SAsSy - Making Decisions Transparent with Argumentation and Natural Language Generation

Nava Tintarev
University of Aberdeen
Aberdeen
n.tintarev@abdn.ac.uk

Roman Kutlak
University of Aberdeen
Aberdeen
r.kutlak@abdn.ac.uk

ABSTRACT
An autonomous system consists of one or more physical or virtual agents that can perform tasks without continuous human guidance. In order to realise their promise, techniques for making such autonomous systems scrutable and transparent are therefore required. To address this issue the Scrutable Autonomous Systems (SAsSy) demo shows how argumentation and natural language can be combined to generate a human understandable dialog explaining the operation of an autonomous system. On the one hand argumentation theory is used to simulate human understandable reasoning mechanisms. On the other, natural language generation tools are used to translate logical statements into simple plain English. The idea is to generate a dialog that enables the user to understand and question the reasoning present in autonomous systems.

Author Keywords
Explanations, Argumentation, Natural Language, Agents

ACM Classification Keywords
H.5.2 User Interfaces: Natural Language, Interaction styles, Graphical user interfaces (GUI)

BACKGROUND AND APPLICATION CONTEXT
An *autonomous system* consists of physical or virtual entities, or *agents*, that can perform tasks without continuous human guidance. Autonomous systems are becoming increasingly ubiquitous, ranging from unmanned vehicles, to robotic surgery devices. Such systems can potentially replace humans in tasks which can be dangerous (such as refuelling a nuclear reactor), mundane (such as crop picking), or require superhuman precision (as in robotic surgery). While increasing reasoning capacity can enable an autonomous system to handle a wider range of situations, modelling and verifying the operation of such systems becomes increasingly difficult.

The increasing amount of independent reasoning that takes place within an autonomous systems 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 understand what is going on in the 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. When things go smoothly transparency may not be that important. However, it is equally vital that a user can identify undesired actions before they are carried out, and interfere appropriately if need be: even if the user understands the system they need to be able to cancel actions or suggest alternatives with relative ease in a timely manner.

The SAsSy project[1] has for the last year been investigating computational mechanisms for providing transparency to humans regarding the internal workings of an autonomous system. We use formal argumentation in combination with natural language to offer explanations to a human. The system explains which sequence or actions, or what plan, have been chosen for execution by the system. More specifically it explains *why* a certain plan has been selected. That is, the user should be able to follow a chain of reasoning with arguments and counter arguments. The system also allows users to provide additional information which can be used to modify the arguments, and subsequently, the plan.

The SAsSy team is working with industrial partners in the hydrocarbon exploration and unmanned vehicle domains to identify users' explanatory needs in these domains. However, the system architecture is flexible enough to model new domains and in this demo we present a simplified scenario built around delivery logistics. The components that are domain specific are: a workflow, domain rules, and a lexicon.

The scenario is based on a delivery driver in Scotland who is delivering a package from Edinburgh to Inverness and driving back. In this case, the plan is a choice of route with a sequence of driving actions to a number of intermediate locations. Each location serves as a potential choice point from which several other locations may be possible. Some routes yield shorter distances between points, but given other factors such as traffic and road conditions the shortest route is not always the best option. The driver executes the plan step by

[1] `http://scrutable-systems.org`

step. At each step they can question the system while executing the actions in the plan.

SYSTEM DESCRIPTION

Explanations have frequently been a component of intelligent systems (IS) such as expert systems [1, 4, 9] and recommender systems [6, 8]. The explanation capabilities in expert systems have often been evaluated with users in terms of whether they increase acceptance of an intelligent system or acceptance of decisions. However, there are other reasons why explanations may be introduced to an IS including transparency and scrutability [8] – helping users understand how decisions were made (transparency), as well as allowing users to tell the system that it is wrong (scrutability).

The demo shows the interaction between two core technologies: human understandable reasoning mechanism (represented through argumentation theory) and natural language generation tools to translate logical statements into plain (natural) English. Our system is developed in Python and is available under the BSD licence[2].

According to the recent classification of explanations in IS by [4], the explanations in our system can be classified as justification, and to certain extent, trace-based explanations. Justification type explanations describe 'why' a decision was taken and supply the descriptive knowledge used to reach that decision. Trace explanations supply information about 'how' the decisions are made, e.g. which rules were applied. In our system they are exposed to the user thanks to argumentation [5].

Our system does not present the user with the full reasoning trace, but rather allows the user to discuss or argue with our system about each subsequent fact that is discussed. The user decides how much information they need, which means the dialog can be limited in size even for very large knowledge bases. Our system also allows alteration to the rule base. Since all knowledge in our system is captured through rules, the information can be updated by the user through adding rules, removing them, or adding exceptions to existing rules. This in turn affects subsequent decisions, since the rules in turn affect the arguments and their outcomes.

Reasoning mechanism. Our system uses two kinds of rules: *defeasible* and *strict*. *Strict rules* capture traditional implication – whenever the literal or literals on the left-hand side are true, so is the literal on the right-hand side. An excerpt of the knowledge base can be seen in Figure 1. The main difference between these two kinds of rules are that defeasible rules can be affected by exceptions, while strict rules cannot.

An example of a strict rule might be: `flood_road −− > closed_road`, which might be read as "*if a road is flooded, it is closed.*"

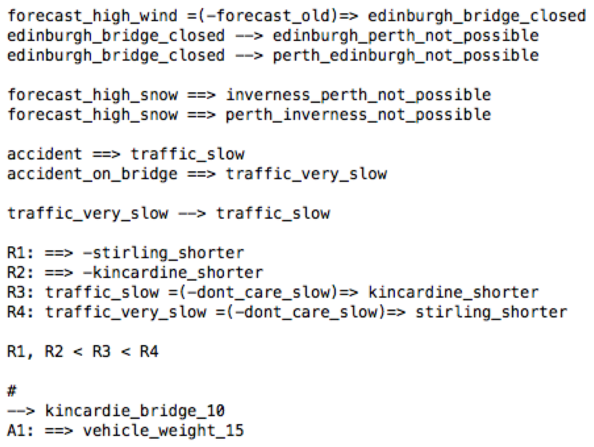
Defeasible rules capture defeasible implication – whenever the literal(s) on the left-hand side are true the right-hand side is **usually** also true. An example of such rule might be: `snow_road ==> closed_road`, which might be read as "*if a road is covered by snow, it is usually closed.*"

[2]**https://bitbucket.org/rkutlak/sassy**

The previous defeasible rule can be changed to the following rule: `snow_road = (-plough_road) => closed_road`, which might be read as "*if a road is covered by snow, it is usually closed unless it has been ploughed.*"

Rules without pre-conditions on the left-hand side are used to represent asserted or assumed information. For example, `==> accident_on_bridge` means that "*there was an accident on the bridge.*" We use the terms *rule* and *information* interchangeably.

Part of the knowledge base represents legal actions in terms of locations that can be reached from each point – we illustrate this in Figure 2. The reasoning mechanism in SAsSy is based on argumentation [5], where the arguments are derived from rules in the knowledge base. Arguments are stored as a directed graph, in which the nodes constitute arguments and arcs between nodes symbolise attacks between arguments. The advantage of using argumentation as opposed to traditional rule-based reasoning, is that it can reason even in the



```
forecast_high_wind =(-forecast_old)=> edinburgh_bridge_closed
edinburgh_bridge_closed --> edinburgh_perth_not_possible
edinburgh_bridge_closed --> perth_edinburgh_not_possible

forecast_high_snow ==> inverness_perth_not_possible
forecast_high_snow ==> perth_inverness_not_possible

accident ==> traffic_slow
accident_on_bridge ==> traffic_very_slow

traffic_very_slow --> traffic_slow

R1: ==> -stirling_shorter
R2: ==> -kincardine_shorter
R3: traffic_slow =(-dont_care_slow)=> kincardine_shorter
R4: traffic_very_slow =(-dont_care_slow)=> stirling_shorter

R1, R2 < R3 < R4

#
--> kincardie_bridge_10
A1: ==> vehicle_weight_15
```

Figure 1. Excerpt of the knowledge base used in our system.

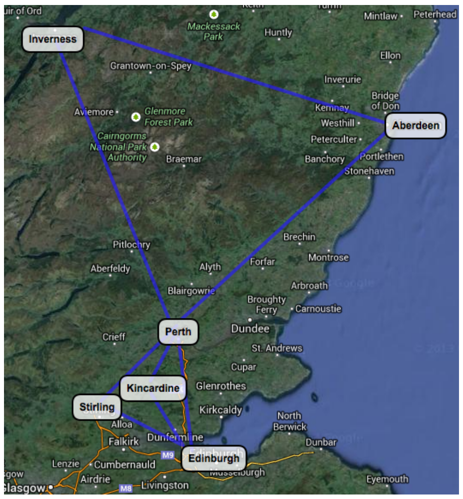


Figure 2. This map is included as an illustration of the different geographic locations used in the example. In our system, routes are represented in the knowledge base as rules of what kinds of actions are allowed. Locations on the map are slightly shifted to increase legibility.

face of contradicting statements. The use of argumentation also enables two important features: non-monotonicity and alternative options. Non-monotonicity means that (when the user) adds new information, it may invalidate past conclusions and justify new ones. Alternative conclusions are represented in the argumentation graph which makes them available for us to both explain what alternatives were considered by our system as well as justify why they were not chosen.

The explanations build on the instantiated grounded game presented in [3]. Our system allows users to step through the argument graph in a sequences of discussion moves presenting arguments and counter arguments. Intuitively, such discussion 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 [7], but a recent strand of work takes its inspiration from human dialogue to find intelligible explanations of an argument's status (i.e., whether to accept or reject the conclusion of an argument) [2].

The Natural Language Generation. *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. In our case, the non-linguistic information are the rules capturing the "knowledge" in our system.

As we have already seen, the rules are formed from literals (e.g., `snow_road`), which can be ambigious or difficult to understand. We use NLG techniques to convert the literals to more natural text as well as to improve the presentation by removing unnecessary information. For example, since defeasible rules capture implications that are usually true, we do not present the exceptions to such rules to the user.

Presenting the available information in concise and unambiguous language could be crucial in e.g., the unmanned vehicle domain, where an operator has limited time to comprehend our system's reasoning and potentially change the course of action. We plan to include a summary of the presented plan as well as to use other NLG techniques such as aggregation (combining simple sentences together for better presentation) and referring expression generation (e.g., using pronouns when referring to past entities) to improve the presentation of the information. Indeed, in order to effectively be able to communicate 'why' certain decisions are preferable, the user needs to be able to understand *what* the recommended plan is first.

DEMO

Our demonstrator shows how argumentation can be used to emulated dialog, and how natural language can be used to present reasoning rules in a way that is familiar to users. The possible actions are represented as a workflow which is visualized as a graph, which is accompanied by natural language descriptions. In addition, the user can contribute to the dialog by asking questions. Figure 3 demonstrates a screenshot of the SAsSy demo.

The system allows the user to ask three different types of questions:

1. *"What can I do next?"*: The left hand pane suggests a next possible location to drive to, e.g. 'Next task: Go to Inverness'. The workflow at the top shows the alternative options, e.g. Stirling1, Kincardine1 and Perth1.

2. *"Why does the system NOT say that I should do Y?"*: likewise, the user can ask why a certain options is rejected: "why out Perth1?". To form a reply the system derives the relevant rules and translates them into natural language in a form such as "Going through Stirling is faster because the traffic is very slow.".

3. *"Why does the system say that a certain thing is true?"*: The user can type a question such as "why traffic_very_slow" in the dialog field and receive an answer such as "The traffic is very slow because of an accident on the bridge.".

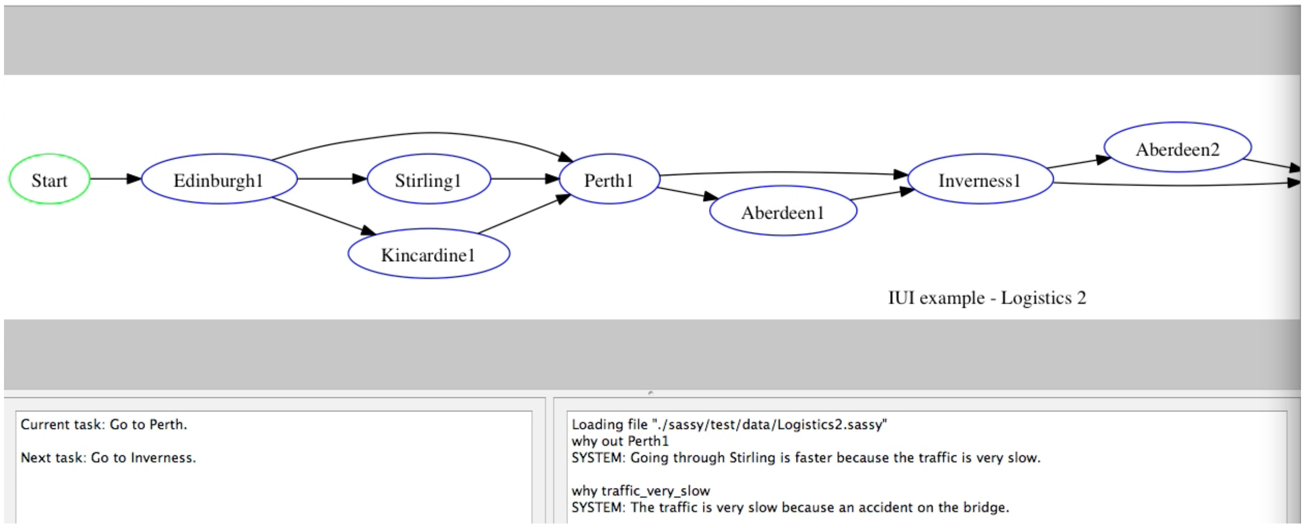
For the time being the user input is limited to controlled symbolic constants such as 'Perth1' or 'traffic_very_slow'. Later versions of the system will make explicit the vocabulary available to a user regardless of whether it is controlled or processed from natural language.

The user may choose to question the system several times. For example, when the system tells the user that they cannot take a certain sub-route the user can query what knowledge was used to derive this argument, such as the case with traffic_very_slow in the screenshot and example above.

A user can choose to manually override or alter the knowledge base. A manual override means they simply proceed as they wish, for example by typing "next Perth1". Conceptually, altering the knowledge base is important if a similar process is likely to be repeated – so that users do not have to manually override repeatedly. For example, the situation may have changed and the user may want to tell the system that the accident is no longer causing issues:
Driver> why traffic_very_slow
SYSTEM> The traffic is very slow because an accident on the bridge.
Driver> retract accident_on_bridge

The system deletes the information of an accident on a bridge, which affects the argument against using the bridge. The system now no longer has a reason to recommend the route though Stirling, and returns to the default of recommending going straight to Perth.The system also represents a preference ordering among rules in case of conflicts to mitigate deadlocks, or situations where the system is undecided. So, if two arguments attack each other, only the attack of the argument with the more preferred rule holds true.

Apart from out of date information, the user and the system may have different information. For example, the system may be telling the user that they cannot go through Kincardine because of the weight of their van, but has gotten the weight wrong:
Driver> why out Kincardine1



```
Current task: Go to Perth.

Next task: Go to Inverness.
```

```
Loading file "./sassy/test/data/Logistics2.sassy"
why out Perth1
SYSTEM: Going through Stirling is faster because the traffic is very slow.

why traffic_very_slow
SYSTEM: The traffic is very slow because an accident on the bridge.
```

Figure 3. Screenshot of the SAsSy system. *Top:* Workflow with all the possible options, represented as a graph. The edges are actions such as 'drive from a to b', and the edges represent states equivalent to reaching a certain location. Perth1 is the state reached in the outbound journey leaving from Aberdeen, whereas Perth2 is that state of the return journey from Inverness. *Bottom left:* Status of system with current and next steps. Note that some steps could be triggered by the system, in which case the change of status and action taken are also represented here. *Bottom right:* Dialog between user and system. This is where the user asks questions and the system responds. The user asks why they cannot go through Perth, and the user supplies a justification (traffic is slow). However, the user may not be willing to accept this justification. They query why traffic is slow and receive an answer for this as well (there was an accident on the bridge).

SYSTEM> you cannot go from Edinburgh to Kincardine because the maximum allowed weight on the bridge is 10 tons and the weight of the vehicle is 15 tons.
Driver> retract vehicle_weight_15
SYSTEM> deleted

In addition, users can introduce new information to the system such as a new weight for the vehicle:
Driver>assert vehicle_weight_5

The explanation capability in SAsSy is undergoing continued development, with another 2 years remaining on the project. Development is guided by ongoing user-centered experimentation. We are currently planning experiments on presenting information differently depending on the areas of responsibility of a user. This would allow the system to tailor the information presentation to, for example, someone who is responsible for deliveries at an airport versus someone who in ensuring the delivery of a specific package.

ACKNOWLEDGMENTS

Many thanks to our esteemed colleagues for their valuable comments. 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. Buchanan, B. G., and Shortliffe, E. H., Eds. *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*. Addison Wesley, 1984.
2. Caminada, M., and Podlaszewski, M. Grounded semantics as persuasion dialogue. In *COMMA*, B. Verheij, S. Szeider, and S. Woltran, Eds., vol. 245 of *Frontiers in Artificial Intelligence and Applications*, IOS Press (2012), 478–485.
3. Caminada, M., Podlaszewski, M., and Green, M. Explaining the outcome of knowledge-based systems; a discussion-based approach. In *AISB* (2013).
4. Darlington, K. Aspects of intelligent systems explanation. *Universal Journal of Control and Automation 1* (2013), 40–51.
5. Dung, P. M. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence 77* (1995), 321–257.
6. McSherry, D. Explanation in recommender systems. *Artificial Intelligence Review 24(2)* (2005), 179 – 197.
7. Rahwan, I., Madakkatel, M. I., Bonnefon, J.-F., Awan, R. N., and Abdallah, S. Behavioural experiments for assessing the abstract argumentation semantics for reinstatement. In *Cognitive Science* (2010).
8. Tintarev, N., and Masthoff, J. Evaluating the effectiveness of explanations for recommender systems: Methodological issues and empirical studies on the impact of personalization. *User Modeling and User-Adapted Interaction 22* (2012), 399–439.
9. Ye, L., Johnson, P., Ye, L. R., and Johnson, P. E. The impact of explanation facilities on user acceptance of expert systems advice. *MIS Quarterly 19*, 2 (1995), 157–172.