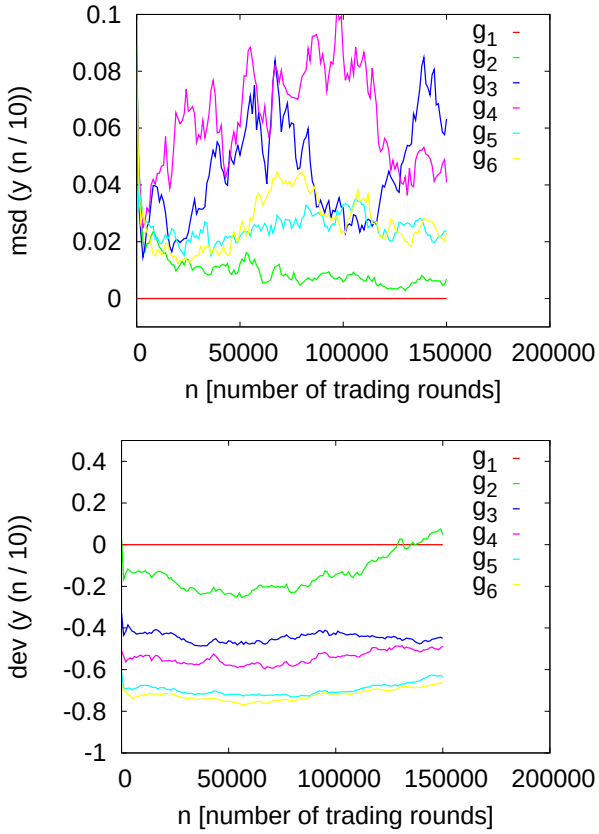


Figure 6. Standard setup but the trading fitness of the interacting agents is incremented if condition (36) (in contrast to (32)) is fulfilled: *msd* (top) and *msd′* (bottom).

The specification is based on the model description presented in [5], on the original model implementation in Delphi kindly made available by the author and on the Java / MASON model re-implementation presented in [4].

The specification is complete in the sense that it allows unambiguous model implementations. We have written one such implementations in C++.

We have used this implementation to run a number of numerical experiments on the dynamics of prices. These experiments partially confirms the results presented in [5]. In particular, we were able to independently reproduce the price dynamics reported in figure 2 of [5], see section 4, figure 3.

On the other hand, the numerical results reported in figure 2 and figure 3 suggest that the emergence of the special, quasi-public equilibrium prices (31) from agent-specific prices (the "private" prices of [5]) under evolutionary (imitation-mutation) dynamics is by no means a robust feature of the model presented in [5]. Price convergence critically depends on properties of the underlying trading game, in particular, of the trade resolving policy.

When bilateral exchanges between agents are rationed, agent-specific prices do not seem to converge towards the quasi-public equilibrium prices (31). This behavior is consistent with the analysis proposed in [8].

A comparison of figures 3, 4 and 5 also suggests that production and consumption induce higher price volatility but do not essentially affect the convergence of prices.

Moreover, figures 2, 6 and 3, 7 suggest that the notion of successful trade or, in other words, the conditions under which the trading fitness of the interacting agents is incremented can have a significant impact on the dynamics of prices. This is particularly true for the case in which bilateral trades are rationed.

Of course, the above "conclusions" can only be preliminary. They need to be confirmed by independent numerical experiments and substantiated by analytical results. On the other hand, the model specification derived in section 3 and, in particular, the specification of the trading policies *odp* and *trp* allow one to draw logical consequences which are independent of numerical results.

One consequence is that rationing constraints are necessary (but certainly not sufficient) for sequences of elementary bilateral trades *tr* at fixed, constant equilibrium prices to drive $x0$ towards the equilibrium allocation correspondent to the given prices. We have discussed this issue in detail in section 4.2 of [3] but the reason why rationing is necessary is obvious: if the amounts of good exchanged in a trade exceed, e.g., the demand of the responder, this is going to be left with an amount of good in excess of its optimal value. Since elementary bilater trades only allow agents to decrease the amount of their sector specific "offer" good, there is no way for the responder to achieve its optimal stock.

In other words, under elementary bilateral trades, optimal

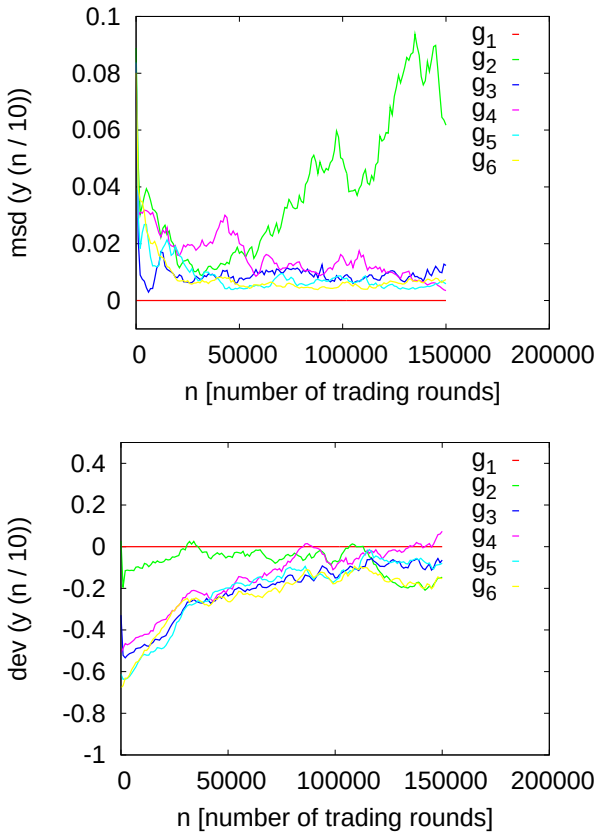


Figure 7. Standard setup but 1) without rationing and 2) the trading fitness of the interacting agents is incremented if condition (36) (in contrast to (32)) is fulfilled: *msd* (top) and *msd′* (bottom).

allocations can only be reached "from below" in the demand goods and "from above" in the offer good. Over-shootings (under-shootings) in the demand (offer) goods cannot be balanced by further interactions.

On the other hand, if rationing turns out to prevent convergence of agent-specific prices (as the numerical results shown in figure 2, 4 and 6 seem to suggest), we are forced to conclude that, in the model of exchange proposed in [5], convergence of allocations[27] and convergence of prices[28] are mutually exclusive.

This would mean that, even under the assumptions at the basis of the model presented in [5][29] we still do not know simple and plausible rules of bilateral interaction which guarantee emergence of equilibrium prices under imitation-mutation dynamics and, for fixed, exogenous equilibrium prices, convergence of allocations towards the correspondent equilibrium allocations.

A final word of precaution is needed. The numerical results presented in [5], [4] and in this paper have all been obtained with a constant utility profile and with the utility function (2), (3). This utility function supports infinitely many equilibrium prices-allocation pairs, see section 3.1 of [3].

To substantiate numerical conjectures on the dynamics of prices, numerical experiments of the kind reported in section 4 and in [5] should be run for less accommodating utility functions.

In particular, the dynamics of agent-specific prices at large times should be investigated for utility functions which support unique equilibrium prices and for the case in which the utility function is different in different sectors. Such investigations go beyond the scope of this paper.

[27] Towards equilibrium allocations, through sequences of elementary bilateral trades at fixed, given equilibrium prices.

[28] Towards quasi-public equilibrium prices, under imitation-mutation dynamics driven by the trading fitness achieved in sequences of elementary bilateral trades at fixed agent-specific prices.

[29] That is, offer and demand policies based on utility maximization, perfect knowledge of the stocks that maximize an agent's utility for every agent for arbitrary prices, fully deterministic elementary bilateral trades between agents.

ACKNOWLEDGEMENTS

The authors thank the reviewers, whose comments have lead to significant improvements of the original manuscript. The work presented in this paper heavily relies on free software, among others on hugs, vi, the GCC compiler, Emacs, LaTeX and on the FreeBSD and Debian / GNU Linux operating systems. It is our pleasure to thank all developers of these excellent products.

REFERENCES

[1] 'Ariane 5 Flight 501 Failure', Technical report, Report by the Inquiry Board, (1996). esamultimedia.esa.int/docs/esa-x-1819eng.pdf.

[2] N. Botta and C. Ionescu, 'Relation based computations in a monadic BSP model', *Parallel Computing*, **33**, 795–821, (2007).

[3] N. Botta, A. Mandel, C. Ionescu, M. Hofmann, D. Lincke, S. Schupp, and C. Jaeger, 'A functional framework for agent-based models of exchange', *Applied Mathematics and Computation*, **218**(8), 4025–4040, (December 2011).

[4] P. Evensen and M Märdin. An Extensible and Scalable Agent-Based Simulation of Barter Economics. Master Thesis. Chalmers Techn. Univ. & U. Gothenburg., 2009.

[5] H. Gintis, 'The emergence of a Price System from Decentralized Bilateral Exchange', *B. E. Journal of Theoretical Economics*, **6**, 1302–1322, (2006).

[6] Douglas Gregor, Jaakko Järvi, Mayuresh Kulkarni, Andrew Lumsdaine, David Musser, and Sibylle Schupp, 'Generic programming and high-performance libraries', *International Journal of Parallel Programming*, **33**(2), (June 2005).

[7] *Agent Based Models for Economic Policy Advice*, eds., B. LeBaron and Winker P., Lucius, 2008.

[8] A. Mandel and N. Botta, 'A Note on Herbert Gintis' "Emergence of a Price System from Decentralized Bilateral Exchange"', *The B.E. Journal of Theoretical Economics*, **9**, (2009).

[9] B. Meyer, *Object-oriented Software Construction*, Prentice-Hall Resource, Prentice-Hall, second edn., 2000.

[10] B. H. Mitra-Kahn. Debunking the Myths of Computable General Equilibrium Models. Schwartz Center for Economic Policy Analysis working Paper 2008-1, 2008.

[11] P. Taylor and L. Jonker, 'Evolutionarily Stable Strategies and Game Dynamics', *Mathematical Biosciences*, **40**, 145–156, (1978).

[12] *Handbook of Computational Economics II: Agent-Based Computational Economics*, eds., L. Tesfastion and K. Judd, North-Holland, 2006.

[13] Sherry Tuckle, *Simulation and Its Discontents*, MIT Press, 2009.

Embedding ACL2 Models in End-User Applications

Jared Davis[1]

Abstract. Formal verification, based on mechanical theorem proving, can provide unique evidence that systems are correct. Unfortunately this promise of correctness is, for most projects, not enough to justify its high cost. Since formal models and proof scripts offer few other direct benefits to system developers and managers, the idea of formal verification is abandoned.

We have developed a way to embed functions from the ACL2 theorem prover into software that is written in mainstream programming languages. This lets us reuse formal ACL2 models to develop applications with features like graphics, networking, databases, etc. For example, we have written a web-based tool for hardware designers in Ruby on top of a 100,000+ line ACL2 codebase.

This is neat: we can reuse the supporting work needed for formal verification to create tools that are useful beyond the formal verification team. The value added by these tools will help to justify the investment in formal verification, and the project as a whole will benefit from the precision of formal modeling and analysis.

1 INTRODUCTION

ACL2 [16] is an interactive theorem prover. It combines a Lisp-based programming language for developing formal models of systems with a reasoning engine that can prove properties about these models. It has been used to formally verify hardware at companies like AMD [25], IBM [26], and Rockwell Collins [30], and software like compilers [23], virtual machines [20], and operating system kernels [24]. ACL2's authors were awarded the 2005 ACM Systems Award for "pioneering and engineering a most effective theorem prover... as a formal methods tool for verifying safety-critical hardware and software." An ACL2 team shared the gold medal with a KIV team in the 2012 VSTTE Software Verification Competition.

Normally ACL2, or any other interactive theorem prover, is used to formally verify an *artifact*—a hardware design, a C program, an algorithm, a protocol, etc. This work is usually done by a team of experts and largely consists of three interrelated activities:

1. *Modeling.* A model of how the artifact behaves is developed in the theorem prover's logic. This is often a large undertaking. For instance, to model a program we may need to develop translation tools like preprocessors, parsers, etc., and may also need to formalize how the programming language behaves.
2. *Specification.* A specification for the artifact is written down as logical formulas. This is easy for some artifacts, e.g., it may only take a few lines to say what a particular operation of an arithmetic hardware unit is to compute. Other cases are much more difficult, e.g., what does it mean for an operating system to be secure?
3. *Proof.* The theorem prover is guided to show that the model meets its specification. Usually this is quite hard. Just how hard depends on the scale and nature of the model and specification, on the team's skill in developing effective proof automation, etc.

[1] Centaur Technology Inc. 7600-C N. Capital of Texas Hwy, Suite 300. Austin TX, 78731, email: jared@centtech.com

Why would anyone go to all this trouble? Formal verification usually reveals subtle bugs in the artifact (which, once exposed, can be fixed). It leads to mathematical proofs, checked by machine, that serve as evidence that the artifact has been designed correctly. The strength of this evidence depends on the precision of the model, the correctness of the specification, and the soundness of the prover. These concerns can often be addressed very convincingly.

1.1 Reusing Formal Models and Specifications

Unfortunately, due to the time and expertise it requires, formal verification is expensive. From a simple economic viewpoint, it only makes sense to formally verify artifacts whose failures could be very costly or tragic. This is still the case despite a lot of good work to reduce the costs of theorem proving by improving proof automation, interfaces, and pedagogy.

A different way to improve the cost/benefit situation is to increase the benefit. One way to do this, and the focus of this paper, is to reuse the modeling and specification efforts from formal verification in useful ways. Here are some examples.

Example 1: Processor Simulators. Normally, long before a processor design is to be manufactured, a program called a *golden model* is written to explain how the hardware is supposed to behave. As the hardware design evolves, it is continually simulated on test cases and compared against the golden model. This is often the primary way that bugs are found in the design. Since writing a golden model is much like the *Specification* activity of formal verification, an idea is to reuse the formal specification as the golden model. [12, 14]

This is not without challenges. A golden model needs to be something that the hardware design team can understand and practically use. A basic requirement is that the model should run at high speeds; fortunately ACL2 models and specifications are typically executable as programs, and ACL2 has many features [13] that allow for efficient execution. Other challenges include, e.g., how to connect the model to simulation tools for hardware design languages so that the design can be tested against the model.

When these challenges can be overcome, what does reuse accomplish? It avoids the need to separately develop the golden model and formal specification, directly reducing costs. It improves confidence in the formal verification effort, since the designers will have exercised the specification in their simulations. It also allows for formal analysis of the golden model, itself.

Example 2: Push-Button Analyzers. The *Modeling* activities needed for formal verification may be even more amenable to reuse.

At Centaur Technology we design an X86 processor. As part of our formal verification effort [27], we wrote an ACL2-based Verilog parsing and translation tool that builds formal ACL2 models of our hardware modules. Since then, we have reused this code in other tools like a linter and an equivalence checker.

These tools can be used by hardware designers with no background in formal verification. They have been quite useful: the linter has found many bugs that testing missed, and circuit designers are frequently using the equivalence checker to check their work.

1.2 The Right Language for the Job

We want it to be very easy to reuse ACL2 models and specifications to develop useful, related applications for end-users outside of our formal verification team. A basic question toward this goal is: what programming language should we use to write these programs?

If the tool we want to write is, say, a simple command-line utility that only needs to read some files and produce some output, then we might just use ACL2 itself as the programming language. This makes it trivial to reuse functions in our formal model.

Unfortunately, ACL2 is a poor platform for developing almost any other kind of program. For instance, it has very little support for working with the file system, limited multi-threading, no networking support, and no graphical interface. It also has no libraries for generating parsers, connecting to databases, working with widely-used data formats like JSON, XML, or YAML, and so on.

Instead, we might write our program in Common Lisp. The ACL2 system itself is a Common Lisp program, and ACL2 models and specifications are compiled into Lisp functions, so it is easy to call ACL2 functions from Lisp. Using Lisp also makes up for some of the deficiencies of ACL2 as our development platform, e.g., Clozure Common Lisp (CCL) has nice threading and networking support.

But frankly, Lisp is a niche language. It lacks the depth of modern, actively developed, well-documented libraries and frameworks enjoyed by mainstream languages. When development time and cost are at a premium, this may limit the kinds of tools we can develop. Using Lisp can also be deterrent to working with developers from other groups since usually they don't know the language.

What we really want, then, is a good way to embed ACL2 models into programs written in other languages—say Ruby, Java, or Python—that are widely known and have plentiful libraries to support working with files, graphics, networks, threads, databases, and so forth. Ideally, we should be able to choose whatever language we think is the best fit for the kind of application we want to develop, and then incorporate our ACL2 models into this language.

This leaves us with a practical problem: how can we effectively integrate ACL2 models into programs written in other languages?

1.3 Contributions

This paper describes the *ACL2 Bridge*, which solves this problem in a general way.

We extend ACL2 with an ACL2 Bridge server that accepts connections from client programs. Clients may be local or remote, and may be written in any practical language. Each client interacts with ACL2 through a kind of read-eval-print loop. Multiple clients can simultaneously interact with the same ACL2 instance. (Section 2).

We describe a Ruby interface to the ACL2 Bridge. We show how a client program can abstract away the details of communicating with the server. Our Ruby interface can execute ACL2 commands in an atomic style. It turns Lisp errors into proper Ruby exceptions, and allows output from ACL2 commands to be streamed as it is produced or collected for analysis. We show how to translate between Ruby and ACL2 data structures. These approaches can be followed to develop clients in other programming languages. (Section 3).

We give a concrete example of a real, end-user program based on the ACL2 Bridge. VL-Mangle is a web-based Verilog refactoring tool. It makes use of a large (100,000+ line) ACL2 codebase for Verilog parsing and transformation. A hardware designer with no knowledge of ACL2 can use the tool, through an attractive GUI, to manipulate sets of Verilog hardware designs and ensure that his changes are correct. (Section 4).

2 THE ACL2 BRIDGE

The ACL2 Bridge works by extending an ACL2 with a server that can respond to client programs. The server code can be loaded with

```
(include-book "centaur/bridge/top" :dir :system)
```

Afterward, a server can be started with the `bridge::start` command. The server listens for connections on a socket. If clients will be run on the same machine, we can listen on a Unix domain socket, which provides some security. For this, we just give the file name for the socket, e.g.,

```
(bridge::start "./my-socket")
```

To support clients on different machines, TCP sockets can be used instead. For this, we just give the port number to use, e.g.,

```
(bridge::start 13721)
```

But TCP sockets have security risks. Any client that can connect to the Bridge can execute arbitrary Lisp commands, including, for instance, running arbitrary programs via system calls. The Bridge has no authentication or encryption mechanisms, so you should never run it on a TCP socket without appropriate firewalls.

2.1 Soundness Considerations

Normally, ACL2 *books* introduce some new logical definitions and prove some theorems, but they do not alter the actual code of the ACL2 system. Unless there is some kind of bug in ACL2 itself, loading a book is *sound*, i.e., it will not allow ACL2 to say it has proved formulas that are not theorems.

The ACL2 Bridge book, however, necessarily extends ACL2 with Common Lisp code for capturing output, starting threads, dealing with sockets, and so forth. Once a server has started, its clients will be allowed to run arbitrary Lisp commands. There are no protections to prevent clients from unsoundly tampering with ACL2's state, e.g., a client could add bad formulas as axioms.

Because of this, loading the Bridge book "infects" ACL2 with a *trust tag*. In short: any ACL2 proofs carried out after the Bridge book is loaded are marked as less trustworthy.

Fortunately, this Common Lisp code is only necessary when you want to start a server and allow clients to connect. To avoid any soundness concerns, our recommended approach (Figure 1) is to not even include the Bridge book during the formal verification effort; the Bridge should only be loaded in the derived application.

It is easy to imagine also using the Bridge to help formal verification engineers develop ACL2 proofs, e.g., by writing graphical tools that make it easier to understand what the prover is doing. It

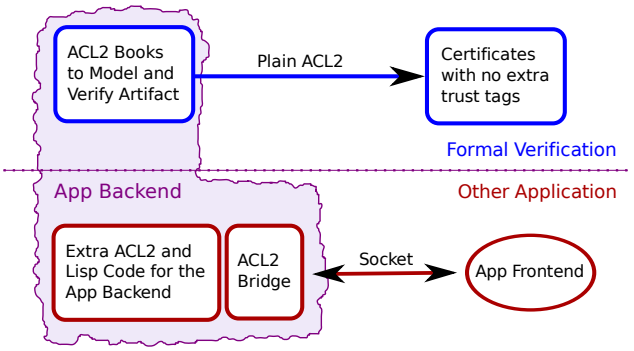


Figure 1. Typical Soundness Approach

should be similarly easy to separate the ACL2 Bridge and such a tool from the formal verification effort. That is, the tool could be used while the proof is being developed, but not when the proof is *certified* (checked).

2.2 Communication

When a client program connects to the ACL2 Bridge, the server's listener thread creates a new worker thread to process its requests. The worker thread provides the client with a kind of read-eval-print loop for executing commands.

At the lowest level, all communication between a worker and its client is carried out using a simple message format which is meant to be easy to produce and parse in any language. In short:

type *len*`\n`
contents`\n`

To be more precise:

- *type* is a label that matches `[A-Z][A-Z0-9_]*` and describes what kind of message this is,
- *len* matches `[0-9]+` and says how many bytes are in *contents* so that no escaping is necessary,
- *contents* are arbitrary bytes of length *len* which are the main part of the message,
- exactly one space separates *type* and *len*, and
- `\n` represents the newline character.

Upon startup, the worker sends the client a `HELLO` message with its own thread-name as the contents. (Some clients might want to remember the name of their worker to implement interrupts.) Then, the work loop begins. The loop has four steps:

Ready. The worker sends an empty `READY` message to indicate it is ready for a command. It then awaits input from the client.

Read. The client sends a command message to the worker. The type of this command can vary, and governs how the return value will be encoded (Section 2.3). The contents should always contain a single Lisp command (an S-expression) for the worker to evaluate.

Eval. If the command is well-formed, the worker runs it. During execution, the worker sends the client `STDOUT` messages with any printed output. These messages are sent as they are generated. A client might choose to display these messages to the end-user as they become available, or to collect them, or to ignore them.

Print. If the command completes successfully, the worker sends a `RETURN` message with the return value, encoded as requested by the client's command. In case of any run-time error, an `ERROR` message containing a description of the problem is sent, instead.

After sending the `RETURN` or `ERROR` message, the loop starts over again with a new `READY` message. The client continues to interact with the same worker until it disconnects.

2.3 Result Encoding

The ACL2 Bridge is intended to make it easy to embed ACL2 models into programs written in other languages. An important part of this is to allow the client to understand return values as proper objects in the client's programming language, not just as text.

In ACL2 and Lisp, objects are printed as S-expressions [21]. There is nothing especially bad about S-expressions, but they are not very popular and few programming languages have a standard library for parsing them. Because of this, if we want to write a client program that does something interesting with an ACL2 return value—say visualize some ACL2 structures, or compare our ACL2 model's answers against those from a hardware simulator—we would first need to write an S-expression parser.

While this would not be too bad, a much more convenient alternative is to have the Bridge encode return values in JSON [9] format. JSON libraries are readily available for any major programming language, and can be especially easy to use. To tie into these facilities, we added a JSON encoder for ACL2 objects to the server.

Clients can choose what kind of encoding should be used for the `RETURN` message on a per-command basis. The choice is just encoded into the command message type. We have four command types. The suffix `MV` here means "multiple values."

Command Type	Result Format
`LISP`	First return value as an S-expression
`LISP_MV`	List of all return values as an S-expressions
`JSON`	First return value as JSON text
`JSON_MV`	List of all return values as JSON text

Dealing with encoding on the ACL2 side, rather than on the client-side, makes each encoding available to clients from all programming languages. We might add other encoding options in the future.

3 A RUBY CLIENT

With the ACL2 Bridge server in place, how much work is it to connect ACL2 to another programming language? To see, we now walk through a Ruby client to the ACL2 Bridge. We first develop an `ACL2Bridge` class that deals with all aspects of messages and the work loop (Section 3.1). We then develop a mechanism for easily translating structured data between the two languages (Section 3.2).

3.1 Client-Side Communication

The `ACL2Bridge` class contains the actual socket and deals with the low-level aspects of communication. Its constructor establishes a connection to the ACL2 Bridge server. Its lowest-level routines implement our message scheme:

- `send_command(type, cmd)` sends the ACL2 server a message with the given `type` and with `cmd` as the contents.

- `read_message()` parses the next message from the server. It returns the type and content for a valid message, or throws an exception for a malformed message.

Higher-level routines bundle up the details of the read-eval-print loop to make it straightforward to just execute a command and get the result as a single step. A nice function is:

```
raw_command(type, cmd, stream=nil)
```

The syntax `stream=nil` means the `stream` argument is optional, and defaults to `nil` when omitted. What does this function do?

- It calls `send_command(type, cmd)` to send the command to the server.
- It reads messages from the server until a `READY` message is encountered. As each message is read, if an output `stream` (any object with a `<<` method) has been provided, any `STDOUT` messages will be forwarded to this stream. This is useful for long-running ACL2 commands that print progress messages; you can show these messages to the user as they are generated, instead of having to wait for the command to complete.
- After `READY` has been read, it checks whether any `ERROR` message was encountered. If so, it throws the error as an exception. It then ensures that a `RETURN` message was encountered or throws an exception. Finally, it returns the contents of the `RETURN` message (i.e., the result of executing the ACL2 command, encoding according to `type`), and the concatenation of all `STDOUT` messages.

This sort of wrapper is very convenient when writing an end-user application. Instead of using `raw_command` directly, we usually use:

```
json_command(cmd, stream=nil),
```

a simple wrapper that calls `raw_command` with `JSON` as the command type, and then bundles up the result and all of the standard output into a single, JSON-encoded string.

Altogether the `ACL2Bridge` class comes to only 250 lines (of which 140 are blank or comments). It bundles up the details of messages and the work loop so that the client application simply submits commands and gets back the results and printed output. It converts any communication errors or ACL2-level errors into real Ruby exceptions, which can be caught and dealt with at the appropriate level of the client application. Porting this code from Ruby to other languages like Java, Python, C#, etc., would be quite easy.

3.2 Data Translation

The `ACL2Bridge` class abstracts away the message scheme and work loop, but still leaves us with an interface that is entirely string based. That is, the commands we send to the server need to be strings containing S-expressions. Likewise, the replies we get back are strings that contain printed S-expressions or JSON text.

Strings are fine in limited cases, but they are not a good representation for structured data. What we would ideally like, instead, is an easy way to translate between Ruby structures and ACL2 objects.

From Ruby to S-Expressions. Ruby has many kinds of objects. It makes sense to translate some of these (integers, symbols, arrays of strings, . . .) into ACL2 objects. But for other kinds of Ruby objects (functions, sockets, its garbage collector, its Math package, . . .) there is no sensible equivalent.

An especially nice translation from sensible Ruby objects into S-expressions can be implemented using Ruby's duck-typing and monkey-patching features.

- We start by extending (monkey patching) classes like `Integer` and `String` with a `to_lisp` method.
- We then similarly extend classes like `Array` and `Hash` with `to_lisp` methods that build suitable S-expressions using the `to_lisp` methods of their elements (duck typing). With just this, we can convert arrays/hashes whose elements are basic types, sub-arrays/hashes of basic types, etc., into S-expressions.
- As we write new Ruby classes for our application, we can add them to our translation scheme just by defining a `to_lisp` method. This immediately extends to arrays and hashes that contain these new objects.

This approach would not be directly portable to more strict languages like Java. In such a language, you might instead have a class with encoders for basic types, and use a `LispEncodable` interface with a `toLisp` function for new classes.

From ACL2 to Ruby Objects. In the other direction, we would like a way to convert ACL2 replies into Ruby objects. Using JSON encoding makes this particularly easy: in Ruby, we can convert some JSON-encoded string, `text`, into a native Ruby object, `obj`, like this:

```
obj = JSON.parse(text)
```

This approach works well for basic structures involving lists, alists, etc., but is not suitable for all ACL2 objects. For instance, if our data is represented as some special cons-tree, its JSON representation may end up being an bizarre nesting of two-element arrays. In these cases, it may be best to write a custom encoding function in ACL2, and explicitly run the encoder in your Lisp command.

4 EXAMPLE APPLICATION: VL-MANGLE

Connecting ACL2 to other languages lets us reuse a formal ACL2 codebase to develop modern applications for end-users beyond the formal verification group. As a concrete example, we have used the ACL2 Bridge to develop a web-based Verilog refactoring tool called VL-Mangle. VL-Mangle allows a hardware designer to mechanically transform sets of Verilog modules in certain ways. For instance, it can be used to:

- Inline away all uses of particular modules,
- Rewrite gate-level constructs into assignment statements,
- Remove excessive intermediate wiring,
- Eliminate unused wires and unnecessary expressions,
- Perform basic logical simplifications,
- Merge bidirectionally connected wires into a single wire, and
- Vectorize compatible sets of assignments.

These sorts of edits are tedious and error-prone to carry out on a large scale by hand. To support a particular design effort, we wanted a tool that hardware designers could use to automate these tasks.

A high-level picture of VL-Mangle's architecture is shown in Figure 2. The backend is written in ACL2. It reuses VL, our large (100,000+ line) ACL2 codebase for Verilog parsing and transformation. VL was originally developed to support our formal verification effort. To prove anything about our processor's modules (which are written in Verilog) we first needed a way to model these modules in ACL2, so we developed a parser and then translation code. Since then, we have significantly extended VL and used it to develop various command-line tools like a linter and equivalence checker.

The frontend of VL-Mangle is a web application written in the Ruby framework Sinatra [29] on the server side, and a typical mix

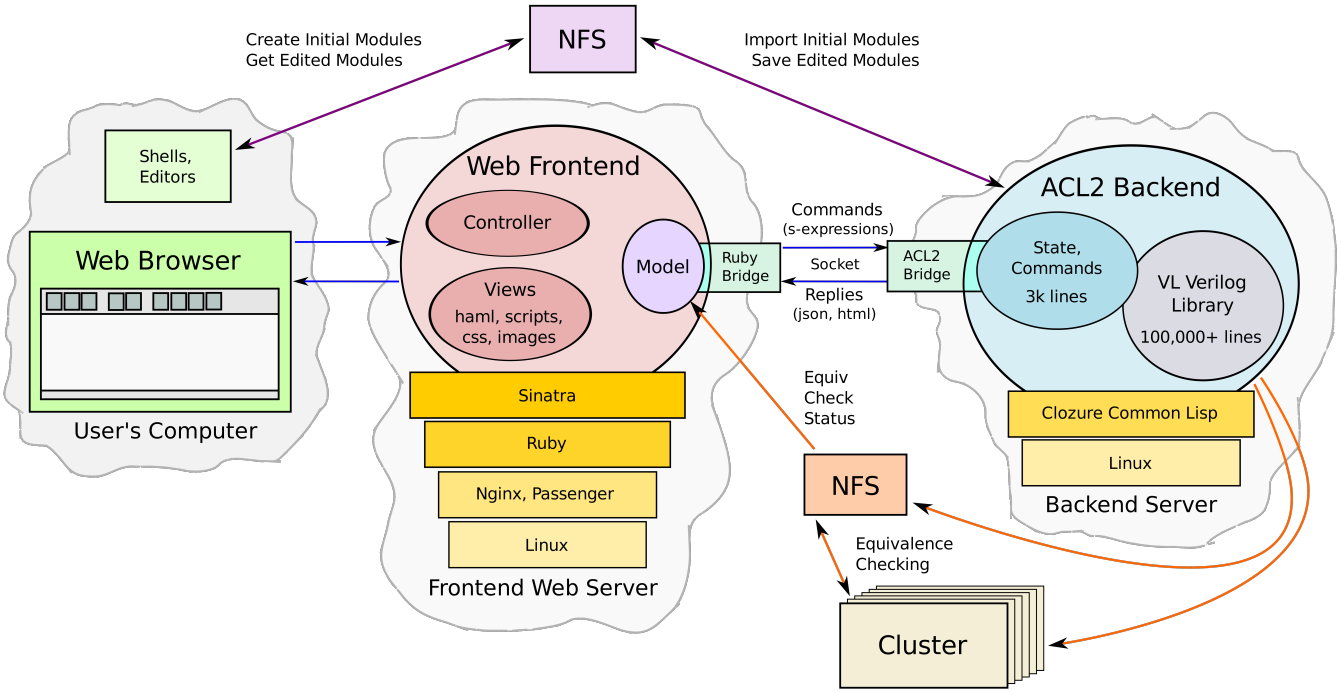


Figure 2. VL-Mangle Architecture

of HTML, CSS, Javascript/JQuery for the client. The end-user interacts with the tool using a graphical interface within his web browser. To give you a sense of the user experience, some screenshots of are shown in Figure 3.

4.1 VL-Mangle Architecture

The frontend follows the Model-View-Controller (MVC) [7] design pattern. In brief, this means it is divided into three parts. The *Model* includes the actual data and the ways of operating on this data. The *Views* are responsible for displaying data from the model to the user. The *Controller* is responsible for interpreting user input and translating it into operations on the model.

This architecture is very typical for web applications. In fact, there is almost nothing to say about our views and controller except that they are entirely conventional. Our controller is written using the Sinatra framework for routing and forms handling. Our views use libraries like HAML, Sass, and JQuery. All of this is separate from ACL2 except that we reuse some of the VL library's routines for pretty-printing Verilog modules.

The model is more interesting. Typical web applications might use an SQL database to hold their data. In VL-Mangle, most of the model is implemented within the ACL2 backend. The frontend, however, hides this behind a Ruby `Model` class that also regards certain files on a shared networked file system (NFS) as part of the model.

The ACL2 Backend. We think of the entire ACL2 backend as part of VL-Mangle's Model. The backend represents almost all of our application's data using ordinary ACL2 data structures. We reuse the Verilog representation from the VL library. The most interesting other data structures are *frames* and the global *state*.

A frame contains a list of Verilog modules and some other information. Each kind of automated edit (e.g., "inline modules") is implemented as a frame transformation. That is, given some starting frame, it produces a new frame that has updated modules. These are just ordinary ACL2 functions.

The global state has two stacks of frames: an undo stack and a redo stack. The top frame on the undo stack is the current frame. Basic undo and redo support is simple: to undo we move a frame from the undo stack to the redo stack; to redo we do the reverse.

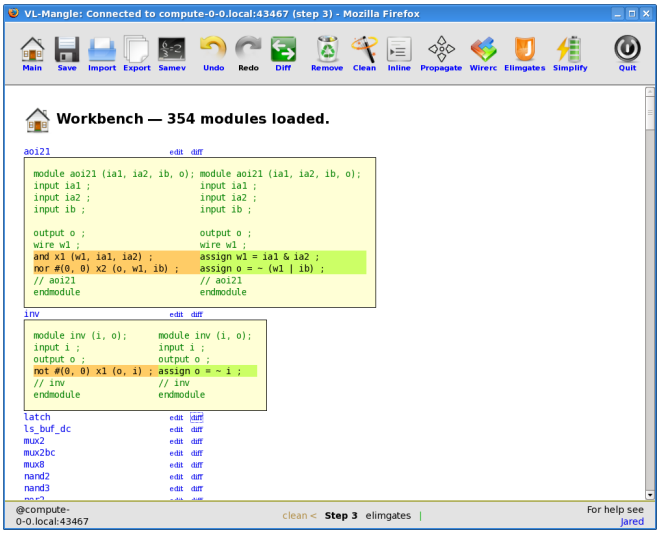
Since the state is an ordinary ACL2 object, it is easy to implement progress saving/reloading that preserves the full undo/redo history. Interestingly, none of this code needs to involve the ACL2 Bridge; we only need the Bridge when we want to connect our ACL2 model to the Ruby web application.

The Ruby `Model` Class. To connect the ACL2 model to our web application, we load the ACL2 Bridge book on the ACL2 side and load our Ruby `ACL2Bridge` class, S-expression encoding code, and the Ruby JSON library on the Ruby side.

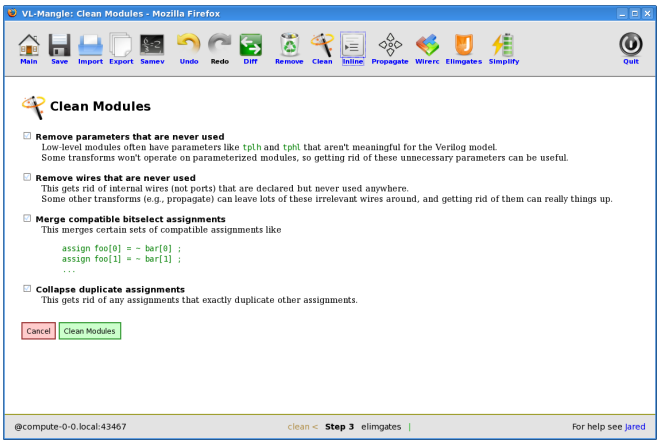
Instead of exposing the `ACL2Bridge` instance to the rest of the Ruby application, we keep it within a `Model` class. In some ways this feels like overkill: there's not much to this class, and it might be simpler to just do without it. On the other hand, there are some nice features of this approach.

Having a `Model` class in Ruby allows us to treat data outside of ACL2 as part of the model. Our main use of this in VL-Mangle is for equivalence checking. To make equivalence checking faster, we set up a separate job to check each module, and run these jobs on a cluster. The VL-Mangle interface lets the user see the progress of these jobs and inspect failures. The data for these views are the log files from the equivalence checking tool. From an MVC perspective, then, it makes sense to regard these log files as part of the model. (Applications other than VL-Mangle might also want to consider various non-ACL2 resources like databases as part of their model.)

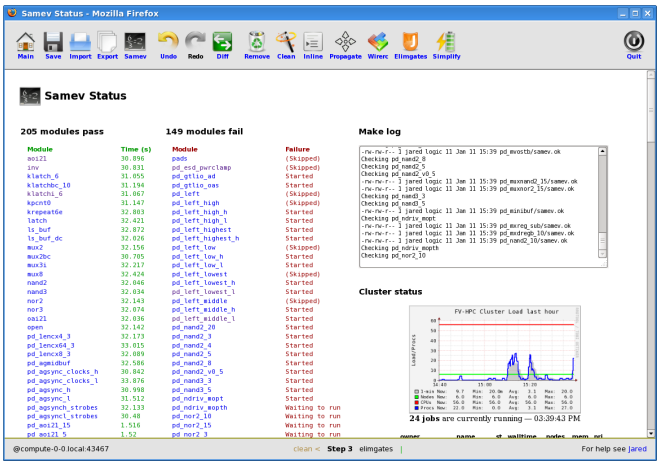
Another advantage of having a `Model` class is that it makes caching ACL2 queries quite easy. A particular view might be assembled out of independent parts. These different parts might each need to know, say, what the current module names are. We could just



(a) The workbench shows the user the current versions of his modules, and lets him compare them with previous versions.



(b) Tool pages let the user transform sets of modules in various ways.



(c) The user can run equivalence checks on a cluster of machines to ensure his edits are behavior-preserving.

Figure 3. VL-Mangle Screenshots

separately query the Bridge each time we need to answer a question like this, but that is not very efficient since each query requires a round-trip to the ACL2 backend. Since the answer to this questions won't change during a single page load, a simple improvement is to cache the answers in the `Model` class so that we only need to consult ACL2 once, for the first query. In the particular case of VL-Mangle, this caching isn't necessary or important—we normally have a single user interacting with a single backend and performance is just not an issue—but for applications other than VL-Mangle, caching might be useful.

4.2 Connecting ACL2 to the Web

A very nice part of this whole system is just how easy it is to transfer input from HTML forms to the ACL2 backend and work with ACL2 replies. The user enters their input into ordinary HTML forms, with input names like this:

```
<input name="clean[parameters]" ... />
<input name="clean[wires]"      ... />
<input name="clean[assigns]"    ... />
```

Using the Sinatra framework, the corresponding handler in our controller can refer to `params[:clean]`, a Ruby hash that binds input names to their values. This makes it trivial to send these inputs directly to our Ruby `Model` instance:

```
reply = @model.clean(params[:clean], out)
```

The model just converts the arguments into an S-expression and runs the corresponding ACL2 command:

```
class Model
  ...
  def clean(args, out)
    @bridge.json_command(
      "(mpost-clean '#{args.to_lisp})", out)
  end
end
```

When we want to add new options and arguments to the cleaning transform, we can just extend the HTML form and its ACL2 implementation, without any changes to the Ruby model or controller.

Implementing an API for use in AJAX queries is also very simple. For instance, in the Ruby `Model` class we have a method to query ACL2 for the current module names.

```
class Model
  ...
  def get_modnames_json()
    @bridge.json_command("(mget-modnames)")
  end
  ...
end
```

Since `get_modnames_json` is already returning JSON-encoded data, we can just send this string directly to the web browser to respond to AJAX requests. All that is needed is an appropriate route in our controller. In Sinatra, this is just:

```
get "/get_modnames" do
  connect
  content_type :json
  @model.get_modnames_json
end
```

4.3 Threading Considerations

Most ACL2 functions can be thought of as pure, functional programs with no side-effects. These functions are especially well-behaved and no special care needs to be taken to make them thread-safe.

But not everything in ACL2 is pure. For greater execution efficiency, ACL2 models can also make use of certain non-pure idioms. For instance, they can use "single-threaded objects" that are updated destructively. At Centaur we actually use ACL2(h) [6], an extended version of ACL2 with hash-consing and memoization features. Unfortunately, its implementation of memoization is not thread-safe, and its implementation of hash-consing is most efficient when only a single thread is creating new hash-conses.

These sorts of features pose challenges when we are developing a multi-client applications where each client is served by a separate worker thread. Two clients might, for instance, simultaneously try to update the same single-threaded object, or both make use of memoized computations. The bridge does not do any automatic locking, so when we develop client programs, we must be aware of these issues and add the appropriate protections.

As a blunt solution, the Bridge does have a special "main thread" feature which is especially useful in the context of ACL2(h). In short: any computation can be wrapped in `in-main-thread` to ensure that it is run only by the main thread. This has the obvious disadvantage that a client may need to wait until the main thread becomes available. Another command, `try-in-main-thread`, is similar but just fails immediately if the main thread is not available. In VL-Mangle, we use this as our main locking mechanism.

5 OTHER APPROACHES

Getting separate programming languages to work together is a well-fought problem. Depending on the kinds of languages involved, we might combine the codebases into a single program by developing a foreign-function interface (FFI) or by sharing a multi-language platform like the Java Virtual Machine (JVM) or the Common Language Runtime (CLR). Alternately, we might keep the separate codebases as independent programs that simply process each others' files, or that communicate over pipes or sockets using anything from messages to elaborate protocols like COM and CORBA. Some of these approaches could perhaps be used, instead of the ACL2 Bridge, to connect ACL2 to other languages.

Common Lisp implementations like CCL and SBCL have foreign function interfaces that can call C functions from Common Lisp code, which could provide access to libraries for graphical interfaces, databases, etc. This can be particularly efficient. Calls through the ACL2 bridge have some communication overhead: the server converts return values into S-expressions or JSON representations, and the client generally has to parse this text into a sensible object. With an FFI, you may be able to directly construct or modify structures of interest in memory, without any parsing or printing. Unfortunately, while an FFI usually makes it reasonable to interface with C, connecting to higher-level languages like Java or Ruby is more difficult.

ACL2 does not run on any Common Lisp implementation that targets a platform like the JVM or CLR. However, there is at least one Common Lisp implementation on the JVM (ABCL) and languages like Clojure are similar to Lisp. Porting ACL2 to these platforms might open up interesting ways to connect it to the other languages that also run on the JVM, but would be a significant undertaking.

Instead of using separate programs that communicate in a cooperative way over a socket, we could perhaps use a pipe to run ACL2 as a sub-process and capture its standard input/output streams. This approach is used in the ACL2 Sedan [11], an Eclipse-based IDE for ACL2. This approach avoids the need to extend ACL2 with a server, which is nice since socket/threading code is not standard across Lisp implementations. Unfortunately, there are many practical difficulties for the client. The client must deal with invoking the process and capturing its streams, which is difficult in some programming languages. It must invent some way to tell when ACL2 is ready for more input, and to distinguish between output, return values, and error messages that are all printed to a single output stream. This approach is also limited to a single client, interacting with ACL2 via a single thread, on the same machine.

6 CONCLUSIONS

The ACL2 Bridge provides a straightforward way to embed formal ACL2 models and specifications into software written in any mainstream language. This allows us to reuse the work of formally modeling and specifying artifacts to develop full-featured applications that can be valuable beyond the verification team.

6.1 Related Work

In closely related work, Greve, Hardin, and Wilding [12, 14] explain how they have developed formal processor models that can be executed efficiently. This allows the formal model to be reused as a traditional processor simulator.

A unique reuse of executable formal hardware models is described by Albin, Brock, and Hunt [1]. They have reused a formal model of the FM9001 processor for post-fabrication testing. Test programs were run on the actual FM9001 chip while it was attached to a logic analyzer that recorded the values of its interfacing pins. These values were then compared against a gate-level formal model of the hardware design to show the physical device was behaving correctly.

To reuse formal models in other software, we need to be able to execute the formal model. ACL2 is unusual among theorem provers in that its logical definitions (i.e., for formal models and specifications) are directly executable as Common Lisp functions. In many other systems, logical definitions are often developed using, e.g., quantifiers, predicates, and relations that are not directly executable.

Even so, many provers have a mechanism for executing models. The Coq [5] system features a *program extraction* [19] capability that can translate subsets of Coq into OCaml programs. This capability has been used to develop some impressive standalone applications. For instance, Leroy [18] describes CompCert, a formally verified C compiler; Koprowski and Binsztok [17] present TRX, a verified parser generator. In each case, the programs extracted from these Coq developments could be useful to a wide audience.

A similar mechanism [3] for translating Isabelle/HOL specifications into ML has been used by Berghofer and Strecker [4] to create a verified compiler for a simplified Java. (It also has other uses within the theorem prover, e.g., Chaieb and Nipkow [8] have developed verified proof procedures for more efficient arithmetic reasoning.)

Similar to program extraction are schemes to *animate* [22] formal models in languages like Z and B, e.g., by translation into Prolog programs. Animation is ordinarily used to build confidence in the formal model by allowing it to be tested on examples. We are not aware of applications based on these animated models, but the ability to execute the model may serve as a useful step in this direction.

Less closely related, there are many cases where theorem provers have been connected to external programs like SAT solvers[28, 10],

SMT solvers [2], symbolic algebra systems [15], and so on, which are usually written in languages like C or C++. These efforts allow formal verification engineers to automatically prove certain kinds of goals, but are not aimed at reusing formal models in applications.

6.2 Availability

The ACL2 Bridge, including for both the server-side ACL2 source code described in Section 2 and the Ruby client described in Section 3, is freely available under the GNU General Public License. It is included in the ACL2 Community Books for ACL2 6.0,

`http://acl2-books.googlecode.com/`,

under `books/centaur/bridge`. The Verilog library used in VL-Mangle, including parsing and many transformations, are also available under `books/centaur/vl`. However, the VL-Mangle web frontend is not publicly available.

ACKNOWLEDGEMENTS

I thank Scott Peterson and Patrick Roberts for very helpful conversations about web application development. I thank Sol Swords for his help to integrate equivalence checking into VL-Mangle. I thank Anna Slobadová, Sol Swords, and the anonymous reviewers for their helpful feedback on drafts of this paper.

REFERENCES

[1] Kenneth L. Albin, Bishop C. Brock, Warren A. Hunt, Jr., and Lawrence M. Smith, 'Testing the FM9001 microprocessor', Technical Report 90, Computational Logic, Inc., (January 1995).

[2] Michaël Armand, Germain Faure, Benjamin Grégoire, Chantal Keller, Laurent Théry, and Benjamin Werner, 'Verifying SAT and SMT in Coq for a fully automated decision procedure', in *PSATTT '11*, (2011).

[3] Stefan Berghofer and Tobias Nipkow, 'Executing higher order logic', in *Types for Proofs and Programs (TYPES)*, volume 2277 of *LNCS*, pp. 24–40. Springer, (2002).

[4] Stefan Berghofer and Martin Strecker, 'Extracting a formally verified, fully executable compiler from a proof assistant', in *Compiler Optimization Meets Compiler Verification (COCV)*, volume 82 of *Electronic Notes in Theoretical Computer Science*, (2003).

[5] Yves Bertot and Pierre Castéran, *Interactive Theorem Proving and Program Development: Coq'Art: The Calculus of Inductive Constructions*, Texts in Theoretical Computer Science, Springer-Verlag, 2004.

[6] Robert S. Boyer and Warren A. Hunt, Jr., 'Function memoization and unique object representation for ACL2 functions', in *ACL2 '06*, pp. 81–89. ACM, (August 2006).

[7] Steve Burbeck. Application programming in Smalltalk-80: How to use Model-View-Controller (MVC). University of Illinois in Urbana-Champaign Smalltalk Archive. `http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html`, 1987. Accessed: January 2013.

[8] Amine Chaieb and Tobias Nipkow, 'Verifying and reflecting quantifier elimination for Presburger arithmetic', in *Logic Programming, Artificial Intelligence, and Reasoning (LPAR '05)*, volume 3835 of *LNCS*, pp. 367–380. Springer-Verlag, (2005).

[9] Douglas Crockford. JSON: The fat-free alternative to XML. `http://www.json.org/fatfree.html`, December 2006. Accessed: January 2013.

[10] Ashish Darbari, Bernd Fischer, and João Marques-Silva, 'Industrial-strength certified SAT solving through verified SAT proof checking', in *ICTAC '10*, volume 6255 of *LNCS*, pp. 260–274. Springer, (2010).

[11] Peter C. Dillinger, Panagiotis Manolios, J Moore, and Daron Vroon, 'ACL2s: The ACL2 Sedan', in *7th Workshop on User Interfaces for Theorem Proving (UITP)*, volume 172 of *Electronic Notes in Theoretical Computer Science*, pp. 3–18. Elsevier, (2006).

[12] David Greve, Matthew Wilding, and David Hardin, 'High-speed, analyzable simulators', in *Computer Aided Reasoning: ACL2 Case Studies*, eds., Matt Kaufmann, Panagiotis Manolios, and J Strother Moore, 89–106, Kluwer, (2000).

[13] David A. Greve, Matt Kaufmann, Panagiotis Manolios, J Strother Moore, Sandip Ray, José Ruiz-Reina, Rob Sumners, Daron Vroon, and Matthew Wilding, 'Efficient execution in an automated reasoning environment', *Journal of Functional Programming*, **18**(1), (January 2008).

[14] David Hardin, Matthew Wilding, and David Greve, 'Transforming the theorem prover into a digital design tool: From concept car to off-road vehicle', in *Computer Aided Verification (CAV)*, volume 1427 of *LNCS*. Springer, (1998).

[15] John Harrison and Laurent Théry, 'A sceptic's approach to combining HOL and Maple', *Journal of Automated Reasoning*, **21**, 279–294, (1998).

[16] Matt Kaufmann, Panagiotis Manolios, and J Strother Moore, *Computer-Aided Reasoning: An Approach*, Kluwer Academic Publishers, June 2000.

[17] Adam Koprowski and Henri Binsztok, 'TRX: A formally verified parser interpreter', *Logical Methods in Computer Science*, **7**, 1–26, (June 2011).

[18] Xavier Leroy, 'Formal verification of a realistic compiler', *Communications of the ACM*, **52**(7), 107–115, (2009).

[19] Pierre Letouzey, 'Extraction in Coq: An overview', in *4th Conference on Computability in Europe (CiE)*, volume 5028 of *LNCS*, (2008).

[20] Hanbing Liu, *Formal Specification and Verification of a JVM and its Bytecode Verifier*, Ph.D. dissertation, University of Texas at Austin, 2006.

[21] John McCarthy, 'Recursive functions of symbolic expressions and their computation by machine, part 1', *Communications of the ACM*, **3**(4), 184–195, (April 1960).

[22] Tim Miller and Paul Strooper, 'Animation can show only the presence of errors, never their absence', in *13th Australian Software Engineering Conference (ASWEC)*, pp. 76–85, (2001).

[23] Lee Pike, Mark Shields, and John Matthews, 'A verifying core for a cryptographic language compiler', in *6th International Workshop on the ACL2 Theorem Prover and its Applications (ACL2)*, pp. 1–10. ACM, (2006).

[24] Raymond J. Richards, 'Modeling and security analysis of a commercial real-time operating system kernel', in *Design and Verification of Microprocessor Systems for High-Assurance Applications*, ed., David S. Hardin, Springer, (2010).

[25] David Russinoff, Matt Kaufmann, Eric Smith, and Robert Sumners, 'Formal verification of floating-point RTL at AMD using the ACL2 theorem prover', in *17th IMACS World Congress: Scientific Computation, Applied Mathematics and Simulation*, (July 2005).

[26] Jun Sawada and Erik Reeber, 'ACL2SIX: A hint used to integrate a theorem prover and an automated verification tool', in *Formal Methods in Computer-Aided Design (FMCAD)*, pp. 161–170. IEEE, (2006).

[27] Anna Slobadová, Jared Davis, Sol Swords, and Warren A Hunt, Jr., 'A flexible formal verification framework for industrial scale validation', in *Formal Methods and Models for Codesign (MemoCode)*, pp. 89–97. IEEE, (July 2011).

[28] Tjark Weber and Hasan Amjad, 'Efficiently checking propositional refutations in HOL theorem provers', *Journal of Applied Logic*, **7**(1), 26–40, (March 2009).

[29] Adam Wiggins and Blake Mizerany. Lightweight web services. RubyConf 2008. `http://rubyconf2008.confreaks.com/lightweight-web-services.html`, November 2008.

[30] Matthew M. Wilding, David A. Greve, Raymond J. Richards, and David S. Hardin, 'Formal verification of partition management for the AAMP7G microprocessor', in *Design and Verification of Microprocessor Systems for High-Assurance Applications*, ed., David S. Hardin, Springer, (2010).

Proof-Guided Ontology Development using Pattern Rules

Liwei Deng and **Alan Bundy** and **Fiona McNeill** and **Alan Smaill** [1]

Abstract. In this paper we present a proof-based method of developing ontologies that satisfy requirements, where the proof process guides the development of the ontology. We formally define our procedure and illustrate it with examples.

1 INTRODUCTION

Ontologies are (formal) models of some aspect of the real world, with applications such as information sharing, semantic search and interoperability. There are various ontologies in use, including many business and biomedical ones.

Developing (and maintaining) such ontologies involves many activities, sources of knowledge and people. The experience over the years have been analysed, organised and developed into *methodologies*, describing the whole process of developing an ontology; see [2] [1] for meta-surveys.

Generally there are three main stages:

1. first, purpose and requirements are identified, relevant knowledge gathered, and an informal ontology may be built;
2. then, the formal ontology is created;
3. finally, the ontology is evaluated against the purpose and requirements, modified accordingly, and maintained as long as necessary.

We aim to find good methods for stages 2 and 3 above, that of creating the formal ontology that satisfies formal requirements. Such ontologies have the quality of *relative correctness*, correct relative to the requirements.

Establishing the satisfaction of some formal requirements can take the form of proofs: if the (statements of the) formal ontology are seen as a set of axioms, these formal requirements are conjectures on the ontology to be proved.

For example, for an anatomy ontology, we could have a requirement

> The human body is left-right symmetric externally;

which can be formalised as

> *external_symmetric(human_body)*.

We will be able to deduce this from the ontology if, for example, the ontology had, for each external component, both a left part and a right one, or is designated self-symmetric.

This approach of formally proving satisfaction of requirements by ontologies was first introduced in TOVE (TOronto Virtual Enterprise) [3], and we will discuss this later in Section 3, Related Work.

[1] University of Edinburgh, Scotland, email: {L.Deng-2@sms.ed.ac.uk, bundy@staffmail.ed.ac.uk, f.j.mcneill@ed.ac.uk, A.Smaill@ed.ac.uk}

In this paper we are only interested in formal requirements that could be stated as conjectures on and proved to follow from the formal ontology and henceforth we will be using the term *formal requirements* to mean only these ones.

For creating formal ontologies where formal requirements could be proved, we propose *Proof-Guided Ontology Development*, where we use the formal requirements to *guide* the development of the ontology. We start with the formal requirements and use their proof attempts to guide the user on the relevant classes, relations and rules to add to the ontology so that it reflects the user's view of the domain, at the same time as giving proofs of the statements. When the development of the ontology is finished, all the requirements should be proved, so we know that they are all satisfied.

This use of formal requirements to guide the development of the ontology is different from existing approaches.

The rest of the paper is structured as follows: in Section 2 we show the use of certain *pattern rules* in guiding the development of the ontology, starting with a simple example in Section 2.1, followed by a formal description of our procedure as an algorithm in Section 2.2, which is then applied to a more complicated example in Section 2.3, then a brief discussion in Section 2.4, and a brief description of the implementation of the example in Section 2.5, followed by its evaluation in Section 2.6; we then discuss related work in Section 3, described on-going and planned work in Section 4, before concluding in Section 5.

2 USING PATTERN RULES

Our development process aims to end in an ontology where all the formal requirements are satisfied, and to get there we aim to go through an iterative series of states in each of which an intermediate ontology satisfies an intermediate set of formal requirements (which could be a subset of the final ones).

We start with a state with no formal requirements and an empty ontology (with no classes, relations, properties, etc.), so that trivially the ontology satisfies the requirements. The user who is building the ontology then changes the state by adding formal requirements, and classes, relations and properties to the ontology. The resulting intermediate ontology is checked to see if it satisfies the requirements. If not, then we aim to use the formal requirements to guide the user to make changes to the current ontology to arrive at a state where the requirements are all satisfied (again). We carry on like this until we get to the ontology we want.

Note from the above description that we are not starting from complete scratch, but assume that the user has some idea of the requirements, and hence classes, relations and axioms, for the ontology and will be providing them.

A key part of this process is the guidance we provide to the user in cases the intermediate ontology does not satisfy the requirements.

We said above that we will use the formal requirements to provide guidance, but on their own usually they will not be enough. In this paper we will show the use of certain pattern rules as rules of implication in giving guidance to the user.

We first illustrate the idea with a simple example.

2.1 A Simple Example

Our simple example is that of a car ontology, where we are trying to model the structure of a car. We start with no formal requirements and an empty ontology, as described above. The user then adds the requirement

The engine is connected to the wheels;

which could be formalised as

connected(engine, wheels).

Now if the requirement is added to the ontology, we would have immediate (and trivial) satisfaction of the requirement by the ontology, but there are various reasons for not doing this. One is that we want to use the requirement to guide the development of the details of the ontology, in a specific way, which we will see below soon. Another is that eventually we want to prove generalised versions of the above requirement and these cannot be axioms of the ontology.

Instead, the above requirement would simply imply the addition of the classes *engine*, *wheels* and the relation *connected* to the ontology.

To these, the user might further add the axioms

connected(engine, axle), connected(axle, wheels)

to the ontology to model how the engine is connected to the wheels.

Now a check can be performed to see if the current ontology satisfies the above requirement. It is clear that we will not be able to deduce it: it is not an axiom of the ontology, but it cannot be implied by the other axioms either as we do not have any rules to use for implication. Now we note that if we had properties for the relation *connected* such as transitivity, which could be formalised as

$$connected(X, Z),\ connected(Z, Y) \rightarrow connected(X, Y), \quad (1)$$

for example, then it could be used as a rule of implication, with the two atoms on the left implying the one on the right.

So suppose we can have such properties, the question is then which ones should we use? A natural first try is the commonly occurring ones, such as transitivity, symmetry, etc. But where do we get such properties from if they are not there?

One possibility is to have a generic versions of these commonly occurring properties, which could be instantiated with any particular relation we might want. For transitivity such a generic property could be formalised as follows

$$P(X, Z),\ P(Z, Y) \rightarrow P(X, Y), \quad (2)$$

with P a variable over relations. If P were instantiated to *connected*, we would get (1) above. In this way, we get rules to try in a proof attempt of the formal requirement; these are our *pattern rules*. A list of some commonly occurring generic properties as pattern rules is shown in Figure 1; this is not intended to be exhaustive.

Our guidance then proceeds by backward reasoning. We start our attempted proof of the formal requirement

connected(engine, wheels)

as the goal by first noting that it is not an axiom of the ontology, so will need to be deduced. Then we look for a rule that could be used to break down the goal into subgoals. We note that properties such as (1) could be used as rules, and first look for a rule of this form, but find none. So next we look for a generic rule of the form (2) to obtain an instantiated rule of form (1), where the relation variable in (2) is instantiated to the relation in our goal. So from (2) we obtain (1); the conclusion of this implication is removed by resolution leaving only the two premises,

$$connected(engine, Z),\ connected(Z, wheels) \quad (3)$$

as subgoals. We note that they could be unified with the axioms in the ontology, thus giving a derivation of our original goal. This triggers a notification to the user of a potential derivation of our goal, depending on the use of the instantiated rule (1), which could be a property for the relation *connected*. The user is asked if (1) is indeed a property for the relation, with the successful derivation providing evidence in its favour. If the user accepts, the property is added to the ontology, and we have guided the user to a state where all formal requirements could be derived.

Having given the example, we can now state our hypothesis for this paper:

Proof-Guided Ontology Development using pattern rules can be used to provide guidance in the development of real ontologies.

We will come back to the hypothesis when we discuss evaluation in Section 2.6, but next we present a formalisation of the above procedure.

2.2 The General Algorithm

Our procedure takes formal requirements and applies them to ontologies using pattern rules, to provide guidance. So before presenting a formalisation of this procedure as an algorithm, we first give formal definitions for each of the three involved elements. Our formalisation is in the style of logic programming, see e.g. [7].

2.2.1 Requirements and Axioms

We first give a basic definition, that of *atoms*, that will allow us to define formal requirements and atomic axioms.

Definition 1 (Atoms) *An* atom *is a formula $P(T_1, \ldots, T_n)$, where P is a predicate and the T_is are terms, which can only be constants or variables. If P is a constant then $P(T_1, \ldots, T_n)$ is a* regular *atom; if the T_is are also constant then it is a* ground *atom. If P is a variable then $P(T_1, \ldots, T_n)$ is a* pattern *atom.*

Remark 1 *The restriction of terms to constants or variables in the above definition has implications for the size of the search space, see Remark 4.*

Example 1 (Atoms) *connected(X, Y) and connected(engine, Y) are both regular atoms, where connected is the predicate in both cases, with X and Y the terms for the first atom, $engine$ and Y the ones for the second.*

We can now define .

Definition 2 (Formal Requirements) *A* formal requirement *is a ground atom, where the predicate is a relation and the terms are classes of the ontology.*

Definition 3 (Atomic Axioms) *An* atomic axiom *is a ground atom, where the predicate is a relation and the terms are classes of the ontology.*

Remark 2 *For this paper formal requirements and atomic axioms have the same definition, but in future work this is unlikely to be the case, as we expect to have e.g. more general requirements (cf discussion about why requirements are not just added to ontologies as axioms at the start of Section 2.1).*

Example 2 (Formal Requirements, Atomic Axioms) $connected(engine,\ wheels)$ *is a formal requirement;* $has\_part(wheels,\ frontwheels)$ *is an atomic axiom (these are taken from our example ontology in Section 2.3 that we will see later).*

2.2.2 Pattern Rules

Having defined atoms, we can use them to define (Horn) clauses of various types, including pattern rules.

Definition 4 (Horn Clauses) *A* Horn clause *is either a rule clause or a goal clause. A* rule clause *is a formula* $Body \implies Head$, *where* $Head$ *is an atom and* $Body$ *is a (possibly empty) set of atoms, where all the terms in each atom are variables. A* goal clause *is a set of atoms. If the goal clause is the empty set it is called the* empty clause.

A rule clause is a regular clause *iff it contains only regular atoms; it is a* pattern clause *if it contains pattern atoms.*

A regular rule clause where the predicate is a relation of the ontology is a property of the relation.

Remark 3 *We could have said* first order *instead of* regular *and* second order *instead of* pattern *in the above definition, but we wanted to emphasise the notion of pattern rules.*

Definition 5 (Pattern Rules) *A* pattern rule *is a pattern clause with only one predicate variable.*

Example 3 (Horn Clauses)

$$connected(X, Z),\ connected(Z, Y) \rightarrow connected(X, Y)$$

is a regular clause; it is also a property of the relation connected *if* connected *is a relation of an ontology;*

$$P(X, Z),\ P(Z, Y) \rightarrow P(X, Y)$$

is a pattern rule (pattern clause), as is

$$connected(X, Z),\ P(Z, Y) \rightarrow P(X, Y);$$

$\{connected(X,\ Y)\}$ *and* $\{connected(engine,\ Y)\}$ *are goal clauses.*

Definition 6 (Ontologies) *An* ontology *contains (non-exhaustively) a set of classes, relations, atomic axioms and regular rules.*

Example 4 (Ontologies) *See Figure 2.*

2.2.3 Proof Guidance Algorithm

Next, we use the various types of clauses to define *search spaces*, and finally our formal algorithm, the *Proof Guidance Algorithm*.

Definition 7 (Search Spaces) *A* search space *is an OR-tree*[2] *in which the nodes are labelled by goal clauses and the arcs are labelled by rule clauses. Exactly one of the atoms of each goal clause is called its* selected goal.

Suppose $\{A_1, A_2, \ldots, A_n\}$ *is the goal clause labelling a node with* A_1 *the selected goal, and the rule clause* $Body \implies Head$ *labels one of the arcs descending from this node. Then the daughter node at the other end of this arc is labelled with* $Body\sigma \cup \{A_2, \ldots, A_n\}\sigma$, *where* σ *is the unique most general unifier of* A_1 *and* $Head$.

A proof *of the goal clause labelling the root of a search space corresponds to a branch in which the leaf node is the empty clause. A* regular proof *is one that only uses regular rules. A* pattern proof *is one that uses at least one pattern rule.*

Example 5 (Search Spaces) *If* $\{connected(engine, wheels)\}$ *is the goal clause labelling a node and the rule clause*

$$connected(X, Z),\ connected(Z, Y) \rightarrow connected(X, Y)$$

labels one of the arcs descending from this node, then the daughter node at the other end of this arc is labelled with $\{connected(engine, Z),\ connected(Z, wheels)\}$.

Definition 8 (Proof Guidance Algorithm) *Given a formal requirement, (ground regular goal clause with one atom), a set of regular rules and a set of pattern rules, the search space is grown depth first in two passes. On the first pass only the regular rules are used. On the second pass, both regular and pattern rules are used. Thus only regular proofs will be found on the first pass and only pattern proofs on the second pass.*

If a pattern proof is found, then the pattern rules used in the proof will have their predicates instantiated to the one in the unifier; these instantiated pattern rules *are presented to the user to ask if they are all true.*

Example 6 (Proof Guidance Algorithm) *If*

$$P(X, Z), P(Z, Y) \rightarrow P(X, Y)$$

is a pattern rule used in the proof and the unifier is $connected$ *then the instantiated pattern rule is*

$$connected(X, Z),\ connected(Z, Y) \rightarrow connected(X, Y).$$

Remark 4 *The restriction of terms to constants or variables in the definition of atoms (Definition 1) means that if the predicate and terms of an atom in a goal clause are instantiated to relations and classes of a finite ontology, there are only finitely many possibilities; so the search space is finite.*

The algorithm does not specify how it is to be applied to ontologies, e.g. how goal atoms are instantiated; this is a deliberate design decision to make it general, so that different search strategies can be applied on top of it; we will see an example in the next section.

[2] OR-trees contrast with AND-trees. In an OR-tree, you can choose one branch and stick with it; in an AND-tree, all branches must be solved. The analogy is with goals of the form $P \vee Q$ versus $P \wedge Q$. Since we will be using depth first search (see Definition 8), our search space is an OR-tree.

2.3 A More Complicated Example

Having presented our algorithm formally, we now apply it to a more complicated version of the previous example of a car ontology; although it is small it is non-trivial.

2.3.1 Pattern Rules for Our Example

As before, we start with no formal requirements and an empty ontology, but this time with a list of generic properties as pattern rules, shown in Figure 1.

This is not meant to be an exhaustive list, but a list of what we regard as among the most commonly occurring properties. Apart from the first, the other three are composed of pattern atoms only and have their familiar meanings. As for the first, we called it *upward inheritance* as it describes the situation where a relation that holds between two classes Z and Y is inherited upwards along hierarchies of *has_part* for one of the classes, Y, so that it also holds between Z and X, the superclass of Y. We singled out inheritance along hierarchies of *has_part*, rather than a generic relation, as we feel that this is a very common situation, making this generic property a mixture of pattern and regular atoms. We further note that the relation that is inherited upwards must not be *has_part*, so that if it is instantiated then it will have two relations, unlike the other three.

These properties are not requirements to be proved and are not part of any ontology, so we still have the (trivial) satisfaction of the requirements.

Type	Generic Property
upward inheritance	$has\_part(X,Y), P(Z,Y) \rightarrow P(Z,X)$
transitivity	$P(X,Z), P(Z,Y) \rightarrow P(X,Y)$
symmetry	$P(Y,X) \rightarrow P(X,Y)$
reflexivity	$P(X,X)$

Figure 1. A list of generic properties.

2.3.2 Starting the Development Process

The user developing the ontology then adds the requirement

The engine is connected to the wheels;

formalised as

connected(engine, wheels)

and some axioms to the ontology; the classes and relations in these are also added to the ontology, which are shown in Figure 2.

We first check that the ontology is consistent. In this case, our atomic axioms are clauses with no negation, so they are consistent. In general, this would involve searching for the empty clause; if we fail then the ontology is consistent. The restriction of terms to constants or variables in the definition of atoms (Definition 1) means the search will terminate, so we will a definite answer.

Having done this, we next check (or the user could initiate the check) if the requirement can be derived from the ontology; we can do this by applying the Proof Guidance Algorithm, since the first pass using only regular rules does exactly this, and has termination. In this case, we have the answer that it cannot be derived. Therefore we use the (second pass of the) Proof Guidance Algorithm to guide the user on the additions to the ontology to make it satisfy the requirement.

Classes	Relations	Atomic Axioms
car	has_part	has_part(car, engine) has_part(car, wheels)
engine axle	connected	connected(engine, axle) connected(axle, leftfrontwheel)
wheels frontwheels leftfrontwheel		has_part(wheels, frontwheels) has_part(frontwheels, leftfrontwheel)

Figure 2. The more complicated car ontology example.

2.3.3 Heuristics for the Algorithm

We have already noted that the algorithm does not specify how it is to be applied to ontologies; to do so we use the following heuristics.

Definition 9 (One Atom Heuristic) *If $\{A_1, A_2, \ldots, A_n\}$ is not the formal requirement (initial goal clause), we require the existence of a unifier σ such that all but at most one of $\{A_1, A_2, \ldots, A_n\}\sigma$ are axioms of the ontology; these are then removed from the goal clause, leaving a goal clause with at most one atom; otherwise the node is a leaf node.*

Example 7 (One Atom Heuristic) *For example, if*

$$\{connected(engine,\ Z),\ connected(Z,\ wheels)\}$$

is the goal clause labelling a node, then Z could be unified with (instantiated to) $axle$ to make the first of the two goals an axiom of the ontology (in Figure 2), which is then removed from the goal clause, leaving

$$\{connected(axle,\ wheels)\};$$

on the other hand if

$$\{connected(leftfrontwheel,\ Z),\ connected(Z,\ wheels)\}$$

is the goal clause labelling a node, then we do not have any unifiers which could make one of the goals in the clause an axiom of the ontology, so the node is a non-empty leaf node (hence does not give a proof).

The idea here is to use the ontology to cut down the search space, hopefully in a way that directs the search towards a proof. If we start with a goal of one atom then when we grow the search space using the Proof Guidance Algorithm then we are unifying the atom with rule clauses to produce goal clauses, usually of more than one atom. Now, if all the atoms in one such goal clause were unifiable with axioms in the ontology, then we would have a proof of the original goal, as in the simple example in Section 2.1.

However, this will rarely be the case; so for our first heuristic, we aim for the next best thing, trying to unify all except at most one of the atoms. This leaves a subgoal of at most one atom to be proved, for which we again attempt to reduce to a subgoal of at most one atom, and so on, until we find a proof or fail.

This heuristic hugely cuts down the search space, though we are aware that it sometimes prevents proofs from being found (see discussion in Section 4).

Definition 10 (Instantiated Pattern Rule Heuristic) *In the second pass of the Proof Guidance Algorithm, instead of using the patterns rules directly, we will have their predicates instantiated to the predicate in the atom of the formal requirement (initial goal clause) and use these* instantiated pattern rules *(the same kind as the ones in Definition 8).*

We also split the second pass into subpasses. On the first subpass only one pattern rule is used; on the next subpass we can use one more pattern rule, and so on, until there are no more pattern rules.

With the Proof Guidance Algorithm, the pattern rules labelling the arcs leading from the formal requirement (initial goal) effectively have their predicates unified with (instantiated to) the one in the requirement, so this is minimal in the number of pattern rules used if an order is imposed on the pattern rules (as is indeed the case in our implementation, see Section 2.5), with the effect on goal clauses lower down the search tree.

By splitting the second pass into subpasses we make our search space small at the start and only grow it if no proof is found.

We can now give a summary of the running of the algorithm with the heuristics. As before, the search space is grown depth first in two passes. On the first pass only the regular rules are used. Each time a new node is grown, its goal clause is reduced using the *One Atom Heuristic* to give either

1. an empty clause, which shows a proof of the original goal has been found;
2. a clause with one atom, which is a leaf node if it has already appeared;
3. a clause with more than one atom, which is a leaf node.

The search space is finite If a proof is not found on the first pass, we go to the second pass, which is split into subpasses. On the first subpass, we obtain an instantiated pattern rule as per the *Instantiated Pattern Rule Heuristic*, which together with the regular rules is used to grow the search space, and then proceed as on the first pass.

If a proof is not found, we go to the next subpass, where we obtain one more instantiated pattern rule, and carry on as before. This goes on until either we find a proof or run out of pattern rules to instantiate.

2.3.4 The Running of the Algorithm

Given these two heuristic we can now give an overview of the running of our algorithm on the car ontology in Figure 2.

We are trying to guide the user on the additions to the ontology to make it satisfy the requirement

connected(engine, wheels),

which is our initial goal.

The first pass using only regular rules is just the check on the derivation of the requirement from the ontology from earlier, which fails, so we need the second pass.

On the second pass, without loss of generality the first subpass uses the first pattern rule from Figure 1,

$$has\_part(X, Y), P(Z, Y) \rightarrow P(Z, X),$$

the upward inheritance property, which we instantiate to

$$has\_part(X, Y), connected(Z, Y) \rightarrow connected(Z, X)$$

using the Instantiated Pattern Rule Heuristic, giving us the upward inheritance rule for *connected*, then unify with the requirement to give

$$has\_part(wheels,\ Y), connected(engine,\ Y)$$

$$\rightarrow connected(engine,\ wheels). \quad (4)$$

The two terms on the right hand side are the subgoals of the initial goal, proof of which will give us the proof of the requirement. We now apply the One Atom Heuristic to the subgoals and note that we can instantiate Y to $frontwheels$ to make the first of the two goals an axiom of the ontology, leaving us to prove

connected(engine, frontwheels).

One further application of the above steps reduces us to proving *connected(engine, leftfrontwheel).*

But now it is easy to see that with only this rule we will run out of possibilities quickly; further unifications with the upward inheritance rule will bring goals that could not be unified with axioms in the ontology.

So we go to the second subpass, and get another instantiated pattern rule

$$connected(X, Z), connected(Z, Y) \rightarrow connected(X, Y), \quad (5)$$

the transitivity rule for *connected*. Applying this to *connected(engine, leftfrontwheel)* gives us

$$connected(engine, Z), connected(Z, leftfrontwheel), \quad (6)$$

and instantiating Z to $axle$ makes both axioms of the ontology, thus completes the proof.

We then present the two instantiated pattern rules used in the proof to the user for approval.

2.4 Discussion

In this paper, to prove our formal requirements we needed to come up with rules of implication which are also a properties of relations. We believe it is advantageous for an algorithm running on a computer to do this job, for the following reasons:

1. arguably, it is easier for a human to understand such rules than to design them, particularly as they have formal logical formulations such as (1);
2. a relation could have many properties, designing them is a diversion from the modelling of the domain, so getting a program to suggest them is a good division of labour;
3. many relations could have similar properties to be added, so together with the above two points this makes the task tedious and repetitive for a human, but computers are well suited to such tasks and can help to reduce the workload.

However, it is outside the scope of this paper to investigate these claims, empirically or otherwise.

2.5 Implementation

The Proof Guidance Algorithm, the two heuristics and example ontology have been implemented in a Prolog [7] program. One of the advantages of using Prolog is that it allows easy meta-programming; and indeed our programs are written as meta-interpreters. Additionally, Prolog's declarative style and logic-based semantics allow it to be used as a ontology language.

Prolog imposes orders on goals and rules, so the sets of atoms in goal clauses become lists (see Definition 4), as do the sets of regular and pattern rules (see Definition 8).

Implementation Note: So that pattern atoms can be unified with other atoms using only first-order unification, an atom $P(T_1, \ldots, T_n)$ is represented internally as $atom(P, T_1, \ldots, T_n)$.

This means

connected(engine, axle)

becomes

atom(connected, engine, axle).

The command

suggest(atom(connected, engine, wheels), Proof, GoalList)

is used to try to prove

connected(engine, wheels)

(here *Proof* is the variable holding the final proof obtained, and *GoalList* holds the current list of goals to catch looping). The command succeeds, with the proof printed out at the end.

Proof = atom(connected, engine, wheels) :-
(atom(has_part, wheels, frontwheels) :- True), (atom(connected, engine, frontwheels) :-
(atom(has_part, frontwheels, leftfrontwheel) :- True), (atom(connected, engine, leftfrontwheel) :-
(atom(connected, engine, axle) :- True), (atom(connected, axle, leftfrontwheel) :- True)))

2.6 Evaluation

Our hypothesis for this paper is as follows:

> *Proof-Guided Ontology Development using pattern rules can be used to provide guidance in the development of real ontologies.*

The evaluation of our hypothesis is on-going, as we are looking for example ontologies where our method is applicable.

One interesting example we found is the OWL-Time Ontology [5]. This ontology has a relation *before*, which is defined to be anti-reflexive, anti-symmetric, and transitive on intervals. For example, the anti-reflexive axiom is given as

$$before(T_1, T_2) \rightarrow T_1 \neq T_2.$$

Given the axioms in OWL-Time, the following is a theorem of the ontology:

$$begins(t_1, T) \;\&\; ends(t_2, T) \;\&\; before(t_1, t_2) \rightarrow ProperInterval(T).$$

This says that if an interval T begins at instant t_1 and ends at instant t_2 and such that t_1 is before t_2, then it is a proper interval, which is itself defined as

$$(\forall T)(ProperInterval(T) \leftrightarrow interval(T)$$

$$\&\; (\forall t_1, t_2)(begins(t_1, T) \;\&\; ends(t_2, T) \rightarrow t_1 \neq t_2)).$$

This theorem is mentioned in the predecessor ontology of the OWL-Time Ontology, the DAML-Time Ontology [4], which has the same basic axioms and same axioms on the relation *before* as OWL-Time.

To prove this theorem, it seems to us that the anti-reflexive axiom on intervals for the relation *before* is needed, though the proof we could see is somewhat complicated.

OWL-Time does not have an anti-reflexive axiom on *instants* for the relation *before*,

$$before(t_1, t_2) \rightarrow t_1 \neq t_2;$$

if it did, the proof of the theorem would be much easier.

We would like to use our method of using a proof attempt of the theorem to suggest the anti-reflexive axiom on instants for *before*; but as it is, we cannot as we are not yet able to deal with quantification and negation. Assuming that we can, and that we have a pattern rule for anti-reflexivity, then it should be quite straightforward to make the suggestion.

We are currently working to extend our method to deal with quantification and negation.

3 RELATED WORK

In terms of overall approach to ontology development, TOVE (TOronto Virtual Enterprise) [3] is the most similar to us, in fact it is TOVE who first introduced the idea of formally proving satisfaction of requirements by ontologies.

In TOVE, requirements are called *competency questions*, a term and notion taken up by several subsequent methodologies.

Competency questions arise and are used in the following way. First, usage scenarios for the proposed ontology are identified; then from these a set of natural language questions, called competency questions, are derived. These are questions that the ontology should be able to answer given the usage scenarios.

These questions and their answers are then used to extract the main concepts, their properties, relations and axioms of the ontology, which are then used in the development of the latter.

Usually, the finished ontology is evaluated to check that it is able to answer the competency questions. In TOVE, this is done formally, with formalised competency questions (which are in the form of logical statements) proved to follow from the (formal) ontology; if the ontology is seen as a set of axioms, the formalised competency questions conjectures on the ontology to be deduced.

The difference between TOVE and our approach is that even though TOVE also intends that the competency questions are used to develop the ontology (axiomatize the ontology) in an iterative way, it is vague about how this is actually to be done. In fact, it [3] states that:

> *There may be many different ways to axiomatize an ontology, but the formal competency questions are not generating these axioms.*

In contrast, we use (the proof attempt) of formal requirements to (guide the) generation of the axiomatization the ontology, and gave an example of a specific way of doing this.

In terms of our example, of the specific way of (guiding the) generating the axiomatization the ontology, [6] is the most similar to us. The authors also propose a way of adding properties such as transitivity or symmetry to relations in ontologies, including automatically generated ontologies such as DBPedpia where this information tends to be lacking, though their method is rather different.

In [6], parts of ontologies of interest are matched with Ontology Design Patterns which already contains the properties, using an algorithm that takes into account both structural and lexical information, and which is configurable by the user. These Ontology Design Patterns are analogous to our Pattern Rules, in that they are also generic versions of commonly occurring constructions in ontologies, where these constructions are collections of related classes, relations and axioms. The algorithm first finds parts of ontologies that structurally match Ontology Design Patterns, e.g. the hierarchies match, then use lexical information such as synonyms to confirm the match. When a match is found, the properties in the Ontology Design Patterns are added to the matched items in the ontology.

In [6] the authors presented a feasibility study though there is no evaluation.

4 CURRENT AND FUTURE WORK

Work is on-going on the usage of pattern rules to provide guidance for ontology development, and we have further work planned.

4.1 Providing Guidance

On the basis of existing definitions for formal requirements, atomic axioms and pattern rules, we are developing more examples of requirements, axioms and more pattern rules, and the associated heuristics needed to deal with these.

We already have an example ontology, which is a more complicated version of the car ontology presented in this paper, for which the current heuristics are insufficient, in that they prevent proofs from being found. For this we have developed and implemented new heuristics.

We are also working on and planning for more general definitions of formal requirements and atomic axioms, for example variables in formal requirements, and the ability to deal with quantification and negation, which will allow us to deal with the OWL-Time Ontology example in full. For these more complicated ontologies heuristics may be insufficient and we will consider other measures.

In all cases, we will be testing our method on 3rd party ontologies.

4.2 Other Aspects of Ontology Development

In this paper we focused on providing guidance for ontology development, and did not consider or simplified other aspects of ontology development. We will consider these aspects in future, including:

- reuse of existing ontologies to provide requirements and axioms, and the issues this brings, such as dealing with equivalence of requirements and axioms;
- the ontology language used and how existing languages like OWL and CL could be used with our approach;
- use of existing tools for e.g. checking consistency.

We have discussed our proposal with a few ontology designers, but we are not currently working with any; this is something we will consider in future.

5 CONCLUSION

In this paper, we introduced the notion of *Proof-Guided Ontology Development*, and the use of pattern rules to guide the development of ontologies. We gave formal definitions of the notions involved, including formal requirements, atomic axioms and pattern rules, and gave a formal description of our algorithm. Our procedure is illustrated with an example, which has been implemented in Prolog programs.

We presented a hypothesis for this paper, the evaluation of which is on-going. We gave the example of the OWL-Time Ontology, to which our method does not yet apply, but noted the problems to be dealt with. These problems are to do with aspects of First Order Logic; though they would bring difficulties it seems reasonable that they could be dealt with. Assuming that they are, we noted that our method could potentially be applied.

Overall, we expect that our method is definitely useful for ontology development, and we are working to extend and evaluate it.

ACKNOWLEDGEMENTS

We would like to thank the referees for their comments which helped improve this paper.

REFERENCES

[1] M. Bergman. A brief survey of ontology development methodologies. online, August 2010. url: http://www.mkbergman.com/906/a-brief-survey-of-ontology-development-methodologies/, last retrived 28 Feb 2013.

[2] N. Dahlem, J. Guo, A. Hahn, and M. Reinelt, 'Towards an user-friendly ontology design methodology.', in *Interoperability for Enterprise Software and Applications*, pp. 180–186. IEEE Computer Society, (2009).

[3] M. Grüninger and M. Fox, 'Methodology for the design and evaluation of ontologies', in *IJCAI'95, Workshop on Basic Ontological Issues in Knowledge Sharing*, (1995).

[4] Jerry R. Hobbs. A DAML ontology of time, 2002.

[5] Jerry R. Hobbs and Feng Pan, 'An ontology of time for the semantic web', *ACM Transactions on Asian Language Processing*, **3**(1), 66–85, (2004).

[6] Nadejda Nikitina, Sebastian Rudolph, and Sebastian Blohm, 'Refining ontologies by pattern-based completion', in *Proceedings of the Workshop on Ontology Patterns (WOP 2009), collocated with the 8th International Semantic Web Conference (ISWC-2009), Washington D.C., USA, 25 October, 2009*, volume 516, (2009).

[7] L. Sterling and E. Shapiro, *The Art of Prolog*, MIT Press, Cambridge, MA, second edn., 1994.